České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Leoš Klik**

Studijní program: Otevřená informatika (magisterský)
Obor: Počítačové inženýrství

Název tématu: **Ovladač PC104 I/O modulu s možností ukládání sad záznamů měřených dat**

Pokyny pro vypracování:

1. Na základě poznatků z bakalářské práce [6] a požadavků uživatele aplikace vytvořte specifikaci požadavků na ovladač I/O modulu s podporou práce v reálném čase.
2. Vytvořte popis architektury ovladače vyhovující specifikovaným požadavkům a umožňující využití karty SDM 8540HR-1.
3. Zvolte vhodnou konfiguraci ADC, tak aby bylo možné dosáhnout periody vzorkování menší než 200 us.
4. Vytvořte v rozhraní ovladače funkce pro synchronizaci RT jádra.
5. Rozdělte program vhodným způsobem do modulů, tak aby cyklomatická komplexita nepřekročila hodnotu 100.
6. Naimplementujte systém pro definování sad záznamů a jejich ukládání včetně konfigurovatelné spouštěcí podmínky
7. Při vývoji ukládejte změny ve zdrojových souborech verzovacím programem Mercurial.

Seznam odborné literatury:

[1] http://rtd.com
[2] Messmer, H., P.: Dembowski, K.: Velká kniha hardware. Computer press, Praha 2005.
[3] Reinhart, D.: Řídicí software systému s maticovým měničem. Diplomová práce ČVUT, Praha 2006.
[4] Flígl, S.: RT PC-based control system - The 2nd generation. Technická dokumentace Ruhr-Universität, Bochum 2003.
[5] Martínek, R.: RT jádro pro PC104 systém. Bakalářská práce ČVUT, Praha 2011.
[6] Klik, L.: Ovladač pro PC104 I/O modul. Bakalářská práce ČVUT, Praha 2011.

Vedoucí: Ing. Stanislav Flígl, Ph.D.

Platnost zadání: do konce letního semestru 2012/2013

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 4. 1. 2012

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídící techniky

Diplomová práce

Ovladač PC104 I/O modulu s možností ukládání sad záznamů měřených dat

Bc. Leoš Klik

Vedoucí práce: Ing. Stanislav Flígl, Ph.D.

Studijní program: Otevřená informatika, Navazující magisterský

Obor: Počítačové inženýrství

8. května 2013

Poděkování

Především děkuji vedoucímu své diplomové práce panu Ing. Stanislavu Flíglovi, Ph.D. za cenné rady, připomínky a za velké množství času strávené konzultacemi k této práci. Dále také kolegovi Bc. Radomíru Martínkovi za pomoc při syntéze a odladění této práce s jeho projektem, panu Ing. Janu Bauerovi za podnětné komentáře k vývoji aplikace a Mgr. Marii Klikové za jazykovou korekturu práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Lubné dne 30. 4. 2013 Leoš Klír .....

Abstract

The work deals with an extension of a driver for the RTD®SDM8540HR-1 card. That is an PCI/104 I/O module. The former version had been developed within my bychelor's thesis. The current driver implements a new approach of internal synchronization. It is possible to take samples from A/D converter more than once during one RT period thanks to this change. The basic configuration of the card has been also modified to be able to shorten RT period below 200 μs. It also implements a system for measured values logging with configurable start/stop condition. All user functions are included in one static library called SDM8540.lib. Since the C language lacks the namespace support, names of all functions start with the SDM8540_ or LOG_ prefix because of better transparency for the user programmed source code. The driver is supposed to be used with the 2nd generation of the real-time kernel developed by R. Martínek and it isn't compatible with its previous version. The driver is designed to be used in a real-time system.

Abstrakt

Cílem této práce je doplnit ovladač pro PCI/104 vstupně/výstupní modul RTD®SDM8540HR-1 vyvinutý v rámci bakalářské práce. Práce implementuje změnu způsobu interní synchronizace kvůli možnosti snímat vzorky A/D převodníkem vícekrát během jedné periody. Dále upravuje konfiguraci karty tak, aby bylo možné dosáhnout RT periody kratší než 200 μs. Také se zabývá implementací systému pro záznam měřených hodnot s konfigurovatelnou start/stop podmínkou. Veškeré funkce jsou obsaženy ve statické knihovně SDM8540.lib a především proto, že jazyk C nepodporuje jmenné prostory, začínají prefixem SDM8540_ nebo LOG_z důvodu lepší rozlišitelnosti ve zdrojovém kódu. Ovladač je vázán na novou verzi jádra R. Martínka a s předchozí verzí není kompatibilní. Rovněž je ovladač navržen s ohledem na využití v systému pracujícím v reálném čase.

Obsah

1 Úvod **1**

2 Popis problému, specifikace cíle **3**
2.1 Cíl práce . . . 3
2.2 Výsledek práce . . . 4
2.3 Existující implementace . . . 4

3 Analýza a návrh řešení **5**
3.1 Specifikace požadavků uživatele aplikace . . . 5
3.1.1 Funkční požadavky . . . 5
3.1.2 Nefunkční požadavky . . . 6
3.2 Použitý programovací jazyk . . . 6
3.3 Vývojové prostředí . . . 6
3.4 DOS a 32 bitové rozšíření . . . 7
3.5 Mercurial . . . 7
3.6 Cyklomatická komplexita . . . 9
3.6.1 Příklady výpočtu cyklomatické komplexity . . . 9
3.6.2 Měření pomocí nástrojů . . . 10
3.7 Číslování verzí . . . 11
3.8 Návrh řešení . . . 11
3.8.1 Návrh struktury systému . . . 11
3.8.2 Inicializace karty SDM8540 . . . 12
3.8.3 Přenos převedených hodnot do paměti PC . . . 12
3.8.4 Rozšířené možnosti synchronizace . . . 13
3.8.5 Ukládání naměřených dat s volitelnou podmínkou . . . 13
3.8.6 Spolupráce s RT jádrem . . . 15

4 Realizace **16**
4.1 Synchronizace RT jádra . . . 16
4.1.1 Interní zdroj synchronizačních pulzů . . . 16
4.1.1.1 Charakteristika časovače Pacer Clock . . . 17
4.1.1.2 Charakteristika časovače Burst Clock . . . 18
4.1.1.3 Možnosti interní synchronizace v původním ovladači . . . 18
4.1.1.4 Možnosti interní synchronizace v nové verzi ovladače . . . 19
4.1.2 Channel Gain Table . . . 20

4.1.2.1 Pause bit . . . 21
4.1.2.2 Skip bit . . . 21
4.1.2.3 Příklad použití CGT . . . 21
4.1.3 Externí zdroj synchronizačních pulzů . . . 22
4.1.3.1 Externí spouštění časovače Pacer Clock . . . 22
4.1.3.2 Externí časovač Pacer Clock . . . 22
4.1.3.3 Externí spouštění časovače Burst Clock . . . 22
4.1.3.4 Připojení externího synchronizačního signálu . . . 23
4.2 Hodiny reálného času . . . 23
4.2.1 Charakteristika obvodu . . . 23
4.2.2 Získání času z RTC . . . 24
4.2.2.1 Vyčtení hodnot registrů . . . 24
4.2.2.2 Update In Progress . . . 25
4.2.2.3 Přítomnost záložního napájení hodin . . . 25
4.2.3 RTC na PC modulu RTD CML147786CX650HR-128 . . . 25
4.3 Ukládání sad záznamů měřených dat . . . 25
4.3.1 Kruhová fronta . . . 26
4.3.1.1 Realizace kruhové fronty . . . 26
4.3.2 Hlavička záznamu . . . 27
4.3.2.1 Pořadové číslo záznamu . . . 28
4.3.2.2 Časové odečty z RTC . . . 28
4.3.2.3 Zabezpečení integrity dat . . . 29
4.3.3 Záznam hodnot z A/D převodníku . . . 29
4.3.4 Záznam hodnot z uživatelské funkce . . . 29
4.3.4.1 Mapa funkcí a proměnných . . . 29
4.3.5 Postup vlastního záznamu . . . 30
4.4 Direct Memory Access (DMA) . . . 31
4.4.1 DMA na kartě SDM8540 . . . 31
4.4.1.1 Non-Chaining Mode . . . 32
4.4.1.2 Chaining Mode . . . 32
4.4.2 Implementace DMA . . . 32
4.5 Popis nových funkcí ovladače . . . 32
4.5.1 Inicializační funkce karty . . . 32
4.5.2 Synchronizační funkce . . . 33
4.5.3 Funkce pro ovládání A/D převodníku . . . 34
4.5.4 Funkce pro využití DMA . . . 34
4.5.5 Funkce ovládající čítače . . . 36
4.5.6 Funkce pro záznam sad hodnot . . . 36
4.5.7 Verzování ovladače . . . 38

5 Testování **39**
5.1 Testovací aplikace . . . 39
5.2 Testování minimální délky periody . . . 39
5.3 Testování záznamu hodnot . . . 40
5.4 Cyklomatická komplexita ovladače . . . 40

6 Závěr **44**

A Seznam zkratek **48**

B Ukázka testovací aplikace **50**

C Report z aplikace Crystal FLOW **52**

D Karta SDM8540HR-1 **53**

E Obsah přiloženého CD **54**

Seznam obrázků

3.1 Prohlížeč repozitáře programu TortoiseHg 8
3.2 Graf toku řízení 10
3.3 Základní model struktury systému 12
3.4 Možnosti vzájemného spouštění časovačů 14

4.1 Možnosti vzájemného spouštění časovačů 17
4.2 Synchronizace na jednu periodu Pacer Clocku 19
4.3 Synchronizace s násobností Pacer Clocku n = 3 20
4.4 Připojení externího triggeru 23
4.5 Rozložení konektorů na CPU modulu CML147786CX650HR-128 26
4.6 Schematické znázornění kruhové fronty 27
4.7 Princip DMA 31

5.1 C/C++ Code Complexity Measurement Tool 41
5.2 Crystal FLOW for C 42
5.3 Ukázka grafu toku řízení 43

C.1 Část reportu z aplikace Crystal FLOW for C 52

D.1 I/O karta RTD SMD8540HR-1 53

E.1 Obsah přiloženého CD 54

Seznam tabulek

4.1 Porty RTC 24

5.1 Cyklomatická komplexita prvních 10 funkcí od nejvyšší 43

Seznam zdrojových kódů

3.1 Cyklomatická komplexita M = 1 9
3.2 Cyklomatická komplexita M = 3 9
4.1 Ukázka nastavení CGT 21
4.2 Čtení z I/O portu 24
4.3 Přepočet BCD na dekadický tvar 25
4.4 Struktura kruhové fronty 27
4.5 Hlavička záznamu 28
4.6 Struktura s časovým záznamem z RTC 28
4.7 Struktura s verzí ovladače 38
5.1 Konfigurace aplikace testující délku periody 39
B.1 Testovací aplikace testující délku periody 50

Kapitola 1

Úvod

Již před vznikem mé bakalářské práce „Ovladač pro PC/104 I/O modul“ bylo jako platforma vybrána standardní 32-bitová architektura IA-32. Vzhledem k vzrůstajícím požadavkům na výkon regulačních prvků, které přichází společně se zvyšující se komplexitou regulačních algoritmů, se toto ukázalo jako velmi vhodný krok. Ve srovnání se specializovanými DSP procesory, které mohou být k provozování regulačních algoritmů rovněž využity, je zde nespornou výhodou možnost mnohostranného využití. Vzhledem ke zpětné binární kompatibilitě sahající až do roku 1978 je pravděpodobné, že i nadále bude k dispozici příslušný hardware. V tom lze spatřovat dvojí ochranu investice spočívající v čase věnovanému vývoji RT jádra a ovladače I/O modulu. Vyvinutý software tak bude použitelný i s rychlejšími zpětně kompatibilními moduly v budoucnu.

Výkon standardního PC umožňuje k samotnému regulačnímu algoritmu doplnit například možnost zaznamenávání naměřených dat, komunikaci s okolím prostřednictví ethernetu nebo sériové linky. Všechny tyto aplikace je možné vyvíjet ve standardním prostředí (včetně využití hardwarové podpory aritmetiky v plovoucí řádové čárce v jednoduché (32-bit) i dvojité řesnosti (64-bit)), které je známé z osobních počítačů, což je pro programátora určitě výhodné. S ohledem na práci aplikace v reálném čase by však měl být systém vyvinut tak, aby jeho vlastní režie nenarušovala běh RT aplikace.

Z důvodu kompaktnosti zařízení je projekt koncipován pro využití na PC formátu PC/104, které poskytuje funkčnost a výkon srovnatelný s klasickým PC, ale vzhledem k velikosti je vhodnější pro využití ve vestavných aplikacích [1]. Práce konkrétně zpracovává ovladač pro I/O kartu SDM8540HR-1 vyvinutou společností RTD, která je prostřednictvím PCI sběrnice připojena k CPU modulu CML147784CX650HR-128 rovněž od společnosti RTD.

Samotný ovladač vyvinutý v rámci bakalářské práce a přepracovaný a doplněný v této práci tvoří rozhraní mezi samotnými hardwarovými prostředky poskytovanými karty SDM8540 a řídícím algoritmem definovaným uživatelem regulačního systému. Ovladač tak odstiňuje uživatele od nutnosti studovat konfigurační registry karty a využívat přímého přístupu do konfigurační paměti a nabízí pro veškeré ovládání a nastavení pohodlnější způsob realizovaný poskytnutými inicializačními a výkonnými funkcemi. Na cestě mezi ovladačem a samotným uživatelem běží ještě samotné RT jádro, které vyvinul v rámci své bakalářské a diplomové

[1] K dispozici jsou i moduly s průmyslovým rozsahem pracovních teplot okolí (industry temperature range) -40°C až 85°C.

práce kolega Radomír Martínek. Toto jádro zajišťuje správnou inicializaci karty a zprostředkovává uživateli samotná volání ovladače.

Kapitola 2

Popis problému, specifikace cíle

2.1 Cíl práce

Hlavním cílem této práce je doplnit ovladač pro PC/104 PCI kartu RTD SDM8540HR-1 vyvinutý v rámci mé bakalářské práce „Ovladač pro PC104 I/O modul“ [6] o možnost ukládání záznamu naměřených dat. Ukládat je možné jak hodnoty získané z A/D převodníku a dalších vstupů karty, tak hodnoty vypočtené v uživatelem definovaných funkcích. Tyto funkce nejsou přímou součástí ovladače realizovaného touto prací, ale jsou typicky definovány v uživatelské aplikaci (např. řídícím algoritmu), který prostředky tohoto ovladače využívá.

Dalším doplněním ovladače je úprava inicializace tak, aby bylo možné zakázat veškerá přerušení na lokální sběrnici. Tato změna vychází především z toho, že v předchozí verzi ovladače nebylo možné dosáhnout dostatečně rychlého čtení převedených hodnot pomocí softwarového vyčítání FIFO fronty A/D převodníku. Generování těchto přerušení znemožňovalo dosáhnout délky periody RT smyčky kratší než 200 μs. Předpokladem řešení tohoto problému bylo na základě konzultace s uživatelskou podporou RTD využití DMA pro přenos hodnot získaných z A/D převodníku do paměti PC pro další zpracování. Ukázalo se však, že využití DMA bez možnosti použít přerušení, které je v aplikaci zakázáno, je značně nekomfortní a úprava inicializace tento problém řeší.

Přepracovány jsou rovněž i možnosti synchronizace karty při běhu v RT režimu. Zde je nyní možné převádět hodnoty pomocí A/D převodníku vícekrát během jedné synchronizační periody [1]. Vše závisí na nastavení uživatelem. Vzhledem k nemožnosti dosáhnout složitějších konfigurací časování s využitím standardních časovačů integrovaných na desce jsou rozvinuty také možnosti externí synchronizace např. pomocí připojené Field-Programmable Gate Array (FPGA) karty.

Samozřejmostí jsou i nadále poskytované funkce konfigurace a ovládání A/D převodníku, digitálních vstupů a výstupů a na kartě integrovaných časovačů, jak byly implementovány a popsány v mé bakalářské práci [6].

Primární předpokládané využití karty je běh ve spolupráci s CPU modulem RTD CML147786CX650HR-128 nebo obecně libovolným x86 modulem formátu PC/104+. Softwarová výbava je zajištěna Real Time (RT) jádrem vyvinutým kolegou Radomírem Martínkem

[1] takto získaná data lze s výhodou vužít v pokročilých algoritmech identifikace regulovaného systému

v rámci jeho bakalářské a diplomové práce [8] [9]. Kvůli využití v RT režimu je jedním ze základních požadavků na ovladačem implementované funkce co možná nejnižší režie programu.

Z hlediska organizace práce ze strany vedoucího práce a koordinace postupu s R. Martínkem je logickým požadavkem využívat pro správu zdrojového kódu verzovací systém Mercurial. Ten je podrobněji popsán dále 3.5.

Práce se zaměřuje zejména na:

- implementaci systému pro definování sad záznamů a jejich ukládání
- konfiguraci start a stop podmínky zaznamenávání
- synchronizaci RT jádra
- dosažení periody vzorkování menší než 200 μs
- možnost převádět A/D převodníkem v průběhu jedné synchronizační periody více sad hodnot
- součinnost s přepracovaným RT jádrem kolegy R. Martínka
- minimální režii ovladače
- verzování ovladače i RT jádra

2.2 Výsledek práce

Tak jako tomu bylo v mé bakalářské práci, i zde je výsledkem práce statická knihovna `SDM8540LIB.lib`, která poskytuje uživateli veškeré inicializační a výkonné funkce pro práci s kartou SDM8540HR-1. Tato knihovna předpokládá použití v systému společně s RT jádrem R. Martínka, které je ve verzi 2 realizováno rovněž jako statická knihovna. Obě tyto knihovny vyžadují slinkování do uživatelské aplikace, která rovněž definuje uživatelské funkce s regulačním algoritmem a provede inicializaci a nastavení karty právě prostřednictvím funkcí poskytnutých ovladačem. Nově implementované funkce jsou i s jejich parametry podrobně popsány v kapitole 4.5.

2.3 Existující implementace

Stejně jako v době vzniku první verze ovladače [6] mi ani teď nejsou pro kartu SDM8540HR-1 známa žádná existující veřejně dostupná řešení. Funkčně srovnatelný je systém vyvinutý vedoucím práce Ing. Stanislavem Flíglem, Ph.D., nazvaný Stan2002 [2]. V tomto systému je využita I/O karta Meilhaus Electronic ME2600i se standardním PCI rozhraním určená pro provoz v klasickém počítači. Tato karta poskytuje, stejně jako SDM8540, 16 kanálů analogového vstupu (8 diferenciálních). Samplovací frekvence této karty je 333 kHz (perioda 3.0 μs) oproti 1.25 MHz (perioda 0.8 μs) u SDM8540. Karta neposkytuje interní časovače. Pro synchronizaci systému Stan2002 bylo tedy využito externího zdroje synchronizačních pulzů. V této implementaci je ovladač karty napsán jako jeden modul jádra a je zakompilován ze zdrojového kódu přímo do RT aplikace. [4] [10] [6]

[2] Ten byl využit následně v několika dizertačních pracech [3] [7] [19] [1].

Kapitola 3

Analýza a návrh řešení

3.1 Specifikace požadavků uživatele aplikace

3.1.1 Funkční požadavky

A/D převodník

- Zvolení vhodné konfigurace A/D převodníku tak, aby byl možný převod v průběhu periody RT systému a hodnoty byly využitelné v témže cyklu pro regulační algortimus.
- Umožnění převodu A/D převodníkem vícekrát během jedné RT periody.

Synchronizace RT jádra

- Vytvoření v rozhraní ovladače funkce pro synchronizaci RT jádra.
- Zvolení implicitní konfigurace interního časování, která bude použita, pokud uživatel nezvolí jinak.

Záznam naměřených hodnot

- Vytvoření systému pro ukládání sad záznamů hodnot, hodnotami mohou být veličiny naměřené A/D převodníkem, případně také hodnoty vypočtené v uživatelské funkci.
- Možnost definovat podmínku ukončení zaznamenávání.
- Poskytnout možnost zaznamenat ještě dané množství hodnot po splnění ukončovací podmínky.
- Opatřit hlavičku každého záznamu hodnot časovým údajem z hodin reálného času v PC.
- Zabezpečit integritu dat a kontrolu správného pořadí záznamů.

Ostatní požadavky

- Poskytnout funkci pro zjištění verze ovladače.
- Kontrolovat verze modulů v rámci celého systému.

3.1.2 Nefunkční požadavky

- Úprava konfigurace tak, aby bylo možné dosáhnout periody vzorkování menší než 200 μs.
- Rozdělení programu vhodným způsobem do modulů podle logiky funkčnosti.
- Cyklomatická komplexita nesmí překročit hodnotu 100.
- Ukládání zdrojových souborů verzovacím systémem Mercurial.

3.2 Použitý programovací jazyk

Ovladač vyvinutý v mé bakalářské práci [6] je implementován stejně jako RT jádro R. Martínka v jazyce C, proto i nadále vývoj probíhal v tomto jazyce. Důvody pro výběr jazyka C byly následující:

- většina funkčnosti implementované v ovladači si vystačí se základními konstrukcemi a příkazy, které jsou dostupné ve většině imperativních jazyků
- ovladač využívá minimum jazykem poskytovaných knihoven
- nejsou využity žádné dynamické datové struktury
- potřeba přímého přístupu k hardware a paměti
- většina konfigurace karty je založena na přímém zápisu do konfigurační paměti

Zejména požadavek na možnost přímého přístupu k hardware v podstatě vylučuje jazyky běžící ve virtuálních strojích a jazyky interpretované. Interpreter, případně virtuální stroj, většinou odstiňuje programátora od hardware počítače a tato vlastnost by byla velmi nevhodná. V časově kritické aplikaci je rovněž z důvodu rychlosti a spolehlivosti výhodnější použít kompilovaný jazyk. Z těchto důvodů je například jazyk Java nevhodný pro implementaci nízkoúrovňového ovladače jako je tento [6].

3.3 Vývojové prostředí

Pro projekt založený již při bakalářské práci [6] bylo na základě doporučení vedoucího práce zvoleno Open Watcom IDE 1.9. Toto prostředí je volně dostupné na adrese `http://www.openwatcom.org`. Poskytuje podporu pro jazyky C, C++ a Fortran a predevším důležitou možnost vytvářet 32 bitové aplikace psané pro OS DOS. Obsahuje také rozsáhlou

nápovědu k podporovaným jazykům a popis standardních knihoven, které jsou v některých detailech v podání Open Watcomu odlišné, například od knihoven známých z gcc nebo jazyka C od Borlandu [1]. Vzhledem k poměrně zastaralému rozhraní Open Watcom IDE, a mnohdy ne příliš intuitivnímu ovládání, jsem většinu zdrojového kódu napsal v editoru Notepad++ v5.9.4, který je rovněž volně dostupný na `http://notepad-plus-plus.org/`. Vzhledem k možnosti využít velké množství doplňků pro Notepad++ jsem pomocí doplňku NppExec využil možnosti kompilace externím kompilátorem přímo v prostředí Notepad++. Doplněk obsahuje konzoli s výstupem kompilátoru přímo do Notepadu++ a pro využití s kompilátorem Open Watcomu stačilo nadefinovat správné cesty ve skriptu pluginu. V prostředí Open Watcomu tak probíhalo již pouze vytváření nových souborů v projektu a kompilace finální `.lib` knihovny.

3.4 DOS a 32 bitové rozšíření

RT jádro, nyní již ve druhé verzi, je spouštěno i nadále z operačního systému DOS. V případě CPU modulu od RTD se jedná o výrobcem dodaný ROM-DOS od společnosti Datalight. Ten byl vyvinut jako operační systém kompatibilní s MS-DOS především pro vestavěné aplikace. Jádro R. Martínka po spuštění přebírá veškerou kontrolu nad hardware a nevyužívá žádné prostředky operačního systému. Kompilováno je nově rovněž jako statická knihovna a pro běh potřebuje být zkompilováno do uživatelem vytvořené aplikace jako 32 bitová aplikace pro DOS, proto i můj ovladač má formát statické knihovny pro DOS. Možnost využití 32 bitového prostředí je založena 32 bitovém rozšíření, které je právě kompilátorem Open Watcomu poskytováno. Toto rozšíření umožňuje programům spuštěným z prostředí DOSu překročit paměťový limit 640 KB a adresovat tak větší množství operační paměti. Rovněž lze využít 32 bitové instrukce procesoru.

Vlastní 32 bitové rozšíření (extender) je využito ve chvíli, kdy uživatel spustí 32 bitový program. Extender se před vlastním spuštěním programu načte do paměti, zavede v prostředí DOSu vlastní systém pro správu paměti a pokud je to potřeba, spuštěný program nahraje do paměti na adresu vyšší než 1MB, kam již samotný DOS zapisovat neumí. Program pak běží ve chráněném režimu a v případě potřeby, například přístup ke službám BIOSu, zajišťuje právě extender přepnutí do režimu reálného. Po skončení programu extender provede uvolnění paměti a sám se rovněž ukončí [15] [6].

Na základě testování a poznatku z bakalářské práce byl pro jádro zvolen extender PMODE/W nabízený mezi dalšími prostředím Open Watcomu. Pro samotnou knihovnu s ovladačem není toto příliš podstatné, protože nahrání knihovny do paměti je zajištěno při spuštění programu, do kterého je knihovna zakompilována. Pro testování jsem však využíval kompilaci projektu do spustitelného formátu `*.exe` a extender PMODE/W jsem rovněž využil.

3.5 Mercurial

Projekt ovladače, a s tím související projekt R. Martínka RT jádro, byl od počátku veden ve verzovacím systému Mercurial. Již s požadavkem v bakalářské práci byl proto založen

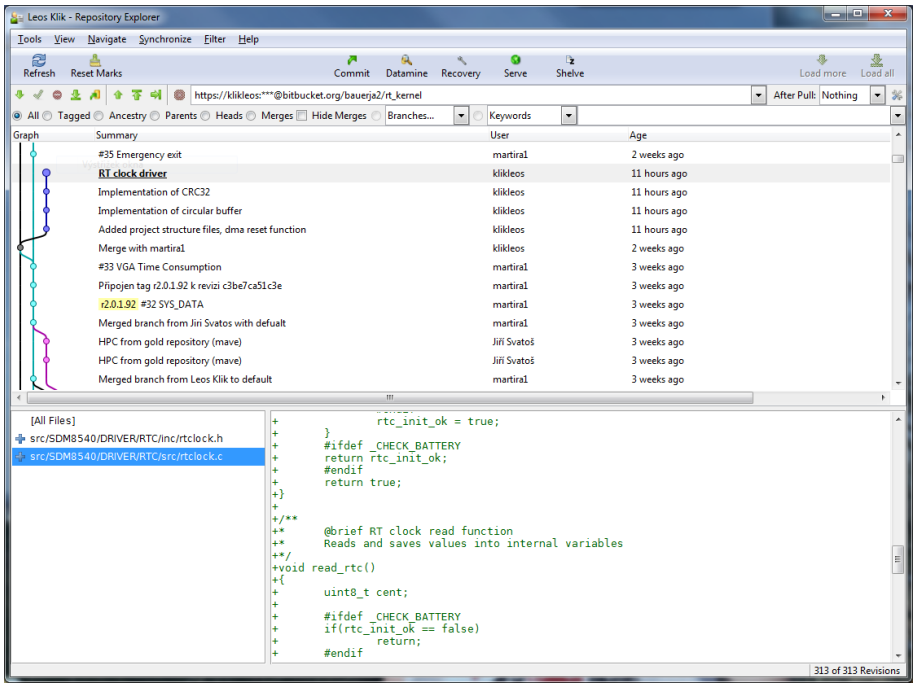
[1] Borland, později Inprise, v současnosti Embarcadero.

repozitář na serveru `http://bitbucket.org`. V této práci byl pro oba projekty Mercurial, konkrétně ve verzi 1.6.4, nadále využíván.

Mezi nesporné výhody Mercurialu patří určitě podobnost příkazů s rozšířeným systémem Subversion a zároveň daleko bohatší možnosti využití srovnatelné se systémem Git. Ve srovnání s Gitem je bezesporu výhodou nativní multiplatformnost. Distribuovaný model repozitáře funguje na principu hlavního repozitáře, který je umístěn na serveru (v případě tohoto projektu `http://bitbucket.org`), lokálních repozitářů pro každého uživatele a pracovních adresářů těchto uživatelů.

Každý uživatel s přístupem do repozitáře pracuje nad svým pracovním adresářem umístěným lokálně na jeho počítači. V něm probíhají veškeré změny v kódu. Pokud je uživatel s provedenou změnou spokojen, přenese změny operací *COMMIT* z pracovního adresáře do svého lokálního repozitáře, který se nachází taktéž u něj v počítači. Druhým krokem je přenesení lokálního repozitáře do hlavního. K tomu slouží operace *PUSH*. Naopak pro stažení změn do svého lokálního repozitáře je určena operace *PULL*. Mezi další důležité příkazy patří *UPDATE* - promítne změny z lokálního repozitáře do pracovního adresáře, *MERGE* - ten slouží ke sloučení více větví, které mohou vzniknout, pokud více uživatelů provede úpravy nad jedním repozitářem a *REVERT* - pro navrácení se k některé minulé verzi.

Jako grafické rozhraní se přímo nabízí využití TortoiseHg (ve verzi 1.1.4), které je dostupné pro Windows, MacOS i Linux. Mercurial včetně TortoiseHg je distribuován zdarma na adrese `http://tortoisehg.bitbucket.org/`



Obrázek 3.1: Prohlížeč repozitáře programu TortoiseHg

Větší část okna zabírá historie verzí v repozitáři. U každé odeslané změny je vidět, kdo a kdy ji provedl. Vlevo jsou barevně zobrazeny jednotlivé větve, jejich sloučení a vznik nové.

Spodní část zobrazuje podrobnosti vybrané změny. V levé části jde o seznam souborů, ve kterých proběhla změna, případně byly přidány/odebrány. V pravé části je zobrazena změna v samotném souboru. Je tak možné zjistit, které řádky byly jakým způsobem modifikovány [6].

3.6 Cyklomatická komplexita

Cyklomatická komplexita je softwarová metrika měřící složitost programu vyvinutá v roce 1976 Thomasem McCabem. Tato metrika se často používa v interních firemních standardech a kontroluje se také při auditech vývoje SW. Principem měření složitosti je zjištění počtu možných průchodů kódem. Vypočítat komplexitu je možné pomocí grafu toku řízení. Takový graf zachycuje všechny rozhodovací body programu a na základě jeho parametrů je komplexita zjistitelná. Komplexita předtavuje počet nezávislých cest takovým grafem. Například cyklomatická komplexita kódu, který neobsahuje cyklus nebo větvení, je rovna jedné. Jedna podmínka `if` například zvedne komplexitu na hodnotu 2, protože podmínka vytvoří dvě možné pokračování průchodu kódem [2] [14].

Obecně je možné vypočítat komplexitu pomocí následujícího vzorce:

$$M = E - N + 2P \tag{3.1}$$

Kde:

- **M** představuje cyklomatickou složitost
- **E** je počet hran v grafu
- **N** je počet uzlů v grafu
- **P** představuje počet připojených komponent, neboli ukončovacích uzlů

3.6.1 Příklady výpočtu cyklomatické komplexity

Následující kód 3.1 obsahuje pouze jednu možnou cestu průchodu, jeho komplexita je tedy M = 1.

```
int get_value(int index)
{
  return array[index];
}
```

Zdrojový kód 3.1: Cyklomatická komplexita M = 1

Tento druhý kód 3.2 již obsahuje dvě větvení, jeho komplexita bude tedy vyšší než u předchozího příkladu. Následuje graf toku řízení 3.2 a výpočet cyklomatické komplexity.

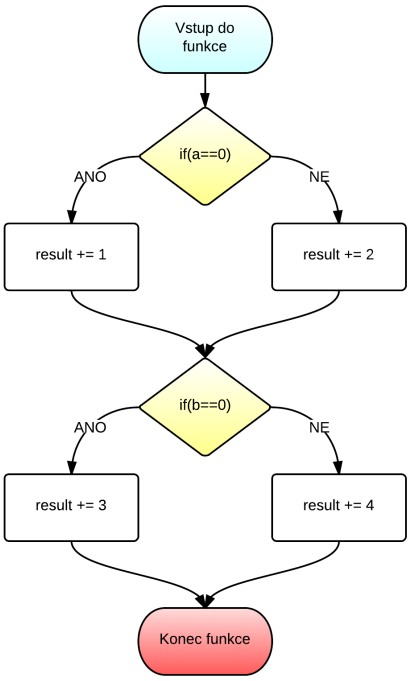
```
int get_result(int a, int b)
{
  int result = 0;
```

```
  if (a == 0)
    result += 1;
  else
    result += 2;

  if (b == 0)
    result += 3;
  else
    result += 4;

  return result;
}
```

Zdrojový kód 3.2: Cyklomatická komplexita M = 3



Obrázek 3.2: Graf toku řízení

Výpočet cyklomatické komplexity pak bude vypadat:

$$M = E - N + 2P = 9 - 8 + 2 * 1 = 3 \tag{3.2}$$

3.6.2 Měření pomocí nástrojů

Pro měření cyklomatické komplexity funkcí ovladače jsem využil Code Complexity Tool, což je volně dostupný nástroj, který lze získat na adrese `https://sites.google.com/site/maltapplication/`. Pro generování grafů toku řízení jsem pak použil software SGV SARC

Crystal Flow for C v. 4.72 v jeho Evaluation verzi, která byla pro tento účel dostačující. Program dokáže rovněž počítat cyklomatickou komplexitu a další metriky zdrojového kódu. Je dostupný na adrese `http://www.sgvsarc.com/downloads.htm`.

3.7 Číslování verzí

Společně s přidáváním nové funkčnosti do již existující aplikace vzniká potřeba udržovat informace o verzích jednotlivých komponent. Hledisko kompatibility je v tomto ohledu nejdůležitější, protože je třeba zajistit kontrolu vzájemné kompatibility jednotlivých modulů aplikace. Tím spíše, pokud jsou moduly vyvíjeny různými programátory. Aplikace, ve které je použit ovladač zpracovaný v rámci této práce, se skládá z několika prvků. Prvním je samotný tento ovladač, dalším je jádro R. Martínka, které ovladač využívá, další částí je například FPGA karta s vlastní aplikací J. Bauera. Z důvodu potřeby sledovat a ve finální aplikaci zajistit kompatibilitu je pro jádro a ovladač zavedeno číslování verzí v následujícím formátu

A.B.CC.DD

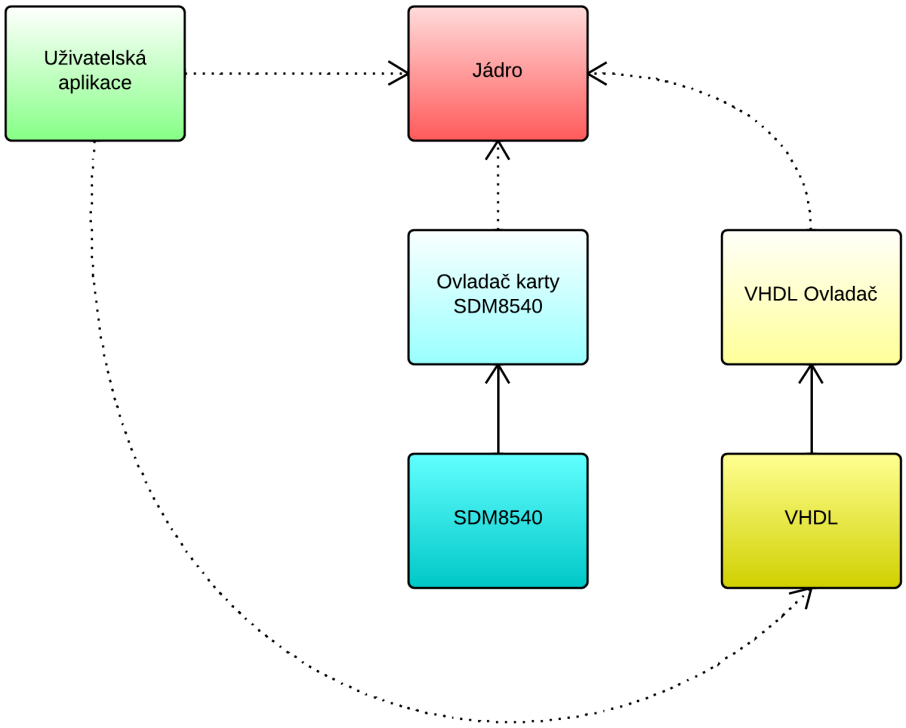
- **A** určuje kompatibilitu verzí, například verze 1.B.CC.DD nezaručuje kompatibilitu s verzí 2.B.CC.DD, možné hodnoty 1..9
- **B** určuje release v rámci kompatibility hlavní verze, možné hodnoty 1..9
- **CC** určuje opravu chyby, servicepack v rámci hlavní verze, možné hodnoty 0..99
- **DD** určuje aktuální sestavení, možné hodnoty 99..0

3.8 Návrh řešení

Základní návrh ovladače vychází ze struktury vzniklé v mé bakalářské práci. Tato první verze ovladače byla složena z množství nezávislých funkcí poskytovaných volajícímu systému (uživateli). Vznikala zde pouze závislost mezi některými inicializačními a výkonnými funkcemi. Nová verze ovladače, vzniklá v rámci této diplomové práce, zahrnuje veškerou tuto funkčnost a přidává některé další moduly, dále popsané. I přes zachování funkčnosti staré verze ovladače není ovladač verze 2 kompatibilní s původním jádrem R. Martínka. Je to dáno především kompletním přepracováním jádra a rovněž změnou některých způsobů inicializace ovladače, především předáním hodnot jako počet převáděných kanálů z uživatelské aplikace do ovladače.

3.8.1 Návrh struktury systému

Protože se celá aplikace skládá z několika různých systémů, je logické její rozdělení do modulů. Závislost modulů ukazuje obrázek 3.3. Tato práce se zabývá implementací ovladače karty SDM8540, diplomová práce kolegy R. Martínka pak RT jádrem a disertační práce pana J. Bauera pak modulem VHDL. Samotná uživatelská aplikace využívá služeb jádra a jeho prostřednictvím volá funkce dalších ovladačů. Obsah uživatelské aplikace je především nastavení parametrů jednotlivých modulů a implementace uživatelského (regulačního) algoritmu.



Obrázek 3.3: Základní model struktury systému

3.8.2 Inicializace karty SDM8540

Inicializace karty probíhá podobným způsobem jako v předešlé verzi ovladače. Karta je připojena k CPU modulu prostřednictvím PCI sběrnice. Je proto nutné skenovat všechny sběrnice a všechna na nich připojená zařízení a na základě identifikace karty číslem výrobce a modelovým číslem zařízení, které lze získat z PCI konfiguračního registru, zjistit její přítomnost. Dalším krokem je nastavení bázové adresy paměťové oblasti, kam jsou namapovány konfigurační a ovládací registry karty. Vymazání staré konfigurace karty je dalším krokem inicializace. Do inicializačního procesu přibyla pouze nutnost zakázat přerušení na lokální sběrnici karty. Důvod tohoto kroku je popsán dále.

3.8.3 Přenos převedených hodnot do paměti PC

Hlavním omezením ovladače z bakalářské práce byla nemožnost dosáhnout kratší periody hlavní smyčky než 200 μs. Původním předpokladem bylo, že toto omezení je způsobeno vyčítáním hodnot z FIFO fronty převedených vzorků pomocí přímého čtení z fronty a nikoliv použitím DMA přenosu. Proto jsem ovladač doplnil o funkce pro konfiguraci a využití integrovaného DMA řadiče. Problémem DMA je ale úzká návaznost na systém přerušení, což je pro vlastní použití problém, protože veškerá přerušení jsou RT jádrem zakázána[2]. DMA

[2]Režije spojená s obsluhou přerušení pře PIC se z důvodu zpětné kompatibility platformy x86 pohybuje v řádu 10 us.

převod je sice možné realizovat i bez přerušení, ale je nutné se neustále dotazovat na stav lokální sběrnice karty, kam jsou v případě zakázání přerušení od PCI přesměrována přerušení. Toto čekání může způsobit „zmeškání“ začátku nové synchronizační periody a tak rozpadu celého časování.

Jako řešení se nakonec ukázalo využít stejného čtení jako v původní verzi ovladače, pouze je nutné při inicializaci karty zakázat přerušení na místní sběrnici. Při vyčítání hodnot z FIFO fronty bez využití DMA se totiž z nejasného důvodu po vyčtení určitého množství vzorků nastavil stav indikující plné FIFO. Tento stav přerušil běh Pacer Clocku a zastavil tak další konverzi. Problém vyčítání byl již v době vzniku bakalářské práce konzultován s technickou podporou RTD, problém s přerušením však zmíněn nebyl.

3.8.4 Rozšířené možnosti synchronizace

Požadavek na rozvinutí synchronizačních možností ovladače zahrnuje především možnost synchronizovat jádro podle interních časovačů karty, ale poskytnout i možnosti pro synchronizaci externí. Obě možnosti již byly částečně zpracovány v původní verzi ovladače z bakalářské práce. Ten ale neumožňoval některé pokročilé možnosti spouštění časovaných událostí v průběhu periody RT jádra.

Například nebylo možné provést odečet hodnot z A/D převodníku jindy než na začátku periody. Předpokládaným řešením bylo využít pro časování takovýchto událostí dalších časovačů integrovaných na kartě. Obrázek 3.4 zachycuje možnosti vzájemného spouštění těchto časovačů. Výrobcem daná konfigurace bohužel neumožňuje cyklicky spouštět alespoň dva vhodné časovače navzájem, což by bylo pro toto řešení nutné.

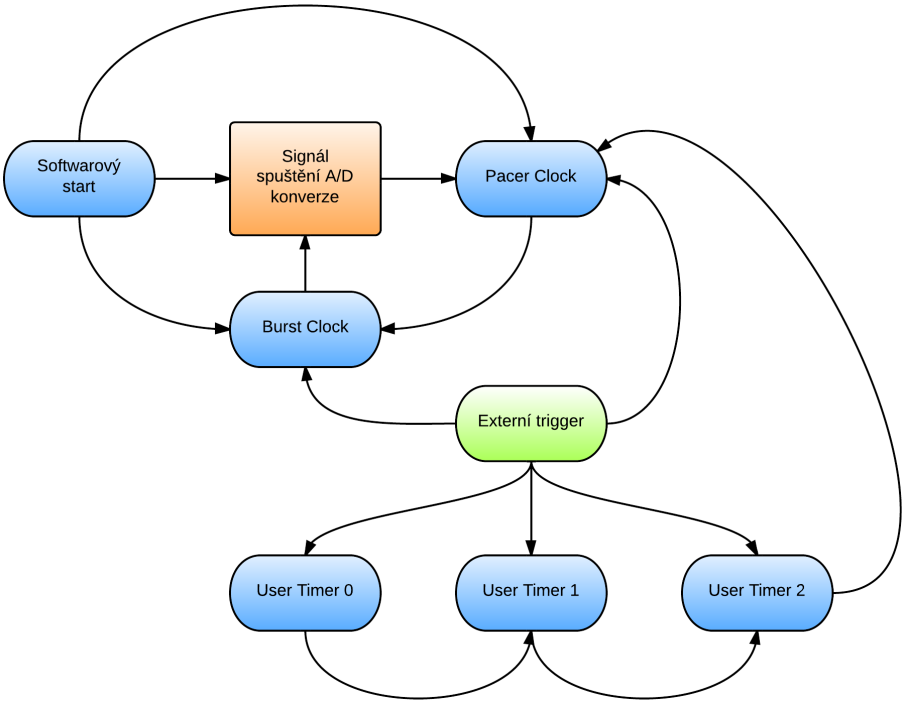
Hlavním problémem je nemožnost spouštět časovačem Pacer Clock, který určuje délku hlavní smyčky, žádný další vhodný časovač. Řešením použitým v ovladači je proto možnost nastavit násobnost čítače Pacer Clock. V praxi to znamená, že během jedné periody RT smyčky proběhne nastavený počet period Pacer Clocku. Na počátku každé této periody je pak možné sejmout vzorky z A/D převodníku. Je tak dosaženo možnosti získat data z převodníku vícekrát za periodu RT smyčky v závislosti na nastavení násobnosti.

Druhou možností synchronizace je synchronizace na externí synchronizační pulz. Tím je možné dosáhnout různé délky jednotlivých period RT jádra. Časovač Pacer Clock totiž nedovoluje tuto možnost proto, že není možné nahrát novou hodnotu časovače pro následující periodu. Takovýto zápis zastaví čítání Pacer Clocku, tím dojde k zastavení A/D převodů a nefunkčnosti celé synchronizace.

3.8.5 Ukládání naměřených dat s volitelnou podmínkou

Ukládat záznamy naměřených dat je vhodný způsob jak zachytit chování regulovaného systému v určitém časovém úseku a následně mít možnost toto chování zpracovat a zobrazit, například formou grafů. Toto zobrazení a další zpracování je úkolem aplikace provozované na PC a není tak součástí tohoto ovladače.

Je umožněno zaznamenávat hodnoty získané z A/D převodníku a to zaregistrováním požadovaných kanálů. Druhou možností, která by měla být poskytnuta, je možnost zaregistrovat k zaznamenávání hodnoty vypočtené v průběhu uživatelské funkce, například regulačního algoritmu. Tato možnost je založena na tom, že kompilátor při kompilaci výsledného



Obrázek 3.4: Možnosti vzájemného spouštění časovačů

programu vytvoří soubor `*.map`, který obsahuje offset adres všech funkcí a proměnných v programu vzhledem k začátku programu. Absolutní adresa platná za běhu programu se změní při nahrání programu do paměti. Bázovou adresu programu v paměti je pak nutno vypočítat z rozdílu dvou známých adres předem vybraných proměnných. Vyhledáním proměnné v souboru a přičtením jejího offsetu je pak možné zjistit její adresu v paměti v době běhu programu. Jediným problémem se jeví nemožnost získat z `*.map` souboru velikost jednotlivých proměnných. Ukázka obsahu souboru následuje.

```
Module: rtclock.obj(c:\diplomka\src\SDM8540\DRIVER\RTC\src\rtclock.c)
0001:0000120f+ update_in_progress_
0001:00001254+ get_rtc_reg_
0001:00001295  init_rtc_
0001:000012ab* print_rtc_
0001:000012ea+ read_rtc_
0001:00001462  get_rt_clock_
0002:00000f20+ _rtc_init_ok
0002:0000176c+ _year
0002:00001770+ _hour
0002:00001771+ _day
0002:00001772+ _min
0002:00001773+ _sec
0002:00001774+ _month
```

První řádek obsahuje informaci o souboru, pro který následuje mapa adres. Na dalších řádcích jsou uvedeny společně funkce i proměnná v zdrojovém souboru použité. Řádky začínající `0001` určují, že se jedná o název funkce. `0002` je začátek řádku s proměnnou, název proměnné je rovněž uvozen podtržítkem. Za dvojtečkou následuje offset adresy proměnné nebo funkce vzhledem k začátku programu. Znak `+` za adresou říká, že se jedná v případě funkce o inline funkci. Znak `*` značí nikdy nevolanou funkci a poslední možností, která zde není zobrazena, je znak `s` určující static funkci [18]. Na základě těchto informací je možné mapu zpracovat a vyhledat v ní použité proměnné.

Samotné zaznamenávání je navrženo jako kruhová fronta, ve které jsou hodnoty přepisovány až do doby splnění stop podmínky. Po jejím splnění je zaznamenán ještě určitý, uživatelem nastavený, počet hodnot a záznam je ukončen. Pak je doplněn o hlavičku obsahující časové informace a kontrolní součty a připraven k odeslání do PC.

3.8.6 Spolupráce s RT jádrem

Spolupráce ovladače s jádrem probíhá mírně odlišným způsobem oproti verzi vyvinuté v rámci bakalářské práce [6]. Hlavní rozdíl spočívá v tom, že jádro lze nyní po doplnění potřebných rozhraní distribuovat ve zkompilovaném stavu. Po kompilaci je archivováno do statické knihovny stejně jako ovladač I/O karty. Obě knihovny, jádro i ovladač musí být slinkovány do uživatelem definované aplikace. Po spuštění aplikace je provedeno vyhledání karty SDM8540. Pokud není karta nalezena, běží celá aplikace pouze v non-realtime režimu. Synchronizace probíhá podle interního 64 bitového časovače, který jádro při startu kalibruje pomocí Programmable Interval Timer (PIT). V případě nalezení karty jsou funkce pro synchronizaci jádra nahrazeny funkcemi poskytovanými ovladačem. Pokud uživatel nenastaví jinak, je zvolena defaultní konfigurace, kdy je délka periody nastavena na 200 μs a násobnost Pacer Clocku na 1. V této fáze je již možný přechod do RT režimu.

Před samotným spuštěním uživatel nastaví především parametry A/D převodníku a naprogramuje Channel Gain Table (CGT) popsanou dále. Při přechodu do RT režimu se spustí zdroj synchronizačních pulzů, při interním časování je jím Pacer Clock a podle něj je délka RT smyčky synchronizována. Samotné převody jsou realizovány A/D převodníkem bez zapojení procesoru a paralelně s nimi je tedy možné provádět další úlohy jádra. O toto přidělování se stará plánovač úloh v jádře. Při přechodu zpět do non-RT režimu je Pacer Clock zastaven, protože synchronozační pulz není potřeba. Kontrolní měření délky jednotlivých úloh je v tomto režimu realizováno samotným jádrem.

Kapitola 4

Realizace

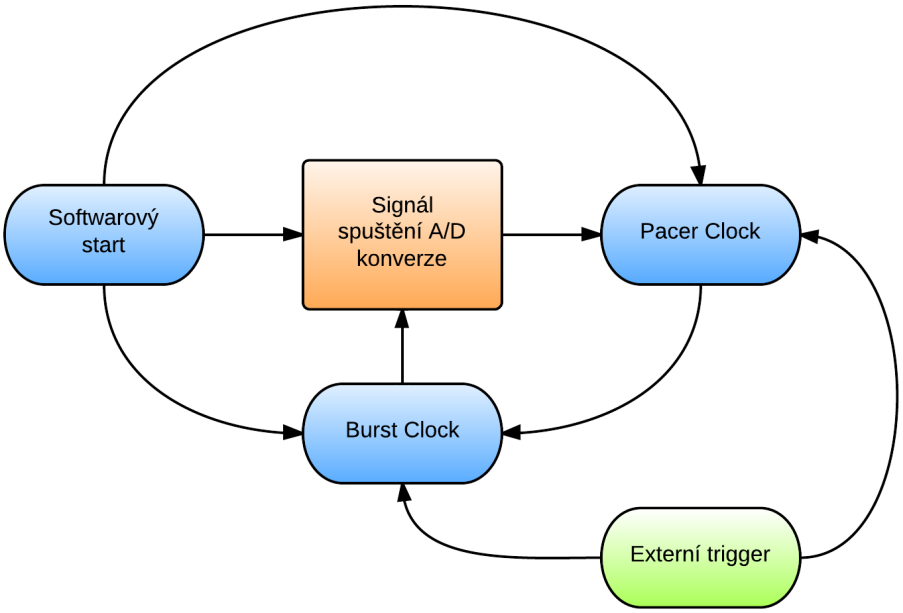
4.1 Synchronizace RT jádra

Synchronizace a správný časový sled událostí je nejdůležitějším prvkem v RT systému. Běh RT aplikace je typicky zajištěn hlavní programovou smyčkou dané délky, během které jsou vykonávány všechny funkce nutné pro samotný běh systému a rovněž výkonné funkce například definované uživatelem aplikace. V případě tohoto ovladače, a s ním spojeného RT jádra R. Martínka, se jedná o vyčtení hodnot z A/D převodníku, případně i několikrát za periodu tak, jak uživatel požaduje, přenos hodnot získaných z převodníku do paměti počítače, dále na straně jádra o servisní rutiny, například zajištění obsluhy zobrazení, čtení z klávesnice, komunikaci po sériové lince a další. Od uživatele do systému poté vstupuje uživatelská funkce systému, například regulační algoritmus v případě využití systému jako regulátoru. V případě hard real time aplikace jako je jádro, pro které je ovladač primárně určen, je nutné na straně jádra zajistit správné přiřazování úloh tak, aby byly v rámci jedné periody provedeny všechny nezbytné funkce a navíc ty, které jsou časově kritické ve správný okamžik. Všechny tyto události jsou synchronizovány v periodě RT systému na základě některého synchronizačního pulzu, který určuje začátek nové periody.

4.1.1 Interní zdroj synchronizačních pulzů

První možností synchronizace a časování je synchronizace na některý pulz (událost), kterou je možné generovat na samotné kartě SDM8540. Karta sama obsahuje množství čítačů a časovačů, které lze použít jako možné zdroje synchronizačních pulzů nebo pro spouštění dalších funkcí, jako například A/D konverze, ovládání výstupů, vzájemné spouštění časovačů a odložený start jiného časovače. Všechny tyto interní časovače jsou na kartě syntetizovány v kontrolním Electrically Programmable Logic Device (EPLD). Druhým typem jsou klasické obvody PIT Intel 8254. Ty jsou určené především pro ovládání digitálních vstupů a výstupů [11] [6].

Pro účely synchronizace na interní synchronizační pulz jsou důležité především časovače Pacer Clock a Burst Clock. Jejich funkčnost a použití je popsáno dále. Následující obrázek zachycuje vzájemné možnosti spouštění některých časovačů.



Obrázek 4.1: Možnosti vzájemného spouštění časovačů

4.1.1.1 Charakteristika časovače Pacer Clock

Časovač Pacer Clock je syntetizován v EPLD jako 32 bitový registr, pro samotné čítání je využito však jen 24 spodních bitů registru. Směr čítání je od nastavené hodnoty dolů k nule. Frekvence čítání je nastavitelná na 8 MHz nebo 20 MHz, pro reálné použití je nutné nastavit děličku, se kterou je možno dosáhnout maximální frekvence 1.25 MHz. Ta odpovídá délce periody 0.8μs, což je délka A/D převodu jednoho kanálu. Interním výstupním signálem čítače je *Pacer Clock signal*, který je během čítání ve stavu logické 1, při dosažení nuly je nastaven na logickou 0 [6].

Nastavení děličky se provádí podle následujícího vzorce:

j

$$dělička = (8(20)\ MHz/Požadovaná\ frekvence\ Pacer\ Clocku) - 1 \tag{4.1}$$

Pacer Clock může být spuštěn několika spouštěcími signály, kromě softwarového startu, kdy je čítač spuštěn zapsáním hodnoty na příslušnou adresu konfigurační paměti. Pro účely využití v aplikaci s RT jádrem jsou důležité způsoby spouštění zachyceny na obrázku 4.1. Jedná se především o již zmíněný softwarový start, který je vhodný provést, pokud je Pacer Clock hlavním synchronizačním signálem pro běh RT jádra. Spuštění Pacer Clocku se pak provede při přechodu aplikace do realtime režimu a aplikace je řízena tímto časovačem. Dále je možné spuštění externím triggerem. Zastavení Pacer Clocku může být provedeno podobně jako spouštění, především tedy softwarově při přechodu aplikace zpět do non-realtime módu.

Pacer Clock signál může být nastaven jako zdroj spouštění dalších časovačů a událostí. Pro naše využití je důležité především spouštění časovače Burst Clock, kterým se dále spouští A/D konverze. Události řízené signálem Pacer Clocku jsou poté různé analogové i digitální výstupy nebo přímo A/D konverze. [16]

Protože Pacer Clock signál není nijak vyveden externě, není možné snadno zjistit, kdy nastalo dosažení nuly, tedy také start nové periody. Pro tento účel poskytuje ovladač funkci, která při volání vyčte hodnotu Pacer Clocku, porovná ji s hodnotou uloženou při minulém čtení (na počátku je tato hodnota nastavena na maximální hodnotu čítače, tedy 2^{24}) a v případě, že nová hodnota je vyšší než hodnota stará, vrátí jedničku, čímž dá volající aplikaci zprávu o tom, že nová perioda začala. Na volající aplikaci je zajištění tohoto pollování s dostatečnou frekvencí, aby nebyl vyhodnocen průběh více period jako jedna perioda dlouhá [6].

4.1.1.2 Charakteristika časovače Burst Clock

Stejně jako časovač Pacer Clock, také Burst Clock je realizován 32 bitovým registrem syntetizovaným v EPLD, z něhož je využito pouze 16 spodních bitů. Čítač má menší rozsah než Pacer Clock především proto, že je výrobcem primárně určen k rychlému spouštění A/D převodů a nikoliv k čítání dlouho trvajících událostí jako Pacer Clock. Primární frekvence čítání je nastavitelná na 8 MHz nebo 20 MHz, pro reálné použití je nutné nastavit děličku, se kterou je možno dosáhnout maximální frekvence 1.25 MHz. Ta odpovídá délce periody 0.8μs, což je délka A/D převodu jednoho kanálu. Interním výstupním signálem čítače je *Burst Clock signal*, který je během čítání ve stavu logické 1, při dosažení nuly je nastaven na logickou 0 [6].

Nastavení děličky se provádí podle následujícího vzorce:

$$dělička = (8(20)\ MHz/Požadovaná\ frekvence\ Burst\ Clocku) - 1 \tag{4.2}$$

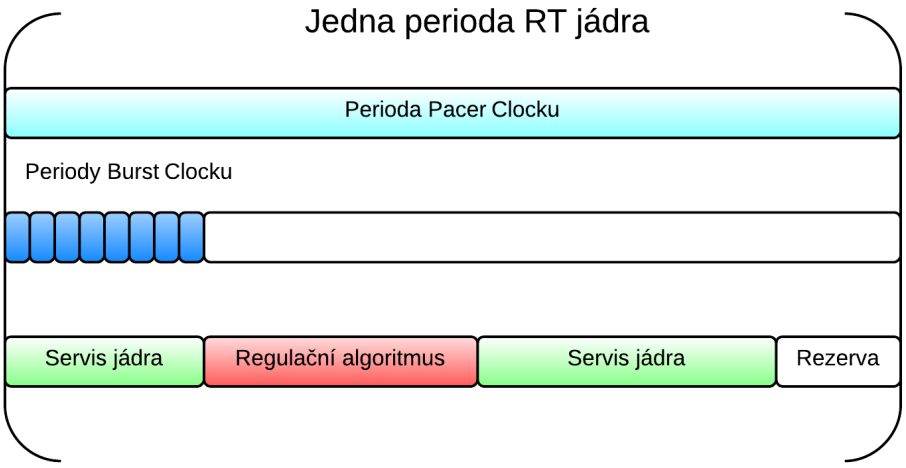
Burst Clock může být spouštěn kromě softwarového startu několika signály, například také softwarovým startem. Pro použití v ovladači je důležité především spouštění signálem *Pacer Clock signal*. Ten zajistí, že při začátku nové periody Pacer Clocku je spuštěn časovač Burst Clock. Druhou důležitou možností spouštění je externí trigger.

Jak předpokládá výrobce karty, Burst Clock je v ovladači použit jako primární zdroj spouštění A/D převodů. Vzhledem k požadavku uživatele převádět především více kanálů najednou by bylo optimální využití funkce Sample & Hold (S&H), která umožňuje sejmout vzorky ze všech kanálů A/D převodníku v jeden časový okamžik a poté je nezávisle na tom převést. Tuto možnost bohužel karta SDM8540 neposkytuje a pro její dosažení by bylo nutné zapojit do aplikace externí S&H modul. Tuto funkčnost lze alespoň přibližně napodobit co nejrychlejším sejmutím všech měřených kanálů za sebou. To se dá realizovat nastavením frekvence snímání na nejvyšší hodnotu, jakou limituje délka jednoho převodu. V naší aplikaci je tato hodnota 0.8μs a Burst Clock je tak nastaven na stejnou délku periody.

4.1.1.3 Možnosti interní synchronizace v původním ovladači

Ovladač vytvořený v rámci bakalářské práce umožňoval při využití interní synchronizace pouze jednoduché schéma, kdy jedna perioda Pacer Clocku představovala zároveň jednu periodu celého RT jádra. Nevýhodou tohoto řešení byla nemožnost provést během jedné periody více sad převodů hodnot z A/D převodníku. Spouštění převodů tak bylo omezeno na signál začátku nové periody, kdy byly převedeny všechny nadefinované kanály. Výhodou

bylo jednodušší přidělování úloh provedených v periodě ze strany RT jádra. Toto schéma zachycuje obrázek 4.2.



Obrázek 4.2: Synchronizace na jednu periodu Pacer Clocku

4.1.1.4 Možnosti interní synchronizace v nové verzi ovladače

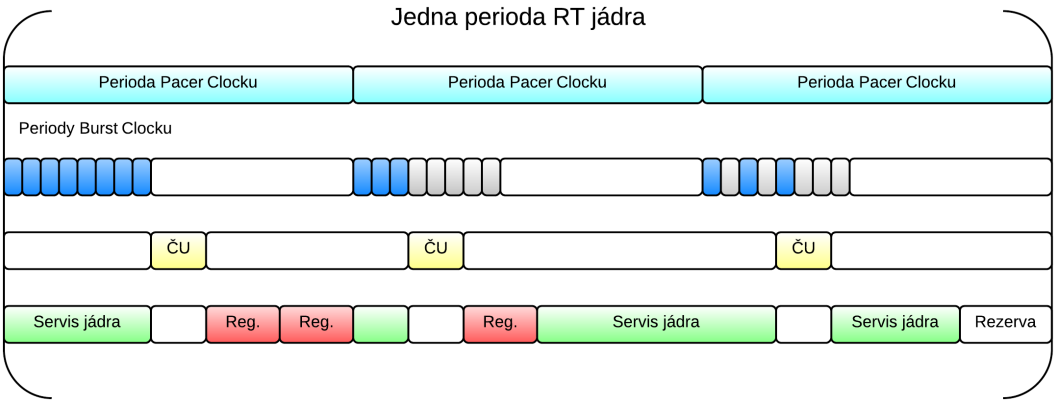
Potřeba přepracovat možností interní synchronizace vznikla především proto, že požadavkem uživatele aplikace byla možnost odečtení dalších hodnot z A/D převodníku během jedné periody RT smyčky. Původním záměrem jak této možnosti docílit bylo využití některých dalších časovačů dostupných na desce. Vzhledem k omezením daných konstrukcí desky není možné dosáhnout takové konfigurace, aby bylo vzájemné spouštění čítačů vhodným způsobem zřetězené. Například pomocí Pacer Clocku, který určuje délku hlavní periody, není možné spustit žádný další časovač, který by pak umožňoval provést jednotlivé konverze v průběhu periody.

Z tohoto důvodu bylo vybráno řešení realizovatelné opět pouze s využitím Pacer Clocku a Burst Clocku. Novým prvkem je možnost nadefinovat násobnost `n` periody Pacer Clocku v uživatelské aplikaci. V praxi tato možnost znamená, že jedna perioda Pacer Clocku nemusí určovat délku celé RT smyčky v případě, že je nastavena násobnost `n > 1`. Perioda Pacer Clocku tak proběhne během jedné periody RT smyčky `n`-krát. Toto schéma umožňuje ve spolupráci s CGT (viz kap. 4.1.2) spustit na počátku každé periody Paceru časovač Burst Clock, který spouští A/D převody nadefinované právě v CGT.

Je tak možno spustit v každé periodě Pacer Clock předem nadefinovaný počet převodů. Všechny takto převedené hodnoty jsou zaznamenány do A/D FIFO fronty, jejíž kapacita je dostačujících 1024 vzorků. Vyčteny jsou vždy po dokončení převodů v první n-tině periody. První sada měřených hodnot je důležitá pro samotný regulační algoritmus v dané periodě, ostatní hodnoty jsou využité pro výpočet dalších veličin a jsou vždy použity ty z předchozí RT periody.

Z důvodu tohoto rozdělení periody RT jádra na kratší periody interního časovače a přidání možnosti spouštět jednorázově přesně časované události, které mají nejvyšší prioritu,

bude nutné v případě krátké periody RT jádra rozdělit servisní funkce jádra i regulační algoritmus na několik kratších bloků tak, aby byly časy spouštění událostí s nejvyšší prioritou dodrženy. Toto je řešeno na straně jádra, rozdělení regulačního algoritmu pak na straně uživatele aplikace. Možnou konfiguraci Pacer Clocku s násobností `n = 3` i s událostmi naplánovanými na konkrétní časy (ČU) zachycuje obrázek 4.3.



Obrázek 4.3: Synchronizace s násobností Pacer Clocku n = 3

4.1.2 Channel Gain Table

Pokud požadujeme převod pouze jednoho kanálu, je nutné nastavit mód Channel Gain Latch (CGL), nastavit jeho parametry a nahrát je do konfigurační paměti. Při konfiguraci převáděného kanálu je nutné nastavit, o jaký kanál se jedná, jeho zesílení, rozsah a zda bude kanál snímán diferenciálně nebo proti referenční zemi. Po zapsání do paměti se tato konfigurace stane aktivní a při spuštění konverze je kanál s odpovídajícími parametry převeden.

V případě převodu více kanálů by bylo neefektivní a pomalé nastavovat každý převáděný kanál takovýmto způsobem. Karta pro tuto možnost poskytuje konfigurační prostředek nazvaný CGT. Jedná se o tabulku, která umožňuje nastavit jednotlivé řádky podobně jako CGL. Kanály mohou takto být naprogramovány v libovolném pořadí s rozdílným rozsahem a zesílením každého kanálu. Tabulka může obsahovat až 1024 řádků a lze tak naprogramovat požadované sekvence převodu kanálu včetně opakování.

Tabulku je nutné naprogramovat před spuštěním konverze, v případě využití s RT jádrem ještě před přechodem do realtime režimu. V ovladači je zápis realizován naplněním struktury typu `_ioctrl_t` a jejím následným zapsáním do paměti reprezentující CGT. Před tímto úkonem je nutné tabulku vymazat. Po dokončení programování tabulky je ukazatel postupu v tabulce nastaven na první řádek. Při spuštění A/D převodů je tento řádek převeden první a ukazatel sám iteruje přes další řádky. Po dosažení konce naprogramované tabulky je opět ukazatel nastaven na první řádek a CGT je prováděna znovu. V případě použití Burst Clocku je tato iterace iniciována právě signálem *Burst Clock signal*. Reset ukazatele je možné provést i softwarově v jakýkoliv okamžik [16].

4.1.2.1 Pause bit

Pause bit je 11. bitem v řádku CGT. Pokud je nastaven na hodnotu 1 a je povoleno zastavení konverze, A/D konverze se zastaví na tomto řádku tabulky a je spuštěna dalším předem nastaveným spouštěcím signálem. Této funkčnosti je možné využít, pokud tabulka obsahuje několik sekvencí převodů. V případě spouštění převodů pomocí Burst Clocku je nastavení tohoto bitu ignorováno.

4.1.2.2 Skip bit

Každému řádku je zde možnost nastavit *Skip bit* (14. bit v řádku), který zajistí, že daný řádek bude sice převeden, ale hodnota nebude zaznamenána ve frontě FIFO[1]. Je tak možné dosáhnout převodu více kanálů bez ukládání nechtěných hodnot.

4.1.2.3 Příklad použití CGT

Zdrojový kód 4.1 ukazuje příklad programování CGT.

```
#define BOARD 0
#define NUM_OF_CHAN 8
#define FIRST_CHANNEL 0

ioctrl_t row;
int i;

SDM8540_clear_cgt(BOARD);

SDM8540_set_channel_count(NUM_OF_CHAN);

for (i = 0; i < NUM_OF_CHAN; i++)
{
        SDM8540_setup_cgt_row(&row, FIRST_CHANNEL+i, GAIN1,
          AIN_BIP10, NRSE_AGND, _GND_SE, CGT_PAUSE_DISABLE,
          0, 0, 0);
        SDM8540_write_cgt_row_analog(BOARD, *((uint16_t *) & row));
}
```

Zdrojový kód 4.1: Ukázka nastavení CGT

Příklad ukazuje konfiguraci CGT pro převod 8 kanálů. Konfigurace všech kanálů je shodná, parametry jsou následující:

- **GAIN1** vypíná zesílání kanálu
- **AIN_BIP10** nastavuje rozsah kanálu na -10V - +10V
- **NRSE_AGND** připojení referenční země

[1] First In, First Out

- **_GND_DIFF** nastavuje kanál jako diferenciální
- **CGT_PAUSE_DISABLE** nastavuje *pause bit* na 0
- **0** říká, že D/A1 nebude použito pro výstup
- **0** říká, že D/A2 nebude použito pro výstup
- **0** nastavuje *skip bit*, data budou uložena

Tento příklad ukazuje nastavení všech kanálů stejným způsobem. Je samozřejmě možné vytvářet sekvence s různým nastavením kanálů - různým zesílením, rozsahem. Je možné naprogramovat do tabulky několik sekvencí oddělených řádky s nastaveným *pause bit*em, který oddělí jejich provedení.

4.1.3 Externí zdroj synchronizačních pulzů

Druhou možností synchronizace karty, a také celé RT aplikace, je synchronizace podle externího zdroje synchronizačních pulzů. Externím triggerem je možné spouštět všechny na desce integrované časovače, stejně jako A/D konverzi. Rovněž je možné tímto signálem v závislosti na konfiguraci běh časovačů zastavit. Jako parametr spouštění externím triggerem je možné nastavit polaritu, která určuje, zda trigger spustí na vzestupné nebo sestupné hraně externího pulzu. Pulz by měl mít trvání alespoň 100 ns [16].

4.1.3.1 Externí spouštění časovače Pacer Clock

Pacer Clock může být při odpovídajícím nastavení spuštěn externím signálem. Karta nabízí dvě možnosti využití externího spouštění. První možností je *External Trigger Standard Mode*, kdy je Pacer Clock spuštěn jedním pulzem na vstupu externího triggeru. Zastavení pak může být provedeno opět externím signálem. Druhou možností je *External Trigger Gated Mode*, při kterém Pacer Clock běží po dobu, kdy je detekována buď logická 1, nebo 0, v závislosti na nastavení polarity, na vstupu externího triggeru. Při změně polarity se Pacer Clock zastaví.

4.1.3.2 Externí časovač Pacer Clock

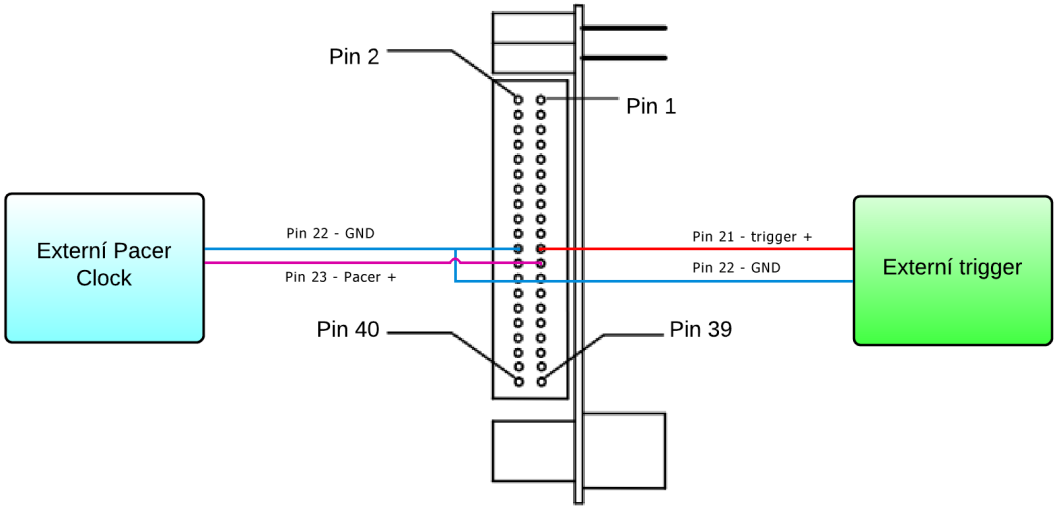
Rovněž je možné interní Pacer Clock nahradit externím signálem. Toto řešení není z pohledu ostatních komponent karty od interního Pacer Clocku rozeznatelné, protože externí zdroj časování generuje na kartě interní signál *Pacer Clock signal* a vše se chová tak, jako kdyby byl generován interním Pacer Clockem. Připojení externího signálu Pacer Clock zobrazuje obrázek 4.4.

4.1.3.3 Externí spouštění časovače Burst Clock

Časovač Burst Clock je rovněž možné spouštět signálem externího triggeru. V tomto případě pak externí zdroj signálu nahrazuje signál spouštící Burst Clock interně generovaný Pacer Clockem. Výhodou tohoto řešení je možnost ovlivnit délku každé periody nastavením zdroje pulzů pro externí trigger. Po přijetí signálu je spuštěn Burst Clock a v závislosti na nastavení v CGT je provedena konverze sekvence nadefinovaných kanálů [16].

4.1.3.4 Připojení externího synchronizačního signálu

V závislosti na softwarovém nastavení polarity (funkce popsána v oddílu 4.5) reaguje externí trigger na vzestupnou nebo sestupnou hranu externího signálu, jak již bylo popsáno výše. Délka pulzu by neměla pro korektní rozpoznání klesnout pod 100 ns. Připojení externího triggeru ke kartě je možné na univerzálním konektoru CN6 tak, jak zobrazuje obrázek 4.4.



Obrázek 4.4: Připojení externího triggeru

4.2 Hodiny reálného času

Hodiny reálného času, Real-Time Clock (RTC), jsou v ovladači využity ve funkcích pro záznam naměřených hodnot. Každý záznam obsahuje hlavičku s několika údaji pro zajištění konzistence tohoto záznamu. Mimo jiné obsahuje hlavička i časy počátku a konce záznamu. Tyto hodnoty jsou získány právě z RTC. Důležitou podmínkou pro možnost zaznamenávání dat je správné nastavení hodin v PC a připojení záložního napájení k CPU modulu RTD CML147786CX650HR-128.

4.2.1 Charakteristika obvodu

Hodiny reálného času, RTC, jsou obvod umístěný na základní desce, jehož úkolem je udržovat informaci o aktuálním čase. Obvykle je RTC součástí stejného čipu jako Complementary Metal–Oxide–Semiconductor (CMOS) paměť. Jedná se o statickou paměť, která obsahuje údaje o hardwarové konfiguraci počítače. Obvod RTC je obvykle napájen z baterie nebo superkapacitoru a čas běží, i když je PC vypnuté. Většina hodin využívá k měření času krystalového oscilátoru. Ten kmitá typicky na frekvenci 32,768 kHz. Přesnost takto měřeného času se odvíjí především od přesnosti použitého krystalu. Druhým způsobem měření času je odvození ze síťové frekvence 50Hz, ale tento způsob je výrazně méně přesný, protože

frekvence 50 Hz není v síti přesně dodržována. Samotné čítání hodin je zajištěno binárním čítačem připojeným ke zdroji kmitů [12].

4.2.2 Získání času z RTC

K obsahu CMOS paměti, a též hodnotě hodin reálného času, je možné přistupovat prostřednictvím I/O portů `0x70` a `0x71`. K paměti je přistupováno po bytech a každý byte je samostatně adresován. Každá adresa je typicky nazývána registrem. Prvních 14 registrů je určeno ke čtení a ovládání RTC. Pro potřeby této práce jsou důležité především tyto registry:

Registr	Obsah
0x00	vteřiny
0x02	minuty
0x04	hodiny
0x07	den v měsíci
0x08	měsíc
0x09	rok
0x32	století
0x4A	Extended Control registr
0x0A	Status registr A

Tabulka 4.1: Porty RTC

4.2.2.1 Vyčtení hodnot registrů

Vyčtení hodnot probíhá prostřednictvím zápisu adresy požadovaného registru na I/O port `0x70` a následným přečtením hodnoty na portu `0x71`. V případě použití kompilátoru Open Watcom je toto realizováno následujícím způsobem:

```
#include <conio.h>

#define CMOS_ADDR 0x70
#define CMOS_DATA 0x71

uint8_t get_rtc_reg(int reg)
{
        outp(CMOS_ADDR, reg);
        return inp(CMOS_DATA);
}
```

Zdrojový kód 4.2: Čtení z I/O portu

Přečtené hodnoty jsou uloženy ve formě Binary Coded Decimal (BCD), binárně reprezentované dekadické číslo, kde každá čtveřice bitů představuje číslici 0 - 9. Hodnoty 1001b - 1111b jsou nevyužity. Vypočtení dekadické hodnoty je provedeno následujícím způsobem:

```
uint8_t sec = get_rtc_reg(0x00);
sec = (sec & 0x0f) + ((sec >> 4) * 10);
```

Zdrojový kód 4.3: Přepočet BCD na dekadický tvar

4.2.2.2 Update In Progress

Update In Progress je stav, kdy dochází k aktualizaci registru udržujícího hodnotu RTC. Stav je indikován bitem 7 ve Status registru A nastaveným na 1 po dobu trvání aktualizace hodin. Nastává jednou za vteřinu ve chvíli, kdy je inkrementován registr vteřin a je nutné zkontrolovat, zda není nutné zvýšit hodnotu minut a dalších registrů vyšších řádů. Během tohoto stavu by neměly být hodiny vyčítány, aby nedošlo k získání nekonzistentní hodnoty času. V případě čtení hodin bez ohledu na tento stav by mohly být například v 9:00 přečteny hodnoty 8:59, 8:60, 8:00 nebo 9:00 [13].

4.2.2.3 Přítomnost záložního napájení hodin

Vzhledem k tomu, že baterie pro napájení RTC nemusí být vždy přítomna, jako je tomu v defaultní konfiguraci CPU modulu RTD CML147786CX650HR-128, je nutné po startu systému tento stav ověřit a zajistit tak, aby nedocházelo k vyčítání nesprávných hodnot času. K tomuto účelu je možné použít bit 7 v Extended Control registru, který indikuje stav baterie.

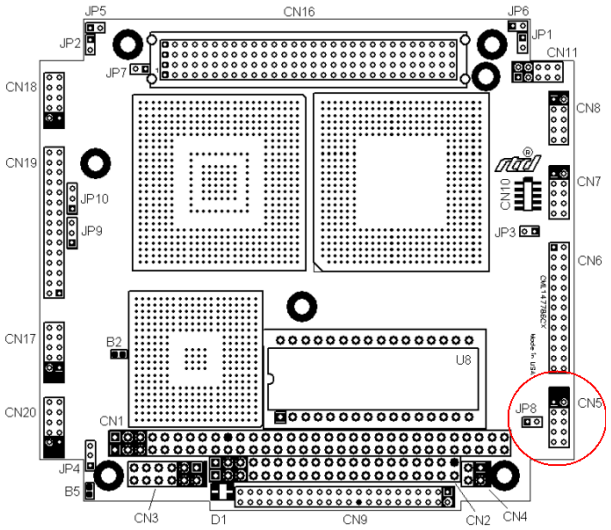
4.2.3 RTC na PC modulu RTD CML147786CX650HR-128

Jak již bylo výše zmíněno, PC modul RTD CML147786CX650HR-128 neobsahuje standardně záložní baterii pro napájení paměti CMOS. Proto je nutné pro použití funkcí pro zaznamenávání naměřených hodnot, které využívají čtení hodnoty RTC, připojit baterii na konektor JP8 (pin 1 - Vbatt, pin 2 - GND), případně na multifunkční konektor CN5 (pin 9 - Vbatt, pin 7 - GND) [17].

4.3 Ukládání sad záznamů měřených dat

Novým modulem ovladače vzniklého v mé bakalářské práci [6] je modul pro ukládání sad záznamů měřených dat. Tento modul nabízí uživateli možnost nakonfigurovat, které hodnoty budou do záznamu zahrnuty a jakým způsobem bude provedeno spuštění a zastavení záznamu. Zaznamenávanými hodnotami mohou být především hodnoty získané z A/D převodníku, v budoucnu také hodnoty vypočítané v průběhu uživatelské funkce. Všechny funkce pro záznam hodnot jsou rovněž obsaženy v ovladači karty, jsou však uvozeny prefixem LOG_ pro odlišení od ostatních funkcí karty.

Zaznamenávání může být spuštěno voláním příslušné funkce pro spuštění záznamu kdekoliv v uživatelem definované funkci. Zastaveno může být rovněž buď voláním odpovídající funkce, nebo splněním předem nastavené ukončovací podmínky. Tato podmínka je typicky nastavena jako práh pro některou hodnotu získanou z A/D převodníku, může být ale také



Obrázek 4.5: Rozložení konektorů na CPU modulu CML147786CX650HR-128

dána například počtem proběhlých period nebo sejmutých vzorků. Ovladač pro tyto možnosti poskytuje rozhraní a je na uživateli, jakým způsobem zaznamenávání nakonfiguruje.

Nastavitelná je rovněž možnost zaznamenávat určité množství dat i po splnění ukončovací podmínky. Toto chování je vhodné zejména pro zachycení změn sledované veličiny kolem ukončovací podmínky z pohledu časové osy z obou stran.

Po ukončení záznamu je tento opatřen hlavičkou informující o velikosti záznamu, která rovněž obsahuje údaje pro kontrolu integrity dat, jak je popsáno níže. Záznam je takto připraven pro přenos do PC, například prostřednictvím sériové linky, k dalšímu zpracování.

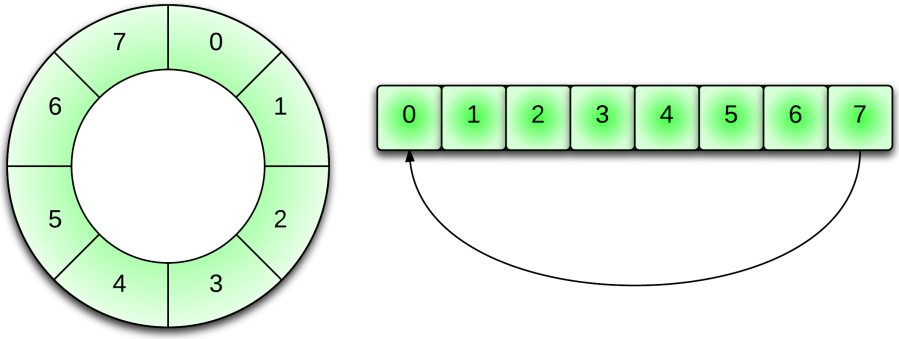
4.3.1 Kruhová fronta

Kruhová fronta, nebo také kruhový buffer, je datová struktura pevně dané velikosti, která se navenek jeví, jako kdyby začátek plynule navazoval na konec. V praxi se tato vlastnost projevuje tak, že je možné do kruhové fronty zapisovat neomezené množství hodnot. Pokud je fronta zaplněna, nejstarší hodnoty se začnou přepisovat hodnotami nejnovějšími [5]. Tento cyklus se neustále opakuje až do skončení zápisu do fronty. Schématické znázornění kruhové fronty je na obrázku 4.6.

Pro použití této datové struktury v modulu pro záznam dat mluví jednoznačně právě možnost data neustále přepisovat. Konec zápisu hodnot je totiž typicky určen předem definovanou událostí a je důležité zachytit sledované hodnoty časově rozložené kolem této události. Čím jsou tedy data starší (a tím a více vzdáleny od ukončovací události), tím jsou zpravidla méně důležitá pro informativní hodnotu vlastního záznamu.

4.3.1.1 Realizace kruhové fronty

Pro účely tohoto ovladače byla kruhová fronta implementována formou pole, jehož velikost musí být mocninou čísla 2. Toto omezení je zde proto, aby nebylo nutné používat



Obrázek 4.6: Schematické znázornění kruhové fronty

operaci modulo pro výpočet počátku a konce fronty při čtení a zápisu. Toto je zde realizováno pouze bitovými operacemi, které jsou možné díky vlastnostem dvojkového doplňku. Proto například výpočet konce fronty, který by byl jinak realizován jako

$$end = (end + 1)\%SIZE, \tag{4.3}$$

je tímto způsobem možno zjistit jako

$$end = end\&(SIZE - 1) \tag{4.4}$$

Operace modulo je výpočetně poměrně náročná a nutnost použít jako velikost fronty mocninu dvou je ušetřeným výpočetním časem více než vyvážena.

Vlastní kruhová fronta je realizována strukturou 4.4. Atribut `start` určuje začátek fronty, atribut `end` pak její konec. Samotný obsah je realizován polem typu `cb_element_t` definované velikosti `SIZE`.

```
typedef struct
{
        int start;
        int end;
        cb_element_t elements [SIZE];
} cbLog;
```

Zdrojový kód 4.4: Struktura kruhové fronty

Typ `cb_element_t` je realizován strukturou, jejíž obsah může být doplněn v závislosti na uživatelem požadovaném formátu a typu ukládaných dat. Obecně obsahuje struktura název a hodnotu ukládané proměnné. Předefinována může být v souboru `cb.h`.

Pro kruhovou frontu jsou dále implementovány standardní funkce pro vložení a vyzvednutí dat z fronty a dále funkce pro zjištění, zda je fronta prázdná a plná.

4.3.2 Hlavička záznamu

Hlavička záznamu je vygenerována po skončení zaznamenávání aktuální sady dat. Obsahuje informace o čase zaznamenání, velikosti logu a kontrolní součty pro ověření integrity

záznamu. Strukturu hlavičky ukazuje zdrojový kód 4.5 a jednotlivé položky jsou dále popsány.

```
typedef struct
{
        int log_number;
        rt_clock_t logging_started;
        rt_clock_t logging_ended;
        int data_size;
        uint32_t previous_header_hash;
        uint32_t log_crc;
} log_header;
```

Zdrojový kód 4.5: Hlavička záznamu

4.3.2.1 Pořadové číslo záznamu

Jedná se o v ovladači inkrementované pořadové číslo záznamu. Po startu aplikace začíná nulou a je po jedné zvyšováno. Může sloužit pro základní ověření správné posloupnosti přijatých dat v případě přenosu záznamu do PC, například prostřednictvím sériové linky.

4.3.2.2 Časové odečty z RTC

Při spuštění i ukončení záznamu je proveden odečet hodnoty hodin reálného času způsobem popsaným v oddílu 4.2. Tato hodnota je nesena strukturou typu rt_clock_t, jejíž formát ukazuje zdrojový kód 4.6.

```
typedef struct
{
        bool valid;
        uint8_t sec;
        uint8_t min;
        uint8_t hour;
        uint8_t day;
        uint8_t month;
        int year;
} rt_clock_t;
```

Zdrojový kód 4.6: Struktura s časovým záznamem z RTC

Kvůli zaznamenání těchto hodnot je nutné, aby byla k CPU modulu připojena záložní baterie a hodiny byly správně nastaveny v SETUPu. Kontrola přítomnosti baterie se provádí při inicializaci ovladače hodin. Tato kontrola může být vypnuta definováním makra `_CHECK_BATTERY` v souboru `rtclock.c`. Příznak `valid` ve struktuře je v případě vypnutí kontroly baterie bez užitku, jinak by měl být aplikací využívající odečtený čas kontrolován při použití časového údaje.

4.3.2.3 Zabezpečení integrity dat

Integrita dat a správné pořadí čtení záznamů je zajištěno několika způsoby. Nejjednodušší kontrolou správného pořadí záznamů je výše popsané ovladačem přidělené interní číslo záznamu. Dalším prvkem zajišťujícím kontrolu pořadí je hash hlavičky předchozího záznamu realizovaný kontrolním součtem Cyclic Redundancy Check (CRC32). V případě, že se jedná o první záznam, je tato hodnota nulová.

Kontrolní součet je 32-bitový (CRC32), z důvodu rychlosti vypočítán metodou *byte-by-byte*, tedy po bajtech, nikoliv po bitech. Jako generující polynom je zvolena hodnota `0xffffffff`. Je rovněž využita vyhledávací tabulka s předem vypočítanými zbytky, opět z důvodu rychlosti [2].

Dalším prvkem je v hlavičce zahrnutá velikost dat `data_size`. Tento atribut nese velikost zaznamenaných dat v kruhové frontě. Po přenesení záznamu do PC je tak možno zkontrolovat, zda přijatá data odpovídají velikostí datům odeslaným. Poslední položkou v hlavičce je kontrolní součet CRC32 samotného záznamu, tj. obsahu kruhové fronty. Kontrolní součet zaznamenaných dat tak může být snadno vypočítán příjemcem dat a porovnáním kontrolních součtů pak ukáže případné poškození dat během přenosu.

4.3.3 Záznam hodnot z A/D převodníku

Pro záznam hodnot z A/D převodníku je nutné, stejně jako pro záznam hodnot z uživatelské funkce, zavolat inicializaci zaznamenávání. Ta ověří, zda je dostupná hodnota hodin reálného času a připraví prostředí pro záznam. Dále je nutné nastavit, které hodnoty získané z A/D převodníku budou zaznamenávány. Počet těchto hodnot záleží na nastavení násobnosti periody Pacer Clocku `n` a leží v intervalu `0..n * počet kanálů`. Na základě tohoto údaje je vytvořeno pole příznaků, které určuje, které hodnoty z A/D převodníku, spíše tedy již převedené hodnoty z FIFO fronty, mají být zapsány do záznamu.

S každou hodnotou je zaznamenán i kanál, ze kterého byla hodnota získána. Samotné zaznamenávání znamená pouze zkopírování hodnoty do struktruy nesoucí hodnotu proměnné a informaci o ní a tato struktura je vložena do kruhové fronty záznamu.

4.3.4 Záznam hodnot z uživatelské funkce

Možnost zaznamenávat vypočtené hodnoty z uživatelské funkce je poněkud složitější než odečet a záznam hodnot z A/D převodníku. Prvním problémem je, jak vlastně získat přístup k proměnným, které uživatel nadefinuje v jiném modulu aplikace než je samotný ovladač. Tento samotný přístup částečně zajišťuje jádro způsobem popsaným dále.

4.3.4.1 Mapa funkcí a proměnných

Při kompilaci výsledného programu vytvoří kompilátor Open Watcomu soubor `*.map`, který obsahuje offset adres všech funkcí a proměnných v programu vzhledem k začátku programu. Tento soubor je nadále předán linkeru, který s ním dále pracuje. Absolutní adresa

[2] Algoritmus vychází z open source kódu http://www.opensource.apple.com/source/xnu/xnu-1456.1.26/bsd/libkern/crc32.c

proměnné nebo funkce platná za běhu programu se však změní při nahrání programu do paměti. Výpočet skutečné adresy zajišťuje samotné jádro porovnáním rozdílu dvou relativních adres ze souboru a skutečných adres platných při běhu programu.

Ideou tohoto řešení je tedy získat v programu tento soubor, což je již zapracováno v jádře R. Martínka tak, že při vlastní kompilaci jádra je soubor načten jako pole bytů a zakompilován do samotné `*.exe` aplikace. Prostřednictvím funkce je pak poskytnuta adresa tohoto pole v paměti a jeho velikost. Pro získání adres samotných proměnných v uživatelské funkci je nutné projít tento „soubor“ a vyhledat je v odpovídající části.

Každý soubor, ze kterého je aplikace zkompilována, má v `*.map` souboru zastoupení v odpovídající části uvozené slovem `Module:`. Například `Module: rtclock.obj(c:\diplomka\src\SDM8540\DRIVER\RTC\src\rtclock.c)` uvozuje zdrojový soubor `rtclock.c`. Následuje seznam všech funkcí a proměnných, jak je popsáno i s příkladem v analýze 3.8.5.

Pro účely získání adres a názvů proměnných jsou důležité pouze řádky uvozené `0002`. Dalším znakem proměnné je uvození jejího jména podtržítkem `_`. Následuje offset proměnné vzhledem k začátku programu. Adresu začátku programu rovněž poskytne jádro, je tak již snadné získat reálnou adresu proměnné. Získané proměnné je pak potřeba uložit do pole struktur obsahujících název proměnné a právě vypočtenou adresu a toto pole setřídit podle názvu proměnné. Všechny tyto úkony jsou provedeny při inicializaci jádra, případně ovladače a nijak tedy nezasahují do realtime režimu.

V případě, že bude chtít uživatel přidat proměnnou do zaznamenávání, zavolá poskytnutou funkci s parametrem jména proměnné. Setříděné pole je prohledáno pomocí binárního půlení, což je v případě setříděného pole záležitost rychlá, provedená se složitostí $O(logn)$. Obsah adresy ze struktury nalezené v poli je pak zaznamenáván do kruhového bufferu záznamu stejně jako hodnoty získané z A/D převodníku.

Problémem tohoto řešení je fakt, že mapa proměnných obsahuje sice názvy a adresy, ale neobsahuje informaci o typu nebo velikosti proměnných. Nelze tak rozlišit, zda se jedná o jednoduchou proměnnou či pole hodnot, zda je typu `int`, `float` či jiného. To může být zásadní problém při čtení, kdy by mohlo při hádání typu dojít k narušení integrity paměti, případně k vyčtení jen části hodnoty. Řešení tohoto problému je dále rozebráno v závěru (viz kap. 6).

4.3.5 Postup vlastního záznamu

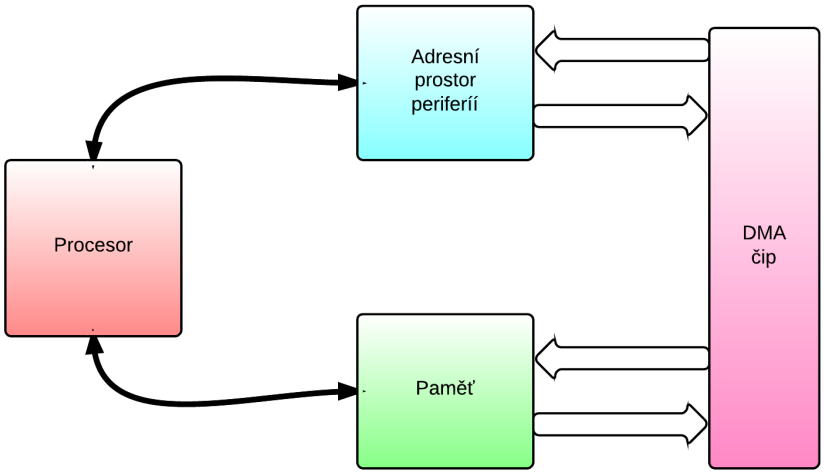
Vlastní použití záznamu probíhá podle následujícího schématu.

- Inicializace modulu zaznamenávání.
- Nastavení ukončovací podmínky záznamu.
- Volitelné nastavení spouštěcí podmínky záznamu, záznam může být jinak spuštěn stiskem klávesy.
- Přidání požadovaných kanálů A/D převodníku do zaznamenávání.
- Přidání požadovaných uživatelských proměnných do zaznamenávání.
- Spuštění záznamu definovanou podmínkou, případně stiskem klávesy.

- Ukončení záznamu definovanou podmínkou.
- Vytvoření hlavičky záznamu a příprava na přenos do PC.
- Případné zaznamenání do souboru při ukončení aplikace.

4.4 DMA

DMA - přímý přístup do paměti je způsob, jak přistupovat do paměti počítače bez využití procesoru. Tuto možnost obvykle obstarává samostatný čip, v případě karty SDM8540 je zahrnut v PCI9080 řadiči PCI sběrnice. Tento přístup se vyplatí zejména ve chvíli, kdy velké množství datových přenosů zabírá procesorový čas a blokuje tak další výpočetní operace. DMA umožňuje přenášet data přímo mezi I/O adresním prostorem, kde jsou typicky získaná z periferií, a hlavní pamětí [11].



Obrázek 4.7: Princip DMA

4.4.1 DMA na kartě SDM8540

DMA na kartě SDM8540 podporuje dva nezávislé obousměrné DMA kanály. Každý kanál má vlastní programovatelný DMA řadič a FIFO na přenášená data. Oba kanály podporují *Chaining Mode* i *Non-Chainig Mode*. Přístup prostřednictvím DMA je umožněn k adresnímu prostoru *LAS1*, který obsahuje vstupní a výstupní FIFO pro periferie karty. Mohou tak být čteny a zapisovány bez přímé intervence procesoru.

Karta podporuje tři základní módy DMA:

- **Single Mode**, ten umožňuje přenést jednu hodnotu během jednoho DMA požadavku
- **Block Mode** přenese, jak je z názvu patrné, celý blok dat daný hodnotou *Transfer Count registru*
- **Demand Mode** přenáší data kontinuálně, dokud není zastaven

4.4.1.1 Non-Chaining Mode

V tomto režimu je v DMA řadiči nastaven zdroj dat - typicky adresa FIFO v LAS1 adresním prostoru, cílová adresa, kam budou data přenášena, počet hodnot a směr přenosu. V případě spuštění tohoto přenosu pak DMA řadič ze zadané adresy kopíruje hodnoty na cílovou adresu. Je možné nastavit automatickou inkrementaci adresy v cílovém prostoru, zdrojová je však stále stejná. Hodí se tak k vyčítání hodnot z FIFO fronty, kde je při přečtení zajištěno odebrání hodnoty a na její adrese je hodnota další. Po skončení přenosu je ve stavovém registru nastaven *Channel done bit* a na jeho základě může být generováno přerušení.

4.4.1.2 Chaining Mode

Režim Chainig Mode umožňuje, na rozdíl od předchozího, číst data z většího rozsahu zdrojové paměti. Nastavení je podobné jako u Non-Chainig Mode, zahrnuje zdrojovou a cílovou adresu, směr přenosu, velikost datového bloku a navíc adresu dalšího datového bloku. Po zahájení přenosu je přenesen datový blok daný nastavením a v případě, že je nastavena adresa dalšího bloku, přenos pokračuje dalším blokem, dokud není v příkazovém registru nastaven příznak *End of Chain*.

4.4.2 Implementace DMA

Kvůli dřívějšímu předpokladu využít DMA na přenos hodnot získaných z A/D převodníku obsahuje ovladač implementaci funkcí pro ovládání kanálu 0 DMA řadiče. Kvůli úzkému spojení DMA s přerušením (generování přerušení při skončení převodu) nakonec není DMA s aktuální verzí jádra využit, protože veškerá přerušení jsou v aplikaci zakázána. Neefektivní pollování stavového registru DMA pak nepřináší žádné výhody oproti standardnímu vyčítání hodnot z FIFO fronty. V případě změny chování aplikace a využití přerušení je možné funkce DMA se stávajícím ovladačem využít.

4.5 Popis nových funkcí ovladače

Zde je uveden přehled funkcí upravených oproti předchozí verzi nebo nově implementovaných v ovladači karty SDM8540. Stručný popis všech funkcí lze nalézt v hlavičkovém souboru *SDM8540LIB.h*. Všechny funkce mají prefix SDM8540_ pro lepší odlišení ve zdrojovém kódu volající aplikace. Parametr všech funkcí nazvaný *device_number* určuje pořadí připojené karty na PCI sběrnici. V případě systému s jednou kartou je to vždy 0 a proto nebude význam tohoto parametru u funkce uvedených níže již znovu popisován.

4.5.1 Inicializační funkce karty

`SDM8540_init_board(int device_number)`

Funkce vyhledá kartu na PCI sběrnici, přečte přidělené bázové adresy z konfigurační oblasti a uloží je do proměnných v ovladači. Ty pak slouží pro přístup ke konfiguračním možnostem karty. Tato funkce se svojí implementací nijak nezměnila oproti předchozí verzi,

je zde uvedena pro úplnost jako důležitá funkce, kterou je nutné vykonat před voláním jakékoliv další funkce poskytnuté ovladačem. 1 v případě úspěšné inicializace karty, jinak 0.
Návratová hodnota: `int`

`SDM8540_clear_board(int device_number)`

Funkce zavolá postupně několik dalších funkcí, které slouží pro softwarový reset karty, vymazání A/D FIFO fronty, D/A FIFO fronty, **DIO!** FIFO fronty, vymazání CGT a masky přerušení. Jednotlivé funkce je možné zavolat samostatně, jejich názvy a popis je v *SDM8540LIB.h* hlavičkovém souboru.
Návratová hodnota: `void`

`SDM8540_set_it_mask(int board_number, int mask)`

Funkce nastavuje masku přerušení generovaných na kartě. Ovlivňuje registr `Interrupt Status/Mask Register`. Lze odmaskovat libovolná přerušení popsaná v manuálu karty [16]. Pro využití s RT jádrem je nutné nastavit masku 0 a vypnout tak všechna přerušení generovaná kartou.
Návratová hodnota: `void`
Parametry: mask - při použití přerušení podle manuálu, v aplikaci s RT jádrem vždy 0.

4.5.2 Synchronizační funkce

`SDM8540_set_pacer_multiplicity(int board_number, int mult)`

Funkce nastavuje násobnost Pacer Clocku. Vzhledem k novým synchronizačním možnostem je možné nastavit, aby jednu periodu RT jádra tvořilo více period Pacer Clocku. Pokud není zavolána, je implicitně nastavena násobnost na 1.
Návratová hodnota: `void`
Parametry: mult - násobnost periody Pacer Clocku v jedné RT periodě. Není nutné ji volat, implicitně 1.

`SDM8540_get_pacer_multiplicity(int board_number)`

Tato funkce vrací násobnost Pacer Clocku, která může být nastavena předchozí funkcí.
Návratová hodnota: `int`

`SDM8540_get_actual_pacer_period(int board_number)`

Funkce vrací hodnotu určující aktuální periodu Pacer Clocku v rámci jedné RT smyčky. Pro n-tou periodu vrací hodnotu n-1.
Návratová hodnota: `int`

`SDM8540_is_new_pacer_period(int board_number)`

Funkce vrací hodnotu 1 v případě, že začala nová perioda Pacer Clocku. Může být využita při násobnosti Pacer Clocku nastavené na 1, případně pro účely lazení aplikace, není vhodná pro využití při jiné násobnosti periody Paceru.
Návratová hodnota: `int`

`SDM8540_is_new_period(int board_number)`

Funkce se chová z hlediska RT jádra stejně jako funkce stejného jména v předchozí verzi ovladače. Vrací hodnotu 1 v případě, že začala nová synchronizační perioda jádra. V případě interní synchronizace poskytuje jádru synchronizační pulz určující začátek nové periody.
Návratová hodnota: `int`

`SDM8540_set_ext_trigger_polarity(int board_number, uint16_t polarity)`

Funkce nastavuje hranu, aktivní hranu externího triggeru. V závislosti na nastavení reaguje událost spouštěná na desce na sestupnou/vzestupnou hranu.
Návratová hodnota: `void`
Parametry:

Parametr	Hodnota	Funkce
polarity	0	Reakce na vzestupnou hranu
	1	Reakce na sestupnou hranu

4.5.3 Funkce pro ovládání A/D převodníku

`SDM8540_set_channel_count(int board_number, int channels)`

Funkcí je nutné nastavit počet převáděných kanálů před spuštěním A/D konverze. Tato hodnota je udržována v ovladači a dostupná prostřednictvím následující funkce.
Návratová hodnota: `void`
Parametry: channels - počet převáděných kanálů.

`SDM8540_get_channel_count(int board_number)`

Tato funkce vrací počet převáděných kanálů. Nahrazuje nesprávné předávání této hodnoty prostřednictvím hlavičkových souborů v minulé verzi jádra a ovladače. Především z důvodu této změny není aktuální ovladač zpětně kompatibilní s první verzí RT jádra.
Návratová hodnota: `int`

4.5.4 Funkce pro využití DMA

`SDM8540_install_dma(int board_number, DMASetup *dmasetup, int fill, char fill_byte, int dma_channel, int dma_buffer_size)`

Funkce provede inicializaci zadaného DMA kanálu. Je provedena alokace bufferu pro zápis hodnot přenesených DMA, proto tato funkce musí být volána před přechodem do RT režimu.
Návratová hodnota: `int`
Parametry:

Parametr	Hodnota	Funkce
dmasetup		Struktura obsahující ukazatel na dma buffer
fill	0/1	Vyplnění DMA bufferu ano/ne
fill_byte	char	Byte kterým je buffer vyplněn
dma_channel	0/1	Číslo inicializovaného kanálu
dma_buffer_size		Velikost bufferu kanálu

`SDM8540_uninstall_dma(int board_number, int dma_channel)`

Funkce provede deinicializaci DMA kanálu.
Návratová hodnota: `int`
Parametry: dma_channel - deinicializovaný kanál (hodnota 0/1)

`SDM8540_setup_dma(int board_number, int dma_source, int dma_channel, int demand_source, int mode, int direction, int offset, int length)`
Návratová hodnota: `int`
Parametry:

Parametr	Hodnota	Funkce
dma_source		Adresa zdroje dat pro přenos
dma_channel	0/1	Číslo nastavovaného kanálu
demand_source		Spouštěcí signál pro Demand Mode
mode	DMAMODE_NORMAL	Single nebo Block Mode
	DMAMODE_DEMAND	Demand Mode
direction	0/1	Směr přenosu z/na kartu
offset		offset zápisu v DMA bufferu
length		počet byte k přenosu

`SDM8540_start_dma0(int board_number)`

Funkce spustí DMA přenos na kanálu DMA0.
Návratová hodnota: `void`

`SDM8540_is_dma0_done(int board_number)`

Funkce vrací 1 v případě, že ve stavovém registru je nastaven bit *Channel Done*, což indikuje dokončení DMA přenosu. Nevýhodou při využití DMA bez přerušení je nutnost pollovat tento bit po celou dobu DMA přenosu.
Návratová hodnota: `int`

`SDM8540_reset_dma0(int board_number)`

Funkce resetuje příznakový registr kanálu DMA0 do defaultního stavu. V případě využití *Singe Mode* nebo *Block Mode* je nutné volat tuto funkci po přenesení každé hodnoty/sady

hodnot.
Návratová hodnota: `void`

4.5.5 Funkce ovládající čítače

`SDM8540_setup_ad_sample_counter(int board_number, uint16_t count)`

Tato funkce nastavuje hodnotu čítače A/D Sample Counter. Jeho funkcí je vyvolat přerušení v případě převedení určitého počtu vzorků a jejich zapsání do A/D FIFO. Je využitelný především pro iniciaci DMA přenosu.
Návratová hodnota: `void`
Parametry: count - počet vzorků po jejichž sejmutí čítač vyvolá přerušení

`SDM8540_read_ad_sample_counter(int board_number)`

Funkce vrací aktuální stav čítače A/D sample counter.
Návratová hodnota: `int`

4.5.6 Funkce pro záznam sad hodnot

`LOG_init()`

Funkce provede inicializaci zaznamenávání. Ověří přítomnost hodin reálného času a inicializuje kruhovou forntu pro záznam.
Návratová hodnota: `bool`

`LOG_start_logging()`

Funkce spustí zaznamenávání. Před jejím voláním je nutné volat výše popsanou funkci `LOG_init()`.
Návratová hodnota: `void`

`LOG_set_stop_condition(uint8_t channel, uint8_t type, int value)`

Funkce nastavuje typ ukončovací podmínky a prahovou hodnotu, případně počet vzorků.
Návratová hodnota: `void`
Parametry:

Parametr	Hodnota	Funkce
channel		Kanál, na který se váže ukončovací podmínka
type	0	Ukončení záznamu, pokud hodnota kanálu < value
	1	Ukončení záznamu, pokud hodnota kanálu = value
	2	Ukončení záznamu, pokud hodnota kanálu > value
	3	Ukončení záznamu při naměření "value"vzorků
value		Hodnota pro ukončovací podmínku

`LOG_set_start_condition(uint8_t channel, uint8_t type, int value)`

Funkce nastavuje typ spouštěcí podmínky a prahovou hodnotu, případně počet vzorků.
Návratová hodnota: `void`
Parametry:

Parametr	Hodnota	Funkce
channel		Kanál na který se váže spouštěcí podmínka
type	0	Spuštění záznamu pokud hodnota kanálu $<$ value
	1	Spuštění záznamu pokud hodnota kanálu $=$ value
	2	Spuštění záznamu pokud hodnota kanálu $>$ value
	3	Spuštění záznamu při naměření "value"vzorků
value		Hodnota pro spouštěcí podmínku

`LOG_add_ad_channel(uint8_t channel)`

Tato funkce přidá kanál zadaný parametrem do zaznamenávání.
Návratová hodnota: `void`
Parametry: channel - číslo kanálu, který má být přidán do zaznamenávání

`LOG_remove_ad_channel(uint8_t channel)`

Funkce odebere kanál zadaný parametrem ze zaznamenávání.
Návratová hodnota: `void`
Parametry: channel - číslo kanálu, který má být odebrán ze zaznamenávání

`LOG_add_user_var(uint8_t var)`

Funkce přidá proměnnou z uživatelské funkce do zaznamenávání [3].
Návratová hodnota: `void`
Parametry: var - index proměnné z uživatelské funkce, která má být přidána do zaznamenávání

`LOG_remove_user_var(uint8_t var)`

Funkce odebere proměnnou z uživatelské funkce ze zaznamenávání [3].
Návratová hodnota: `void`
Parametry: var - index proměnné z uživatelské funkce, která má být odebrána ze zaznamenávání

[3] Tato funkce není plně implementována, viz 6

4.5.7 Verzování ovladače

`get_fw_version()`

Funkce vrací strukturu obsahující verzi ovladače. Formát struktury ukazuje zdrojový kód 4.7. Je definována ve zdrojovém souboru `version.h`.

```
typedef struct
{
  char letter;    /** FW VERSION LETTER, values: V, P, R, B */
  uint8_t major;  /** FW VERSION COMPATIBILITY, values: 1..9  */
  uint8_t minor;  /** FW VERSION RELEASE, values: 0..9  */
  uint8_t sp;     /** FW VERSION SERVICE PACK, values: 0..99 */
  uint8_t build;  /** FW BUILD, values: 99..0 */
} version_t;
```

Zdrojový kód 4.7: Struktura s verzí ovladače

Návratová hodnota: `version_t`

Kapitola 5

Testování

Během práce na ovladači jsem průběžně prováděl testování jednotlivých funkcí ovladače přímo na CPU modulu s kartou SDM8540. Vzhledem k závislosti ovladače na konkrétním hardware není jiný snadný způsob ověřování funkčnosti možný.

5.1 Testovací aplikace

Ovladač je zkompilován jako statická knihovna **.lib*, pro testování je proto potřeba volat funkce poskytované touto knihovnou z testovací aplikace. Proto jsem v Open Watcomu vytvořil 32 bitový projekt pro DOS s PMODE/W 32 bitovým rozšířením, který obsahoval vytvořenou knihovnu a testovací kód volající funkce ovladače. Tato aplikace pak byla spouštěna přímo na CPU modulu s připojenou kartou SDM8540 a testována tak knihovna ovladače. Testování pomocných funkcí pro zaznamenávání hodnot, jako je například kontrolní součet, kruhová fronta a sestavení hlavičky záznamu, jsem prováděl přímo na PC s OS Windows, protože tyto funkce nevyžadují přítomnost rozhraní samotného ovladače.

5.2 Testování minimální délky periody

Problémem v bakalářské práci byla nemožnost dosáhnout délky jedné periody, ve které dochází jedenkrát k vyčtení hodnot z A/D převodníku kratší než 200 μs. Po odstranění tohoto problému, bylo nutné otestovat možnost zkracovat periodu na co možná nejnižší hodnotu. Vytvořil jsem si proto několik testovacích konfigurací s délkou periody od 200 μs až do 10 μs. Následně jsem je spouštěl na přípravku a monitoroval činnost karty, zda nedojde k zastavení A/D převodu.

Testovací konfiguraci zachycuje následující kód 5.1. Zdrojový kód celé testovací aplikace je obsažen v příloze B.

```
SDM8540_clear_board(BOARD);
SDM8540_clear_cgt(BOARD);
SDM8540_set_channel_count(NUM_OF_CHAN);

for (i = 0; i < NUM_OF_CHAN; i++)
```

```
{
        SDM8540_setup_cgt_row(&row, 0+i, 0, 0, 0, 0, 0,
          0, 0, 0);
        SDM8540_write_cgt_row_analog(BOARD,
          *((uint16_t *) & row));
}

SDM8540_set_pacer_clock_trig(BOARD, PCLK_START_SOFTWARE,
  PCLK_STOP_SOFTWARE, PCLK_NO_REPEAT);
SDM8540_set_pacer_clock_freq(BOARD, PCLK_INTERNAL,
  RATE, CLOCK_8MHZ);
SMD8540_setup_burst_clock(BOARD, 1, BURST_RATE, CLOCK_8MHZ);
SDM8540_set_conversion_start_source(BOARD, 2); //2=burst
SDM8540_set_burst_start_source(BOARD, 1); //1=pacer
```

Zdrojový kód 5.1: Konfigurace aplikace testující délku periody

S touto konfigurací, kde jsem postupně měnil hodnotu `RATE` až na *100000.0 Hz*, což odpovídá periodě *10 μs*, konverze probíhala v pořádku a nedošlo k zastavení převodů.

5.3 Testování záznamu hodnot

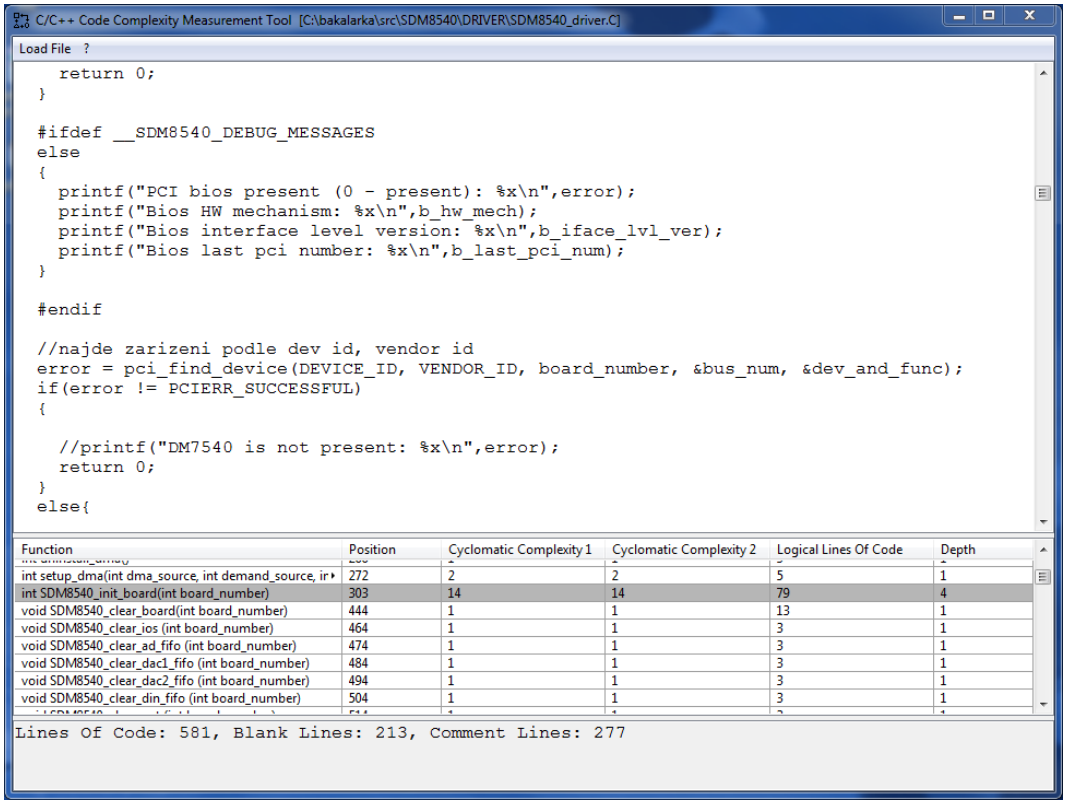
Testování zaznamenávání hodnot jsem prováděl postupně, jak vznikaly jednotlivé části tohoto modulu. Provedl jsem základní otestování funkčnosti kruhové fronty tak, že jsem do ní zapisoval v cyklu velké množství hodnot při nastavení poměrně menší velikosti fronty a poté jsem hodnoty vyčetl. Dále jsem otestoval výpočet CRC32 a porovnal ho s výpočtem na `http://www.lammertbies.nl/comm/info/crc-calculation.html`.

Dále jsem provedl v testovací aplikaci test čtení hodnot z A/D převodníku a jejich zápis do kruhové fronty. Jako ukončovací podmínku záznamu jsem zvolil definovaný počet hodnot, které mají být zaznamenány. Výstupem testovací aplikace pak byla kruhová fronta s vytvořenou hlavičkou obsahující informace o záznamu a kontrolní součet. Takto připravený záznam je možné buď zapsat do souboru, nebo přetypovat na pole bytů a odeslat po sériové lince aplikaci na počítači.

5.4 Cyklomatická komplexita ovladače

Provedl jsem analýzu programu z hlediska cyklomatické komplexity. Základní měření po jednotlivých souborech jsem napřed provedl pomocí softwaru C/C++ Code Complexity Tool `https://sites.google.com/site/maltapplication/`. Nevýhodou zde je to, že je možné provést analýzu pouze jednoho souboru, takže je nutné všechny soubory projektu otevírat zvlášť. Dále program poskytuje jen jednoduché měření komplexity, počtu řádků kódu a hloubky funkcí.

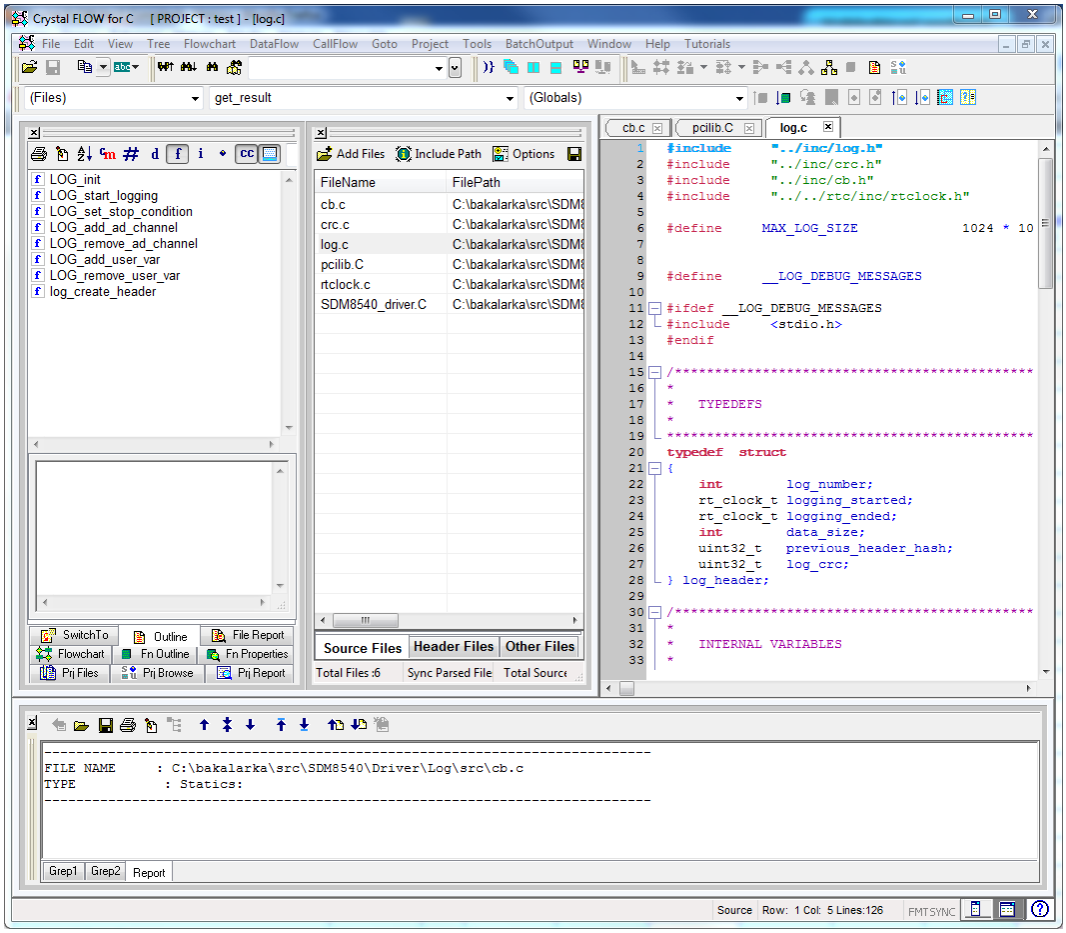
Obrázek 5.1 ukazuje analyzovaný soubor `SDM8540_driver.c`. V horní části okna je vidět náhled zdrojového kódu a ve spodní pak samotné hodnoty komplexity jednotlivých funkcí. S hodnotami v tomto programu nejde nijak pracovat či je třídit. Jedná se o velice základní možnost testování cyklomatické komplexity, která je vhodná spíše pro malé programy s minimálním množstvím zdrojových souborů.



Obrázek 5.1: C/C++ Code Complexity Measurement Tool

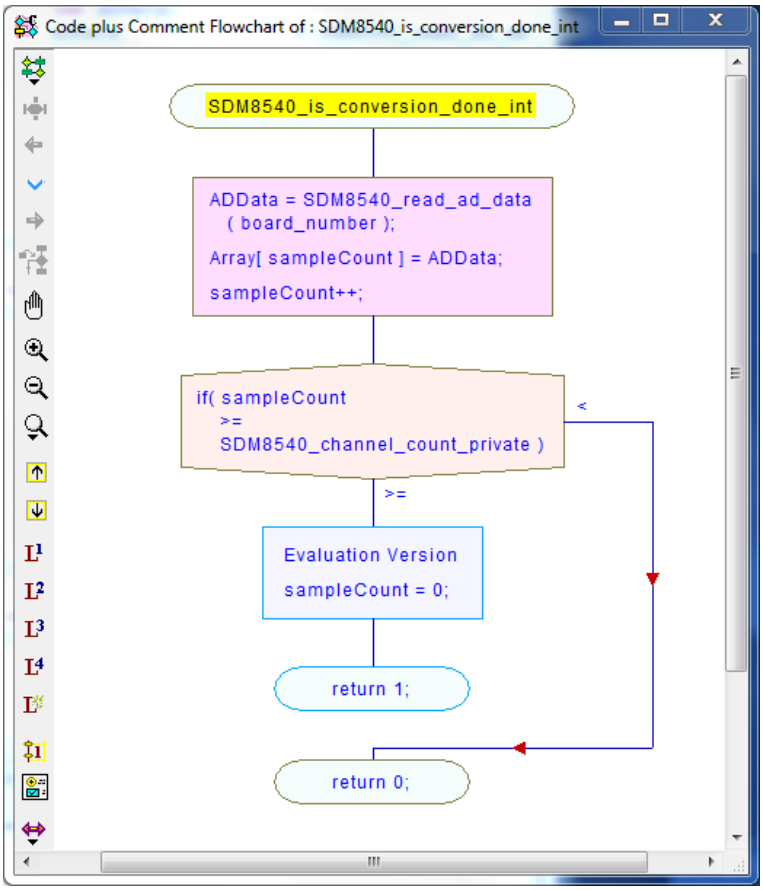
Detailnější testování celého projektu jsem provedl v programu Crystal FLOW for C. Je dostupný na adrese `http://www.sgvsarc.com/downloads.htm` jako Trial verze. Poskytuje však poměrně detailní informace o komplexitě funkcí. Lze s ním také generovat různé grafy, jako například stromy volání, nebo pro tuto práci zajímavější grafy toku řízení. Obrazovka programu s otevřeným projektem je na obrázku 5.2.

V levé části okna je vidět seznam funkcí v aktuálně otevřeném zdrojovém souboru, který lze vybrat uprostřed. V pravé části je pak jeho obsah. Již podle rozhraní programu je vidět, že software nabízí množství funkcí pro analýzu kódu. Vygenerovaný graf toku řízení pro funkci `SDM8540_is_conversion_done_int(int board_number, int * Array)` zobrazuje obrázek 5.3.



Obrázek 5.2: Crystal FLOW for C

Následující tabulka pak ukazuje cyklomatickou komplexitu a počet řádků prvních 10 funkcí ovladače seřazených od nejvyšší komplexity. Část reportu vygenerovaného programem Crystal FLOW je k nalezení v příloze C. Funkce s nejvyšší komplexitou je `pci_dump_device()`. Jedná se o funkci, která vypisuje typy zařízení připojených na PCI sběrnici. Tato funkce není určená k volání z RT jádra ani v non-real time režimu. V ovladači je především pro testovací účely a analýzu zařízení na PCI sběrnici v testovací aplikaci. Ostatní funkce začínající `pci_` jsou určeny k nalezení karty na PCI a jsou volány před přechodem do RT režimu při instalaci ovladače.



Obrázek 5.3: Ukázka grafu toku řízení

Funkce	Počet řádků	Cyklomatická komplexita
pci_dump_device	104	20
SDM8540_init_board	74	14
SDM8540_set_pacer_clock_freq	18	7
pci_is_bios_present	30	7
pci_list_all_devices	12	5
pci_find_class	22	5
pci_find_device	24	5
SMD8540_setup_burst_clock	18	5
SDM8540_set_dig_port_0	27	4
SDM8540_set_ut_mode	14	4

Tabulka 5.1: Cyklomatická komplexita prvních 10 funkcí od nejvyšší

Kapitola 6

Závěr

Na základě poznatků z bakalářské práce [6] a požadavků uživatele aplikace jsem vytvořil specifikaci požadavků na ovladač I/O modulu SDM8540HR-1 tak, aby bylo možné využít tento ovladač v RT systému R. Martínka. Základním předpokladem tohoto použití je například vyhnutí se čekacím smyčkám v kódu ovladače, aby například čekání na převod kanálu neblokovalo jiné úlohy jádra. Takovéto čekání je samozřejmě někdy nutné, ale je realizováno ze strany jádra s ohledem na plánování úloh.

Zvolil jsem vhodnou konfiguraci A/D převodníku tak, aby bylo možné dosáhnout periody vzorkování menší než 200 μs. Základním problémem dřívějšího ovladače byla nemožnost zkrátit periodu RT jádra danou délkou periody čítače Pacer Clock pod hodnotu 200 μs. Samotná délka periody Pacer Clocku problémem nebyla, jednalo se především o zastavení A/D převodníku, který je spouštěn rychlým časovačem Burst Clock. Spouštěcí signál Burst Clocku je tvořen signálem Pacer Clock signal, který je generován při začátku nové periody Pacer Clocku. Se zkracující se periodou Pacer Clocku tak dochází k rychlejší iniciaci Burst Clocku a tedy A/D převodu. Při testování převodu osmi diferenciálních kanálů došlo dříve k zastavení převodu na přibližné hodnotě 64 000 vzorků. Technická podpora RTD doporučila tento problém řešit využitím DMA přenosu hodnot z FIFO fronty A/D převodníku do paměti počítače. Problémem při implementaci DMA byla nemožnost využití přerušení, protože v aplikaci jádra je veškeré přerušení zakázáno. Proto jsem ponechal způsob přenosu hodnot beze změny, i když v ovladači je podpora DMA naimplementována, a v případě využití jádra podporujícího přerušení je tak možné tento způsob využít. Problém samotného zastavování převodu jsem vyřešil zakázáním přerušení na samotné kartě. Při zakázání všech přerušení v jádře sice nedocházelo k vyvolání přerušení na PCI sběrnici, ale na kartě samotné došlo k nastavení bitu Board FIFO Write inactive v Interrupt Status/Mask registru a zastavení převodu. Tato událost není v uživatelském manuálu zdokumentována, protože jediná možnost jak toto může nastat, je podle manuálu přeplnění FIFO fronty, její status však byl i po zastavení FIFO Empty.

Vytvořil jsem v ovladači funkce pro synchronizaci RT jádra poskytující možnosti interní a externí synchronizace, jak je v práci popsáno. Na základě požadavku uživatele aplikace došlo ke změně synchronizace tím způsobem, že jedna perioda Pacer Clock časovače nemusí nutně určovat délku jedné RT periody jádra. Je možné nastavit násobnost periody Pacer Clocku a tak během jedné RT periody převést hodnoty z A/D převodníku několikrát. Výhoda tohoto řešení je, že mohou být získány i jiné hodnoty než ty potřebné do regulačního algoritmu

a nezávisle na hodnotách pro regulaci mohou být počítány další parametry regulovaného systému.

Změřil jsem cyklomatickou složitost všech komponent programu. Maximální hodnoty dosahuje funkce `pci_dump_device()` z knihovny `pcilib.c`. Tato funkce slouží k vypsání zařízení na PCI sběrnicích počítače a v normálním provozu ovladače není volána. Slouží spíše k testovacím účelům a její cyklomatická komplexita je 20. Všechny ostatní funkce mají cyklomatickou komplexitu nižší. Velká část funkcí ovladače má komplexitu 1. To je dáno tím, že se funkce často skládají z několika zápisů do konfiguračních registrů a není nutné využívat větvení či cykly.

Naimplementoval jsem systém pro definování sady záznamů a jejich ukládání včetně konfigurovatelné spouštěcí i ukončovací podmínky. Podmínka může být nastavena několika způsoby, které jsou popsány v oddíle 4.5. Je možné zaznamenávat hodnoty získané z A/D převodníku. Při testování programu uživatelem aplikace se objevil nad rámec zadání diplomové práce požadavek na možnost ukládat i hodnoty vypočtené v průběhu uživatelského algoritmu. Analýza a návrh řešení této možnosti byla provedena v této práci, nicméně tato možnost zatím není v ovladači dostupná. Je to především proto, že podle navrženého řešení není možné z mapy proměnných, která je generována při kompilaci, získat údaj o jejich velikosti či typu. Řešením tohoto problému je možnost kontaktovat komunitu vyvíjející Open Watcom a požádat ji o doplnění tohoto údaje do generované mapy, případně se pokusit o tuto úpravu svépomocí, protože Open Watcom je distribuován jako open source. Další možností by bylo využít některého skriptovacího jazyka a analyzovat na základě mapy jednotlivé zdrojové soubory a z nich tak získat údaje o proměnných. Z časových důvodu nebyla tato možnost zatím vyřešena.

Práci jsem koordinoval s autorem diplomové práce na téma RT rozšíření OS DOS pro platformu PC 104, Radomírem Martínkem. To bylo nutné zejména z důvodu konfigurace synchronizace na základě kartou generovaných pulzů. Při vývoji jsem zdrojové soubory ukládal pomocí verzovacího systému Mercurial.

Možným zdokonalením navrženého systému by bylo zařadit před analogové vstupy karty Sample & Hold komponentu pro zachycení hodnot na více kanálech paralelně, případně analogové komparátory pro realizaci rychlých ochran.

Literatura

[1] BAKÓN, T. *Einsatz von Stromrichtern in der Energieversorgung zur Netzstützung und Fehlerstromlöschung.* Bochum : Ruhr-Universität Bochum, 2008.

[2] COLE, R. K. *Code Metrics: Cyclomatic Complexity and Unit Tests* [online]. 2007. [cit. 5. 5. 2013]. Dostupné z: <http://www.rkcole.com/articles/other/CodeMetrics-CCN.html>.

[3] FLÍGL, S. *Matrix Converter in Hybrid Drives.* Praha : FEL ČVUT, 2007.

[4] FLÍGL, S. *RT PC-based control system.* Bochum : Ruhr-Universität Bochum, 2003.

[5] KEOGH, J. - DAVIDSON, K. *Datové struktury bez předchozích znalostí.* Brno : Computer Press, 1st edition, 2006. 69 s.

[6] KLIK, L. *Ovladač pro PC104 I/O modul.* Praha : FEL ČVUT, 2011.

[7] KUZMANOVIČ, D. *Field oriented control and direct torque control of matrix converter-fed induction motor.* Praha : FEL ČVUT, 2011.

[8] MARTÍNEK, R. *RT jádro pro PC104 systém.* Praha : FEL ČVUT, 2011.

[9] MARTÍNEK, R. *RT rozšíření OS DOS pro platformu PC 104.* Praha : FEL ČVUT, 2013.

[10] Meilhaus Electronic. ME-2x00i und ME-3000 Datenblatt, 2008. http://www.meilhaus.de.

[11] MESSMER, H. P. *The Indispensable PC Hardware Book.* Harlow : Addison-Wesley, 3rd edition, 1999.

[12] Přispěvatelé OSDev Wiki. *RTC* [online]. 2012. [cit. 22. 4. 2013]. Dostupné z: <http://wiki.osdev.org/RTC>.

[13] Přispěvatelé OSDev Wiki. *CMOS* [online]. 2012. [cit. 22. 4. 2013]. Dostupné z: <http://wiki.osdev.org/CMOS#The_Real-Time_Clock>.

[14] Přispěvatelé Wikipedie. *Cyclomatic complexity* [online]. 2013. [cit. 5. 5. 2013]. Dostupné z: <http://en.wikipedia.org/wiki/Cyclomatic_complexity>.

[15] Přispěvatelé Wikipedie. *DOS extender* [online]. 2011. [cit. 23. 5. 2011]. Dostupné z: <http://cs.wikipedia.org/wiki/DOS_extender>.

[16] RTD Embedded Technologies Inc. *SDM8540/7540/DM8520/7520/PCI4520 USER MANUAL*. www.rtd.com : RTD Embedded Technologies, Inc., State College, PA, Rev. G edition, 2010.

[17] RTD Embedded Technologies Inc. *CML147786CX cpuModule User's Manual'*. www.rtd.com : RTD Embedded Technologies, Inc., State College, PA, Rev. C edition, 2007.

[18] Vývojáři Open Watcom. *Over-large Code Segments* [online]. 2009. [cit. 30. 4. 2013]. Dostupné z: <http://www.openwatcom.org/index.php/Over-large_Code_Segments#The_Linker_Error_and_The_Map_File>.

[19] WüRFLINGER, K. *Aktive Bedämpfung von niederfrequenten Störströmen in Bahnstromnetzen mittels der Umrichterregelung*. Bochum : Ruhr-Universität Bochum, 2007.

Příloha A

Seznam zkratek

A/D	Analog to Digital
BCD	Binary Coded Decimal
BIOS	Basic Input Output System
CGL	Channel Gain Latch
CGT	Channel Gain Table
CMOS	Complementary Metal–Oxide–Semiconductor
CPU	Central Processor Unit
CRC32	Cyclic Redundancy Check
D/A	Digital to Analog
DMA	Direct Memory Access
DSP	Digital Signal Processing
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPLD	Electrically Programmable Logic Device
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
I/O	Input/Output
LAS0	Local Address Space 0
LAS1	Local Address Space 1
PC	Personal Computer
PCI	Periprheral Component Interconnect

PIT	Programmable Interval Timer
RAM	Random-access memory
ROM	Read-only memory
RTC	Real-Time Clock
RT	Real Time
RTD	RTD Embedded Technologies, Inc.
S&H	Sample & Hold

Příloha B

Ukázka testovací aplikace

```
#include <stdio.h>
#include <conio.h>
#include <stdint.h>
#include "../driver/SDM8540_driver.h"

#define BOARD       0   // Select first board.
#define NUM_OF_CHAN   8   // Number of channels of burst.
#define RATE        20000.0   // Sample rate in [Hz]
#define BURST_RATE     1250000.0   // Burst rate in [Hz].

int main(int argc, char *argv[])
{
  float DataSet[NUM_OF_CHAN];
  int i;
  int cnvDone=0;
  int cnt1=0;
  int cnt2=0;
  ioctrl_t row;

  if(SDM8540_init_board(BOARD))
    cprintf("Board found & initialized!\n\r");
  else
    return 1;

  SDM8540_clear_board(BOARD);
  SDM8540_clear_cgt(BOARD);
  SDM8540_set_channel_count(NUM_OF_CHAN);

  for (i = 0; i < NUM_OF_CHAN; i++)
  {
    SDM8540_setup_cgt_row(&row, 0+i, 0, 0, 0, 0, 0,
      0, 0, 0);
```

```
    SDM8540_write_cgt_row_analog(BOARD,
      *((uint16_t *) & row));
  }

  SDM8540_set_pacer_clock_trig(BOARD, PCLK_START_SOFTWARE,
    PCLK_STOP_SOFTWARE, PCLK_NO_REPEAT);
  SDM8540_set_pacer_clock_freq(BOARD, PCLK_INTERNAL,
    RATE, CLOCK_8MHZ);
  SMD8540_setup_burst_clock(BOARD, 1, BURST_RATE, CLOCK_8MHZ);
  SDM8540_set_conversion_start_source(BOARD, 2); //2=burst
  SDM8540_set_burst_start_source(BOARD, 1); //1=pacer
  SDM8540_cgt_enable(BOARD, CSC_CGT);
  SDM8540_cgt_digi_enable(BOARD, 0);
  SDM8540_clear_ad_fifo(BOARD);
  SDM8540_start_pacer_clock(BOARD);

  while (!kbhit())
  {
    while(!cnvDone)
    {
      while(SDM8540_is_ad_fifo_empty(BOARD));
      cnvDone = SDM8540_is_conversion_done_v(BOARD, 10.0, DataSet);
    }
    cnvDone = 0;
    cnt1++;
    if(cnt1 % 10000 == 0)
      cprintf(".");

    while(!SDM8540_is_new_pacer_period(BOARD))
        {
      cnt2++;
      if(cnt2 % 1000000 == 0)
          cprintf("-");
        }
  }
```

Zdrojový kód B.1: Testovací aplikace testující délku periody

Příloha C

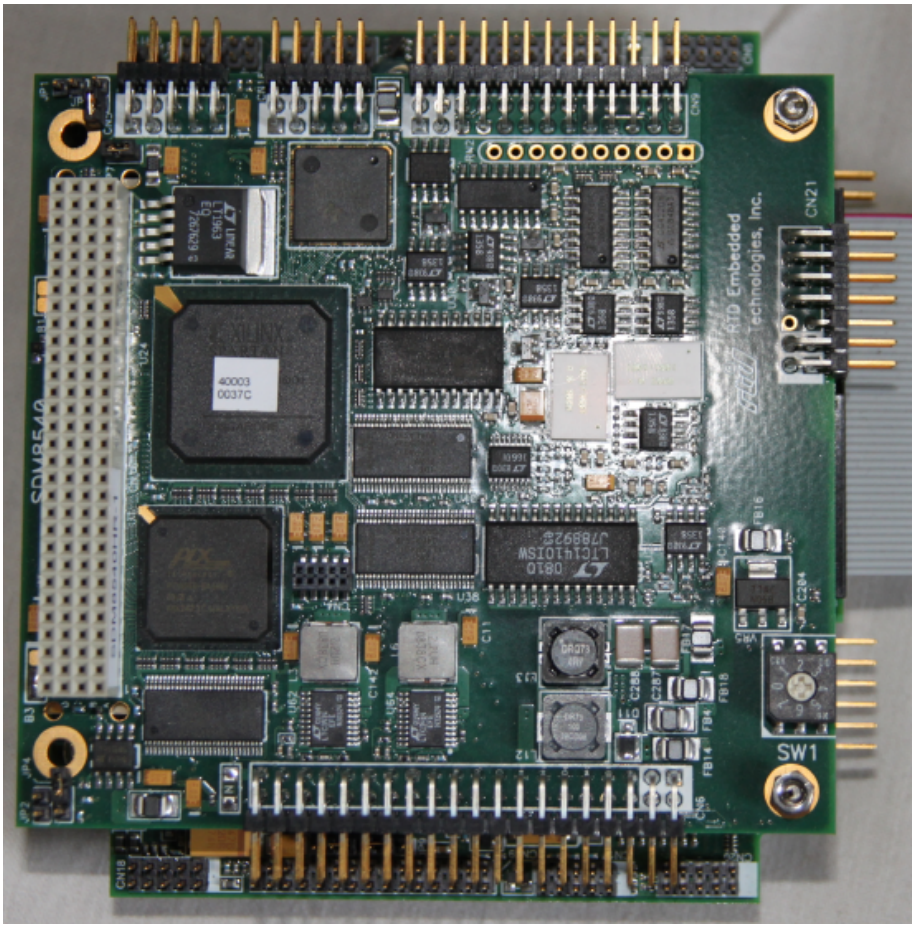
Report z aplikace Crystal FLOW

Function	Lines LoC	Case LBLs	Complexities				Densities		
			Cyclomatic v(G)	Modified Cyclomatic	Design iv(G)	Essential ev(G)	Cyclomatic vd(G)	Design id(G)	Essential ed(G)
pci_dump_device	104	18	20	3	1	1	0.03	0.33	
SDM8540_init_board	74		14	14	1	10	0.19	0.07	0.71
SDM8540_set_pacer_clock_freq	18		7	7	1	1	0.39	0.14	
pci_is_bios_present	30		7	7	1	1	0.23	0.14	
pci_list_all_devices	12		5	5	1	1	0.42	0.20	
pci_find_class	22		5	5	1	1	0.23	0.20	
pci_find_device	24		5	5	1	1	0.21	0.20	
SMD8540_setup_burst_clock	18		5	5	1	1	0.28	0.20	
SDM8540_set_dig_port_0	27		4	4	1	1	0.15	0.25	
SDM8540_set_ut_mode	14		4	4	1	2	0.29	0.25	0.50
SDM8540_set_ut_div	16		4	4	1	2	0.25	0.25	0.50
SDM8540_install_dma	12		3	3	1	2	0.25	0.33	0.67
SDM8540_is_new_period	8		3	3	1	2	0.38	0.33	0.67
SDM8540_read_dataset	7		3	3	1	1	0.43	0.33	
read_configuration_area	21		3	3	1	1	0.14	0.33	
LOG_init	11		2	2	1	2	0.18	0.50	1.00
SDM8540_is_dma0_done	4		2	2	1	2	0.50	0.50	1.00
pci_special_cycle	16		2	2	1	1	0.13	0.50	
pci_read_cfg_byte	6		2	2	1	1	0.33	0.50	
pci_read_cfg_word	6		2	2	1	1	0.33	0.50	
pci_read_cfg_dword	6		2	2	1	1	0.33	0.50	
crc32	4		2	2	1	1	0.50	0.50	
SDM8540_is_new_pacer_period	8		2	2	1	2	0.25	0.50	1.00
SDM8540_is_conversion_done_v	8		2	2	1	2	0.25	0.50	1.00
SDM8540_is_conversion_done_int	8		2	2	1	2	0.25	0.50	1.00
cb_write	4		2	2	1	1	0.50	0.50	
setup_dma	3		2	2	1	2	0.67	0.50	1.00
write_configuration_area	18		2	2	1	1	0.11	0.50	
SDM8540_read_cf_flag	4		2	2	1	2	0.50	0.50	1.00
init_rtc	13		2	2	1	1	0.15	0.50	
read_rtc	20		2	2	1	1	0.10	0.50	
SDM8540_setup_dma	17		2	2	1	2	0.12	0.50	1.00
install_dma	3		2	2	1	2	0.67	0.50	1.00
get_rtc_reg	2		1	1		1	0.50		

Obrázek C.1: Část reportu z aplikace Crystal FLOW for C

Příloha D

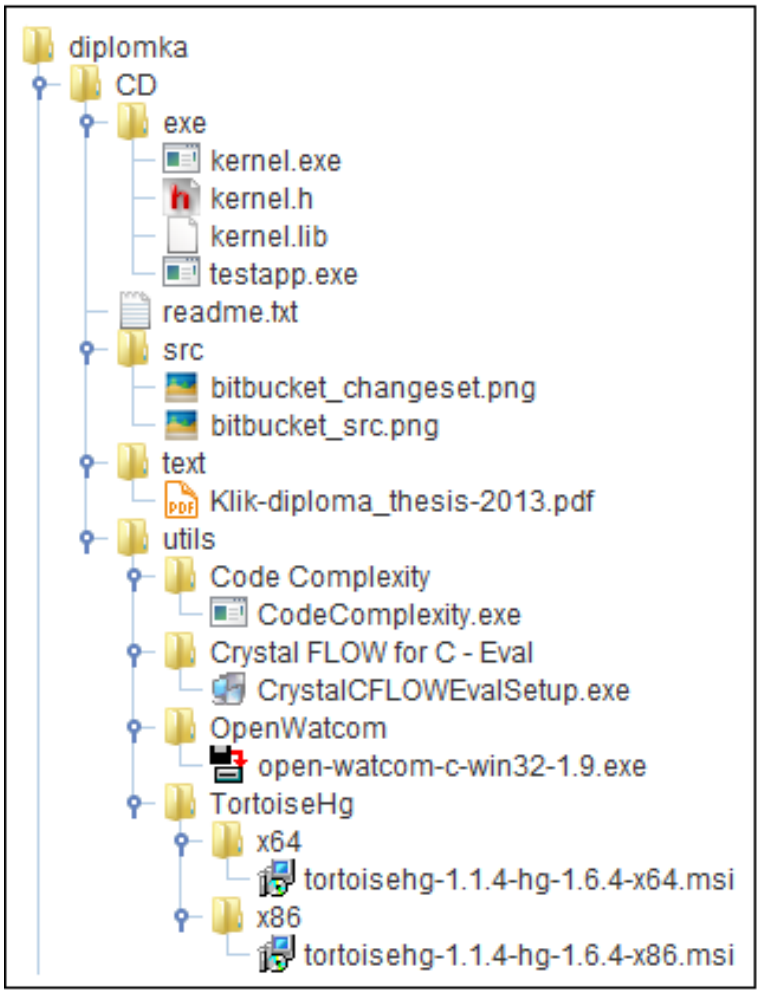
Karta SDM8540HR-1



Obrázek D.1: I/O karta RTD SMD8540HR-1

Příloha E

Obsah přiloženého CD



Obrázek E.1: Obsah přiloženého CD