VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ALGORITMY PRO ŘÍZENÍ ELEKTRICKÝCH MOTORŮ
ALGORITHMS FOR ELECTRICAL MOTOR CONTROL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Antonín Lyko**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Blaha, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Antonín Lyko ***ID:*** 134546
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Algoritmy pro řízení elektrických motorů

POKYNY PRO VYPRACOVÁNÍ:

1.Seznamte se s problematikou řízení elektrických motorů z hlediska požadovaných vstupů a výstupů.
2. Seznamte se s procesory AURIX TC27x od firmy Infineon a jejich periferiemi.
3. Nastudujte softwarovou architekturu AUTOSAR a filosofii využití ovladačů MC-ISAR.
4. Naprogramujte obsluhu číslicových vstupů a výstupů s využitím modulů PORT a DIO. Pomocí modulu GTM realizujte softwarový modul pro generování signálů pulzně šířkové modulace. Pro nastavení periferií použijte aplikaci EB Tresos Studio.
5. Software otestujte na vývojové desce AURIX Application Kit TC277.

DOPORUČENÁ LITERATURA:

[1] Sul, S.K.: Control of Electric Machine Drive Systems. February 2011, Wiley-IEEE Press. ISBN: 978-0-4-0-59079-9.

Firemní literatura firmy Infineon a Elektrobit, další dle doporučení vedoucího.

Termín zadání: 6.2.2017 ***Termín odevzdání:*** 15.5.2017

Vedoucí práce: doc. Ing. Petr Blaha, Ph.D.
Konzultant:

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

ABSTRAKT

Tato práce popisuje strukturu a základní prvky softwarové architektury Autosar. Dále jsou představeny možnosti generování konfiguračních kódu jak pro ovladače MC-ISAR tak pro hardwarové moduly procesoru AURIX TriCore TC277 pomocí nástroje EB tresos Studio. Pro účely možné integrace řídících algoritmů elektrických motorů byly vytvořeny a popsány konfigurace hardwarových modulů GTM a VADC umožňující generaci PWM signálů spolu se synchronně spouštěnými paralelními analogově digitálními převody. K tomuto účelu bylo také vytvořeno a popsáno aplikační rozhraní včetně PWM ovladače.

KLÍČOVÁ SLOVA

Autosar, EB tresos Studio, MC-ISAR, Aurix TC275, ADC, PWM

ABSTRACT

This paper presents the structure and basic elements of the Autosar software architecture. In addition, the configuration code generation options are presented for both MC-ISAR drivers and AURIX TriCore TC277 hardware modules using EB tresos Studio. For the purpose of possible integration of the electric motor control algorithms, configurations of the GTM and VADC hardware modules have been created and described to enable the generation of PWM signals along with synchronously triggered parallel analogue-to-digital conversions. For this purpose, an application interface including the PWM driver was also developed and described.

KEYWORDS

Autosar, EB tresos Studio, MC-ISAR, Aurix TC275, ADC, PWM

LYKO, Antonín. *Algoritmy pro řízení elektrických motorů*. Brno, 2017, 82 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřící techniky. Vedoucí práce: doc. Ing. Petr Blaha, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Algoritmy pro řízení elektrických motorů“ jsem vypracoval(a) 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(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) 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(-ky)

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(-ky)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ

1 ÚVOD

Hned v úvodu je mou autorskou povinností upozornit čtenáře na fakt, že ačkoli je práce nazvaná 'Algoritmy pro řízení elektrických motorů', její obsah není zaměřen na konkrétní algoritmy, nýbrž na popis a použití vhodné softwarové infrastruktury - ovladačů a pracovního rámce, umožňující následnou integraci algoritmu na zvolený Hardware.

V době, kdy cílové aplikace vestavěných systémů musí plnit množství dílčích úloh, mezi které neodmyslitelně patří například zajištění bezpečnosti a spolehlivosti, nároky na objem vložené práce napříč různými disciplínami značně stoupají.

Jako prvním odvětvím, v řetězci vývoje, testování, výroby a následného uplatnění výrobku na trhu, kde je možné projevy těchto tendencí pozorovat je u výrobců polovodičových součástek a digitální řídící techniky. Složitost komponent jako např. mikrokontrolérů roste. Standardem se stávají 32-bité architektury. Počet jader větší než jedna není raritou. Rozměr dokumentace je v řádech jednotek tisíců stran.

Dalším článkem řetězce, na který se tím proto přesouvají vzrůstající nároky je vývoj ovladačů. Zde existuje prakticky pouze jediná možnost jak tento proces urychlit a to je použití již vytvořených ovladačů z externího zdroje, ať už volně dostupných nebo placených. Tyto ovladače však často pokrývají z veškeré funkcionality a variability hardwaru pouze její omezenou podmnožinu. Tím se nachází řešení tohoto problému opět na svém počátku a to je vytvoření nových, či minimálně úprava dostupných externích ovladačů.

Následujícím krokem vývoje celkové aplikace a jejím možným urychlením je úvaha, do jakého prostředí aplikaci či algoritmus zasadit. Stačí pouze toto 'jednoúlohové' prostředí disponující pouze ovladači pracující s konkrétními hardwarovými moduly? Vzhledem k výše popsanému líčení dnešní situace je zřejmé, že nestačí. Aplikace musí plnit několik dílčích úloh, jako je např. výše zmíněná bezpečnost či v případě této práce řízení motoru. Je proto potřeba převést plnění těchto dílčích úloh na entity, které budou podle priorit distribuovány a přerozdělovány v rovině procesorového času. Ze systémového pohledu, kde mají být jednotlivé logické úlohy realizovány samostatnými softwarovými komponenty, tato část spadá pod operační systém (OS).

Pro vývojáře, či pracovníka jež se snaží verifikovat a otestovat svůj algoritmus to proto znamená nutnost nastudování všech závislostí a omezení použitého hardwaru, vytvoření či modifikaci ovladačů, migraci a integraci zvoleného operačního systému a nakonec i implementaci samotného algoritmu.

Čas vložený do režií, netýkajících se přímo implementace algoritmu je proto značný. Vznikají tak tendence a snahy jak tento proces urychlit a z automatizovat.

Nástroje umožňující takto činit pracují jako generátory zdrojového kódu. Je

možné generovat kód pro konfiguraci hardwarových modulů mikrokontrolérů, je možné zvolit použití operačního systému, tím je zajištěna integrační a část OS na specifický HW. Je také možné generovat kód samotného algoritmu podle simulačního schématu. To umožňuje např. DSP System Toolbox využívající MATLAB a Simulink [1].

Tyto dva výše zmíněné nástroje, tedy generátor hardwarové konfigurace i s operačním systémem a generátor kódu algoritmu jsou však stále značně vzdáleny od realizace konkrétní aplikace. Úloha stále obsahuje nutnost nastudování specifických vlastnosti použitého hardwaru.

Tuto problematiku se snaží řešit již rozvinutější a komplexnější softwarové architektury. Za cíl si kladou vytvoření uniformního a standardizovaného softwarového prostředí pro implementaci cílové aplikace či aplikací, s odstíněním hardwarových závislostí a diferencí. Pro programátora cílové aplikace se poté jeví výsledné prostředí vždy stejně bez ohledu na konkrétní hardware, na kterém je aplikace spuštěna.

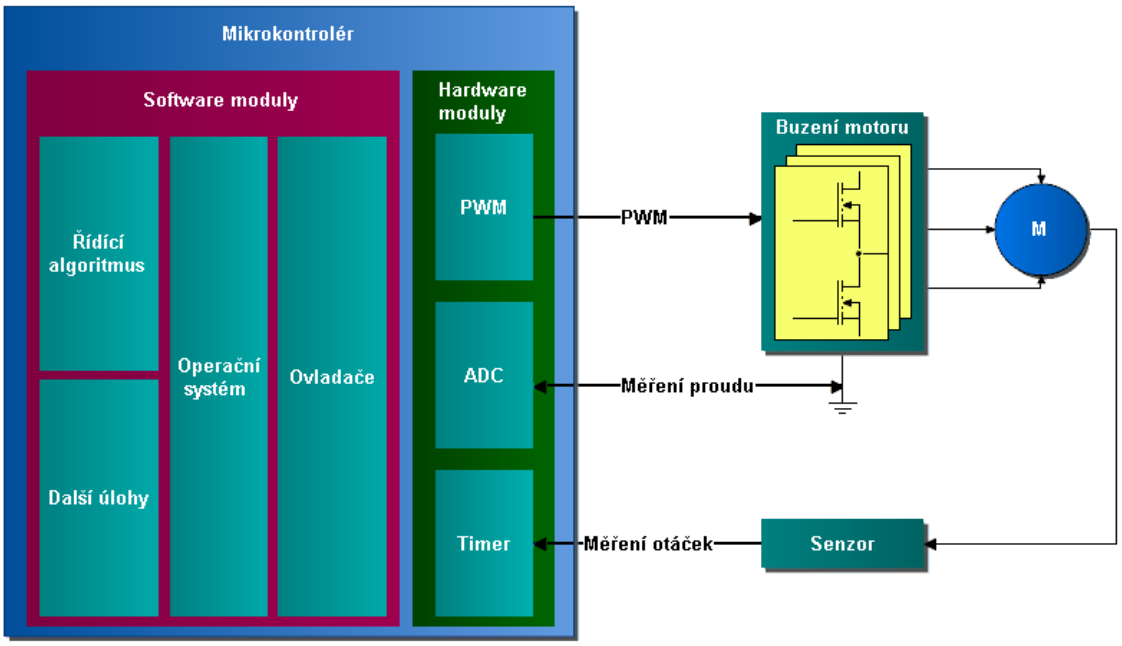
Takovouto softwarovou architekturu můžeme nalézt např. v automobilovém průmyslu. Zkráceně nese jméno AUTOSAR - AUTomotive Open System ARchitecture [2].

Nyní již byly představeny všechny nezbytné bloky a komponenty, které mají za cíl u velkých projektů značně usnadňovat a hlavně urychlovat celkový proces vývoje.

Obsahem a cílem této práce je následné přiblížení a popsání možného způsobu zacházení a manipulace s některými z výše zmíněných nástrojů.

2 OBECNÝ ŘÍDÍCÍ SYSTÉM

V obecném pohledu se řídící systém elektrických motorů skládá ze dvou základních funkčně-logických celků. Jsou jimi silová část, zajišťující odpovídající buzení daného motoru, a řídící část, zpracovávající informace o stavu motoru v daném okamžiku (jakými mohou být například poloha, rychlost apod.) a zpětně generující patřičné řídící signály (ovlivňující následný průběh stavových veličin). Takovýto systém, kde řídící úlohu plní mikrokontrolér ukazuje obrázek 2.1.

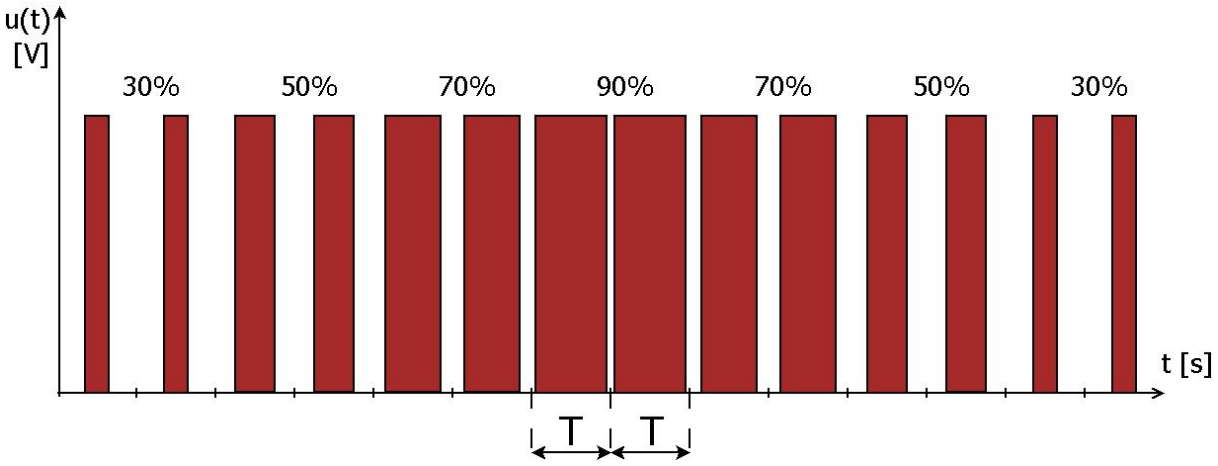


Obr. 2.1: Obecný řídící systém

V dnešní praxi jsou pro účely řídících signálů běžně užívány signály s modulovanou šířkou pulsů PWM. S různými variantami vzorů PWM signálů se lze setkat u řízení BLDC, PMS, DC a dalších motorů. Konkrétní podoba vždy závisí na typu motoru a jemu zvoleném způsobu (algoritmu) řízení. Možnou podobu provedení centrovaného PWM signálu ukazuje obrázek 2.2, kde jednotlivé pulsy jsou umístěny vždy ve středu periody T.

Dalšími podstatnými informacemi potřebných pro algoritmy řízení elektrických motorů jsou např. informace o proudu jednotlivými fázemi motoru či informace o nad-proudu. Všechny tyto informace vstupují do řídícího systému v podobě analogového napětí. Pro jejich zpracování je tedy využito analogově-digitálního převodníku ADC, jenž může být součástí samotného mikrokontroléru či může být využit externí.

Zpracování signálů o poloze či rychlosti otáčení motoru závisí na povaze použitého snímače. Tato informace může být, např. v případě použití inkrementálního



Obr. 2.2: Pulsně šířková modulace

enkodéru v podobě napěťových pulzů, či v případě absolutního enkodéru zase v podobě diskrétních stavů, reprezentovanými napěťovými digitálními úrovněmi na sadě pinů. Z povahy těchto signálů lze pro zpracování využít hardwarové periferie vyhodnocující signály v čase, např. časovač či obecných vstupních pinů mikrokontroléru (často zvaných GPIO - General Purpose Input Output), kde vyhodnocení probíhá na straně CPU (Central Proccessing Unit).

Z pohledu optimálního využití zdrojů mikrokontroléru je přínosnější pro zpracování vstupních či generaci výstupních signálů využít k tomu specializované periferie či moduly. Opačný přístup ponechává zbytečnou zátěž na CPU a ubírá tak z procesorového času, jenž by mohl být využit např. pro výpočty samotných řídících algoritmů a dalších úloh.

Potřebné periferie a moduly mikrokontroléru musí tedy plnit následující úlohy:

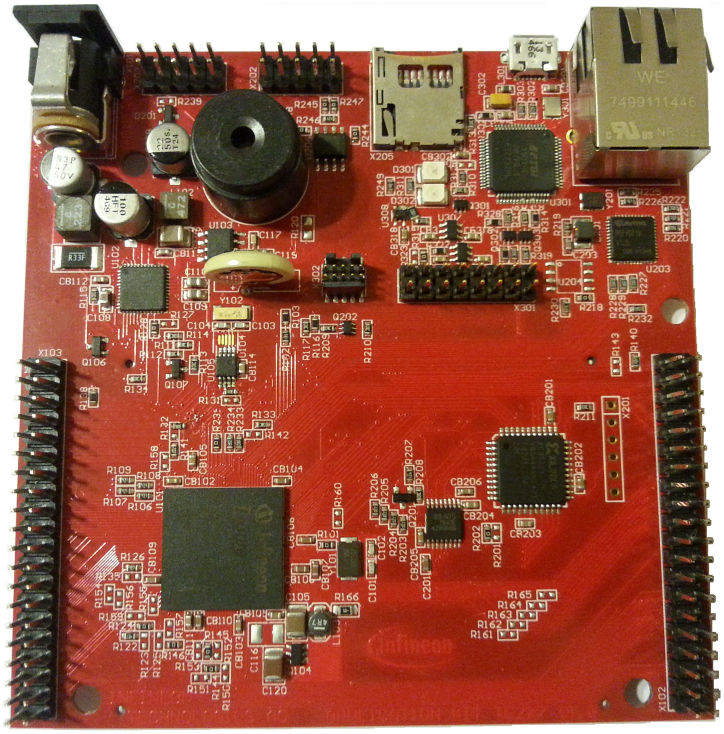
- generace signálů s modulovanou pulsní šířkou (PWM)
- převod analogových signálů na digitální (ADC)
- zpracování pulsních signálů v čase (timer)
- čtení digitálních vstupů (GPIO)

Toto je tedy jen obecný výčet, kdy konkrétní použité periferie a moduly mikrokontroléru budou představeny v následujících kapitolách.

3 CÍLOVÝ HARDWARE

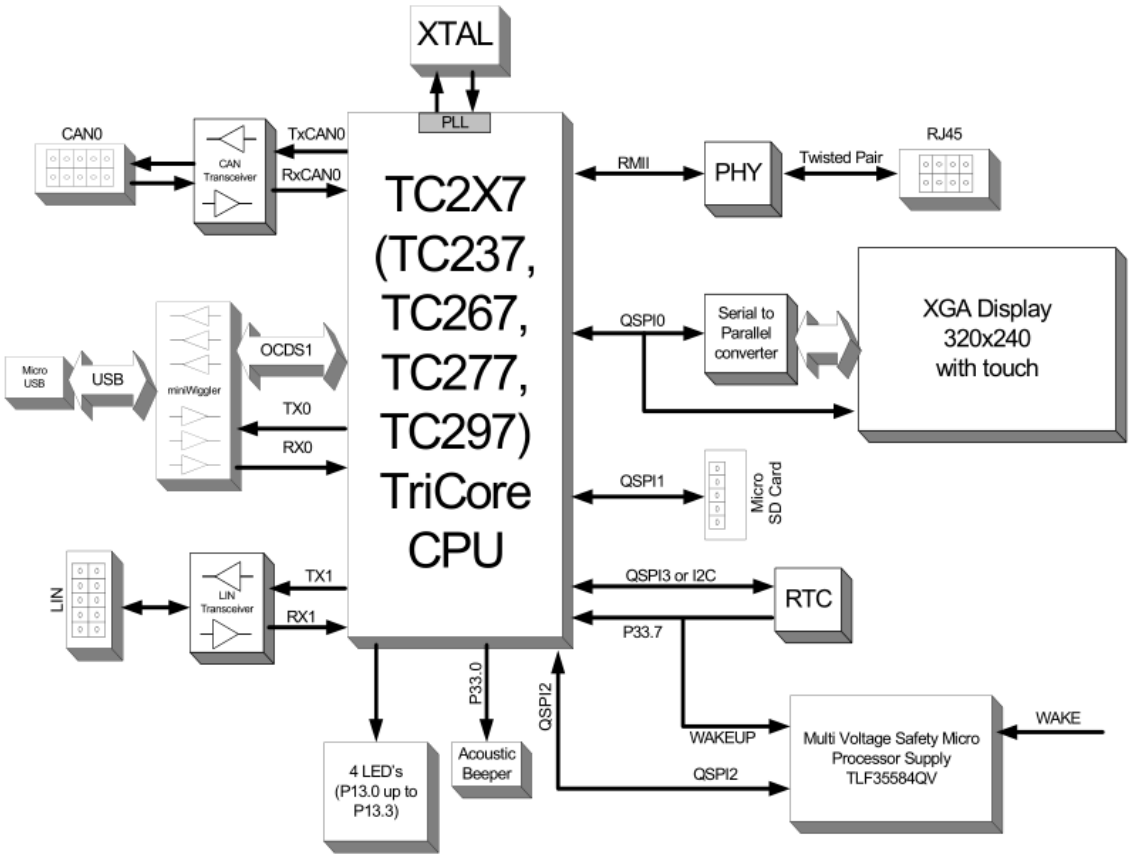
AURIX Application Kit TC277 od společnosti Infineon představuje na trhu nízkonákladovou variantu vývojové platformy postavené na 32-bitové AURIX více jádrové TriCore rodině mikrokontrolérů, konkrétně TC277, jehož zadní stranu ukazuje obrázek 3.1. Za účelem usnadnění vývoje samotných aplikací je kit osazen rozmanitým množstvím komponent a periferií, jakými jsou např.:

- LCD displej
- slot mini SD karet
- vysokorychlostní CAN vysílače
- převodník USB UART
- LIN vysílač
- 20MHz krystal
- USB miniWiggler JDS pro snadné odlaďování
- čtyřmi signalizačními LED
- RTC s alarmem nebo
- více napěťově-úrovňovým bezpečnostním napájecím mikroprocesorem TLF35584 jak je možné vidět na blokovém diagramu 3.2 [3].



Obr. 3.1: AURIX Application Kit TC277 - zadní strana

Aplikační kit může být napájen buďto prostřednictvím napájecího konektoru (3.0 až 40V) či USB (5V). Jsou požadovány tři různě napěťové úrovně +1,3V, +3,3V a +5V pro mikrokontrolér, CAN a ADC. Ty poskytuje inteligentní napájecí mikroprocesor TLF35584QV, s nímž je komunikováno prostřednictvím SPI, jak je možné vidět na obrázku 3.2 a který dále poskytuje služby jako monitorování těchto napěťových úrovní, vyvolání resetu mikrokontroléru či funkci watchdogu [4].



Obr. 3.2: Blokový diagram aplikačního kitu [4]

Společně s vývojovým kitem je poskytováno také vývojové prostředí Free TriCore Entry Tool Chain s nimž lze zdrojové kódy snadno kompilovat, prostřednictvím USB portu přeprogramovat paměť mikrokontroléru a následně také odladovat, bez potřeby či nutnosti dalších nástrojů. Součástí tohoto prostředí jsou také vzorové příklady jednoduchý aplikací prezentující obsluhu a práci s časovači, přerušeními, vstupně výstupními porty či LCD displejem.

K zahájení vývoje aplikací pracujících s cílovými procesory AURIX TC27x od firmy Infineon tedy postačuje pouze výše zmiňovaný AURIX Application Kit TC277 a vývojové prostředí Free TriCore Entry Tool Chain.

4 AUTOSAR

Zkratka AUTOSAR (AUTomotive Open System ARchitecture) reprezentuje projekt zabývající se vývojem otevřeného softwarového standardu pro vestavěné elektronické řídící systémy v automobilech.

Počátky projektu sahají do roku 2002, kdy přední světoví výrobci automobilů jako jsou BMW nebo Volkswagen, započali diskusi na téma společných výzev a cílů v této oblasti. Technický tým jehož úkolem bylo stanovit implementační strategii byl zřízen koncem téhož roku. V následujících letech se do programu jako klíčoví partneři zapojili také společnosti Ford Motor Company, Toyota Motor Corporatoin, General Motors a jiné [2].

Mezi další významné účastníky nejen z oblasti výroby automobilů, patří také společnosti působící na poli výroby polovodičových součástek, jako je např. Infineon, nebo vývoje softwaru, kam se řadí např. Elektrobit.

Motivace

K hlavním motivačním prvkům pro zahájení vývoje AUTOSARu patří zejména zvyšující se aplikační nároky na samotná vozidla. Ty jsou především z uživatelské a legislativní sféry a mohou být mnohdy protichůdné. Z legislativního pohledu jde o environmentální a bezpečnostní prvky. Z pohledu cestujících zase o pohodlí a zábavu. Požadavky řidiče patři do oblastí navigace a asistence v kriticky jízdních situacích. Z těchto důvodu se také staly současné vestavěné elektronické řídící systémy natolik složité, že je potřeba změna technologického přístupu k uspokojivému zvládnutí všech výše zmíněných požadavků [2].

Souhrnně lze tedy říci, že zvyšující se nároky na šetrnost k životnímu prostředí, bezpečnost cestujících, dynamické jízdní vlastnosti a komfortní prvky se odráží ve stále složitějším návrhu a komplexnosti vestavěných elektronických řídících systémů. Zajištění pokračování tohoto rostoucí trendu v požadovaných funkčních oblastech a zároveň snížení nákladů na vývoj takových systémů vyžaduje větší míru modularity, standardizace a metodologie v oblasti návrhu softwaru.

Cíle

K základním cílům patří zejména snazší vývoj, implementace, rozšiřitelnost a přenositelnost aplikačního softwaru takovýchto systémů. Splnění následujících podmínek je proto zásadní.

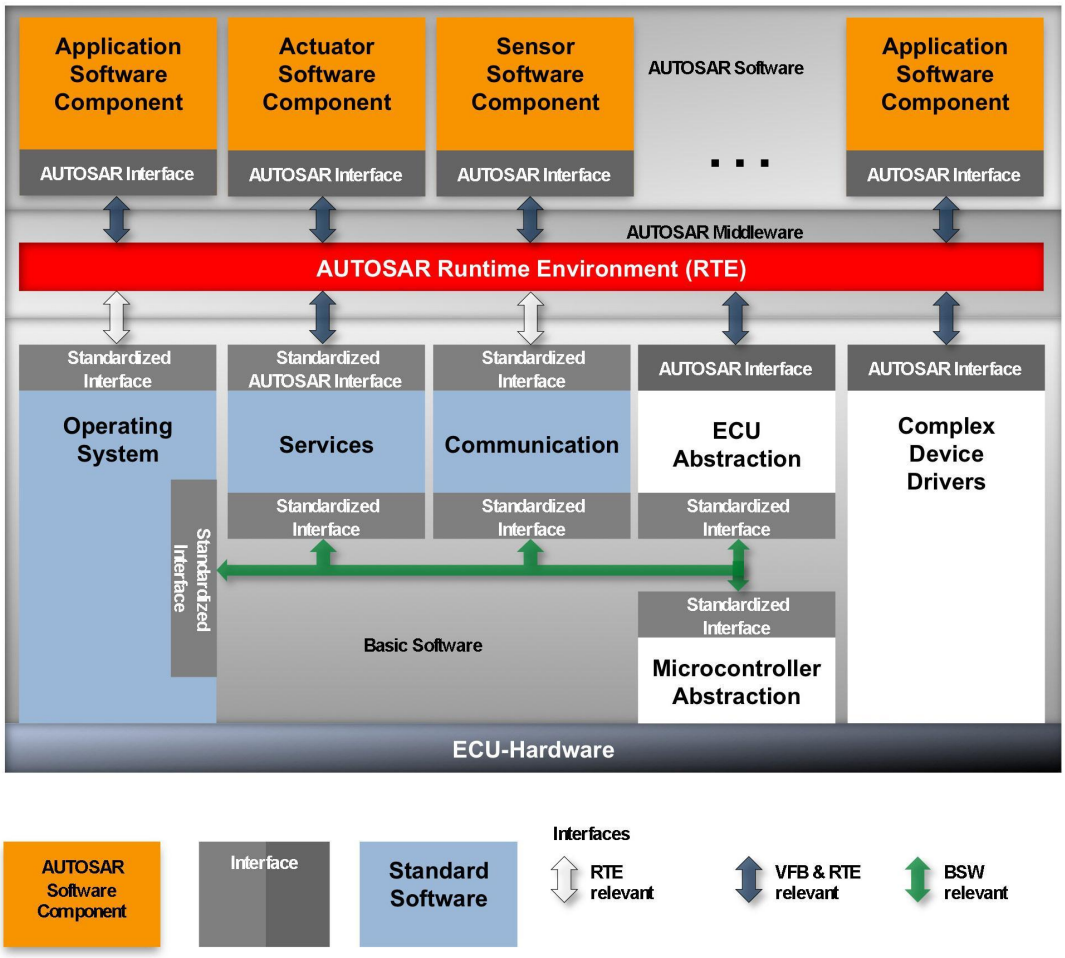
- Standardizace základního softwarového vybavení řídících jednotek a jeho rozhraní.
- Definice otevřené, vrstvené softwarové architektury.

Díky tomu dojde k oddělení hardwarově závislého softwaru od toho aplikačního a také ke zvýšení rozšiřitelnosti, flexibility integrace a přenositelnosti aplikačního softwaru.

AUTOSAR tak posouvá tradiční návrhové schéma založené na vývoji softwaru pro jednotlivé řídící jednotky ECU (Electronic Control Unit) více k aplikačně orientovanému návrhu [2].

4.1 Architektonické prvky [2]

Architektura AUTOSAR definuje třívrstvou softwarovou strukturu. Ta se skládá z nejvyšší aplikační, střední komunikační RTE (Runtime Environment) a nejnižší vrstvy základního programového vybavení BSW (Basic Software). Tento model je pro jedinou ECU jednotku zobrazen na Obr. 4.1.



Obr. 4.1: Softwarová architektura jediné ECU jednotky modelu AUTOSAR [2]

Aplikační vrstva se skládá ze softwarových komponent formálně zvaných *SwComponentType*. Ty tvoří základní aplikační bloky v celém modelu a využívají služeb nižších dvou vrstev.

RTE představuje abstraktní komunikační vrstvu pro softwarové komponenty, které si skrze ní vyměňují data či přistupují k periferiím mikrokontroléru.

BSW obsahuje operačním systémem, modul pro komunikaci a služby a abstraktní vrstvu mikrokontroléru a samotné ECU jednotky. Umožňuje tak nepřímo přistupovat k hardwarovým periferiím mikrokontroléru.

4.1.1 Softwarové komponenty (SWC)

Elementární aplikační jednotku ztělesňují softwarové komponenty SWC, které existují v několika typech. Mezi ty nejpoužívanější patří především *AtomicSwComponentType* a *CompositionSwComponentType*.

První jmenovaný typ, *AtomicSwComponentType*, je nejmenší možnou komponentou a popisuje své chování v pojmech jako *vnitřní data* nebo *výkonné jednotky*. Se svým okolím komunikuje pomocí přesně definovaných přípojných bodů jménem *PortPrototype*. Standard AUTOSAR však neurčuje přesný počet portů, které mohou *AtomicSwComponentType* obsahovat, či podobu vnitřního chování těchto komponent.

Účelem užití *CompositionSwComponentType* je integrace a shlukování jiných softwarových komponent do celků s libovolným uspořádáním a vyšší konečnou složitostí. Jde tedy o architektonické prvky, kterými je realizována funkce rozšiřitelnosti. Ulehčují tak prohlížení nebo vytváření nové logické softwarové struktury. Zároveň však nemají žádný vliv na způsob, jakým komunikují v nich obsažené komponenty z vnějším světem a také jim nepřidávají novou funkčnost.

4.1.2 Komunikační vrtsva (RTE)

Veškerý datový tok mezi aplikačními komponentami SWC samotnými či mezi nimi a základním programovým vybavením BSW probíhá skrze tuto abstraktní komunikační vrstvu RTE. Ta dále implementuje virtuální sběrnici VFB
(Virtual Functional Bus).

Při komunikaci softwarových komponent není z jejich pohledu rozlišeno, zda dochází k přenosu dat uvnitř jediné ECU jednotky či mezi různými jednotkami. To teoreticky umožňuje přesouvat aplikace napříč jednotkami. V praxi však tuto situaci stěžují časová zpoždění vzniklá přenosem signálu[2].

Konečná podoba této vrstvy závisí na konkrétní konfiguraci dané ECU jednotky a musí jí být na míru přizpůsobena. Z tohoto důvodu se také budou jednotlivé RTE lišit mezi různými jednotkami [2].

4.1.3 Virtuální sběrnice (VFB)

Jak ukazuje obr.4.1, jednotlivé aplikační komponenty nekomunikují přímo skrze vrstvu RTE, nýbrž prostřednictvím virtuální sběrnice VFB. Standardizace její podoby umožňuje realizovat jeden z hlavních cílů, jimž je přenositelnost aplikačního softwaru. Dochází tak k oddělení aplikací od základního programového vybavení BSW (ovladače, operační systém nebo modul služeb) a hardwaru z nižší vrstvy. Tím je usnadněna integrace nových softwarových komponent, což je výhodné zejména při nárůstu složitosti celého systému [2].

Komponenty a Porty

Jak již bylo zmíněno dříve, softwarové komponenty obsahují přípojné body nazývané *PortPrototype*, které se vyskytují v následujících typech

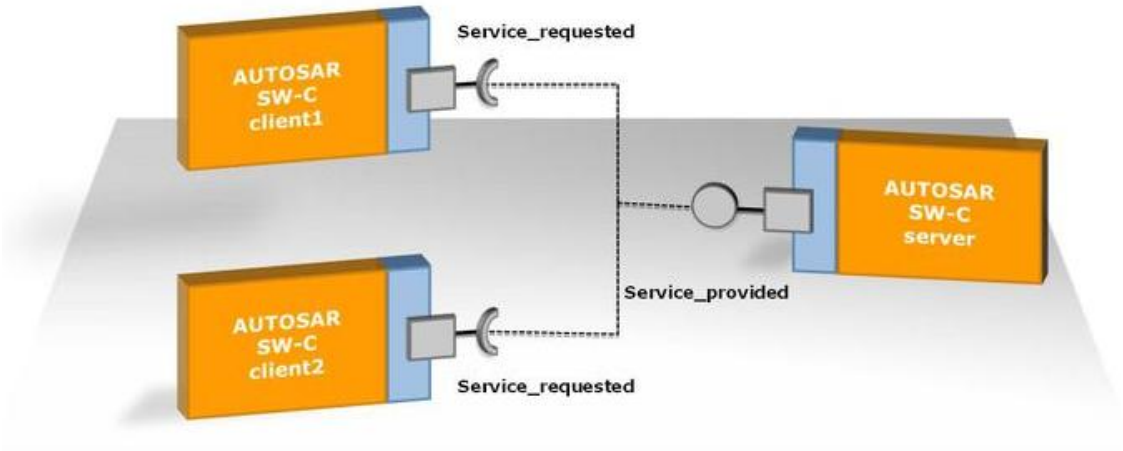
- *PPort* (Provide)
- *RPort* (Request)
- *PRPort* (ProvideRequest)

podle toho, zda poskytují, požadují či poskytují i požadují data nebo služby [2].

Tyto body dále implementují rozhraní jménem *PortInterface*, jež určuje způsob komunikace, která bývá nejčastěji buď

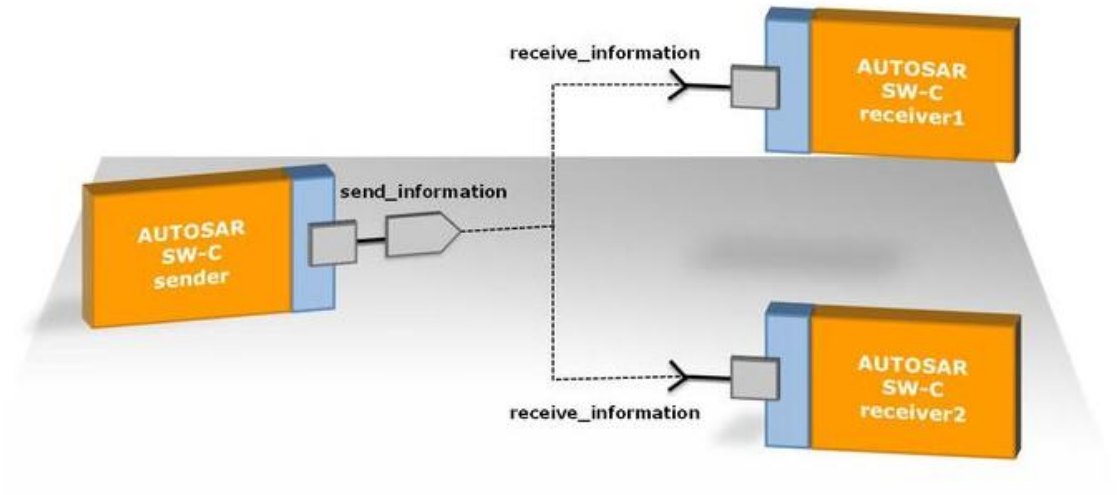
- datově orientovaná (vztah přijímač-vysílač)
- operačně orientovaná (vztah klient-server)

Na obr.4.2 jsou ukázány tři komponenty ve vztahu klient-server. Komponenta, která vystupuje jako server a tedy poskytuje rozhraní skrze port *PPortPrototype*, také implementuje definované operace. Komponenty jež požadují rozhraní (klienti) prostřednictvím portu *RPortPrototype* mohou tyto operace vyvolat.



Obr. 4.2: Klient-server komunikace [2]

Obr.4.3 znázorňuje tři komponenty ve vztahu přijímač-vysílač. Komponenta vystupující jako vysílač s portem *PPortPrototype* generuje data a to nezávisle na počtu komponent, která je požadují. Odesílatel zde poskytuje informace a je již pouze na příjemci, kdy a jak tyto informace využije.



Obr. 4.3: Datová komunikace [2]

Jednotlivé komponenty mohou obsahovat několik portů, avšak jeden port musí náležet právě a pouze jedné komponentě. Ty tak mohou současně vystupovat např. jako klient i server.

4.1.4 Základní programové vybavení (BSW) [2]

Nejnižší vrstvu AUTOSAR modelu tvoří základní programové vybavení BSW. Tato vrstva poskytuje služby softwarovým komponentám SWC a je nezbytnou součástí pro vykonání jakékoli aplikační části softwaru. Zároveň však sama o sobě neplní aplikační funkci. Tvoří ji:

- operační systém (OS),
- modul systémových služeb,
- komunikační komponenta,
- abstraktní vrstva ECU jednotky,
- komplexní ovladače CDD (Complex Device Drivers),
- abstraktní vrstva mikrokontroléru MCAL (Microcontroller Abstraction Layer).

Aplikační komponenty proto mohou přistupovat k hardwaru pouze nepřímo, prostřednictvím vrstvy BSW. Ta obsahuje ovladače k základním periferiím mikrokontroléru.

Operační systém (OS)

Jelikož je cílem vytvoření softwarové architektury, která je uplatněna ve všech funkčně-řídících oblastech (jednotkách) vozidla , standard klade požadavky také na použitý operační systém. Jsou jimi:

- statická konfigurace
- práce v reálném čase
- prioritně řízené plánování úloh
- bezpečnostní funkce pracující v době vykonávání programu
- práce na nízkoúrovňových kontrolérech bez nutnosti dalších zdrojů

AUTOSAR umožňuje použití vlastních operačních systémů. Ty však musí splňovat výše zmíněné požadavky a zároveň dodržovat a podporovat všechna potřebná rozhraní. Jako standard je doporučen OSEK OS.

Abstraktní vrstva mikrokontroléru MCAL

Přístup k hardwaru mikrokontroléru je veden skrze vrstvu MCAL s cílem omezit přímý přístupu k registrům z vyšších softwarových vrstev. Její součástí je proto správa (ovladače) periferií mikrokontroléru jako jsou:

- digitální vstupy a výstupy (DIO)
- analogově digitální převodník (ADC)
- modulátor šířky pulsu (PWM)
- paměť EEPROM (EEP)
- paměť flash (FLS)
- komparační jednotka (OCU)
- watchdog timeru (WDT)
- I2C modul (IIC)

Tím je vytvořeno standardizované rozhraní pro další moduly vrstvy BSW, jak je možné vidět na obrázku 4.1.

Systémové služby, komunikační modul a komplexní ovladače (CDD)

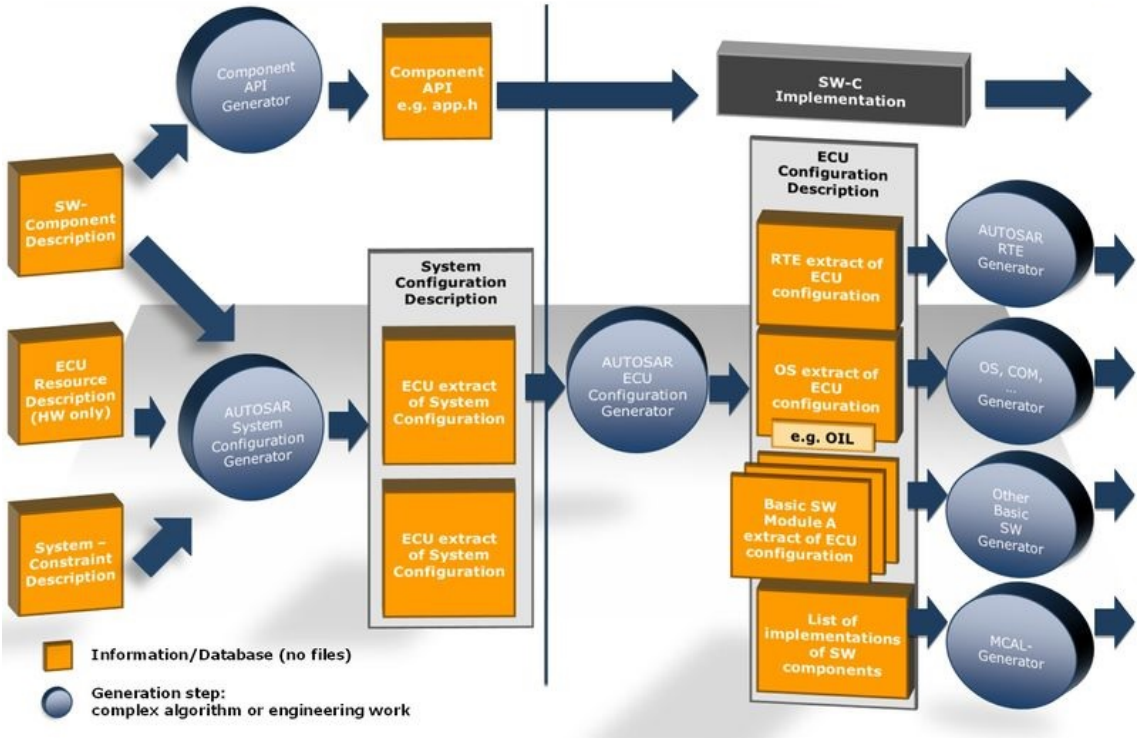
Systémové služby plní diagnostické funkce jako je např. správa paměti NVRAM.

Komunikační komponenta umožňuje obsluhu sběrnic jako jsou CAN, LIN FlexRay a další. Obecně tedy zajišťuje službu správy sítí.

Komplexní ovladače naproti tomu umožňují přistupovat přímo k hardwaru. Takováto funkcionalita je vhodná zejména pro aplikace, jejíž průběh je kriticky závislý na práce se zdroji.

4.2 Metodologie [2]

Informační tok návrhových kroků k vytvoření systému na bázi technologie AUTOSAR ukazuje obrázek 4.4. V levé polovině se vyskytují informace o celém systému a pravá již symbolizuje jedinou ECU jednotku. Proces generace konfiguračního kódu uplatňuje přístup tzv. shora dolů (v případě obrázku 4.4 zleva doprava), kde na začátku stojí popis celého systému a na jejím konci se nachází již potřebný kód.



Obr. 4.4: Informační tok metodologie AUTOSAR [2]

Jako vstupní informace jsou použity popisy softwarových komponent SWC, hardwarových zdrojů pro jednotlivé ECU jednotky a možných omezení systému. Jejich následné zpracování provádí generátor systémové konfigurace a generátor API komponent. Výsledkem je popis celého systému, kde vazby mezi jednotlivými ECU jednotkami musejí být validní a logicky kompatibilní.

Extrakcí ze systémové konfigurace je získán popis samotných ECU jednotek. Ten definuje podobu vrstvy RTE, konfiguraci OS a BSW a seznam implementovaných softwarových komponent. Z těchto informací je již možné vygenerovat kód pro moduly BSW (tedy také MCAL) a implementovat SWC kód.

5 EB TRESOS STUDIO

EB tresos studio je **vývojový nástroj** zacílený na oblast základního programového vybavení BSW vestavěných řídících jednotek ECU s podporou standardu AUTOSAR.

Jeho hlavními funkcemi jsou **konfigurace** základního programového vybavení BSW (např. jednotlivých modulů-periferií mikrokontroléru), **validace** takto vzniklých konfiguračních dat a následné **generování kódu**, popisujícího vytvořenou konfiguraci podle standardu AUTOSAR.

EB tresos Studio je postaveno na open source vývojové platformě Eclipse, která vyniká především svou rozšiřitelností pomocí pluginů. Tímto způsobem je tak možné rozšiřovat tresos Studio o nové uživatelské funkce.

5.1 Základní Funkce

Konfigurační editor

Tento nástroj slouží jako grafické uživatelské rozhraní GUI k vytváření a editaci konfigurací pro BSW moduly podle standardu AUTOSAR. Podobu tohoto grafického rozhraní ukazuje obr.5.1.

Ke snadnější orientaci v celkové konfiguraci jsou jednotlivé parametry popsány tzv. XPath adresou. Ta určuje přesné umístění parametru v celé síti parametrů vytvářené konfigurace.

Výsledná konfigurace je pro potřeby EB tresos Studia uložena ve formátu XMD.

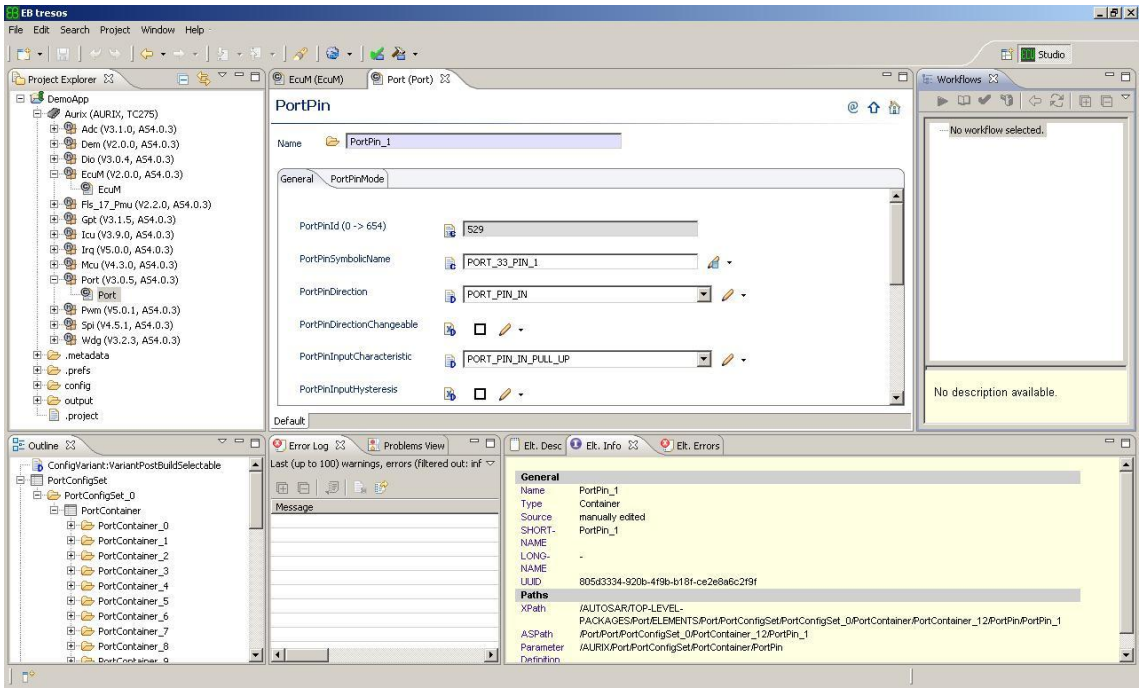
Validace dat

EB tresos Studio provádí validaci vytvářené konfigurace. Na pozadí jsou neustále vyhodnocovány vzájemné vztahy mezi parametry a jejich neslučitelnost je uživateli bezprostředně oznámena pomocí zprávy o chybě a XPath adresy daného uzlu. Díky tomu je uživatel schopen okamžitě identifikovat konfliktní parametry.

Generování kódu

Na základě vytvořené a verifikované konfigurace jsou pomocí generátorů kódu vytvořeny výstupní zdrojové kódy s dodržením standardu AUTOSAR.

Tyto generátory musí být dodány externím způsobem prostřednictvím pluginů.



Obr. 5.1: Grafické uživatelské rozhraní EB tresos Studia

Podpora příkazové řádky

Ovládání některých funkcí jako např. importování konfigurace projektu, verifikace, generování výstupního kódu a další jsou podporovány z příkazové řádky. To umožňuje využití služeb EB tresos Studia i v jiných vývojových prostředích IDE.

Workflow

Průvodcem vytváření konfigurace je nástroj Workflow, který může být opět dodán externě pomocí pluginů.

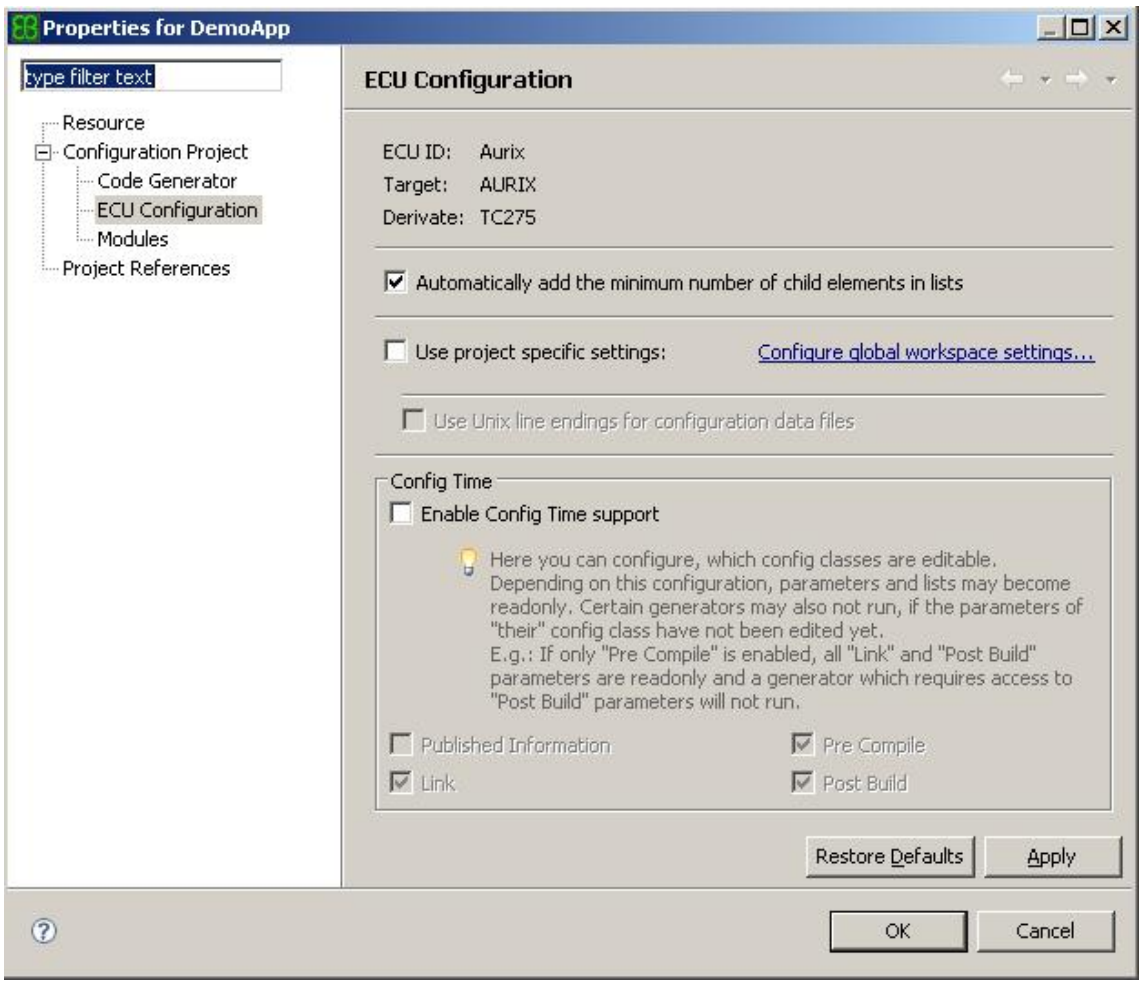
5.2 EB tresos Projekt

Jak již zbylo zmíněno dříve, k započetí vytváření konfigurace MCAL komponenty (oladačů a HW periferií mikrokontroléru) je nejprve potřeba integrace patřičných generátorů v podobě pluginů do tresos Studia.

5.2.1 Konfigurační třídy [5]

Ve standardu AUTOSAR náleží každý konfigurovatelný parametr vždy do jedné z následujících konfiguračních tříd: *PublishedInformation*, *PreCompile*, *Link* a *PostBuild*.

Při vytváření nového projektu/konfigurace je možné přiřadit projektovému parametru jménem *doba konfigurace* (*config time*) několik z výše uvedených tříd. Tím je určeno, které ze zvolených konfiguračních tříd a jim náležících parametrů je možné editovat. Podle této volby jsou také spouštěny příslušné generátory kódu. Okno s nastavením možností projektu ukazuje obr.5.2

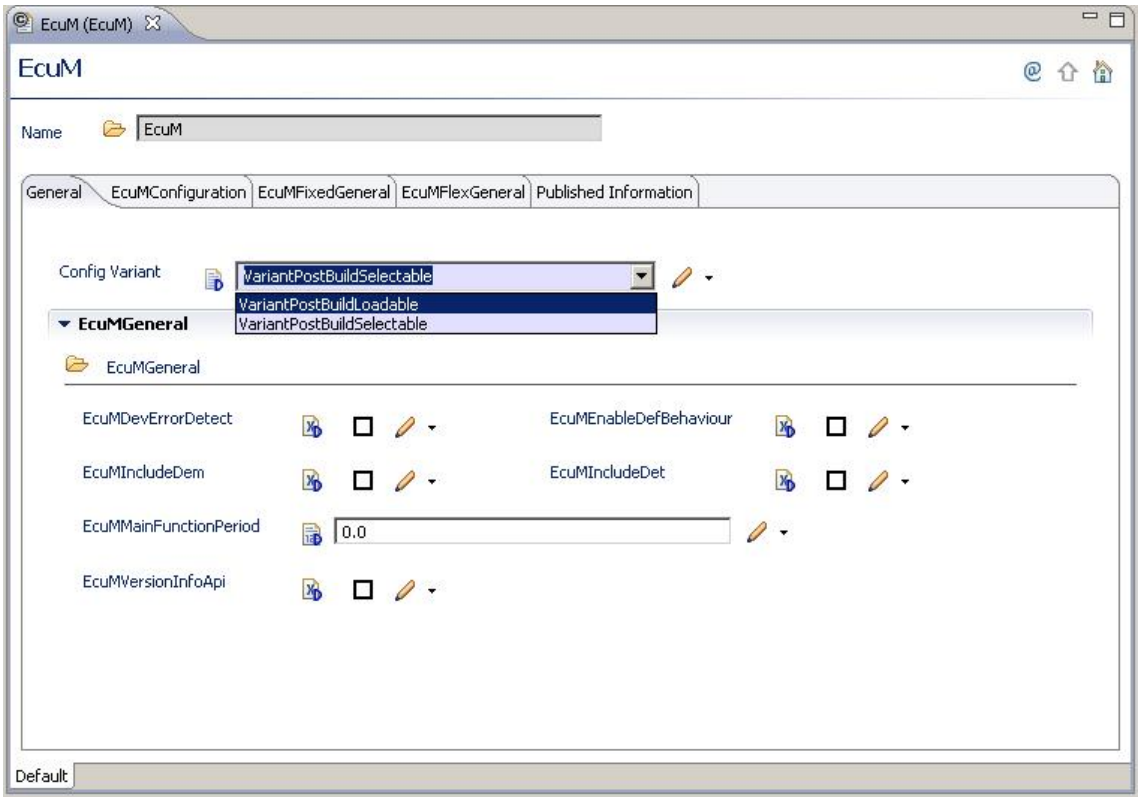


Obr. 5.2: Volby nastavení konfigurace/projektu ECU jednotky

Všem modulům je možné nastavit volbu tzv. *konfigurační varianty*. Ta určuje, do jaké z konfiguračních tříd modelu AUTOSAR budou náležet všechny parametry daného modulu (např. periferie mikrokontroléru). Volbu tohoto nastavení ukazuje obrázek 5.3.

Třída *PreCompile* stanovuje, že je modul nakonfigurován ještě před samotnou kompilací zdrojového kódu.

Třída *Link* definuje, že je modul konfigurován během linkovací fáze. To znamená, že objektový kód daného modulu obdrží část své konfigurace z jiného souboru objektového kódu.



Obr. 5.3: Nastavení konfigurační varianty ECU modulu

Třída *PostBuild* určuje, že moduly obdrží své konfigurační parametry buď naháním samostatného konfiguračního souboru do paměti jednotky ECU (*PostBuild-Loadable*) nebo definováním několika konfiguračních sad a vybráním jedné během vykonávání programu (run-time) (*PostBuild-Selectable*).

5.2.2 Konfigurovatelné moduly

Při vytváření konfigurace ECU jednotky je možné nastavovat pouze ty moduly, jež byly dodány do tresos studia v podobě pluginů. Nejzákladnější úlohou je v tomto smyslu konfigurace MCAL vrstvy a tedy konfigurace periferií mikrokontroléru. Tím je také určena skupina konfigurovatelných modulů.

5.3 Výstupní kód

Výstupní generovaný konfigurační kód je uložen do souborů konvenčně pojmenovaných

- *Module*_Cfg.h
- *Module*_PBCfg.c

- *Module*_LCfg.c

kde *Module* je nahrazeno zkratkou daného modulu. Každý parametr je tak podle své náležitosti konfigurační třídě (7.2.1) implementován v patřičném souboru.

5.4 Praktické poznámky

- Pokud nedojde po přidání nového modulu do vytvářené konfigurace k vyplnění záložky PublishedInformation, je potřeba přepnout Worspace znovu na ten sám. To se provede pomocí File -> Switch Worspace. Dokud nedojde k opravě vzniklé situace, bude ve výpisu chyb o tomto stavu zobrazena zpráva a generování výstupního kódu nebude možné.
- Při vytváření nové konfigurace je vhodné v projektovém nastavení zvolit možnost *automatického přidání minimálního množství potřebných elementů v seznamech*, viz obr.5.2
- Při vytváření nové konfigurace je vhodné v projektovém nastavení neaktivovat možnosti *konfigurační doby* (viz. obr.5.2). Veškeré parametry konfigurace tak budou přístupné a editovatelné.

6 FREE TRICORE ENTRY TOOL CHAIN

Jako integrované vývojové prostředí bylo použito Free TriCore Entry Tool Chain IDE založené na platformě Eclipse.

Při vytváření projektu však nestačí pouze integrovat vygenerovaný konfigurační kód spolu s ovladači a projekt zkompilovat. Je zde potřeba podniknout jisté kroky, jako definování globálních symbolů na úrovni projektu, eliminace případně zachování implicitních start-up kódu pro procesor a s ní související volba použitého linker scriptu případně nastavení cest pro standardní hlavičkové soubory.

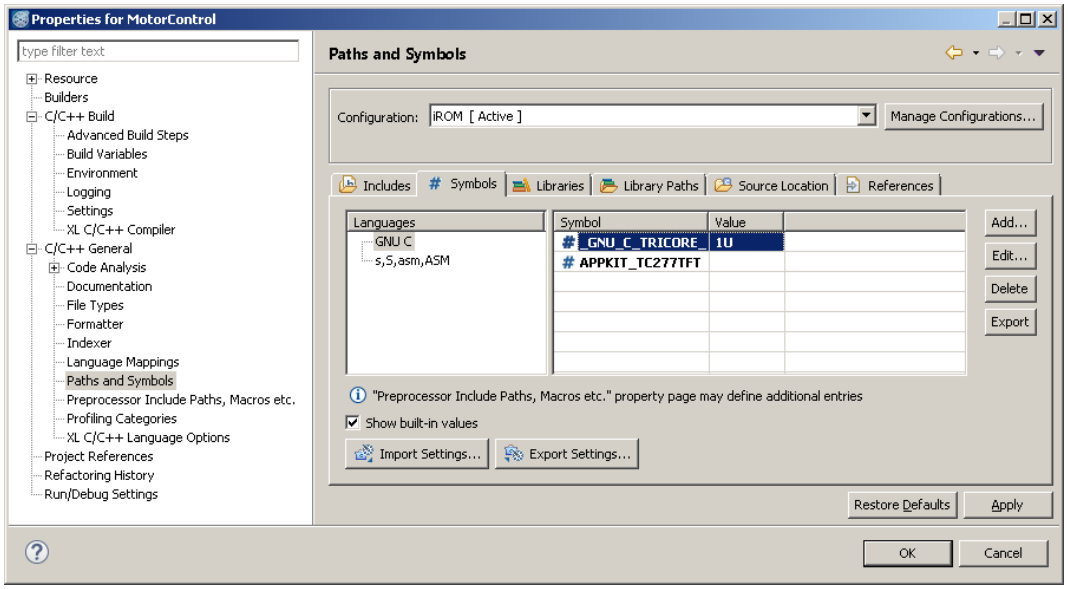
Uživatel může také požadovat strukturovat projekt jiným způsobem, než poskytují standardní demo projekty, dostupné z IDE.

Tyto kroky budou tedy následně krátce popsány.

6.1 Nastavení globálních symbolů na úrovni projektu

6.1.1 Uživatelské symboly

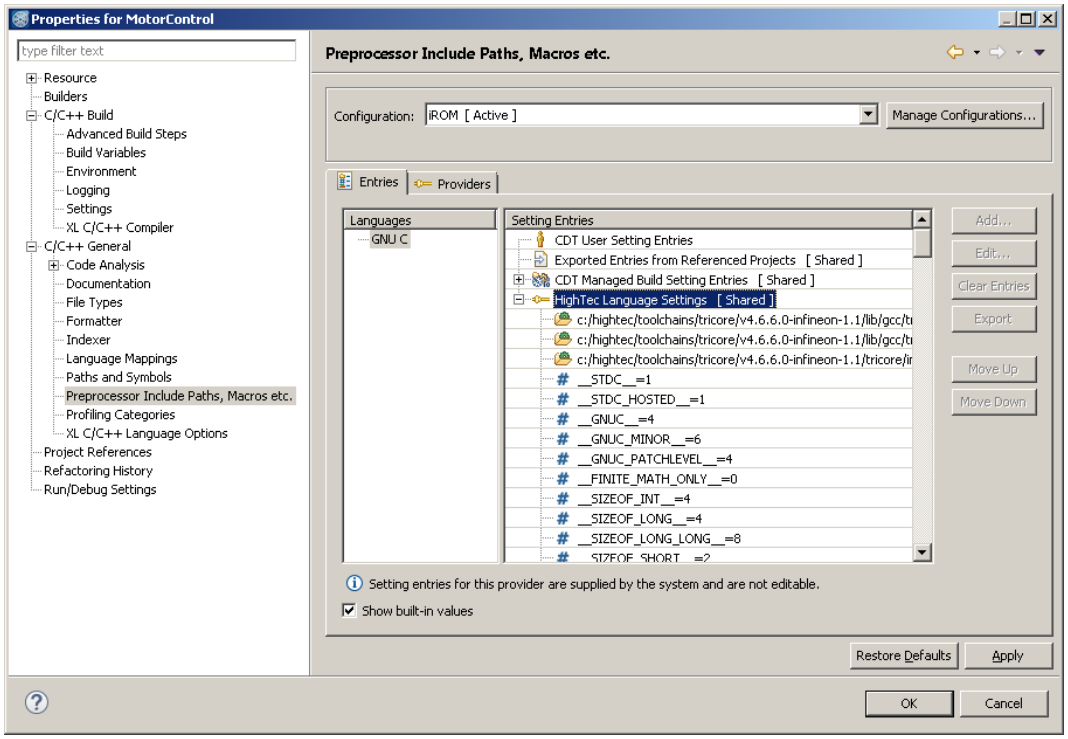
Globální symboly pro celý projekt lze definovat v okně 'properties' (pravá myš na kořenový adresář projektu v záložce 'Project Explorer'), skupina 'C/C++ General', podskupina 'Path and Symbols' a následně karta 'symbols'. Zde se nalézají uživatelsky definované symboly, viz. obrázek6.1.



Obr. 6.1: Definování uživatelských globálních symbolů na úrovni projektu

6.1.2 Systémové symboly

Systémové globální symboly lze nalézt v téže skupině jako uživatelské symboly, podskupina je však nyní nazvaná 'Preprocessor Includes Paths, Macros etc.', karta 'Entries' viz. obrázek 6.2.

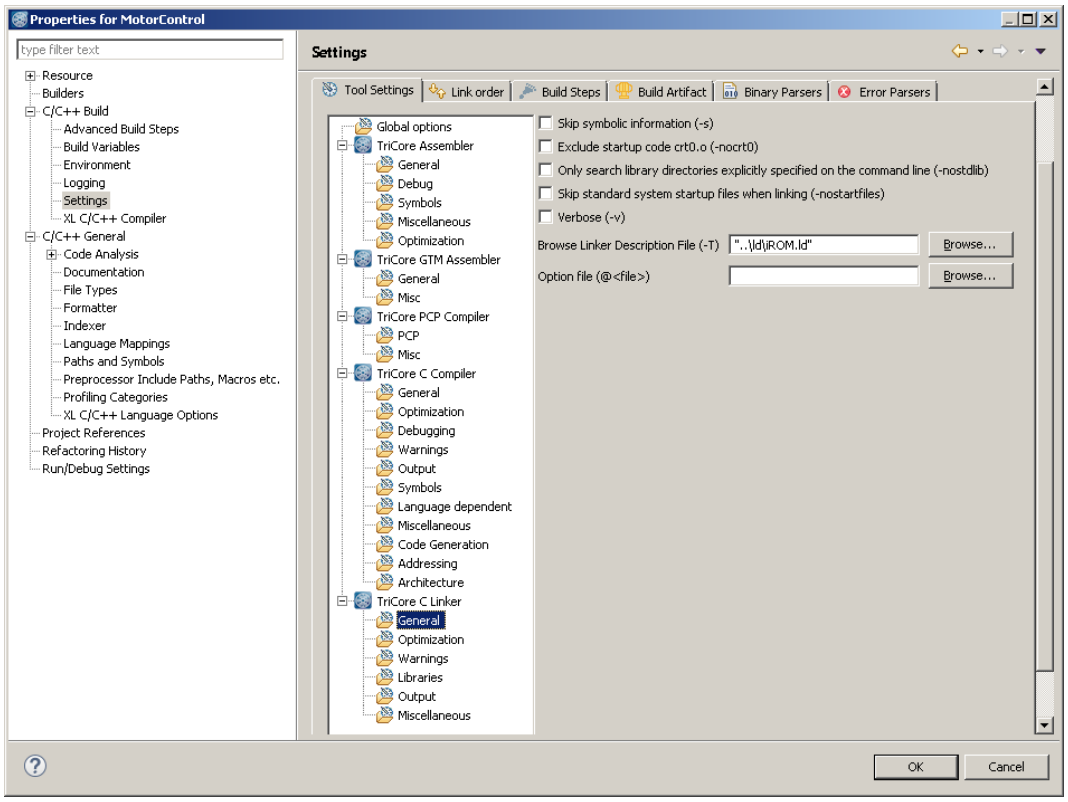


Obr. 6.2: Systémové globálních symboly na úrovni projektu

6.2 Volba použitého start-up kódu a linker skriptu

Možnost použití buďto standardních impicitních start-up kódů pro daný procesor nebo externích start-up kódu a s nimi také svázaný linker skript se nastavuje v okně 'Properties' daného projektu. Sekce 'C/C++ Build', podsekce 'Settings', karta 'Tool Settings', skupina 'TriCore C Linker' volba 'General' viz. obrázek 6.3.

To může být žádoucí při využití ovladačů MC-ISAR které samy obsahují vlastní start-up kód, linker skript a soubor mapující části kódu do patřičných sekcí paměti (memory map soubor).

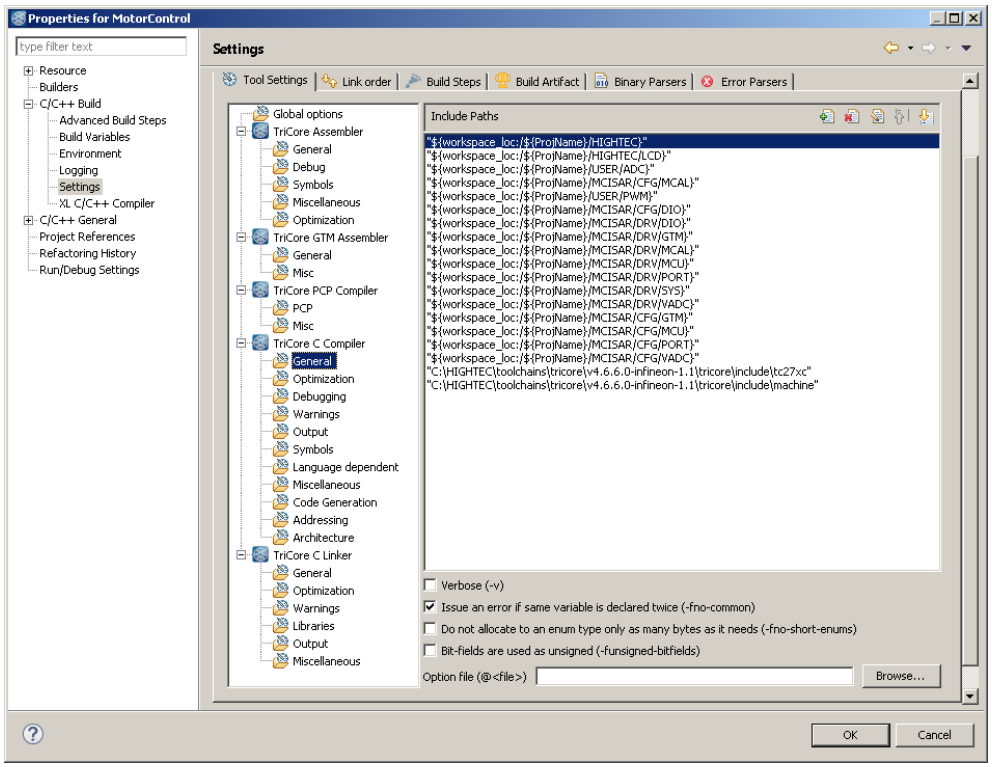


Obr. 6.3: Volba použitých start-up kódu a linker skritpu

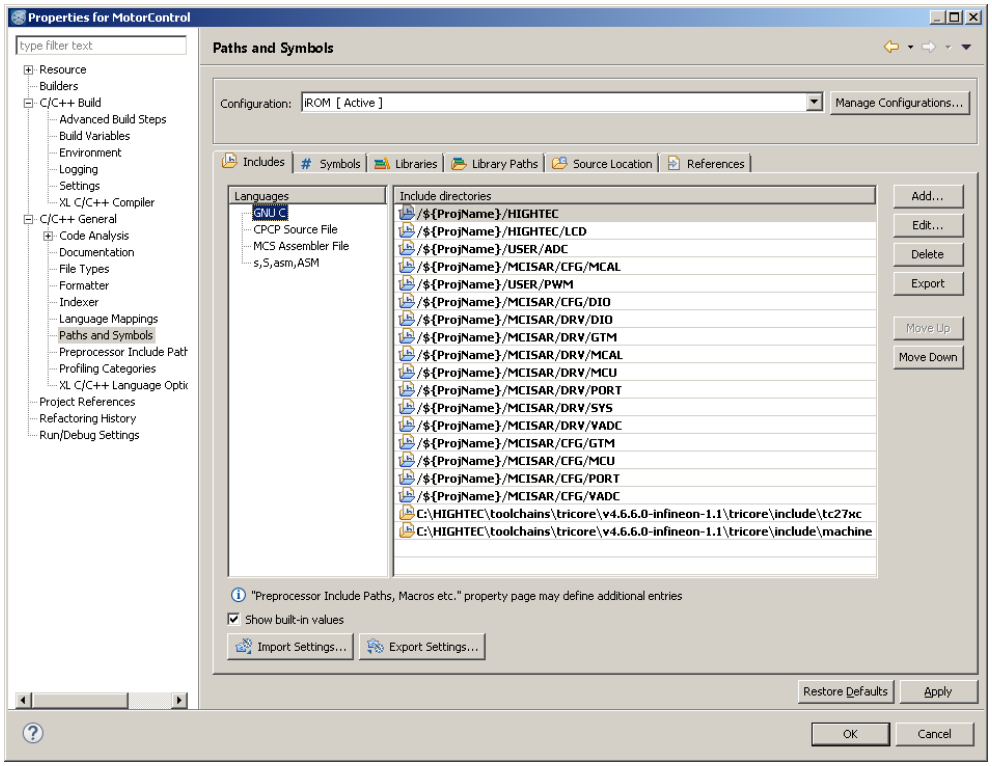
6.3 Nastavení cest pro hlavičkové soubory

Nastavení cest pro hlavičkové soubory jde provést několika způsoby. Pro projekt, zajišťující přenositelnost je výhodnější definovat umístění hlavičkových souborů relativně vůči kořenovému adresáři projektu, než-li použití absolutních cest, které se mohou při migraci kódu do jiného projektu změnit. To je možné nastavit ve volbě podle obrázku 6.4.

Další možností jak nastavit umístění hlavičkových souborů ukazuje obrázek 6.5. Tato volba se osvědčila jako spolehlivá a stabilní pro nastavení 'systémových - neuživatelských' cest, kdežto první varianta se jeví jako spolehlivá pro nastavení uživatelských cest.



Obr. 6.4: Nastavení cest pro uživatelské hlavičkové soubory



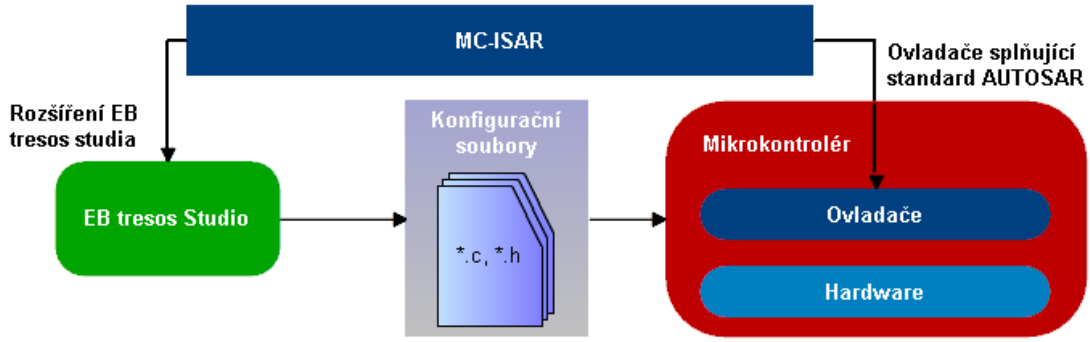
Obr. 6.5: Nastavení cest pro systémové hlavičkové soubory

7 OVLADAČE MC-ISAR

Jak již bylo zmíněno dříve, k započetí vytváření konfigurace vrstvy MCAL a hardwarových modulů mikrokontroléru je nutné nejprve dodat potřebná rozšíření do EB tresos studia. Přitom je však nutné mít stále na zřeteli, že tato rozšíření musí respektovat na jedné straně standard AUTOSAR a na straně druhé také povahu konkrétně užitého hardwaru.

Po vytvoření popisu požadované konfigurace, její validace a vygenerování kódu je však stále zapotřebí výkonná část kódu, jenž je schopná uvést tato konfigurační nastavení v platnost - tedy ovladače.

Je proto zřejmé, že hledaný produkt musí splňovat požadavky standardu AUTOSAR a zároveň respektovat specifikace cílového hardwaru. Dále musí být dostupný ve dvou provedeních, jednak jako rozšíření pro EB tresos studio a dále také v podobě samotných ovladačů použitelných v mikrokontroléru jak je možné vidět na obrázku 7.1. Takovýto produkt poskytuje samotný výrobce cílového hardwaru Infineon pod označením MC-ISAR - MicroControler Infineon Software ARchitecture.



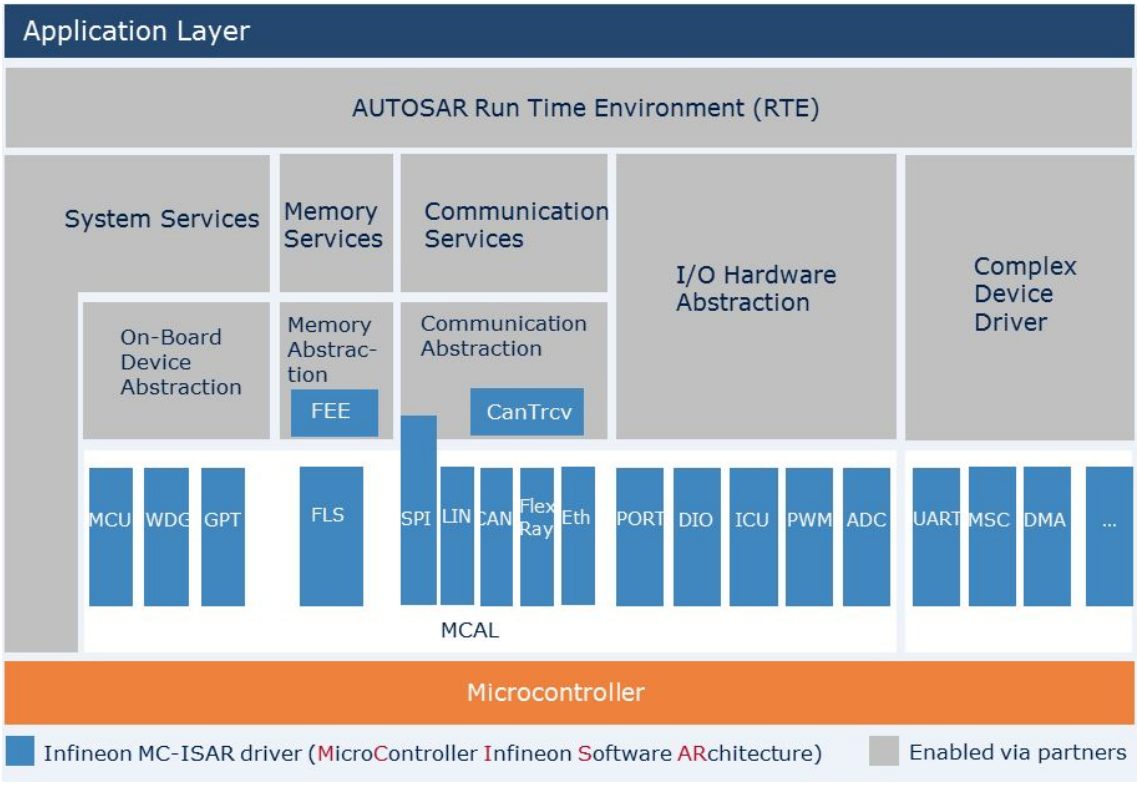
Obr. 7.1: Uplatnění/užití produktu ovladačů MC-ISAR

Strukturu těchto ovladačů a jejich umístění v modelu AUTOSAR ukazuje obrázek 7.2. Ovldače jednotlivých periferií jsou společností Infineon distribuovány v tzv. balících, jenž reprezentují danou skupinu podle její společné funkcionality. Jsou jimi balíky základní (Basic), komunikační (COM), paměťový (MEM) a balík komplexních ovladačů (CD).

7.1 Základní balík ovladačů MC-ISAR [6]

Součásti základního balíku jsou ovladače pro

- MCU - základní inicializace mikrokontroléru, vypnutí, reset.



Obr. 7.2: Struktura ovladačů MC-ISAR [6]

- WDG - změny operačních módu, manipulace s watchdog.
- GPT - časovač.
- PORT - inicializace portů.
- DIO - čtení/zápis na piny, porty a skupiny portů.
- ICU - zpracování PWM signálu (čítání pulsů, měření frekvence a periody)
- PWM - ovladač generování PWM signálu.
- ADC - analogově digitální převody.
- MCAL - start-up kód, řízení přístupu do SFR
- ECUm - abstrakce ECU jendotky, inicializace dílčích modulů

Komparací tohoto výčtu společně s rozborem kapitoly 2 je zřejmé, že funkcionalita zakladního balíku je pro cílovou aplikaci dostačující.

7.2 Konfigurační třídy parametrů a varianty ovladačů

7.2.1 Konfigurační třídy parametrů [7]

Jak již bylo zmíněno v kapitole 5.2.1, konfigurovatelné parametry mohou patřit konfiguračních do tříd *PreCompile, Link a PostBuild*, čímž je určeno jejich implementační řešení a tím také doba, kdy jsou aktivně uplatněny.

PreCompile

Parametry konfigurační třídy *PreCompile* jsou implementovány pomocí direktivy preprocesoru, jak ukazuje výpis 7.1 a proto musí být jejich nastavení známo před samotnou kompilací. Jejich užití je možné pouze s distribucí ovladačů v podobě zdrojových kódů a jsou vhodné zejména pro konfiguraci samotných ovladačů, kdy mohou povolovat určitou funkcionalitu nebo ji v případě nepotřeby neuplatňovat ve výstupním kódu.

Výpis 7.1: Příklad implementace parametrů z konfigurační třídy *PreCompile*

```
/* An example of setting a PreCompile parameter */
#define MCISAR_DRIVER_API    STD_ON

/* An example of using PreCompile parameter */
#if ( MCISAR_DRIVER_API == STD_ON )
/* API function prototype */
void DriverFunction( void );
#else
 /* API is not used */
#endif
```

Link

Parametry patřící do konfigurační třídy *Link* nejsou uplatněny při procesu kompilace, ale až v průběhu linkování. Mohou tak být použity spolu s distribucí ovladačů v podobě knihoven, kdy konfigurační parametry samotné jsou umístěny v externím souboru který je následně přilinkován k ovladačům v průběhu procesu sestavení projektu.

PostBuild

Parametry náležící konfigurační třídě *PostBuild* neovlivňují samotný proces kompilace ovladačů a uplatňují se až při vykonávání programu (slouží jako vstupní parametry pro funkce ovladačů při inicializaci). Do této skupiny proto patří parametry u nichž může být výhodné přeprogramování v době run-time (např. různé konfigurační varianty nastavení periferií mikrokontroléru). Příklad užití parametrů třídy *PostBuild* ukazuje výpis 7.2.

Výpis 7.2: Příklad implementace parametrů z konfigurační třídy *PostBuild*

```
/* Initialization of PostBuild parameter */
PostBuildParam_t moduleCfg = {
    MODULE_ON, CLK_DIV_5 };

/* Using of PostBuild parameter */
ModuleInit( &moduleCfg );
```

7.2.2 Konfigurační varianty ovladačů [7]

Podobně jako je tomu u konfiguračních tříd parametrů, ovladače jsou taktéž distribuovány ve třech variantách nazvaných *Variant PC*, *Variant LT* a *Variant PB*.

- *Variant PC* se omezuje pouze na správu parametrů třídy *PreCompile*.
- *Variant LT* umožňuje kombinovat parametry tříd *PreCompile* a *Link Time*.
- *Variant PB* zachází s parametry tříd *PreCompile* a *PostBuild*.

Samotné konfigurovatelné parametry se mohou nacházet v různých konfiguračních třídách podle toho, v jaké variantě ovladačů jsou použity. Tzn., že jeden a ten sám parametr může být implementován např. ve variantě ovladačů *Variant PC* ve třídě *PreCompile*, kdežto pro distribuci ovladačů *Variant PB* ve třídě *PostBuild*.

7.3 MC-ISAR Demo

Součástí distribuce balíku MC-ISAR je také kromě potřebných rozšíření pro EB tresos Studio či samotných ovladačů demo aplikace spolu s návody, jak sestavit spustitelný *.elf soubor.

Děje se tak prostřednictvím příkazové řádky a před-připraveného dávkového souboru. V procesu sestavení dochází ke generování výstupní konfigurace pomocí EB tresos studia (opět z před-připraveného projektu) a následně ke kompilaci všech souborů.

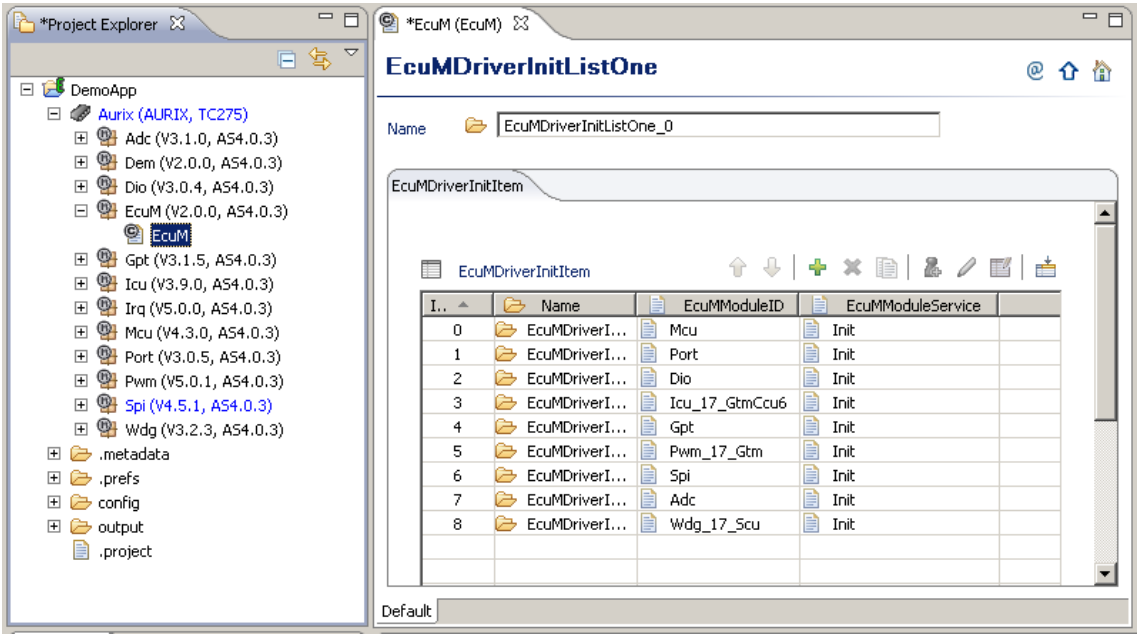
Souborová struktura dema je členěna na soubory samotných ovladačů, vytvořené konfigurace, demo aplikace a poté tzv. makefiles souborů, popisující proces kompilace. Tyto makefiles soubory svou strukturou zrcadlí předchozí zmiňované kategorie (ovladače, konfigurace, demo).

Jak uvádí dokumentace demo aplikace balíku MC-ISAR, popis sestavení každého modulu je rozdělen do souborů *_cfg.mak, *_check.mak, *_defs.mak a *_rules.mak podle standardu AUTOSAR.

Jako nevýhodné se jeví použití absolutního adresování napříč jednotlivými makefiles a zdrojovými soubory v rámci demo aplikace. Toto řešení značně znesnadňuje následnou možnou úpravu souborové struktury či eliminaci některých nepotřebných modulů.

Postup sestavení demo aplikace prostřednictvím dávkového souboru oproti tradičnímu projektovému řešení dále znesnadňuje orientaci napříč aplikací.

Použité moduly v demo aplikaci ukazuje obrázek 7.3.

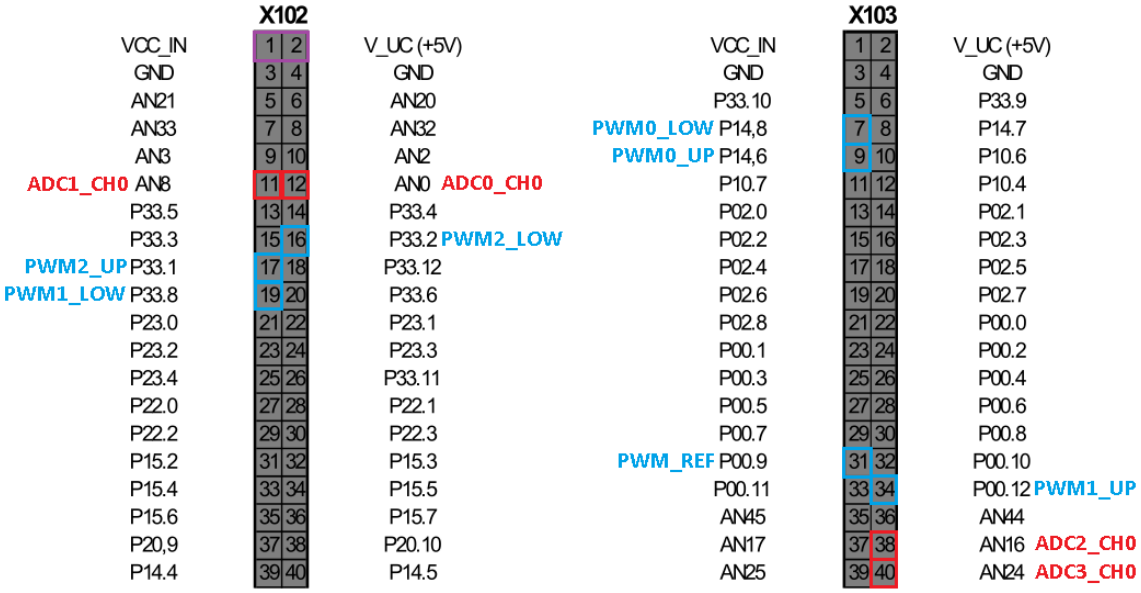


Obr. 7.3: Projektová podoba konfigurace MC-ISAR demo aplikace

Po následném nahrání již sestavené demo aplikace do mikrokontroléru a jeho spuštění dojde k nezamýšlenému jevu a to nezinicializování inteligentního napájecího mikroprocesoru TLF35584 pomocí SPI. Ten následně po uplynutí doby 'watchdogu' vyvolá reset mikrokontroléru. Po několika takovýchto resetech dojde k vypnutí mikroprocesoru TLF35584 a tím také k odstavení napájecího napětí pro zbytek aplikačního kitu, který je tím prakticky vyřazen z provozu. Tato situace dále znesnadňuje následné přehrání samotného mikrokontroléru.

Jako rychlé řešení této situace se proto jeví dodání napájecího napětí z externího zdroje. To může být realizováno na pinech aplikačního kitu pouhým přemostěním vstupního napájecího napětí použitého pro mikroprocesor TLF35584 na jeho výstupní větev. Důležité je však při tomto řešení dodržet potřebné napěťové úrovně, čili předchozí vstupní napětí v rozsahu 3 až 40V není přípustné, použije se 5V.

Umístění patřičných pinů na konektorech aplikačního kitu určených k přemostění napájecího napětí ukazuje obrázek 7.4



Obr. 7.4: Využití výstupních pinů aplikačního kitu

Z výše popsaných důvodů bylo od využití aplikačního dema, jako výchozího bodu upuštěno a byla zvolena metoda postupné integrace ovladačů MC-ISAR do projektu vývojového prostředí Free TriCore Entry Tool Chain.

Pozn.: Rutina inicializace napájecího mikroprocesoru je součástí demo příkladů z vývojového prostředí.

8 INTEGRACE OVLADAČŮ MC-ISAR

Jelikož integrace původního start-up kódu, linker scriptu a souboru užitého k mapování jednotlivých částí kódu do patřičných sekcí paměti (MemMap.h) byla neúspěšná již na úrovni sestavení projektu z důvodu přetékání kódu z jednotlivých sekcí, bylo od tohoto řešení upuštěno.

K těmto účelům byl využit start-up kód a linker skritp poskytovaný vývojovým prostředím. Obsah původního memmap.h souboru byl eliminován.

Mezi obecně potřebné soubory, nepatřící do žádné z uvedených skupin základního balíku ovladačů patří:

- Compiler_Cfg,h
- Compiler.h
- Dem_Types.h
- MemMap.h
- Os.c
- Os.h
- Platform_Types.h
- Std_Types.h

Pro upřesnění volby kompilátoru je potřeba definovat uživatelský symbol na úrovni projektu podle 6.1.1, a to v tomto konkrétním případě jako
_GNU_C_TRICORE == 1U.

8.1 MCAL

Softwarový modul MCAL ovladačů MC-ISAR obsahuje z hlavní části start-up kód procesoru a dále rutiny potřebné napříč celými ovladači (např. řízení přístupu do SFR, error handler apod.).

K eliminaci původního start-up kódu (Mcal.c) byl využit uživatelky vytvořený přepínač, viz. výpis 8.1. Původní MemMap.h soubor byl ponechán, jeho obsah však nikoli.

Výpis 8.1: Uživatelský vypínač určený k eliminaci start-up kódu softwarového modulu MCAL ovladačů MC-ISAR

```
#define REDUCE_MCAL_STARTUP_CODE  ( STD_ON )
```

Výčet souborů náležících softwarovému modulu MCAL je následující:

- Mcal_Compiler.h
- Mcal_DmaLib.h
- Mcal_Options.h
- Mcal_TcLib.c

- Mcal_TcLib.h
- Mcal_WdgLib.c
- Mcal_WdgLib.h
- McalOsConfig.h
- Mcal.c
- Mcal.h

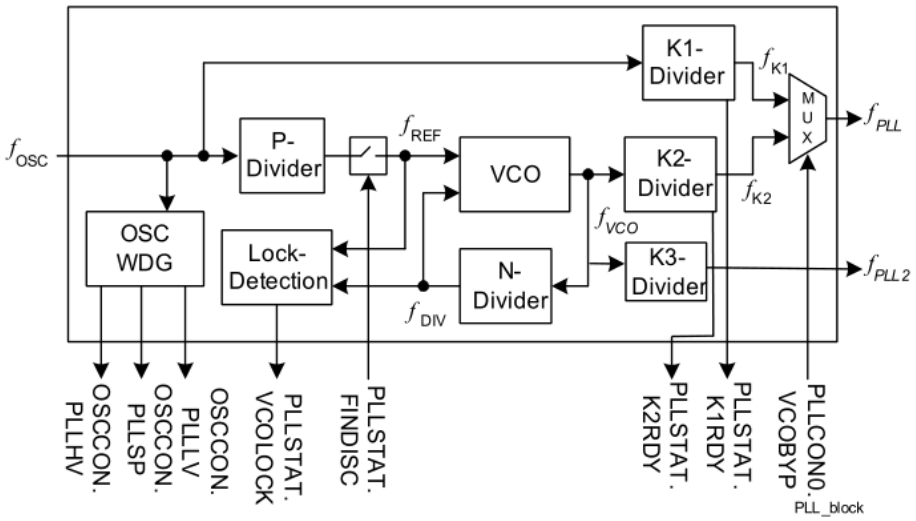
8.2 Mcu

Ovladač Mcu umožňuje inicializovat a zacházet s funkcionalitou procesoru jako jsou nastavení vnitřních hodin, volba módu (např. režim spánku), inicializace modulů GTM, ERU, CCU nebo DMA.

Pro účely požadované aplikace bude výhodné využít nejprve možnosti nastavení vnitřních hodin procesoru.

8.2.1 Konfigurace vnitřních hodin procesoru

Procesor TC277 je vybaven několika moduly s fázovým závěsem PLL pracujících v několika módech a umožňujících poskytnout vnitřním modulů a periferiím mikrokontroléru zdroj hodin s kmitočtem až 200MHz. Strukturu jednoho takového PLL modulu ukazuje obrázek 8.1.



Obr. 8.1: Vnitřní struktura modulu s fázovým závěsem

Výstupní hodinový signál PLL modulu je dále zpracováván a následně distribuován napříč celým procesorem k jednotlivým periferiím.

Všechny nastavitelné parametry (PLL tak i distribuce) musí svými hodnotami ležet v určitých přijatelných intervalech, dodržovat vzájemné poměry a dále být nastaveny v přesně určených krocích (nebo nenastaveny v případě omezení) definovaných specifikací mikrokontroléru.

Znalost postupu inicializace všech potřebných registrů uchovává samotný ovladač MCU. Pro správné nastavení všech zbývajících parametrů (celkem 51), je společně s distribucí ovladačů dodán také Excel soubor, jenž umožní pomocí několika málo vstupních informací jako jsou např. kmitočet použitého externího krystalu, požadovaná výstupní frekvence PLL modulu či typ procesoru vygenerovat až 9 různých nastavení. Je poté pouze na uživateli které nastavení podle požadavků na taktování dílčích periferií procesoru zvolí. Vygenerované hodnoty poté stačí pouze vhodně umístit do části hodinové konfigurace modulu MCU v tresos studiu.

8.2.2 Integrace

Pro účely integrace ovladače a konfigurace do projektu jsou použity následující soubory. Proceduru inicializace ukazuje výpis 8.2.

- Ovladače
  - Mcu_Local.h
  - Mcu_Platform.c
  - Mcu.c
  - Mcu.h
- Konfigurace
  - Mcu_Cfg.h
  - Mcu_PBCfg.c

Výpis 8.2: Rutina inicializace ovladače MCU spolu s vnitřním hodinovým signálem procesoru [7]

```
/*MCU initialization*/
Mcu_Init(&Mcu_ConfigRoot[0]);
/* System Clock Initialization - 160MHz */
RetVal = Mcu_InitClock(0);/*Clock Id*/
if(RetVal == E_OK)
{
 while ((Mcu_GetPllStatus()) == 0) {};
 Mcu_DistributePllClock();
}
```

8.3 PORT

Ovladač nazvaný PORT umožňuje inicializovat vstupně výstupní piny procesoru při jeho startu.

8.3.1 Konfigurace

Jednotlivým pinům je možné nastavit vlastnosti jako symbolické jméno, orientaci (vstup/výstup), použití pull-up rezistoru, počáteční úroveň po inicializaci, mód v jakém má být pin užit (obecně GPIO či pro různé alternativní funkce) či v jakých napěťových úrovních má pracovat.

Veškerou znalost povolených nastavení a jejich variací nyní pojímá EB tresos studio a uživatel je tak zbaven potřeby detailního seznámení se s použitým hardwarem.

8.3.2 Integrace

Pro účely integrace ovladače a konfigurace je použit výčet následujících souborů. Rutinu inicializace ukazuje výpis 8.3.

- Ovladače
  - Port_Ver.h
  - Port.c
  - Port.h
- Konfigurace
  - Port_Cfg.h
  - Port_PBCfg.c

Výpis 8.3: Rutina inicializace portů mikrokontroléru [8]

```
/* PORT initialization */
Port_Init(&Port_ConfigRoot[0]);
```

8.4 DIO

Ovladač DIO umožňuje číst či zapisovat na jednotlivé piny, porty či skupiny pinů mikrokontroléru.

8.4.1 Konfigurace

Konfigurace ovladače DIO umožňuje přiřadit jednotlivým pinům, portům či skupinám pinů symbolická jména. Pokud jsou pro danou aplikaci dostačující pevně

přiřazené piny či porty pinů, tedy vždy ty samé a stejné, není nutné v konfiguraci provádět úpravy, stačí vygenerovat pouze impicitní variantu a použít interní označení adresující svým jménem daný port a pin.

8.4.2 Integrace

Pro účely integrace jsou použity následující soubory. Výpis 8.4 ukazuje příklad inicializace ovladače DIO a zápis na pin.

- Ovladače
  - Dio_Ver.c
  - Dio_Ver.h
  - Dio.c
  - Dio.h
- Konfigurace
  - Dio_Cfg.h
  - Dio_PBCfg.c

Výpis 8.4: Příklad inicializace ovladače DIO a použití API pro zápis na pin [9]

```
/* DIO initialization */
Dio_Init(&Dio_ConfigRoot[0]);

/* write DIO channel */
Dio_WriteChannel( DIO_CHANNEL_14_5, STD_LOW );
```

9 GENEROVÁNÍ PWM

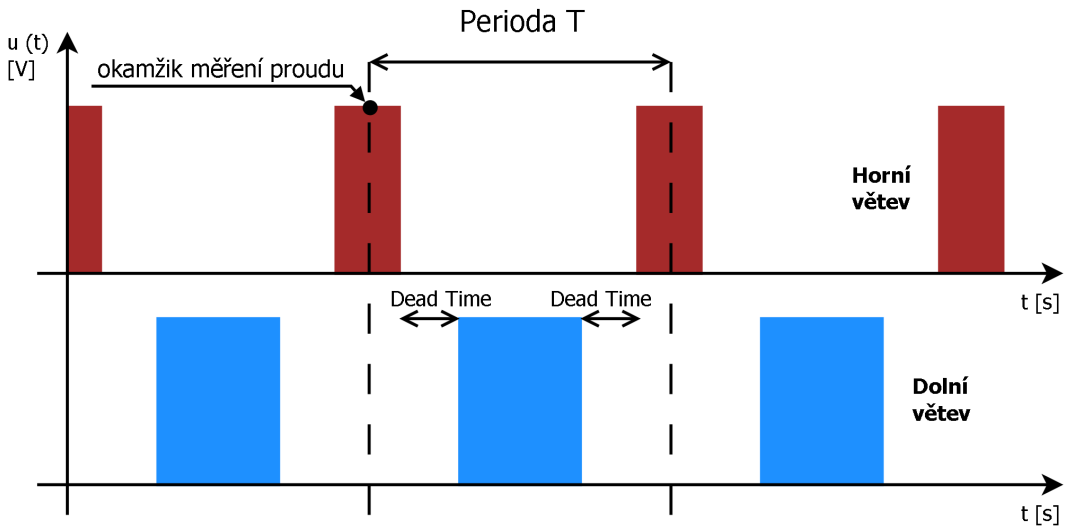
Existuje několik obecných požadavků na časové průběhy generovaných PWM signálu:

1. PWM signály jsou generovány v komplementárních párech - pro horní a dolní tranzistor budiče motoru.
2. Mezi oběma signály existuje v každé periodě neaktivní pásmo definované tzv. dobou *Dead Time*.
3. Signály jsou centrované s cílem dosažení synchronizovaného měření proudu s periodou PWM signálů.

Dead Time

Jelikož se dnešní aktivní budící prvky jako jsou tranzistory vyznačují konečnou dobou sepnutí (tranzistor nemůže přecházet z plně otevřeného stavu do plně zavřeného a naopak za nekonečně krátkou dobu) je potřeba vybavit řídící signály určitými neaktivními intervaly mezi oběma komplementárními kanály tzv. *Dead Time*. Tím jsou ošetřeny kritické stavy, kdy dochází k přepínání jednotlivých tranzistorů a je tak bráněno vzniku spojení nakrátko v napájecí větvi.

Polohu a umístění doby *Dead Time* v časových průbězích komplementárních kanálů ukazuje obrázek 9.1.



Obr. 9.1: Požadavky na časové průběhy generovaného PWM signálu

Synchronizované měření proudu

Průběhy proudů jednotlivými fázemi motoru jsou v každé periodě řídícího signálu PWM proměnné. Pro potřeby řídících algoritmů je proto nutné zajistit ekvidistantní vzorkování hodnot těchto proudů synchronizované z řídícími průběhy PWM signálů. Toho je docíleno použitím centrované PWM, kdy je na počátku každé periody spuštěno měření proudu pomocí ADC převodníku.

Tento okamžik je opět vyznačen v časových průbězích na obrázku 9.1.

9.1 Generic Timer Module (GTM) [10]

Součástí GTM modulu jsou 3 pod-moduly TOM, prostřednictví kterých je možné generovat výstupní PWM signál. Tyto TOM moduly jsou taktovány hodinovým signálem z jednotky CMU.

9.1.1 Clock Manamegent Unit (CMU) [10]

Vnitřní strukturu jednotky CMU ukazuje obrázek A.1, na kterém je také vyznačena logická trasa hodinového signálu použitého k taktování modulů TOM.

K tomuto účelu může být využit některý z pěti signálů fixních hodin označených *CMU_FXCLKx*. Výstupu fixních hodin jsou vždy předřazeny také děličky kmitočtu s právě pevným nenastavitelným dělícím poměrem. Zdrojů signálu sekce fixních hodin *FXU* může být několik, kde jeden konkrétní je vybrán volbou *FXCLK_SEL*. V případě této aplikace byl vybrán zdroj signálu z konfigurovatelné sekce *CFGU*, jenž obsahuje děličky kmitočtu s již volitelným programovatelným dělícím poměrem.

9.2 Timer Output Module (TOM) [10]

Základní strukturu jediného TOM modulu ukazuje obrázek A.2. Modul disponuje 16-ti výstupními kanály, formálně nazvanými TOM_CHx. Každé skupině 8-mi kanálů vždy přiléhá jedna řídící jednotka TGC.

9.2.1 Tom Global Channel Control (TGC) [10]

Vnitřní podoba jediné TGC jednotky, jež slouží k řízení 8-mi přilehlých kanálů je ukázána na obrázku A.3 .

Registry ENDIS a OUTEN

Zapnutí či vypnutí jednoho kanálu je možné prostřednictvím registru ENDIS_STAT. Zapnutí či vypnutí výstupu jednoho kanálu je prováděno skrze registr OUTEN_STAT. Oba tyto registry jsou vybaveny tzv. stínovými registry označenými registr_CTRL. Obsah stínových registrů je přenesen do řídících registrů po příchodu spouštěcí události. Touto spouštěcí událostí může být např. impuls od softwaru (zápis do registru HOST_TRIG).

Registr UPEN_CTRL

Označuje kanál, jenž má pro práci se svými vnitřními registry využít jim odpovídající stínové registry. Bližší popis této funkce bude proveden při rozboru vnitřní struktury jediného kanálu TOM_CHx.

9.2.2 Výstupní kanál (TOM_CHx) [10]

Vnitřní strukturu výstupního kanálu TOM_CHx ukazuje např. obrázek A.4. Hlavní bloky tvoří jednotky CCU0 (Capture Compare Unit), CCU1 a SOU (Signal Output Generation Unit).

CCU0

Tato jednotka je tvořena čítačem CN0, komparačním registrem CM0 a samotným komparátorem. S každým pulsem hodinového signálu (pokud je kanál zapnut prostřednictvím TGC jednotky - signál ENDIS) dochází k inkrementaci hodnoty registru čítače CN0. Pokud je hodnota v registru čítače vyšší, nežli hodnota uložená v komparačním registru CM0, komparátor nastaví svůj výstup.

Registr CM0 je vybaven také stínovým registrem SR0. Ke překopírování hodnot ze stínového registru SR0 do komparačního CM0 dojde vždy v případě resetu bloku CCU0 a tedy i čítače CN0 za podmínky aktivního signálu UPEN(x).

K resetu CN0 může dojít prostřednictví aktivního výstupu bloku CCU0 (tedy platí CN0 = CM0, obrázek A.4), nebo prostřednictvím synchronizačního signálu některého z předchozích kanálů *TRIG_x* (obrázek A.5). Tuto volbu řídí selektor RST_CCU0.

V případě první varianty určuje hodnota v komparačního registru CM0 (SR0) délku periody generovaného PWM signálu.

V případě druhé varianty určuje hodnota v komparačním registru CM0 (SR0) polohu nástupné hrany generovaného signálu. Perioda je v takovém případě definována okamžikem příchodu resetovacího impulsu.

CCU1

Jednotka CCU1 je tvořena komparačním registrem CM1 a samotným komparátorem. Ten porovnává hodnoty v registru čítače CN0 a komparačního registru CM1, který je taktéž vybaven svým stínovým protějškem SR1. Okamžik překopírování je určen stejně jako je tomu v případě jednotky CCU0 a to resetem čítače CN0 za podmínky aktivního signálu UPEN(x).

V případě první varianty resetu (CN0 = CM0) určuje hodnota uložená v komparačním registru CM1 střídu generovaného PWM signálu.

V případě druhé varianty resetu (od *TRIG_x*) určuje hodnota uložená v komparačním registru CM1 okamžik sestupné hrany generovaného signálu.

SOU

Součástí jednotky SOU je klopný obvod RS na jehož vstupy jsou přivedeny signály z jednotlivých komparátorů jednotek CCU0 a CCU1. Jednotka tak za předpokladu aktivního signálu OUTEN(x) generuje na svém výstupu požadovaný PWM signál.

Tato jednotka také umožňuje nastavit polaritu výstupního signálu.

9.3 Konfigurace modulu GTM

Konfigurace modulu GTM je součástí ovladače MCU. V tresos studiu lze proto nález v modulu MCU také patřičnou kartu *GtmConfiguration*.

Nastavení vnitřních hodin modulu

Jako první krok je potřeba provést nastavení vnitřních hodin modulu. To je provedeno prostřednictvím karty/modulu CMU. Konkrétní podobu zvolené konfigurace ukazuje tabulka B.1 společně s obrázkem A.1.

Při konfiguraci vnitřních hodin procesoru byla zvolena varianta, kdy je GTM jednotka taktována signálem s maximálním možným kmitočtem a to 100MHz. Spolu s nastaveními před-děliček obsažených v modulu CMU je poté perioda jednoho taktu hodin TOM modulu 50ns.

9.3.1 Konfigurace kanálů (TOM_CHx)

Pro realizaci tří-fázových průběhů PWM signálů (včetně komplementárních) je potřeba 6 výstupních kanálů TOM_CHx. K jejich synchronizaci je využit nultý kanál, jenž definuje velikost jedné periody všech PWM signálů a generuje tak resetovací signál *TRIG(x)* v patřičných časových intervalech.

Hardwarové zapojení nultého referenčního kanálu TOM_CH0 ukazuje obrázek A.4. Konfiguraci v tresos studiu popisuje tabulka B.2.

Zapojení zbylých aktivních výstupních kanálů popisuje obrázek A.5 spolu s tabulkou B.3.

Při konfiguraci výstupních kanálů je důležité správně zkonfigurovat také výstupní porty v tresos studiu pod modulem PORT. Zde je nutné nastavit zvolený pin jako výstupní a alternativní mód 1 tzn. pin je připojen k jednotce GTM. Nastavení ukazuje tabulka B.4.

Výstupní Piny

Mapování výstupních pinů aplikačního kitu na jednotlivé kanály TOM modulu ukazuje obrázek 7.4.

9.4 PWM ovladač

Samotný ovladač PWM základního balíku MC-ISAR nepodporuje generování komplementárních signálů a také zavádí další, pro účely naší aplikace nepotřebnou, rozsáhlou funkcionalitu. Z těchto důvodů byla zvolena varianta vytvoření vlastního PWM ovladače.

Vytvořené rozhraní ukazuje výpis 9.1. Je podporováno zadávání střídy pro jednotlivé kanály (fáze) buďto v procentech pomocí PWM3_SetDutyCycle nebo ve stovkách nanosekund skrze PWM3_SetPulseWidth. Kalkulaci hodnot pro komplementární kanály již provádí samotný ovladač.

Výpis 9.1: API PWM ovladače

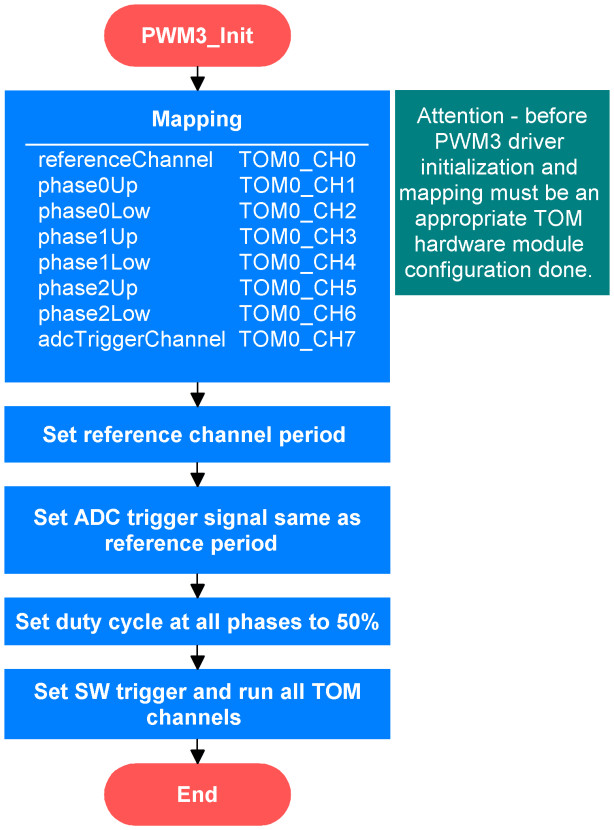
```
/* user type */
typedef struct{
uint32 phase[PWM3_NUMBER_OF_PHASES];
}PWM3_Timing_t;

/* PWM driver API */
void PWM3_Init ( void );
void PWM3_SetPulseWidth (
     PWM3_Timing_t * const timing_ptr );
void PWM3_SetDutyCycle (
     PWM3_Timing_t * const dutyCycle_ptr );
```

Konfiguraci ovladače ukazuje výpis 9.2. Je možné zvolit délku periody od 50µs do 100µs. Velikost parametru *Dead Time* ve stovkách nanosekund či velikost minimálního možného impulsu taktéž ve stovkách nanosekund.

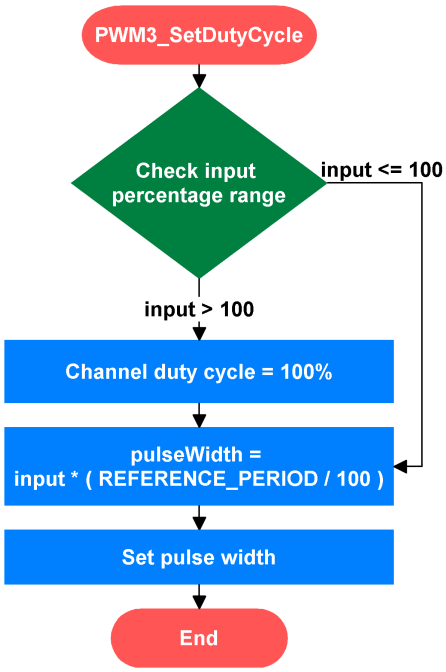
Výpis 9.2: Konfigurace PWM ovladače

```
/* PWM3 DRIVER - USER SETTINGS */
#define PWM3_REFERENCE_PERIOD_IN_MICROSECONDS     ( 100U )
#define PWM3_DEADTIME_IN_HUNDREDS_OF_NANOSECONDS  ( 30U )
#define PWM3_MINIMAL_POSSIBLE_IMPULSE             ( 30U )
```

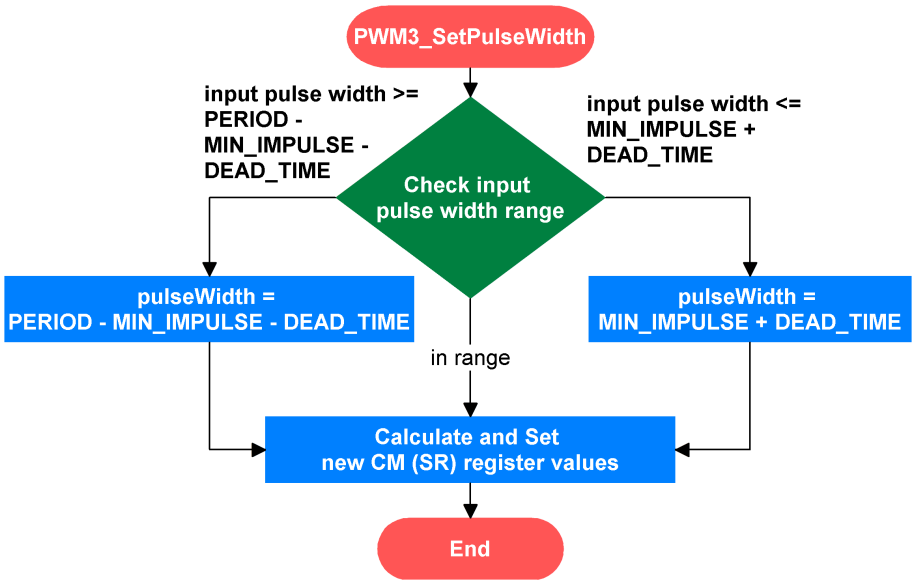
9.4.1 Blokové diagramy



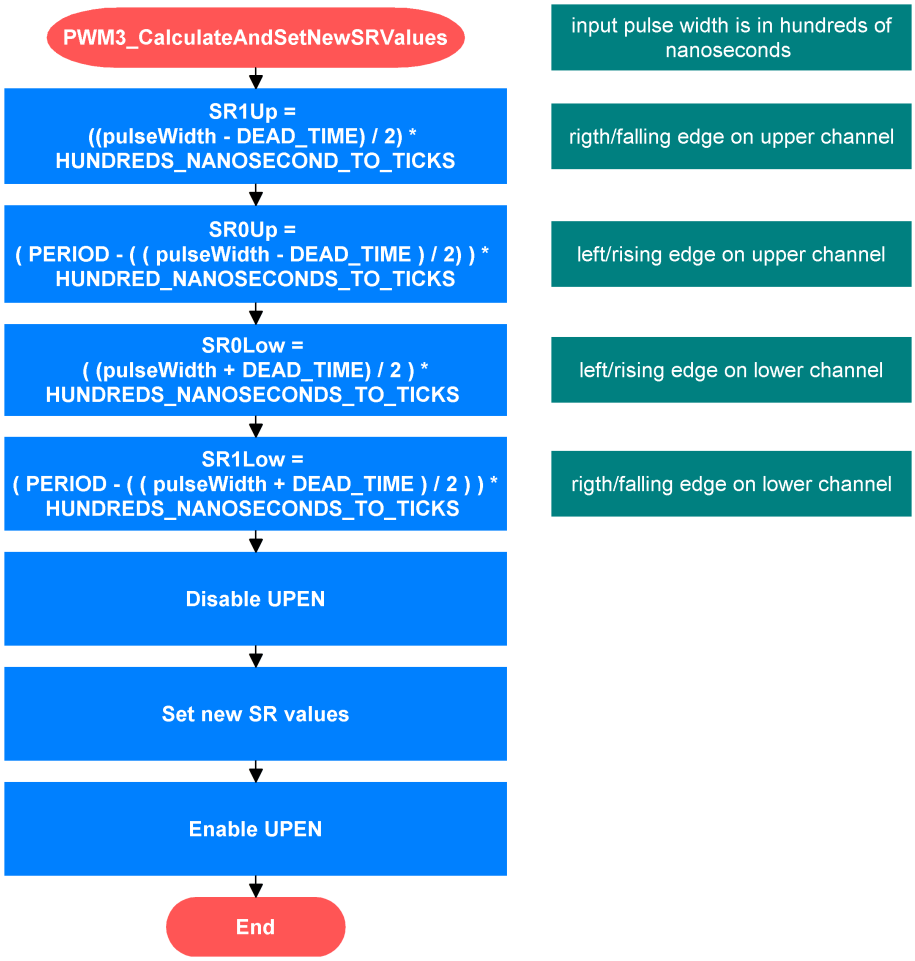
Obr. 9.2: Blokový diagram inicializace PWM ovladače



Obr. 9.3: Blokový diagram rutiny SetDutyCycle PWM ovladače



Obr. 9.4: Blokový diagram rutiny SetPulseWidth PWM ovladače



Obr. 9.5: Blokový diagram rutiny kalkulace hodnot vnitřních registrů jednoho kanálu

10 ANALOGOVĚ DIGITÁLNÍ PŘEVOD

Periferie obecného analogově digitálního převodníku (VADC - Versatile Analog-to-Digital Converter) dostupná na čipu mikrokontroléru obsahuje 8 samostatných analogově digitálních převodníků, kdy ke každému jednomu takovému je připojeno hned 8 analogových vstupních kanálů.

Z těchto výše zmíněných důvodů není možné pouze intuitivní nastavení hardwarové konfigurace převodníků v tresos studiu, nýbrž je potřeba nejprve nastudovat specifikaci této periferie.

10.1 Obecný analogově digitální převodník (VADC)

Samotné převodníky jsou typu s postupnou aproximací vybaveny obvody Sample & Hold. Jak již bylo zmíněno, periferie obsahuje 8 těchto převodníků, kdy každému náleží 8 vstupních kanálů. V jednom časovém okamžiku může být převáděn pouze jeden kanál na jednom převodníku. K volbě převáděného kanálu slouží interní multiplexer. Výběr, který z oněch 8-mi kanálů má být převáděn je proveden pomocí soutěžní strategie založené na prioritách a okamžicích příchodu požadavku na převod. Každý převodník je navíc vybaven nástroji k obsluze externího multiplexeru, čímž se počet vstupních kanálů dostupných jedinému převodníku ještě rozšíří. Obrázek 10.1 ukazuje vnitřní blokovou strukturu periferie VADC.

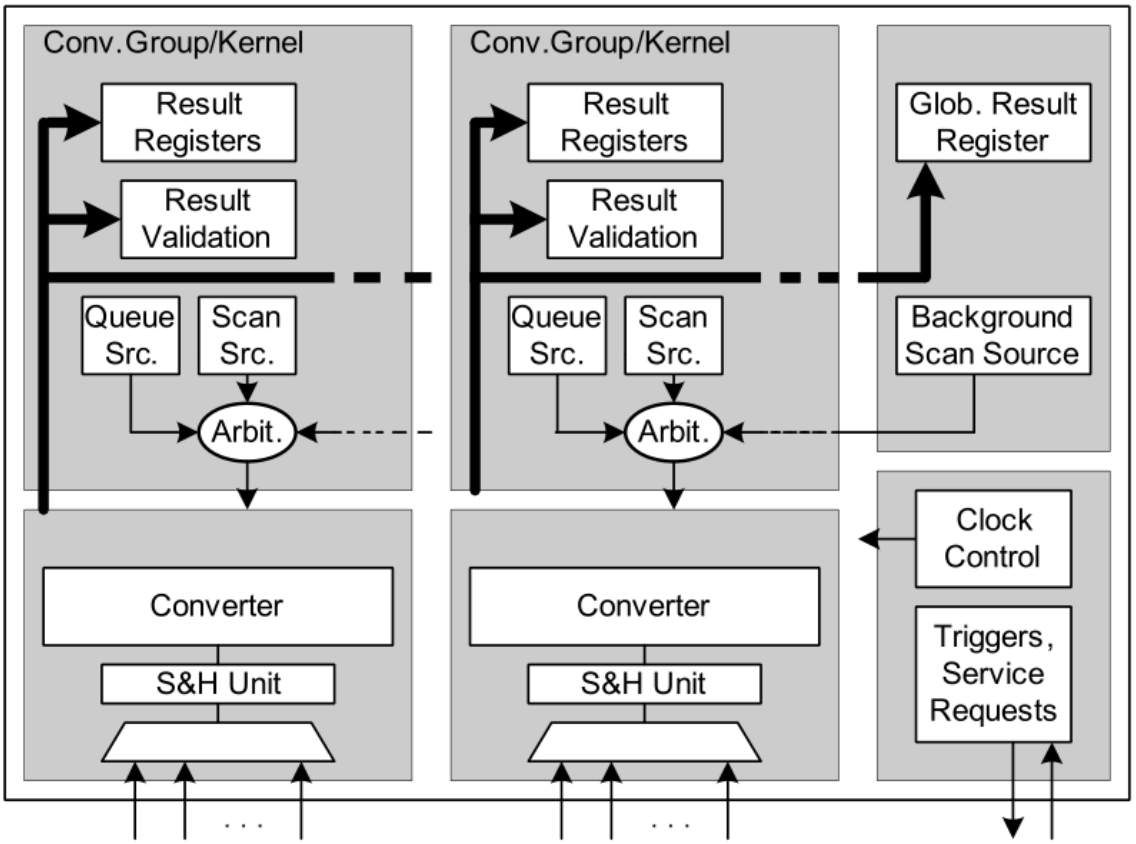
Proces volby vítězného požadavku spravuje modul Arbiter. K němu jsou připojeny tři možné zdroje těchto požadavků na převod. Jsou jimi *Queued source*, *scan source* a *background source*.

Určení, který kanál má být převáděn v jakem okamžiku je dáno procesem tzv. arbitrace, kterého se účastní požadavky na převod na jednotlivých kanálech a podle zvolených priorit vzejde z arbitrace vítězný požadavek k převodu. Podle nastavení způsobu ukončení právě probíhajícího převodu je pokračováno dále.

Tyto zdroje požadavků mohou být aktivovány buď z vrstvy softwaru nebo prostřednictvím některého z interních spouštěcích signálů (impuls od jiného vnitřního modulu mikrokontroléru např. časovače generujícího PWM nebo z vstupních pinů). Zdroje *Queued source* a *scan source* jsou vlastní (jedinečné) pro každý převodník. Zdroj *background source* je společný všem převodníkům.

Každý takový zdroj požadavků může pracovat buď v kontinuálním režimu, kdy jsou převody po prvním požadavku opakovány periodicky nebo v módu, kdy je převod na jeden požadavek vykonán pouze jednou.

První zdroj pro generování požadavků na převod, tzv. source 0 nebo podle dokumentace *queued source* umožňuje definovat sekvenci libovolně řazených převodů na



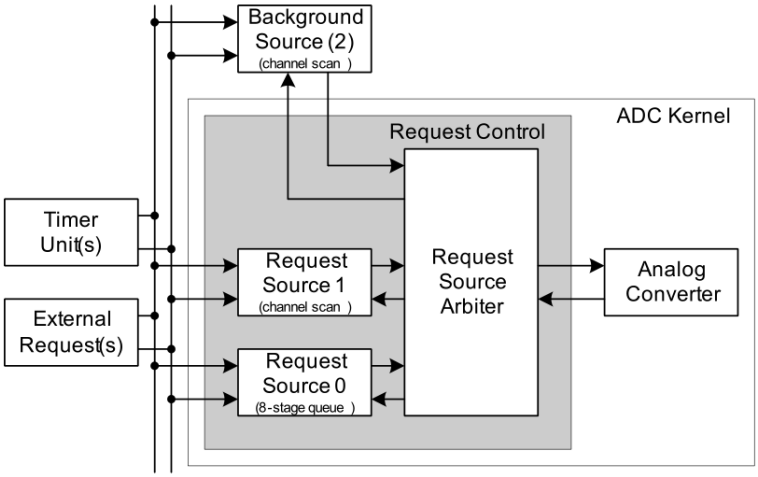
Obr. 10.1: Vnitřní bloková struktura periferie VADC [10]

kanálech, kdy i vícenásobné převody na jednom kanále jsou možné. Druhý ze specifických zdrojů danému pevodníku je source1 - *scan source*, který umožňuje spustit sekvenci převodů na kánálech řazenych od kanálu s nejvyšším identifikačním číslem až po kanál s tím nejnižším. Poslední *background source*, společný pro všechny převodníky umožňuje zadat požadavek na jeden převod na jednom kanále.

Queued Source [10]

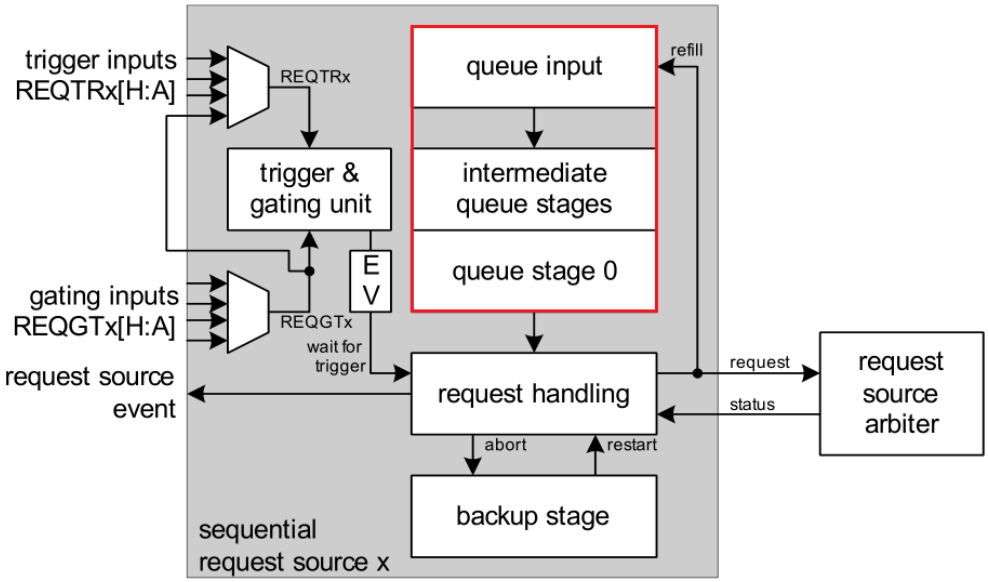
Sekvence nebo-li pořadí, v jakém mají být převody na daných kanálech konkrétního převodníku vykonány je uložena ve frontě, realizované pamětí FIFO. Každá položka uchovává číslo kanálu k převodu. Sekvence je tedy definováná vkládáním požadavků do vtupního registru fronty .

Dalšími položkami každého prvku fronty (kanál k převodu) je např. informace, zdali má převod čekat na externí spouštěcí signál (trigger) nebo se má provést okamžitě nebo zda-li ma dojít k přerušení (interruptu) po dokončení převodu nebo



Obr. 10.2: Zdroje požadavků na převod a arbiter [10]

zdali se má požadavek umístit znovu do fronty, tzv. *refil*. Vnitřní strukturu zdroje požadavků *Queued Source* ukazuje obrázek 10.3.



Obr. 10.3: Blokové schéma zdroje *Queued Source* [10]

Scan Source [10]

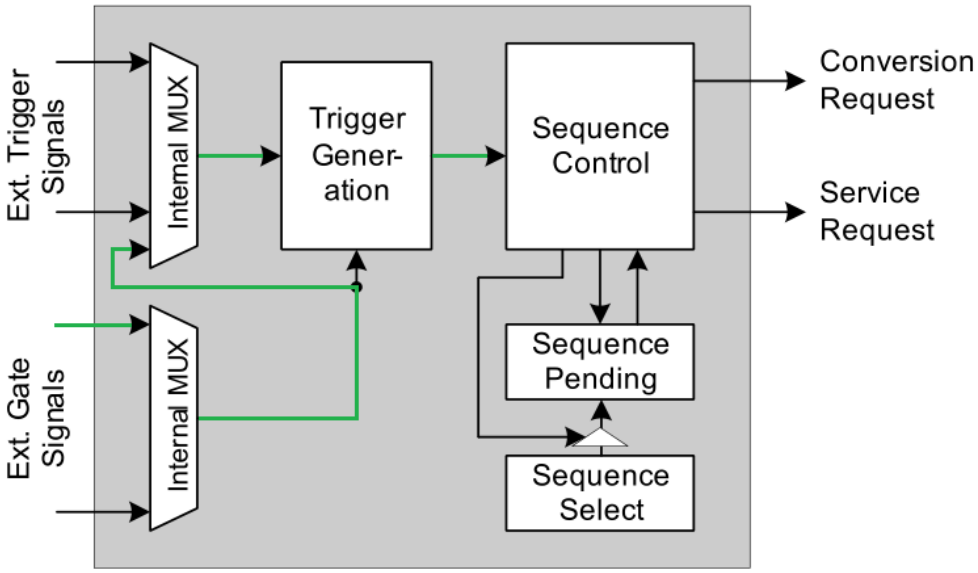
Mezi zdroje požadavků na převod tohoto typu patří source1 (*Scan Source*) a source2 (*Background Source*). Oba tyto zdroje mají stejný interface (registry pro jejich řízení). Source2 může požádat o převod na všech kanálech všech převodníků, kdežto

source1 je specifický pro každý převodník.

Každý kanál může být zahrnut nebo vynechán z dané sekvence převodů. Např. může být vyžadován převod pouze na kanálech 1,4 a 7. Pořadí převodů je sestupné podle čísla kanálu, tedy 7.,4. a 1. kanál budou převedeny v tomto pořadí.

Na událost zvanou *load Event* je započata sekvence převodů definovaná v patřičném registru *Scan Source* zdroje. Tato *Load Event* událost může být vyvolána buďto prostřednictvím softwaru nebo spouštěcím signálem (trigger) od jiného modulu mikrokontroléru.

Vnitřní blokovou strukturu zdroje *Scan Source* ukazuje obrázek 10.4.



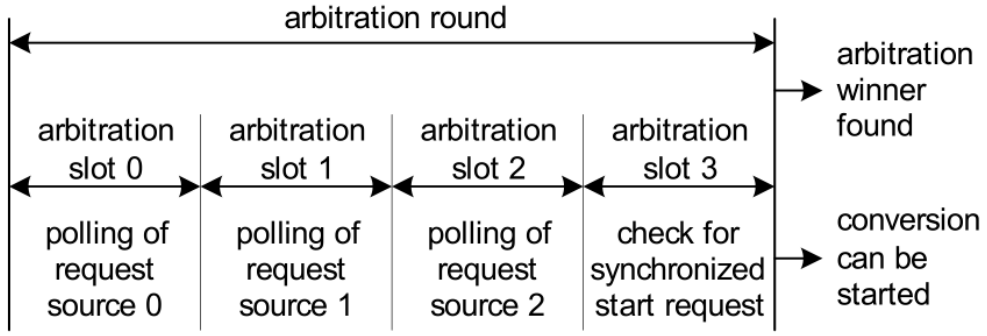
Obr. 10.4: Vnitřní bloková struktura *Scan Source* zdroje požadavků k převodům [10]

Arbiter [10]

Arbiter má několik slotů, kdy každému slotu odpovídá jeden ze zdrojů požadavků na převod. Priorita každého z těchto zdrojů je nastavitelná v registrech arbiteru daného převodníku.

Nad všemi sloty je prováděno vyhodnocení vítěze v tzv. arbitračním kole (*Arbitration Round*), jehož délka je nastavitelná a určuje tím tak časovou jemnost pro detekování příchozích požadavků k převodům. Jeden průběh arbitračního kola ukazuje obrázek 10.5.

Za čtvrtý zdroj požadavků k převodům jsou považovány požadavky k synchronním převodům na několika převodnících najednou. Tento čtvrtý zdroj má vždy nejvyšší prioritu a v arbitračním kole je vyhodnocen naposled.



Obr. 10.5: Arbitrační kolo *Arbitration Round* [10]

Zpracování vítězného požadavku k převodu [10]

Vítězné požadavky jsou zpracovány převodníkem podle aktuálního stavu, v jakém se převodník nachází. Ten může být v klidovém stavu (*idle*), v takovém případě požadavku vyhoví, nebo může provádět konverzi s vyšší nebo nižší prioritou než právě žádaná konverze. V takovém případě záleží na uživatelském nastavení.

Aktuálně zpracovávaná konverze může být buďto okamžitě ukončena, poté provedena konverze s vyšší prioritou a následně dokončena předešlá přerušená konverze. Druhou variantou je dokončení právě probíhající konverze kdy vítězný požadavek arbitračního kola bude zpracován následně.

Výsledky převodu [10]

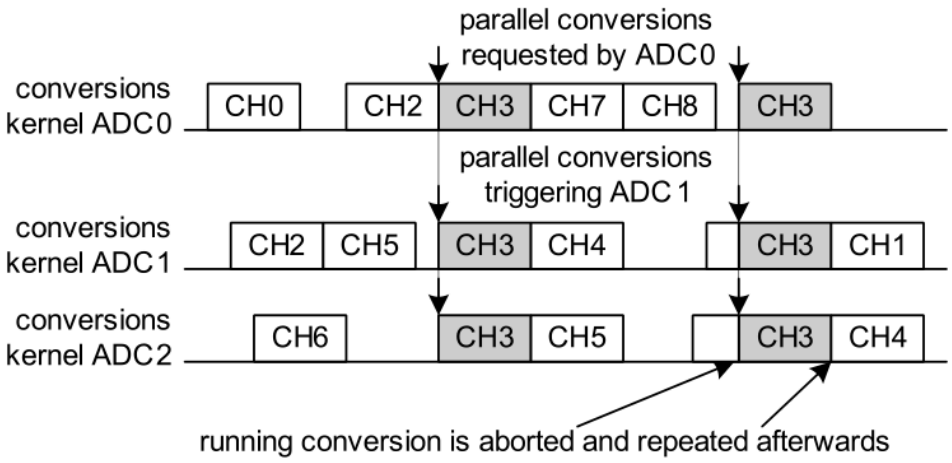
Všem kanálům je možné nastavit cílový registr do kterého má být výsledek převodu uložen. Každý převodník disponuje 16-ti takovými registry. Dále je k dispozici také globální registr společný všem převodníkům jak ukazuje obrázek 10.1. Bitovou šířku je možné zvolit pro každý kanál v rozsahu hodnotnot 8,10 nebo 12 bitů. Výsledek převodu lze uložit se zarovnáním vlevo či vpravo.

Kterýkoli kanál lze přiřadit do některé z tzv. *Input Class*, které definují použitou bitovou šířku výsledku a počet hodinových cyklů periferie VADC určených ke vzorkování. Každému převodníku náleží dvě vlastní takové třídy (*Group-Specific Class 0, 1*). Další dvě tyto třídy (*Global Class 0, 1*) jsou k dispozici jako globální, dostupné všem převodníkům společně.

Paralelní převody [10]

Několik převodníku může být synchronizováno k současnému měření na svých analogových vstupních kanálech. Synchronní mechanizmus pro paralelní konverze zajišťuje, že vzorkovací fáze příslušných kanálů začne současně.

V zapojení převodníků pro paralelní převody je rozlišen jeden hlavní převodník tzv. *Master*, na který jako jediný přichází požadavek k převodu a poté zbylé zpřažené převodníky tzv. *Slaves*. Kanál, který je požadován k převodu na *Master* převodníku je převáděn také na zbylých převodnících, jak je možné vidět na obrázku 12.1.



Obr. 10.6: paralelní převod [10]

Paralelní převody jsou možné pouze na dvou skupinách převodníků. Jsou jimi skupina A do které náleží převodníky s indexy 0,1,2 a 3 a skupina B kam patří zbylé převodníky 4,5,6 a 7.

Spouštěcí signál od PWM [10]

K vyvolání požadavku k převodu je možné využít vnitřní signály z jiných modulů mikrokontroléru a synchronizovat tak např. spouštění převodu s generovanými PWM signály.

Zdroje požadavků k tomuto účelu obsahují vstupy pro tzv. *Gate* a *Trigger* signály jak je možné vidět např. na obrázku 10.3 nebo 10.4.

10.2 Konfigurace VADC

Pro účely aplikace algoritmů řízení elektrických motorů je vhodná konfigurace, kdy dochází k paralelním převodům na více převodnících (např. měření proudů fázemi motoru), synchronizovaných s začátkem periody PWM signálu jak ukazuje obrázek 9.1.

Konfigurace ovladače Adc

Pro účely využití výše zmíněné funkcionality jako jsou paralelní převody či synchronizace se signály PWM je nutné provést konfiguraci samotného ovladače Adc a zpřístupnit tak některé potřebné funkce a API. Nezbytné položky, které je potřeba nastavit ukazuje tabulka C.1, zbylé nastavení lze ponechat původní.

Konfigurace periferie VADC

Do této kategorie spadá nastavení vnitřních hodin převodníků. Mezi nezbytné položky k nastavení v tresos studiu patří vytvoření reference k celkovému nastavení vnitřních hodin poskytované ovladačem Mcu. Jako další může být upravena např. hodnota vnitřní děličky kmitočtu apod. Možné nastavení ukazuje tabulka C.2.

V této části je také možné vhodně nastavit některou ze dvou *Global Input Class*, jež lze následně využít k definování doby vzorkování a rozlišení výsledku převodu pro všechny použité kanály.

Master převodník, kanál

Jednotlivé převodníky lze v tresos studiu nalézt pod označením AdcHwUnit. Základní nezbytné nastavení převodníku pracujícího jako master ukazuje tabulka C.3.

Konfiguraci samotného kanálu pojímá karta AdcChannel. Zde minimální konfiguraci ukazuje tabulka C.4. Důležitou položku zde představuje číslo kanálu jenž je lineárně rostoucí napříč všemi převodníky. Tzn. že první kanál druhého převodníku nese identifikační číslo 8. První kanál třetího převodníku zase 16. V případě zvoleného Master převodníku 0 a jeho nultého kanálu tomu odpovídá číslo 0.

Nastavení spouštěcího signálu od generované PWM je možné pod kartou AdcGroup. V těchto skupinách je možné seskupovat několik kanálů patřících jedinému převodníku.

Jak uvádí specifikace mikrokontroléru, ke spouštění převodníku od GTM modulu nelze využít přímo signálové větve nazvané *Trigger* nýbrž je potřeba nepřímo využít signálové cesty *Gate* jak je možné vidět na obrázku 10.4. Možné minimální nastavení ukazuje tabulka C.5.

Tímto je vytvořeno signálové spojení na straně analogově digitálního převodníku 0. Dále je potřeba vytvořit opačný konec tohoto spojení také na straně GTM modulu. To je možné provést v sekci GtmConnections modulu GTM pod kartou GtmTriggerForAdc. Vzhledem k možnostem hardwaru (nabídka použitelných kanálu je omezená) a současnému mapování jednotlivých kanálů k účelu generování PWM signálu se nabízí pouze 7. kanál TOM modulu 0. Je proto také potřeba rozšířit referenční signál z kanálu TOM0_CH0 na 7. kanál. To zajišťuje ovladač PWM.

Nastavení synchronizačního spojení na straně GTM modulu kazuje tabulka C.6.

Slave převodníky, kanály

Při konfiguraci zpřažených převodníku je potřeba vhodně zvolit odpovídající převodník, uvést, že se jedná o slave převodník a vhodně vybrat při konfiguraci číslo kanálu. Zbylá konfigurace je již obdobná jako v případě master převodníku vyjma vytvoření synchronizačního spojení s modulem GTM. Možnou podobu konfigurace ukazují tabulky C.7, C.8 a C.9.

Vstupní piny převodníků

Většina vstupních pinů převodníku slouží pouze k tomuto účelu a není proto potřeba je konfigurovat. Některé však mohou plnit úlohu také běžných vstupně výstupních pinů. V takovém případě je potřeba provést konfiguraci v modulu PORT podle tabulky C.10. V případě námi zvolených kanálů AD převodníků se toto týká kanálu 0 převodníku 3, jehož vstupní pin může také sloužit jako port 40 pin 0.

Mapování vstupních pinů aplikačního kitu na jednotlivé kanály AD převodníků ukazuje obrázek 7.4.

10.3 Ovladač

Jelikož podstatnou část správy analogově digitálních převodníků tvoří jejich hardwarová konfigurace a navíc jsou spouštěny signály od GTM modulu, postačuje provést pouze inicializaci. Tu uvádí výpis 10.1.

Výpis 10.1: Inicializace AD převodníku [11]

```
/* VADC initialization */
Adc_Init( &Adc_ConfigRoot[0] );

/* enable HW trigger from GTM-TOM module to VADC */
Adc_EnableHardwareTrigger( adcGroup0 );
```

Integrace

Pro účely integrace ovladače analogově digitálních převodníku jsou potřeba následující soubory:

- Ovladače
  - Adc_Calibration.c
  - Adc_ConvHandle.c
  - Adc_ConvHandle.h
  - Adc_Dbg.h

  - Adc_HwHandle.c
  - Adc_HwHandle.h
  - Adc_Utility.h
  - Adc_Ver.c
  - Adc.c
  - Adc.h
  - SchM_Adc.h
  - SchM.c
- Konfigurace
  - Adc_Cfg.h
  - Adc_PBCfg.c

11 OBSLUHA PŘERUŠENÍ

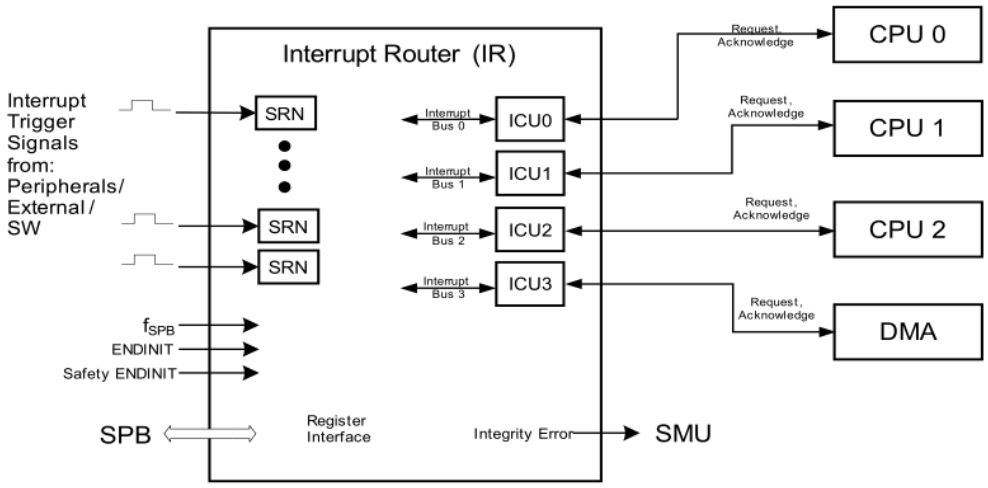
Přerušení (*interrupt*) představuje vhodný prostředek zpracování výsledků analogově digitálních převodů. Případně je možné umístit celý řídící algoritmus do rutiny přerušení. Sekvence zpracování jednotlivých úkonů by mohla být následující.

1. Přerušení programu vyvolané dokončením analogově digitálních převodů.
2. Zpracování výsledků převodů řídícím algoritmem.
3. Generace nových průběhů PWM vypočítaných řídícím algoritmem.

Přerušení spravuje modul zvaný IR (Interrupt Router).

11.1 Interrupt Router (IR) [10]

Blokovou strukturu *Interrupt Routeru* ukazuje obrázek 11.1. Ten má na své vstupy přivedeny spouštěcí signály z jiných periferií mikrokontroléru, které dále pokračují na tzv. SRN uzly (Service Request Node). Každému zdroji přerušení (vnitřní modul mikrokontroléru) náleží určitá oblast uzlů SRN, kde jejich adresa je pevně určená specifikací.



Obr. 11.1: Bloková struktura Interrupt Routeru

Další vnitřním blokem IR jsou čtyři tzv. ICU jednotky (Interrupt Control Units). K těmto jednotkám je možné připojit libovolný výstup některého z uzlů SRN, čímž je určen zdroj požadavku k přerušení. Funkcionalitou poskytovanou jednotkami ICU je proces arbitrace, během kterého je určen vítězný požadavek v případě jejich souběhu. Výstup každé ICU jednotky je již pevně připojen k jednomu z poskytovatelů přerušení. Těmi mohou být CPU0, CPU1 a CPU2. Posledním možným zdrojem přerušení je DMA kontrolér.

Service Request Node (SRN)

Každý uzel SRN obsahuje kontrolní registr SRC (Service Request Control Register) který slouží k jeho nastavení. Je možné zvolit poskytovatele přerušení, tedy ICU jednotku ke které bude daný uzel připojen, jeho prioritu a v neposlední řadě i samotné zapnutí tohoto uzlu.

11.2 Interrupt Service Routine (ISR) [10]

Tabulka přerušení, jenž uchovává příslušné adresy rutin přerušení a je uložená v každém CPU zvlášť je řazena podle priorit uzlů SRN. Tzn., že se zvolenou prioritou je spřažena konkrétní rutina. Oproti klasickému přístupu, kdy je řazení provedeno podle zdrojů přerušení umožňuje toto řešení aby měl jeden modul několik přerušení pro různé účely.

11.3 Ovladač MC-ISAR

Ovladač přerušení základního balíku MC-ISAR je typu *Variant PC*. Tzn. že všechny jeho konfigurační parametry jsou implementovány pomocí direktivy preprocesoru.

Pokud je v tresos studiu nějakému uzlu SRN náležícímu analogově digitálnímu převodníku nastavena priorita vyšší než nula, stane se patřičná servisní rutina aktivní součástí kódu. Každé této rutině předchází část kódu v jazyce symbolických adres, jenž umístí odpovídající adresu této rutiny to tabulky přerušení.

Samotná tabulka přerušení je inicializována v start-up kódu, jenž byl eliminován. Z těchto důvodů je použití ovladače MC-ISAR ke správě přerušení nemožné.

11.4 HighTec řešení obsluhy přerušení

K nastavení přerušení od události uložení nového výsledku analogově digitálního převodu byly využity rutiny obsažené v základních příkladech dodaných spolu s vývojovým prostředím.

Těmito rutinami jsou:

- InterruptInit - inicializuje tabulku perušení
- InterruptInstall - zavede rutinu do tabulky přerušení

Instalace přerušení tímto způsobem se jeví jako znatelně implementačně jednodušší.

12 APLIKAČNÍ ROZHRANÍ

Všechny nezbytné moduly a periferie pro generování komplementárních PWM signálů, paralelní měření analogových napětí či obsluhu přerušení (GTM, VADC, IR) k testování řídících algoritmů elektrických motorů již byly nakonfigurovány a inicializovány. Zbývá tak představit podobu konečného aplikačního rozhraní.

Prototyp uživatelské rutiny přerušení vyvolanou po dokončení analogově digitálních převodů ukazuje výpis 12.1. Konečná implementace rutiny je plně na uživateli. Řídící algoritmus může být umístěn také zde.

Výpis 12.1: Uživatelská rutina přerušení po dokončení analogově digitálních převodů

```
void AdcIrq_UserRoutine( void ) __attribute__((weak));
```

Ke čtení získaných výsledků analogových převodů slouží funkce, jejíž prototyp ukazuje výpis 12.2.

Výpis 12.2: Rutina čtení výsledku paralelních převodů

```
StatusType AdcIrq_GetResults(
   uint16 * const buffer_ptr, const uint32 bufferLength);
```

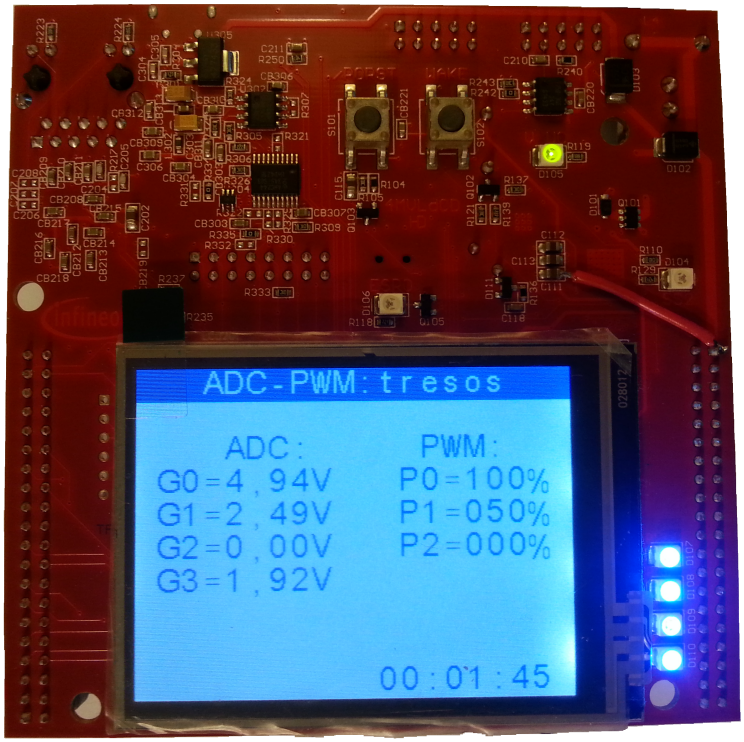
K nastavení a generování PWM signálů slouží API popsané podkapitole 9.4.

Příklad použití aplikačního rozhraní

Pro účely prezentace aplikačního rozhraní byla vytvořena jednoduchá aplikace, jenž moduluje výstupní PWM signály na základě vstupního napětí měřeného na prvních třech použitých analogových převodnících. Ke čtení výsledků převodů a novému nastavení PWM signálů dochází právě v uživatelské rutině přerušení.

Aktuální hodnoty měřených napětí a střídy PWM signálu jsou průběžně zobrazovány na displeji, který je obsluhován mimo přerušení.

Průběh této vzorové aplikace na aplikačním kitu ukazuje obrázek 12.1.



Obr. 12.1: Vzorová aplikace využívající vytvořené aplikační rozhraní

13 ZÁVĚR

Bylo vytvořeno aplikační rozhraní umožňující generování tří-fázových komplementárních centrovaných PWM signálů, paralelní měření na čtyřech kanálech analogových převodníků synchronizovaných s průběhy PWM a obsluhou přerušení vyvolanou dokončením všech převodů pro procesor firmy Infineon AURIX TriCore TC277. Testování proběhlo na vývojové desce AURIX Application Kit TC277.

K hardwarové konfiguraci většiny modulů (vyjma *Interrupt Routeru* pro obsluhu přerušení) bylo využito EB tresos studio společně s ovladači MC-ISAR podporující otevřenou softwarovou architekturu AUTOSAR, jejíž filosofie byla v práci představena.

Tato práce tak může sloužit jako předloha k rychlé a snadné konfiguraci modulu GTM pro generování PWM signálů nebo modulu VADC k paralelním analogově digitálním převodům synchronizovaných s průběhy PWM či k testování jednoduchých algoritmů na cílovém hardware.

Plná integrace ovladačů MC-ISAR podle standardu AUTOSAR se nezdařila. Důvody byli značná rozsáhlost projektové struktury v podobě *makefiles* za současné omezené dostupnosti popisné dokumentace společně s konfliktními informacemi obsažených v *Linker Skriptu* oproti zamýšlenému start-up kódu (potřebné velikosti jednotlivých sekcí nebyly dostačující). Z těchto důvodů bylo zvoleno řešení částečné (postupné) integrace ovladačů jednotlivých modulů a periferií mikrokontroléru s podporou vývojového prostředí Free TriCore Entry Tool Chain. Možný postup v této oblasti podněcuje dokument [12]. Podpora je však poskytována pouze na komerční bázi.

Praktická zkušenost s nástroji jako EB tresos studio umožňující generaci konfiguračního kódu jak samotného hardware tak použitých ovladačů podle standardu AUTOSAR ukazuje na značný nárůst potřebných znalostí k dosažení těchto cílů. Jedna z hlavních myšlenek standardu AUTOSAR je odstínění specifických vlastností použitého hardwaru. Při realizaci této práce bylo však postupováno od studia hardware přes koncepční testy až k vytvoření potřebné konfigurace splňující standard. První kontakt s touto technologií tedy nepřinesl slibované výhody. Je zřejmé, že opakované užití těchto nástrojů zkrátí dobu potřebnou k vývoji vhodného aplikačního prostředí. Pokud však nebudou jednotlivé projekty realizovány napříč různou škálou procesorů a jejich výrobců, nemusí tomu tak vždy býti.

Jako jeden z hlavních kroků před samotným použitím těchto technologií tedy považují vytvoření odhadu množství jejich užití, kde důležitým faktorem je různorodost hardware, v porovnání s výhodami které mohou přinést.

LITERATURA

[1] C Code Generation from Simulink. *Mathworks* [online]. ©1994-2017 [cit. 2017-04-18]. Dostupné z: https://www.mathworks.com/help/dsp/ug/generate-code-from-simulink.html

[2] *Autosar* [online]. [cit. 2017-04-18]. Dostupné z: https://www.autosar.org/

[3] Infineon. *AURIX Application Kit TC277 TFT* [online]. Infineon, ©1999-2017 [cit. 2017-05-09]. Dostupné z: https://www.infineon.com/cms/en/product/evaluation-boards/KIT_AURIX_TC277_TFT/productType.html?productType=5546d4624b0b249c014ba6c1283c32c3

[4] *Application Kit TC2X7: User's Manual* [online]. V 1.0. Infineon, 2015-04 [cit. 2017-05-09]. Dostupné z: https://www.infineon.com/cms/en/product/evaluation-boards/KIT_AURIX_TC277_TFT/productType.html?productType=5546d4624b0b249c014ba6c1283c32c3

[5] *EB tresos Studio user's guide*. v13.0. Elektrobit, 2012 [cit. 2017-05-09].

[6] Infineon. *AUTOSAR Software* [online]. ©2011-2017 [cit. 2017-05-08]. Dostupné z: https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/embedded-software-solutions/infineon-autosar-software/channel.html?channel=db3a30431b3e89eb011b4aac01f07b7d

[7] INFINEON. *AURIX Microcontroller Infineon Software Architecture: User's Manual, MCU driver*. V4.1. Infineon, 2015-05 [cit. 2017-05-08].

[8] INFINEON. *AURIX Microcontroller Infineon Software Architecture: User's Manual, PORT Driver*. V4.3. Infineon, 2015-04 [cit. 2017-05-08].

[9] INFINEON. *AURIX Microcontroller Infineon Software Architecture: User's Manual, DIO Driver*. V8.1. Infineon, 2015 [cit. 2017-05-08].

[10] INFINEON. *AURIX TC27x C-Step: User's Manual*. V2.2. Infineon, 2014-12 [cit. 2017-05-08].

[11] INFINEON. *AURIX Microcontroller Infineon Software Architecture: User's Manual, ADC Driver*. V5.6. Infineon, 2015 [cit. 2017-05-08].

[12] HIGHTEC. *Preferred Design House: Basic & Premium Consultancy Model* [online]. Infineon, 2016 [cit. 2017-05-08]. Dostupné z: https://www.infineon.com/dgdl/Infineon-High_TecP_referred_Design_House+-PP-v01_00-EN.pdf?fileId=5546d46258fc0bc101598335d6ca367f

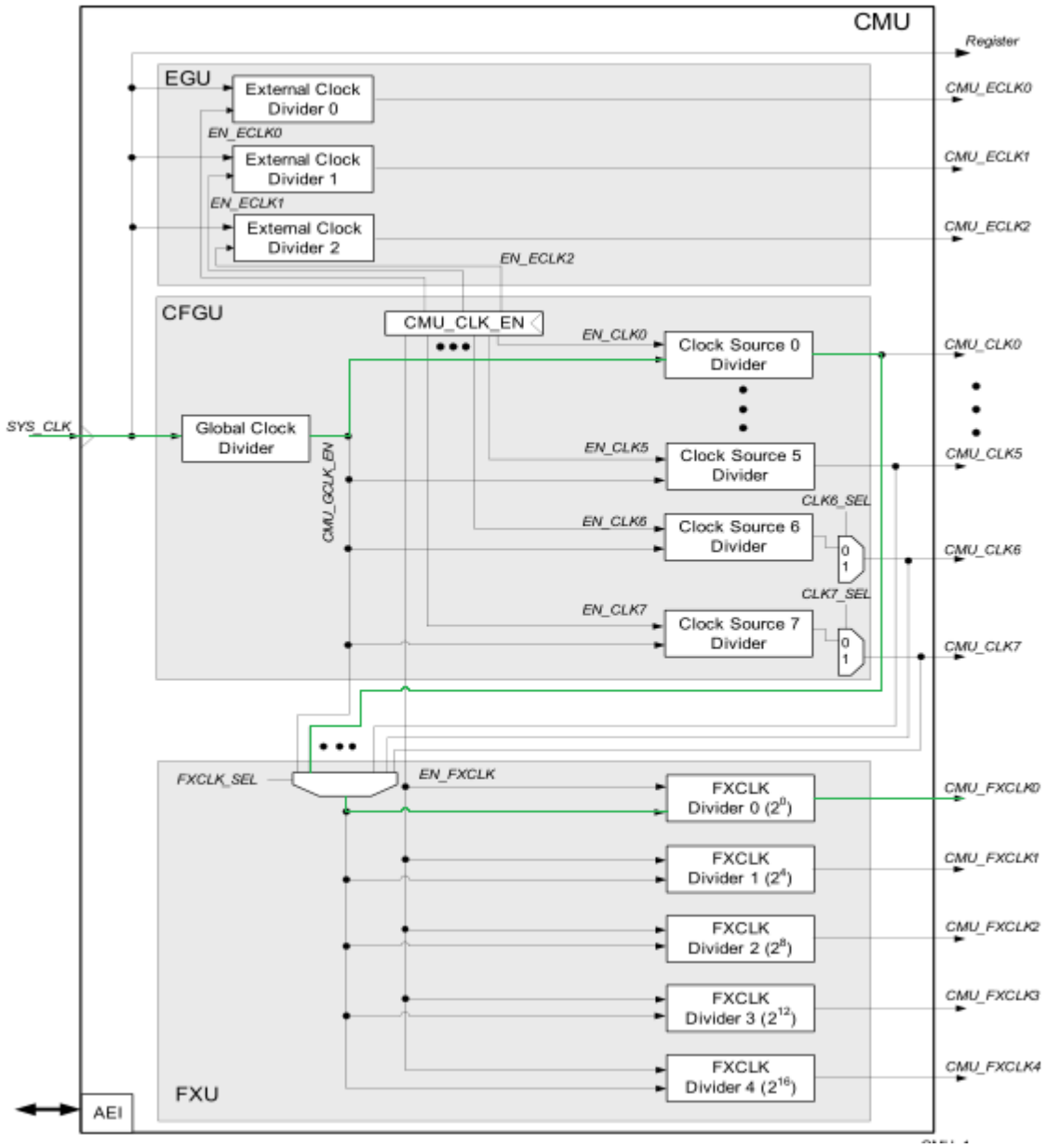
SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ADC	Analogově digitální převodník
API	Application Programming Interface
ARU	Advanced Rounting Unit
AUTOSAR	AUTomotive Open System ARchitecture
BLDC	Brushless DC
BSW	Basic Software
CAN	Controller Area Network
CCU	Counter Compare Unit
CCU	Capture Compare Unit
CDD	Complex Device Drivers
CMU	Clock Manamegent Unit
CPU	Central Proccessing Unit
DC	Direct Current
DMA	Direct Memory Access
EB	Elektrobit
ECU	Electronic Control Unit
GPIO	General Purpose Input Output
GUI	Graphical User Interface
GTM	Generic Timer Module
HW	Hardware
ICU	Interrupt Control Units
IDE	Integrated Development Environment
IR	Interrupt Router
ISR	Interrupt Service Routine
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
LIN	Local Interconnect Network
MCAL	Microcontroller Abstraction Layer
MC-ISAR	MicroControler Infineon Software ARchitecture
MCU	Microcontroller Unit
OS	operační systém
PLL	Phase-Locked Loop
PMS	Permanent Magnet Synchronous
PWM	Pulse Width Modulation
RTC	Real-Time Clock
RTE	Runtime Environment
SD	Secure Digital

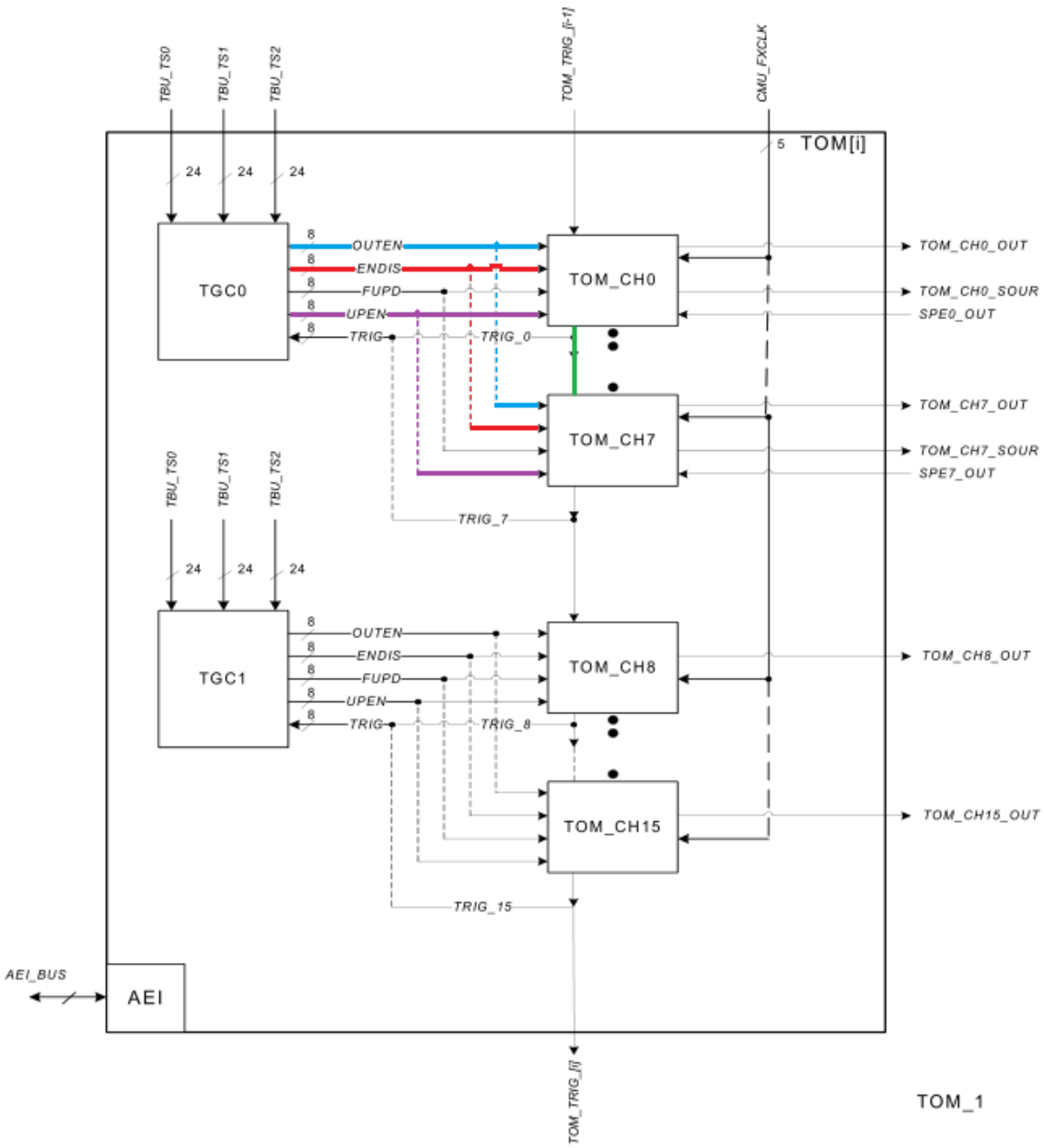
SOU	Signal Output Generation Unit
SPI	Serial Peripheral Interface
SRC	Service Request Control Register
SRN	Service Request Node
SW	software
SWC	Software Component
TGC	Tom Global Channel Control
TOM	Timer Output Module
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VADC	Versatile Analog-to-Digital Converter
VFB	Virtual Functional Bus

SEZNAM PŘÍLOH

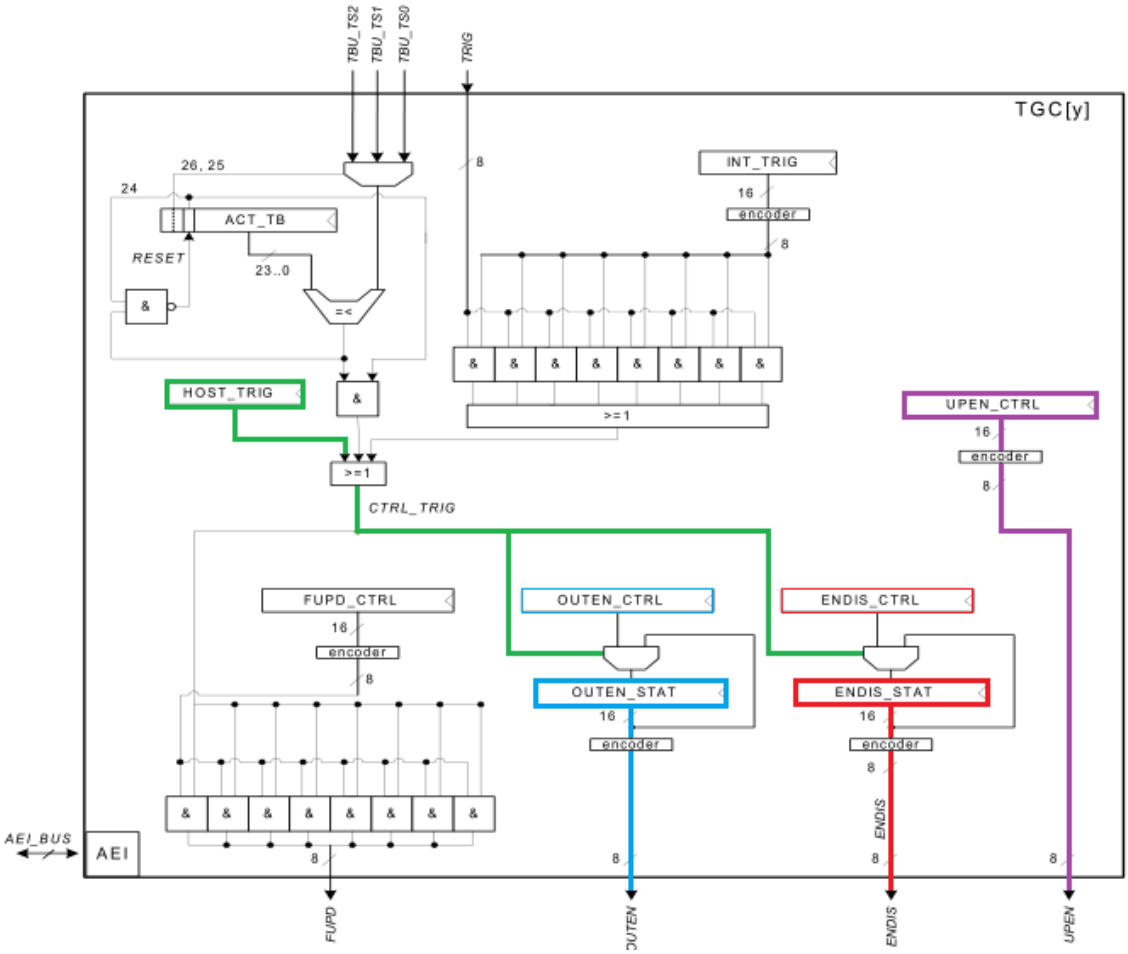
A GTM KONFIGURACE



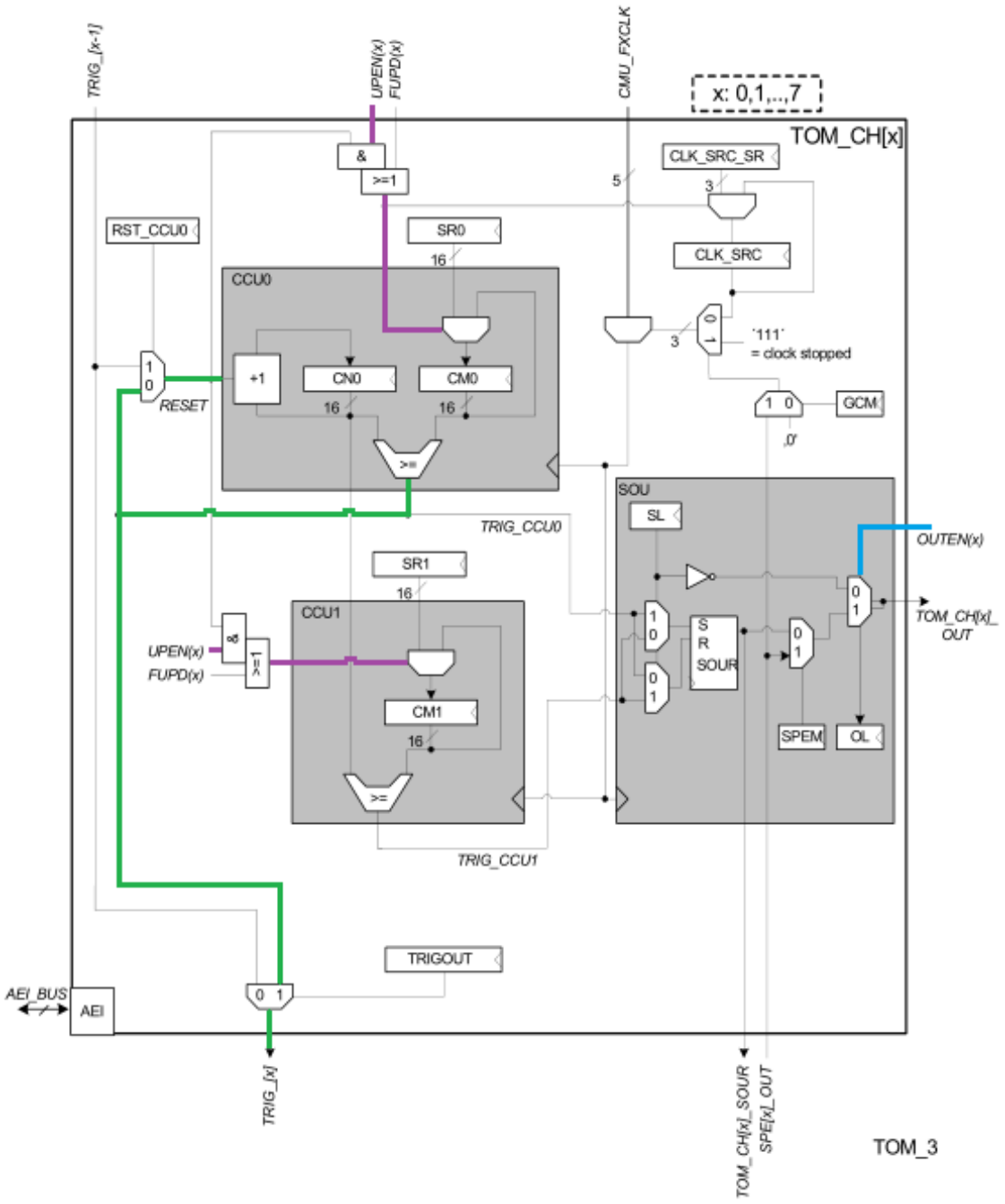
Obr. A.1: Nastavení hodinového signálu pro TOM moduly [10]



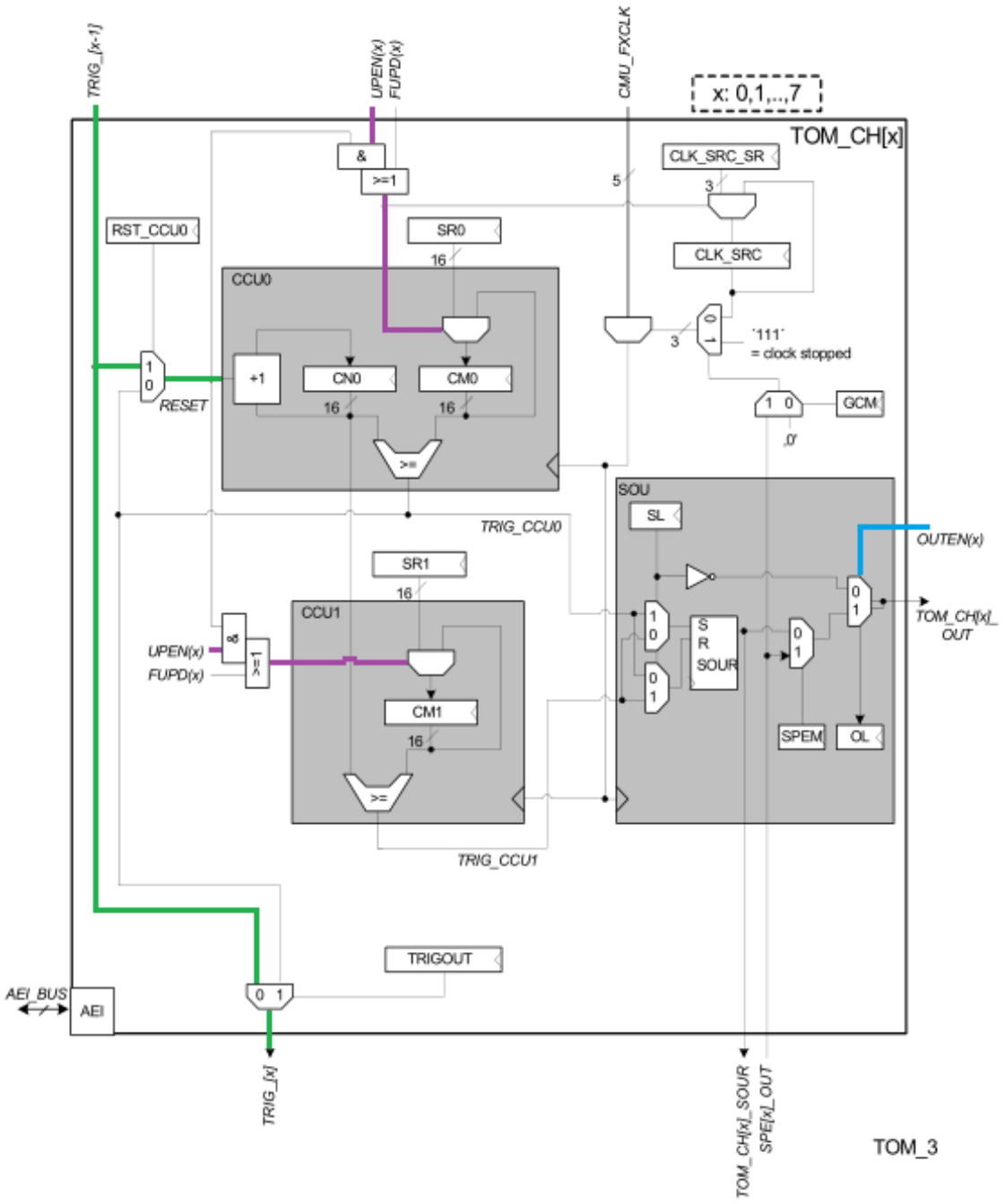
Obr. A.2: Vnitřní struktura jediného TOM modulu [10]



Obr. A.3: Vnitřní struktura TGC jednotky [10]



Obr. A.4: Vnitřní zapojení spouštěcího signálu referenčního kanálů [10]



Obr. A.5: Vnitřní zapojení spouštěcího signálu zbylých kanálů [10]

B GTM TRESOS KONFIGURACE

Tab. B.1: Konfigurace modulu CMU

CMU	
CmuEnableAllFixedClocks	TRUE
CmuFxdClkSourceSelect	CMU_CLOCK_0_DIVIDER
CmuEnableConfigurableClk0	TREU
CmuConfigurableClk0Div	4

Tab. B.2: Konfigurace modulu TOM - referenční kanál

TOM0_CH0	
TomChannelUsage	USED_BY_GTM_DRIVER
TomChannelEnable	ON_GLOBAL_TRIGGER
TomChannelOutputControl	ENABLE_OUTPUT_ON_GLOBAL_TRIGGER
TomChannelOutputSignalLevel	HIGH
TomChannelPortPinSelect	TOUT18_SELA_PORT0_PIN9
TomChUpdateEnableOnCn0Reset	FALSE
TomChannelClockSelect	GTM_FIXED_CLOCK_0
TomChannelCounterCn0Reset	ON_COMPARE_MATCH_ON_CHANNEL
TomChannelTriggerOutput	CCU0_TRIG_OF_CHANNEL

Tab. B.3: Konfigurace modulu TOM - zbylé kanály

TOM0_CH1	
TomChannelUsage	USED_BY_GTM_DRIVER
TomChannelEnable	ON_GLOBAL_TRIGGER
TomChannelOutputControl	ENABLE_OUTPUT_ON_GLOBAL_TRIGGER
TomChannelOutputSignalLevel	HIGH
TomChannelPortPinSelect	TOUT86_SELA_PORT14_PIN6
TomChUpdateEnableOnCn0Reset	FALSE
TomChannelClockSelect	GTM_FIXED_CLOCK_0
TomChannelCounterCn0Reset	ON_TRIGGER_FROM_PREV_CHANNEL
TomChannelTriggerOutput	TRIG_FROM_PREVIOUS_CHANNEL

Tab. B.4: Konfigurace výstupních kanálů modulu TOM

PORT	
PortPinDirection	PORT_PIN_OUT
PortPinInitialMode	PORT_PIN_MODE_ALT1

C VADC TRESOS KONFIGURACE

Tab. C.1: Konfigurace ovladače Adc - karta General

Adc driver	
AdcHwTriggerApi	TRUE
AdcMasterSlaveSync	TRUE

Tab. C.2: Konfigurace periferie VADC

VADC	
AdcSystemClock	/Mcu/Mcu/McuModuleConfiguration_0 /McuClockSettingConfig_0/McuClockReferencePoint
AdcAnalogClockDivider	0 až 31
AdcGlobChSampleTime	0
AdcGlobChResolution	CH_RES_12BIT

Tab. C.3: Konfigurace master převodníku

AdcHwUnit_0	
AdcArbitrationRoundLength	ARBITRATION_SLOTS_4
AdcHwUnitId	HWUNIT_ADC0
AdcSyncConvMode	ADC_MASTER

Tab. C.4: Konfigurace master kanálu

Adc0Channel_0	
AdcAnChannelNum	0
AdcInputClassSelection	/Adc/Adc/AdcConfigSet_0/AdcGlobInputClass_0
AdcSyncChannel	TRUE

Tab. C.5: Konfigurace synchronizace master převodníku s PWM

Adc0Group_0	
AdcGroupRequestSource	REQSRC1_NCH_SCAN
AdcGroupTriggSrc	ADC_TRIGG_SRC_HW
AdcHwExtTrigSelect	ADC_USE_GATE_PIN_TO_TRIG
AdcHwGatePin	ADC0_GTSEL_RS0_RS1_TIM_TRIG0
AdcHwTrigType	HW_TRIGG_EXT_REQ
AdcHwTrigSignal	ADC_HW_TRIG_RISING_EDGE
AdcGroupDefinition	/Adc/Adc/AdcConfigSet_0/AdcHwUnit_0/Adc0Channel_0

Tab. C.6: Konfigurace synchronizace master převodníku s PWM na straně GTM modulu

GtmTriggerForAdc_0	
GtmTrigger0Select	TRIG_2_TOM0_CH7

Tab. C.7: Konfigurace slave převodníků

AdcHwUnit_1	
AdcHwUnitId	HWUNIT_ADC1
AdcSyncConvMode	ADC_SLAVE

Tab. C.8: Konfigurace kanálu slave převodníků

Adc1Channel_0	
AdcAnChannelNum	8
AdcSyncConvMode	ADC_SLAVE
AdcInputClassSelection	/Adc/Adc/AdcConfigSet_0/AdcGlobInputClass_0
AdcSyncChannel	FALSE

Tab. C.9: Konfigurace skupiny slave převodníků

Adc1Group_0	
AdcGroupRequestSource	REQSRC1_NCH_SCAN
AdcGroupDefinition	/Adc/Adc/AdcConfigSet_0/AdcHwUnit_1/Adc1Channel_0

Tab. C.10: Konfigurace vstupních pinů převodníků

Port40_Pin0	
PortPinAnalogInput	ENABLE

D OBSAH PŘILOŽENÉHO CD

Na přiloženém CD je možné nalézt samotnou diplomovou práci, dále konečný konfigurační projekt z prostředí EB treosos studio 2012a a konečný projekt z prostředí Free TriCore Entry Tool Chain v4.6.6.0 se všemi integrovanými ovladači, jejich i hardwarovou konfigurací a vzorovou aplikací využívající vytvořené aplikační rozhraní.