VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ZAMEZENÍ VÝPOČETNÍHO PŘETÍŽENÍ POČÍTAČOVÉHO SYSTÉMU V DŮSLEDKU PŘERUŠENÍ

PREVENTING COMPUTER SYSTEM FROM COMPUTATIONAL OVERLOAD DUE TO INTERRUPTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. TOMÁŠ HAJDÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Hajdík Tomáš, Bc.**
Program: Informační technologie Obor: Počítačové a vestavěné systémy
Název: **Zamezení výpočetního přetížení počítačového systému v důsledku přerušení**
Preventing Computer System from Computational Overload Due to Interrupts
Kategorie: Vestavěné systémy

Zadání:

1. Shrňte typické přístupy k detekci událostí počítačovými systémy, jejich výhody, nevýhody a aplikační oblasti, detailně se zaměřte na detekci událostí s využitím přerušení.
2. Zdokumentujte a diskutujte i) možné příčiny a důsledky výpočetního přetížení počítačového systému vlivem nadměrné frekvence přerušení a ii) techniky snížení dopadu těchto důsledků na výpočetní vlastnosti počítačového systému.
3. Po dohodě s vedoucím zvolte vhodnou platformu (např. mikrokontrolér) a její aplikační vrstvu, na kterých experimentálně ověříte a vyhodnotíte dopad přerušení na jejich vlastnosti.
4. Pro platformu a aplikační vrstvu z bodu 3 implementujte techniky z bodu 2.ii a na základě vhodné sady experimentů vyhodnoťte a diskutujte i) jejich schopnost zamezit výpočetnímu přetížení platformy a ii) související efekty.
5. Shrňte a zhodnoťte dosažené výsledky a navrhněte možné směry pokračování v projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 26. října 2018

Abstrakt

Diplomová práca sa zaoberá technikami zamedzenia výpočtového preťaženia počítačového systému v dôsledku nadmernej frekvencie prerušení. Cieľom je zdokumentovať vplyv prerušení na zvolenú výpočtovú platformu obsahujúcu procesorové jadro ARM Cortex-M4. Práca popisuje a implementuje možné softvérové techniky znižujúce dopad dôsledkov preťaženia vplyvom nadmernej frekvencie prerušení. Práca zároveň na vhodnej sade experimentov overuje a porovnáva efektívnosť jednotlivých implementovaných techník.

Abstract

The master thesis deals with the techniques to prevent computer system from computational overloading due to excessive frequency of interruptions. The goal is to document the effect of interupts on a selected computing platform containing the ARM Cortex-M4 processor core. The work describes and implements possible software techniques that reduce the impact of consequences of overload due to excessive interruption frequency. At the same time the work verifies and compares the effectiveness of the particular implemented techniques by appropriate set of experiments.

Kľúčové slová

Obsluha prerušení, Prerušovací podsystém, Latencia prerušení, Vektor prerušenia, Striktný softvérový plánovač, Zhlukový softvérový plánovač, FITkit 3.0 Minerva, ARM Cortex-M4, Radič prerušení NVIC, DWT, Kinetis Design Studio

Keywords

Interrupt service routine, Interrupt subsystem, Interrupt latency, Interrupt vector, Strict software scheduler, Bursty software scheduler, FITkit 3.0 Minerva, ARM Cortex-M4, Nested vector interrupt controller, Data WatchPoint and Trace unit, Kinetis Design Studio

Citácia

HAJDÍK, Tomáš. *Zamezení výpočetního přetížení počítačového systému v důsledku přerušení.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Zamezení výpočetního přetížení počítačového systému v důsledku přerušení

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením vedúceho diplomovej práce pána Ing. Josefa Strnadela, Ph.D.. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Tomáš Hajdík
20. mája 2019

Poďakovanie

Týmto by som chcel poďakovať predovšetkým vedúcemu práce Ing. Josefovi Strnadelovi, Ph.D. za jeho odborné vedenie, pomoc a cenné rady počas tvorby tejto diplomovej práce.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

V súčasnosti sa prevažná väčšina mikrokontrolérov implementuje do mnohých komplexných systémových celkov a vstavaných systémov, od ktorých je vyžadovaná maximálna miera presnosti a reaktívnosti. Mikrokontroléry riadia bezdrôtové telefónne služby, automatizované výrobné linky, autonómne vozidlá, ale aj vojenské a obranné systémy. Každý komplexný počítačový alebo vstavaný systém obsahuje mnoho externých zariadení, akými sú sieťové rozhrania, senzory, akčné členy, pamäťové disky, ktoré vyžadujú stálu pozornosť a záruku spracovania dát a obsluhy v reálnom čase. Reakciou na vznik nových asynchrónnych udalostí sa zaoberá predovšetkým technika spracovania prerušení. Prerušenia sú však v takýchto situáciách potenciálne nebezpečné, pretože majú implicitne vyššiu prioritu spracovania ako vykonávanie hlavného kontextu.

Prístup vyvolania a spracovania prerušení stavia na predpoklade, že každé prerušenie je považované za kritickú časť spracovania programu. Z tohto dôvodu je vykonávané s maximálnym ohľadom na dobu trvania a rýchlosť zahájenia reakcie. Tento prístup však môže mať nedostatky v prípade, ak dochádza k asynchrónnym udalostiam príliš často alebo je ich odbavenie príliš zdĺhavé.

Hlavným cieľom tejto práce je preto zdokumentovať možné príčiny výpočtového preťaženia počítačového systému, ktoré je spôsobené nadmernou frekvenciou prerušení. Popísať dopady a dôsledky tohto preťaženia na počítačový systém. Ďalším cieľom je preskúmať a zdokumentovať techniky, ktoré sa snažia týmto nedostatkom vyhýbať, obmedziť ich vplyv alebo odstrániť nežiadúce dôsledky nadmernej frekvencie prerušení. Súčasťou práce je taktiež implementácia jednotlivých techník zabraňujúcich preťaženiu a overenie ich výsledkov na vhodnej sade experimentov.

Implementované techniky a vyhodnotenie experimentov je realizované na zvolenej platforme FITkit 3.0 Minerva, ktorá obsahuje 32 bitový mikrokontrolér KINETIS MK60 s nízkym príkonom obsahujúci procesorové jadro ARM Cortex-M4. Ide o vhodnú platformu, pretože je prispôsobená širokým možnostiam využitia a zároveň poskytuje vysokú mieru variability, či už z hľadiska hardvéru, softvéru alebo konfigurovateľných vstupno-výstupných rozhraní.

V druhej kapitole sú podrobne popísané rozdielne prístupy k detekcii udalostí v počítačových systémoch. Kapitola sa taktiež zaoberá ich výhodami, nevýhodami a oblasťami praktického použitia.

Kapitola pokračuje popisom rôznych techník, ktorých cieľom je zabrániť preťaženiu počítačových systémov v dôsledku nadmernej frekvencie prerušení. Zároveň podrobne popisuje implementačné detaily, princíp, činnosti a spôsoby zmiernenia dôsledkov preťaženia alebo spôsoby predchádzania ich vzniku.

Tretia kapitola je venovaná výberu vhodnej realizačnej platformy a jej hardvérovým a softvérovým prostriedkom. Taktiež obsahuje podrobný popis možností procesorového jadra ARM Cortex-M4.

Štvrtá kapitola popisuje návrh, implementáciu a zistené výsledky experimentov s cieľom špecifikovať limity zvolenej platformy z hľadiska výkonu pri vykonávaní obsluhy prerušení.

V piatej kapitole sú popísané konkrétne zvolené techniky zmierňujúce dopad výpočtového preťaženia vplyvom nadmernej frekvencie prerušení, ich implementácia a dosiahnuté výsledky na platforme FITkit 3.0 Minerva.

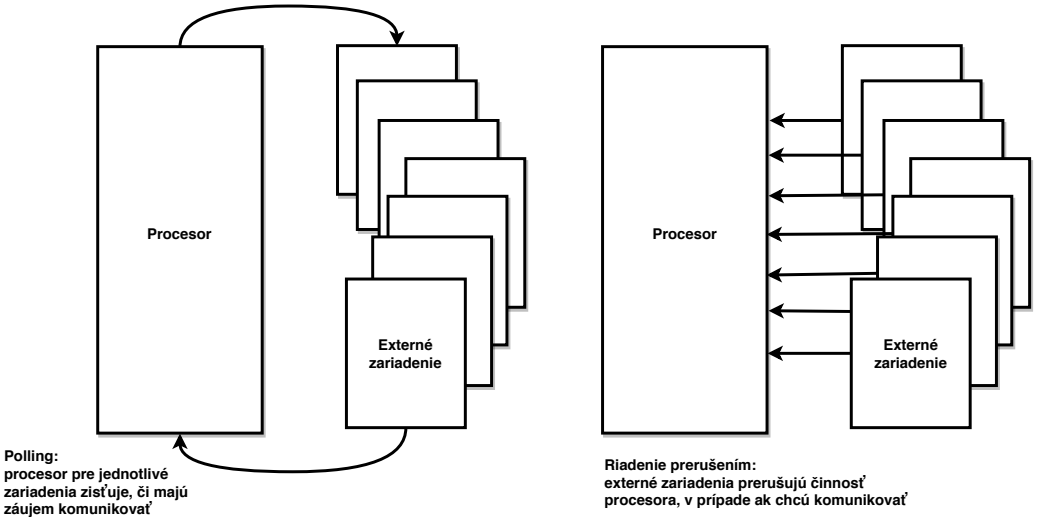
Kapitola 2

Udalosťami riadené počítačové systémy

Táto kapitola zhŕňa typické prístupy k detekcii udalostí počítačovými systémami. Zaoberá sa a vysvetľuje ich výhody, nevýhody a aplikačné oblasti využitia. Detailne popisuje detekciu udalostí s využitím prerušení a zameriava sa na výhody, nevýhody a možné limity tejto metódy.

2.1 Prístupy k detekcii udalostí počítačovými systémami

Počítačový systém môže vykonávať veľa úloh a zahŕňa rôzne typy externých zariadení, ako sú diskové jednotky, tlačiarne, monitory a klávesnice. Existujú dve hlavné metódy, ktoré môže procesor použiť, ak chce komunikovať s pripojenými zariadeniami. Jednou z metód je aktívne dopytovanie sa (polling) externých zariadení na zmenu a určenie, či sú pripravené na komunikáciu s procesorom, prípadne či chcú prijať alebo odoslať dáta. Tento prístup je známy ako aktívne pýtanie sa zariadení. Hlavnou nevýhodou tejto metódy je to, že zbytočne spotrebováva procesorový čas a to najmä pri komunikácii s relatívne pomalými zariadeniami [11].



Obr. 2.1: Ilustrácia rozdielu medzi pollingom a prerušením pri detekcii udalostí [11].

Lepším spôsobom je umožniť zariadeniam prerušiť procesor v činnosti práve vtedy, keď si vyžadujú jeho pozornosť. V závislosti od druhu prerušenia procesor opustí aktuálny program a prejde do špeciálneho programu nazvaného rutina obsluhy prerušenia (ISR). Tento program vie, ako komunikovať so zariadením a ako spracovávať alebo odosielať dáta. Po dokončení programu prerušenia sa vykonávanie vráti a pokračuje v programe, ktorý bol spustený pred prerušením. Obrázok č. 2.1 ilustruje dve hlavné metódy [11].

2.2 Polling

Je to metóda používaná pre komunikáciu CPU s externými periférami. Polling sa používa napríklad v prípade potreby prenosu senzorických dát z hardvérového senzoru do CPU. Ide o periodickú kontrolu. Polling sa používa najmä v prípade, ak dáta alebo signály nie sú natoľko urgentné, aby nemohli byť prijaté a spracované ďalšou periódou pollingu. Taktiež sa tento prístup využíva v prípade, ak hardvér senzoru generujúceho dáta alebo signály nie je schopný generovať prerušenia a ide o jednoduché senzory ako fotorezistor alebo tenzometer.

Model pollingu je najjednoduchší spôsob, ako skontrolovať nové dáta alebo hardvérové signály. Polling metóda kontroluje dostupnosť nových dát alebo hardvérových signálov v momente, keď nie sú veľmi naliehavé a dá sa zaručiť, že čas medzi snímaním dát bude dostatočne rýchly. Jednotlivé výzvy pollingu môžu byť pravidelné, realizované pomocou jednoduchého časovača, alebo príležitostné. Periodický polling používa časovač, ktorý indikuje, kedy by mali byť navzorkované dáta. Príležitostné snímanie dát je pre systém vhodné vykonávať napríklad medzi hlavnými systémovými funkciami alebo v určitom bode vykonávania opakovaného cyklu. Oportunistické vykonávanie pollingu je podľa definície menej pravidelné, ale má menší vplyv na včasnosť vykonávania iných aktivít, na ktoré je systém určený.

Oportunistický polling

Vyznačuje sa funkciou `poll()`, ktorá získava hodnoty a stav pripojených zariadení a odovzdáva tieto informácie na spracovanie CPU, prípadne príslušnému procesu. Rozdiel medzi oportunistickým a periodickým pollingom spočíva v tom, že periodický obsahuje inicializáciu časovača a blokuje funkcionalitu počas priebehu vykonávania pollingu.

Periodický polling

Rovnako ako oportunistický polling aj periodický obsahuje funkciu `poll()`, ktorá skenuje pripojené zariadenia a získava údaje a informácie o stave a predáva ich na spracovanie. Okrem toho obsahuje premennú pre nastavenie času pollingu a funkcie pre spustenie a zastavenie pollingu. Funkcia nastavujúca obsluhu pollingu vkladá adresu obslužnej rutiny prerušenia (ISR) do tabuľky prerušovacích vektorov a funkcia spustenia pollingu inicializuje časovač. Funkcia zastavenia pollingu zastaví časovač, ale vektor ponechá v tabuľke vektorov prerušení.

Spracovanie dát

Dáta získané pollingom sú následne priradené a predané príslušnému obslužnému a spracovávaciemu programu. V niektorých prípadoch to môže byť jeden klient alebo obslužný program, ktorý obsluhuje a spracováva dáta od všetkých externých komponentov, ale vo väčšine prípadov má každý komponent svoj vlastný obslužný program.

Dôsledky

Výhodou pollingu je jednoduchosť nastavenia a používania v porovnaní s obsluhou za pomoci prerušení (ISR). Periodický polling sa však zvyčajne implementuje práve pomocou časovača, ktorý je obsluhovaný pomocou obsluhy prerušení (ISR).

Pokiaľ ide o zmenu stavu, polling môže kontrolovať viacero rôznych zariadení naraz, ale zvyčajne je v porovnaní s prerušením menej rýchly a oneskorenie reakcie na zmenu tak trvá dlhšie. Z tohto dôvodu je potrebné dbať na to, aby v prípade, ak existujú časové limity pre spracovanie signálov alebo dát zo zariadení obsluhovaných pollingom, bol čas pollingu plus čas odozvy vždy menší ako je časový limit spracovávaných dát. Ak dáta prichádzajú rýchlejšie ako je čas pollingu, dôjde k ich strate. V niektorých aplikáciách to nespôsobuje problém, no v iných to môže mať fatálne následky.

Implementačné stratégie

Najjednoduchšou stratégiou implementácie je vloženie hardvérovej kontroly do stredu hlavného spracovacieho cyklu, ktorý sa vykonáva počas celého fungovania systému. Tento prístup sa nazýva symetrický oportunistický polling, pretože pracuje po celý čas rovnakým spôsobom, avšak dĺžka trvania spracovávacích cyklov sa môže značne líšiť. Asymetrický oportunistický polling sa implementuje umiestnením kontroly nových dát na vhodné miesta nesúvisiace so samotným spracovaním. Tento prístup umožňuje lepšie ladenie, nakoľko môžu byť pridané ďalšie kontroly pre zvýšenie citlivosti, ale má väčší vplyv na primárny tok a je náročnejší na udržiavanie.

Periodický polling sa uskutočňuje v pravidelnom časovom intervale (nazývanom perióda) s ohraničenou odchýlkou (nazývanou jitter). Pre dosiahnutie pravidelnosti tejto kontroly nových dát je iniciácia vykonávaná časovačom viazaným na prerušenie. Periodický variant polling metódy je špeciálnym prípadom metódy spracovania pomocou prerušení. Viac o tejto metóde je možné sa dočítať v literatúre [12].

Využite

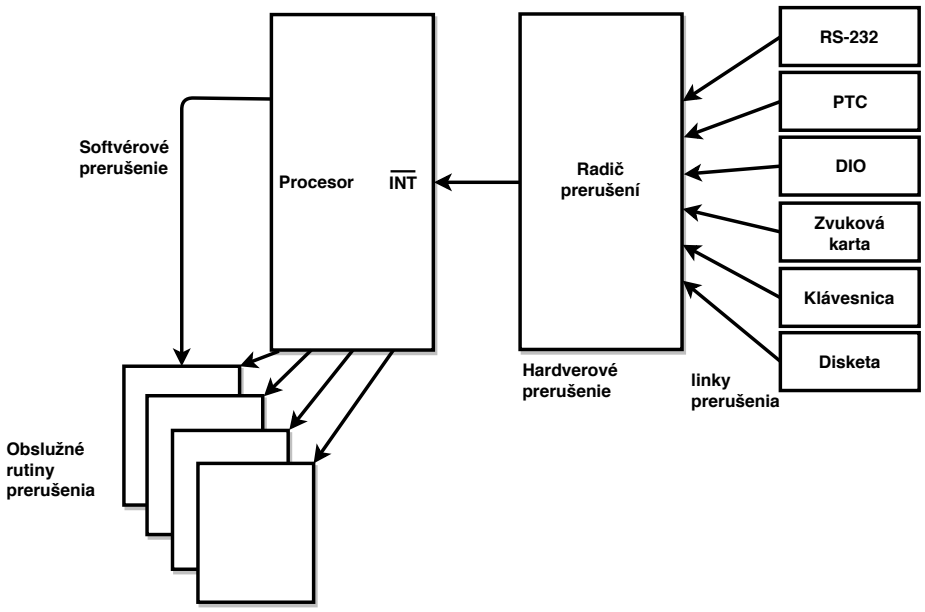
Táto metóda prístupu k detekcii udalostí sa majoritne využíva v situáciách, ktoré pre svoju jednoduchú štruktúru a zoskupenie často nie sú schopné inak komunikovať s externými hardvérovými komponentmi, nakoľko tie nedisponujú žiadnym komunikačným rozhraním. Získanie dátových údajov alebo vzorkovanie externých signálov tak musí byť vykonávané priamo zariadením, ktoré iniciovalo vzorkovanie. Príkladom využitia je napríklad použitie v senzorických zariadeniach pri komunikácii s pasívnym senzorom, kde primárnou funkciou je zaznamenávať a vzorkovať dáta z pripojeného elektronického senzoru. Na tieto aplikácie sú využívané často tie najjednoduchšie mikrokontroléry.

2.3 Detekcia udalostí s využitím prerušení

Princíp

Vo všeobecnosti sa prerušenia a ich obsluha používajú na riadenie situácií a prípadov s vysokou prioritou, vyžadujúce prerušenie kódu, ktorý procesor práve vykonáva. Generovanie prerušenia môže nastať hardvérom alebo softvérom, ako je znázornené na obrázku č. 2.2. Ak zariadenie chce prerušiť procesor, informuje programovateľný radič prerušenia (PIC). Radič prerušenia rozhodne, či by mal procesor prerušiť v činnosti. Ak dôjde k prerušeniu

procesora, potom procesor načíta vektor prerušenia, aby určil, ktoré zariadenie spôsobilo prerušenie. Následne v závislosti od zariadenia, ktoré spôsobilo prerušenie, sa uskutoční volanie prerušovacej rutiny ISR. ISR komunikuje so zariadením a spracováva potrebné dáta. Po dokončení prerušovacieho programu sa vráti do pôvodného programu. Prerušenie softvéru spôsobí, že program preruší jeho vykonanie a prejde na rutinu obsluhy prerušenia. Medzi typické prerušenia softvéru patrí čítanie klávesy z klávesnice, odosielanie textu na obrazovku alebo načítanie aktuálneho dátumu a času [11].



Obr. 2.2: Schéma obsluhy prerušenia pomocou radiča prerušení [11].

Typy prerušení

Prerušenia môžu byť kategorizované a rozdelené do týchto odlišných skupín.

- **Maskovateľné prerušenia (MI):** ide o hardvérové prerušenie, ktoré môže byť ignorované nastavením bitu v registri bitovej masky prerušení (IMR). Hardvérové prerušenie, ktorému chýba bitová maska, sa označuje ako nemaskovateľné prerušenie (NMI).

- **Nemaskovateľné prerušenia:** nemôžu byť nikdy ignorované. Používajú sa pre úlohy s najväčšou prioritou, ako sú časovače, špeciálne Watchdog časovače. Maskovateľné a nemaskovateľné prerušenia sú používané napríklad vstupno-výstupnými zariadeniami (I/O) pre signalizáciu operácií ako sú: zápis, čítanie, stav, chyby a iné I/O operácie.

- **Medziprocesorové prerušenia (IPI):** sú špeciálny prípad prerušenia, ktoré sú generované jedným procesorom na prerušenie iného procesoru v multiprocesorovom systéme.

- **Hardvérové prerušenia:** sú generované iným zariadením ako je napríklad časovač, radič vstupno-výstupného zariadenia alebo radič DMA.

- **Softvérové prerušenia:** sú generované pri splnení istých podmienok, ktoré sa vyskytnú ako dôsledok vykonávania inštrukcií. Patrí sem napríklad aritmetické pretečenie, delenie nulou alebo pokus o vykonanie nelegálnej inštrukcie. Softvérové prerušenia sa taktiež používajú pre implementáciu systémových volaní.

- **Falošné prerušenia:** patria sem hardvérové prerušenia, ktoré sú nežiadúce a typicky sú generované napríklad elektrickou interferenciou na prerušovacej linke, výpadkom napájania alebo nesprávnym návrhom hardvéru. Taktiež môžu byť zapríčinené chybami v pamäti, ako je napríklad chyba parity pamäte [17].

Prehľad hardvérových a softvérových prerušení

Hardvérové prerušenia využívajú zariadenia na komunikáciu a vyžiadanie pozornosti zo strany operačného systému. Hardvér počítačového systému obsahuje mnoho radičov vstupno-výstupných zariadení a mechanizmus prerušení musí byť schopný identifikovať zdroj žiadosti o prerušenie. Na tento účel sa vo všeobecnosti používajú žiadosti o prerušenie (IRQ), kde každá je určená pre iný radič zariadenia. Pre jednotlivé vektory prerušení je v pamäti vyhradené miesto, na ktorom je uložená začiatočná adresa rutiny na spracovanie prerušenia. Ak konkrétne zariadenie žiada o prerušenie, vysiela jeho signál žiadosti o prerušenie. To spôsobí modifikáciu programového čítača CPU zodpovedajúcim vektorom prerušení.

Hardvér priraďuje každej žiadosti o prerušenie úroveň priority prerušenia. CPU register ukladá aktuálnu prioritu procesoru. Ak je aktuálna úroveň priority procesora väčšia alebo sa rovná priorite žiadosti, prerušenie sa ignoruje. Ak je úroveň naopak menšia, priorita procesora sa nastaví na aktuálnu hodnotu priority prerušenia a riadenie sa predá zodpovedajúcej obsluhe prerušení. Po dokončení je priorita procesora nastavená na pôvodnú hodnotu a pokračuje sa v predošlej činnosti. Je potrebné poznamenať, že obslužná rutina prerušenia (ISR) môže byť prerušená iným prerušením vyššej priority.

Prerušenia môžu byť taktiež vyvolané softvérom, a to buď z dôvodu výnimočných situácií alebo špeciálnou inštrukciou, ktorá spôsobí prerušenie vykonávania. Prvý prípad sa nazýva výnimkou a používa sa na chyby alebo udalosti vyskytujúce sa počas vlákna vykonávania programu, ktoré sú výnimočné a nedá sa s nimi zaobchádzať v rámci samotného vlákna. Napríklad ak dôjde v aritmeticko logickej jednotke procesora k deleniu nulou, čo nie je možné, dôjde k *„výnimke delenia nulou“*. To spôsobí, že počítač opustí výpočet a zobrazí správu o chybe. Inštrukcie softvérového prerušenia je tak možné využiť na implementáciu systémových volaní, napríklad na obsluhu nízkoúrovňového systémového softvéru akým sú ovládače zariadení. V operačnom systéme to môže byť napríklad komunikácia s ovládačom disku pri vyžiadaní načítania alebo zápisu dát na disk.

Prerušenia zabezpečujú dobrú latenciu a nízku réžiu pri nízkej záťaži. Významne však degradujú v prípade vysokej miery prerušení, pokiaľ nie je dané žiadne opatrenie aby sa predišlo týmto problémom. Ak systém trávi priveľa času obsluhou prerušení, môže dochádzať k výpadkom. V extrémnych prípadoch, akými môže byť napríklad vysoká sieťová prevádzka, môže dôjsť k úplnému zablokovaniu systému. Pre zabránenie týmto problémom je potrebné starostlivo plánovať obsluhu prerušení [17].

Programovateľný radič prerušení

Pri spracovaní rôznych zdrojov prerušenia sa bežne využíva podporný hardvér pre CPU, ktorý môže byť buď externý alebo integrovaný na čipe. Tento hardvér sa označuje ako programovateľný radič prerušení (PIC). PIC obsahuje niekoľko vektorov žiadostí o prerušenie

(IRQ), ktoré sa používajú pri vysielaní žiadostí z externých zariadení. Zároveň obsahujú interné výstupné prepojenie určené pre vyžiadanie si prerušenia od procesora. PIC je zodpovedný za zaradenie žiadostí o prerušenie z rôznych periférií do fronty a ich následné posielanie jednej za druhou do CPU v poradí podľa priorít. Priority sú očíslované a zoradené, pričom prerušenia s číslom 0 majú najvyššiu prioritu. Samotný hardvér systémovej dosky určuje, ktoré zariadenia vyvolávajúce prerušenia sú prepojené s riadkom žiadosti o prerušenie v PIC [17].

Vektory prerušenia

Vektory prerušenia sú adresy, ktoré informujú obsluhu prerušenia o tom, kde nájsť rutinu prerušenia ISR. Všetky prerušenia majú priradené číslo od 0 do 255. Vektory prerušenia spojené s číslom prerušenia sú uložené v dolných 1024 bajtoch pamäte počítača. Napríklad prerušenie 0 je uložené od 0000:0000 do 0000:0003, prerušenie 1 od 0000:0004 do 0000:0007 atď. Prvé dva bajty ukladajú posun a ďalšie dva ukladajú adresu segmentu. Každému číslu prerušenia je priradená vopred určená úloha, ktorá je uložená v tabuľke prerušení [11].

Cyklus potvrdzujúci prerušenie

CPU reaguje na žiadosť o prerušenie potvrdzovacím cyklom prerušenia (INTA). Vo väčšine súčasných procesorov reakcia na prerušenie obsahuje nasledujúce kroky:

1. Hardvérové zariadenie generuje signál žiadosti o prerušenie (IRQ).
2. PIC priradí žiadosti o prerušenie prioritu a porovná ju s ostatnými žiadosťami, ktoré čakajú. Žiadosť o prerušenie je preposlaná procesoru.
3. Ak sú prerušenia povolené, CPU potvrdí prijatie potvrdzujúcim signálom INTA.
4. Reakciou na potvrdenie je umiestnenie prerušovacieho vektora zariadením na dátovú zbernicu.
5. CPU načíta vektor na základe ktorého vyhľadá korešpondujúcu adresu ISR.
6. Nakoniec procesor uloží aktuálny kontext na zásobník, zakáže prerušenia a prejde na spracovanie obslužnej rutiny prerušenia ISR. [17].

Obslužná rutina prerušenia

Obsluha prerušenia, označovaná ako ISR, je softvérová funkcia buď vo firmvéri mikrokontrolérov, v operačnom systéme alebo ovládači zariadenia. Obsluha prerušení je iniciovaná hardvérovým prerušením, softvérovým prerušením alebo softvérovou výnimkou. Obsluhy jednotlivých prerušení vykonávajú rozdielne funkcie, ktoré sa líšia príčinou vzniku prerušenia a rýchlosťou, s ktorou dané prerušenie dokončí svoju úlohu. Vykonanie obslužnej rutiny prerušenia vyžaduje nasledujúcu sekvenciu krokov, ktoré sú vysvetlené nižšie.

1. Uloženie kontextu systému. Prvou úlohou ISR je uložiť všetky registre aktuálneho kontextu (vlákna), aby bolo možné po skončení ISR obnoviť pôvodný kontext. To sa vykoná umiestnením registrov a príznakov na zásobník.
2. Zablokovanie všetkých prerušení. Obvykle sa tento krok vykonáva počítačovým hardvérom automaticky. Všetky prerušenia na úrovni procesoru musia byť zakázané.

3. Povolenie prerušenia vyššej priority, ktoré by bolo možné spracovať počas obsluhy prerušenia. ISR musí byť vykonaná v minimálnom čase kým sú zakázané prerušenia. Nesmie však dôjsť k strate prerušenia vyššej priority.

4. Zistenie príčiny prerušenia. Aktivácia prerušenia môže byť zapríčinená z rôznych dôvodov. Pred vykonaním prerušenia musí obsluha určiť príčinu.

5. Vykonanie samotnej obsluhy prerušenia. Obsluha prerušení sa líši v závislosti od zdroja vyvolania prerušenia. Niektoré externé zariadenia vyžadujú prijatie potvrdzovacieho signálu. ISR musí však prioritne vykonať samotnú obsluhu zariadenia, ako napríklad prijatie alebo odoslanie dát zo sériového portu. Najdôležitejšie je vyhnúť sa vykonávaniu nadmerného spracovania v rámci ISR. Trvanie ISR musí byť čo najkratšie, nakoľko sú blokované prerušenia s nižšou alebo rovnakou prioritou.

6. Obnova kontextu systému. Po dokončení obslužného kódu prerušenia musí byť obnovený systémový kontext zo zásobníka.

7. Povolenie prerušenia akejkoľvek priority zablokovanej počas vykonávania obsluhy. ISR musí poslať príkaz o ukončení prerušenia (EOI) do PIC ešte pre vykonaním inštrukcie IRET.

8. Pokračuje sa vo vykonávaní pôvodného kódu pred prerušením. Obsluha ISR sa vráti na pôvodné miesto programu vykonaním inštrukcie IRET. [17].

2.4 Príčiny a dôsledky výpočtového preťaženia počítačového systému vplyvom nadmernej frekvencie prerušení

Mnoho vstavaných systémov riadených prerušením je citlivých na preťaženie v dôsledku prerušení. Ide o stav, pri ktorom je frekvencia vonkajších prerušení natoľko vysoká, že u ostatných aktivít na procesore dochádza k vyhladoveniu. Prerušenia sú nebezpečné v tom, že majú vyššiu prioritu ako spracovávanie v iných kontextoch, napríklad vláknach. Zanedbaním maximálnej hodnoty času príchodu prerušení tak vzniká významná medzera v predpokladoch vedúcich k bezpečnému a odolnému systému.

Preto sa pri systémoch, ktoré sa snažia aktívne zabraňovať preťaženiu používajú rôzne techniky. Snahou je vyvinúť plánovač prerušení tak, aby spĺňal nasledujúce vlastnosti:

- Priemerné zaťaženie procesora musí byť úmerné rýchlosti príchodu prerušení počas odľahčenia.

- Priemerné zaťaženie procesora musí byť ohraničené konštantou počas stavu preťaženia.

- Musí byť možné vyčísliť najhoršie prípady oneskorenia z dôvodu prerušení s nízkou prioritou a neprerušovanú prácu, aby bolo možné poskytnúť garanciu výkonu.

- Plánovače by mali byť dynamicky parametrizovateľné a tým umožňovať systémom pružne vybrať vhodné úrovne ochrany pred preťažením v dôsledku prerušení.

- Režijné náklady musia byť primerané.

- Plánovač by mal pracovať v zhode s inými plánovačmi.

Prístup k eliminácii preťaženia v dôsledku prerušení by mal brať do úvahy taktiež tri hlavné charakteristiky vstavaných systémov.

Prvou je, že vstavané systémy sú často veľmi limitované zdrojmi. Či už ide o veľkosť pamäti alebo spotrebu energie, vylučuje sa tým využitie vyšších abstrakcií založených na rezerváciách, ktoré boli vyvinuté, aby zabránili preťaženiu procesora v otvorených systémoch. Rezervácie na úrovni vlákien a následné predanie kontextu inému vláknu taktiež nemôžu priamo zabrániť preťaženiu v dôsledku prerušení, aj keď môžu oddialiť jeho nástup. Pridávanie vlákien je však samo o sebe problém na pomalých procesoroch, kde je réžia vlákien relatívne drahá. Taktiež pre ušetrenie spotreby energie používajú režim spánku, ktorý trvá kým nedôjde k prerušeniu, čím sa výrazne predlžuje činnosť batérie.

Druhou charakteristikou je, že vstavané systémy majú tendenciu mať prísne časové požiadavky kvôli úzkemu prepojeniu medzi softvérom a hardvérom. Napríklad, ak je oneskorenie softvéru príliš veľké, môžu byť nestabilné spätné väzby snímačov a aktuátorov, čo následne vedie k poruche. Taktiež aj vstavané periférie ako napríklad sériové porty vyžadujú isté nároky na oneskorenie z dôvodu obmedzenia veľkosti vyrovnávacej pamäte.

Treťou charakteristikou je čo najprijateľnejšia cena, ktorá vedie k používaniu lacných zariadení s malým výpočtovým výkonom. To znamená, že pre mnoho vstavaných systémov nie sú vývojári schopný upraviť firmvér tak, aby nebol hlavný procesor preťažený nadmernými prerušeniami.

Alternatíva k prerušeniu, polling, funguje dobre v prípade preťaženia, ale znižuje výkon a spotrebováva energiu generovaním zbytočnej práce. Preťaženie v dôsledku prerušení nemusí byť nevyhnutne spôsobené vysokým zaťažením pri prerušení, ale skôr neočakávaným vysokým zaťažením pri prerušení. Rýchly procesor s krátkou rutinou prerušenia dokáže jednoducho spracovať stovky tisíc prerušení za sekundu. Naopak, u pomalého procesora, ktorý pracuje s dlhým kódom prerušenia, môže dôjsť k preťaženiu stovkami prerušení za sekundu. [19].

2.5 Techniky zabraňujúce preťaženiu v dôsledku nadmerných prerušení

Existuje viacero rozdielnych implementácií prerušení. Napríklad v prípade rodiny procesorov Atmel AVR má každé prerušenie dva špeciálne hardvérové bity. Povoľovací bit a čakajúci bit. Taktiež obsahuje jeden globálny bit prerušenia, ktorý je možné použiť na vypnutie všetkých obslužných programov prerušenia. Ak nie je povolený globálny bit prerušenia a dôjde k prerušeniu, procesor pokračuje vo vykonávaní svojho bežného inštrukčného toku. Ak je tento bit nastavený, je na obsluhu vybraté prerušenie s najnižším indexom, ktoré má nastavený povoľovací bit. Procesor potom atomicky:

- vynuluje čakajúci bit prerušenia
- vynuluje globálny povoľovací bit prerušenia
- uloží programový čítač a stav procesora na zásobník
- načíta adresu vektora vybraného prerušenia do počítadla programov

Hardvérovo založená implementácia môže jednoducho filtrovať nežiadúce signály na výstupe zbernice. Jedinou možnosťou softvérového plánovacieho mechanizmu prerušení je znižovať mieru nastavovania povoľovacieho bitu prerušenia. Základný rozdiel medzi hardvérovým

a softvérovým plánovačom spočíva v tom, že zatiaľ čo hardvérový je mimo procesorovej rozhodovacej logiky prerušení, tak softvérový je jej súčasťou [19].

2.5.1 Striktný softvérový plánovač

Technika je implementovaná výlučne v softvéri a presadzuje minimálny čas medzi prerušeniami. Minimálny čas medzi prerušeniami alebo jeho inverziu, maximálnu frekvenciu prerušení, špecifikujú návrhári systému. Podrobnosti o algoritme je možné nájsť v literatúre č. [19]. Prológ prerušenia je upravený tak, aby zahŕňal kód, ktorý vynuluje povoľovací bit prerušenia a nastaví jednorázový časovač, ktorý expiruje po určitom intervale v budúcnosti. Keď časovač expiruje, handler znovu povolí prerušenie. Tým je zároveň vylúčený konfliktný prístup k časovaču (opätovné nastavenie časovača ak je už nastavený). Toto riešenie funguje dobre na nemodifikovanom hardvéri, ale spôsobuje určité režijné náklady, pretože zdvojnásobuje počet prerušení.

2.5.2 Zhlukový softvérový plánovač

Zhlukový softvérový plánovač je taktiež založený na softvéri. Má nižšie režijné náklady ako striktne softvérový plánovač, ale poskytuje slabšiu izoláciu. Plánovač zablokuje prerušenia iba ak zaznamená zhluk viacerých žiadostí o prerušenie. Tento plánovač má dva vstupy, maximálnu veľkosť zhluku a maximálnu rýchlosť príchodu zhlukov jednotlivých prerušení (je to rozdielne oproti maximálnej rýchlosti príchodu jednotlivých žiadostí o prerušenie v predošlom prípade). Podrobný popis je možné nájsť v literatúre č. [19].

Pri implementácii tohto plánovača prerušení je prológ prerušenia modifikovaný tak, aby postupne zvyšoval počítadlo prerušení a keď je počet väčší alebo sa počet rovná maximálnej veľkosti zhluku, tak vynuluje povoľovací bit prerušenia, pretože hrozí riziko preťaženia. Počítadlo prerušení je vynulované periodickým časovačom, ktorý beží asynchrónne vzhľadom na pracovnú záťaž prerušení. Frekvencia tohto časovača je rýchlosť príchodu zhlukov prerušení. Tento časovač taktiež nastavuje povoľovací bit prerušenia, ak bol predtým vynulovaný.

Užitočnou optimalizáciou je ponechať periodický časovač vypnutý, zatiaľ čo je počítadlo pod stanovenou prahovou hodnotou a tým sa vyhnúť réžii časovača, pričom sa očakáva, že sa prah prekračuje len zriedka. Toto funguje len v prípade ak je pre plánovač prerušení dedikovaný hardvérový časovač. Pri odľahčení má totiž periodický časovač tendenciu ponechať čakanie po celý čas. Taktiež je potrebné vynulovať čakajúci bit prerušenia tesne pred povolením prerušenia časovača, aby sa zabránilo prípadu, kedy by sa mohol bezprostredne vyskytnúť zhluk viacerých prerušení.

Maximálnou veľkosťou zhluku a maximálnou rýchlosťou príchodu zhlukov sa dá regulovať plánovač tak, aby dosahoval rôzne výkonové výnosy. Zvyšovanie veľkosti zhluku a znižovanie rýchlosti príchodu zhlukov ovplyvňuje maximálnu rýchlosť príchodu prerušení, pretože sa znížia režijné náklady. Tým, že sú zhluky dlhšie sa zároveň zvyšuje čas, ktorý môže spôsobiť oneskorenie kódu, ktorý beží spustený v systéme.

Výhodou zhlukového plánovača je to, že umožňuje amortizovať čas prerušenia časovača v priebehu niekoľkých prerušení zariadenia, čím sa znižujú režijné náklady. Druhou výhodou je, že niektoré zariadenia, ako napríklad sieťové rozhrania, sú inherentne zhlukové, a preto je žiadúce pracovať s celým zhlukom, radšej ako len s prvým prerušením zo zhluku. Prípadne len s prvým a vynechaním ďalších, tak ako by to robil striktný softvérový plánovač [19].

2.5.3 Hardvérový plánovač

Hardvérová technika plánovania je založená na filtrovaní signálov žiadostí o prerušenie (IRQ) z toku žiadostí o prerušenie, ešte predtým ako sa dostanú do riadiacej jednotky prerušení v procesore. Hlavnou výhodou sú nulové režijné náklady tohto riešenia. Vzhľadom na dizajn je toto riešenie plánovača prerušení v porovnaní s predošlými variantami najjednoduchšie. Pracuje paralelne s hlavným CPU a tým vôbec neznižuje jeho efektivitu. Existujú dve možnosti implementácie hardvérového plánovača. Prvá pomocou ďalšieho mikrokontroléru. Druhá pomocou modifikovanej verzie procesora implementovanej v FPGA technológii. Detaily sú popísané v literatúre č. [19]. Algoritmus v tejto časti sa vzťahuje na obe verzie. Hardvérový plánovač používa počítadlo, ktorého hodnota sa automaticky znižuje konštantnou rýchlosťou. Keď príde prerušenie, existujú dve možnosti:

1. Počítadlo je na nule. V tomto prípade sa počítadlo resetuje na inicializačnú hodnotu a prerušenie je propagované do CPU.

2. Počítadlo nie je nulové. V tomto prípade je iba oznámený príchod prerušenia. Keď počítadlo dosiahne nulu nastane prvý prípad. Neexistuje však žiadna dodatočná fronta. Ak príde niekoľko prerušení počas toho ako sa znižuje stav počítadla, do CPU sa doručí iba jedno prerušenie v okamihu, keď počítadlo dosiahne nulu. Frekvenciu odpočítavania a počiatočnú hodnotu počítadla je možné zmeniť softvérovo.

2.5.4 Plánovanie viacerých zdrojov prerušenia

Plánovanie viacerých zdrojov prerušenia pomocou striktného softvérového plánovača je jednoduché s využitím replikácie hardvéru alebo softvéru pre každú linku prerušenia. Zhlukový plánovač predstavuje zaujímavé možnosti pre optimalizáciu, použitím jedného periodického časovača na vynulovanie viacerých počítadiel pre viacero zdrojov prerušení.

Najdôležitejšie je nájsť správnu veľkosť zhluku, aby došlo k rovnováhe medzi réžiou a ochranou pred preťažením. V systéme s viacerými zdrojmi prerušenia existuje viac možností. Najjednoduchší je výber najmenšieho spoločného násobku rýchlostí príchodu zhlukov pre všetky zdroje prerušenia. Pravdepodobne to bude mať za následok veľmi vysokú frekvenciu a tým aj slabý celkový výkon.

Maximálnu povolenú hodnotu môžeme zaokrúhliť na deliteľ najväčšieho čísla, čo umožňuje jedinému pomalému časovaču, aby obsluhoval všetky zdroje prerušenia a zároveň si zachoval primeranú ochranu pred preťažením. Hardvérové časovače poskytované vstavaným systémom častokrát natívne podporujú len určité frekvencie, takže je pravdepodobné, že môže byť nevyhnutné určité zaokrúhlenie, detaily viď literatúra č. [19].

2.5.5 Integrovaný model úloh a prerušení

Tradičný model obsluhy prerušení je silne podporovaný hardvérom, prináša rýchlu reakciu na vonkajšie udalosti a nízke režijné náklady. Z tohto dôvodu je táto metóda používaná vo väčšine operačných systémov pre vstavané a real-time systémy (systémy pracujúce v reálnom čase). Napriek tomu však môže používanie spôsobiť isté ťažkosti.

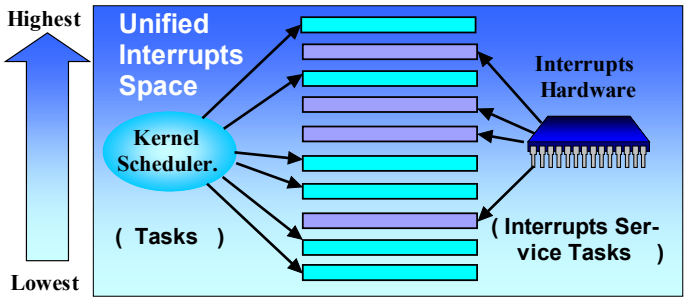
V real-time systémoch nemusí vždy platiť, že časové nároky na ISR sú dôležitejšie ako spracovanie úloh (Tasks) operačného systému. Nároky na čas odozvy niektorých ISR môžu byť dokonca v rovnakej kategórii ako úlohy s vysokou aktivačnou frekvenciou. V tomto prípade sa môžu oba priestory priorít vzájomne ovplyvňovať. Úlohy operačného systému s vysokou prioritou sú pod interferenciou hardverových prerušení s nízkou prioritou. Hardvér

obslujujúci prerušenia je zodpovedný za plánovanie ISR podľa hardvérových priorít, zatiaľ čo úlohy sú plánované jadrom podľa ich softvérových priorít. Model integruje prioritnú úroveň úloh a prioritnú úroveň prerušení do jedného uniformného priestoru. Vytvára taktiež jednotný mechanizmus plánovania a synchronizácie.

Princíp obsluhy

Systém spočíva v integrácii oboch typov asynchrónnych činností (úloh a prerušení) prostredníctvom zjednoteného mechanizmu synchronizácie a plánovania. Integrácia synchronizačného mechanizmu sa dosiahne skrytím prerušení na najnižšej úrovni jadra, ktoré ich premení na synchronizačné udalosti, pomocou abstrakcií komunikácie a synchronizácie medzi úlohami. S týmto modelom sa obslužné rutiny prerušení (ISR) stanú obslužnými úlohami prerušení (ISTs) a zostanú nečinné len do výskytu prerušenia. V tomto integrovanom modeli môže dôjsť k zablokovaniu IST napríklad čakaním na semafore. Keď dôjde k prerušeniu, najskôr univerzálna ISR na najnižšej úrovni jadra vykoná nastavenia potrebné na to, aby bolo možné spustiť samotný IST. Tento prístup poskytuje abstrakciu, ktorá eliminuje rozdiely medzi ISR a úlohami.

Existencia jediného typu asynchrónnych udalostí a jednotného synchronizačného a komunikačného mechanizmu medzi úlohami a preruseniami ponúka značné výhody. Zjednotenie mechanizmu synchronizácie je len potrebným, ale nie dostatočným krokom. Integrovaný mechanizmus, znázornený na obrázku č. 2.3, obsahuje jednotný a flexibilný priestor dynamických priorít pre všetky aktivity, ktoré sú aktivované hardvérovými alebo softvérovými udalosťami. Táto schéma tak umožňuje priradenie priorít všetkým činnostiam real-time systému v súlade s ich časovými nárokmi.



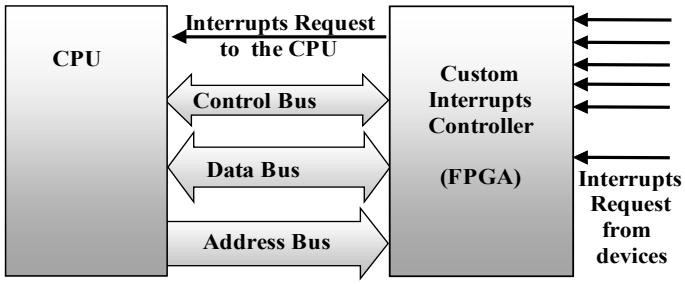
Obr. 2.3: Priority integrovaného modelu prerušení a úloh [13].

Výhodou je taktiež možnosť riešiť situácie vzniku preťaženia z dôvodu prerušení pomocou niektorých plánovacích techník, ako sú napríklad sporadické servery.

Hardvérová implementácia

Návrh implementácie je realizovaný pomocou technológie FPGA, nakoľko poskytuje potrebnú flexibilitu. Týmto spôsobom sa zodpovednosť za plánovanie bežných úloh (aktivovaných softvérovo) a IST (aktivovaných hardvérom) prenesie na plánovač jadra a radič prerušení (CPIC), viď obrázok č. 2.4. Oba komponenty spolupracujú na dosiahnutí úplne integrovaného systému priorít. V rámci tejto konfigurácie môže procesor do sady príkazových registrov CPIC zapisovať dynamicky priority každej žiadosti o prerušenie a aktuálnu úroveň priority. Priorita práve vykonávanej úlohy bude vždy rovnaká ako priorita aktuálnej

úrovne prerušenia. Navrhovaný CPIC potom bude mať za cieľ doručiť žiadosť o prerušenie na CPU len vtedy, ak je jej priorita vyššia než aktuálna úroveň prerušenia.



Obr. 2.4: Radič prerušení implementovaný pomocou technológie FPGA [13].

Z obrázku č. 2.4 je možné vidieť, že riadiaca zbernica medzi CPU a FPGA podporuje protokol žiadosť/potvrdenie prerušenia CPU. Dátová a adresová zbernica umožňuje konfiguráciu CPIC. Viac informácií o tejto technike je možné sa dočítať v literatúre [13] a [15].

Dva najdôležitejšie moduly sú manažment prerušení jadra (KRNLINT) a vrstva abstrakcie prerušení hardvéru (INTHAL). KRNLINT obsahuje hardvérovo nezávislý kód, zatiaľ čo INTHAL obsahuje hardvérový kód. Tieto dva hlavné moduly sú doplnené nízkoúrovňovým synchronizačným a komunikačným modulom. Úlohou KRNLINT je poskytnúť nízkoúrovňové mechanizmy, ktoré umožňujú zvyšku systému spracovávať prerušenia s uplatnením rovnakých pravidiel plánovania a synchronizácie ako tie, ktoré sa používajú pre real-time úlohy. Komponent INTHAL poskytuje manažment prerušenia na najnižšej úrovni. Je zodpovedný za tie aspekty, ktoré závisia od hardvéru vyvolávajúceho prerušenia, takže systém sa stáva čo najviac nezávislý od architektúry počítača. Podrobný popis jednotlivých komponent je dostupný v literatúre [14].

2.5.6 Prioritný časový subsystém TimerShield s vysokým rozlíšením

Ide o techniku, ktorá selektívne spomaľuje obsluhu prerušení časovača s nižšou prioritou zatiaľ čo sa vykonáva úloha s vysokou prioritou. Táto technika je zameraná na úroveň operačných systémov a využíva vyššiu úroveň abstrakcie v porovnaní s predošlými technikami. Technika bola publikovaná v literatúre č. [18].

Moderné real-time operačné systémy využívajú veľmi presné časovače s využitím rôznych plánovačov (napríklad blokovanie procesu určitý počet nanosekúnd). Je to možné vďaka integrácii dedikovaných, jednorazových časovačov s vysokým rozlíšením do moderných procesorov, a umožneniu operačným systémom prácu s nimi pomocou subsystémov.

Príkladom podsystému v jadre systému Linux je *hrtimer*, ktorý udržiava množinu všetkých neobslúžených softvérových časovačov v systéme. Problém môže vznikať z dôvodu fungovania prerušovacieho hardvéru, nakoľko prerušenia generované časovačmi sú vykonávané s najvyššou prioritou, bez ohľadu na prioritu procesu, ktorý vytvoril časovač. V dôsledku toho môžu prerušenia generované časovačmi úloh nízkej priority predbehnúť procesy s vyššou prioritou. Situáciu zhoršujú systémy založené na POSIX, ktoré neobmedzujú limit a počet vytvorených časovačov. To má za následok zvýšenie času odozvy a zníženie výkonu úloh s vysokou prioritou.

TimerShield je časový subsystém pre real-time operačné systémy s fixnou prioritou. Poskytuje rovnaké primitívy pre správu časovača (vytvorenie, vymazanie, expirácia) ako Li-

nuxový *hrtimer*. Naviac však zavádza pojem priority časovačov nastavovaný užívateľskými procesmi. Plánovaním procesu s určitou prioritou sa pred reprogramovaním hardvérového časovača „maskujú“ všetky časovače procesov s nižšou prioritou. Pre minimalizovanie pridaných režijných nákladov sa využívajú špecializované dátové štruktúry.

ARM procesory majú taktiež časovač s vysokým rozlíšením. Operačný systém jednorazovo nakonfiguruje časovač, ktorý s každým cyklom znižuje počítadlo, a ak počet dosiahne nulu je vyvolané prerušenie. Model prerušení sa v Linuxe rozdeľuje na dve časti: krátku časovo kritickú *hornú polovicu* a dlhšiu *spodnú polovicu*. Aplikácia časovačov v systéme Linux je delená na časovače s vysokým rozlíšením (používané napr. pre udalosti s vysokou presnosťou) a časovače s nízkym rozlíšením (napr. limit pre signalizáciu chybových podmienok I/O). Časovače s nízkym rozlíšením sú často zrušené ešte pred expiráciou.

Subsystém Hrtimer

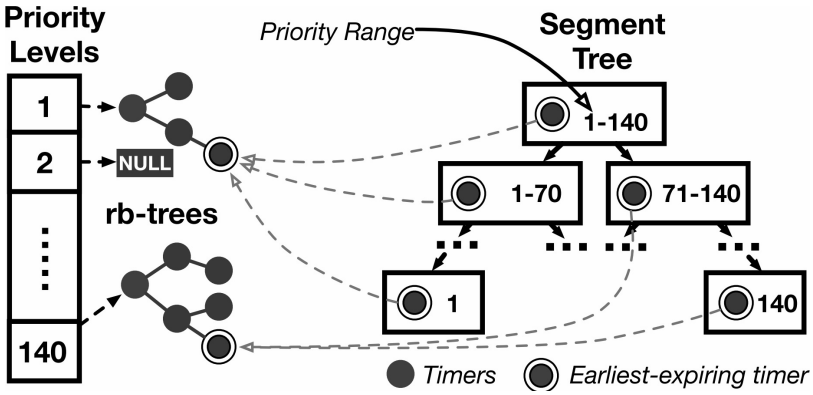
Časovače s vysokým rozlíšením sú obsluhované prostredníctvom *hrtimer* subsystému. Každý hrtimer je v jadre reprezentovaný štruktúrou obsahujúcou okrem iného absolútnu dobu expirácie a ukazovateľ spätného volania. Tieto štruktúry udržiavajú záznamy vo vyváženom strome (rb-strom) v poradí podľa najmenšej expiračnej doby. Umožňuje vkladanie $O(\log\ n)$ a odstraňovanie $O(1)$ s prístupom k najkratšiemu časovaču. Pre každé jadro je zvyčajne udržiavaný jeden strom. Pri vkladaní alebo vymazávaní z rb-stromu jadro kontroluje, či hardvér časovača vyžaduje preprogramovanie. Subsystém hrtimer tvorí základ pre implementáciu systémového volania POSIX `clock_nanosleep()`, pre pozastavenie aktuálne vykonávaného procesu. Toto je dôležité v real-time systémoch, kde sa používa na presné časové periodické spúšťanie riadiacich úloh. Avšak, *hrtimer* subsystém pri preprogramovaní nezohľadňuje priority procesov, ktoré časovač vytvorili. V dôsledku toho dochádza k prerušeniu aktuálnej úlohy s vysokou prioritou aj v prípade, ak bol časovač vytvorený úlohou s nižšou prioritou.

Subsystém TimerShield

TimerShield rozširuje architektúru *hrtimer* o dátové štruktúry umožňujúce sledovať prioritu procesu, ktorý ho vytvoril. Dátový typ údajov je rozšírený o pole s prioritami a umožňuje tak uložiť plánovaciu prioritu procesu, ktorý časovač vytvára. Modifikáciou je vytvorenie poľa viacerých rb-stromov, pričom každý je pre jednu úroveň priority, čím sa redukuje efektivita vyhľadávania podľa priority späť na $O(1)$. Linux podporuje 140 prioritných úrovní procesov a preto každý hardvérový časovač má pole 140 odpovedajúcich rb-stromov asociovaných s TimerShield.

TimerShield taktiež zavádza nové rozhranie pre dotazy na najrýchlejšie expirujúci časovač podľa priority. Kľúčovou výzvou pre zistenie priority je rýchlosť. Technika využíva segmentový strom pre nízku spotrebu pamäte a rýchlejšiu aktualizáciu. Ide o kompletný binárny strom, kde každý uzol predstavuje súvislý interval celkového priestoru pre vyhľadávanie. Každý uzol predstavuje rozsah prioritných hodnôt a ukladá ukazovateľ na najrýchlejšie expirujúci časovač v tomto rozsahu priorít ako je vidieť na obrázku č. 2.5. Strom segmentov podporuje dve základné operácie: aktualizáciu a otázky. Aktualizácia sa vykoná napríklad pri vypršaní najkratšieho časového limitu. Otázky na časovač prebiehajú smerom od koreňa nadol.

TimerShield modifikuje expiračnú logiku tak, aby sa obslúžili iba časovače, ktorých priorita je vyššia alebo rovná aktuálnemu procesu. Pri vytvorení časovača sa najskôr získa priorita aktuálneho procesu a časovač sa vloží do rb-stromu, ktorý je priradený k tejto



Obr. 2.5: Ilustrácia dátových štruktúr používaných v technike TimerShield [18].

úrovni priority. Vymazanie časovača jednoducho znamená jeho odstránenie z rb-stromu. Po expirácii časového limitu, TimerShield obslúži iba časovače s vyššou alebo rovnakou prioritou aktuálne spusteného procesu. Akékoľvek expirované časovače s nižšou prioritou sa ignorujú, čím sa zabráni zbytočným oneskoreniam. Nakoniec TimerShield preprogramuje hardvérový časovač.

Pre časovače plánovača je potrebné zabezpečiť aby tieto prerušenia neboli nikdy maskované. Preto sú využité kritické časovače s najvyššou systémovou prioritou, zabezpečujúce okamžitú obsluhu. V systéme Linux sú však plánovače hierarchicky usporiadané do tried pretože každé prereušenie plánovača nemusí mať najvyššiu systémovú prioritu.

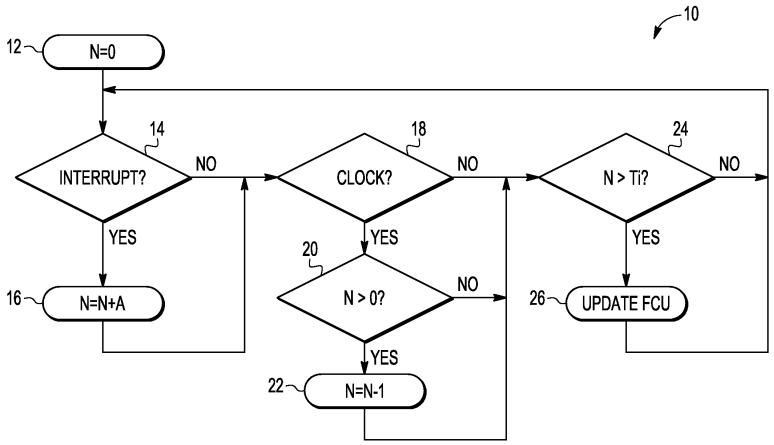
Hlavným predpokladom v subsystéme TimerShield je, že priorita úlohy, ktorá vytvára časovač, zostáva nezmenená až dovtedy, kým sa časovač nespustí. TimerShield rozširuje subsystém *hrtimer* tak, aby časovač softvéru s nízkou prioritou prerušenia nikdy neblokoval úlohy s vysokou prioritou. Viac podrobností o tejto technike je možné sa dočítať v literatúre [18].

2.5.7 Detekcia vysokofrekvenčných prerušení

Táto technika priamo nezamedzuje výpočtovému preťaženiu ale detekuje chyby v dôsledku zvýšenej miery výskytu prerušení. Ide o patent firmy NXP [20], ktorý umožňuje detekovať a porovnať priemerný výskyt žiadostí o prerušenie s užívateľsky definovaným limitom. Takýto chybne zvýšený výskyt prerušení sa môže vyskytnúť v dôsledku bežných porúch, ako sú porušené kontakty alebo vodiče. Vysokofrekvenčné prerušenia sa zvyčajne spravujú pomocou in-line systémov, medzi periférnymi zariadeniami a procesorom, ktoré detekujú a blokujú nadmernú aktivitu prerušení. Monitorovanie prerušení neovplyvňuje spracovanie prerušení softvérom, ale ide iba o systém, ktorý indikuje prekročenie nakonfigurovaného limitu výskytu prerušení.

Princíp techniky spočíva v inkrementácii počítadla po príchode každého prerušenia o definovanú hodnotu. Počítadlo je naopak znížené o jedna pri nástupnej hrane hodín CLK v prípade ak je hodnota väčšia ako nula. Rýchlosť prerušení je tak vymedzená hodnotou počítadla. V prípade ak miera prerušení prekročí prahovú hodnotu, aktualizuje sa jednotka zhromažďovania porúch (FCU). FCU môže následne klasifikovať akciu buď iba ako varovanie, ktoré nevyžaduje žiadne ďalšie reakcie, alebo ako vážnu poruchu vyžadujúcu okamžitú reakciu (napr. vypnutie periférneho zariadenia). FCU na záver obnoví stav počítadla na nulovú hodnotu.

Nasledujúci obrázok č. 2.6 schematicky ilustruje spôsob detekcie vysokofrekvenčných prerušení. Počítadlo (N) sa najskôr resetuje na nulu. V prípade detekcie aspoň jedného prerušenia, sa údaj N zvýši o konfigurovateľnú hodnotu (A), a následne sa vyhodnotí stav hodín. Ak nedošlo k žiadnemu prerušeniu, metóda pokračuje vyhodnotením stavu hodín bez zmeny hodnoty N. Zmena stavu hodín je definovaná ako nábežná hrana. V prípade, ak je detekovaná zmena hodín a zároveň je hodnota N väčšia ako nula, veľkosť N je znížená o jedna. Následne sa vyhodnotí počet prerušení N, aby bolo možné určiť, či došlo k prijatiu nadmerného počtu prerušení. V prípade ak N prekročí prahovú hodnotu Ti, definovanú užívateľom, dôjde k aktualizácii FCU a následne sa metóda vráti do stavu, kedy monitoruje príchod ďalších prerušení. Ak nedošlo k prekročeniu prahovej hodnoty, metóda prejde bezprostredne do tohto stavu. Taktiež je možné spojiť vyhodnocovanie stavu hodín a prekročenie prahovej hodnoty do jednej operácie.

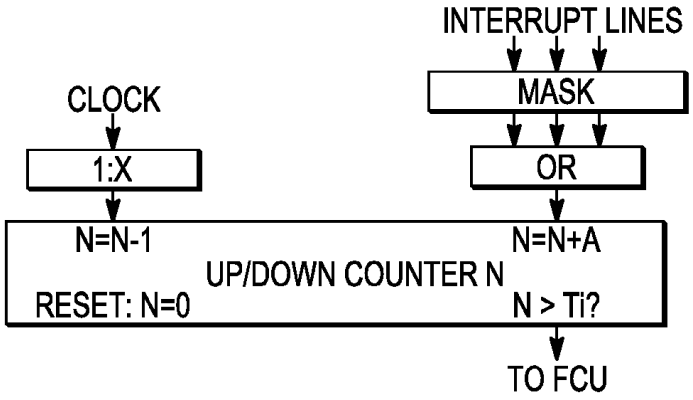


Obr. 2.6: Vývojový diagram spôsobu detekcie vysokofrekvenčných prerušení [20].

Táto metóda sa využíva najmä pre vysokofrekvenčné prerušenia založené na asynchrónnom systéme, kedy jednotlivé zmeny nastávajú bez ohľadu na stav hodinového signálu. V synchrónnej modifikácii je možné vykonávať detekciu príchodu prerušenia a kontrolu stavu hodín súbežne.

Popis činnosti je možné v jednoduchosti popísať nasledujúcim spôsobom. Zdroj generuje prerušenie, ktoré sa prenáša jednou z viacerých prerušovacích liniek, viď obrázok č. 2.7. Zariadenie umožňuje vstupovať viacerým prerušovacím linkám. Linky sú selektívne aktivované alebo deaktivované maskou. Povoľovacia maska predstavuje konfigurovateľný register. Signály prechádzajú funkciou OR, ktorá vytvára indikátor prerušenia. Hodiny môžu byť delené deličkou v deliacom pomere $1 : X$, pričom pomer je konfigurovateľný priamo v hardvéri. Indikátor prerušenia zvyšuje počítadlo N smerom nahor. Hodinový signál znižuje počítadlo smerom nadol. Ak je počítadlo nulové, hodiny už viac počet neznižujú. Ak počet prekročí prahovú hodnotu Ti odosiela sa indikačný signál o nadmernej frekvencii do jednotky FCU. FCU môže na podnet reagovať rôznymi spôsobmi.

Princíp konfigurácie závisí od počiatočného stanovenia jednotlivých hodnôt. V prípade ak je napríklad pomer hodín X nastavený na 1 (tj. bez deliaceho pomeru), hodnota A nakonfigurovaná na 4 a prahová hodnota Ti nastavená na hodnotu 4, je povolený príchod jedného prerušenia každé 4 takty hodinového signálu. Príchod prerušenia nastaví N na hodnotu 4, a následné 4 hodinové takty ju znížia na nulu. Pri nastavení hodnoty A na 1 a prahovej hodnoty Ti na hodnotu 4, je povolený výskyt štyroch prerušení v bezpro-



Obr. 2.7: Schematický pohľad na realizáciu detektoru vysokofrekvenčných prerušení [20].

strednej následnosti pre každé 4 takty hodinového signálu. Viac podrobností o možnostiach tohto detektora a jeho konfigurácie je možné nájsť v literatúre [20].

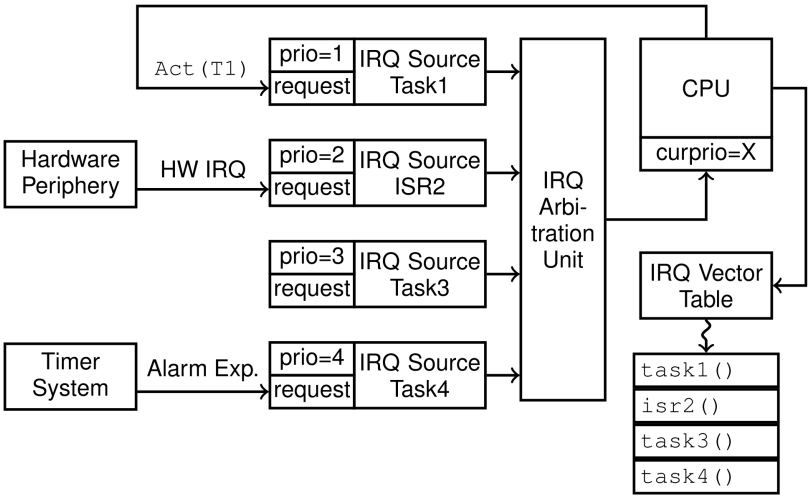
2.5.8 Technika využívajúca jadro Sleepy Sloth

Táto technika využíva generalizovanú abstrakciu vláken, pričom umožňuje hardvéru vykonávať plánovanie. Ruší rozlišovanie medzi vláknami a ISR. Vlákna môžu byť obslúžené rovnako efektívne ako obsluha prerušenia a obsluhy prerušenia môžu byť zároveň naplánované s rovnakou flexibilitou ako vlákna. Vo všeobecnosti je hlavný rozdiel medzi vláknami a ISR ten, že vlákna sú plánované softvérovými mechanizmami, zatiaľ čo ISR sú spravované hardvérom. Vykonávanie vláken môže byť zablokované a obnovené, kým ISR musia bežať až do ukončenia. Ich prioritné priestory sú bežne vzájomne oddelené, pričom ISR majú väčšiu prioritu.

Sémantika riadiaceho toku spolu s prioritou by nemala byť definovaná zdrojom udalostí ale na základe požiadaviek real-time aplikácie. Niektoré metódy využívajú propagáciu ISR na podprocesy, čo zapríčiňuje presunutie riadenia a plánovania z hardvéru na operačný systém. ISR v tom prípade pracujú tak, že vždy aktivujú iba ich obslužné vlákno a ukončia sa. Tento prístup využíva napríklad OS Solaris.

Jadro Sloth však využíva opačný prístup, kedy sú vlákna spracovávané ako ISR. Aktivácia vlákna prebieha spúšťaním odpovedajúceho prerušenia z jadra operačného systému. Jadro Sloth sa pritom spolieha na priestor priorít prerušení riadený hardvérom, ktorý vykonáva ich plánovanie, viď obrázok č. 2.8. Výhodou je ich vykonávanie v rámci jedného uniformného priestoru priorít. Rozšírenie metódy umožňuje kombinovať výhody riadenia toku Sloth s funkcionalitou blokovania jednotlivých úloh, nakoľko pôvodné prerušenia s viacerými úrovňami pracujú striktne LIFO zásobníkovým spôsobom (prerušenie vyššej priority sa dokončí skôr ako prerušenie s nižšou prioritou). Blokovanie umožňuje pozastavenie alebo opätovné obnovenie úlohy. Podporu pre blokovanie úloh poskytuje pripojenie prologu ku každej úlohe. Prolog sa vykonáva vždy pri prvom pridelení radičom prerušení ale aj pri opätovnom spustení po pozastavení úlohy.

Systém Sloth pri riadení úloh využíva statickú konfiguráciu priorít známu už pri kompilácii. Každá úloha je navrhnutá ako obsluha prerušenia, pričom jej IRQ obsahuje príslušnú prioritu. Odpovedajúci bit žiadosti o prerušenie je nastavovaný softvérovo. Radič IRQ rozhoduje o preempcii na základe aktuálnej priority CPU a priorít čakajúcich IRQ. V systéme Sloth neexistuje žiadny rozdiel medzi úlohami a ISR. Jadro nerozlišuje, či bolo IRQ vyvo-



Obr. 2.8: Dizajn systému Sleepy Sloth, využívajúceho obsluhu prerušení pre implementáciu vláken [16].

lané hardvérom alebo softvérom. Podmienkou však je, aby bol v systéme počet dostupných priorít aspoň tak veľký aký je počet vláken a ISR. Taktiež je potrebné vopred pri kompilácii určiť graf priorít, ktorý obsahuje informácie o tom, ktorý riadiaci tok má byť uprednostnený na základe priority pred iným. To umožňuje vynechať v prologu kontrolu situácií, ktoré nikdy nemôžu nastať. Jednotlivé implementačné detaily prologu, pozastavenia, blokovania a obnovenia úlohy je možné nájsť v literatúre [16].

2.5.9 Záťažovo adaptívny monitorovací hardvér pre správu prerušení riadený prioritou

V niektorých prípadoch sa môže stať, že interval medzi príchodom prerušení klesne pod čas obsluhy prerušenia. To zapríčiní, že nezostane žiaden procesorový čas pre vykonávanie hlavného programu. Obzvlášť to platí pre systémy (tzv. real-time, RT), ktorých dokonalosť je založená na správnosti a aktuálnosti výstupov. V týchto systémoch je každá udalosť zvyčajne spojená s výpočtovou jednotkou (tzv. úlohou) zodpovednou za správnu reakciu na príslušnú udalosť. Existujú dva základné typy RT úloh:

- silné - časové obmedzenia musia byť striktne splnené
- slabé - pri ktorých stačí, že sú časové obmedzenia aspoň niekedy splnené

Vlastnosti tejto metódy sú:

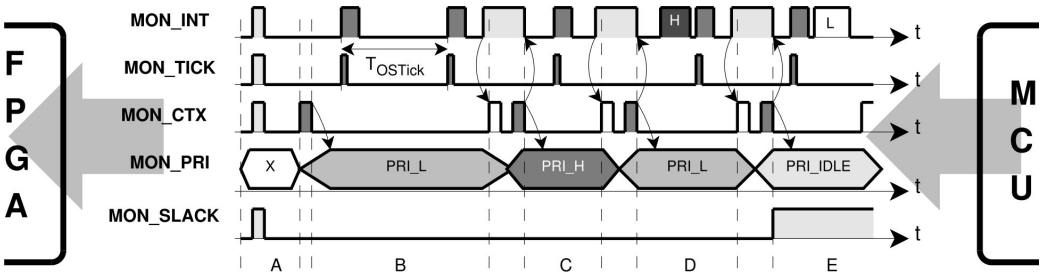
- Dosiahnuteľnosť: snaha ponúknuť riešenie problému preťaženia v dôsledku nadmernej frekvencie prerušení na báze nástrojov bežne dostupných na trhu, použitím komponentov ako sú napríklad MCU/FPGA a operačné systémy.
- Všeobecnosť: riešenie má za následok architektúru, ktorá je dostatočne všeobecná a abstraktná od konkrétnych architektúr a výrobcov.
- Jednoduchosť: riešenie znižuje potrebu zmeny existujúcich komponentov na minimum.
- Adaptabilita: riešenie je schopné prispôsobiť mieru obsluhy prerušení aktuálnej záťaži platformy a obmedzeniam vyplývajúcich zo špecifikácií systému.

Keďže predpokladá, že softvér bežiaci na jednotke MCU by mal byť bezpečný a časovo kritický, je nežiadúce posielať do MCU prerušenia, kým je preťažené alebo vykonáva úlohu s vyššou alebo rovnakou prioritou ako je priorita prerušenia. Manažment prerušení je realizovaný v FPGA, ktorý je dizajnovaný na rozpoznanie a pozastavenie prerušení, prípadne na uvoľnenie nežiadúceho prerušenia. Detaily techniky sú uvedené v literatúre č. [21].

Generovanie signálov

Pre sledovanie softvérového zaťaženia je použité jednoduché rozhranie, ktoré je realizovateľné akýmkoľvek komerčným výrobkom COTS (Commercial off-the-shelf). Rozhranie je zložené zo signálov `MON_INT`, `MON_TICK`, `MON_CTX`, `MON_PRI` a `MON_SLACK`, ktoré sú produkované na strane MCU a pre účely monitorovania distribuované do FPGA.

Generovanie signálov začne, keď je jadro pripravené na prevádzku. Musia byť inicializované štruktúry RT operačného systému a nakonfigurovaný hardvérový časovač (TIM), ktorý produkuje periodické prerušenie T_{OSTick}. Generovanie štartu je signalizované do FPGA, vytvorením krátkeho impulzu v úrovni HIGH na každom zo signálov `MON_INT`, `MON_TICK`, `MON_CTX` a `MON_SLACK` viď obrázok č. 2.9. To umožňuje synchronizovať FPGA s MCU a následne spustiť monitorovanie signálov. Je to však za cenu zvýšenia času spustenia systému vopred definovaným konštantným počtom cyklov procesora.



Obr. 2.9: Ilustrácia priebehu monitorovacích signálov rozhrania [21].

Taktiež musí byť modifikovaný prológ (epilóg) prerušenia, ktorý nastaví signál `MON_INT` do úrovne HIGH (LOW) len na začiatku (konci) tela, rutina obsluhy prerušenia (ISR). To umožňuje počas behu programu monitorovať dobu vykonávania ISR a čas začatia a ukončenia ISR. Vykonanie ISR hardvérového časovača TIM je signalizované generovaním krátkeho impulzu na linke `MON_TICK`. To umožňuje FPGA sledovať čas operačného systému a pozorovať tak javy ako je nestabilita.

Keďže interné zdroje sú veľmi obmedzené (napríklad v porovnaní s desktopovým počítačom) nie sú povolené vnorené ISR a doba ich vykonávania musí byť čo najkratšia, aby nedochádzalo k oneskorenému vykonávaniu po sebe nasledujúcich ISR. Pre zaistenie monitorovania množstva času, ktorý CPU venuje ukladaniu a obnovovaniu kontextu úloh sa signál `MON_CTX` pri začatí (skončení) nastaví do úrovne HIGH (LOW). Signál `MON_PRI` sa používa na monitorovanie priority bežiacej úlohy. Signál `MON_PRI` je nastavovaný vo fáze obnovovania kontextu hneď ako je z kontextu prevzatá priorita úlohy.

Ak v systéme nie je žiadna čakajúca úloha, tak sa kontext prepne do režimu nečinnosti. Signál `MON_PRI` je tak nastavený na hodnotu PRI_IDLE. Taktiež hneď pri nastavení signálu `MON_CTX` na hodnotu HIGH je aj signál `MON_SLACK` nastavený na hodnotu HIGH.

Princíp fungovania platformy určenej pre riadenie prerušení založenej na monitorovaní činnosti CPU je opísaný v nasledujúcom algoritme č. 1.

Algoritmus 1: Popis činnosti monitorovacej funkcie

```
1  if nastane prerušenie alebo zmena signálu MON_* then
2      if priorita prerušenia > MON_PRI then
3          odošli prerušenie do MCU
4          return
5      else if nie je prekročený maximálny povolených počet prerušení v kritickom
         plávajúcom okne then
6          odošli prerušenie do MCU
7          return
8      else if signál MON_SLACK je v hodnote HIGH then
9          odošli prerušenie do MCU
10         return
11     else
12         odkladaj prerušenia pokiaľ nebude systém pripravený ich obslúžiť
13         return
14     end if
15 end if
```

Kapitola 3

Realizačná architektúra

Táto kapitola sa venuje výberu vhodnej platformy pre riešenie projektu. Popisuje jej základné parametre. Zameriava sa na možnosti čipu zvolenej architektúry a jeho činnosť. V kapitole je taktiež spomenutá možnosť využitia externých komunikačných rozhraní. Kapitola je zakončená spôsobom vývoja softvéru pre zvolenú platformu a spôsobom programovania a testovania.

3.1 Platforma FITkit

Platforma FITkit je určená pre podporu výučby v predmetoch orientovaných na hardvér na Fakulte informačných technológií Vysokého učení technického v Brne. Ide o samostatný hardvér obsahujúci mikrokontrolér s nízkym príkonom, hradlové pole FPGA (Field Programmable Gate Array). Dôležitým aspektom je možnosť využitia pokročilého reprogramovateľného hardvéru na báze hradlových polí. Platformu FITkit je tak možné takmer neobmedzene konfigurovať nie len softvérovo, ale taktiež po hardvérovej stránke. K tomu slúžia vhodné programovacie jazyky (ako je napr. VHDL), čím návrh softvérovej a hardvérovej stránky prebieha do značnej miery obdobne [22].

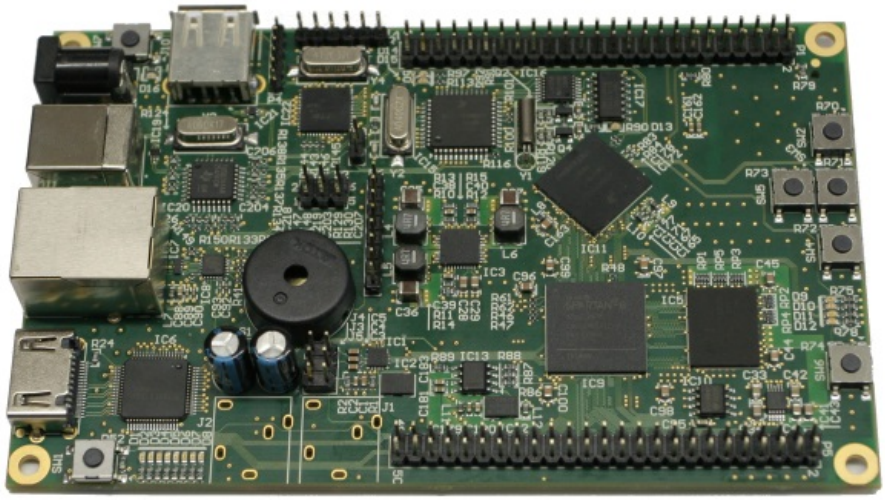
Platforma je prispôsobená širokým možnostiam využitia, pričom poskytuje vysokú mieru variability, či už z hľadiska vstupno-výstupných rozhraní, ale aj z hľadiska konfigurácie hardvéru a softvéru. Platforma existuje vo viacerých realizáciách od verzie 1.0, 1.2, 2.0 až po verziu 3.0.

3.2 Architektúra platformy FITkit 3.0 Minerva

Verzia FITkit 3.0 Minerva (ďalej len FITkit 3.0), je najnovšia verzia platformy FITkit a bola zvolená na riešenie projektu. Platforma obsahuje hlavný výkonný 32 bitový mikrokontrolér KINETIS MK60DN512VMD10 s nízkym príkonom, ktorý vyrába firma Freescale Semiconductor. Mikrokontrolér je založený na harvardskej architektúre, tj. pamäť pre dáta a pre program sú navzájom oddelené. Programová pamäť a dátová pamäť nemajú rovnako dlhé dátové slovo. Mikrokontrolér obsahuje procesor ARM Cortex-M4 s frekvenciou 100 MHz, 512 KB pamäť FLASH pre program, 128 KB pamäť SRAM pre dáta a rozhrania CAN, I2C, SPI, UART a USB.

Vývojový kit Minerva obsahuje taktiež FPGA Spartan 6 (XC6SLX9) vyrábaný firmou XILINX Inc., ktorý zahŕňa 1430 slices, 90kb distribuovanej RAM, 32 dvojportových BRAM s kapacitou 18 kB a 200 I/O pinov. Okrem spomenutých obvodov, obsahuje FITkit

3.0 taktiež: MCU Freescale HCS08 konkrétne MC9S08JM60, prevodník USB/COM FTDI Vinculum-II (VNC2), pamäťový modul DRAM ISSI 512 Mbit typu IS43DR16320B taktovaný na frekvencii 333 MHz, pamäť NAND-FLASH ST 4 Mbit typu M25P40 taktovanú na 25 MHz, stereo audio kodek Freescale SGTL5000, sériový 4-port USB 2.0 Hub Texas Instruments typu TUSB2046B s rýchlosťou 12 Mbit/s, DVI rozhranie TFP410PAP Texas Instruments, LAN radič SMSC 10/100 Ethernet typu 8720A a radič napájania TPS65251RHA od firmy Texas Instruments. Mikrokontrolér HCS08 s 8 bitovou šírkou zbernice, 64 KB FLASH pre program a 4 KB RAM pre dáta je možné využívať ako prostriedok pre ladenie a debuging ostatných komponentov Minerva kitu. Je taktovaný frekvenciou 48 MHz.



Obr. 3.1: Vývojová platforma FITkit 3.0 Minerva [8].

Platforma poskytuje širokú konektorovú výbavu: konektor USB typ B pre pripojenie k PC, konektor pre externé napájanie, HDMI konektor obsahujúci DVI interface, ethernetový konektor RJ45, USB konektor typu A, konektor pre SD kartu umiestnený zo spodnej strany kitu, dva audio konektory IN/OUT a rozširujúce konektory (pinheaders). V porovnaní s predchádzajúcimi verziami, verzia 3.0 neobsahuje LCD displej ani maticovú klávesnicu.

FITkit 3.0 obsahuje zároveň zabudovaný reproduktor, sadu ovládacích tlačidiel, sadu indikačných LED diód a oddeľovače s linkovými budičmi typu SN74HCT125D. Kit je taktiež možné rozšíriť ďalšími modulmi. Ide o modul bezdrôtovej komunikácie RF 2.4 GHz v bezlicenčnom pásme obsahujúci integrovanú anténu a 20 pinový konektor pre pripojenie k FITkitu. Ďalší modul obsahuje RJ45 konektor a poskytuje 10 Mbit/100 Mbit ethernetové pripojenie v sieťach LAN/WAN a môže byť pripojený k FPGA pomocou pinheaders konektoru [8].

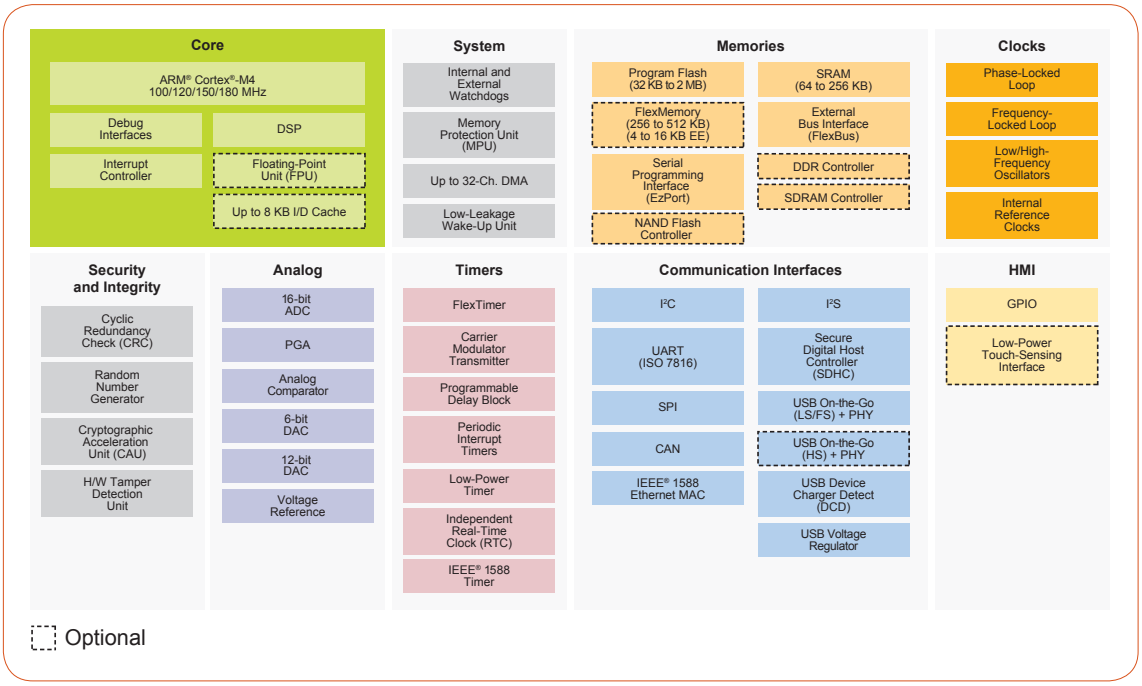
3.3 Mikrokontrolér Kinetis MK60DN512VMD10

Mikrokontrolér Kinetis MK60DN512VMD10 je osadený na vývojovom kite FITkit 3.0. Ide o mikrokontrolér založený na jadre ARM Cortex-M4 s DSP taktovaným na 100 MHz, ktorý poskytuje SIMD rozšírenie a je vybavený pokročilou analógovou integráciou a sériovou komunikáciou, viď obrázok č. 3.2. Je vhodný pre použitie v biomedicínskych monitorovacích

zariadeniach, zber dát v IoT zariadeniach, domácu automatizáciu, riadenie budov či priemyselných zariadení.

Z hľadiska spotreby sa vyznačuje ultranízkou spotrebou energie. Ponúka 10 režimov úrovní napájania a hodín pre optimálnu aktivitu periférií a obnovovacie časy 4 μs prebudenia z režimu Stop. Pamäťové a analógové operácie sú pod hranicou 1.71 V pre predĺženie doby napájania z batérie. Obsahuje low-power časovač pre nepretržitú prevádzku v redukovanom napájacom režime. Obsahuje 512 KB FLASH pamäť a 128 KB SRAM pamäť. Poskytuje dva vysokorýchlostné 16 bitové analógovo-digitálne konvertory s konfigurovateľným rozlíšením, schopné pracovať v jednoduchom alebo diferenciálnom výstupnom režime pre zlepšenie redukcie šumu. Doba konverzie je na úrovni 863 ns. Dva 12 bitové digitálno-analógové prevodníky pre generovanie signálov, pre zvukové aplikácie a tri vysokorýchlostné komparátory.

Mikrokontrolér obsahuje 16 kanálový DMA, obsluhujúci periférie a pamäte, pre zníženie zaťaženia procesora a rýchlejšiu výkonnosť systému a krížový prepínač umožňujúci konkurenčné multi-master prístupy, čo zvyšuje šírku pásma zbernice. Ďalej obsahuje nezávislé flash banky umožňujúce konkurenčné vykonávanie kódu a aktualizáciu firmvéru bez degradácie výkonu, tri FlexTimers časovače s 12 kanálmi, ktoré môžu byť použité pre generovanie PWM signálu. Má štvorkanálový 32 bitový periodický časovač, ktorý poskytuje časovú bázu pre plánovač úloh RTOS alebo spúšťanie ADC konverzie a programovanie čakacích blokov. Obsahuje hardvérovú podporu pre dotykové rozhranie s vysokou citlivosťou, odstraňujúce problém metódy softvérového pollingu.



Obr. 3.2: Bloková schéma mikrokontroléru Kinetis MK60DN512VMD10 zachytávajúca vnútorné členenie a štruktúru komponent [6].

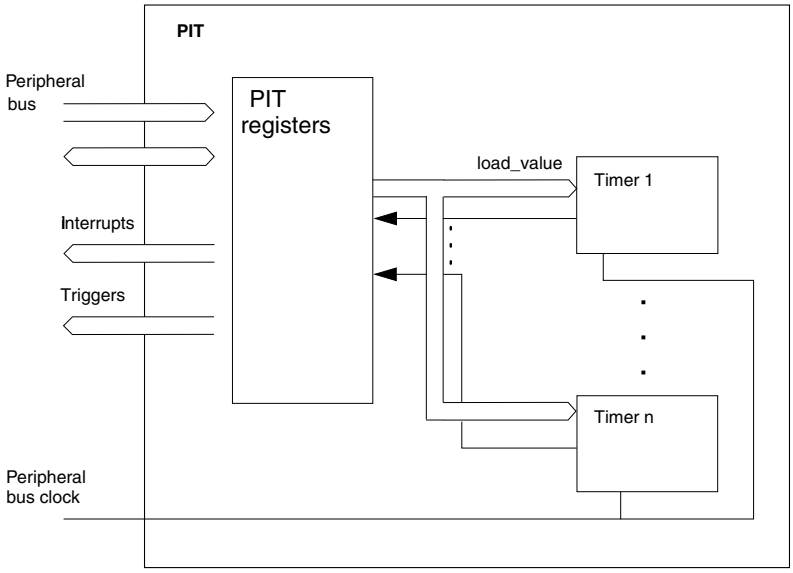
Konektivitu mikrokontroléru dopĺňa IEEE 1588 Ethernet MAC s hardvérovými časovými razítkami poskytujúci presnú synchronizáciu hodín pre priemyselné riadenie v reálnom čase, USB 2.0 s optimalizáciou nabíjacieho prúdu/času pre prenosné zariadenia umožňujúce

dlhšiu životnosť batérie, ďalej päť UART s podporou IrDA vrátane jednej UART s podporou smart kariet, sériové rozhranie I2S pre podporu audia, dva CAN moduly pre možnosť priemyselného pripojenia, tri DSPI a dve I2C rozhrania.

Z hľadiska bezpečnosti zabezpečenia a spoľahlivosti je mikrokontrolér vybavený hardvérovým koprocesorom pre bezpečný prenos a ukladanie dát, ktorý je rýchlejší ako softvérová implementácia a zmenšuje zaťaženie CPU. Podporuje širokú škálu algoritmov: DES, 3DES, AES, MD5, SHA-1, SHA-255. Obsahuje jednotku ochrany pamäte pre všetky master jednotky krížového prepínača, čím zvyšuje spoľahlivosť softvéru. Obsahuje nezávislý časovač COP, ktorý stráži skreslenie hodín. Komponenta externého watchdog monitoru, zabezpečuje pre externé zariadenia bezpečný stav na výstupných pinoch v prípade ak nastane watchdog udalosť [1].

3.3.1 Periodický časovač prerušenia - PIT

Periodický časovač prerušenia je komponent časovača nachádzajúci sa na čipe mikrokontroléru Kinetis MK60DN512VMD10. PIT modul obsahuje pole časovačov, ktoré môžu byť použité k rôznym účelom v rámci mikrokontroléru. Jednotka PIT neobsahuje žiadne externé piny a je súčasťou integrovaných komponent priamo na čipe. Hlavnými úlohami tejto komponenty sú: schopnosť časovačov ovládať generovanie spúšťacích pulzov DMA a schopnosť časovačov generovať prerušenia. Generované prerušenia sú maskovateľné. Konfigurácia vektorov prerušení pre jednotlivé časovače je v rozsahu od 84 pre kanál 0 po 87 pre kanál 3. Jednotlivé čísla IRQ je možné vypočítať ako: *číslo vektoru prerušenia* −16. Komponenta PIT umožňuje nastavenie nezávislých limitov pre jednotlivé kanály časovačov. PIT tvoria konfiguračné registre a štyri nezávislé kanály indexované od 0 po 3, , viď obrázok č. 3.3 .



Obr. 3.3: Bloková schéma jednotky PIT mikrokontroléru Kinetis MK60DN512VMD10 zachytávajúca vnútorné členenie a štruktúru [9].

Hlavný konfiguračný register celého modulu je register PIT_MCR. Tento register zapína alebo vypína hodiny pre všetky kanály časovača PIT a taktiež ovláda nastavenie časovačov v prípade, kedy PIT vstupuje do Debug režimu. Nastavením je možné dosiahnuť zmrazenie aktuálnej hodnoty časovača v Debug režime. To umožňuje pomoc pri vývoji softvéru,

pretože pri zastavení procesoru je možné preskúmať aktuálny stav systému vrátane hodnôt časovača a následne pokračovať vo vykonávaní. Každý kanál časovača má štyri oddelené konfiguračné registre PIT_LDVALn, PIT_CVALn, PIT_TCTRLn a register PIT_TFLGn, pričom písmeno n určuje kanál konkrétneho časovača.

Register PIT_LDVALn s šírkou 32 bitov umožňuje nastavenie štartovacej hodnoty časovej periódy pre vyvolanie prerušenia. Časovač dekrementuje nastavenú hodnotu, kým nedosiahne nulu, následkom čoho vygeneruje prerušenie a znovu načíta pôvodnú hodnotu do registra. V prípade nastavenia novej hodnoty sa zmena v perióde časovača prejaví až po uplynutí časovača. Ak je potrebné zmeniť aktuálnu periódu spusteného časovača, je potrebné časovač najskôr vypnúť, následne nastaviť v registri novú hodnotu periódy a opätovne spustiť časovač. Hodnotu periódy časovača je možné zmeniť aj počas spusteného časovača zápisom do PIT_LDVAL registra, perióda však bude aktualizovaná až po vyvolaní udalosti. Ak je časovač zapnutý, je možné v prípade potreby z registra PIT_CVALn zistiť aktuálnu hodnotu časovača.

Riadiacim registrom konkrétneho kanálu časovača je register PIT_TCTRLn. Register obsahuje Timer Enable bit pre zapnutie alebo vypnutie konkrétneho kanálu časovača. Ďalší bit Timer Interrupt Enable umožňuje zapnúť alebo vypnúť prerušenie z konkrétneho kanálu časovača. Ak je prerušenie v stave čakajúce alebo je nastavený bit PIT_TFLGn[TIF], povolením bitu prerušenia sa okamžite vyvolá prerušenie. Je potrebné tomu predchádzať vynulovaním bitu čakajúceho prerušenia príslušného kanálu časovača v registri PIT_TFLGn[TIF]. Riadiaci register taktiež obsahuje Chain Mode bit, ktorý umožňuje vzájomné zreťazenie jednotlivých časovačov. Zreťazenie vyžaduje expiráciu časovača $n-1$, následkom čoho dôjde k dekrementácii časovača n o 1. Prvý časovač tak už nemôže byť ďalej zreťazený.

Pre indikáciu ukončenia periódy časovača slúži v registri PIT_TFLGn bit TIF, ktorý sa nachádza na pozícii nultého bitu. Príznak je nastavený do logickej úrovne 1 na konci periódy časovača. Nastavený bit TIF spôsobí vyvolanie žiadosti o prerušenie. Zápisom hodnoty 1 je možné príznak vynulovať. Nové prerušenie môže byť vygenerované až po predošlom vynulovaní bitu TIF.

Konfigurácia periódy časovača je závislá od aktuálnej frekvencie časovača PIT modulu. Vzťah $LDVAL = \left(\frac{timer\ period}{clock\ period}\right) - 1$, popisuje výpočet hodnoty časovej periódy registra PIT_LDVALn. V prípade, ak je frekvencia hodín modulu PIT napríklad $50\,MHz$ (tj. perióda $20\,ns$) a požadovaná frekvencia časovača $30\,ms$, bude hodnota konfiguračného registra rovná: $(30\,ms/20\,ns) - 1 = 1499999$ cyklom. Do registra je hodnota následne zapísaná v tvare `0x0016E35F`. Viac informácií je možné nájsť v referenčnom manuáli [9].

3.4 Procesorové jadro ARM Cortex-M4

Procesorové jadro ARM Cortex-M4 je RISC procesor založený na Harvardskej architektúre s oddelenou pamäťou pre dáta a programovou pamäťou. Patrí do architektúrnej kategórie ARMv7-M. Procesor Cortex-M4 používa 32 bitovú architektúru. Interné registre v registrovej sade, dátové cesty a zbernicové rozhrania sú všetky 32 bitové.

Inštrukčná sada je založená na Thumb-2 technológii s podporou zmiešaných 16 bitových a 32 bitových inštrukcií, čo umožňuje redukovať veľkosť programu v programovej pamäti. Procesor využíva dizajn trojstavového zreťazenia s predikciou skokov. Súčasťou ARM Cortex-M4 je konfigurovateľný radič prerušení nazývaný NVIC, ktorý podporuje až 256 úrovní prerušení. Procesor taktiež podporuje rôzne funkcie pre implementáciu operačného systému ako sú napríklad systémový časovač alebo tieňový ukazovateľ na zásobník [23].

3.4.1 Programovací model

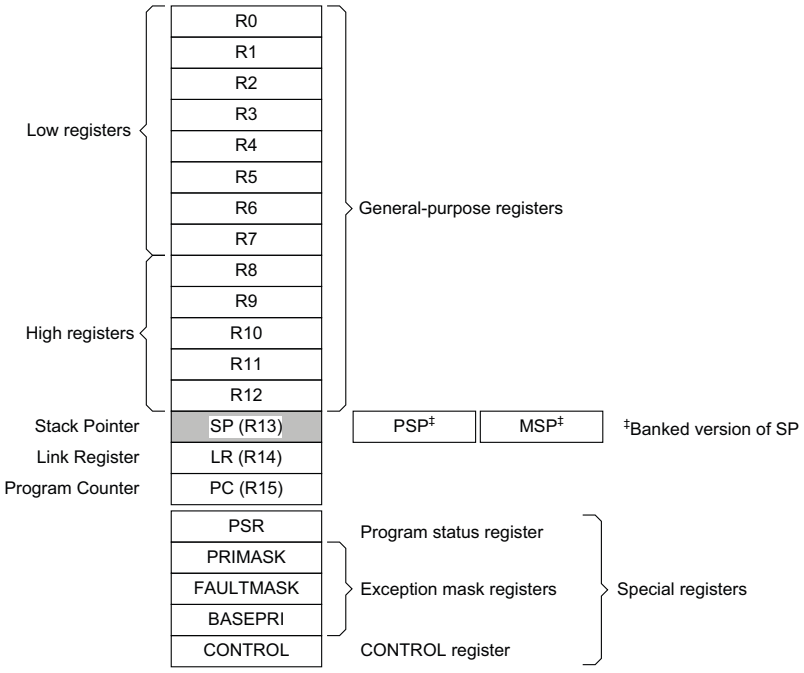
Programovací model procesorového jadra ARM Cortex-M4 je vysoko konzistentný. K dispozícii je 16 registrov, R0 až R15 a registre PSR, CONTROL a PRIMASK. Taktiež dva špeciálne registre FAULTMASK a BASEPRI viď obrázok č. 3.4.

Register BASEPRI definuje minimálnu prioritu pre spracovanie výnimiek. Keď je register BASEPRI nastavený na nenulovú hodnotu, zabraňuje aktivácii všetkých výnimiek s rovnakou alebo nižšou prioritou ako hodnota BASEPRI.

Register CONTROL uchováva stav aktuálne aktívneho stack-pointru v SPSEL. Hodnota 0 zodpovedá MSP a hodnota 1 zodpovedá PSP. Taktiež definuje v nPRIV úroveň privilégií pre vykonávanie softvéru ak je procesor v Thread móde. Vo verzii ARMv7-M je dostupný neprivilegovaný exekučný režim.

Register PRIMASK zabraňuje aktivácii všetkých výnimiek s konfigurovateľnou prioritou. Register FAULTMASK zabraňuje aktivácii všetkých výnimiek s výnimkou prerušenia bez maskovania (NMI).

Na register PSR je mapovaný aplikačný register PSR (APSR), exekučný register PSR (EPSR) a PSR register prerušenia (IPSR). Register APSR obsahuje jednotlivé príznaky procesoru: N, Z, C, V, Q a GE, ktoré prislúchajú: Negative flag, Zero flag, Carry alebo Borrow flag, Overflow flag, DSP overflow a Greather or Equal flag. Register IPSR obsahuje číslo aktuálne vykonávanej ISR. Register EPSR obsahuje bit Thumb stavu.

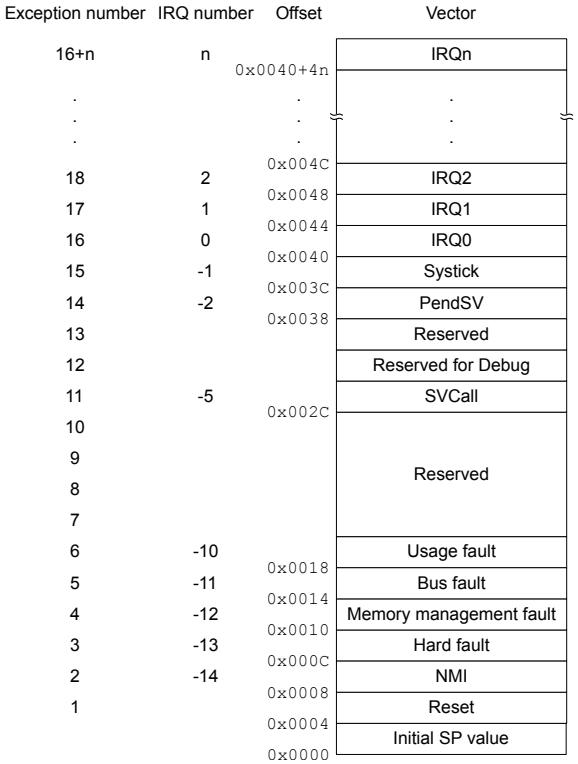


Obr. 3.4: Registre procesorového jadra ARM Cortex-M4 [3].

Thumb stav slúži pre vykonávanie inštrukcií procesoru. Procesor môže byť v jednom z dvoch módov. Prvým je Thread mód, ktorý slúži pre vykonávanie aplikačného softvéru. Procesor vstupuje do tohto režimu po resete. Druhým módom je Handler režim, používaný pre obsluhu výnimiek. Procesor ukončí tento režim a vráti sa do režimu Thread, keď dokončí vykonávanie všetkých výnimiek [3].

3.4.2 Prerušovací podsystém

Procesor pri spracovaní prerušení využíva vektorovú tabuľku obsahujúcu počiatočnú adresu pre obsluhu prerušení a štandardne sa nachádza na začiatku pamäte (adresa 0x00) viď obrázok č. 3.5. V obrázku je možné vidieť, že periférie používajú pozitívne IRQ čísla a CPU výnimky používajú negatívne IRQ čísla. Počiatočnú adresu tabuľky je možné zmeniť použitím funkcionality nazývanej VTOR (Vector Table Offset Register). To je užitočné pre premiestnenie vektorovej tabuľky do SRAM pamäte pre rýchlejšie načítanie vektorov v prípade ak je pomalá FLASH pamäť.



Obr. 3.5: Tabuľka vektorov prerušení v procesore ARM Cortex-M4 [25].

Obsluha prerušení v jednoduchosti pozostáva z nasledujúcich krokov:

1. Radiču prerušení NVIC príde žiadosť od zdroja prerušenia.
2. Radič prerušení na základe priority prerušenia a aktuálneho stavu procesora rozhodne, či bude žiadosť o prerušenie akceptovaná, ignorovaná alebo bude obsluha prerušenia odložená. Aktuálne stav procesora závisí od nastavenia vnútorných registrov a povolenia vnoreného vykonávania prerušení.
3. Procesor najskôr pozastaví aktuálne vykonávané inštrukcie. Následne uloží aktuálny stav registrov na zásobník a do registra PC uloží z tabuľky vektorov prerušení hodnotu adresy prerušovacieho podprogramu.
4. V okamihu dokončenia prerušovacieho podprogramu je obnovený pôvodný stav registrov zo zásobníka a pokračuje sa vykonávaním ďalších inštrukcií.

Exekučné stavy obsluhy výnimiek sú:

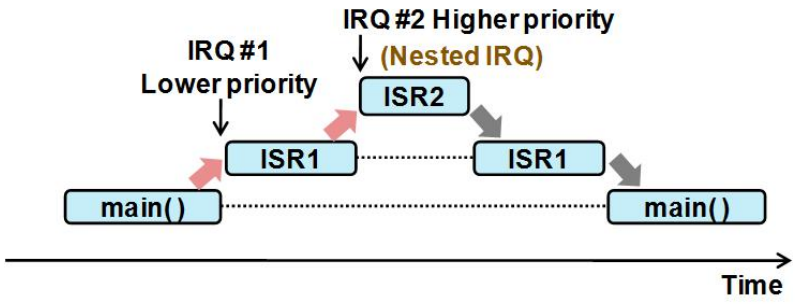
- Neaktívne: Výnimka nie je aktívna ani čakajúca.
- Čakajúce: Výnimka čaká kým bude obslúžená procesorom.
- Aktívna: Výnimka je obsluhovaná procesorom, ale ešte nie je dokončená.
- Aktívna a čakajúca: Výnimka je obsluhovaná procesorom a iná výnimka z rovnakého zdroja je v stave čakajúca.

Žiadosť o prerušenie z periférneho zariadenia alebo softvéru môže zmeniť stav odpovedajúceho prerušenia na čakajúci.

NVIC navyše umožňuje úrovňovú a pulznú detekciu prerušovacích signálov a externé nemaskovateľné prerušenia NMI (Non Maskable Interrupt). Procesor automaticky ukladá a obnovuje svoj stav pri vyvolaní výnimky a ukončení výnimky s nulovými režijnými nákladmi na inštrukcie [25].

3.4.3 Podpora vnorenej obsluhy prerušení

Skutočný vstavaný systém môže mať mnoho zdrojov, pričom normálne má každý zdroj prerušenia priradenú úroveň priority. Procesorová architektúra jadra ARM Cortex-M4 podporuje vnorené prerušenia, čo znamená, že pri vykonávaní obslužnej rutiny prerušenia (ISR) s nízkou prioritou, môže obsluhu prerušiť obslužná rutina prerušenia s vyššou prioritou. ISR s nízkou prioritou sa pozastaví a obnoví sa po dokončení ISR s vysokou prioritou viď obrázok č. 3.6.



Obr. 3.6: Vykonávanie vnorených ISR v čase procesorom ARM Cortex-M4 [25].

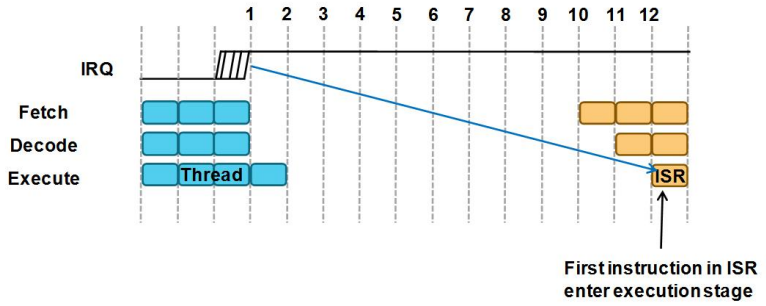
Mnoho vstavaných systémov vyžaduje vnorené spracovanie prerušení. V prípade, keď je spracovávané prerušenie s vysokou prioritou, žiadosti o prerušenie s nízkou prioritou sú pozdržané. To má za dôsledok zvyšovanie latencie pri prerušeniach s nízkou prioritou [24].

3.4.4 Latencia prerušení

Latencia prerušení je jednou z kľúčových vlastností mikrokontroléru a je dôležitá pre real-time aplikácie. Ide o počet cyklov hodín potrebných na odpoveď žiadosti o prerušenie. Teda počet taktov medzi zachytením žiadosti o prerušenie a vykonaním prvej inštrukcie ISR viď obrázok č. 3.7.

Latencia prerušenia procesoru Cortex-M4 je extrémne nízka, až na hodnote 12 taktov od zachytenia žiadosti po pripravenie prvej inštrukcie ISR, pri predpoklade nulového čakania na pamäť. V praxi je skutočná doba latencie ovplyvnená čakaním na pamäťový systém. Musia byť taktiež splnené podmienky, že obsluha prerušenia nie je blokovaná iným aktuálne

vykonávaným prerušením alebo výnimkou. Počet 12 taktov je limitovaný ukladaním na zásobník pri obsluhe prerušení, kde sa najskôr uložia na zásobník hodnoty registrov R0 až R3, následne registre R12, LR a xPSR.



Obr. 3.7: Zobrazenie latencie prerušenia v procesorovom jadre ARM Cortex-M4 [25].

Procesor umožňuje dobu latencie prerušení znižovať ďalšími technikami. V procesore Cortex-M4 sa pri príchode inštrukcie vo väčšine prípadov zruší aktuálne vykonávaná inštrukcia a obnoví sa až po dokončení ISR. Ak procesor prijme žiadosť o prerušenie počas inštrukcie viacnásobného načítavania/ukladania (pamäťového prístupu), aktuálny stav viacnásobného prenosu sa automaticky uloží ako súčasť registra stavu programu (PSR) a keď je ISR dokončená, viacnásobný prenos je možné obnoviť z miesta, kde bol predtým zastavený, pomocou uložených informácií z PSR.

V prípade ak príde žiadosť o prerušenie v okamihu dokončenia obslužnej rutiny ISR iného prerušenia a prebieha proces obnovy stavu zo zásobníku, je proces obnovy zastavený a ihneď je vykonávané ISR pre nové prerušenie.

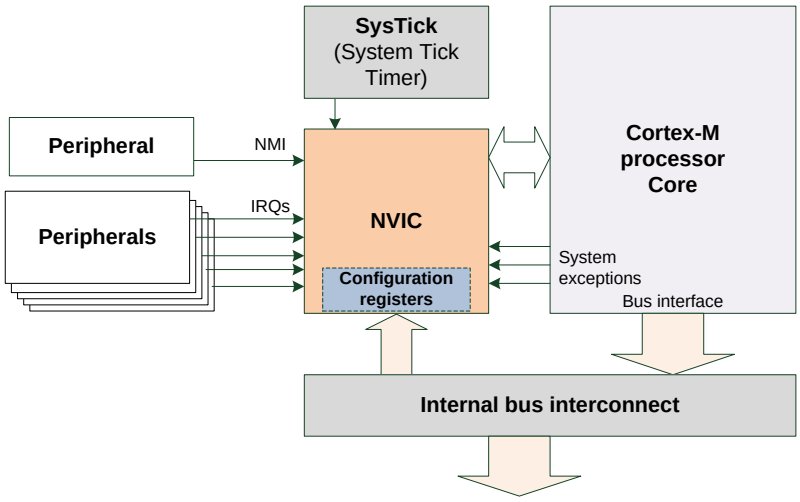
Tieto mechanizmy, spolu s technikami Tail-chaining a Late-arriving popísanými v kapitole 3.4.5, zabezpečujú a umožňujú udržiavať extrémne nízku latenciu spracovania prerušení [24].

3.4.5 Radič prerušení NVIC

Procesor obsahuje vstavaný konfigurovateľný radič prerušení NVIC (Nested Vectored Interrupt Controller). Prerušenia radiča môžu pochádzať z rôznych pripojených komponentov, ktorými sú systémové výnimky jadra procesoru, vnútorný systémový časovač, periférne zariadenia mikrokontroléru a periférie pripojené pomocou vstupno-výstupných portov viď obrázok č. 3.8. Radič poskytuje až 240 perifémych prerušení.

Jednotlivé prerušenia je možné individuálne povoliť, v registroch NVIC_ISER0 až NVIC_ISER7, alebo zakázať v registroch NVIC_ICER0 až NVIC_ICER7. Každý IRQ má svoj vlastný povoľovací/zakazovací bit v registroch. Nastavením bitu na hodnotu 1 v príslušnom registri dôjde k povoleniu/zakázaniu zvoleného prerušenia.

Radič poskytuje programovateľnú úroveň priorít od 8 (3 bity) až po 256 úrovní (8 bitov) pre každé prerušenie. Vyššia úroveň zodpovedá nižšej priorite a úroveň 0 je najvyššia úroveň priority. Priority jednotlivých prerušení je možné konfigurovať v registroch NVIC_IPR0 až NVIC_IPR59. Každý register je rozdelený na štyri 8 bitové bloky, pričom každý blok je vyhradený pre jeden zdroj prerušenia. Procesor má funkciu prioritného zoskupovania, ktorá umožňuje rozdeliť registre priorít do skupín priorít a sub-priorít. Podporuje taktiež dynamickú zmenu priorít prerušení počas behu programu.



Obr. 3.8: Bloková schéma radiča prerušení procesorového jadra ARM Cortex-M4 s jednotlivými zdrojmi prerušení [25].

Taktiež umožňuje efektívne zreťazenie spracovávania prerušení nazývané tail-chaining, ktoré umožňuje po obsluhe aktuálneho prerušenia obslúžiť nadväzne ďalšie aktuálne čakajúce prerušenie s nižšou prioritou bez návratu do pôvodného programu. To redukuje zápis hodnôt do registrov zo zásobníka a následné opätovné ukladanie na zásobník, čím umožňuje redukovať exekučný čas.

Radič poskytuje taktiež podporu pre mechanizmus predbiehania prerušení nazývaný Late-arriving. V prípade ak dôjde k výnimke s vyššou prioritou počas ukladania stavu na zásobník pre predchádzajúcu výnimku, procesor prepne spracovanie na výnimku s vyššou prioritou a iniciuje načítanie adresy z tabuľky vektorov prerušení pre túto výnimku. Ukladanie aktuálneho stavu nie je žiadno ovplyvnené neskorým príchodom iného prerušenia, pretože uložený stav je rovnaký pre obe výnimky, a preto sa ukladanie dokončí. Procesor môže akceptovať výnimky s oneskoreným príchodom dovtedy, kým neprejde prvá inštrukcia obslužnej rutiny pôvodnej výnimky do fázy vykonávania procesora. Pri návrate z obslužného podprogramu oneskorenej výnimky platia pravidlá pre zreťazené spracovanie výnimiek. Viac informácií o radiči prerušení je možné sa dozvedieť v dokumente [3].

3.4.6 SysTick časovač

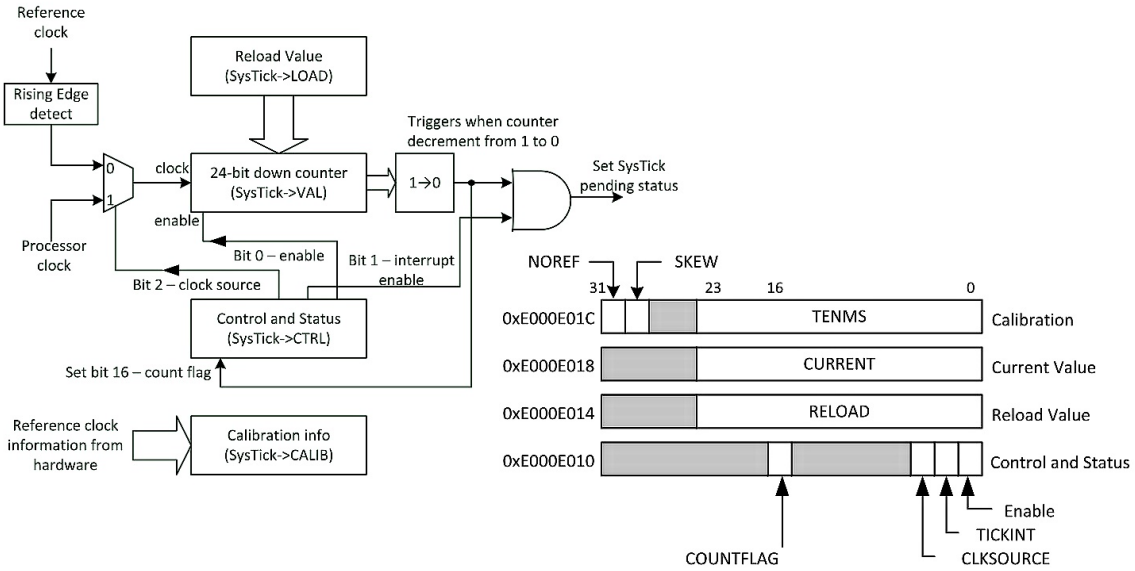
Procesor ARM Cortex-M obsahuje malý integrovaný časovač nazývaný SysTick. Ide o časovač integrovaný ako časť radiču prerušení NVIC a umožňuje generovanie SysTick výnimiek (výnimka číslo 15).

V moderných operačných systémoch, je periodické prerušenie potrebné pre zaistenie správnej činnosti operačného systému. Jadro operačného systému tak môže pravidelne vyvolávať napríklad manažment vláken a prepínanie kontextu. To umožňuje procesoru obsluhovať v rozdielnych časových úsekoch rôzne vlákna. Dizajn procesoru zaisťuje, že vlákna aplikácie spustené v neprivilegovanom režime nemôžu ovládať SysTick časovač, pretože by sa mohlo stať, že vlákno vypne časovač a zablokuje tak celý systém.

Dôvod prečo sa časovač nachádza priamo vo vnútri procesora je zlepšenie softvérovej prenositeľnosti operačných systémov napísaných pre Cortex-M procesory. V prípade ak nie je potrebné v aplikácií využiť operačný systém, SysTick časovač môže byť použitý

ako jednoduchý periférny časovač pre generovanie periodických prerušení, meranie čakacích intervalov, alebo časový manažment.

SysTick časovač je jednoduchý 24 bitový dekrementačný časovač, ktorý môže byť spustený na procesorovej frekvencií alebo na frekvencii referenčného zdroja hodín, viď obrázok č. 3.9. Konkrétne pri implementácii s referenčným zdrojom hodín môžu byť referenžné hodiny najviac dvakrát pomalšie v porovnaní s procesorovými hodinami, kvôli detekčnej logike nábežnej hrany.



Obr. 3.9: Zjednodušený blokový diagram SysTick časovača v ARM Cortex-M4 [23].

SysTick časovač obsahuje štyri konfiguračné registre, viď tabuľka č. 3.1. Keď je časovač zapnutý nastavením 0 do riadiaceho registru, register aktuálnej hodnoty sa s každým taktom procesorových hodín dekrementuje. Ak dosiahne nulovú hodnotu, znovu načíta pôvodnú nakonfigurovanú hodnotu z registra SYST_RVR a pokračuje. V prípade že je povolenie generovanie prerušení v riadiacom registri, pri dosiahnutí nulovej hodnoty sa zároveň vyvolá prerušenie.

Adresa:	**Názov:**	**CMSIS označenie:**	**Popis:**
0xE000E010	SYST_CSR	SysTick->CTRL	Riadiaci register SysTick
0xE000E014	SYST_RVR	SysTick->LOAD	Register opätovného načítania hodnoty
0xE000E018	SYST_CVR	SysTick->VAL	Register aktuálnej hodnoty
0xE000E01C	SYST_CALIB	SysTick->CALIB	Kalibračný register

Tabuľka 3.1: Zhrnutie SysTyck regostrov [23], [2].

Pri programovaní SysTick časovača je doporučená nasledujúca postupnosť:

1. Vypnutie SysTick časovač vložením nuly do riadiaceho registru.
2. Zápis novej konfiguračnej hodnoty časovača.

3. Vynulovanie registru s aktuálnou hodnotou zápisom ľubovolnej hodnoty.

4. Spustenie časovača v riadiacom registri.

Dátová štruktúra nazývaná SysTick je definovaná v hlavičkovom súbore CMSIS-Core a umožňuje jednoduchší prístup k registrom. Ďalej sa v práci bude používať CMSIS-Core označenie registrov. Viac podrobností o SysTick časovači je možné nájsť v literatúre č. [23] a v referenčnom manuáli ARMv7-M [2].

3.4.7 Data WatchPoint and Trace jednotka - DWT

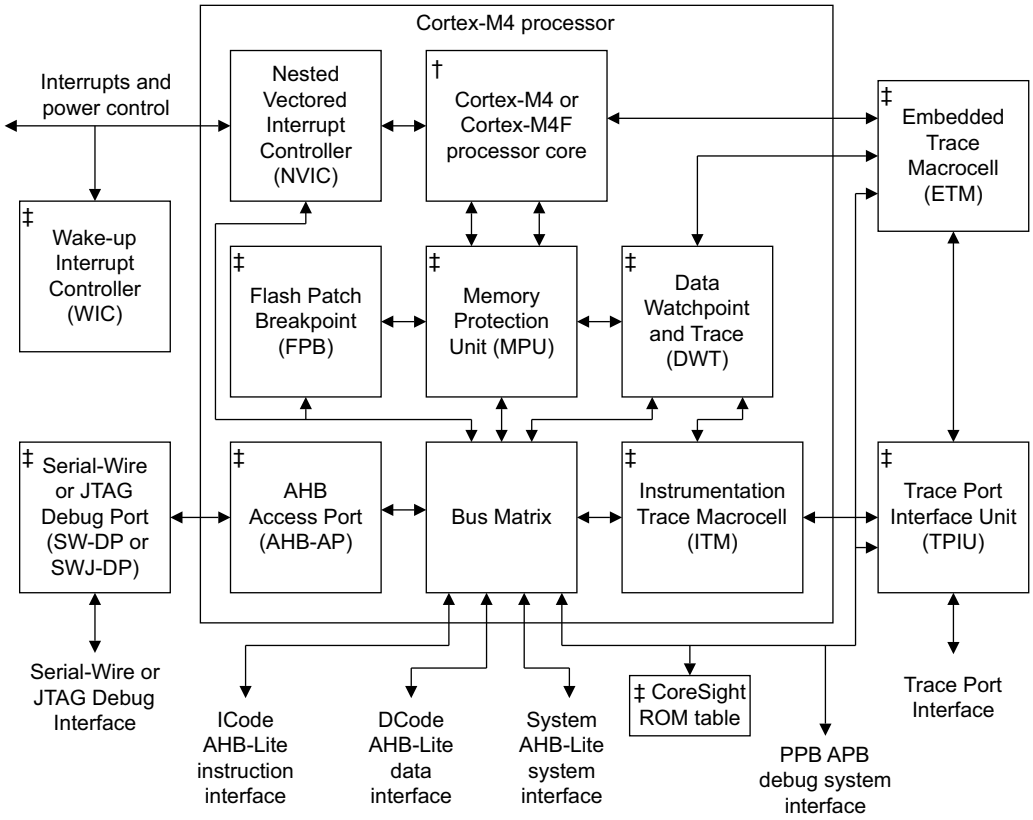
Ide o komponent obsiahnutý v procesore ARM Cortex-M4. Data WatchPoint and Trace jednotka, skrátene DWT, je súčasťou komponent pre podporu ladenia a debugovania nachádzajúcich sa na čipe, viď obrázok č. 3.10. Zaznamenáva informácie a údaje o dátach, udalostiach a profilovacie informácie. Táto komponenta je schopná sledovať prístupy k dátam a programový čítač (PC) CPU. DWT komponenta obsahuje štyri komparátory, ktoré môžu byť použité na spúšťanie rôznych akcií v okamihu, keď došlo ku zhode. Môže periodicky odosielať údaje o programovom čítači (PC), obslužnej rutine prerušenia (ISR) a prístupy k dátam. Okrem toho je zároveň schopná počítať špecifické typy cyklov CPU.

Medzi selektívne sledovanie dát patria napríklad informácie o pamäťových umiestneniach (kombinácie adries, hodnoty dát a časové značky), ktoré je možné zaznamenávať v každom čase, kedy procesor pristúpi k danej lokácii. Medzi profilovacie záznamy môžu patriť informácie o počte cyklov hodín CPU použitých v rôznych operáciách, napríklad prístup k pamäti alebo doba spánku. Medzi záznamy o udalostiach patria informácie poskytujúce dobu trvania a históriu prerušení alebo výnimiek, ktoré boli obslúžené procesorom [25].

Pre nastavenie komponentu slúži DWT_CTRL register. Register poskytuje informácie o konfigurácii a aktuálnom stave komponentu a používa sa taktiež na riadenie funkcií komponentu. Podpora sledovania výnimiek je voliteľná funkcia. Ak je podporovaná, softvér umožňuje sledovanie výnimiek nastavením bitu EXCTRCENA v registri DWT_CTRL na hodnotu 1. Ak je povolené sledovanie výnimiek, DWT vygeneruje trasovací paket výnimky v prípade, ak nastane niektorá z nasledujúcich udalostí:

- Procesor vstúpi do obsluhy výnimiek, z režimu Thread, alebo prepnutím vlákna.
- Procesor ukončí spracovanie výnimky s vektorom EXC_RETURN.
- Procesor sa vráti z obsluhy výnimky a opätovne vstúpi do kódovej sekvencie prepnutého vlákna.

Register DWT_EXCCNT sa používa pre získanie informácií o počte cyklov strávených vykonávaním prerušenia. Pre inkrementáciu počtu taktov je využitých spodných 8 bitov registra. V registri je uchovaný počet taktov réžie obsluhy prerušenia. Počítadlo sa zvyšuje s každým taktom spojeným so vstupom alebo návratom z obsluhy prerušenia. To znamená, že počíta cykly spojené s ukladaním na zásobník alebo vyberaním zo zásobníka, preempciou a dalšími režijnými činnosťami. Čítač sa inicializuje na 0 v okamihu nastavenia bitu EXCEVTENA v registri DWT_CTRL na hodnotu 1. Komponenta taktiež obsahuje ďalšie 8 bitové registre DWT_CPICNT pre odhad počtu taktov inštrukcie, DWT_SLEEPCNT pre počítanie réžie režimu spánku, DWT_LSUCNT register pre počítanie počtu taktov load-store inštrukcií a register DWT_FOLDCNT pre počítanie zastavených inštrukcií.



Obr. 3.10: Blokový diagram zobrazujúci komponentu DWT a jej prepojenie v procesore ARM Cortex-M4 [25]. †Pre Cortex-M4F procesor, jadro zahŕňajúce Floating Point jednotku (FPU). ‡Voliteľná komponenta.

Register DWT_CYCCNT je určený pre počítanie počtu vykonaných cyklov. Ide o 32-bitový register. Keď je povolený povoľovací bit DWT_CTRL.CYCCNTENA, čítač inkrementuje svoju hodnotu s každým vykonaným taktom procesorových hodín. K inkrementácii čítaču nedochádza, keď je procesor pozastavený v Debug režime. Následne je možné softvérovo pristúpiť k registru a prečítať aktuálnu hodnotu registra. Pri používaní je potrebné brať do úvahy dve možné situácie. Prvou je možné pretečenie čítača do nuly. Druhou je potenciálne nežiadúce povolenie alebo zakázanie, prípadne zmena hodnoty. Viac podrobností o komponente DWT je možné nájsť v referenčnom manuáli architektúry ARMv7-M [2].

Trace pripojenie umožňuje debuggeru zhromažďovať užitočné informácie o vykonávaní programu v reálnom čase, len s malým oneskorením. Výstupné údaje môžu byť prenášané dvomi spôsobmi:

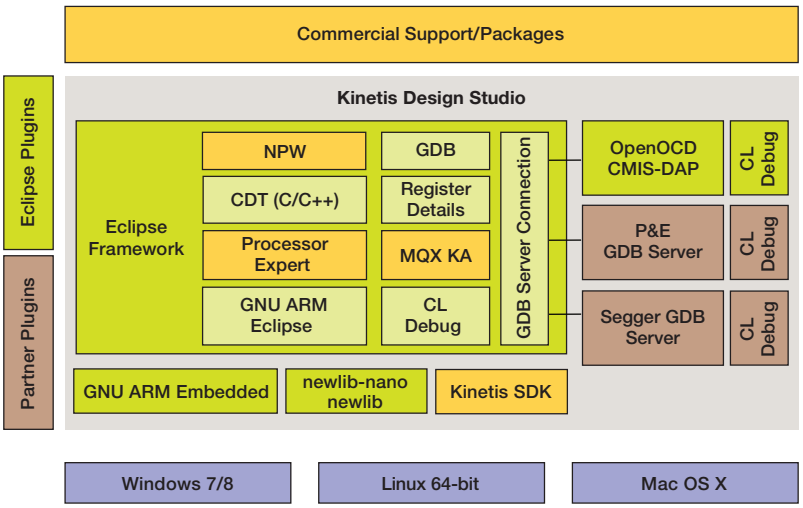
- Trace port - viacnásobný dátový pin spolu s hodinovým signálom. Poskytuje oveľa väčšiu šírku pásma v porovnaní s SWV. Na čipe Cortex-M4 obsahuje Trace port štyri dátové a jeden hodinový pin.
- Serial Wire Viewer (SWV) - jednopinové pripojenie, ktoré podporuje selektívne sledovanie údajov, udalostí, profilovacích a meracích záznamov.

Monitorovacie a sledovacie funkcie sú široko podporované rôznymi dodávateľmi integrovaných vývojových prostredí a informácie je možné vizualizovať rôznymi spôsobmi.

3.5 Kinetis Design Studio

Pre implementáciu jednotlivých techník zamedzujúcich preťaženiu v dôsledku nadmernej frekvencie prerušení, na platforme FITkit 3.0, bolo zvolené vývojové prostredie Kinetis Design Studia (KDS). Ide o pokročilé bezplatné integrované vývojové prostredie, určené na vývoj softvéru pre mikrokontroléry Kinetis, vytvorené spoločnosťou NXP Semiconductors, umožňujúce kompiláciu, a debuggovanie vytvorených projektov.

Je založené na voľnom, open-source softvéri zahŕňajúcom Eclipse, GNU kompilátor (GCC) a GNU debugger (GDB), pre ladenie vytvoreného programu. Má podporu pre dodatočne stiahnuteľné zásuvné moduly vrátane RTOS. Okrem toho ponúka Processor Expert softvér umožňujúci urýchliť vytváranie výkonných aplikácií výberom jednotlivých komponentov pre konkrétny mikrokontrolér, viď obrázok č. 3.11. Obsahuje taktiež SDK (Source Development Kit), periférne ovládače Kinetis SDK a spúšťací kód kompatibilný s CMSIS. Zvolený SDK je možný konfigurovať na internetovej stránke podľa konkrétneho typu mikrokontroléru, pre ktorý má byť implementácia určená. Taktiež je možné pre konfigurovaný SDK zvoliť požadovaný RTOS. Vygenerovaný SDK je následne možné pripojiť k vývojovému prostrediu. Poskytuje jazykovú podporu pre Assembler, C a C++. Taktiež podporuje operačné systémy MS Windows a Linux.



Obr. 3.11: Schéma integrovaného vývojového prostredia Kinetis Design Studio [5].

V súčasnosti už nie je Kinetis Design Studio IDE aktívne vyvíjané, čo na jednej strane nie je doporučované pre nové mikrokontroléry, zároveň je však vhodný pre všetky doterajšie architektúry, nakoľko poskytuje odladenú a dlhodobo otestovanú funkčnosť. Neposkytuje podporu len pre najnovšie MCU, pre ktoré spoločnosť NXP ďalej doporučuje vývojové prostredie MCUXpresso [5].

3.5.1 CMSIS-Core

CMSIS je štandardizované softvérové rozhranie pre mikrokontroléry založené na Cortex-M, pre ktoré zabezpečuje jednotnú hardvérovú abstrakčnú vrstvu[1]. CMSIS-Core (Cortex-M) implementuje základný run-time systém pre zariadenia Cortex-M ako konzistentnú a nezá-

[1]Popis špecifikácie softvérového rozhrania Cortex Microcontroller je dostupný na www.arm.com/cmsis

vislú vrstvu od výrobcu a poskytuje tak užívateľovi prístup k jadru procesora a periféram zariadenia. Zjednodušuje softvérové rozhrania s procesorom a jeho perifémymi zariadeniami.

Definuje jednotný prístup k registrom CPU, registrom preriférií, SysTick časovaču, radiču prerušení NVIC, vektorom výnimiek ale aj MPU a FPU. Ďalej definuje mená registrov periférií a názvy vektorov výnimiek. Taktiež poskytuje rozhranie pre ladenie aplikácie nezávislé na zariadení. Podrobnú špecifikáciu jednotlivých funkcií umožňujúci unifikovaný prístup a konfiguráciu periférií je možné nájsť v literatúre [10].

Kapitola 4

Experimentálne vyhodnotenie vplyvu nadmerných prerušení

Kapitola experimentálne vyhodnotenie dopadu prerušení na platformu FITkit 3.0 podrobne opisuje jednotlivé experimenty vykonávané na zvolenej platforme. Hlavným cieľom bolo navrhnúť a implementovať experimenty zamerané na charakterizovanie a zdokumentovanie výkonových vlastností procesorového jadra v okamihu preťaženia z dôvodu nadmernej frekvencie generovaných prerušení. Návrh a realizácia experimentov vychádzala z dôsledkov dopadu nadmernej frekvencie prerušení popísaných v kapitole č. 2.4.

Úlohou navrhnutých experimentov bolo skúmať a vyhodnotiť výkonnostné parametre mikrokontroléru Kinetis MK60DN512VMD10 na zvolenej platforme. Zároveň čo najpresnejšie špecifikovať a charakterizovať výpočtovú výkonnosť procesorového jadra ARM Cortex-M4, ktorým mikrokontrolér disponuje, pri rôznom stupni preťaženia z dôvodu nadmernej frekvencie generovaných prerušení. Pri návrhu bolo potrebné zohľadniť možnosti konfigurácie radiča prerušení NVIC zahrnutého v procesorovom jadre ARM Cortex-M4.

Samotné experimenty boli realizované a rozdelené do troch hlavných nezávislých implementačných častí.

- Prvý experiment vyhodnocuje časovú réžiu spojenú s obsluhou prerušení a rôznym časom stráveným v obsluhe ISR, viď kapitola č. 4.1.
- Ďalší experiment je orientovaný na zistenie vplyvu prerušení na výpočtový výkon platformy FITkit 3.0, vzhľadom na frekvenciu a čas obsluhy generovaných prerušení viď kapitola č. 4.2.
- Tretí experiment sa zameriava na stanovenie referenčných výkonnostných limitov pri preťažení zvolenej platformy v dôsledku prerušení, viď kapitola č. 4.3.

Každý z experimentov pozostáva z návrhu experimentu, popisu implementácie experimentu a zhodnotenia nameraných výsledkov v jednotlivých realizovaných experimentoch. Z dôvodu lepšej čitateľnosti a jednoduchšej orientácie čitateľa, bolo členenie na návrh, implementáciu a zhodnotenie výsledkov zvolené aj v nasledujúcich kapitolách.

Všetky experimenty boli implementované v programovacom jazyku C vo vývojovom prostredí Kinetis Design Studio s využitím CMSIS-Core, popísanom v kapitole 3.5.1.

4.1 Experiment 0 - Časové réžie spojené s obsluhou prerušení

Cieľom prvého experimentu bolo zistiť časové vlastnosti vykonávaných obslúh prerušení ISR. Hlavná myšlienka spočíva v experimentálnom stanovení miery závislosti počtu taktov procesorového času, ktoré procesor venuje obsluhe konkrétnej obslužnej rutiny prerušenia v závislosti od rozsahu vykonávanej úlohy resp. počte vykonávaných inštrukcií v rámci obslužnej rutiny prerušenia.

4.1.1 Návrh experimentu

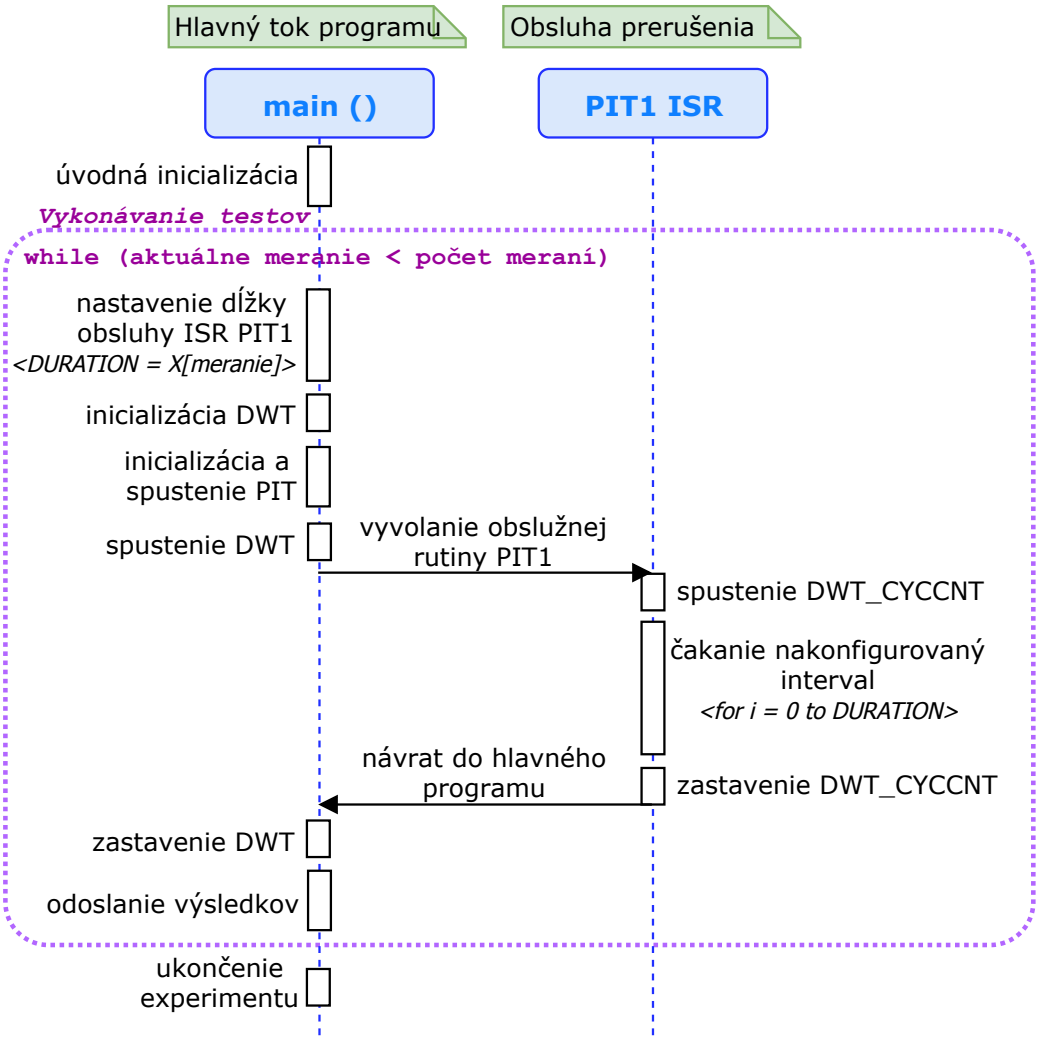
Pri realizácii experimentu je vhodné použiť komponentu PIT pre generovanie prerušení a komponentu DWT pre vykonávanie meraní.

- **PIT: Generovanie prerušenia** určeného pre meranie zvolených hodnôt je vhodné realizovať pomocou periodického časovača PIT umiestneného priamo na čipe mikrokontroléru Kinetis. Modul PIT je v porovnaní s inými časovačmi pre tento účel vhodný najmä pre jeho možnosť konfigurovať čas vyvolania prerušenia s presnosťou na jednotlivé takty. Modul je vhodný taktiež pre jeho jednoduchú inicializáciu a konfiguráciu, popísanú v kapitole 3.3.1. Nakoľko modul časovača PIT poskytuje štyri nezávislé, samostatne konfigurovateľné kanály, v tomto experimente je potrebné využiť iba jeden z dostupných kanálov.
- **DWT: Meranie a stanovenie presného počtu procesorových taktov**, ktoré jadro ARM Cortex-M4 venuje obsluhe prerušenia v ISR, zabezpečuje dedikovaná komponenta DWT umiestnená priamo v procesorovom jadre, popísaná v kapitole 3.4.7. Jednotku DWT je vhodné v rámci experimentu použiť ako pre meranie taktov s využitím registra CYCCNT, tak aj pre meranie réžie potrebnej k vykonaniu prerušenia, ktorá zahŕňa uloženie aktuálneho kontextu na zásobník pred vstupom do obslužnej rutiny prerušenia („stacking“), tak aj opätovné obnovenie kontextu zo zásobníka po obslúžení prerušenia („unstacking“). Pre meranie počtu taktov réžie prerušenia je vhodné využiť register EXCCNT.

Myšlienkou experimentu bude nakonfigurovať potrebné komponenty pre vykonávanie merania počtu taktov. Merania budú vykonávané v jednotlivých krokoch. Kroky sa budú od seba navzájom odlišovať počtom vykonávaných inštrukcií v obslužnej rutine prerušenia. Meranie bude vykonávať komponenta DWT pomocou príslušných registrov. Objektom merania bude počet taktov vo fázach „stacking“, „unstacking“ a v tele obsluhy prerušenia. Návrh režimu činnosti experimentu je zobrazený sekvenčným diagramom, viď obrázok č. 4.1.

4.1.2 Softvérová implementácia experimentu

Experiment skúmajúci časové vlastnosti obslužnej rutiny prerušenia je implementovaný s využitím knižníc `MK60D10.h` a `core_cm4.h`, pričom implementácia bola vyvíjaná vo vývojovom prostredí KDS. Pri implementácii boli využité súbory `printf.c`, `uart5.h` a `uart5.c`, poskytnuté vedúcim práce, ktoré implementujú odosielanie znakov prostredníctvom protokolu UART cez sériovú komunikáciu, a pre účely realizácie experimentu si vyžadovali miernu modifikáciu.



Obr. 4.1: Sekvenčný diagram zobrazujúci priebeh a riadenie experimentu.

Implementácia vychádza z kapitoly návrhu a experiment je implementovaný ako samostatný Kinetis SDK Prokejt s názvom Experiment0. Podprogramy implementujúce jednotlivé podporné funkcie návrhu sa nachádzajú spolu s hlavnou logikou programu v podadresári `Sources` v súbore `main.c`.

Algoritmus č. 2 zobrazuje celkovú štruktúru a následnosť vykonávania krokov implementovaného experimentu. Konkrétna implementácia jednotlivých krokov vychádzajúca z návrhu, je ďalej podrobne vysvetlená a popísaná.

Úvodná inicializácia

Úvodná fáza experimentu zahŕňa inicializáciu, kde je najskôr potrebné inicializovať samotný mikrokontrolér a správne nastaviť režim generátoru hodinového signálu (algoritmus č. 2, riadok 1). Generátor hodinového signálu je nakonfigurovaný z režimu FEI, do ktorého je uvedený po resete, do režimu BLPE, ktorý využíva externý referenčný zdroj hodín a tým dosahuje vyššiu mieru presnosti v porovnaní s režimom FEI, kde sa využíva interný oscilátor, ktorý môže byť ovplyvnený vplyvom zmeny teploty. Režim BLPE síce vyžaduje

Algoritmus 2: Algoritmus experimentu skúmajúceho časové vlastnosti obslužnej rutiny prerušenia

```
1  inicializuj generátor hodín (MCG) do režimu BLPE
2  vypni watchdog
3  inicializuj sériovú komunikáciu cez UART
4  inicializuj indikačné diódy LED
5  while index aktuálneho merania < celkový počet meraní do
6      inicializuj komponentu DWT
7      inicializuj SysTick časovač
8      inicializuj a spusti PIT časovač
9      spusti merania réžie a počtu vykonaných inštrukcí v ISR PIT1, pomocou
         komponenty DWT
10     čakaj na vyvolanie prerušenia
11     zastav merania pomocou komponenty DWT
12     ulož namerané výsledky a inkrementuj počet vykonaných meraní
13     odošli namerané výsledky cez UART
14     inkrementuj index aktuálneho merania
15 end while
```

počiatočnú konfiguráciu ale udržuje frekvenciu hodín stabilne na hodnote 50 MHz, čo je dôležité pri vykonávaných meraniach. Prepnutie do režimu BLPE je realizované funkciou `set_MCG_to_BLPE()`, ktorá najskôr pomocou riadiaceho registra MCG_C1 zvolí externý referenčný zdroj hodín nastavením binárnej hodnoty „10“. Následne softvér počká, kým sa hodiny prepnú na externé referenčné hodiny nastavením hodnoty „0“ na pozíciu IREFST a nastavením hodnoty „10“ na pozíciu CLKST v registri MCG_S. Na záver prejde do režimu BLPE konfiguráciou hodnoty registru MCG_C2_LP na hodnotu „1“.

Ďalším krokom je vypnutie watchdogu (algoritmus č. 2, riadok 2). Watchdog je vypnutý vynulovaním bitu WDOGEN v registri WDOG_STCTRLH. Následne je potrebné inicializovať sériovú komunikáciu s mikrokontrolérom, ktorú je možné použiť pre prenos nameraných výsledkov z platformy FITkit 3.0 do počítača (algoritmus č. 2, riadok 3). Pre inicializáciu sériovej komunikácie prostredníctvom UART bola modifikovaná funkcia `uart5_init` zdrojového súboru `uart5.c`. Konfigurácia nastaví rýchlosť komunikácie na hodnotu 115200 baudov pri zvolenej taktovacej frekvencii 50 MHz. Vďaka tomu je pre odosielanie textového reťazca s nameranými výsledkami možné použiť funkciu `printf()` implementovanú v priloženom súbore `printf.c`.

Pre umožnenie vizuálnej kontroly stavu a priebehu experimentu na zvolenej platforme bola zvolená signalizácia pomocou LED diód D9 a D12 umiestnených na doske plošných spojov (algoritmus č. 2, riadok 4). LED D12 je zvolená pre signalizáciu vykonávania obslužnej rutiny prerušenia PIT časovača a LED D9 dokončenie vykonávaných meraní. Inicializácia LED diód je implementovaná funkciou `init_LED`, ktorá zapne hodiny na PORTB v registri SIM_SCGC5, nastaví príslušným pinom GPIO funkcionalitu a nakonfiguruje ich ako výstupné. Pre ovládanie stavu diód sú implementované funkcie pristupujúce k registrom PTB_PDOR alebo PTB_PTOR, pre vypnutie alebo prepnutie aktuálneho stavu. Pre nastavovanie konkrétneho bitu výstupného pinu je implementované makro `GPIO_PIN()`, ktoré určí pozíciu pre nastavenie hodnoty pinu.

Vykonávanie meraní

Hlavná časť experimentu obsahuje vykonávanie jednotlivých nakonfigurovaných experimentálnych meraní, uloženie nameraných výsledkov a ich následné odosielanie pomocou asynchrónnej sériovej komunikácie UART do konzoly pripojeného počítača. Namerané výsledky je tak po prijatí možné následne spracovať a vyhodnotiť v externom tabuľkovom editore.

Experiment musí byť realizovaný vo viacerých krokoch, respektíve iteráciách hlavného cyklu, ktoré sa od seba navzájom odlišujú nastaveným počtom inštrukcií vykonávaných v rámci obslužnej rutiny prerušenia PIT časovača `PIT1_IRQHandler()`. V každom kroku sa zmení počet vykonaných inštrukcií v obslužnej rutine prerušenia (algoritmus č. 2, riadok 5). Postupné navyšovanie vykonávaných inštrukcií v obslužnej rutine prerušenia ISR časovača PIT je realizované postupným zvyšovaním počtu iterácií aktívneho čakacieho cyklu.

Čakací cyklus v ISR PIT1 nie je určený na vykonávanie žiadnej efektívnej funkcie. Úlohou čakacieho cyklu je postupne inkrementovať premennú umiestnenú vo vnútri cyklu a tým navyšovať množstvo vykonávaných inštrukcií, ktoré sa musia v rámci obsluhy prerušenia vykonať. Tým sa dosiahne postupné vygenerovanie prerušení s rôznym počtom vykonaných inštrukcií tj. rôznou dĺžkou času vykonávania obslužnej rutiny.

V každom kroku experimentu sa hodnota iterácií čakacieho cyklu zväčší o vopred stanovenú hodnotu. Hodnoty čakajúceho cyklu v obsluhe prerušenia sú uložené v globálnom poli `PIT1_interrupt_duration`, ktoré umožňuje prístup k hodnotám aj z obslužnej rutiny prerušenia PIT. Počet jednotlivých krokov experimentu je 16 a obsahuje hodnoty počtu iterácií čakacieho cyklu. Vhodne boli zvolené hodnoty: 0, 2, 4, 6, 8, 10, 20, 40, 60, 80, 100, 200, 400, 600, 800 a 1000. Index aktuálne nastavenej hodnoty je uchovaný v premennej `PIT1_current_duration` a je aktualizovaný v každom kroku experimentu.

Inicializácia komponenty DWT

Pre uskutočnenie merania pomocou komponenty DWT procesoru Cortex-M4, v rámci každého vykonaného kroku, je potrebná jej úvodná inicializácia (algoritmus č. 2, riadok 6). Inicializácia DWT komponenty je implementovaná funkciou `init_DWT()`. Funkcia využívajúca CMSIS-Core v úvode povolí komponentu DWT nastavením 24. povoľovacieho bitu TRCENA v registri DEMCR. Následne vynuluje kontrolný register CTRL. Zároveň je dodatočne vynulovaný 32-bitový register DWT_CYCCNT určený pre počítanie taktov, a všetkých päť 8-bitových registrov určených pre zaznamenávanie režijných meraní. Ide o profilovacie čítače inkrementované pri zaznamenaní asociovanej udalosti. V experimente sú podstatné hodnoty registra DWT_CYCCNT a registra DWT_EXCCNT určeného pre zaznamenávanie počtu taktov spojených s réžiou prerušení.

V závere funkcie sú nastavené premenné `mask_DWT_CTRL_cyccntena` a `mask_DWT_CTRL`. Ide o globálne premenné, slúžiace ako združujúca maska pre spustenie alebo zastavenie požadovaných meraní. Nastavenie príslušných bitov je realizované pomocou operátora logickej funkcie „OR“ medzi jednotlivými povoľovacími maskami, implementovanými v knižnici `core_cm4.h`, príslušných čítačov DWT komponenty.

Masky je možné následne použiť pre implementáciu makier `start_DWT()`, `stop_DWT()`, `start_DWT_cyctena()` a `stop_DWT_cyctena()` pre súbežné spustenie a zastavenie všetkých čítačov určených pre zaznamenávanie meraných hodnôt. Implementácia pomocou makra sa vyhodnocuje v čase prekladu a v porovnaní s volaním funkcie tak nepridáva časovú réžiu počas vykonávania programu. Implementácia konfigurácie DWT komponenty je ralizovaná podľa doporučení z referenčného manuálu ARMv7-M Architecture [2].

Inicializácia SysTick časovača

Následne je inicializovaný systémový časovač, ktorý bude slúžiť pre porovnanie, kontrolu a zároveň overenie správnosti referenčných výsledkov nameraných pomocou komponenty DWT (algoritmus č. 2, riadok 7). Inicializácia je implementovaná funkciou `init_SysTick()`, a využíva taktiež CMSIS-Core. Časovač je nakonfigurovaný tak, aby bolo možné pomocou neho merať trvanie zvoleného časového úseku, konkrétne telo obsluhy prerušenia. V tele funkcie je vynulovaný kontrolný register CTRL, čo spôsobí vypnutie SysTick časovača v prípade, ak bol náhodou spustený už predtým. Do registra LOAD, určeného pre počiatočnú inicializáciu prvotnej hodnoty, je podľa doporučení z literatúry nastavená maximálna hodnota `0xFFFFFFFF`, nakoľko SysTick časovač číta v zostupnom poradí. Ďalším krokom je vynulovaná aktuálna hodnota nastavená v registri VAL.

Nakoniec je nastavená hodnota `0x5` do kontrolného registra CTRL, čím sa opätovne povolí časovač s použitím procesorových hodín. Na záver je vo funkcii potrebné počkať, kým je aktuálna hodnota registra VAL rovná 0. Je to z dôvodu aby nedošlo k pretečeniu časovača počas vykonávania merania. SysTick časovač je spustený neustále. Pre výpočet meraného počtu taktov požadovaného intervalu sú použité implementované makrá `start_SysTick()` a `stop_SysTick`, ktoré zaznamenajú aktuálnu hodnotou časovača. Kým SysTick dekrementuje čítač, hodnota `start_time` je väčšia ako hodnota `stop_time` a výpočet intervalu sa získa odčítaním zaznamenaných hodnôt.

Do implementácie funkcie je možné taktiež zahrnúť kontrolu pretečenia pri meraní pomocou nastaveného bitu COUNTFLAG. V implementovanom experimente je však meraný interval dostatočne krátky a tak k pretečeniu čítača nedôjde, preto túto situáciu nie je potrebné implementovať. Systémový časovač SysTick je možné využívať iba bez použitia vstavaného operačného systému a to z dôvodu, že operačný systém daný časovač využíva a nemôže tak byť použitý v aplikačných úlohách. Konfigurácia a výpočet požadovaného intervalu je implementovaný podľa doporučení z literatúry [23].

Týmto krokom je dokončená inicializácia podporných štruktúr určených pre zaznamenávanie požadovaných hodnôt počtu taktov, ktoré procesorové jadro vykonáva obsluhu prerušení. Je dôležité správne zachovanie poradia pri inicializácii.

Inicializácia a spustenie PIT časovača generujúceho prerušenia

Je nevyhnutné inicializovať jeden zvolený kanál z komponenty PIT časovača, ktorý generuje maskovateľné prerušenia po vopred nakonfigurovanom počte taktov procesora (algoritmus č. 2, riadok 8). Pri implementácii bol použitý kanál časovača PIT1. Inicializácia časovača je vykonávaná vo funkcii `init_PIT()`. V úvode inicializácie je pomocou funkcie z knižnice CMSIS-Core `NVIC_ClearPendingIRQ()`, v radiči prerušení NVIC odstránený čakajúci stav prípadného nevykonaného prerušenia IRQ prvého kanálu PIT časovača. Parametrom funkcie je číslo IRQ prvého kanálu, ktorý má hodnotu 69 a je možné ju vypočítať ako číslo vektoru prerušenia mínus 16. Ďalším krokom je povolenie hodín pre modul PIT v konfiguračnom registri SIM_SCGC6 a zapnutie samotného modulu nastavením hodnoty `0x00` do registra PIT_MCR.

Nasleduje priradenie samotného časového limitu. Hodnotu vypršania časovača a vyvolania prerušenia je potrebné stanoviť vyššiu ako je počet taktov, ktoré trvá spustenie merania komponenty DWT preto, aby bolo možné spustiť meranie ešte pred tým, ako dôjde k vyvolaniu prvého prerušenia. Hodnota bola preto stanovená na 100000 taktov, čo zodpovedá intervalu 2 ms. Zároveň je v radiči prerušení NVIC funkciou `NVIC_EnableIRQ()` povolené prerušenie a funkciou `NVIC_SetPriority()` nastavená priorita na hodnotu 1. Na záver je

potrebné v registri PIT_TCTRL pomocou bitu TIE povoliť vyvolanie prerušenia v PIT časovači pri dosiahnutí nulovej hodnoty. Konfigurácia časovača PIT bola implementovaná podľa špecifikácie v referenčnom manuáli [9] a prezentácie k periférium mikrokontroléru Freescale Kinetis [4].

Po inicializácii a spustení časovača PIT je zároveň spustené samotné meranie počtu taktov komponentov DWT strávených v obsluhe najbližšieho vyvolaného prerušenia. Spustenie merania je implementované pomocou definovaných makier.

Spustenie merania a uloženie výsledkov

Dodatočne je potrebné počkať na uplynutie PIT časovača a vyvolanie prerušenia. Meranie počtu taktov réžie vyvolaného prerušenia, v registri EXCCNT, je spustené ešte pred čakaním na prerušenie (algoritmus č. 2, riadok 9). Tým sa zaznamená a zmeria veľkosť réžie pri ukladaní a obnovovaní kontextu zo zásobníka. Meranie počtu taktov vykonávaných vo vnútri obslužnej rutiny prerušenia, v registri CYCCNT, je spustené až v samotnej obslužnej rutine. Tým sa zaznamená počet taktov v rámci jeho obsluhy, a to tak, že začiatok merania počtu taktov bude umiestnený na začiatku obsluhy prerušenia a zastavenie merania sa bude nachádzať pred jeho ukončením. Tým je možné zmerať závislosť počtu taktov strávených v obsluhe prerušenia od počtu vykonávaných inštrukcií v rámci konkrétnej obsluhy prerušenia. Ukončenie merania nebude zahŕňať časť z konca obsluhy, ktorá je potrebná pre réžiu vypnutia časovača.

Čakanie na vyvolanie prerušenia je implementované aktívnym čakaním, ktorého vykonávanie trvá dlhšie ako perióda vyvolania prerušenia nastavená v PIT časovači (algoritmus č. 2, riadok 10). Tým sa zaistí zachytenie vyvolaného prerušenia a nedôjde tak k predčasnému ukončeniu merania.

Vyvolané prerušenie je obsluhované v obslužnej rutine prerušenia `PIT1_IRQHandler()`. V úvode obslužnej rutiny je zapnuté meranie počtu taktov strávených v obsluhe prerušenia. Ďalším krokom je odstránený čakajúci stav prerušenia v radiči NVIC, a vynulovaný príznakový bit vyvolaného prerušenia v registri PIT_TFLG1 zápisom logickej 1. Pre signalizáciu, že došlo k prerušeniu, je preklopený stav LED D12. Následne je v aktívnom čakacom cykle inkrementovaná premenná, čím je dosiahnutý rozdielny počet vykonaných inštrukcií v tele obslužnej rutiny jednotlivých experimentálnych meraní. Na záver je ukončené meranie počtu taktov v tele obslužnej rutiny, inkrementovaná premenná určujúca počet vykonaných prerušení a v radiči NVIC je zakázaná obsluha ďalšieho vyvolaného prerušenia. Kanál PIT1 je vypnutý, aby počas čakania na prerušenie nedošlo k vyvolaniu viacerých prerušení a tým k nesprávnym výsledkom.

Po dokončení každého merania sa výsledky z registrov komponenty DWT uložia do štruktúry typu `result`, ktorá obsahuje polia s veľkosťou rovnajúcou sa počtu vykonávaných meraní (algoritmus č. 2, riadok 12). Pri každom meraní sa zapíše stav sledovaných registrov do poľa na index aktuálneho kroku merania. Štruktúra taktiež obsahuje index aktuálneho merania. Ukladanie výsledkov prebieha vo funkcii `store_DWT_result_and_incr_pos()`. Namerané hodnoty sú taktiež zároveň odoslané sériovou komunikáciou do konzoly pripojeného počítača (algoritmus č. 2, riadok 13).

Porovnanie výsledkov pomocou SysTick časovača

Pre nezávislé overenie výsledkov nameraných pomocou komponenty DWT, je realizované duplicitné meranie rovnakých úsekov programu, pomocou malého integrovaného systémového časovača SysTick. Jednotlivé inštrukcie pre spustenie a zastavenie časovača sú umiest-

nené v rovnakom úseku implementovaného programu pre obsluhu prerušenia ako spustenie a zastavenie merania pomocou komponenty DWT.

Overenie výsledkov merania pomocou malého integrovaného časovača SysTick však nie je možné realizovať súčasne s meraním taktov pomocou komponenty DWT. Je to z dôvodu, že by bolo meranie počtu taktov, vykonaných v rámci obsluhy prerušenia, zaťažené chybou spojenou s vykonaním inštrukcií pre spustenie, prípadne zastavenie merania časovača. Výsledná implementácia tak obsahuje verziu s meraním pomocou komponenty DWT, ktorej výsledky sú nezávislé od verzie určenej pre overenie meraní, implementovanej s využitím integrovaného časovača SysTick. Tieto výsledky bolo potrebné získať v dvoch nezávislých spusteniach implementovaného experimentu.

4.1.3 Vyhodnotenie nameraných výsledkov

Meranie prebiehalo na zvolenej platforme. Výsledné hodnoty priebehu experimentu sú uložené v adresári `Experiment0` v podadresári `Results`. Adresár obsahuje súbor s výsledkami nameranými pomocou komponenty DWT, ale zároveň súbor s kontrolnými hodnotami nameranými pomocou systémového časovača SysTick. Výstup experimentu bol po prijatí sériovým rozhraním uložený v príslušných súboroch a následne boli údaje spracovávané v tabuľkovom editore. Doba vykonávania experimentov je 16 sekúnd.

Výstup experimentu je priložený na pamäťovom médiu v prílohách v súbore s názvom `DWT_CTRL_CYCCNTENA - meranie dlzky prerusenia.txt`. Súbor zobrazuje počet aktuálne vykonaných iterácií v obsluhe prerušenia, aktuálne nastavenú prioritu vyvolaného prerušenia, počet vyvolaných prerušení, nameranú réžiu prerušenia a nameraný počet taktov strávených obsluhou prerušenia. Taktiež zobrazuje pre každý test celkový počet taktov trvania jedného prerušenia ako súčet dvoch predošlých hodnôt. Vypočítaná hodnota však nezahŕňa časť obsluhy, ktorá je nevyhnutná z dôvodu riadenia experimentov.

Počas prvotnej fáze implementácie, kedy nebol využitý cyklus na dynamickú zmenu veľkosti spracovávanej úlohy, a telo obslužnej rutiny bolo merané bez čakacieho cyklu, bol počet taktov strávených v obslužnej rutine prerušenia 68. Pridaním cyklu sa táto hodnota navýšila o 27 taktov na hodnotu 95.

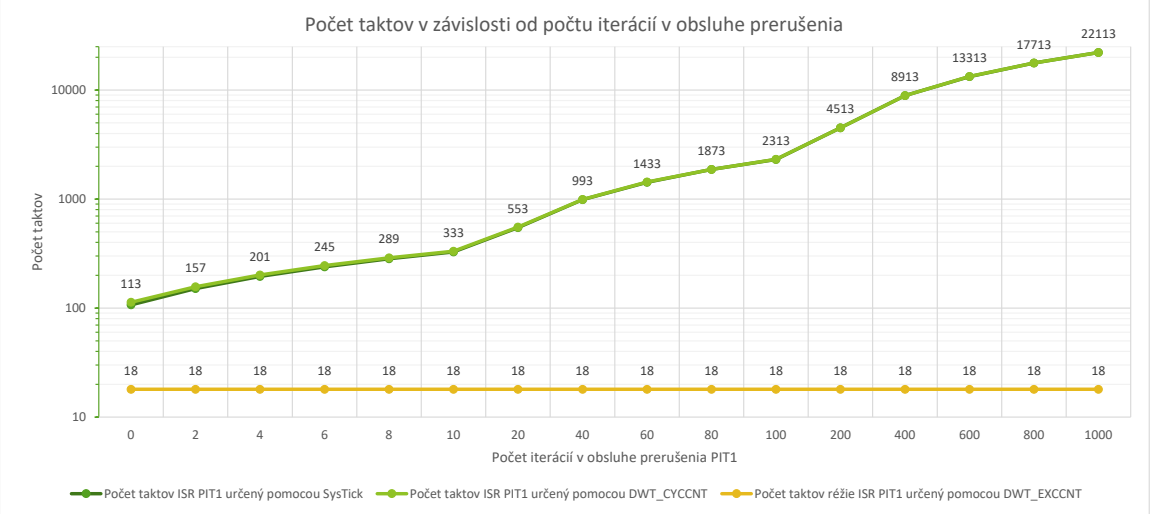
Pri spúšťaní experimentu pre rôzny počet iterácií je možné z nameraných výsledkov odvodiť vzťah pre stanovenie počtu taktov strávených v obsluhe prerušení pre určitý počet iterácií. Pri nulovom počte iterácií je počet taktov obsluhy tela prerušenia 95, v každej ďalšej iterácii táto hodnota stúpne o 22 taktov. Preto vykonanie inštrukcií jednej iterácie cyklu zaberie 22 taktov. Z toho je možné odvodiť vzťah:

$$Step\_count = (Iteration\_count * 22) + 95$$

Vzťah umožňuje vypočítať počet taktov zotrvaných v obsluhe prerušení pre ľubovoľný počet iterácií. Súčasťou výsledkov je taktiež súbor `Výsledky.txt`, v ktorom je uvedený výpočet z nameraných hodnôt.

Z nameraných hodnôt registra `DWT_EXCCNT` komponenty DWT vyplýva, že réžia obslúženia jedného vyvolaného prerušenia je 18 taktov pre „stacking“ a „unstacking“. Pri vykonávaní experimentov bolo možné miernou modifikáciou implementovaného programu zistiť čas pre „stacking“ a pre „unstacking“ osobitne tak, že ukončenie alebo začatie merania réžie bolo situované v tele obsluhy prerušenia. Tým sa doba merania špecifikovala iba na „stacking“ alebo „unstacking“. Týmto meraním bol overený predpoklad, že každá z režijných operácií, či už ide o ukladanie kontextu na zásobník alebo obnovovanie kontextu zo zásobníka, trvá procesoru 9 taktov.

Zo získaných výsledných hodnôt je možné vidieť priamu závislosť medzi počtom vykonaných inštrukcií v obsluhe prerušení, resp. počtom iterácií čakacieho cyklu, a počtom taktov strávených v obsluhe prerušenia, viď graf na obrázku č. 4.2. Graf žltou farbou zobrazuje konštantnú réžiu spojenú s vyvolanou obsluhou prerušenia. V grafe sú pre porovnanie zobrazené tmavozelenou farbou aj hodnoty namerané pomocou systémového časovača SysTick, na ktorých je možné vidieť konštantnú odchýlku 6 taktov oproti hodnotám nameraným jednotkou DWT. Odchýlka môže byť spôsobená réžiou konfigurácie DWT pri spúšťaní a zastavovaní merania.



Obr. 4.2: Počet taktov strávených obsluhou prerušenia v závislosti od počtu vykonaných iterácií v tele prerušenia.

Presné hodnoty taktiež sú uložené v prílohách v súbore `Experiment0.xlsx` v adresári `Výsledkové_grafy`. Hodnoty namerané v tomto experimente je možné asociovať s výsledkami nasledujúcich experimentov a použiť ich ako prepočet z počtu cyklov strávených v obsluhe prerušenia na počet vykonaných procesorových taktov. V nasledujúcich experimentoch budú preto hodnoty uvádzané v počte vykonaných cyklov

4.2 Experiment 1 - Odhad vplyvu frekvencie prerušení na výpočtový výkon

Cieľom experimentu bolo odhadnúť vplyv frekvencie generovaných prerušení na výpočtový výkon platformy FITkit 3.0. Myšlienkou je zistiť závislosť medzi frekvenciou generovaných prerušení, ktoré je potrebné obslúžiť, počtom inštrukcií vykonávaných v obsluhe prerušenia a počtom inštrukcií, ktoré je procesor schopný vykonať mimo obsluhy prerušenia v hlavnom toku programu.

4.2.1 Návrh experimentu

Koncepcia experimentu pozostáva z merania veľkosti úlohy, ktorú stihne mikrokontrolér vykonať v hlavnej funkcii `main()` pri rôznom stupni zaťaženia vplyvom generovaných prerušení, ktoré musí zároveň obslúžiť, a nie je možné ich obsluhu zakázať.

Úloha, ktorú musí procesor vykonávať v hlavnom programe môže byť časovo kritická, čo znamená, že za určitý časový interval reálneho času musí mať úloha k dispozícii dostatok procesorového času (výpočtových taktov) určeného na spracovanie príslušného množstva inštrukcií predstavujúcich danú úlohu. Príkladom takýchto úloh bežne vykonávaných procesorovým jadrom sú napríklad:

- Úlohy pracujúce s grafickým výstupom, kde je potrebné garantovať dodržanie stanoveného času výpočtu grafického výstupu.
- Úlohy zamerané na spracovanie zvukového vstupu, v ktorých sa od procesorového jadra vyžaduje presné vzorkovanie vstupného zvukového signálu.

Nadmerným generovaním prerušení, však dochádza k znižovaniu procesorového času, ktorý má samotná úloha k dispozícii, nakoľko obsluha prerušení má vyššiu prioritu. Experiment vychádza z predpokladu, že kritický časový interval je dostatočne veľký a z tohto dôvodu nie je vhodné obsluhu prerušení zakázať, prípadne samotné prerušenia vykonávajú ďalšie funkcie, kritické z hľadiska funkcionality celého systému.

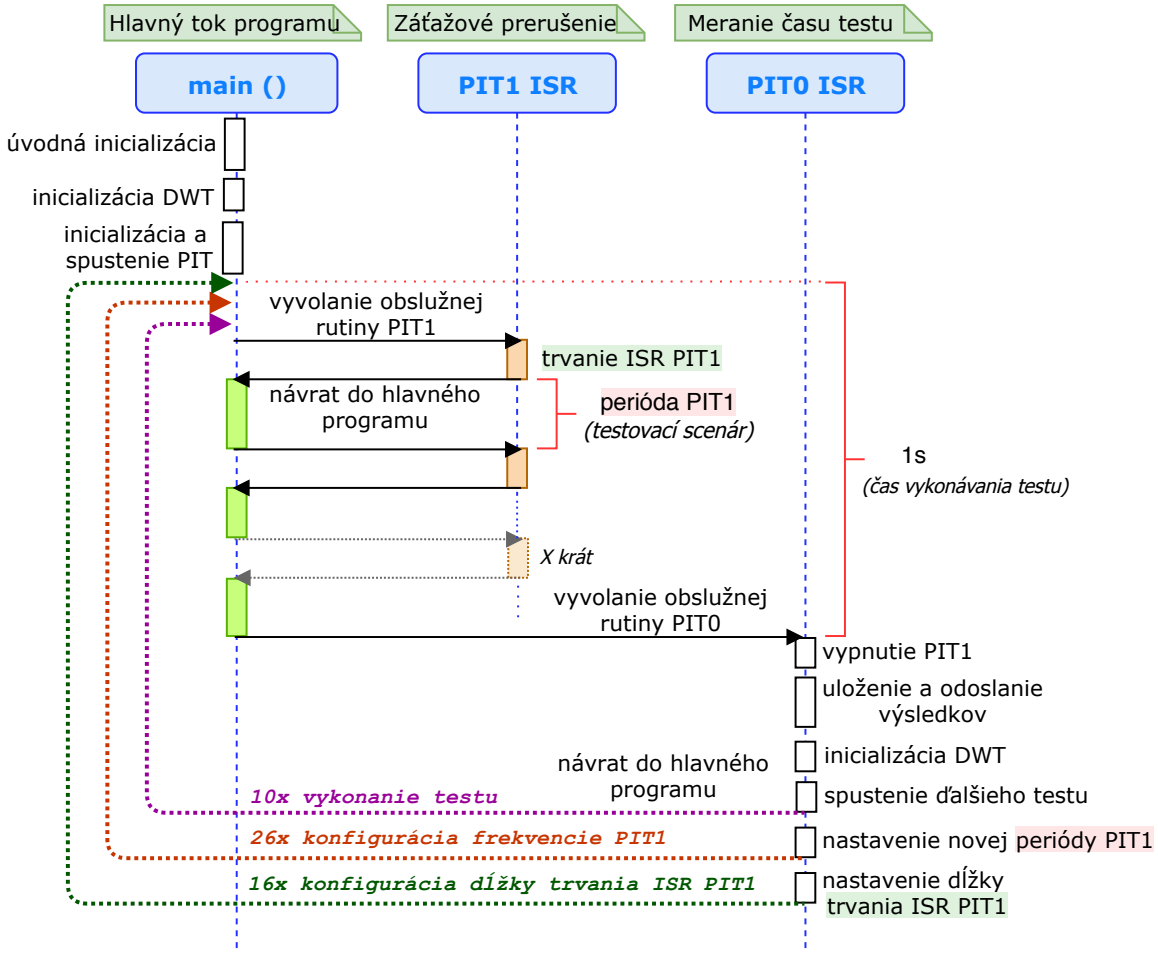
V tomto experimente bude kritický časový interval stanovený až na dobu jednej sekundy pre zvýraznenie rozdielov hodnôt nameraných v experimente. Meranie množstva procesorového času (taktov), ktoré počas kritického časového intervalu poskytuje procesorové jadro hlavnému programu, bude z dôvodu zjednodušenia a jednotného porovnávania a vyhodnocovania, predstavovať cyklus počítajúci iterácie strávené vykonávaním hlavného toku programu.

Periodický časovač PIT bude v tomto experimente použitý na presné vymedzenie kritického časového intervalu jednej sekundy, počas ktorej bude prebiehať nakonfigurovaný test. Zároveň bude PIT ďalším nezávislým kanálom počas tohto intervalu generovať záťažové prerušenia. Vstupnými parametrami experimentu budú:

- dĺžky trvania obsluhy vyvolaného prerušenia, resp. počty vykonaných inštrukcií
- frekvencie generovania záťažových prerušení (ďalej označované ako testovacie scenáre)

V experimente sa budú, v module PIT, systematicky meniť všetky vzájomné kombinácie preddefinovaných konfiguračných parametrov generovania prerušení. Najskôr sa nakonfiguruje dĺžka obsluhy vyvolaného prerušenia, pre ktorú sa následne vyhodnotia všetky frekvencie generovania záťažových prerušení (všetky scenáre). Počet testov jedného scenára je

stanovený na 10. Následne sa predĺži dĺžka obsluhy jedného prerušenia a zopakuje sa vyhodnotenie scenárov, až kým sa takto nezozbierajú výsledky meraní pre všetky kombinácie, viď obrázok č. 4.3.



Obr. 4.3: Sekvenčný diagram zobrazujúci priebeh a riadenie experimentu 1.

Predpoklad: Predpokladom je, že množstvo vykonaných iterácií hlavnej smyčky počas intervalu jednej sekundy nemusí byť pri viacerých opakovaniach rovnaký. Z tohto dôvodu bude každá konfigurácia (test) vykonávaná 10 krát, a tým bude možné stanoviť odchýlku merania a strednú hodnotu medzi jednotlivými testami.

Objekt merania: Objektom merania experimentu bude veľkosť úlohy, ktorú je schopné procesorové jadro vykonať v hlavnom toku programu počas obsluhy prerušení. Pre kontrolu priebehu experimentu bude, na meranie počtu taktov strávených v obsluhe prerušenia, použitá komponenta DWT, obdobne ako v predošlom experimente.

4.2.2 Softvérová implementácia experimentu

Experiment bol implementovaný podľa kapitoly návrhu, v ďalšom oddelenom Kinetis SDK Projekte. Implementácia sa nachádza na priloženom médiu v adresári `Experiment1`, podadresári `Sources`. Experiment je implementovaný v súbore `main.c`. Implementácia experimentu využíva rovnako ako v predošlom prípade podporné súbory `printf.c`, `uart5.h` a `uart5.c`.

Detailný pohľad na postup jednotlivých krokov v rámci vykonávania experimentu je zobrazený algoritmom č. 3. Kroky algoritmu sú ďalej podrobne vysvetlené.

Algoritmus 3: Algoritmus experimentu pre odhad vplyvu frekvencie prerušení na výpočtový výkon

```
1  inicializuj generátor hodín (MCG) do režimu BLPE
2  vypni watchdog
3  inicializuj sériovú komunikácu cez UART
4  inicializuj komponentu DWT
5  inicializuj indikačné diódy LED
6  inicializuj PIT časovače
7  nakonfiguruj v NVIC prioritu prerušení pre PIT časovač
8  spusti PIT časovače
9  for ;; do
10     inkrementuj počet vykonaných iterácií
11 end for
```

Úvodná inicializácia

Po spustení experimentu je potrebné inicializovať mikrokontrolér (algoritmus č. 3, riadky 1 až 5). Úvodné inicializačné kroky, medzi ktoré patrí: inicializácia generátora hodín a jeho nastavenie do režimu BLPE, vypnutie watchdogu, inicializácia sériovej komunikácie prostredníctvom rozhrania UART, inicializácia komponenty DWT určenej na meranie a inicializácia indikačných LED diód, sú z hľadiska konfigurácie zhodné s inicializačnými krokmi v predchádzajúcom experimente popisovanými v kapitole 4.1.2.

Inicializácia časovača PIT

Po inicializácii a úvodnej konfigurácii predošlých komponent a rozhraní, nasleduje inicializácia modulu časovača PIT. Inicializácia najskôr vyžaduje povolenie hodín pre daný modul (algoritmus č. 3, riadok 6). V tomto experimente je potrebné použiť dva kanály časovača, ktorými sú kanál PIT0 a kanál PIT1.

Do konfiguračného registra LDVAL kanálu PIT0 je nakonfigurovaný časový interval jednej sekundy definovaný v mikrosekundách pomocou `TEST_TIME`. Tento časovač slúži na presné meranie dĺžky trvania vykonávaného testu. Konfiguráciu zabezpečuje funkcia `compute_PIT_period()`, ktorá vykonáva konverziu zadaných mikrosekúnd na počet cyklov PIT časovača. Pre konverziu je použitý vzťah:

$$(\frac{milisekundy}{1000} * 50 * 10^6) - 1$$

Vzťah je odvodený z konfiguračného vzorca v kapitole 3.3.1, nakoľko frekvencia generátora hodín je nastavená na 50 MHz. Pre potreby experimentu zadávať periódu časovača v mikrosekundách je možné tento vzťah zjednodušiť na:

$$((mikrosekundy) * 50) - 1$$

Priorita prerušení PIT0 časovača je v radiči prerušení NVIC nastavená na najvyššiu možnú úroveň hodnotu 0 (algoritmus č. 3, riadok 7). Je to z dôvodu, aby pri ukončovaní testu nedošlo k čakaniu na obslúženie iného prerušenia s vyššou prioritou.

Druhý kanál časovača PIT1 je v experimente použitý pre generovanie záťažových prerušení počas intervalu jednej sekundy meraného kanálom PIT0. Frekvencie generovania záťažových prerušení sú preto uložené v poli `PIT0_interrupt_test_scenarios`. V implementácii je vhodne zvolených 26 hodnôt periódy: 100000, 80000, 60000, 40000, 20000, 10000, 8000, 6000, 4000, 2000, 1000, 800, 600, 400, 200, 100, 80, 60, 40, 20, 10, 8, 6, 4, 2 a 1 mikrosekunda. Priorita časovača PIT1 musí byť menšia ako je hodnota priority časovača PIT0. Z tohto dôvodu je priorita nastavená na hodnotu 1. Po nastavení priority oboch časovačov je v konfiguračných registroch povolené generovanie prerušenia z časovačov a časovače sú následne spustené (algoritmus č. 3, riadok 8). Zároveň sú povolené prerušenia v radiči NVIC.

Obslužná rutina záťažového prerušenia obsahuje čakací cyklus rovnako ako v experimente č. 4.2, ktorý umožňuje navyšovať počet vykonaných inštrukcií v rámci obsluhy prerušenia a tým dĺžkou obsluhy prerušenia. Hodnoty iterácie čakacieho cyklu sú rovnaké ako v predošlom experimente. V tomto experimente je pre každú dĺžku obsluhy vykonaná celá sada testovacích scenárov, tj. nakonfigurovaná každá z frekvencií generovania záťažových prerušení. Do obslužnej rutiny je dodatočne pridané kontrolné počítadlo vykonaných obslúh prerušení `PIT1_interrupt_load_access`.

Týmto krokom je dokončená fáza inicializácie a nasleduje vykonávanie cyklu inkrementujúceho globálnu premennú `main_iter_count` (algoritmus č. 3, riadky 9 až 11). Premenná slúži na stanovenie množstva procesorového času, ktoré procesor venuje vykonávaniu hlavného toku programu.

Riadenie vykonávaných testov z obslužnej rutiny časovača PIT0

Riadenie priebehu jednotlivých testov prebieha v obslužnej rutine vyvolanej časovačom PIT0. Zjednodušený algoritmus č. 4 popisuje jednotlivé kroky obslužnej rutiny. Časovač PIT0 po ukončení jedného jednosekundového testu najskôr vynuluje príznak prerušenia a následne vypne modul PIT, a tým aj generovanie ďalších prerušení, z dôvodu budúceho nastavenia novej periódy záťažového časovača PIT1 (algoritmus č. 4, riadky 1 a 2). Taktiež dôjde k preklopeniu stavu indikačnej diódy LED D9 signalizujúcej koniec jedného testu.

Algoritmus 4: Algoritmus obslužnej rutiny prerušenia riadiaceho časovača PIT0 pre riadenie jednotlivých testov

```
1  vynuluj príznak prerušenia v PIT module
2  vypni časovače PIT z dôvodu zmeny periódy
3  ulož výsledky z DWT
4  inicializuj komponentu DWT
5  aktualizuj minimálnu hodnotu počtu cyklov hlavnej smyčky
6  aktualizuj maximálny počet vykonaných obslúh záťažového prerušenia PIT1
7  odošli výsledky aktuálneho testu
8  inkrementuj index aktuálneho testu
9  if aktuálny test >= počet testov jedného scenára then
10     odošli výsledky aktuálne dokončeného scenára
11     inicializuj index aktuálneho testu
12     nastav parametre nového scenára
13 end if
14 spusti PIT časovače
```

Pri ukončení jedného vykonaného testu sú výsledky zo sledovaných registrov komponenty DWT uložené do štruktúry `results` (algoritmus č. 4, riadok 3), rovnako ako v prípade predošlého experimentu v kapitole č. 4.1.2. Jedinou odlišnosťou je doplnenie uchovávania maximálnej hodnoty registra DWT_CYCCNT v štruktúre, pre uloženie najhoršieho prípadu spomedzi sady desiatich testov s rovnakou počiatočnou konfiguráciou (jedného testovacieho scenára). Po uložení výsledkov sa zároveň v štruktúre inkrementuje index `position` ukladajúci najbližšiu voľnú pozíciu pre uloženie výsledkov nasledujúceho testu, nakoľko tento index nie je možné iným spôsobom, v obsluhe prerušenia PIT0, dopočítať. Následne je komponenta DWT opätovne inicializovaná funkciou `init_DWT()` a pripravená na zaznamenávanie výsledkov ďalšieho testu (algoritmus č. 4, riadok 4). Výsledky sú odoslané cez sériovú linku do pripojeného počítača.

Aktualizácia hodnôt minimálneho počtu cyklov hlavného programu a maximálneho počtu vykonaných obslúh záťažového prerušenia PIT1 bola implementovaná z dôvodu zistenia najhorších prípadov, ktoré nastali počas vykonávania sady desiatich testov s rovnakým nastavením počiatočných parametrov (algoritmus č. 4, riadky 5 a 6). Predpokladom bolo, že výsledky v rámci jednej sady testov sa môžu vzájomne líšiť (viď. kapitola č. 4.2.1). Výsledky tohto kroku implementácie budú podrobnejšie vysvetlené v podkapitole č. 4.2.3. Inicializované sú aj hodnoty počítadla vykonaných cyklov hlavného programu a počítadla prístupov do obsluhy prerušenia záťažového časovača PIT1.

Podmienka overuje, či došlo k vykonaniu všetkých testov v rámci aktuálneho scenára (algoritmus č. 4, riadok 9). Ak sú už vykonané všetky testy z aktuálneho scenára, dôjde k odoslaniu zozbieraných výsledkov, ktorými sú práve minimálny počet cyklov hlavného programu, maximálny počet vykonaných obslúh záťažového prerušenia PIT1 a uložené výsledky zaznamenané pomocou DWT. Zároveň sa index pre ukladanie výsledkov novej sady testov inicializuje na nulu (algoritmus č. 4, riadky 10 a 11). Najdôležitejším krokom, pri vykonaní podmienky, je nastavenie nových parametrov ďalšieho testovacieho scenára (algoritmus č. 4, riadok 12). Po tom, ako sú parametre nového scenára nastavené, sú opätovne spustené oba časovače a povolené vyvolanie prerušení pri uplynutí limitu časovača.

Nastavenie parametrov nového scenára

V prípade ak sú dokončené všetky testy jednej testovacej sady aktuálneho scenára, je potrebné nakonfigurovať nové vstupné parametre experimentu. Konfiguráciu parametrov zabezpečuje funkcia `set_new_PIT1_period_or_PIT1_duration()`. Algoritmus č. 5 zobrazuje postupnosť krokov tejto konfigurácie.

Najskôr je potrebné overiť, či sa vykonali všetky scenáre v rámci rovnakej konfigurácie času obsluhy PIT1 prerušenia. V prípade ak je podmienka splnená, dôjde k nastaveniu novej periódy záťažového časovača PIT1 do registra LDVAL, a k inkrementovaniu indexu aktuálneho scenára (algoritmus č. 5, riadky 1 až 3). Zvolené hodnoty testovacích scenárov sú spomenuté v predošlej časti o inicializácii časovača PIT.

Nasledujúca podmienka overí prípad vykonania všetkých scenárov a potrebu nakonfigurovať novú dĺžku obsluhy ISR záťažového prerušenia PIT1 (algoritmus č. 5, riadky 4 až 7). V tomto prípade sa zvýši index `PIT1_current_duration` do globálneho poľa `PIT1_interrupt_duration` uchovávajúceho konfiguráciu parametrov pre dĺžku obsluhy ISR PIT1. Nastaví sa počiatočná frekvencia generovania záťažových prerušení zo zvolenej sady, a vynuluje sa index aktuálneho scenára. Tým sa začne, pre novú dĺžku ISR PIT1, vykonávať ďalšia sada testovacích scenárov a v nich jednotlivé sady testov.

Algoritmus 5: Algoritmus nastavenia nového scenára, v obsluhe PIT1, pre vykonávanie testov

```
1  if aktuálny scenár < počet scenárov then
2      nastav novú frekvenciu generovania záťažových prerušení v PIT1
3      inkrementuj index aktuálneho scenára
4  else if index aktuálneho trvania ISR PIT1 < počet trvaní ISR PIT1 then
5      nastav novú dĺžku obsluhy záťažového prerušenia PIT1
6      nastav počiatočnú frekvenciu generovania záťažových prerušení v PIT1
7      vynuluj index aktuálneho scenára
8  else
9      ukonči experiment
10 end if
```

4.2.3 Vyhodnotenie nameraných výsledkov

Výsledky meraní z tohto experimentu sú uložené na priloženom pamäťovom médiu (viď príloha č. A), v adresári `Experiment1` v podadresári `Results`. Merania sú rozdelené do viacerých súborov `PIT1_interrupt_duration`, pomenovaných podľa počtu cyklov vykonávaných v obsluhe ISR PIT1 časovača. Každý súbor obsahuje kompletnú sadu výsledkov z testovacích scenárov, pre všetky možnosti frekvencií generovania záťažových prerušení.

Vyhodnotenie experimentu prebiehalo na zvolenej platforme. Počet možností pre trvanie obsluhy prerušenia ISR PIT1 bol rovnako ako v predošlom experimente 16. Počet možností pre frekvenciu generovaných prerušení bol 26. Nakoľko testy sa vykonávali po desiatkach a výpočtová doba jedného testu bola 1 sekunda, celková doba priebehu testov tak presahovala 1 hodinu 9 minút a 20 sekúnd. Počas priebehu experimentu bolo nameraných 12 480 číselných hodnôt, z ktorých bolo 1 248 vizuálne spracovaných.

Textový výstup experimentu

Príklad výstupu jedného testovacieho scenára je možné vidieť na obrázku č. 4.4. Prvý riadok výstupu zachytáva aktuálnu konfiguráciu záťažového časovača. Nasledujúcich 10 riadkov zodpovedá jednotlivým testom v rámci testovacieho scenára. Riadky zobrazujú počet iterácií v hlavnej smyčke programu a počet prístupov do obsluhy záťažového prerušenia PIT1. Po dokončení desiatich testov nasleduje celkový výpis počtu taktov strávených v tele ISR PIT1 záťažového časovača počas vykonávania testov.

Vizualizácia nameraných výsledkov

Namerané hodnoty z výsledkových súborov boli prenesené do súboru `Experiment1.xlsx` v adresári `Výsledkové_grafy`, kde je možné vidieť presné namerané číselné hodnoty. Údaje boli z dôvodu zvýšenia informačnej hodnoty spracované do grafickej reprezentácie vo forme priestorových 3D grafov. Táto možnosť bola zvolená ako jedna z metód prehľadnej vizualizácie nameraných výsledkov. Na osiach spodnej podstavy sú vynesené hodnoty frekvencie generovania záťažových prerušení, udávané v Hz, a dĺžka obsluhy jedného prerušenia, udávaná v počte vykonaných cyklov. Je potrebné upozorniť, že hodnoty osí spodnej podstavy sú nanesené v logaritmickom merítku.

```
100000 us perioda prerusovacieho casovacu, PIT_LDVAL konstanta: 4999999
00. Hodnota priebehu testu: 4532375, PIT1_interrupt_load_access: 10
01. Hodnota priebehu testu: 9065694, PIT1_interrupt_load_access: 10
02. Hodnota priebehu testu: 13599013, PIT1_interrupt_load_access: 10
03. Hodnota priebehu testu: 18132332, PIT1_interrupt_load_access: 10
04. Hodnota priebehu testu: 22665651, PIT1_interrupt_load_access: 10
05. Hodnota priebehu testu: 27198970, PIT1_interrupt_load_access: 10
06. Hodnota priebehu testu: 31732289, PIT1_interrupt_load_access: 10
07. Hodnota priebehu testu: 36265608, PIT1_interrupt_load_access: 10
08. Hodnota priebehu testu: 40798927, PIT1_interrupt_load_access: 10
09. Hodnota priebehu testu: 45332246, PIT1_interrupt_load_access: 10

        MIN_main_iter_count: 4532375, MAX_interrupt_load_access: 10
        DWT_CYCCNT: 133038
        DWT_CYCCNT: 133037
        DWT_CYCCNT: 133037
        DWT_CYCCNT: 133037
        DWT_CYCCNT: 133037
        DWT_CYCCNT: 133037
        DWT_CYCCNT: 133037
        DWT_CYCCNT: 133037
        DWT_CYCCNT: 133037
        DWT_CYCCNT: 133037
        MAX_DWT_CYCCNT: 133038
```

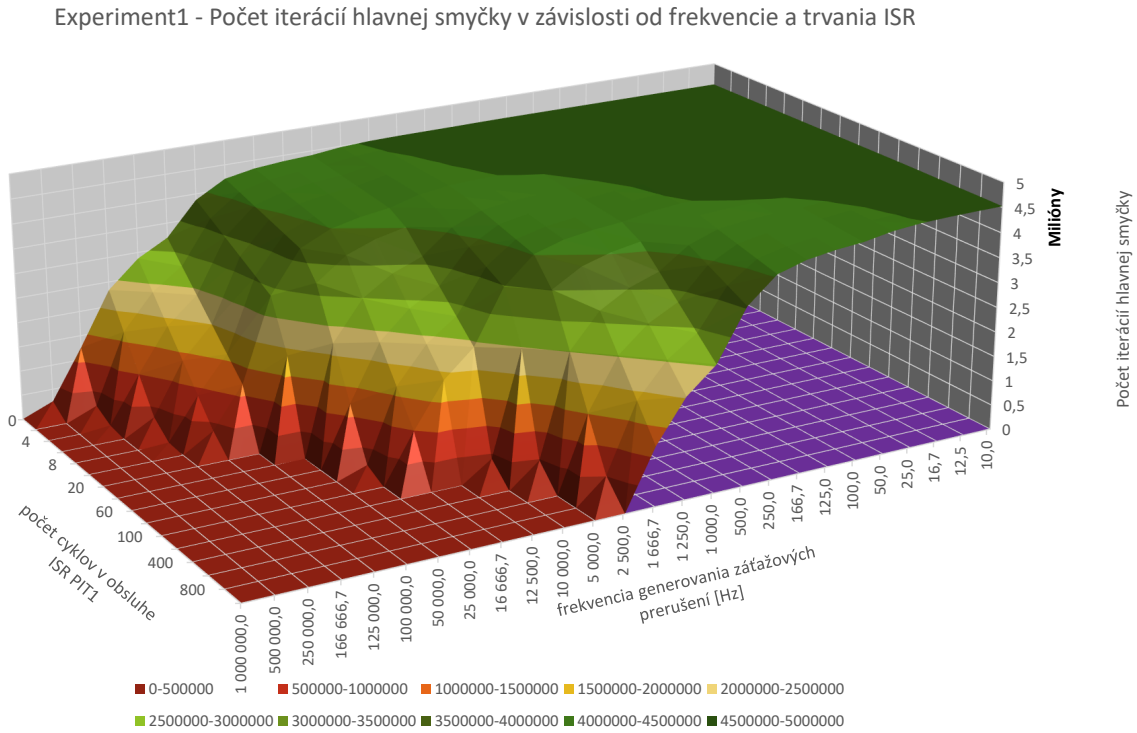
Obr. 4.4: Výstupné hodnoty jedného testovacieho scenára pre 100 ms periódu generovania prerušení a 600 iteráciami oneskorovacieho cyklu v ISR PIT1.

Prvý graf na obrázku č. 4.5, zobrazuje množstvo cyklov, ktoré je schopné procesorové jadro vykonať počas jednej sekundy (doba vykonávania testu) pri súčasnom obsluhovaní vyvolaných prerušení. Počet cyklov je možné pomocou výsledkov z predošlého experimentu, v kapitole č. 4.1.3, prepočítať na počet taktov strávených v obslužnej rutine.

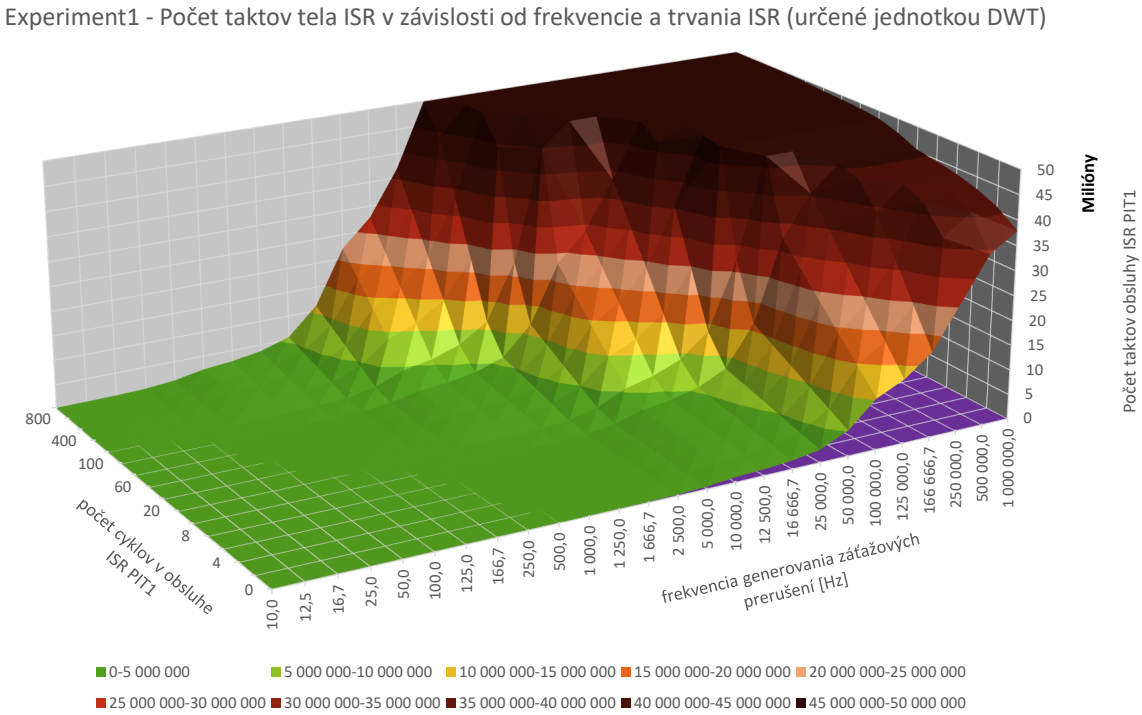
Tmavo zelená oblasť vyznačuje situáciu, kedy je napriek obsluhe prerušení k dispozícii dostatok procesorového času na vykonávanie hlavného programu. Z grafu je možné vidieť, že po dosiahnutí určitého limitu v dĺžke obsluhy a frekvencii generovania prerušení začne počet iterácií vykonaných v hlavnom programe klesať. Táto hodnota v najhoršom prípade môže klesnúť až na nulu, viď červená oblasť grafu. Situácia nastane vtedy, ak je frekvencia generovania prerušení dostatočne vysoká a je potrebné obslúžiť veľké množstvo prerušení, alebo v prípade, kedy obsluha jednotlivých prerušení spotrebuje veľké množstvo procesorového času.

V grafe je možné si všimnúť, že vyťaženie klesá na úroveň nuly najskôr pri vykonaní malého množstva dlhotrvajúcich prerušení s viac ako 800 iteráciami zaťažovacieho cyklu v ISR (17 713 taktov), a to už pri frekvencii 5 kHz, čo zodpovedá perióde 200 μs. Naopak hodnota poklesne na nulu neskôr pri generovaní kratších prerušení s počtom cyklov od 0 do 8, čo zodpovedá maximálne 289 taktom v tele obslužnej rutiny. V tomto prípade je potrebná frekvencia prichádzajúcich prerušení nad 500 kHz, čo zodpovedá perióde 2 μs.

V grafe na obrázku č. 4.6 je možné vidieť množstvo hodinových taktov procesorového jadra, ktoré boli počas intervalu jednej sekundy venované obsluhe záťažových prerušení PIT1. Počet taktov bol meraný komponentov DWT. Zelená oblasť grafu zobrazuje časť, kedy je frekvencia a trvanie prerušení prijateľné. Táto oblasť postupne prechádza do červenej, kde sa počet taktov postupne zvyšuje. Tmavočervená oblasť znázorňuje maximálny počet taktov strávených v ISR počas doby trvania testu. Je možné taktiež vidieť, že pre krátku dobu trvania ISR a vysokú frekvenciu prerušení táto plocha nedosahuje maximum, čo je spôsobené taktami spotrebovanými celkovou réžiou obslužných rutín, nakoľko dochádzalo k obsluhe veľkého množstva prerušení, a bolo potrebné časté prepínanie kontextu.

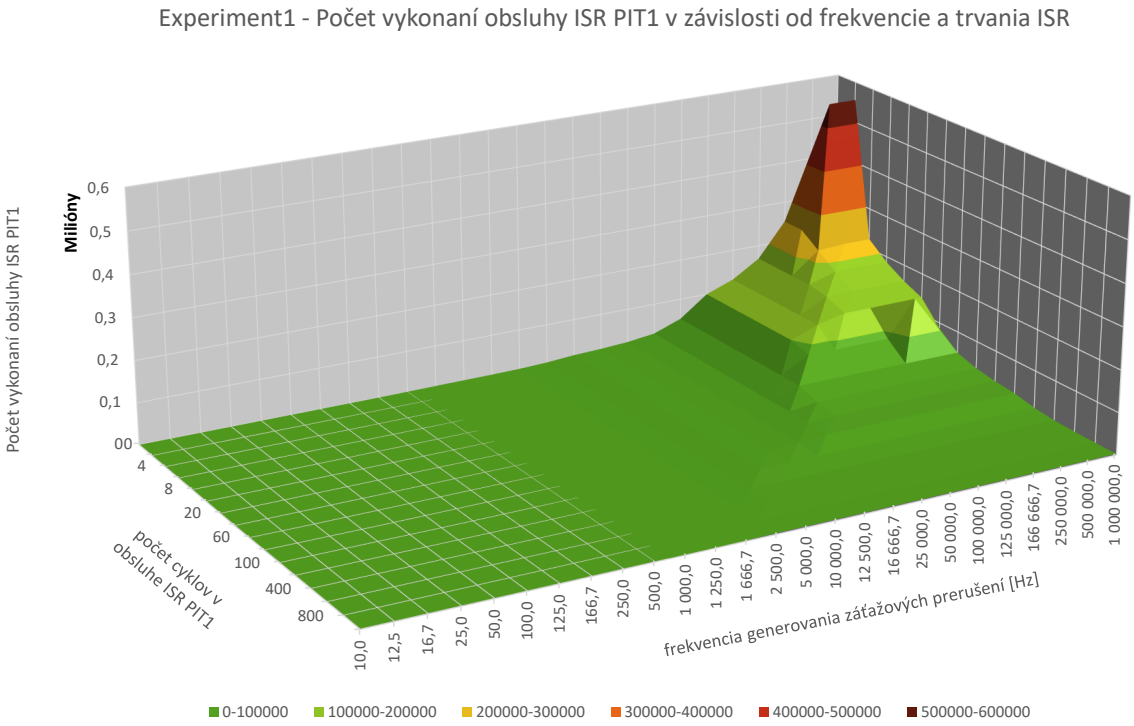


Obr. 4.5: Minimálny počet iterácií hlavného programu v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1.



Obr. 4.6: Maximálny počet taktov strávených obsluhou prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1.

V nasledujúcom grafe na obrázku č. 4.7, je zobrazený práve počet vykonaných obslúh prerušení počas doby trvania jedného testu. Z grafu je možné vidieť, že práve najväčší počet prerušení 555 555, počas jednej sekundy, bolo vykonaných pri krátkej dobe obsluhy ISR (nulový počet vykonaných cyklov v ISR) a najväčšej frekvencii generovaných prerušení. Táto oblasť je v grafe znázornená červenou farbou.



Obr. 4.7: Maximálny počet obslúžených prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1.

Rovnomernosť prírastku v grafoch na obrázkoch č. 4.6 a č. 4.7 je možné vidieť na obrázkoch v prílohách č. B.1 a č. B.2, v ktorých sú všetky osi grafov zobrazené logaritmicky.

V týchto grafoch je taktiež možné si všimnúť mierne odchýlky a dislokácie v nameraných hodnotách. Najviditeľnejšia napríklad pri hodnote 20 iteráciach a frekvencii 500 kHz (tj. 2 μs). Snahou bolo zistiť príčinu vzniku týchto nepresností a tieto odchýlky v čo najväčšej miere eliminovať.

Problémy vzniknuté pri meraní výsledkov

Predpoklad pri implementácii a vykonávaní testov bol, že namerané hodnoty sa budú medzi jednotlivými testami mierne líšiť. Z tohto dôvodu bol každý z testov vykonávaný desaťkrát a vždy sa vyberalo maximum alebo minimum podľa meranej veličiny. Výsledky experimentu však ukázali, že počet vykonaní obsluhy ISR PIT1 časovača pri zvolenej frekvencii je vždy rovnaký. Taktiež počet taktov strávených v obslužnej rutine prerušenia bol v každom teste približne rovnaký. Neočakávaným výsledkom bolo, že počet iterácií v tele hlavného programu sa medzi jednotlivými testami líši rádovo v násobkoch hodnoty prvého testu z príslušnej sady testov, viď obrázok č. 4.4. Na druhom riadku je možné vidieť, že výsledok prvého testu je 4 532 375, pričom výsledky ďalších testov sa pohybujú v približných násob-

koch. Pre rôzne scenáre sa táto hodnota nedefinovane odlišovala, aj keď bola vždy medzi každým testom vynulovaná.

Výsledky experimentu naznačovali, že nedochádza k vynulovaniu čítača pre počet vykonaných iterácií v hlavnej smyčke aj keď počet obslúžených prerušení a počet taktov strávených v obsluhe prerušení sa nemenil. Situácia nastávala aj napriek tomu, že bola táto premenná uvedená kľúčovým slovom `volatile`, ktoré sa musí použiť v prípade ak dochádza k modifikácii ľubovolného objektu z tela ISR.

Z tohto dôvodu bola implementácia doplnená o výpočet minimálneho počtu cyklov hlavnej smyčky, maximálneho počtu taktov strávených v ISR a maximálneho počtu vykonaných obslúh záťažového prerušenia PIT1. Cieľom bolo získať z odlišných hodnôt čo najrelevantnejšie výsledky, ktoré odpovedajú prípadom, kedy dôjde počas jedného testu k najväčšiemu počtu vykonaných prerušení spomedzi celej sady testov v rámci jedného scenára, a zároveň určiť minimálnu hranicu pre počet cyklov vykonaných v hlavnom programe (WCET). Tým stanoviť aspoň približné limity pre okrajové situácie zaznamenávané experimentom.

4.2.4 Modifikácia experimentu

Pri snahe odstrániť samotný vznik neželaných odchýlok, ktoré sa vyskytovali pri vyhodnocovaní experimentov bolo implementovaných niekoľko upravených verzií pôvodného experimentu.

Prvá verzia implementovala nahradenie časovača PIT0, ktorý v experimente slúžil na meranie intervalu doby vykonávania jednotlivých testov. Časovač bol nahradený iným dostupným časovačom RTC, ktorý poskytuje zvolená platforma a disponuje požadovaným časovým rozsahom. RTC časovač bol podľa referenčného manuálu (viď literatúra č. [9], kapitola 44), nakonfigurovaný na meranie intervalu jednej sekundy. Nahradením časovača však nedošlo k odstráneniu vzniknutého problému.

Ďalšou variantou bolo pridanie inštrukčných synchronizačných bariér `__ISB()` pre prístup k premennej `mainq_iter_count`, viď obrázok č. 4.8. Bariéry boli pridané ako v hlavnom toku programu tak aj v tele obslužnej rutiny prerušenia. Detailný popis inštrukčnej synchronizačnej bariéry je popísaný v dokumente ARM C Language Extensions (ACLE) [7]. Synchronizačná bariéra spôsobí vykonaním inštrukcie ISB vyprázdnenie zreťazenej linky procesora, čo znamená, že všetky inštrukcie, nasledujúce po dokončení ISB, je potrebné znovu načítať z pamäte cache.

```
1. __ISB();
2. main_iter_count++;
3. __ISB();
```

Obr. 4.8: Ilustrácia použitia inštrukčných synchronizačných bariér pre prístup k premennej uchovávajúcej počet vykonaných iterácií v tele hlavného programu.

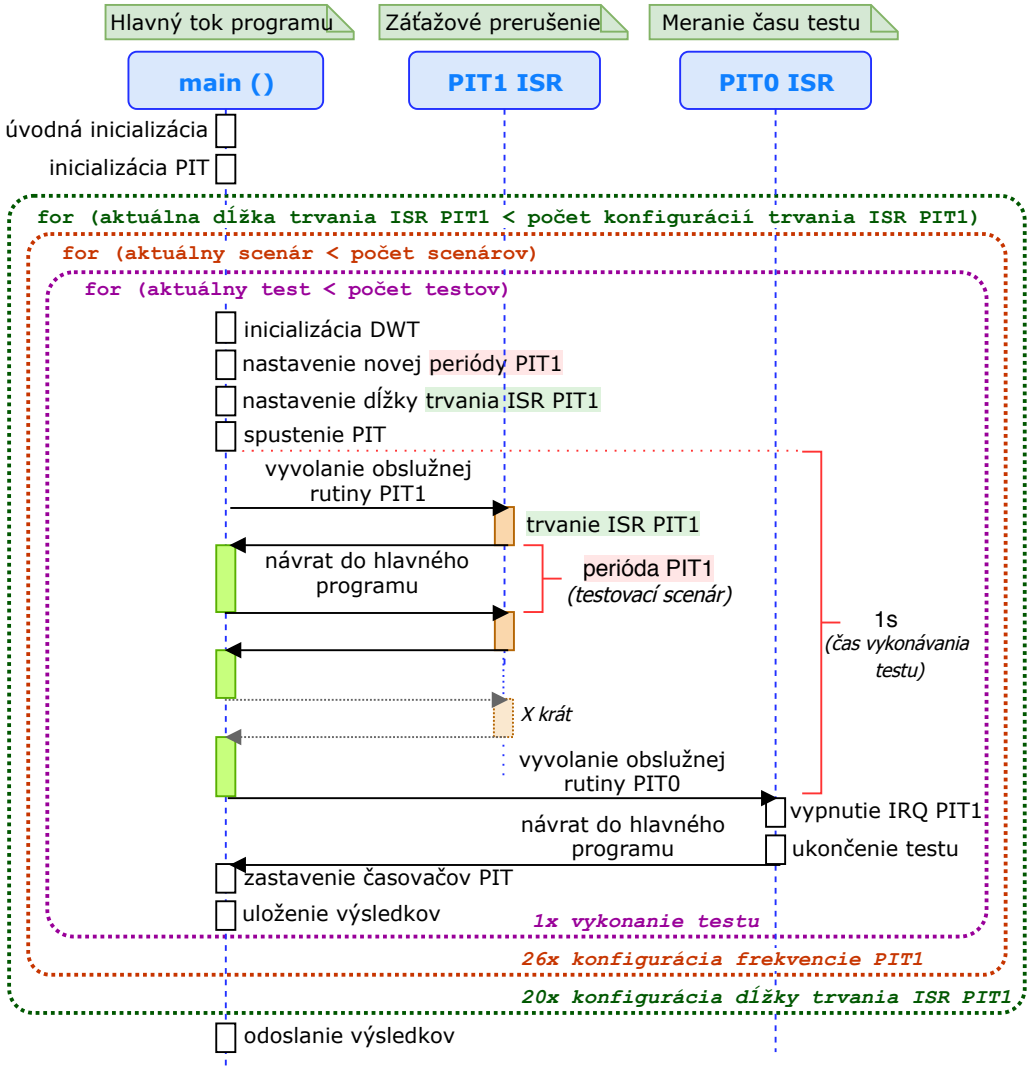
K odchýlkam vo výsledkoch po tejto úprave nedochádzalo tak často v porovnaní s pôvodnou verziou. Odchýlky však dosahovali výraznejšie rozdiely, a nebolo možné ich úplne eliminovať. Táto modifikácia je dostupná v prílohách na priloženom pamäťovom médiu, viď príloha A. Z tohto dôvodu bola zvolená úplne iná koncepcia experimentu popísaná v kapitole č. 4.3.

4.3 Experiment 2 - Stanovenie limitov výpočtového preťaženia

Experiment sa zameriava na stanovenie limitu výpočtového preťaženia spôsobeného vplyvom nadmernej frekvencie prerušení. Cieľom v poradí tretieho experimentu bolo zmeniť koncepciu priebehu experimentu z predošlej varianty a odstrániť nielen vznik neželaných odchýlok, ale aj potrebu vykonávania viacerých testov s rovnakým nastavením počiatočných parametrov.

4.3.1 Návrh experimentu

Hlavná myšlienka experimentu spočíva v presunutí riadenia jednotlivých testov z obslužnej rutiny časovača PIT0, ako to bolo v predošlom experimente, do hlavného toku programu, viď. obrázok č. 4.9. Ďalším významným rozdielom v porovnaní s predošlým experimentom je uchovávanie všetkých nameraných výsledkov počas priebehu experimentu v pamäti, pričom výpis výsledkov bude prebiehať až na záver, po dokončení vykonávania všetkých testovacích scenárov.



Obr. 4.9: Sekvenčný diagram zobrazujúci priebeh a riadenie experimentu 2.

Predpokladom je, že presunutím riadenia experimentu do hlavného toku programu a odosielaním hodnôt po dokončení všetkých experimentov sa dosiahne spresnenie nameraných výsledkov a odstránenie nechcených javov prítomných v predošlom experimente.

Zmena logiky experimentu prispeje k sprehľadneniu jednotlivých fáz vykonávaných počas experimentu. Taktiež týmto krokom dôjde k skráteniu vykonávania obslužnej rutiny PIT0 časovača, ktorý tak bude slúžiť len na spúšťanie a zastavovanie experimentov.

Kanál časovača PIT1 bude zohrávať rovnakú úlohu generovania záťažových prerušení, objektom meraní budú rovnaké veličiny a meranie bude zabezpečované pomocou komponenty DWT, rovnako ako v predošlom experimente. Vstupné parametre pre konfiguráciu PIT časovača budú zhodné s predošlým experimentom. Časový interval jedného testu zostane taktiež rovnaký.

4.3.2 Softvérová implementácia experimentu

Implementácia experimentu vychádza z implementácie predošlého experimentu a aplikuje úpravy z kapitoly návrhu 4.3.1. Úpravy sa týkali najmä hlavnej riadiacej logiky experimentu. Experiment bol implementovaný ako samostatný Kinetis SDK projekt, ktorý je dostupný na priloženom pamäťovom médiu v adresári s názvom `Experiment2`. Zjednodušený algoritmus č. 6 zachytáva postupnosť krokov experimentu. Je v ňom možné vidieť spomenuté modifikácie v porovnaní s predošlým experimentom. Zároveň týmito úpravami vznikli iné problémy, ktoré bolo potrebné pri implementácii ošetriť.

Algoritmus 6: Algoritmus experimentu skúmajúceho limity výpočtového preťaženia vplyvom nadmernej frekvencie prerušení

```
1  inicializuj generátor hodín (MCG) do režimu BLPE
2  vypni watchdog
3  inicializuj indikačné diódy LED
4  inicializuj PIT časovač
5  for k ← 0 to k < COUNT_OF_PIT1_DURATIONS do
6      for j ← 0 to j < COUNT_OF_PIT0_SCENARIOS do
7          for i ← 0 to i < TEST_COUNT do
8              nastav index aktuálneho merania
9              inicializuj komponentu DWT
10             nakonfiguruj a spusti PIT časovač
11             nakonfiguruj v NVIC prioritu prerušení
12             while PIT1_ISRevent_flag do        // Prebieha obsluha prerušení
13                 inkrementuj počet vykonaných iterácií
14             end while
15             zakáž v NVIC prerušenia od PIT
16             vypni časovače PIT z dôvodu zmeny periódy
17             zastav meranie pomocou komponenty DWT
18             ulož namerané výsledky
19         end for
20     end for
21 end for
22 inicializuj sériovú komunikáciu cez UART
23 odošli namerané výsledky cez UART
```

Úvodná inicializácia

Kroky, z ktorých pozostáva úvodná inicializácia, sú identické s predošlým experimentom (algoritmus č. 6, riadky 1 až 4). Medzi tieto kroky patrí inicializácia generátora hodín, vypnutie watchdogu a inicializácia indikačných LED diód. Inicializácia PIT časovača je vykonávaná pomocou funkcie `init_PIT()`, ktorá má v tomto prípade za úlohu iba inicializovať časovač povolením hodín, zapnutím samotného modulu a vypnutím zreťazenia. Konfigurácia a spustenie časovača je vykonávané až pri nastavení parametrov konkrétneho testu.

Riadenie vykonávaných testov z hlavného programu

Hlavným rozdielom v porovnaní s predošlým experimentom je v presunutí riadenia priebehu jednotlivých testov z obsluhy PIT0 do hlavného programu. Prvý cyklus, s riadiacou premennou k, slúži pre počítanie indexu aktuálneho nastavenia dĺžky obsluhy záťažového prerušenia z PIT1 časovača. Druhý cyklus (algoritmus č. 6, riadok 6) slúži na iterovanie medzi jednotlivými konfiguráciami frekvencií generovania záťažových prerušení. Posledný cyklus (algoritmus č. 6, riadok 7) slúži na iterovanie po jednotlivých testoch pri rovnakej konfigurácii vstupných parametrov. V hlavnom programe sú podľa riadiacych premenných k, j a i nastavené počítadlá indexov pre vykonávané testy.

Telo vnútorného cyklu hlavného programu najskôr inicializuje komponentu DWT, ktorú je potrebné inicializovať pred každým začatím vykonávania jedného testu (algoritmus č. 6, riadok 9). Inicializácia je vykonávaná pomocou funkcie `init_DWT()`, ktorá je podrobne popísaná v kapitole č. 4.1.2. Rozdiel však spočíva v tom, že z inicializácie bola odstránená konfigurácia zvyšných registrov, ktoré sa ukázali ako nepotrebné pre vykonávanie meraní v experimente.

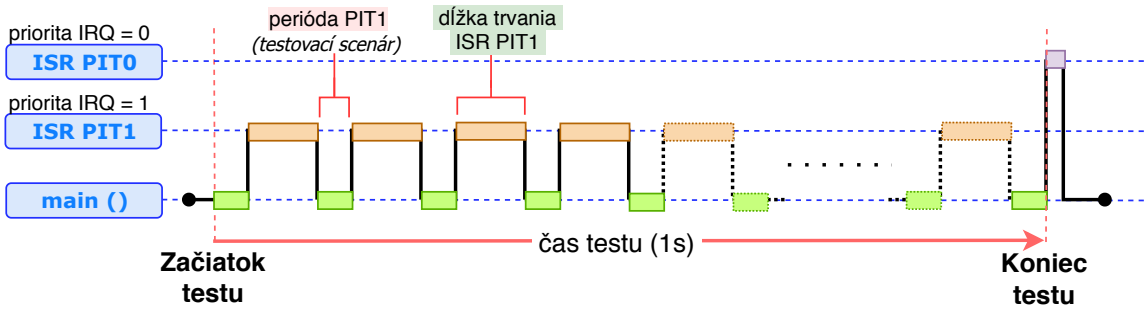
Nasledujúcimi krokmi v tele cyklu sú nastavenie periódy jednotlivým časovačom (algoritmus č. 6, riadok 10). Perióda časovača PIT0 zostala na základe rovnakej úvahy ako v predošlom experimente na hodnote jednej sekundy. Z hľadiska možnosti porovnania vzájomných výsledkov medzi experimentami sa nezmenili ani parametre konfigurácie PIT1 časovača. Nastavenie oboch časovačov je s každým spustením nového scenára rekonfigurované. Po nastavení a spustení časovačov dôjde v radiči NVIC ku konfigurácii priority prerušení generovaných PIT modulom (algoritmus č. 6, riadok 11). Opäť musí byť priorita časovača PIT0 väčšia, nastavená na hodnotu 0, ako priorita PIT1 časovača nastavená na úroveň 1.

Priebeh testu

Za konfiguráciou časovačov je umiestnená čakacia smyčka reprezentujúca vykonávanie primárnej výpočtovej úlohy v tele hlavného programu procesorovým jadrom (algoritmus č. 6, riadky 12 až 14). Toto zjednodušenie je popísané v predošlom experimente, viď kapitola č. 4.2.1. Vykonávanie cyklu je riadené globálnou premennou `PIT0_ISRevent_flag`, ktorá je uvedená kľúčovým slovom `volatile` nakoľko jej nastavovanie prebieha z tela ISR PIT0. V tele cyklu prebieha inkrementácia premennej `main_iterq_count` pre počítanie vykonaných cyklov, obdobne ako v predošlom experimente. Zároveň s vykonávaním cyklu v hlavnom programe sú vykonávané obslužné rutiny záťažových prerušení PIT1 časovača.

Ukončenie vykonávania cyklu nastane po vyvolaní prerušenia PIT0 pre ukončenie aktuálneho testu, viď obrázok č. 4.10. Hlavným rozdielom v porovnaní s predošlým experimentom je, že v obsluhe prerušenia PIT0, ktoré ukončuje beh testu, sa v tomto prípade nastaví

premenná `PIT0_ISRevent_flag` na nulu, čo spôsobí, že po jej dokončení dôjde k ukončeniu vykonávania cyklu while v hlavnom programe.



Obr. 4.10: Zobrazenie vykonávania záťažových ISR PIT1 počas priebehu jedného testu.

Jedným z problémov, ktoré bolo potrebné ošetriť, bolo vypnutie generovania žiadostí o prerušenia záťažového časovača PIT1 ešte v obslužnej rutine časovača PIT0, nastavením bitu TIE kontrolného registra TCTRL kanálu 1 na hodnotu nula. Tým za zamedzí generovaniu záťažových prerušení po uplynutí časového intervalu jedného testu. V prípade ak by sa tak nestalo, pri nadmernom generovaní záťažových prerušení PIT1 s periódou 1 μs, dochádza iba k obsluhe prerušení a nikdy tak nedôjde k overeniu podmienky pre ukončenie cyklu hlavného programu, čím by zároveň nikdy nedošlo k ukončeniu priebehu testu. Po dokončení testu je zakázaná obsluha prerušení v radiči NVIC. Taktiež sú vypnuté oba časovače, nakoľko ďalšia rekonfigurácia ich periódy je možná iba vo vypnutom stave (algoritmus č. 6, riadky 15 až 16).

Uloženie a odoslanie nameraných výsledkov

Po dokončení vykonávania testu nasleduje fáza ukladania výsledkov (algoritmus č. 6, riadok 18). Pre uloženie nameraných hodnôt sú implementované a použité tri funkcie. Funkcie uložia hodnoty do pripravenej štruktúry pre ukladanie výsledkov.

- Funkcia `store_DWT_result()` – uloží hodnotu z CYCCNT registra DWT komponenty, počítajúcu takty strávené obsluhou záťažových prerušení PIT1.
- Funkcia `store_iteration_result()` – uloží hodnotu premennej `main_iter_count`, počítajúcu vykonané iterácie v hlavnom programe.
- Funkcia `store_PIT1_access_result()` – uloží aktuálny počet vykonaných prerušení ako hodnotu z premennej `PIT1_interrupt_access_counter`.

Tieto údaje sú počas experimentu ukladané v štruktúre typu `result`. Významným rozdielom v porovnaní s predošlým experimentom je potreba ukladať všetky výsledky meraní počas vykonávania experimentu, nakoľko sú odosielané cez UART až po dokončení všetkých testov (algoritmus č. 6, riadky 22 a 23).

V tomto experimente, narozdiel od predošlých, štruktúra `result` obsahuje tri trojrozmerné polia dátového typu `uint32_t`.

1. Prvá dimenzia slúži pre uloženie výsledkov vykonanej sady testov.
2. Druhá dimenzia slúži pre ukladanie výsledkov rozdielnej konfigurácie testovacích scenárov.

3. Tretia dimenzia slúži pre uloženie výsledkov rozdielneho nastavenia dĺžky trvania záťažového prerušenia.

Spolu sú v štruktúre ukladané aktuálne indexy každej z troch dimenzií. Tieto indexy sú aktualizované pred každým spustením testu funkciou `set_measuerment_indexes()` a uchovávajú index pozície, na ktorú budú v štruktúre uložené výsledky aktuálneho merania (algoritmus č. 6, riadok 8).

Po dokončení všetkých testov a po uložení nameraných výsledkov sú hodnoty odosielané prostredníctvom formátovacích funkcií. Formátovacie funkcie `print_DWT_result_CSV()`, `print_iteration_result_CSV()` a funkcia `print_PIT1_access_result_CSV()` odosielajú údaje vo formáte CSV, ktorý je vhodný na ďalšie spracovanie v tabuľkovom procesore.

4.3.3 Vyhodnotenie nameraných výsledkov

Výsledky namerané v tomto experimente sú dostupné v adresári `Experiment2` v podadresári `Results`, na priloženom médiu. Výsledky meraní sú rozdelené v trojici CSV súborov. Prvý súbor s názvom `Počet iterácií hlavnej smyčky.csv` obsahuje hodnoty iterácií hlavného toku programu, ktoré boli vykonané počas priebehu jednotlivých testov. Druhý súbor s názvom `Počet taktov tela ISR PIT (namerané DWT).csv` obsahuje počet taktov strávených v obslužnej rutine záťažového prerušenia PIT1 a tretí súbor s názvom `Počet vykonaní obsluhy ISR PIT1.csv` obsahuje hodnoty s počtom vykonaných obslužných rutín PIT1 časovača počas vykonávania testov.

Popis výstupných súborov

Každý zo súborov obsahuje údaje konkrétnej meranej veličiny. Rozdelenie a vnútorná štruktúra týchto súborov je však v každej z týchto troch verzií rovnaká. Tabuľka v súbore obsahuje 22 stĺpcov a 27 riadkov. Prvý riadok obsahuje nadpis. V nasledujúcom riadku sa nachádza popis hlavičiek jednotlivých stĺpcov.

Prvý stĺpec obsahuje hodnoty periódy generovania záťažových prerušení PIT1. Tieto hodnoty sú v rozmedzí od 100 ms do 1 μs. Intervaly medzi hodnotami sú stanovené rovnako ako v Experimente 1, viď kapitola č. 4.2.2. Každý riadok v súbore zodpovedá jednému nastaveniu testovacieho scenára.

Druhý stĺpec je vyhradený pre dopočítanie frekvencie ako prevrátenej hodnoty periódy z prvého stĺpca. Ďalších 20 stĺpcov v rozmedzí 0 až 8000 určuje aktuálne nastavenie dĺžky trvania jednej obslužnej rutiny, záťažového časovača PIT1, v počte vykonaných cyklov. Hodnota v príslušnom riadku a stĺpci určuje výsledok testu na zvolenej platforme pre danú konfiguráciu frekvencie a dĺžky trvania prerušení počas jedného testu.

Odstránenie duplicitného vykonávania testov

Po vykonaní viacerých nezávislých spustení experimentu a vzájomnom porovnaní výsledkov bolo možné dospieť k záveru, že hodnoty boli pri každom spustení rovnaké a nedochádzalo tak u nich k neželaným náhodným odchýlkam ako v prípade predošlého experimentu. Z tohto dôvodu bol zároveň minimalizovaný počet vykonávaných testov. V porovnaní s predošlým experimentom, kde bolo potrebné pre každú konfiguráciu vykonať najmenej 10 iterácií testov, z ktorých sa následne vybral najhorší možný prípad, bol v tomto experimente počet testov stanovený na jeden test pre každú konfiguráciu, nakoľko sa hodnoty viacerých testov navzájom neodlišovali. Presunutím riadiacej logiky do hlavného programu

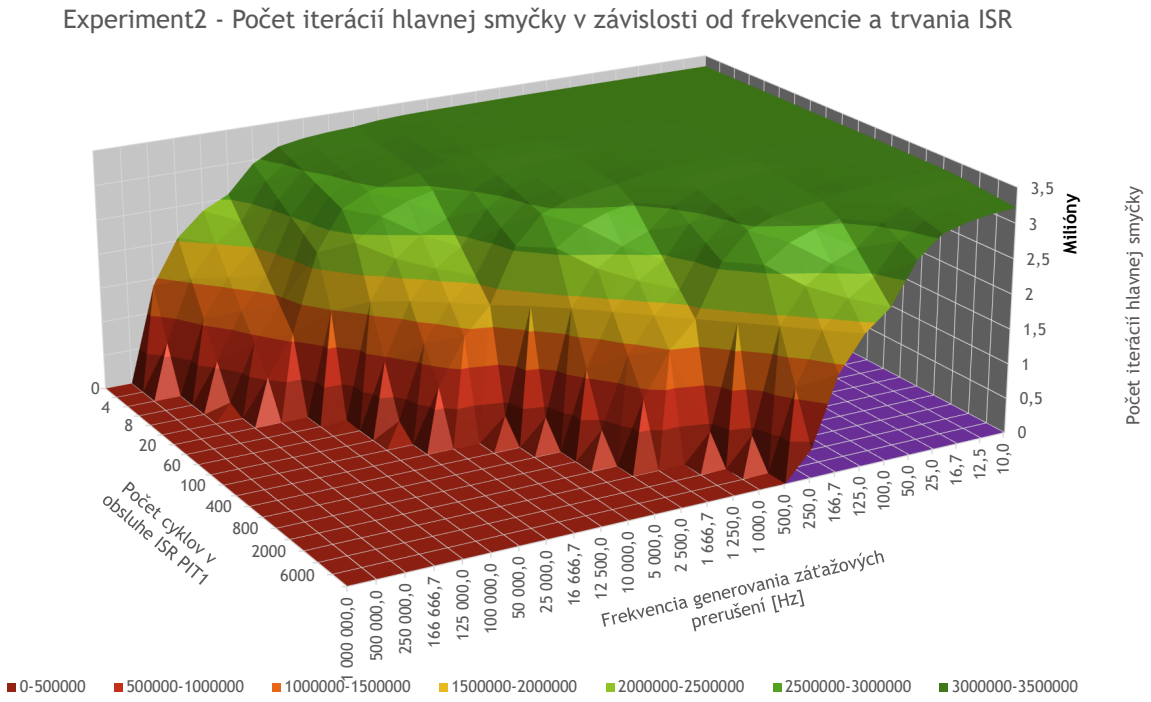
a presunutím odosielania výsledkov na záver po dokončení testov tak došlo k odstráneniu neželaného neočakávaného chovania programu z predošlého experimentu.

Nakoľko v tejto implementácii nedochádzalo k odchýlkam medzi jednotlivými testami rovnakého scenára, postačovalo vykonanie jedného testu pre každú konfiguráciu parametrov. Tým sa zároveň výrazne skrátila doba vykonávania experimentu z viac ako jednej hodiny v predošlom prípade, na necelých 9 minút. V jednom experimente prebiehalo 26 testovacích scenárov pre 20 rôznych konfigurácií dĺžky obsluhy prerušenia PIT1, pričom každý z testov trval jednu sekundu. Pri jednom spustení Experimentu2 s rovnakým množstvom konfigurácií ako v predošlom experimente bolo nameraných 1 560 hodnôt, ktoré boli vizuálne spracované do podoby grafov.

Vizualizácia nameraných výsledkov

Zobrazované grafy sú analógiou korešpondujúcich grafov z predošlého experimentu. Zobrazujú rovnaké merané veličiny. V grafoch je možné vidieť, že neobsahujú vyššie spomenuté odchýlky v meraných hodnotách a odstraňujú nedostatok potreby vykonávania viacerých testov s rovnakou konfiguráciou. Vďaka tomu, že pre získanie hodnôt tak nebolo potrebné vykonávanie viacero nezávislých spustení testov, a hodnoty bolo možné namerať jedným testom pre každú konfiguráciu, bol mierne navýšený aj počet možností konfigurácie pre počet vykonaných cyklov v ISR PIT1.

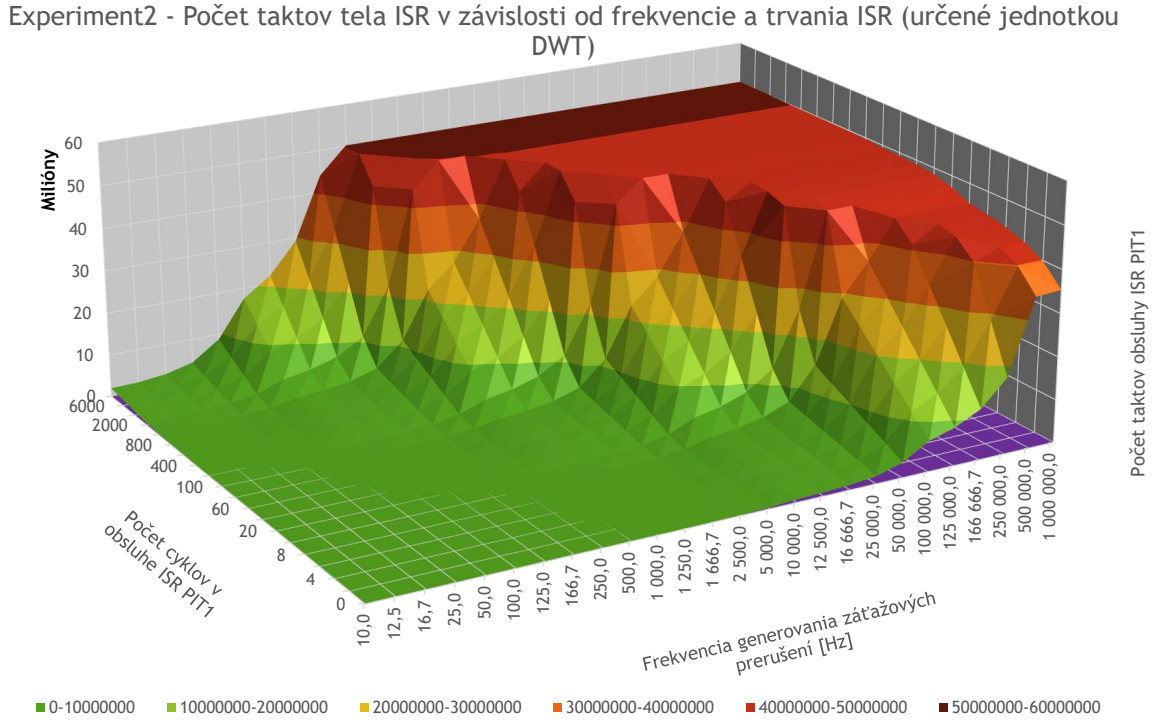
V prvom grafe na obrázku č. 4.11, je zobrazený počet vykonaných iterácií cyklu v tele hlavného programu, počas doby vykonávania jedného testu, pri súčasnej obsluhe záťažových prerušení PIT1. Počet cyklov je možné rovnako previesť na počet taktov podľa výsledkov v kapitole č. 4.1.3.



Obr. 4.11: Počet iterácií hlavného programu v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1.

Tmavozelená oblasť rovnako ako v predošlom experimente vyznačuje situáciu kedy je procesor napriek obsluhe prerušení schopný vykonávať dostatočný počet inštrukcií v tele hlavného programu. Naopak tmavo červená oblasť po prekročení určitej hranice klesne až na nulovú hodnotu, kde je procesor maximálne vyťažený obsluhou prerušení a nedokáže vykonávať hlavný program.

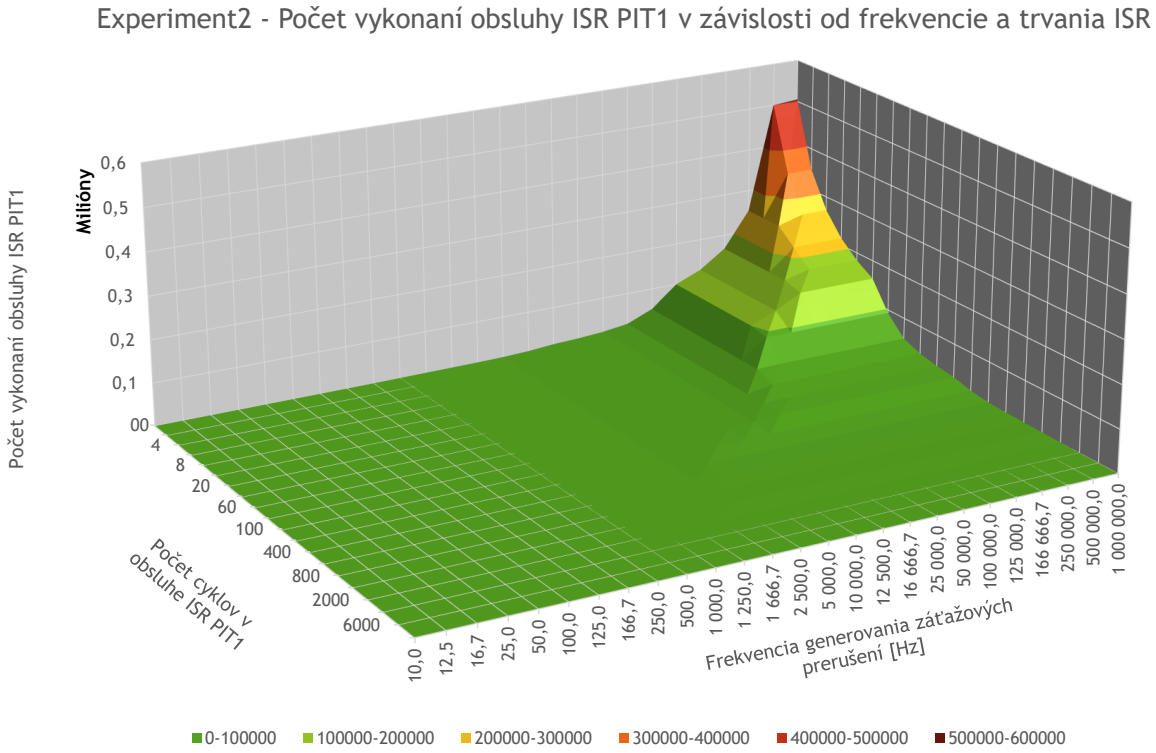
Graf na obrázku č. 4.12 zobrazuje hodnoty množstva procesorových taktov, ktoré procesor využíval počas jednej sekundy k obsluhe záťažových prerušení. Zelená oblasť zobrazuje hodnoty kedy je počet takov v obsluhe ISR PIT1 z hľadiska obsluhy hlavného programu prijateľný. Červená oblasť zobrazuje limit, kedy dochádza k zahlteniu procesorového jadra obsluhou prerušení. V pravej časti grafu, pre hodnoty minimálneho počtu cyklov v ISR PIT1 a maximálnu frekvenciu, je možné presnejšie vidieť pokles, oproti maximálnemu počtu vykonaných taktov, spôsobený réžiou vykonávaných prerušení.



Obr. 4.12: Počet taktov strávených obsluhou prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1.

V poslednom grafe na obrázku č. 4.13 je zobrazené množstvo vykonaných ISR PIT1 v priebehu jedného testu. Najvyššia hodnota zodpovedá konfigurácii kde došlo v predošlom grafe k poklesu z dôvodu zvýšenia réžie spojenej s obsluhou prerušení. V tomto grafe je najzreteľnejšie viditeľné odstránenie neželaných odchýlok v porovnaní s predošlým experimentom číslo 1, čím bolo možné zredukovať počet vykonaných testov a zároveň výrazne aj čas vykonávania experimentu, viď príloha č. G.

Maximálne hodnoty z predošlých grafov experimentu sú zároveň zobrazené v tabuľke č. 4.1. Grafy sú zobrazené v logaritmickom merítku v prílohách č. C.1 a C.2, kde je možné výraznejšie vidieť závislosti medzi nameranými hodnotami príslušných veličín. Všetky grafy experimentu sú dostupné v súbore `Experiment2.xlsx` na priloženom pamäťovom médiu, viď príloha A.



Obr. 4.13: Počet obslúžených prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1.

	Počet iterácií v main()	**Počet vykonaných ISR PIT1**	**Počet taktov obsluhy ISR PIT1**	Frekv. IRQ PIT1	Počet cyklov v jednej ISR PIT1
Maximálny počet iterácií v main() pri minimálnom zaťažení CPU prerušeniami	3 333 249	10	719	10 Hz	0 (113 taktov)
Maximálny počet vykonaných ISR PIT1 prerušení	0	505 050	35 353 588	1 MHz	0 (113 taktov)
Maximálny počet taktov vykonávania obsluhy ISR PIT1	0	285	50 180 040	1 MHz	8 000 (176 095 taktov)

Tabuľka 4.1: Prehľad maximálnych hodnôt z nameraných výsledkov experimentu.

Pretečenie registra EXECCNT komponenty DWT

Pri implementácii bola snaha zaznamenávať zároveň pomocou komponenty DWT a jej registra EXCCNT aj počet taktov strávených réžiou spojenou s vykonávaním obsluhy záťažových prerušení časovača PIT1. Register EXCCNT však k ukladaniu taktov réžie z 32 bitovej šírky využíva iba 8 bitov, a táto hodnota nie je dostatočná pre test vykonávaný počas intervalu jednej sekundy, nakoľko dochádzalo k pretečeniu hodnoty registra. Z tohto dôvodu bola táto možnosť neskôr z experimentu odobratá. Dôkaz o pretečení registra EXCCNT je uložený v súbore `vysledky DWT_EXCCNT.txt`, ktorý zaznamenáva hodnoty registra pre všetky konfigurácie frekvencií časovača PIT1, viď príloha č. A.

Rozšírený počet konfigurácií vstupných parametrov

Experiment bol taktiež spustený s rozšíreným počtom konfigurácií vstupných parametrov pre zjemnenie vykresľovanej siete stavového priestoru. Pri modifikácii došlo k zmenšeniu intervalu dĺžky obsluhy ISR PIT1 z 8 000 iterácií na 1 000 iterácií. Zároveň bol ale zvýšený počet možných nastavení dĺžky trvania obsluhy ISR PIT1 na 29 hodnôt, čím došlo k zjemneniu siete vykresľovanej v grafoch, pre možnosť detailnejšieho zobrazenia.

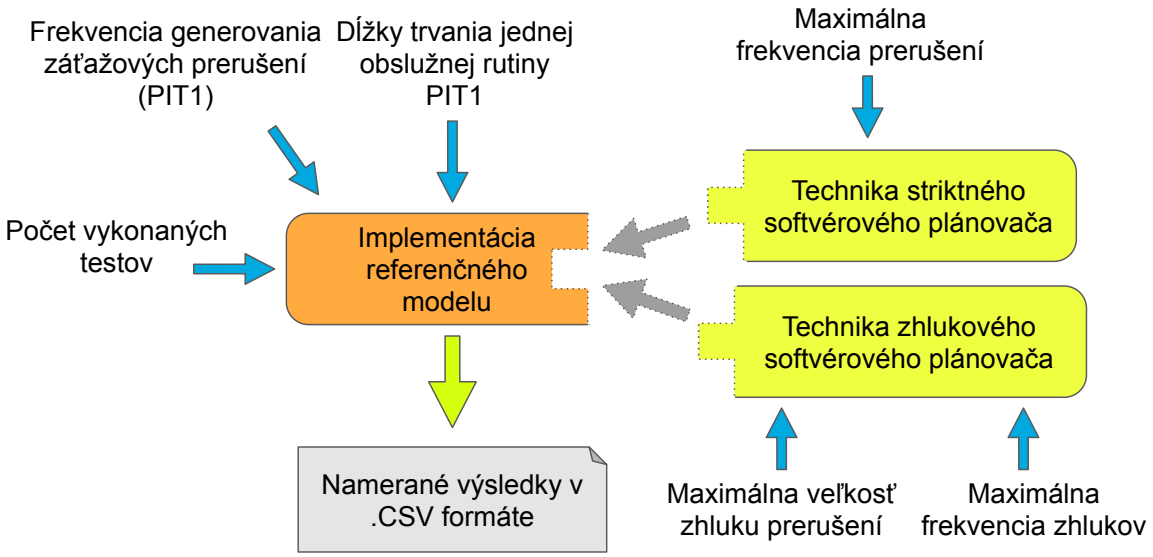
Výsledky tejto konfigurácie sú uložené v adresári `Results` v rovnomenných súboroch s predponou `29prvkov`. Grafy zobrazujúce detailnejší pohľad s menším rozostupom prvkov sú zobrazené v prílohách, viď príloha č. C.2. Konkrétne číselné hodnoty sa nachádzajú v príslušných súboroch k Experimentu 2 na priloženom pamäťovom médiu v samostatnom hárku s názvom *„29 hodnôt - detailnejši pohľad“*.

4.4 Referenčný model vyhodnocovania

Experiment 2, viď 4.3, vyhodnocuje dopad nadmernej frekvencie generovaných prerušení na výpočtový výkon platformy FITkit 3.0. Vďaka zmene koncepcie experimentu sa podarilo odstrániť vznik odchýlok pri meraní požadovaných veličín z predošlej implementácie. Výsledky merania získané pomocou implementácie Experimentu 2 definujú limity výpočtového preťaženia pre túto platformu. Na základe nameraných a vyhodnotených výsledkov je tak možné stanoviť referenčnú hranicu výkonových možností mikrokontroléra.

Výsledky namerané v tomto experimente budú slúžiť ako referenčné hodnoty určené pri porovnaní s výsledkami implementovaných techník zamedzujúcich výpočtovému preťaženiu systému v dôsledku nadmernej frekvencie generovania prerušení. Zároveň bude implementácia tohto experimentu slúžiť ako referenčný testovací model pre testovanie výkonnosti jednotlivých implementovaných techník, zamedzujúcich výpočtovému preťaženiu mikrokontroléra, ktoré sú podrobne popísané v nasledujúcej kapitole č. 5.

Úlohou referenčného modelu je čo najlepšie pokryť a zdokumentovať stavový priestor výpočtového preťaženia na zvolenej platforme. Referenčný model postupne vzájomne skúma všetky možnosti konfigurácie vstupných parametrov. Zámerom je vykonať meranie pri každom nastavení a namerané výsledky zobraziť ako plochu, ktorá popisuje merané vlastnosti platformy. Pridaním techniky zabraňujúcej preťaženiu do referenčného modelu sa zároveň rozšíri množina vstupných parametrov. Výsledky implementovanej techniky je tak možné porovnať s referenčnými výsledkami. Bloková schéma referenčného modelu je zobrazená na obrázku č. 4.14.



Obr. 4.14: Bloková schéma referenčného modelu určeného na vyhodnocovanie vlastností systému vplyvom nadmernej frekvencie prerušení.

Kapitola 5

Techniky zamedzujúce preťaženiu

Kapitola podrobne opisuje implementáciu softvérových techník na platforme FITkit 3.0, ktorých snahou je zamedziť výpočtovému preťaženiu alebo prípadne zmierniť dopad výpočtového preťaženia z dôvodu nadmernej frekvencie prerušení.

Kapitola sa zaoberá implementovaním softvérových techník na zvolenej platforme a zaistením realizácie a správnej činnosti ich fungovania. Zároveň experimentálne vyhodnocuje schopnosť týchto techník potláčať vznik výpočtového preťaženia pri rôznej frekvencii príchodu prerušení a rôznej dĺžke vykonávania ich obsluhy. Dokumentuje výsledky použitých techník a posudzuje vznik súvisiacich neželaných efektov.

Z techník popísaných v kapitole č. 2.5 boli pre implementáciu vybraté dve varianty. Prvou je technika Striktného softvérového plánovača prerušení, ktorej princíp je vysvetlený v kapitole č. 2.5.1. Druhou zvolenou technikou je technika Zhlukového softvérového plánovača, ktorej princíp je popísaný v kapitole č. 2.5.2.

Tieto techniky boli spomedzi iných zvolené z dôvodu, že ide o techniky založené na softvérovej implementácii, ktoré k činnosti využívajú vnútorné komponenty mikrokontroléru ako sú časovače. Výhodou týchto softvérových plánovacích techník je, že sú súčasťou procesorovej logiky. Techniky nevyžadujú dedikované hardvérové komponenty pre plánovanie prerušení a zároveň ide o techniky, ktoré nevyužívajú pri plánovaní operačný systém. To umožnilo detailné experimentovanie s týmito technikami priamo na úrovni programu bez nutnosti využitia operačného systému. Pri implementácii boli využívané možnosti rozhrania CMSIS-Core. Nasledujúce kapitoly sú členené na návrh, implementáciu a zhodnotenie výsledkov z dôvodu lepšej čitateľnosti a jednoduchšej orientácie.

5.1 Technika striktného softvérového plánovača

V procesorovom jadre ARM Cortex-M4 platformy FITkit 3.0 bola implementovaná technika striktného softvérového plánovača, ktorá umožňovala limitovať množstvo procesorového času určeného pre obsluhu žiadostí o prerušenie a tým zabezpečila dostatočné množstvo procesorového času pre vykonávanie hlavného riadiaceho programu. Zároveň umožňovala zistiť ako vplýva stanovený limit na výkonnosť systému počas preťaženia.

5.1.1 Návrh realizácie na platforme FITkit 3.0 Minerva

Technika striktného softvérového plánovača je podrobne popísaná v kapitole č. 2.5.1. Ide o techniku plánovania prerušení, ktorá je vyhodnocovaná samotným procesorom. Konfiguráciou tejto techniky je možné špecifikovať minimálny čas medzi obsluhou dvoch po sebe

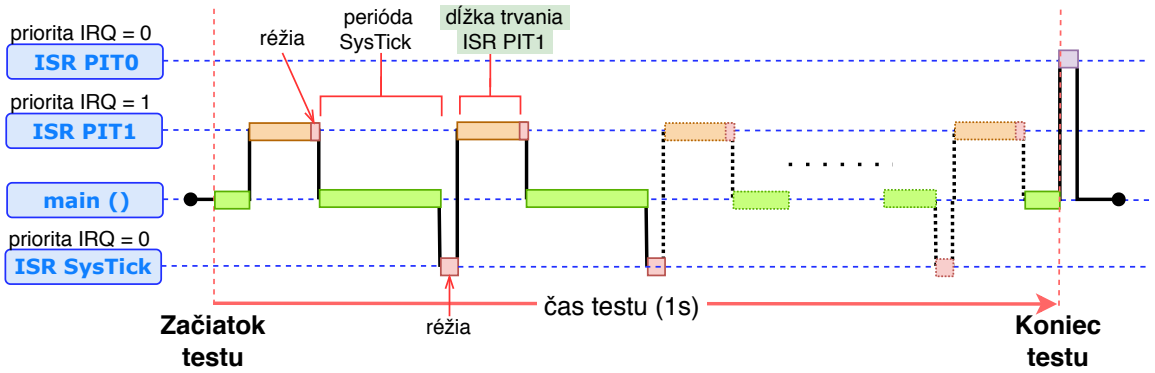
nasledujúcich prerušení. Tým sa zabezpečí dostatočné množstvo procesorového času pre vykonávanie hlavného programu.

Technika k činnosti vyžaduje jednorazový časovač určený na meranie minimálneho času medzi preruseniami. Počas tohto intervalu je v radiči prerušení zakázaná obsluha prerušenia a nemôže tak dôjsť k obsluhe ďalšieho prerušenia. Pokiaľ to zdroj prerušenia umožňuje, najvhodnejším riešením je úplne zakázať generovanie nových žiadostí o prerušenie priamo v zdroji prerušení. Na tento účel bol v implementácii zvolený systémový časovač SysTick.

- **SysTick časovač:** Meranie minimálneho času medzi obsluhou dvoch po sebe nasledujúcich prerušení v striktnom softvérovom plánovači prerušení je vhodné realizovať pomocou systémového časovača SysTick. SysTick zastáva v technike striktného plánovania funkciu upozorňujúceho jednorazového časovača. Počas jeho behu sa procesor venuje obsluhe hlavného programu a po vypršaní limitu je mu dovolené venovať sa obsluhe ďalšieho prerušenia. Po dokončení obsluhy musí znovu nastaviť jednorazový časovač a garantovať tým to, že určité percento výpočtového času bude venovať obsluhe hlavného programu. Detailnejší popis a vlastnosti komponenty sú zhrnuté v kapitole č. 3.4.6.

Vstupným parametrom jednorazového časovača je minimálny čas, resp. počet taktov počas ktorých nedochádza ku generovaniu nových prerušení. Tento údaj je možné previesť na prevrátenú hodnotu, ktorá špecifikuje maximálnu povolenú frekvenciu obsluhy žiadostí o prerušenie.

Úlohou je upraviť obsluhu prerušenia tak, aby obsahovala kód, ktorý zakáže obsluhu prerušení v radiči NVIC a spustí časovač s vopred špecifikovaným časovým limitom. Prerušenie bude generované časovačom PIT1 ako v predošlých experimentoch. Obsluha ďalšieho prerušenia bude povolená až po uplynutí časového limitu v obslužnej rutine vyvolanej systémovým časovačom. Tým sa zároveň vylúči možnosť konfliktného spustenia časovaču. Princíp činnosti je možné vidieť v diagrame na nasledujúcom obrázku č. 5.1.



Obr. 5.1: Princíp činnosti techniky striktného softvérového plánovača na platforme FITkit 3.0.

Pre zistenie a vyhodnotenie vlastností techniky striktného softvérového plánovača bude použitý referenčný model experimentálneho získavania výsledkov, ktorý bol implementovaný ako Experiment 2 v kapitole č. 4.3. Táto technika plánovania prerušení bude vyhodnocovaná podľa rovnakých kritérií a bude obsahovať množinu rovnakých vstupných parametrov, rozšírenú o možnosť konfigurácie periódy jednorazového SysTick časovača.

Prerušenia budú generované pomocou modulu časovača PIT a zaznamenávané komponentou DWT. Referenčný model zároveň umožní jednotné porovnávanie nameraných výsledkov s referenčnými limitmi stanovenými v predošlom experimente.

5.1.2 Implementácia striktného softvérového plánovača

Implementácia techniky striktného softvérového plánovača vychádza z kapitoly návrhu 5.1.1 a popisu činnosti fungovania, viď kapitola č. 2.5.1. Technika plánovača využíva pri vyhodnocovaní implementáciu z Experimentu 2 ako referenčnú, na základe jej vhodných vlastností a stability nameraných výsledkov. Celá implementácia je dostupná na priloženom médiu v adresári `Technika1_Strikt_SW`, viď príloha č. A.

Z dôvodu, že technika striktného softvérového plánovača je priamo integrovaná do upravenej implementácie z experimentu č. 2, viď kapitola č. 4.3, sú ďalej popisované iba časti zaoberajúce sa implementáciou súvisiacou s technikou striktného softvérového plánovača. Hlavným krokom je konfigurácia a obsluha jednorazového SysTick časovača.

Inicializácia a konfigurácia SysTick časovača

V implementácii jednorazového časovača je použitý systémový časovač SysTick. Časovač je inicializovaný pomocou funkcie `init_SysTick()`, ktorej vstupným parametrom je počet hodinových taktov. Počet taktov je vypočítaný z hodnoty minimálnej periódy medzi obsluhovanými prerušeniami, pomocou makra `compute_SysTick_period()`. Funkcia využíva na prepočet hodnoty z mikrosekúnd na počet taktov rovnaký vzorec ako na inicializáciu PIT časovača, viď kapitola č. 4.2.2.

Funkcia pred nastavením periódy najskôr v riadiacom registri CTRL vypne systémový časovač, čo sa doporučuje z hľadiska znovupoužiteľnosti kódu a z dôvodu, že časovač mohol byť v predchádzajúcom stave zapnutý. Následne sa do registra LOAD nastaví počet časovacích taktov, od ktorého je dôležité odčítať 12 taktov z dôvodu odstránenia latencie pri vyvolaní obsluhy prerušenia časovača.

Vo fáze inicializácie je potrebné pomocou `NVIC_SetPriority()` nakonfigurovať v radiči NVIC prioritu prerušenia vyvolaného časovačom na najvyššiu možnú úroveň hodnotou 0. Na záver je v registri VAL vynulovaná aktuálna hodnota časovača. Implementácia je realizovaná podľa doporučení v literatúre [23].

Spustenie SysTick časovača

Spustenie časovača je implementované pomocou makra `start_SysTick()`, ktoré vynuluje aktuálnu hodnotu časovača a následne aktivuje časovač nastavením bitov CLKSOURCE, TICKINIT a ENABLE. Nastavením bitov sú zvolené ako zdroj procesorové hodiny, je povolené vyvolanie prerušenia pri dosiahnutí nulovej hodnoty a povolené spustenie časovača.

Spustenie časovača prebieha vždy na záver po dokončení vykonávania obslužnej rutiny prerušenia časovača PIT1 generujúceho záťaž. Predtým je však potrebné v radiči NVIC zakázať obsluhu prerušení z PIT1 pomocou funkcie `NVIC_DisableIRQ(69)`. Tento postup je jednou z hlavných činností striktného softvérového plánovača. Povolenie obsluhy prerušení pre časovač PIT1 bude opäť spustené až v obsluhe prerušenia jednorazového systémového časovača SysTick.

Obsluha prerušenia vyvolaného jednorazovým SysTick časovačom

Po vypršaní limitu časovača dôjde k vyvolaniu obslužnej rutiny `SysTick_Handler()`. Obslužná rutina časovača v úvode vykoná vypnutie SysTick časovača pomocou riadiaceho registra CTRL. Tým sa docieli jednorázovosť spustenia časovača, nakoľko časovač deaktivuje sám seba.

Následne je pomocou `NVIC_EnableIRQ(69)` v radiči prerušení NVIC opäť povolená obsluha prerušení pochádzajúcich od modulu časovača PIT1. Tým je uzavretý kolobeh vzájomného zakázania a povolenia obsluhy prerušení v striktnom softvérovom plánovači. Vzhľadom na zvolený priebeh meraní nie je potrebné zakazovať a povoľovať všetky prerušenia, ale postačuje povoliť obsluhu prerušení iba pre záťažový časovač PIT1.

5.1.3 Priebeh experimentálneho merania vlastností

Pre vyhodnotenie implementovanej techniky striktného softvérového plánovača bolo zvolených 13 rôznych nastavení limitu časovača. Tieto hodnoty sú uložené v poli s názvom `SysTick_Reload_Val_interr_limit`. Hodnoty pre konfiguráciu minimálneho limitu časovača boli vhodne zvolené pre: 400 μs, 200 μs, 160 μs, 80 μs, 40 μs, 20 μs, 16 μs, 12 μs, 8 μs, 4 μs, 2 μs a 1 μs.

Pre každú z konfigurácií systémového časovača bol vykonaný celý beh Experimentu 2 popísaný v kapitole č. 4.3. Tento princíp umožňuje vzájomne porovnať namerané výsledky medzi jednotlivými rozdielnymi konfiguráciami časovača.

Pri získavaní výsledkov bola najskôr nakonfigurovaná jedna z hodnôt periódy SysTick časovača. Pre danú konfiguráciu bola vykonaná celá sada testovacích scenárov definovaných v Experimente 2. Po vykonaní celého experimentu bola nastavená v cykle s riadiacou premennou l (algoritmus č. 7, riadok 5) ďalšia konfigurácia periódy a znovu sa vykonal celý Experiment 2. Takto sa pokračovalo pre všetky zvolené konfigurácie. Priebeh zaznamenávania výsledkov je prehľadne zobrazený v algoritme č. 7.

Algoritmus 7: Algoritmus experimentálneho merania vlastností techniky striktného softvérového plánovača

```
1  inicializuj generátor hodín (MCG) do režimu BLPE
2  vypni watchdog
3  inicializuj indikačné diódy LED
4  inicializuj PIT časovač
5  for l ← 0 to l < COUNT_OF_SYSTICK_LIMITS do
6      nastav index aktuálneho limitu časovača
7      inicializuj a nakonfiguruj systémový časovač SysTick
8      [ vykonaj cyklické meranie výsledkov podľa algoritmu z Experimentu 2 ]
9  end for
10 inicializuj sériovú komunikáciu cez UART
11 for l ← 0 to l < COUNT_OF_SYSTICK_LIMITS do
12     odošli namerané výsledky z každej konfigurácie SysTick časovača cez UART
13 end for
```

Meranie prebiehalo v trinástich iteráciach pre každú konfiguráciu. Čas merania výsledkov jednej konfigurácie bol rovnaký ako v Experimente 2, 8 minút a 40 sekúnd. Nakoľko sa toto meranie vykonávalo 13 krát celkový čas merania stúpol na 1 hodinu, 52 minút a 40

sekúnd. Počas tejto doby sa vykonalo 6 760 testov s rôznou konfiguráciou vstupných parametrov. Počas priebehu experimentálneho merania vlastností implementovanej techniky bolo nameraných 20 280 číselných údajov, ktoré boli po dokončení merania odoslané sériovou linkou z platformy a vyžadovali následné spracovanie a vhodnú formu vizualizácie týchto výsledkov pre zjednodušenie porovnania.

Uloženie a odoslanie nameraných výsledkov

Všetky výsledky boli počas priebehu experimentálneho merania implementovanej techniky zaznamenávané v poli štruktúr `result`, ktorá je identická so štruktúrou Experimentu 2 a slúži pre zaznamenanie výsledkov jedného experimentu.

Nakoľko pri vyhodnocovaní sa vykonáva viacero iterácií tohto experimentu pre každú konfiguráciu časovača SysTick, štruktúry obsahujúce výsledky jedného experimentu je potrebné spojiť do poľa štruktúr s názvom `measurement_1`[`COUNT_OF_SYSTICK_LIMITS`], kde každá položka poľa obsahuje výsledky jedného experimentu.

Hodnoty, ktoré boli namerané počas experimentálneho vyhodnotenia techniky, sú po dokončení odoslané sériovou komunikáciou do pripojeného počítača. Pre odosielanie sa využívajú funkcie implementované v Experimente 2. Odosielanie hodnôt je však potrebné vykonávať cyklicky pre každú z trinástich konfigurácií jednorazového SysTick časovača.

5.1.4 Vizualizácia a vyhodnotenie nameraných výsledkov

Získané výsledky striktného softvérového plánovača z platformy FITkit 3.0 sú uložené v adresári `Technika1_Strikt_SW` v podzložke `Results`. Zložka obsahuje 13 súborov s názvom `Systick limit` pre jednotlivé konfigurácie jednorazového časovača. Každý súbor je označený číslom vyjadrujúcim počet taktov, ktoré boli použité pre inicializáciu časovača. Každý zo súborov obsahuje trojicu tabuliek v CSV formáte pre počet iterácií hlavnej smyčky, počet taktov strávených v tele ISR PIT1 a počet vykonaných ISR PIT1 počas jednej sekundy. Výstup uložený v jednom súbore je tak analógiou k trojici súborov z predošlého experimentu.

Zo všetkých nameraných údajov boli v tabuľkovom procesore vytvorené 3D grafy. Pre každý z grafov uvedených v predošlom Experimente 2, bola vytvorená sada 13 grafov. Jeden pre každú konfiguráciu systémového časovača od 400 μs do 1 μs. Kompletná sada grafov je dostupná na pamäťovom médiu v adresári `Výsledkové_grafy` v podadresári `Technika1`.

Pre popis a detailnejšie porovnanie v nasledujúcich kapitolách boli spomedzi všetkých trinástich vybrané iba tri možnosti, zo začiatku, stredu a konca konfiguračného intervalu. Konkrétne s hodnotou limitu 200 μs, 20 μs a grafy s 2 μs hodnotou expiračnej doby SysTick časovača. Na tejto trojici hodnôt budú ďalej demonštrované dosiahnuté výsledky.

Počet iterácií v tele hlavného programu

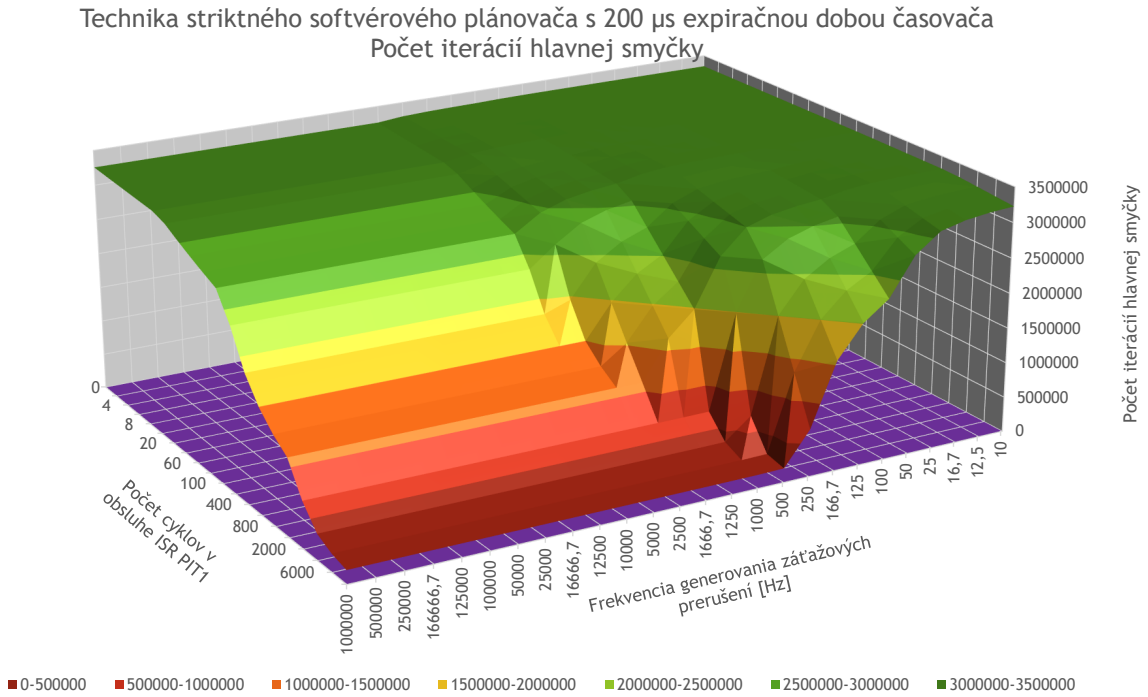
Nasledujúca trojica grafov, viď obrázky č. 5.2 až č. 5.4, zobrazuje výsledky meraní počtu iterácií, ktoré procesorové jadro venovalo vykonávaniu hlavnej smyčky programu. Zelenou sú zobrazené oblasti, kedy je pre obsluhu hlavného programu k dispozícii dostatok výpočtového času. Hodnoty prechádzajú až do červenej oblasti, ktorá zobrazuje naopak situáciu, kedy je procesorové jadro maximálne vyťažené obsluhou prerušení. Z hľadiska hlavného programu dochádza k efektu *„vyhladovania“*.

V porovnaní s referenčnými hodnotami Experimentu 2, viď obrázok č. 4.11, je v týchto grafoch možné vidieť, že s využitím techniky striktného softvérového plánovača dochádza

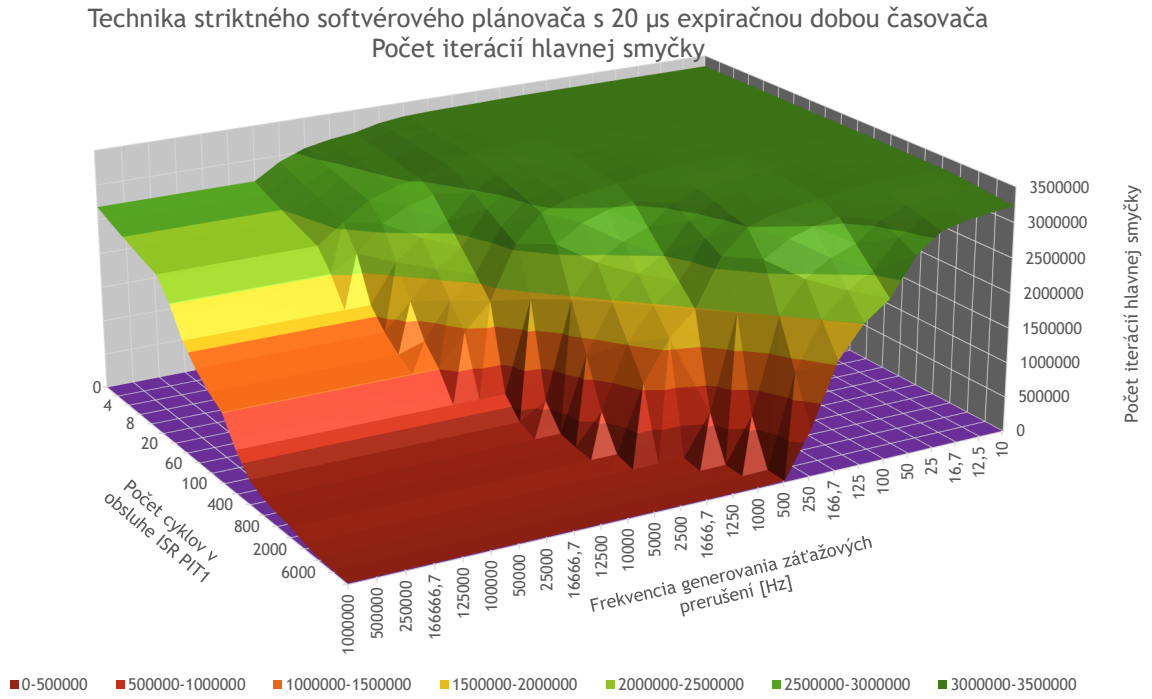
od určitej hranice k pevnému obmedzeniu, od ktorého počet iterácií ďalej neklesá. Obmedzenie sa týka stavu, kedy by podľa referenčných výsledkov mala hodnota počtu iterácií postupne klesať na nulovú úroveň. Technika striktného softvérového plánovača v tomto prípade zabezpečí, aby množstvo procesorového času venované hlavnému programu nekleslo pod vopred nakonfigurovaný limit.

Najhorším prípadom je situácia s veľkým počtom cyklov v ISR PIT1 (6 000 a viac, viď červená oblasť grafu na obrázku č. 5.2), kedy je čas vykonávania obslužnej rutiny príliš dlhý. Veľká doba obsluhy ISR zapríčiní, že počas jednej sekundy (čas testu) sa vykoná len malý počet prerušení. Malý počet prerušení zodpovedá malému počtu nastavení jednorazového SysTick časovača, pretože v technike striktného softvérového plánovača sa jednorazový časovač nastavuje v tele ISR. Z dôvodu malého počtu nastavení SysTick časovača nie je možné výrazne zvýšiť čas venovaný obsluhe hlavného programu. To je zároveň dôvodom, že technika striktného softvérového plánovača sa viditeľnejšie prejavuje v situáciách s krátkou dobou obsluhy ISR PIT1, kde dochádza k častejšiemu nastavovaniu SysTick časovača.

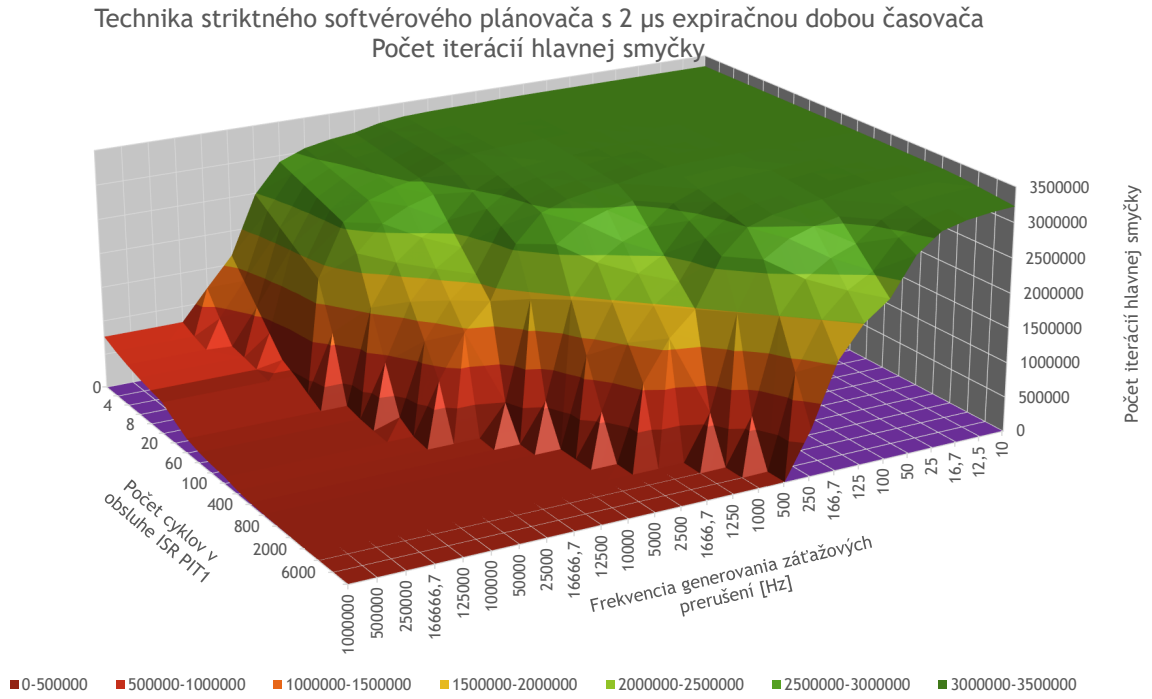
V tejto sérii grafov je možné vidieť, ako sa pri každom desaťnásobnom zmenšení inicializačnej doby jednorazového SysTick časovača, zmenší aj limitujúca hranica počtu vykonaných iterácií v tele hlavného programu. Z trojice grafov je najvýraznejšie obmedzenie zaznamenané pri nastavení veľkého intervalu časovača 200 μs. V grafe na obrázku č. 5.2 je možné vidieť, že pre najkratšiu dobu obsluhy ISR PIT1 počet iterácií v hlavnom programe takmer neklesol. Pri nastavení kratšieho intervalu 20 μs, viď graf na obrázku č. 5.3, je možné vidieť, že limitná hranica počtu vykonaných iterácií v hlavnom programe výrazne klesla. Technika striktného softvérového plánovača sa prejavila najmenej pri 2 μs intervale SysTick časovača.



Obr. 5.2: Počet iterácií hlavného programu pri technike striktného softvérového plánovača s 200 μs expiračnou dobou SysTick časovača.



Obr. 5.3: Počet iterácií hlavného programu pri technike striktného softvérového plánovača s 20 μs expiračnou dobou SysTick časovača.



Obr. 5.4: Počet iterácií hlavného programu pri technike striktného softvérového plánovača s 2 μs expiračnou dobou SysTick časovača.

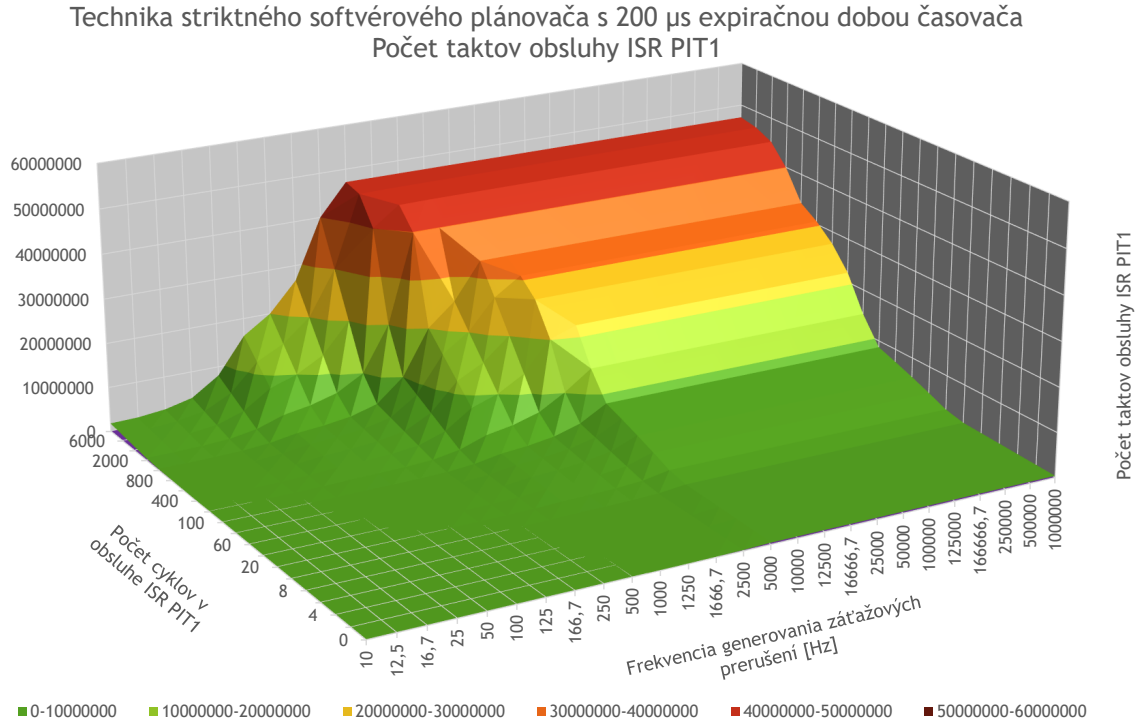
Počet taktov v tele ISR záťažového časovača PIT1

Trojica grafov na obrázkoch č. 5.5 až č. 5.7, zobrazuje počet taktov venovaných obsluhe ISR záťažového časovača PIT1. Zelenou je zobrazená prijateľná úroveň pre počet taktov venovaný obsluhe prerušení. Tmavočervená oblasť zobrazuje stav, kedy dochádza k preťaženou vplyvom prerušení.

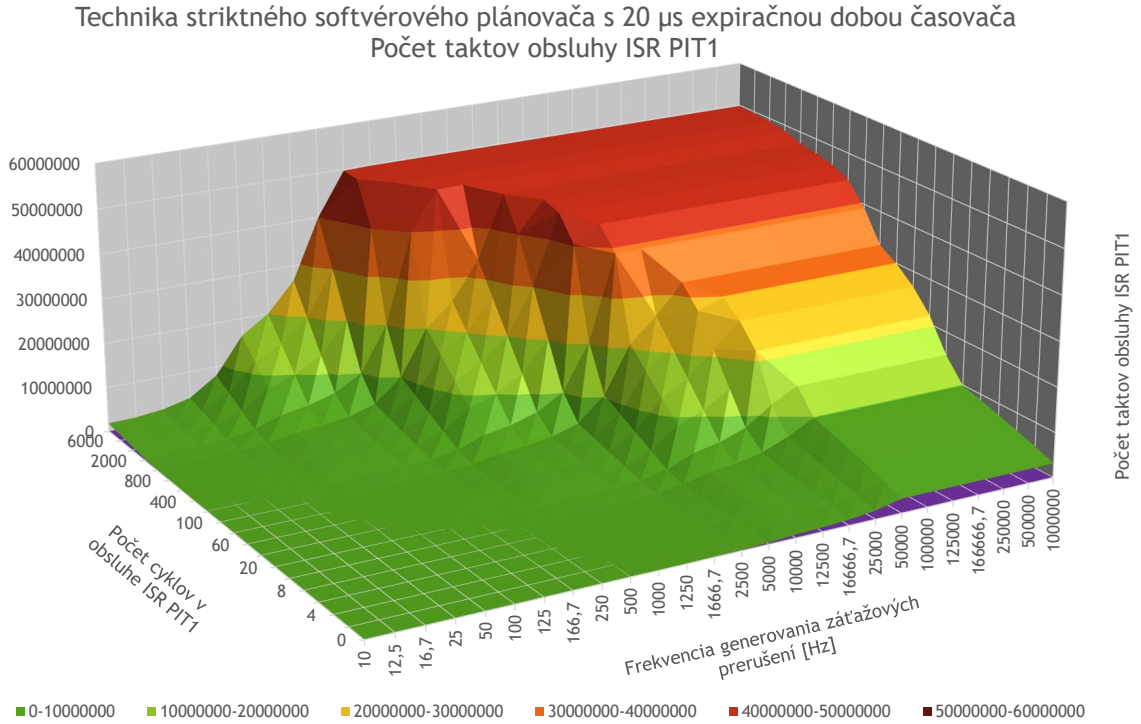
Technika striktného softvérového plánovača umožňuje pevne limitovať počet taktov venovaných obsluhe prerušení. V porovnaní s referenčnými hodnotami z grafu na obrázku č. 4.12, je možné vidieť, že nárast počtu taktov je od určitej konfigurácie zastavený a v prípade ak aj dochádza k zvyšovaniu frekvencie generovania záťažových prerušení z PIT1, počet taktov venovaný ich obsluhe už ďalej nenarastá.

Čím je interval v SysTick časovači striktného plánovača väčší, tým skôr dôjde k zastaveniu narastania počtu taktov strávených v obsluhe prerušení. V grafe na obrázku č. 5.5, kde je hodnota SysTick časovača 200 μs, je možné vidieť, že k limitácii dôjde najneskôr pri 5 kHz. Pri zmenšovaní intervalu časovača sa limitná hranica postupne vzďaľuje, viď graf na obrázku č. 5.6. Až v prípade 2 μs intervalu, viď graf na obrázku č. 5.7, kde dôjde k limitácií najneskôr pri frekvencii 250 kHz. V tomto prípade je však možné vidieť, že počet taktov v ISR, s viac ako 6 000 iteráciami, je takmer na maximálnej hodnote počtu taktov z referenčných výsledkov, viď graf na obrázku č. 4.12.

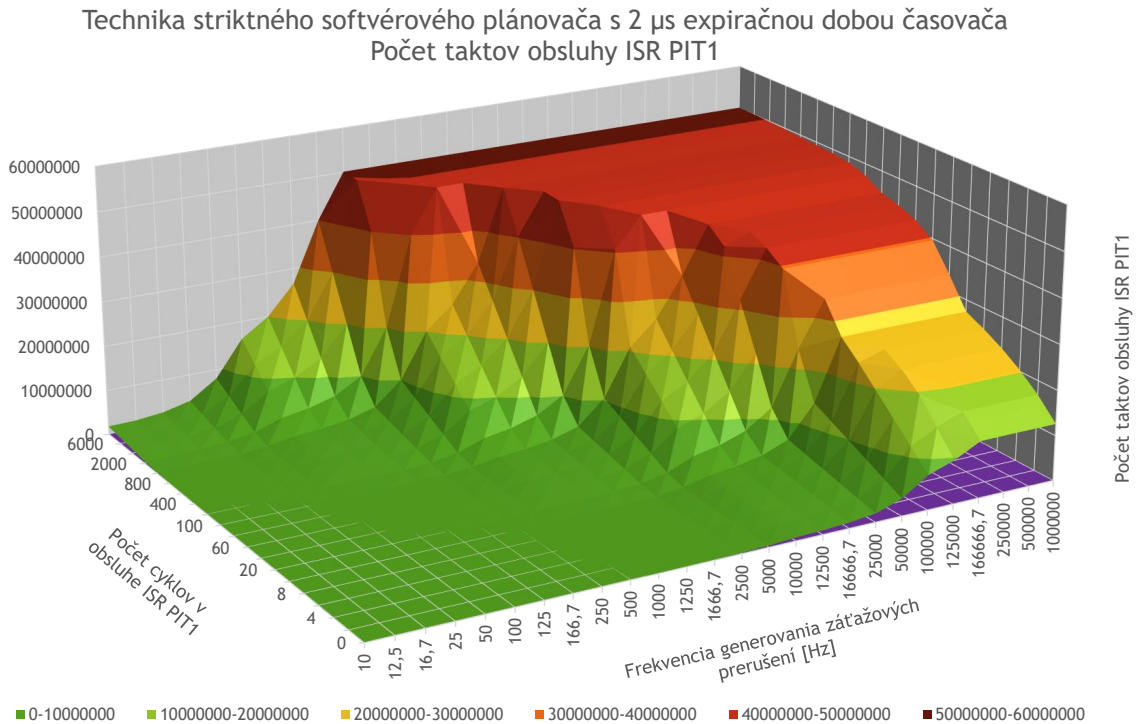
Tento fakt potvrdzuje výsledok zistený z predošlej sady grafov, že prínos techniky striktného softvérového plánovača je najviditeľnejší pri krátkotrvajúcej obslužnej rutine prerušenia.



Obr. 5.5: Počet taktov obsluhy prerušení PIT1 pri technike striktného softvérového plánovača s 200 μs expiračnou dobou SysTick časovača.



Obr. 5.6: Počet taktov obsluhy prerušení PIT1 pri technike striktného softvérového plánovača s 20 μs expiračnou dobou SysTick časovača.



Obr. 5.7: Počet taktov obsluhy prerušení PIT1 pri technike striktného softvérového plánovača s 2 μs expiračnou dobou SysTick časovača.

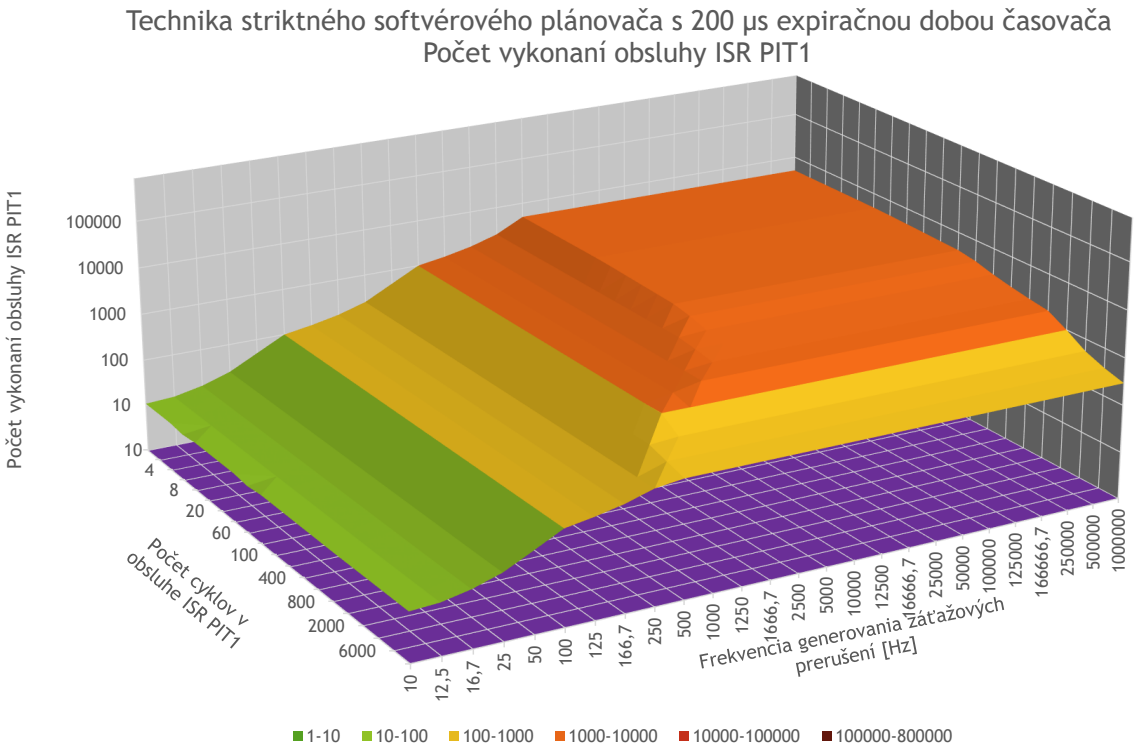
Počet obslúžených prerušení záťažového časovača PIT1

Posledná trojica grafov, viď obrázky č. 5.8 až č. 5.10, zobrazuje počet vykonaných obslužných rutín prerušenia záťažového časovača PIT1. Je potrebné poznamenať, že údaje v týchto grafoch nie sú odvodené z údajov predošlej série grafov, nakoľko sa dĺžka trvania záťažových prerušení PIT1 mení podľa parametrov experimentu. Na rozdiel od referenčných hodnôt z Experimentu 2, viď obrázok č. 4.13, boli v tomto prípade hodnoty zobrazené v logaritmickom merítku. To umožňuje prehľadnejšie zobraziť rozdiely medzi jednotlivými nastaveniami techniky striktného softvérového plánovača.

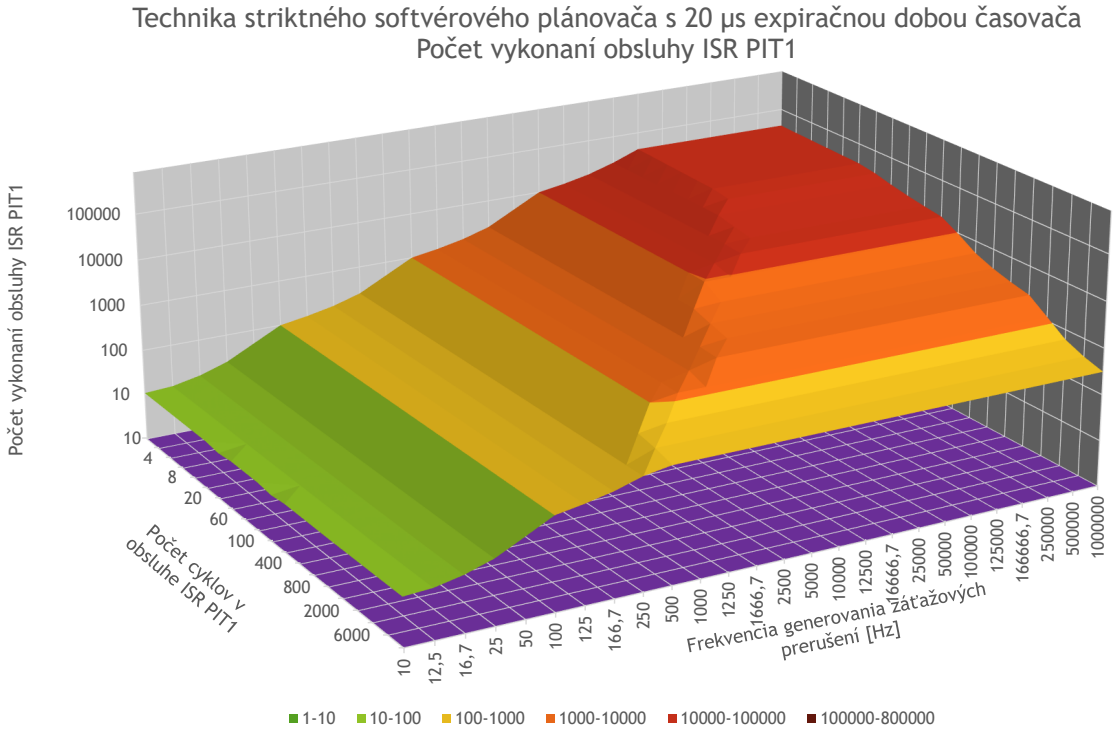
V grafoch je možné vidieť ako technika striktného softvérového plánovača ovplyvňuje počet vykonaných prerušení. Počet vykonaných prerušení stúpa priamo úmerne s hodnotou frekvencie a nepriamo úmerne s dĺžkou obsluhy jedného prerušenia.

Rovná plocha v oranžovej oblasti, viď graf na obrázku č. 5.8, zobrazuje stav, kedy technika striktného softvérového plánovača nedovolí vykonať viac prerušení ako je limit, nastavený pomocou jednorazového SysTick časovača. Znižovaním tohto limitu, viď graf na obrázku č. 5.9 a graf na obrázku č. 5.10, sa počet vykonaných prerušení postupne zvyšuje.

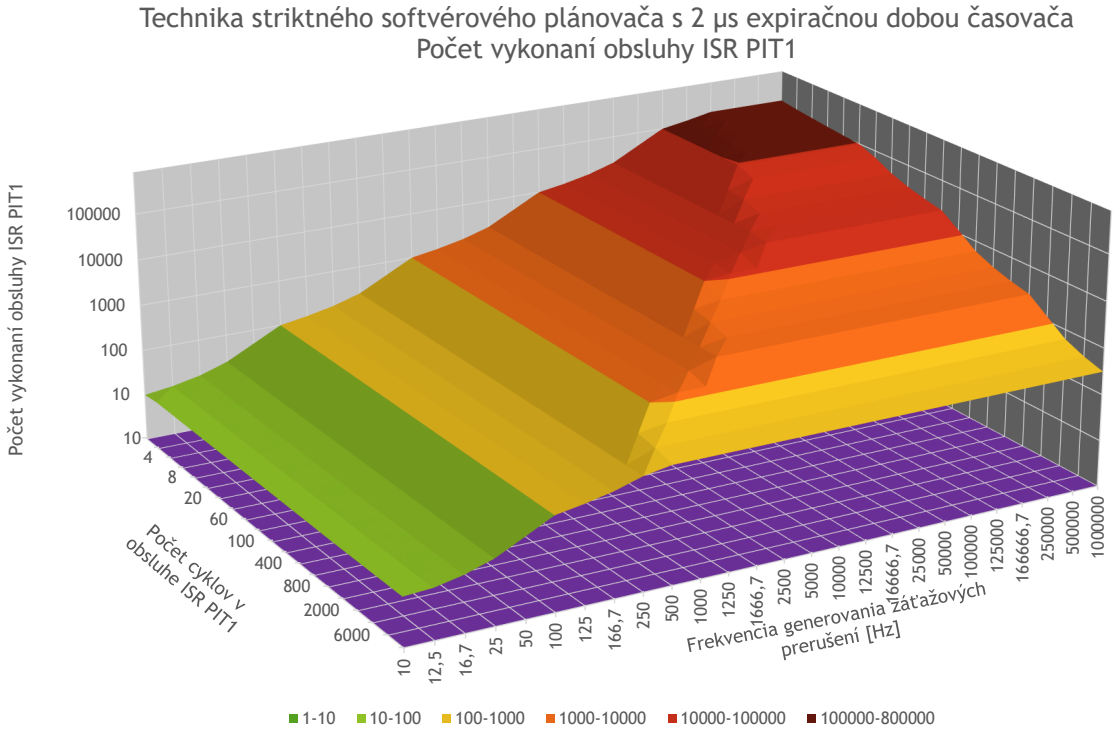
Tento fakt zároveň negatívne vplýva na počet vykonaných iterácií v tele hlavného programu. Z tohto dôvodu je potrebné zdôrazniť, že zvyšné prerušenia, ktoré sú nad hranicou nastaveného limitu, sú počas aktivity SysTick časovača ignorované a nedochádza tak k ich obsluhe. Toto správanie je možné s ohľadom na hlavný tok programu v určitých aplikáciách tolerovať z dôvodu garancie prideleného výpočtového času.



Obr. 5.8: Počet obslúžených prerušení PIT1 pri technike striktného softvérového plánovača s 200 μs expiračnou dobou SysTick časovača.



Obr. 5.9: Počet obslúžených prerušení PIT1 pri technike striktného softvérového plánovača s 20 μs expiračnou dobou SysTick časovača.



Obr. 5.10: Počet obslúžených prerušení PIT1 pri technike striktného softvérového plánovača s 2 μs expiračnou dobou SysTick časovača.

5.1.5 Réžia techniky striktného softvérového plánovača

Úlohou bolo zmerať réžiu spojenú s technikou striktného softvérového plánovača. Meranie réžie spojenej s technikou striktného plánovača prebiehalo pomocou miernej modifikácie pôvodnej implementácie.

Modifikácia bola potrebná z dôvodu, že komponenta DWT v každom prípade zaznamenávala rozdielne údaje a nemohla byť použitá pre súčasné meranie dvoch rozdielnych úsekov programu. Zároveň bol pre meranie réžie vytvorený samostatný nezávislý Kinetis SDK projekt, a to najmä z dôvodu, aby nedochádzalo k vzájomnému ovplyvňovaniu výsledkov medzi meraním výkonnosti techniky a meraním časovej réžie. Implementácia je dostupná na priloženom médiu v adresári `Technika1_Strikt_SW_overhead_measure`, viď príloha č. A.

V upravenej verzii bola na meranie časovej réžie použitá komponenta DWT, a makrá `start_DWT_cyctena()` a `stop_DWT_cyctena()` pre spustenie a zastavenie jej činnosti. Objektom merania pomocou komponenty DWT bol:

- Čas potrebný pre zakázanie obsluhy prerušení a spustenie SysTick časovača v tele ISR PIT1.
- Čas potrebný k obsluhe prerušenia vyvolaného SysTick časovačom, ktoré znovu povolí obsluhu prerušení z PIT1.

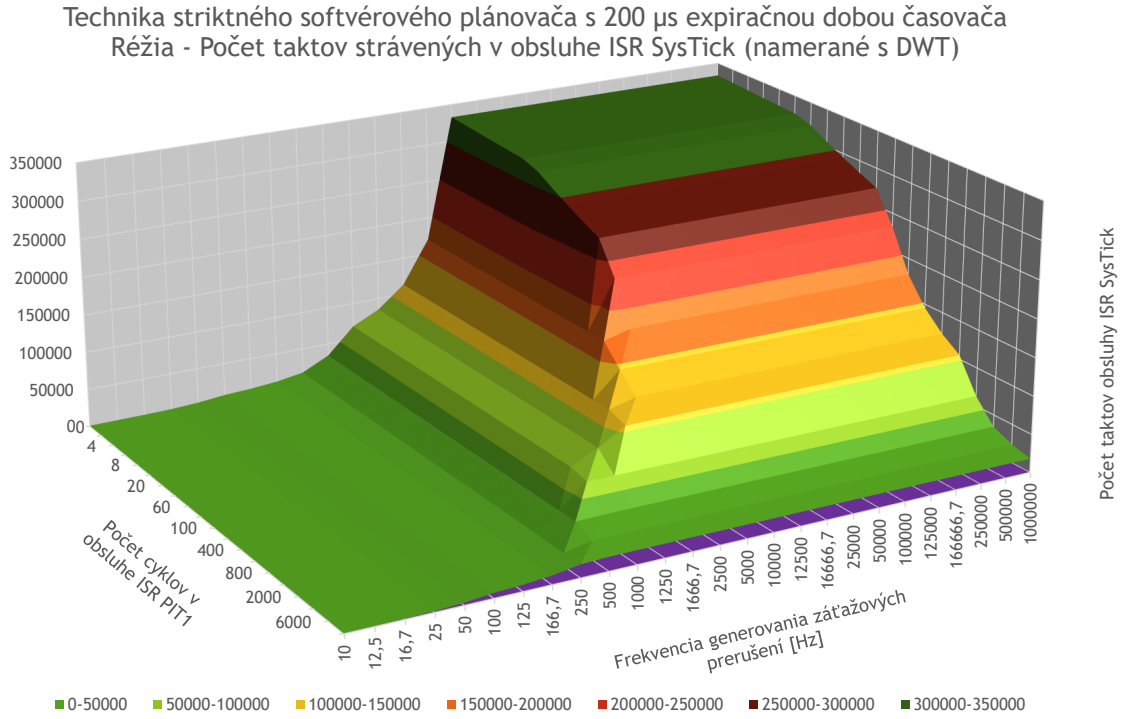
Pomocou komponenty DWT bolo možné zmerať, že pre zakázanie obsluhy prerušení a spustenie SysTick časovača v tele ISR PIT1 je potrebných 57 taktov. Tieto takty sú nevyhnutnou réžiou každého vyvolaného prerušenia záťažového časovača PIT1, čo je zanedbateľný počet v porovnaní s dĺžkou vykonávania celej obslužnej rutiny.

Druhou časťou bolo zmerať réžiu spojenú s obsluhou ISR SysTick časovača. Meranie prebiehalo nezávisle od predošlého. Pre meranie taktov ISR SysTick bola použitá komponenta DWT. V nasledujúcej trojici grafov na obrázkoch č. 5.11 až č. 5.13 je možné vidieť počet taktov, ktoré bol procesor nútený venovať obsluhe ISR SysTick počas jednej sekundy. Tento počet taktov pridáva technika striktného softvérového plánovača nad rámec iných činností procesorového jadra.

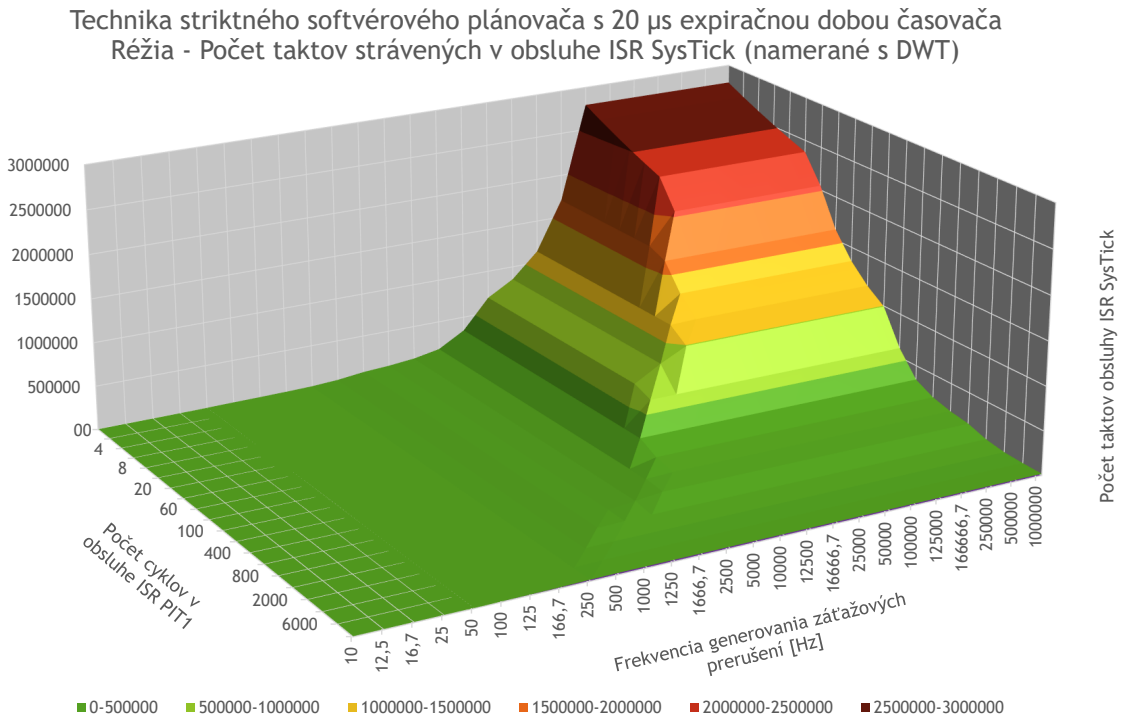
Najpodstatnejšou skutočnosťou vyplývajúcou z grafov je, že réžia ISR SysTick po dosiahnutí určitého maximálneho počtu taktov, so zvyšovaním frekvencie generovaných PIT1 záťažových prerušení ďalej nestúpa a udržuje si konštantnú úroveň. Je to vďaka limitu SysTick časovača, ktorý zakáže obsluhu nových záťažových prerušení PIT1. V grafoch je možné vidieť, že počet taktov réžie SysTick časovača dosahuje maximum pri záťažových prerušeniach PIT1 s krátkou dobou obsluhy (0 iterácií v ISR PIT1).

Počet vykonaných režijných prerušení SysTick časovača je možné vidieť v prílohe č. D. Počet taktov réžie potrebných pre vykonanie jednej ISR SysTick časovača je 68. Nakoľko počet inštrukcií vykonávaných v tele ISR SysTick je vždy konštantný, túto hodnotu je možné overiť podelením hodnôt grafu s počtom taktov strávených obsluhou ISR SysTick (napr. obr. č. 5.11) hodnotami z grafu s počtom vykonaných ISR SysTick (napr. obr. č. D.1).

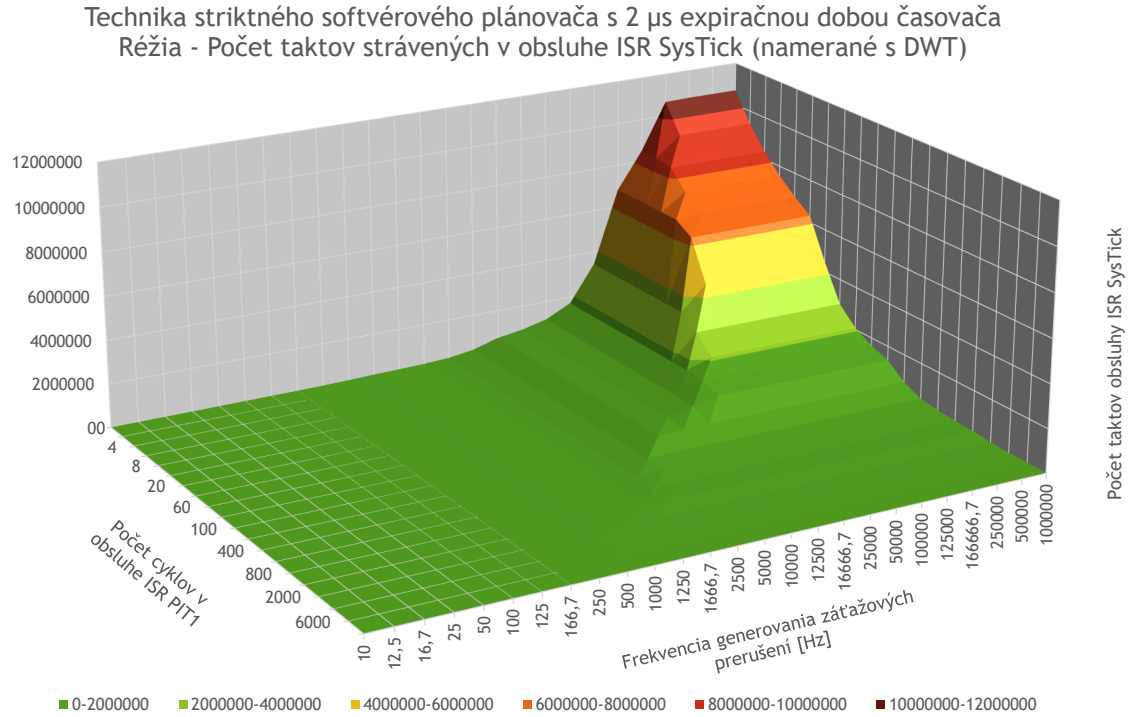
Grafy na obrázkoch č. 5.11 až č. 5.13 dokazujú, že hlavnou nevýhodou techniky striktného softvérového plánovača je vysoká réžia spojená s obsluhou jednorazového SysTick časovača. Počet prerušení rastie lineárne s počtom vykonaných záťažových prerušení. Na vykonanie jednej ISR PIT1 pripadá jedna obsluha ISR SysTick, ktorá si vyžaduje 68 taktov v tele ISR, plus 18 taktov „stacking“ a „unstacking“. Hlavným nedostatkom je, že táto réžia sa výrazne prejavuje najmä pri vysokej frekvencii záťažových prerušení. Meranie réžie trvalo rovnako ako meranie vlastností techniky 1 hodinu, 52 minút a 40 sekúnd.



Obr. 5.11: Počet taktov réžie v ISR SysTick počas 1 s pri technike striktného softvérového plánovača s 200 μs expiračnou dobou SysTick časovača.



Obr. 5.12: Počet taktov réžie v ISR SysTick počas 1 s pri technike striktného softvérového plánovača s 20 μs expiračnou dobou SysTick časovača.



Obr. 5.13: Počet taktov réžie v ISR SysTick počas 1 s pri technike striktného softvérového plánovača s 2 μs expiračnou dobou SysTick časovača.

5.2 Technika zhlukového softvérového plánovača

Na zvolenej platforme bola implementovaná optimalizovaná verzia softvérovej techniky plánovača, ktorá k plánovaniu obsluhy prerušení pristupuje v zhlukoch. Technika umožňovala zdokumentovať vplyv nastavenia parametrov plánovača na schopnosť zamedziť výpočtovému preťaženiu a popísať súvisiace efekty, ktoré vznikajú pri technike zhlukového softvérového plánovača. Zároveň sa pokúša znížiť režijné náklady spojené s plánovaním obsluhy prerušení.

5.2.1 Návrh realizácie na platforme FITkit 3.0 Minerva

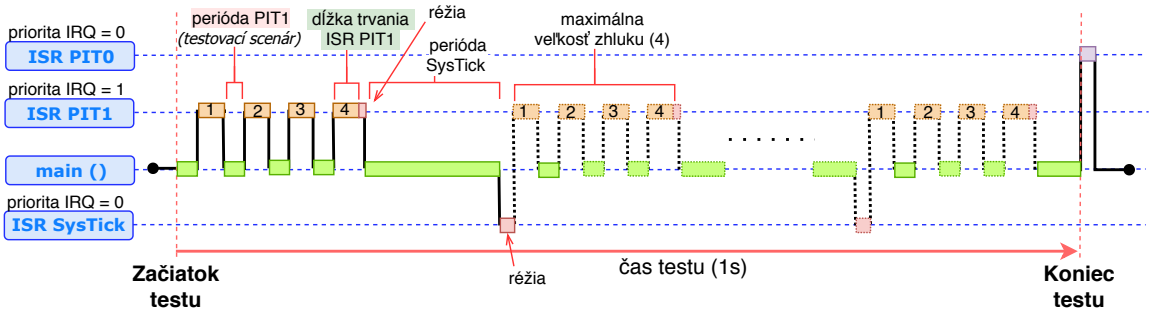
Detaily techniky zhlukového softvérového plánovača sú podrobne vysvetlené v kapitole č. 2.5.2. Nakoľko ide o softvérovú techniku, jej činnosť a vyhodnocovanie je v réžii procesorového jadra. Technika musí umožňovať konfiguráciu dvomi vstupnými parametrami.

- Prvým parametrom je maximálna veľkosť jedného zhluku prerušení, ktorá sa obsluhuje v bezprostrednej následnosti príchodu prerušení.
- Druhým parametrom je maximálna frekvencia príchodu zhlukov jednotlivých prerušení alebo jeho inverziu, minimálny čas (počet taktov) medzi príchodom zhlukov.

Pri realizácii tejto techniky je potrebné využiť hardvérový časovač SysTick, integrovaný priamo v procesorovom jadre, ktorý bude vykonávať jednorazové meranie limitu medzi príchodom jednotlivých zhlukov prerušení.

Počas činnosti časovača je zakázaná obsluha záťažových prerušení v NVIC a procesor sa môže venovať vykonávaniu hlavného programu. Po uplynutí limitu SysTick časovača je

opätovne povolená obsluha prerušení a inicializované počítadlo maximálnej veľkosti zhluku. Procesor môže v prípade potreby obslúžiť príslušný počet prerušení. Po prekročení maximálnej veľkosti zhluku dôjde k opätovnému zakázaniu obsluhy prerušení a spusteniu SysTick časovača. Princíp činnosti je zobrazený v diagrame na nasledujúcom obrázku č. 5.14.



Obr. 5.14: Princíp činnosti techniky zhlukového softvérového plánovača na platforme FITkit 3.0.

Snahou je implementovať optimalizovanú verziu, ktorá zbytočne periodicky nevykonáva vynulovanie počítadla veľkosti zhluku a nastavovanie časovača. Optimalizovaná verzia vykoná spustenie časovača až v prípade, ak sa prekročí maximálna veľkosť zhluku.

Vyhodnotenie vplyvu nastavenia parametrov techniky zhlukového softvérového plánovača bude vykonávané v referenčnom modeli z Experimentu 2, viď kapitola č. 4.3, upravenom pre potreby techniky plánovača. Vstupné parametre referenčného modelu boli rozšírené o parametre techniky zhlukového plánovača. Generovanie záťažových prerušení a zaznamenávanie výsledkov budú realizovať komponenty PIT1 a DWT a bude možné ich porovnať s referenčnými hodnotami Experimentu 2.

5.2.2 Implementácia zhlukového softvérového plánovača

Zhlukový softvérový plánovač bol implementovaný v referenčnom modeli s využitím spomenutých komponent, viď kapitola č. 5.2.1, čo umožňuje vzájomné porovnanie nameraných výsledkov s referenčnými výsledkami. Implementácia zhlukového softvérového plánovača je dostupná na priloženom pamäťovom médiu v adresári `Technika2_Zhluk_SW`, viď príloha č. A.

Konfigurácia SysTick časovača

Pri implementácii je využitý hardvérový časovač SysTick. Inicializáciu a konfiguráciu časovača zabezpečuje funkcia `init_SysTick()`, ktorá je zhodná s implementáciou v technike striktného softvérového plánovača, viď kapitola č. 5.1.2. Vstupný parameter funkcie však v tomto prípade určuje minimálny čas medzi príchodmi zhlukov prerušení.

Túto vlastnosť je potrebné zohľadniť pri vhodnom nastavení limitu časovača vo fáze vykonávania experimentov. V prípade ak by množina hodnôt definujúcich limit časovača zostala rovnaká ako pri experimentálnom meraní predošlej techniky, limit by bol príliš nízky vzhľadom k tomu, že v tomto prípade sa prerušenia nevykonávajú jednotlivo ale v zhlukoch.

Spustenie SysTick časovača

Spustenie a zastavenie časovača je v programe realizované pomocou implementovaných makier `start_SysTick()` a `stop_SysTick()` popísaných v kapitole č. 5.1.2. Časovač SysTick je spustený na konci obslužnej rutiny záťažového prerušenia PIT1, ktoré je obslúžené ako posledné z aktuálneho zhluku záťažových prerušení. Pre zistenie, či je aktuálne obsluhované prerušenie posledné v zhluku, je na konci obslužnej rutiny implementovaná podmienka, viď obrázok č. 5.15.

```
1. // časť vyžadovaná zhlukovým SW plánovačom
2. Burst_size++;
3. // hrozí preťaženie (ak je veľkosť zhluku >= maximálnej veľkosti zhluku)
4. if (Burst_size >= Max_burst_size[Current_Max_burst_index]){
5.     NVIC_DisableIRQ(69);
6.     start_SysTick();
7. }
```

Obr. 5.15: Overenie veľkosti zhluku v tele obslužnej rutiny záťažového prerušenia.

Obsluha prerušenia vyvolaného SysTick časovačom

V technike zhlukového softvérového plánovača je potrebné v obslužnej rutine prerušenia jednorazového SysTick časovača vykonať nielen opätovné povolenie prerušení pomocou `NVIC_EnableIRQ(69)`, ale zároveň sa obsluha prerušenia využije k vynulovaniu aktuálnej veľkosti zhluku `Burst_size = 0`. To zaistí, že po dokončení obslužnej rutiny bude možné obslúžiť nový zhluk čakajúcich prerušení. Zvyšok obsluhy je rovnaký ako v predchádzajúcej technike.

5.2.3 Priebeh experimentálneho merania vlastností

Vyhodnotenie vlastností implementovanej techniky prebiehalo vo viacerých iteráciách. Pre konfiguráciu limitu časovača bolo zvolených 6 možností (`COUNT_OF_SYSTICK_LIMITS = 6`). Hodnoty sú uložené v poli `SysTick_Reload_Val_burst_limit`. Hodnoty limitu medzi obsluhou zhlukov prerušení boli zvolené pre: 400 μs, 160 μs, 20 μs, 12 μs, a 4 μs. Nastavenie novej hodnoty časovača pre ďalší experiment zabezpečuje cyklus s riadiacou premennou l (algoritmus č. 8, riadok 7).

Pri konfigurácii veľkosti zhluku sa vychádzalo z výsledkov získaných v predošlej technike, na základe ktorých je možné predpokladať, že veľkosť zhluku musí byť dostatočne veľká na to, aby došlo k amortizácii taktov réžie medzi jednotlivé prerušenia zo zhluku. Z tohto dôvodu bola veľkosť zhluku plánovacej techniky vhodne zvolená pre: 4, 8, 16 a 32 prerušení. Počet možných konfigurácií tak bol nastavený ako (`COUNT_OF_BURST_SIZE = 4`). Veľkosť zhluku pri vykonávaní experimentov je konfigurovaná v cykle s riadiacou premennou b (algoritmus č. 8, riadok 5).

Pri meraní vlastností techniky zhlukového softvérového plánovača sa najskôr nakonfigurovala maximálna veľkosť zhluku, pre ktorú sa nastavila každá z možností limitu SysTick časovača. Pre každú konfiguráciu SysTick časovača sa následne vykoná celá sada experimentov špecifikovaná v referenčnom modeli z Experimentu 2. Po dokončení meraní sa postupuje na nastavenie novej veľkosti zhluku, až kým sa nevykonajú všetky z dostupných kombinácií.

Pole štruktúr `result` pre zaznamenávanie nameraných výsledkov počas experimentu bolo rozšírené pridaním ďalšej dimenzie. To umožňuje ukladať všetky hodnoty počas experimentu v `measurement_1[COUNT_OF_BURST_SIZE][COUNT_OF_SYSTICK_LIMITS]`, a po jeho

Algoritmus 8: Algoritmus experimentálneho merania vlastností techniky zhlukového softvérového plánovača

```
1  inicializuj potrebné komponenty
2  for b ← 0 to b < COUNT_OF_BURST_SIZE do
3      nastav index aktuálnej veľkosti zhluku
4      for l ← 0 to l < COUNT_OF_SYSTICK_LIMITS do
5          nastav index aktuálneho limitu časovača
6          inicializuj a nakonfiguruj systémový časovač SysTick
7          [ vykonaj cyklické meranie výsledkov podľa algoritmu z Experimentu 2 ]
8      end for
9  end for
10 inicializuj sériovú komunikáciu cez UART
11 for b ← 0 to b < COUNT_OF_BURST_SIZE do
12     for l ← 0 to l < COUNT_OF_SYSTICK_LIMITS do
13         odošli namerané výsledky z každej konfigurácie SysTick časovača cez UART
14     end for
15 end for
```

dokončení ich v cykle odoslať pomocou sériovej komunikácie v CSV formáte do pripojeného počítača (algoritmus č. 8, riadok 14).

Pre konfiguráciu veľkosti zhluku boli zvolené štyri možnosti. Pre nastavenie limitu jednorazového SysTick časovača bolo zvolených 6 konfigurácií. Experimentálne meranie výsledkov počas jednej konfigurácie experimentu zaznamená 1 560 hodnôt. Pri meraní vlastností tejto techniky bolo vyskúšaných vzájomne všetkých 24 kombinácií pre veľkosť zhluku a limit časovača. Čas merania tak dosiahol 3 hodiny a 28 minút. Pri experimente celkovo bolo nameraných 37 440 údajov.

Nakoľko pri meraní vlastností nedokázalo KDS, v inicializačnej fáze debug režimu taký počet údajov spracovať, meranie bolo konfigurované ručne pre každú veľkosť zhluku. Namerané hodnoty boli odoslané z platformy sériovou komunikáciou.

5.2.4 Vizualizácia a vyhodnotenie nameraných výsledkov

Všetky výsledky sú uložené v adresári `Technika2_Zhluk_SW` v podadresári `Results`. Adresár obsahuje 4 podzložky pre každú veľkosť zhluku. V zložke sa nachádza 6 súborov pre limit SysTick časovača v CSV formáte. Súbory obsahujú rovnaké merané veličiny ako v predošlej technike. Z nameraných údajov bolo v tabuľkovom procesore vytvorených celkovo 120 3D grafov, ktoré umožňujú prehľadnejšiu vizualizáciu a vzájomné porovnanie. Všetky grafy sú spolu s číselnými hodnotami zároveň dostupné na pamäťovom médiu v adresári `Výsledkové_grafy` v podadresári `Technika2`. Grafy z každého merania sú rozdelené do zložiek podľa veľkosti zhluku.

Ďalej budú podrobne popisované rozdiely nameraných údajov medzi veľkosťou zhluku pre 4 a 32 prerušení. Tieto hodnoty boli zvolené preto, že ide o okraje zvoleného intervalu maxilmálnej veľkosti zhluku, čo zvýrazní rozdiel výsledkov. Namerané údaje pre veľkosť zhluku 8 a 16 sú analógiou k zvoleným veľkostiam a presné číselné údaje je možné nájsť na priloženom pamäťovom médiu. Farebná reprezentácia grafov je identická s predošlými experimentmi.

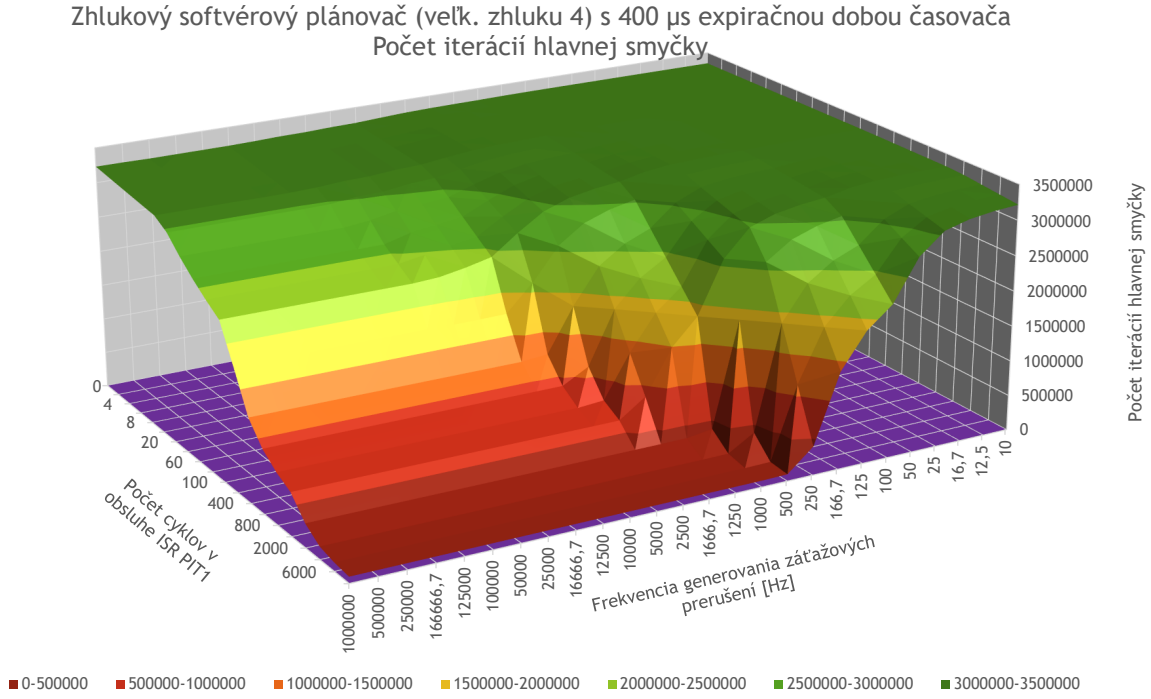
Počet iterácií v tele hlavného programu

Nasledujúce grafy zobrazujú počet iterácií v tele hlavného programu počas merania vlastností techniky zhlukového softvérového plánovača. Grafy na obrázkoch č. 5.16 a č. 5.17 zobrazujú výsledky pre veľkosť zhluku 4 prerušení a grafy na obrázkoch č. 5.18 a č. 5.19 pre veľkosť zhluku 32 prerušení. Dvojice boli vybraté z každej sady, nakoľko ide o merania s najväčšou 400 μs a najmenšou 4 μs dĺžkou minimálneho limitu príchodu zhlukov. Grafy medzi zvolenými hodnotami limitu sú priložené na pamäťovom médiu, z dôvodu rozsahu však nie sú v texte zobrazené.

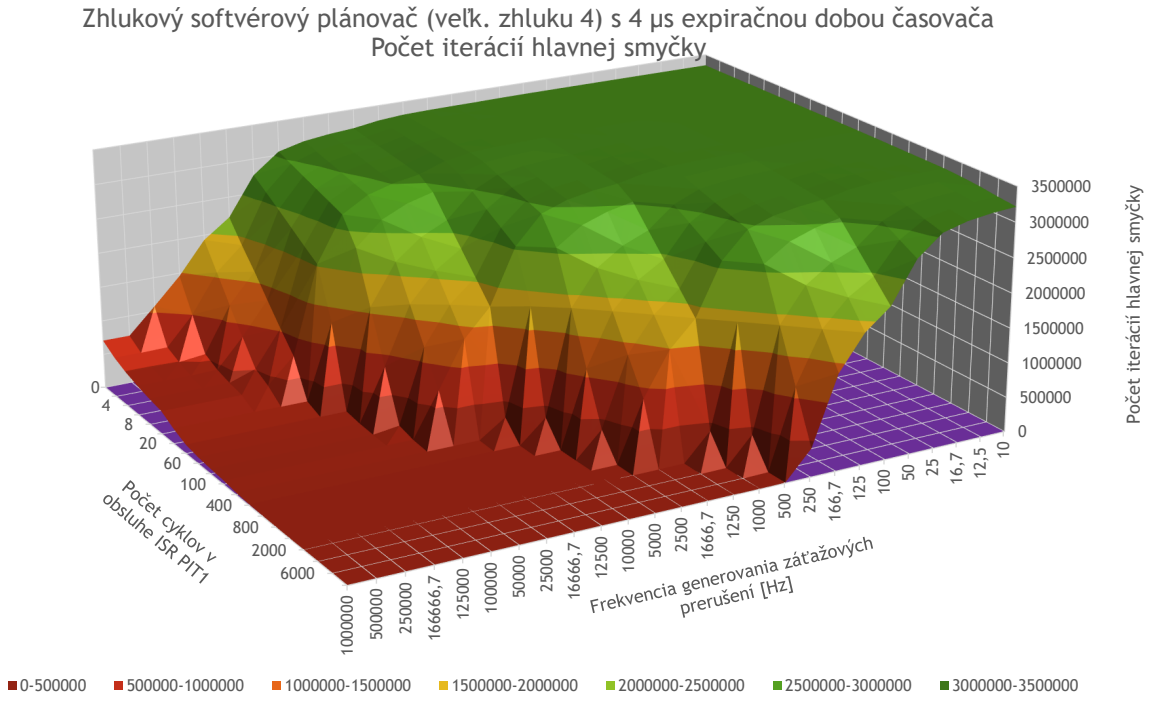
V týchto grafoch je možné vidieť, že technika zhlukového plánovača sa prejavuje podobne ako technika striktného plánovača, a so zvyšovaním frekvencie prerušení od určitého okamihu nedochádza k znižovaniu počtu cyklov vykonaných v tele hlavného programu, ale počet vykonaných cyklov je ďalej konštantný. Tým je v softvéri možné garantovať pridelenie stanoveného výpočtového času vykonávaniu hlavného programu. V tomto prípade je taktiež viditeľný negatívny efekt, kedy so zvyšujúcim časom obsluhy prerušenia tento limit klesá.

V každej dvojici grafov s rovnakou veľkosťou zhluku je možné vidieť, že znížením limitu jednorazového SysTick časovača dôjde k poklesu limitnej hranice pre počet iterácií v tele hlavného programu, čo je zapríčinené nárastom počtu prerušení vykonaných počas testu.

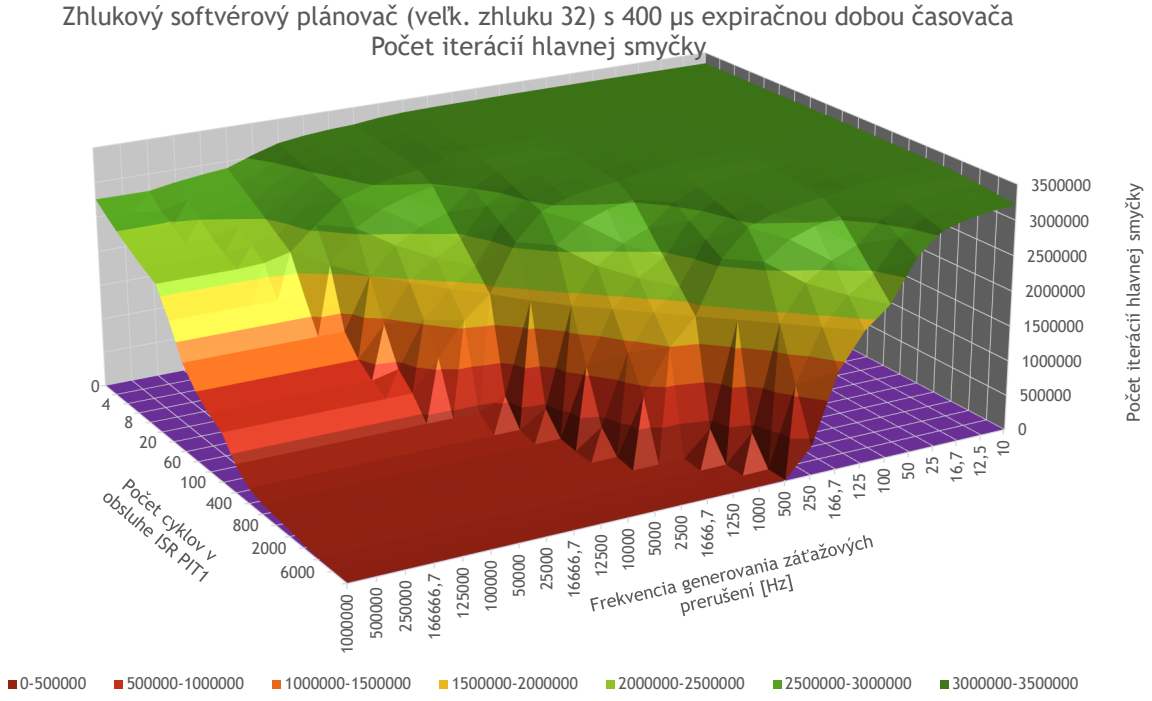
Údaje v grafoch s rovnakým limitom časovača ale rozdielnou veľkosťou zhluku dokazujú, že zvyšovaním maximálnej veľkosti zhluku dochádza taktiež k poklesu vykonaných iterácií v tele hlavného programu. Čím je veľkosť zhluku väčšia, tým viac to umožní vykonanie väčšieho množstva prerušení počas trvania jedného testu (1 s). To je dôvodom, prečo nie je vhodné stanoviť veľkosť zhluku prehnane veľkú. Plynulosť tohto poklesu pre veľkosť zhluku 8 a 16 prerušení je možné vidieť v grafoch na priloženom médiu.



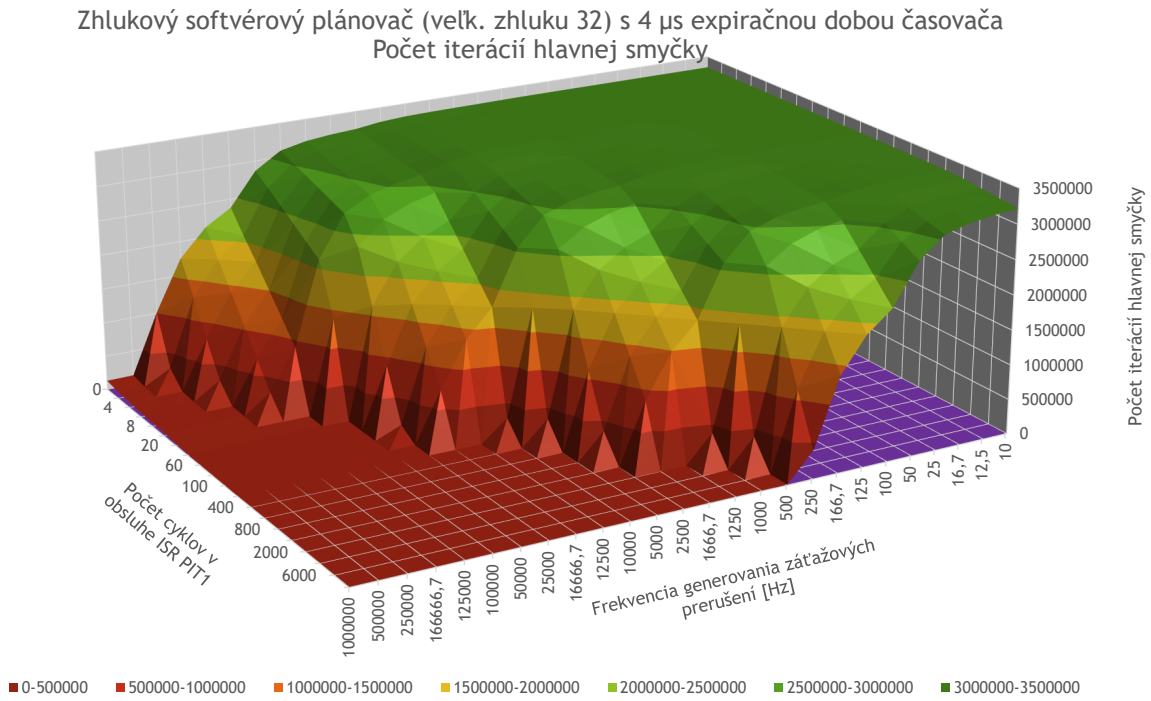
Obr. 5.16: Počet iterácií hlavného programu pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 400 μs expiračnou dobou SysTick časovača.



Obr. 5.17: Počet iterácií hlavného programu pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 4 μs expiračnou dobou SysTick časovača.



Obr. 5.18: Počet iterácií hlavného programu pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 400 μs expiračnou dobou SysTick časovača.



Obr. 5.19: Počet iterácií hlavného programu pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 4 μs expiračnou dobou SysTick časovača.

Počet taktov v tele ISR záťažového časovača PIT1

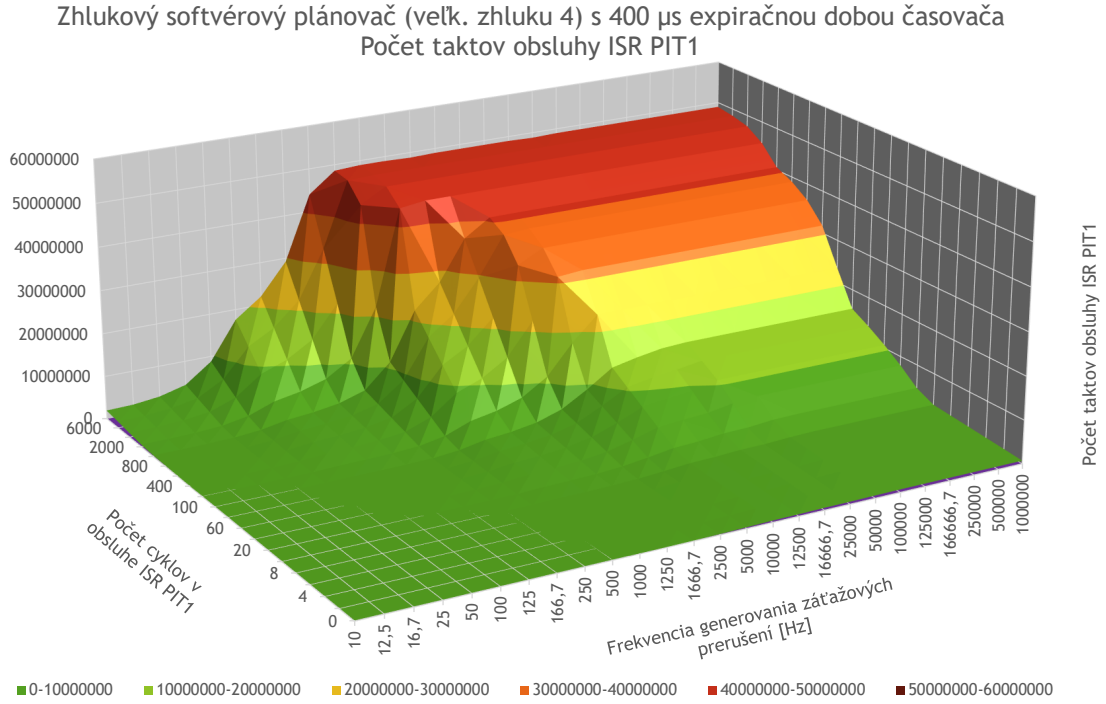
V nasledujúcich dvojiciach grafov je zobrazený počet vykonaných taktov počas priebehu jedného testu (1 s) pri použití techniky zhlukového softvérového plánovača.

Grafy na obrázkoch č. 5.20 a č. 5.21 zobrazujú hodnoty pre nakonfigurovanú maximálnu veľkosť zhluku 4. Ďalšia dvojica grafov na obrázkoch č. 5.22 a č. 5.23 zobrazuje hodnoty pre veľkosť zhluku 32 prerušení. Grafy zobrazujú hodnoty pre rovnaký limit príchodu zhlukov ako v predošlom prípade 400 μs a 4 μs.

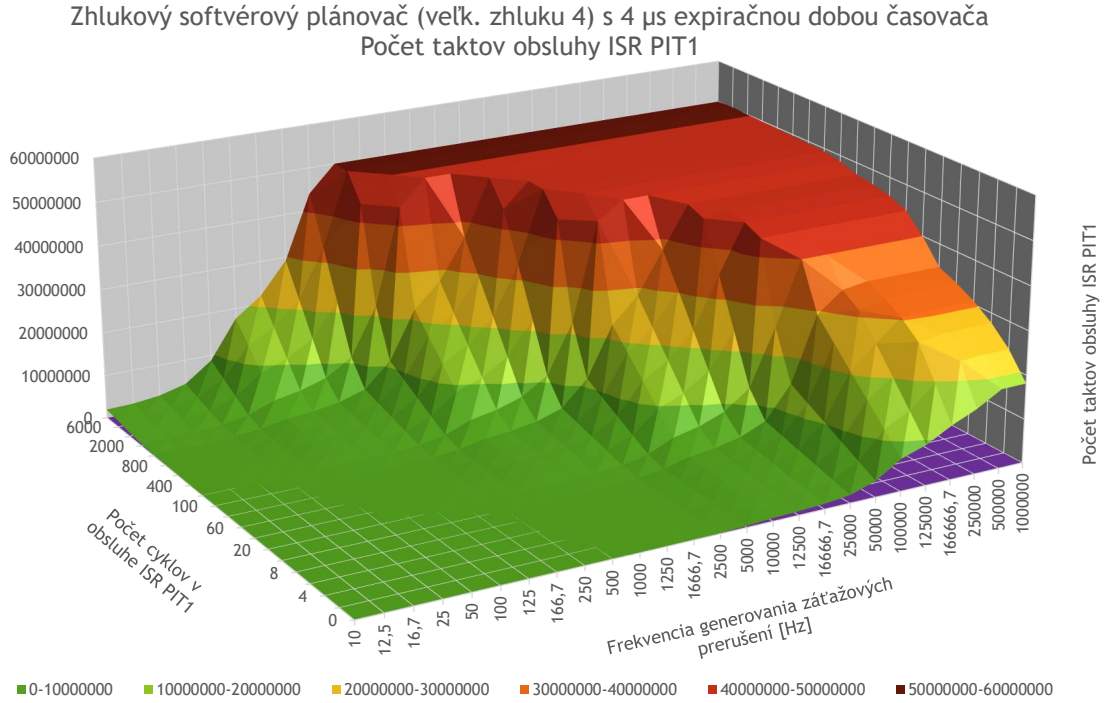
V nasledujúcich grafoch je možné vidieť ako pri zvyšovaní frekvencie príchodu záťažových prerušení dochádza v porovnaní s referenčnými hodnotami, viď. graf na obrázku č. 4.6, k limitovaniu počtu taktov, ktoré procesor venuje ich obsluhe.

Z dvojíc grafov v rámci rovnakej veľkosti zhluku je možné vidieť rovnaký efekt ako pri striktnom softvérovom plánovači. Znížením limitu maximálnej rýchlosti príchodu zhlukov zo 400 μs na 4 μs sa zvýši hranica, od ktorej je počet taktov v rámci ISR PIT1 počas testu konštantný. Zníženie maximálnej rýchlosti príchodu zhlukov najvýraznejšie ovplyvní rast počtu taktov prerušení s krátkou ISR. Najvýraznejší nárast v oboch prípadoch je pri 0 iteráciách ISR PIT1 a frekvencii 1 MHz, kedy pri veľkosti zhluku 4 stúpol počet taktov z 711 110 na 18 131 209, a pri veľkosti zhluku 32 stúpol počet taktov z 4 784 321 na 26 625 950.

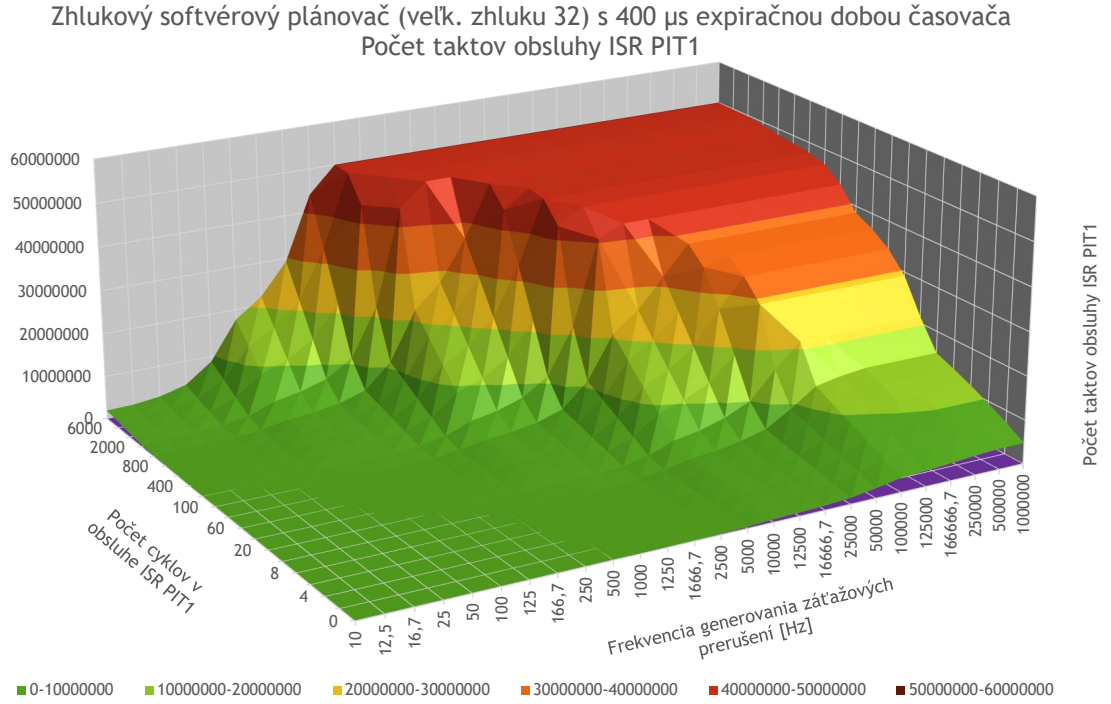
Pri vzájomnom porovnaní grafov s rovnakou rýchlosťou príchodu zhlukov ale rozdielnou veľkosťou zhluku je výsledkom zvýšenie počtu taktov venovaných obsluhe ISR PIT1. Nakoľko čas medzi zhlukmi zostane nezmenený, ale navýši sa maximálna veľkosť zhluku, dochádza k obsluhe väčšieho počtu prerušení a tým aj k zvýšeniu počtu taktov ich obsluhy.



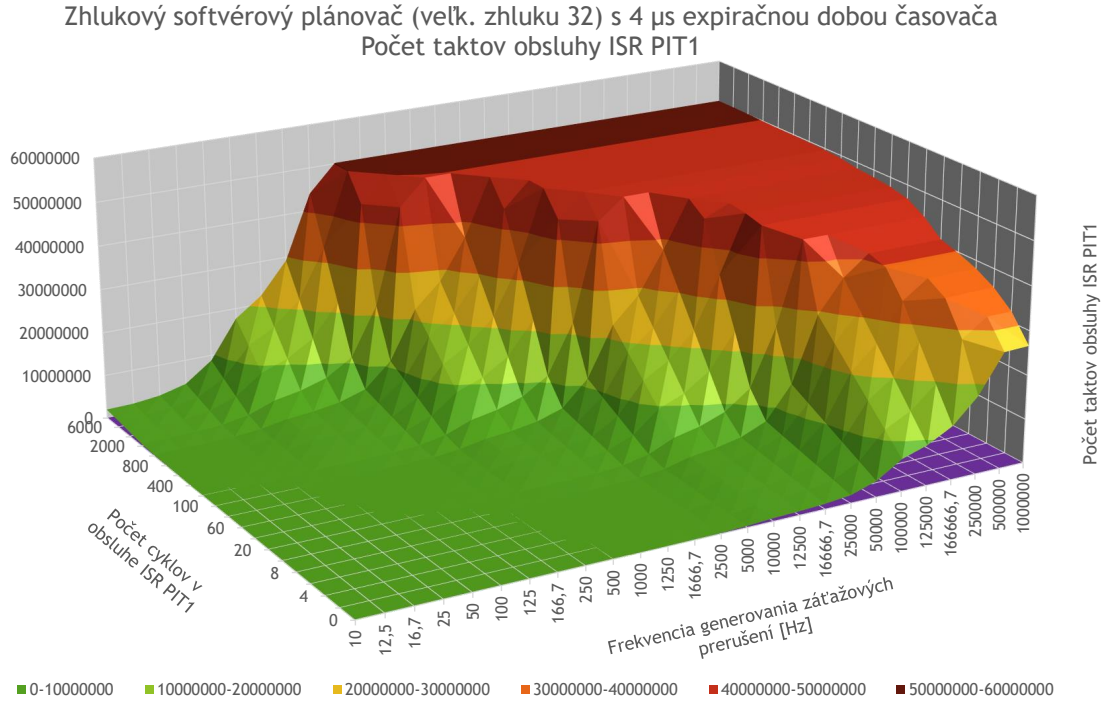
Obr. 5.20: Počet taktov obsluhy prerušení PIT1 pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 400 μs expiračnou dobou SysTick časovača.



Obr. 5.21: Počet taktov obsluhy prerušení PIT1 pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 4 μs expiračnou dobou SysTick časovača.



Obr. 5.22: Počet taktov obsluhy prerušení PIT1 pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 400 μs expiračnou dobou SysTick časovača.



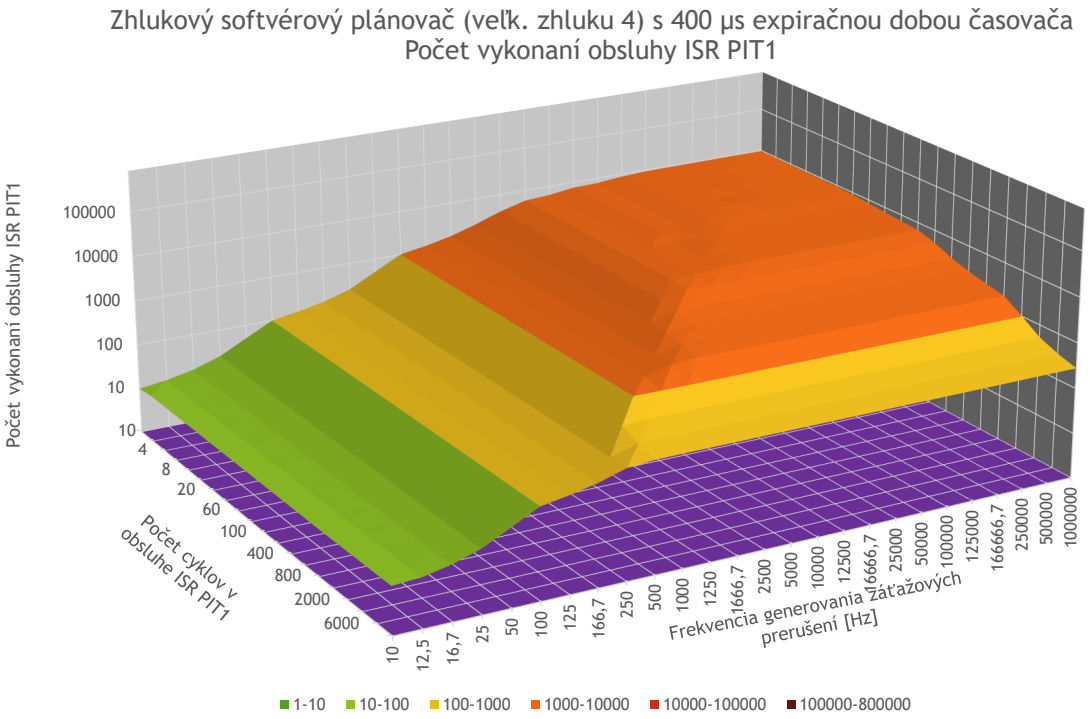
Obr. 5.23: Počet taktov obsluhy prerušení PIT1 pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 4 μs expiračnou dobou SysTick časovača.

Počet obslúžených prerušení záťažového časovača PIT1

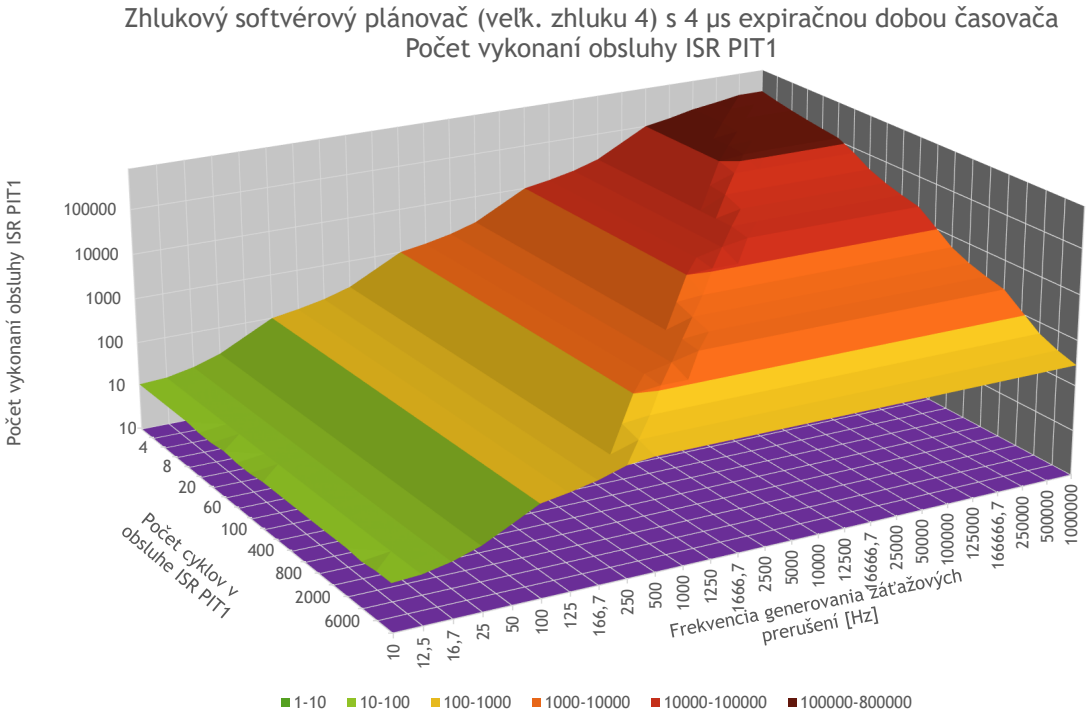
Posledná štvorica grafov zobrazuje počet vykonaných obslužných rutín záťažového prerušenia PIT1 počas trvania testu.

Z grafov je možné vidieť ako technika zhlukového softvérového plánovača postupne obmedzuje počet vykonaných prerušení počas jednej sekundy. Limit obmedzenia však v porovnaní s technikou striktného softvérového plánovača nie je taký nekompromisný. Najvýraznejšie sa obmedzenie prejavuje pre malú veľkosť zhluku a dlhý interval medzi zhlukmi, viď graf na obrázku č. 5.24. Znižovaním intervalu postupne narastá počet vykonaných prerušení, až po hodnoty v grafe na obrázku č. 5.25.

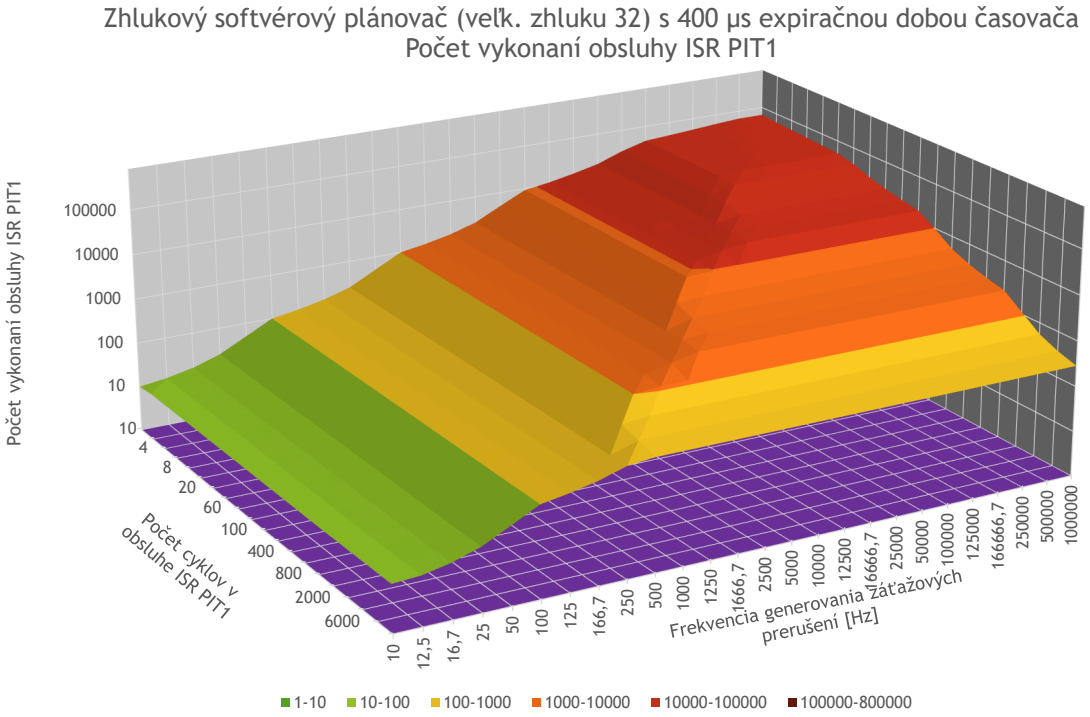
Zväčšením veľkosti zhluku zo 4 na 32 prerušení, viď graf na obrázku č. 5.26, pri zachovaní rovnakého intervalu medzi príchodom prerušení dôjde taktiež k nárastu prerušení počas trvania testu. V prípade nastavenia prehnane vysokej maximálnej veľkosti zhluku a príliš krátkeho intervalu medzi zhlukmi, viď graf na obrázku č. 5.26, technika zhlukového plánovania prerušení takmer vôbec nezabraňuje preťaženiu systému. Práve naopak, pridanou réžiou zvyšuje zaťaženie systému. V grafe je možné vidieť, že počet prerušení nie je nijako limitovaný a dosahuje referenčné hodnoty z Experimentu 2, viď graf na obrázku č. 4.13. Je to spôsobené tým, že veľké zhluky s krátkym časovým rozostupom neposkytujú procesorovému jadru dostatok času na vykonávanie hlavného programu, čo bolo vidieť aj v grafe na obrázku č. 5.19.



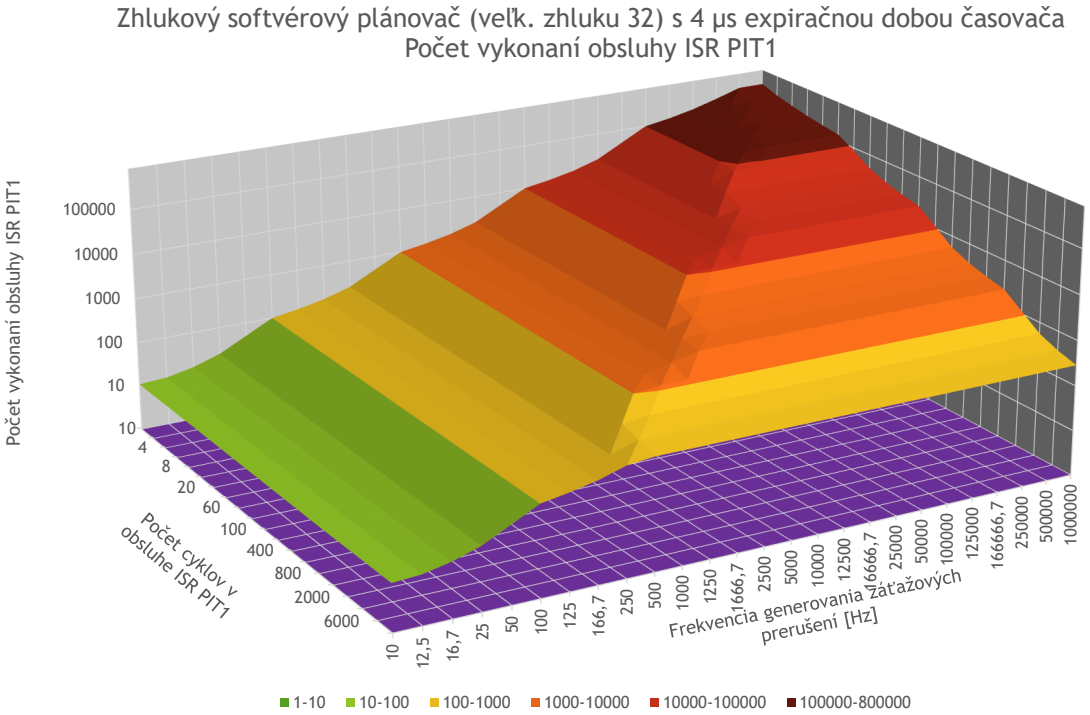
Obr. 5.24: Počet obslúžených prerušení PIT1 pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 400 μs expiračnou dobou SysTick časovača.



Obr. 5.25: Počet obslúžených prerušení PIT1 pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 4 μs expiračnou dobou SysTick časovača.



Obr. 5.26: Počet obslúžených prerušení PIT1 pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 400 μs expiračnou dobou SysTick časovača.



Obr. 5.27: Počet obslúžených prerušení PIT1 pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 4 μs expiračnou dobou SysTick časovača.

5.2.5 Réžia techniky zhlukového softvérového plánovača

Aj v tomto prípade bolo pre meranie časovej réžie techniky zhlukového softvérového plánovača potrebné vytvoriť samostatnú oddelenú implementáciu. Implementácia je na priloženom médiu v adresári `Technika2_Zhluk_SWq_overhead_measure`, viď príloha č. A. Dôvodom bolo zabránenie vzájomnému ovplyvňovaniu výsledkov medzi meraním výkonnosti plánovacej techniky a meraním časovej réžie. Ďalším dôvodom oddelenej implementácie bolo využitie komponenty DWT pre meranie procesorových taktov rovnakým spôsobom, ako pri meraní réžie striktného softvérového plánovača, opísaným v kapitole č. 5.1.5.

Réžia v tele obslužnej rutiny záťažového časovača PIT1

Pri meraní réžie boli sledované dva úseky. V prvom úseku bol zaznamenávaný počet taktov, ktoré sú pridané nad rámec ISR záťažového časovača PIT1. Konkrétne ide o počet taktov potrebných pre vykonanie kódu zobrazeného v obrázku č. 5.15, inkrementovanie hodnoty obsahujúcej aktuálnu veľkosť zhluku, overenie dosiahnutia maximálnej veľkosti zhluku a prípadné zakázanie obsluhy prerušení a spustenie časovača SysTick.

Pomocou komponenty DWT bolo zistené, že na inkrementovanie aktuálnej veľkosti zhluku a overenie podmienky s negatívnym výsledkom procesor vynaloží 33 taktov. V prípade ak dôjde ku kladnému vyhodnoteniu podmienky a vykonaniu bloku v podmienke, počet taktov vzrastie na 76.

Z týchto hodnôt je možné odvodiť minimálnu veľkosť zhluku pre zhlukový softvérový plánovač tak, aby bola celková úroveň réžie menšia ako pri striktnom softvérovom plánovači. Pri striktnom softvérovom plánovači bola pridaná réžia v ISR PIT1 57 taktov. Pre minimálnu veľkosť zhluku X preto platí nasledujúci vzťah:

$$(X * 57) \geq (X * 33) + 76$$

Z rovnice pre hodnotu X je možné vyčísliť, že $X \geq \frac{19}{6}$. Nakoľko ale veľkosť zhluku musí byť celočíselná hodnota, je preto pri technike zhlukového softvérového plánovača vhodné zvoliť maximálnu veľkosť zhluku najmenej 4.

Z rovnice vyplýva, že čím väčšia bude maximálna veľkosť zhluku, tým viac sa rozloží réžia v tele ISR záťažového časovača PIT1 medzi počet vyvolaných prerušení. Zároveň bolo v kapitole č. 5.2.2 pri počte vykonaných iterácií v tele hlavného programu zistené, že nadmerným zväčšovaním maximálnej veľkosti zhluku technika zhlukového softvérového plánovača stráca schopnosť zamedziť preťaženiu systému vplyvom nadmernej frekvencie prerušení.

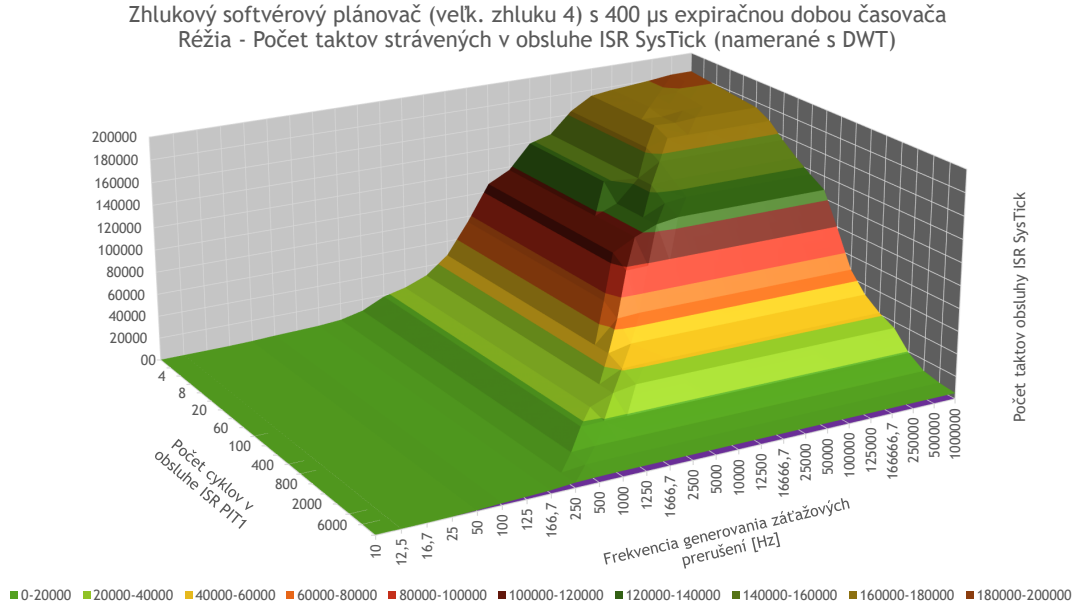
Réžia jednorazového časovača SysTick

Druhým sledovaným úsekom bol čas potrebný pre vykonanie ISR SysTick časovača pre opätovné povolenie prerušení a inicializáciu počítadla zhluku. V nasledujúcich grafoch na obrázkoch č. 5.28 až č. 5.31 je možné vidieť réžiu v tele ISR SysTick časovača pre zvolené hodnoty maximálnej veľkosti zhluku a limitu SysTick časovača, počas priebehu testu. Grafy s počtom vykonaných prerušení SysTick časovača počas 1 s je možné vidieť v prílohe E.

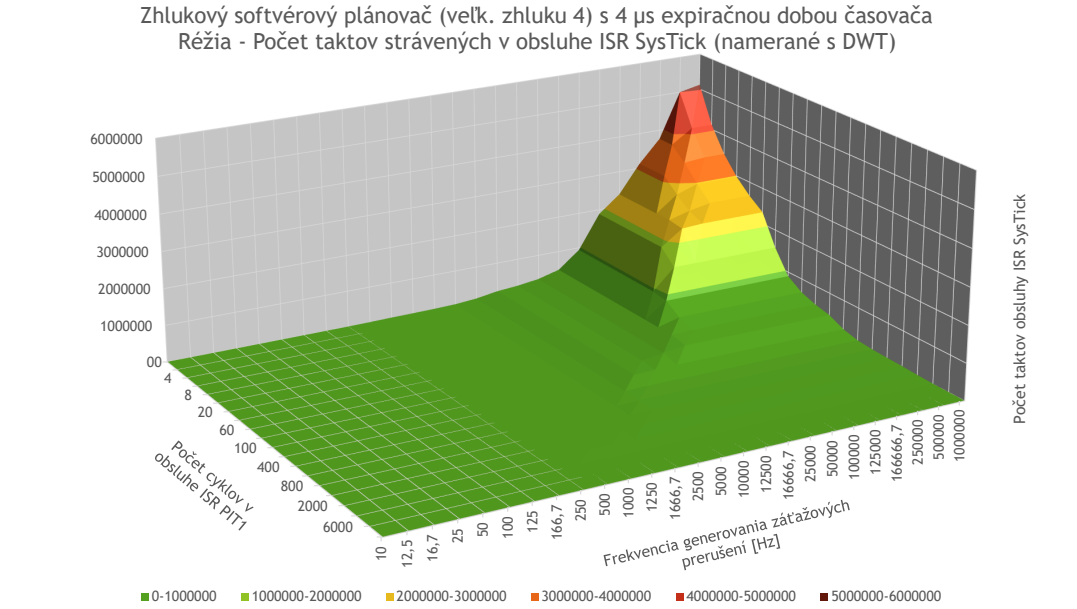
Telo obslužnej rutiny obsahuje navyše oproti striktnému softvérovému časovaču vynulovanie aktuálnej veľkosti zhluku. To spôsobilo navýšenie z pôvodných 68 na 75 taktov. Túto hodnotu je možné overiť vydelením hodnôt z grafu s počtom taktov (napr. obr. č. 5.28), hodnotami z grafu s počtom vykonaných ISR SysTick (napr. obr. č. E.1).

Napriek tomu, že počet taktov v obsluhe ISR SysTick je v porovnaní s predošlou technikou o 7 vyšší, tento počet sa pri technike zhlukového plánovača vzťahuje na celý zhluk, čím opäť dochádza k výraznej amortizácii počtu taktov réžie medzi jednotlivé prerušenia.

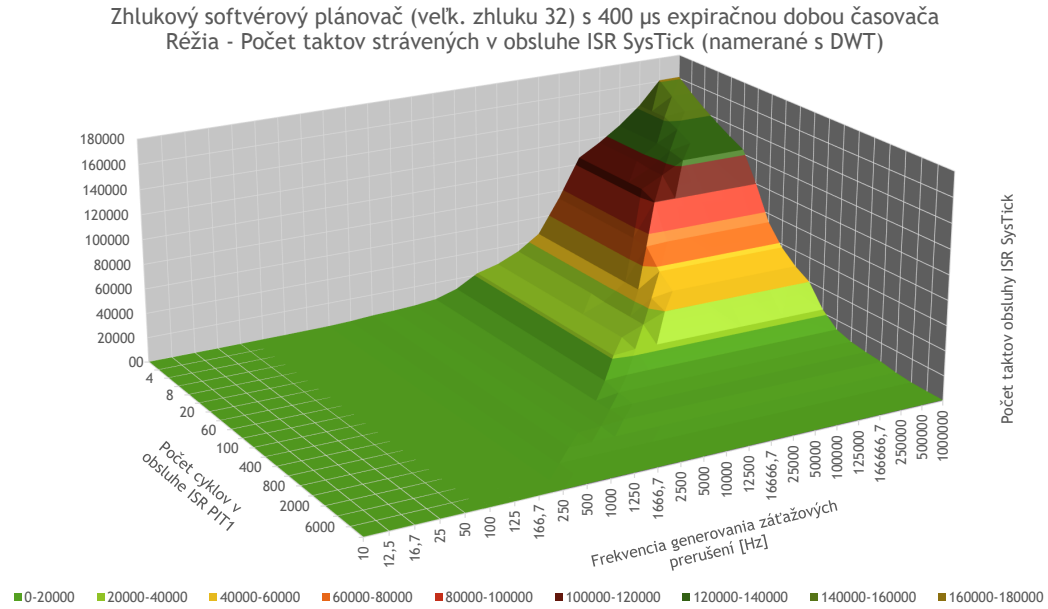
Hlavnou prednosťou zhlukového softvérového plánovača je teda redukcia počtu vyvolaných obslužných rutín jednorazového SysTick časovača. Tým sa vďaka rozloženiu dĺžky trvania ISR medzi zhluk obslúžených prerušení zníži celková réžia techniky zhlukového plánovača.



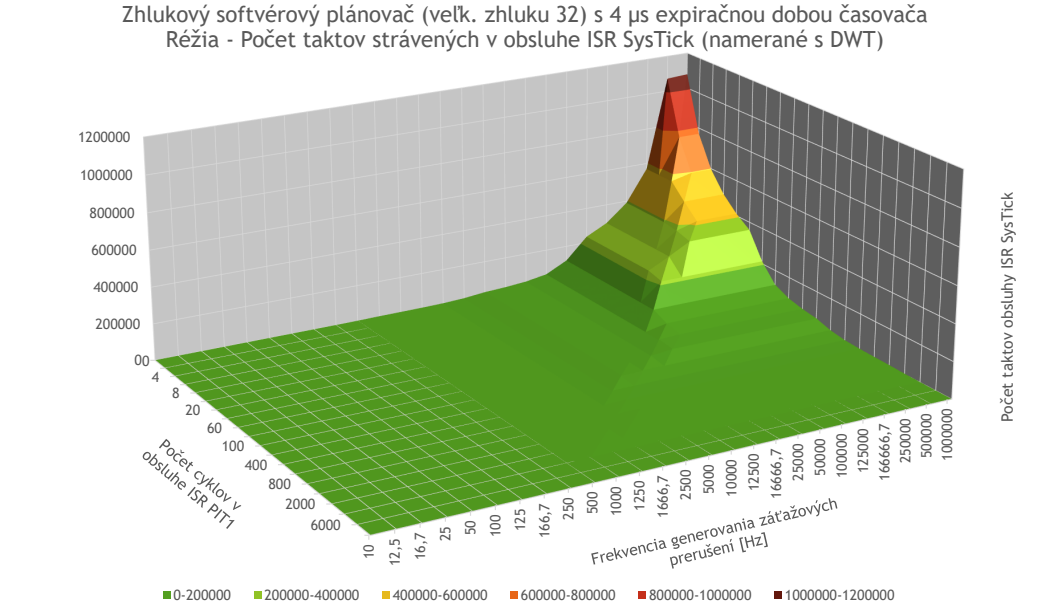
Obr. 5.28: Počet taktov réžie v ISR SysTick počas 1 s pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 400 μs expiračnou dobou SysTick časovača.



Obr. 5.29: Počet taktov réžie v ISR SysTick počas 1 s pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 4 μs expiračnou dobou SysTick časovača.



Obr. 5.30: Počet taktov réžie v ISR SysTick počas 1 s pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 400 μs expiračnou dobou SysTick časovača.



Obr. 5.31: Počet taktov réžie v ISR SysTick počas 1 s pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 4 μs expiračnou dobou SysTick časovača.

Nakoľko meranie prebiehalo opakovane, ale na rovnakej sade vstupných parametrov, meranie réžie trvalo taktiež 3 hodiny a 28 minút. V tomto prípade však bolo získaných len 24 960 údajov, polovica pre počet taktov strávených v ISR SysTick a druhá polovica pre počet vykonaných obslužných rutín SysTick časovača. Z nameraných hodnôt bolo vytvorených 48 grafov dostupných na priloženom médiu v adresári `Technika2` rozdelených podľa maximálnej veľkosti zhluku, viď príloha A.

5.3 Porovnanie výsledkov implementovaných techník

Všetky experimentálne namerané hodnoty techník striktného a zhlukového softvérového plánovača sú vizualizované vo forme grafov, ktoré zobrazujú ako zvolené techniky ovplyvňujú stavový priestor obsluhy prerušení na platforme FITkit 3.0, a z ktorých je možné vidieť efektívnosť plánovacích techník vzhľadom k referenčným výsledkom nameraným bez použitého plánovača.

Pri vzájomnom porovnaní techník je však potrebné uviesť, že porovnanie výsledkov medzi technikou zhlukového softvérového plánovača a technikou striktného softvérového plánovača nie je možné vykonávať iba na základe korešpondujúcej konfigurácie periódy SysTick časovača.

Dôvodom je, že každá technika používa SysTick časovač na meranie rozdielnych úsekov. Pri zhlukovom plánovači je použitý na meranie časového limitu medzi zhlukmi, pri striktnom plánovači na meranie časového limitu medzi jednotlivými prerušeniami. Z tohto dôvodu je vylúčená vzájomná korelácia podľa limitu SysTick časovača, nakoľko počas trvania jedného testu (1 s), dôjde v každej technike k obslúženiu rozdielneho počtu prerušení. Pri snahe o porovnanie techník je preto potrebné výsledky porovnávať individuálne.

Pre demonštráciu a overenie vlastností implementovaného striktného a zhlukového plánovača je vzájomné porovnanie realizované pre rôzne nastavenie plánovacích techník, ale pre jeden konkrétny prípad konfigurácie záťažových prerušení PIT1 časovača, viď tabuľka č. 5.1.

Porovnanie je vykonávané pre približne 2 miliónový počet iterácií vykonaných v tele hlavného programu. Podľa toho je pre porovnanie zvolená aj príslušná konfigurácia plánovacej techniky. Pre striktný softvérový plánovač je to pri čase 400 μs medzi prerušeniami. Pri technike zhlukového softvérového plánovača je to pri čase 1 ms medzi zhlukmi pre veľkosť zhluku 4 a pri čase 8 ms pre veľkosť zhluku 32. Záťažové prerušenia vo všetkých prípadoch prichádzali s frekvenciou 1 MHz a v tele obsluhy sa vykonával nulový počet iterácií (tj. obsluha ISR PIT1 trvala 113 taktov).

Konfigurácia záťažových prerušení PIT1 1 MHz, 0 iterácií v ISR	Použitá technika		
	Striktný SW plánovač 400 μs	Zhlukový SW plánovač	
		veľkosť zhluku 4 1 ms	veľkosť zhluku 32 8 ms
Počet vykonaných prerušení PIT1	82 510	124 381	131 850
Počet taktov venovaných obsluhe ISR PIT1	6 353 270	9 111 068	9 513 804
Počet iterácií v hlavnom cykle	2 035 187	2 013 336	2 189 249
Počet vykonaných ISR SysTick	80 907	32 637	4 435
Počet taktov venovaných obsluhe ISR SysTick	5 501 674	2 447 773	332 623

Tabuľka 5.1: Porovnanie výsledkov implementovaných techník pre konkrétnu konfiguráciu generovania záťažových prerušení PIT1.

Pri technike striktného softvérového plánovača je počas testu vykonaných 2 035 187 iterácií ale obslúži sa iba 82 510 prerušení. Pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 bolo vykonaných 124 381 prerušení ale zároveň sa znížil počet iterácií venovaných obsluhe hlavného programu.

Najlepšie výsledky dosahuje v tomto prípade technika zhlukového softvérového plánovača pre veľkosť zhluku 32, kedy je vykonaných až 131 850 prerušení PIT1 a zároveň je v hlavnom programe vykonaných až 2 189 249 iterácií. To dokazuje, že v tomto prípade sa vykoná najviac prerušení pri najväčšom množstve času venovaného obsluhe hlavného programu. Je to najmä vďaka redukcii počtu režijných prerušení SysTick časovača, ktorých bolo v tomto prípade vykonaných len 4 435.

V tabuľke je možné vidieť ako súvisí efekt redukovania počtu vykonaných prerušení jednorazového plánovacieho časovača SysTick s efektivitou plánovacej techniky prerušení. Kým v technike striktného plánovača bol počet záťažových prerušení a počet režijných prerušení približne rovnaký, v technike zhlukového plánovača umožňovala veľkosť zhluku efektívne amortizovať počet prerušení plánovača medzi zhluk záťažových prerušení tým výraznejšie, čím bol zhluk väčší.

V prílohe č. F je dostupná tabuľka zobrazujúca porovnanie plánovacích techník pre inú zvolenú konfiguráciu časovača PIT1 generujúceho záťažové prerušenia.

Kapitola 6

Záver

Cieľom tejto práce bolo zdokumentovať techniky prístupu k detekcii výskytu udalostí v počítačových systémoch a možné príčiny vzniku výpočtového preťaženia počítačového systému, ktoré sú spôsobené nadmernou frekvenciou prerušení. Práca popisuje dopady tohto typu preťaženia na počítačový systém. V práci sú preskúmané a zdokumentované techniky, ktoré sa snažia týmto nedostatkom vyhýbať, obmedziť ich vplyv alebo odstrániť nežiadúce dôsledky vznikajúce pri nadmernej frekvencii prerušení.

Práca pomocou návrhu a implementácie vhodnej sady experimentov vyhodnocuje vplyv nadmernej frekvencie prerušení na zvolenú výpočtovú platformu FITkit 3.0 Minerva s procesorom ARM Cortex-M4.

Súčasťou práce je implementácia techník striktného a zhlukovného softvérového plánovača zabraňujúcich preťaženiu z dôvodu prerušení na zvolenej platforme. Schopnosti týchto techník zamedzovať preťaženiu sú overené a vyhodnotené na vhodne navrhnutej sade experimentov. Overenie výsledkov zahŕňa porovnanie efektivity jednotlivých techník a poukazuje na možné nedostatky a prípadné zlepšenia.

Výsledky zároveň poukazujú na to, ako je pre správnu funkčnosť počítačového systému dôležitý výber vhodnej plánovacej techniky obsluhy prerušení. To platí najvýznamnejšie pri vstavaných a real-time systémoch, od ktorých sa vyžaduje presná garancia maximálneho oneskorenia pri spracovaní alebo garancia minimálneho času venovaného vykonávaniu hlavného programu.

Porovnanie implementovaných techník a výsledky experimentov tejto práce môžu byť v budúcnosti využité na zefektívnenie a integráciu jednotlivých techník zamedzenia výpočtového preťaženia z dôvodu nadmernej frekvencie prerušení, priamo do hardvéru radiča prerušení s prípadnou možnosťou výberu požadovanej techniky a jej parametrov.

Výsledky práce môžu byť taktiež v budúcnosti využité na návrh efektívnejšieho radiča prerušení z hľadiska preempcie, plánovania a rýchlejšieho spracovania výskytu asynchrónnych prerušení v počítačových systémoch a vstavaných zariadeniach.

Literatúra

[1] Kinetis K60-100 MHz Mixed-Signal Integration Microcontrollers based on Arm Cortex-M4 Core. NXP Semiconductors, 2006 - 2019, [Online; navštíveno 1.11.2018].
URL https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/kinetis-cortex-m-mcus/k-seriesperformancem4/k6x-ethernet/kinetis-k60-100-mhz-mixed-signal-integration-microcontrollers-based-on-arm-cortex-m4-core:K60_100

[2] ARMv7-M Architecture Reference Manual. 2010: str. 1020, [Online; navštíveno 1.11.2018].
URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0403e.b/index.html

[3] Cortex-M4 Devices Generic User Guide. 2010-2011: str. 277, [Online; navštíveno 1.11.2018].
URL http://infocenter.arm.com/help/topic/com.arm.doc.dui0553b/DUI0553.pdf

[4] Peripherals of Freescale Kinetis microcontrolers. 2013, [Online; navštíveno 24.01.2019].
URL https://www.dsi.fceia.unr.edu.ar/images/downloads/InformaticaAplicada/PheriperalsPart1.pdf

[5] Kinetis Design Studio IDE. Freescale Semiconductors, 2014-2015, [Online; navštíveno 1.11.2018].
URL https://www.nxp.com/docs/en/fact-sheet/KINDESTDSOFS.pdf

[6] Kinetis K6x MCU Family. Freescale Semiconductors, 2015, [Online; navštíveno 1.11.2018].
URL https://www.nxp.com/docs/en/fact-sheet/KINK6XFS.pdf

[7] ARM C Language Extensions. 24.03.2016, [Online; navštíveno 20.02.2019].
URL http://infocenter.arm.com/help/topic/com.arm.doc.ihi0053d/IHI0053D_acle_2_1.pdf

[8] MINERVA KIT Výuková a experimentální platforma s FPGA a MCU. c2012-2019, [Online; navštíveno 1.11.2018].
URL https://minerva.php5.cz/minerva/minerva.php

[9] K60 Sub-Family Reference Manual, Jun 2012, [Online; navštíveno 1.11.2018].
URL https://www.nxp.com/docs/en/reference-manual/K60P144M100SF2V2RM.pdf

[10] CMSIS Version 5.5.1. Mar 21 2019, [Online; navštíveno 24.01.2019].
URL https://www.keil.com/pack/doc/CMSIS/General/html/index.html

[11] Buchanan, W.: *Applied PC Interfacing, Graphics and Interrupts*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996, ISBN 0-201-87728-7.

[12] Douglass, B. P.: *Design Patterns for Embedded Systems in C : An Embedded Software Engineering Toolkit*. Oxford: Elsevier Science & Technology, 2010, ISBN 9781856177078.

[13] Leyva-del Foyo, L. E.; Mejía-Alvarez, P.: Custom Interrupt Management for Real-Time and Embedded System Kernels. 2004: str. 8.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.7025

[14] Leyva-del Foyo, L. E.; Mejia-Alvarez, P.; de Niz, D.: Predictable Interrupt Management for Real Time Kernels over conventional PC Hardware. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, April 2006, ISSN 1545-3421, s. 14–23, doi:10.1109/RTAS.2006.34.

[15] Leyva-del Foyo, L. E.; Mejia-Alvarez, P.; de Niz, D.: Integrated Task and Interrupt Management for Real-Time Systems. *ACM Trans. Embed. Comput. Syst.*, ročník 11, č. 2, Júl 2012: s. 32:1–32:31, ISSN 1539-9087, doi:10.1145/2220336.2220344.
URL http://doi.acm.org/10.1145/2220336.2220344

[16] Hofer, W.; Lohmann, D.; Schröder-Preikschat, W.: Sleepy Sloth: Threads as Interrupts as Threads. In *2011 IEEE 32nd Real-Time Systems Symposium*, Nov 2011, ISSN 1052-8725, s. 67–77, doi:10.1109/RTSS.2011.14.

[17] Mejia-Alvarez, P.; Leyva-del Foyo, L. E.; Diaz-Ramirez, A.: *Interrupt Handling Schemes in Operating Systems*. SpringerBriefs in Computer Science, Cham: Springer International Publishing, 2018, ISBN 978-3-319-94492-0.

[18] Patel, P.; Vanga, M.; Brandenburg, B. B.: TimerShield: Protecting High-Priority Tasks from Low-Priority Timer Interference (Outstanding Paper). In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017, s. 3–12, doi:10.1109/RTAS.2017.40.

[19] Regehr, J.; Duongsaa, U.: Preventing Interrupt Overload. In *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES '05, New York, NY, USA: ACM, 2005, ISBN 1-59593-018-3, s. 50–58, doi:10.1145/1065910.1065918.
URL http://doi.acm.org/10.1145/1065910.1065918

[20] Schlagenhaft, R. D.; Robertson, A. P.: *Detector for high frequency interrupts*. US9612894B2. Přihlášeno 1. 6. 2015. Uděleno 4. 4. 2017. Zapsáno 1. 12. 2016.
URL https://patentimages.storage.googleapis.com/89/12/3d/a170bd1533ad92/US9612894.pdf

[21] Strnadel, J.: On design of priority-driven load-adaptive monitoring-based hardware for managing interrupts in embedded event-triggered real-time systems. In *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2013, s. 24–29, doi:10.1109/DDECS.2013.6549783.

[22] Vašíček, Z.: Úvod - FITkit. FIT VUT v Brně, 206 - 2012, [Online; navštíveno 1.11.2018].
URL https://merlin.fit.vutbr.cz/FITkit/uvod.html

[23] Yiu, J.: *The definitive guide to ARM CORTEX-M3 and CORTEX-M4 processors.* Technology/electronics/engineering, Oxford: Elsevier ; Newnes, tretie vydanie, 2014, ISBN 978-0-12-408082-9.

[24] Yiu, J.: Beginner guide on interrupt latency and Arm Cortex-M processors. 2016, [Online; navštíveno 1.11.2018].
URL
https://community.arm.com/processors/b/blog/posts/beginner-guide-on-interrupt-latency-and-interrupt-latency-of-the-arm-cortex-m-processors

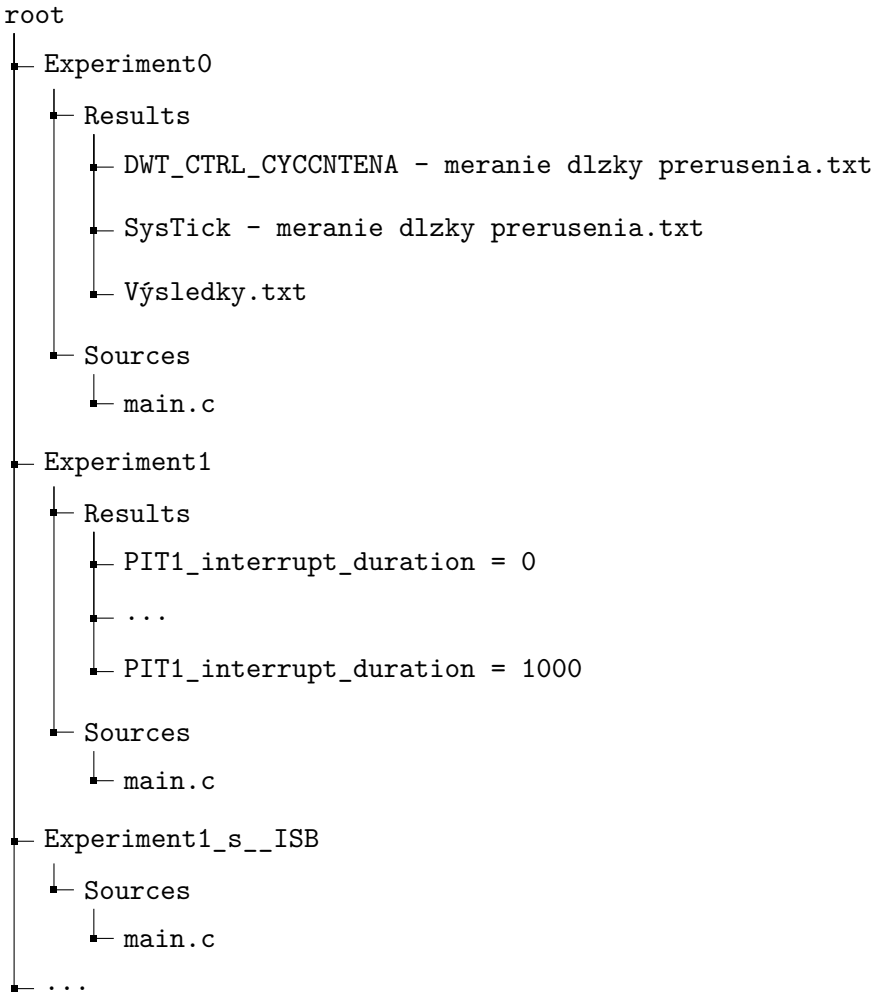
[25] Yiu, J.: ARM Cortex-M for Beginners. September 2016, [Online; navštíveno 1.11.2018].
URL https://community.arm.com/processors/b/blog/posts/white-paper-cortex-m-for-beginners-an-overview-of-the-arm-cortex-m-processor-family-and-comparison

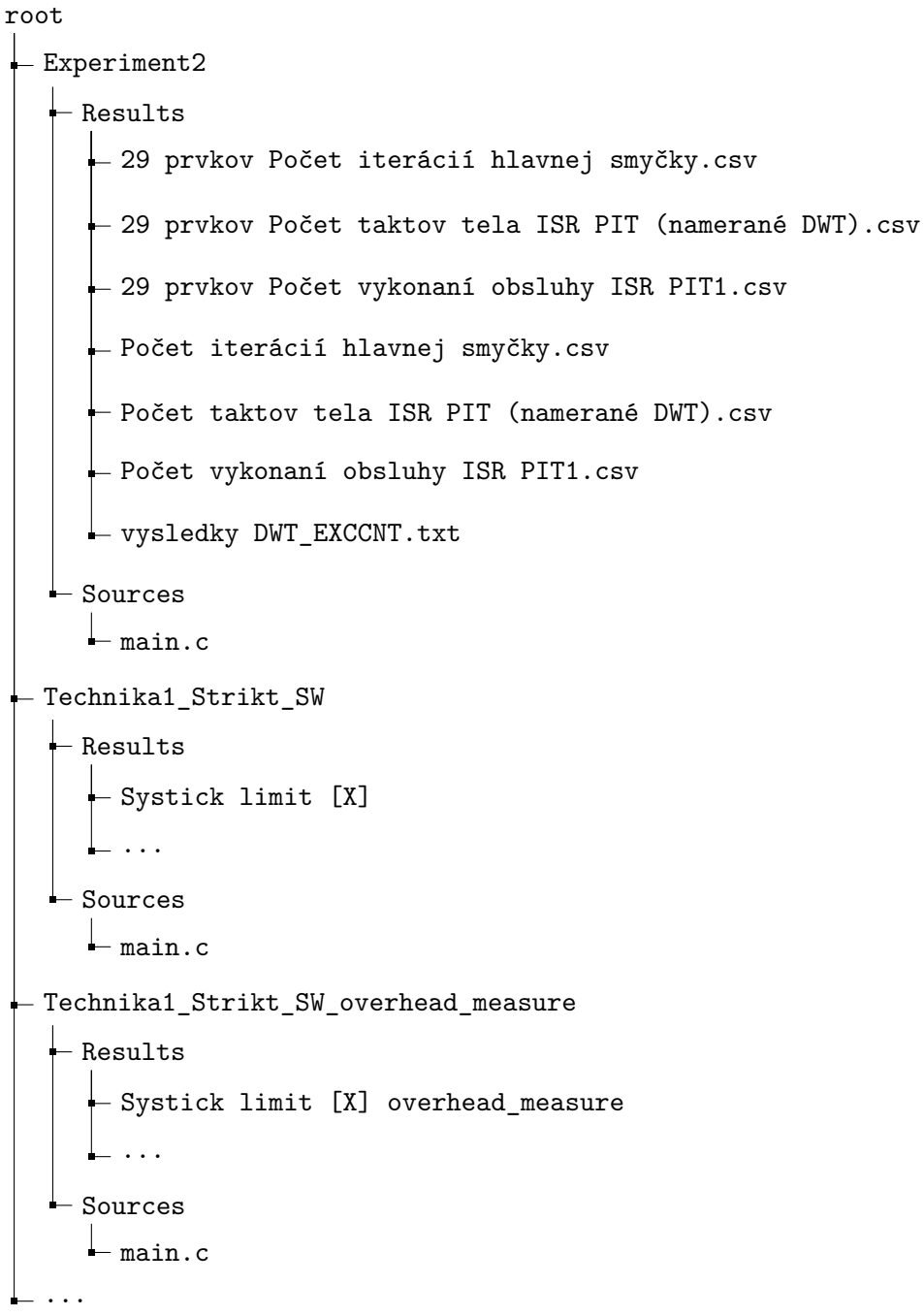
Zoznam použitých skratiek

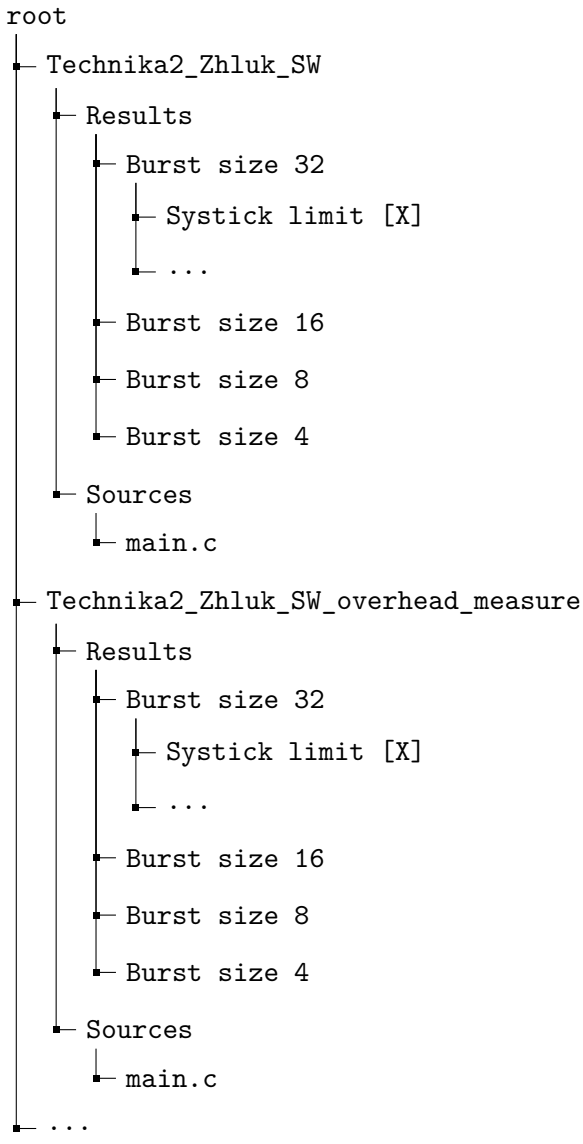
ACLE	ARM C Language Extension
APSR	Application Program Status Register
BLPE	Bypassed Low Power External
CMSIS	Cortex Microcontroller Software Interface Standard
CPIC	Custom Programmable Interrupts Controller (Programovateľný radič prerušení)
DWT	Data WatchPoint and Trace jednotka
EOI	End of Interrupt (Koniec prerušenia)
EPSR	Execution Program Status Register
FCU	Fault colection unit (Jednotka zhromažďovania porúch)
FEI	FLL engaged internal
FPGA	Field Programmable Gate Array (Programovateľné hradlové pole)
GCC	GNU Compiler Collection
GDB	GNU Debugger
IMR	Interrupt mask register (Register bitovej masky prerušení)
INTA	Interrupt acknowledge cycle (Cyklus potvrdzujúci prerušenie)
IPI	Inter-procesor interrupt (Medziprocesorové prerušenia)
IPSR	Interrupt Program Status Register
ISB	Instruction Synchronization Barrier (Inštrukčná synchronizačná bariéra)
ISR	Interrupt service routine (Obslužná rutina prerušenia)
IoT	Internet of Things (Internet vecí)
KDS	Kinetis Design Studio
MCG	Multipurpose Clock Generator (Viacúčelový generátor hodín)
MI	Maskable interupt (Maskovateľné prerušenie)
NMI	Non Maskable Interrupt (Nemaskovateľné prerušenie)
NMI	Non-maskable interupt (Nemaskovateľné prerušenie)
NVIC	Nested Vectored Interrupt Controller (Radič vnorených prerušení)
PIC	Programmable Interupt Controller (Programovateľný radič prerušení)
PIT	Periodic Interrupt Timer (Preiodický časovač generujúci prerušenia)
PSR	Program Status Register
RISC	Reduced Instruction Set Computing (Redukovaná inštrukčná sada)
SWV	Serial Wire Viewer
TIF	Timer Interrupt Flag (Príznak prerušenia z časovača)
VTOR	Vector Table Offset Register
WCET	Worst-case execution time (Najhorší čas vykonávania)

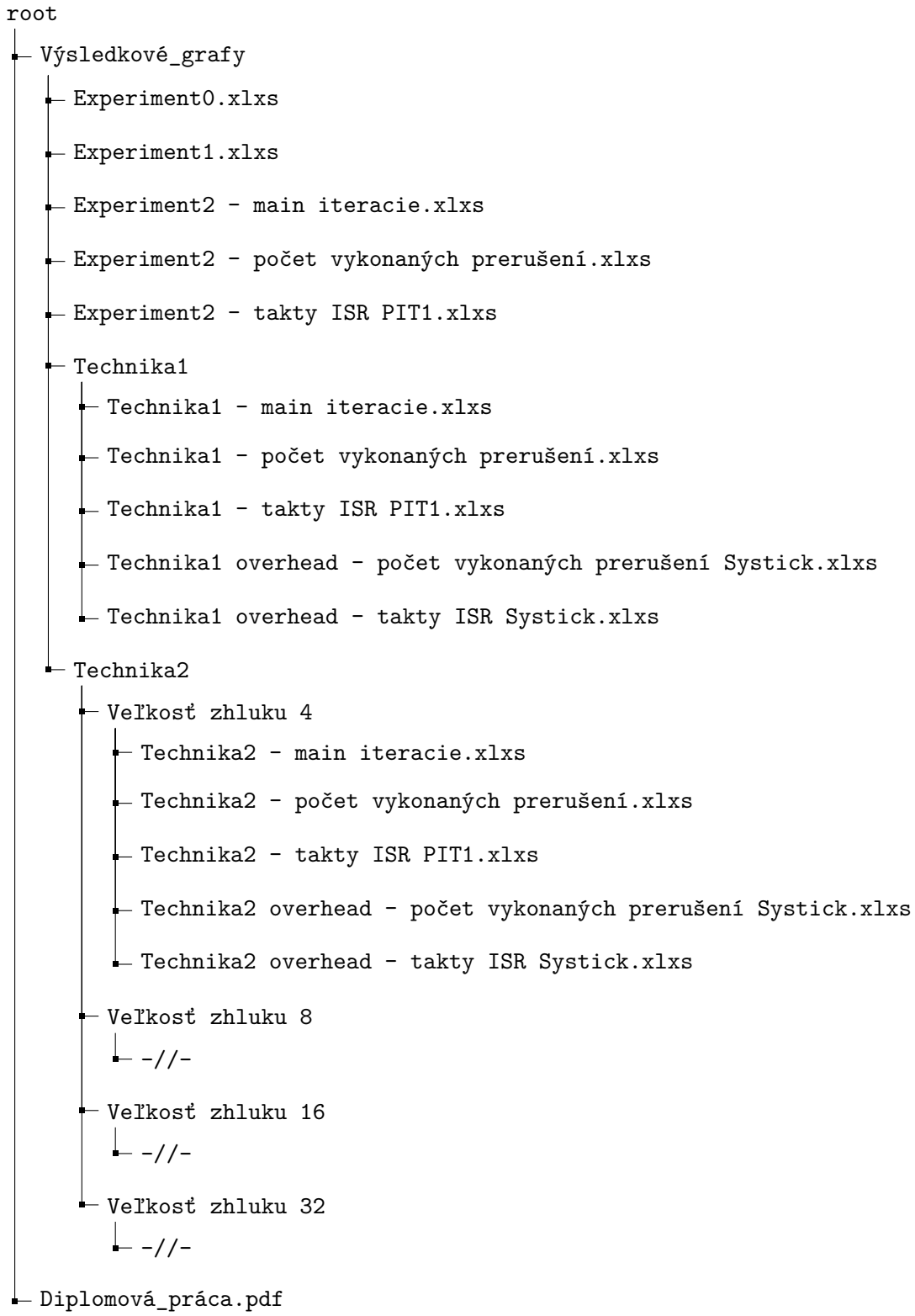
Príloha A

Obsah priloženého pamäťového média



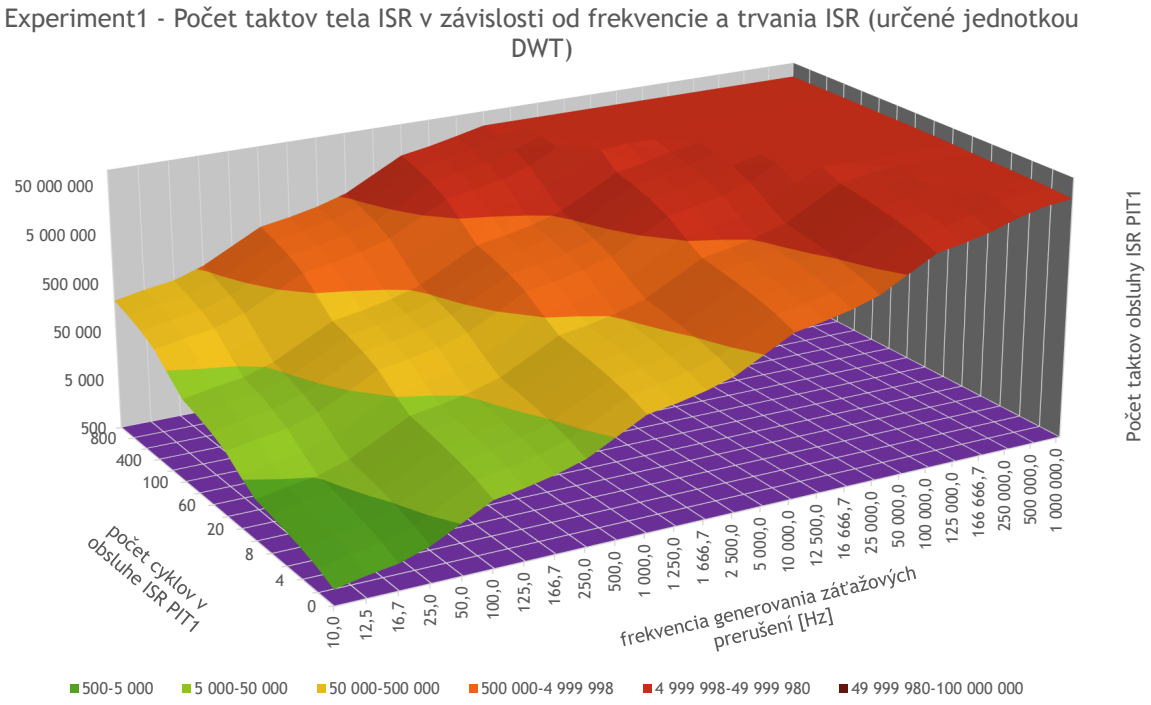

root
Experiment2
Results
29 prvkov Počet iterácií hlavnej smyčky.csv
29 prvkov Počet taktov tela ISR PIT (namerané DWT).csv
29 prvkov Počet vykonaní obsluhy ISR PIT1.csv
Počet iterácií hlavnej smyčky.csv
Počet taktov tela ISR PIT (namerané DWT).csv
Počet vykonaní obsluhy ISR PIT1.csv
vysledky DWT_EXCCNT.txt
Sources
main.c
Technika1_Strikt_SW
Results
Systick limit [X]
...
Sources
main.c
Technika1_Strikt_SW_overhead_measure
Results
Systick limit [X] overhead_measure
...
Sources
main.c
...


root
Technika2_Zhluk_SW
Results
Burst size 32
Systick limit [X]
...
Burst size 16
Burst size 8
Burst size 4
Sources
main.c
Technika2_Zhluk_SW_overhead_measure
Results
Burst size 32
Systick limit [X]
...
Burst size 16
Burst size 8
Burst size 4
Sources
main.c
...

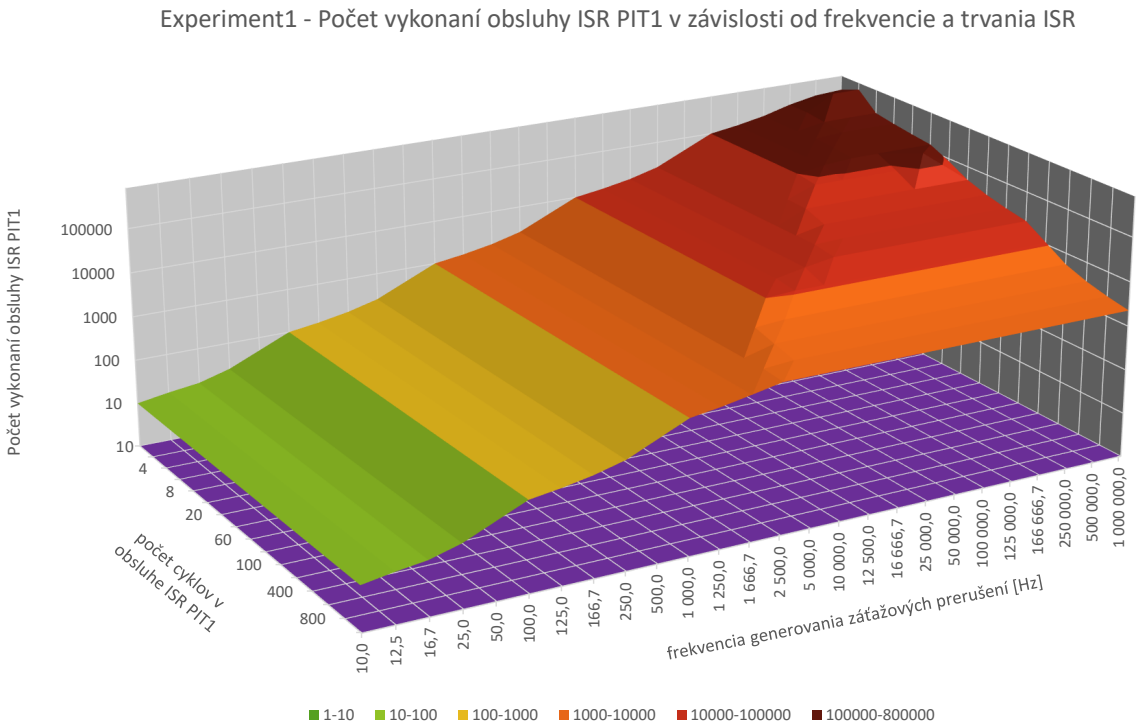

root
Výsledkové_grafy
Experiment0.xlxs
Experiment1.xlxs
Experiment2 - main iteracie.xlxs
Experiment2 - počet vykonaných prerušení.xlxs
Experiment2 - takty ISR PIT1.xlxs
Technika1
Technika1 - main iteracie.xlxs
Technika1 - počet vykonaných prerušení.xlxs
Technika1 - takty ISR PIT1.xlxs
Technika1 overhead - počet vykonaných prerušení Systick.xlxs
Technika1 overhead - takty ISR Systick.xlxs
Technika2
Veľkosť zhluku 4
Technika2 - main iteracie.xlxs
Technika2 - počet vykonaných prerušení.xlxs
Technika2 - takty ISR PIT1.xlxs
Technika2 overhead - počet vykonaných prerušení Systick.xlxs
Technika2 overhead - takty ISR Systick.xlxs
Veľkosť zhluku 8
-//-
Veľkosť zhluku 16
-//-
Veľkosť zhluku 32
-//-
Diplomová_práca.pdf

Príloha B

Experiment 1



Obr. B.1: Maximálny počet taktov strávených obsluhou prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1 v logaritmickom merítku.

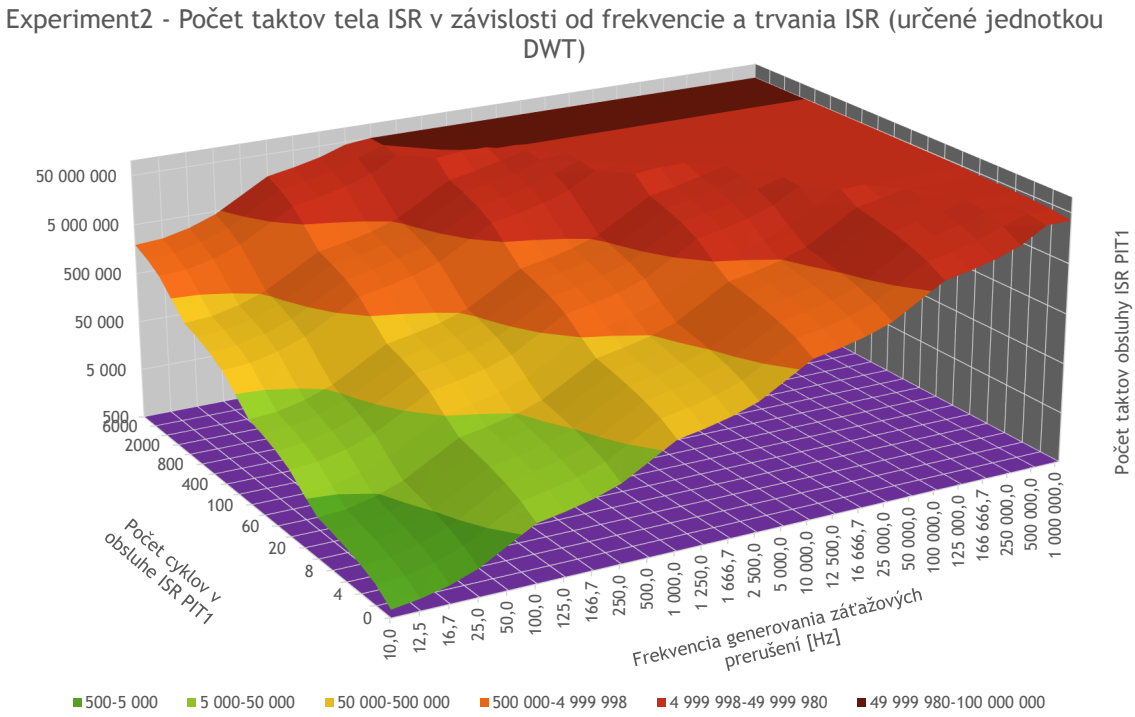


Obr. B.2: Maximálny počet obslúžených prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1 v logaritmickom merítku.

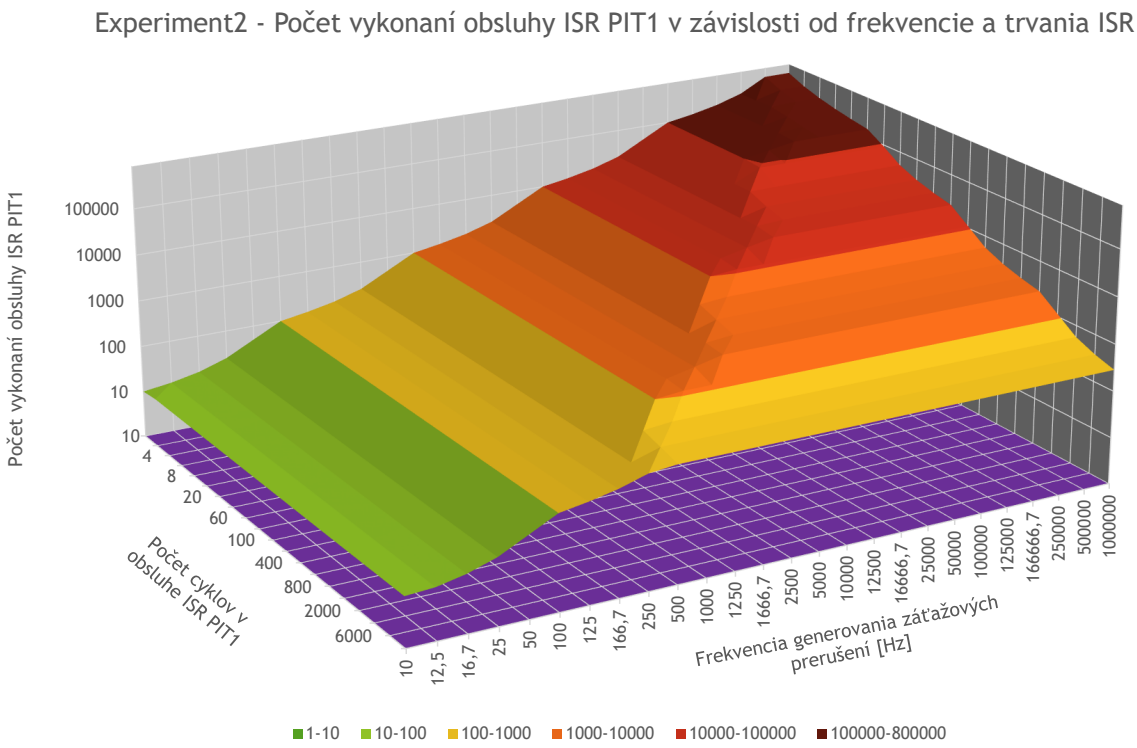
Príloha C

Experiment 2

C.1 Základná konfigurácia



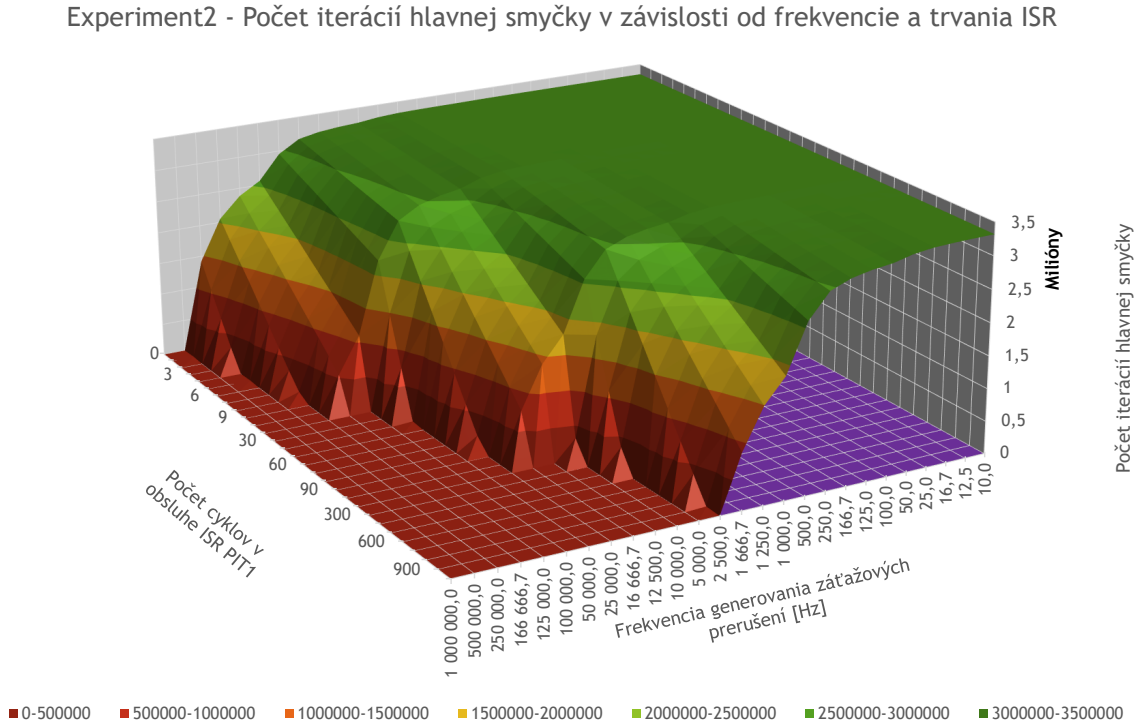
Obr. C.1: Počet taktov strávených obsluhou prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1 v logaritmickom merítku.



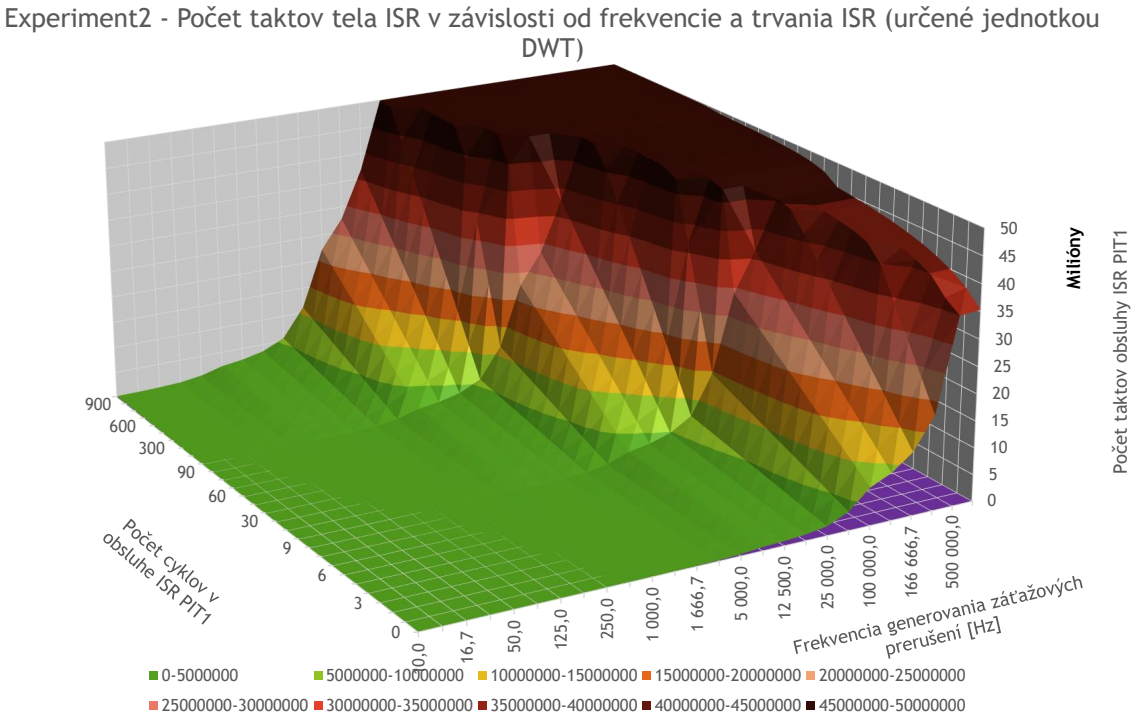
Obr. C.2: Počet obslúžených prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1 v logaritmickom merítku.

C.2 Rozšírená konfigurácia

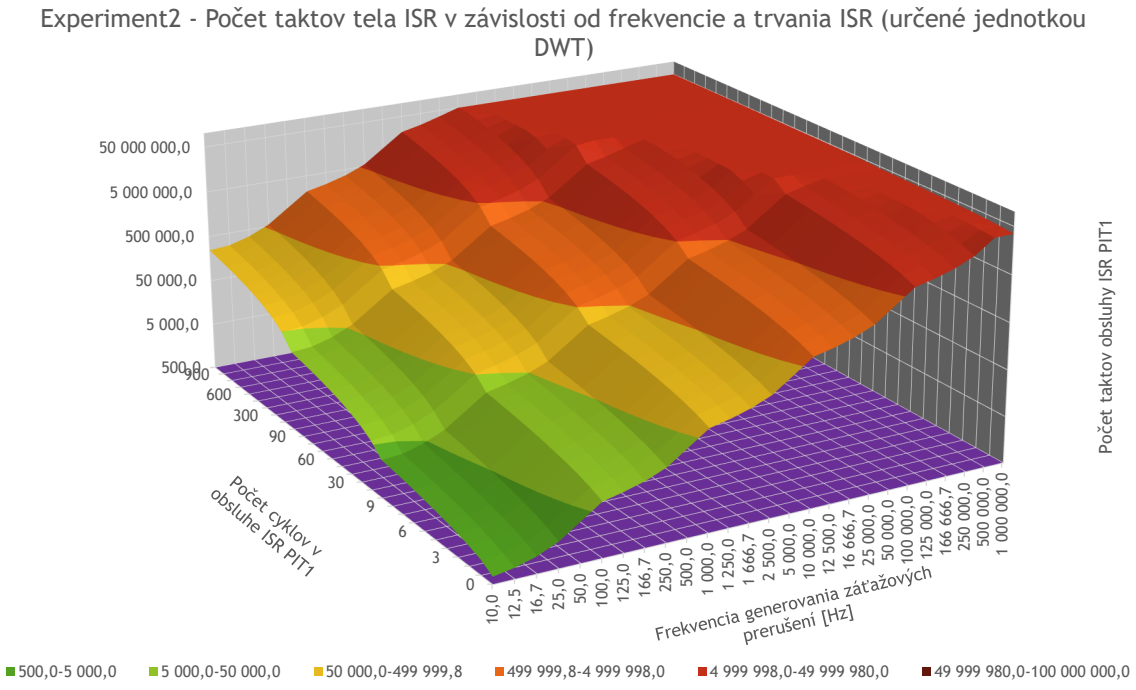
Rozšírená konfigurácia experimentu modifikovaná zmenšením intervalu medzi jednotlivými konfiguráciami. Konfigurácia bola modifikovaná zmenou dĺžky trvania ISR PIT1 v rozsahu hodnôt 1 až 1000 iterácií záťažového cyklu s menším krokom. Grafy Experimentu 2 z kapitoly 4.3 sú tak zobrazené pre menšiu oblasť s menším vykresľovacím krokom.



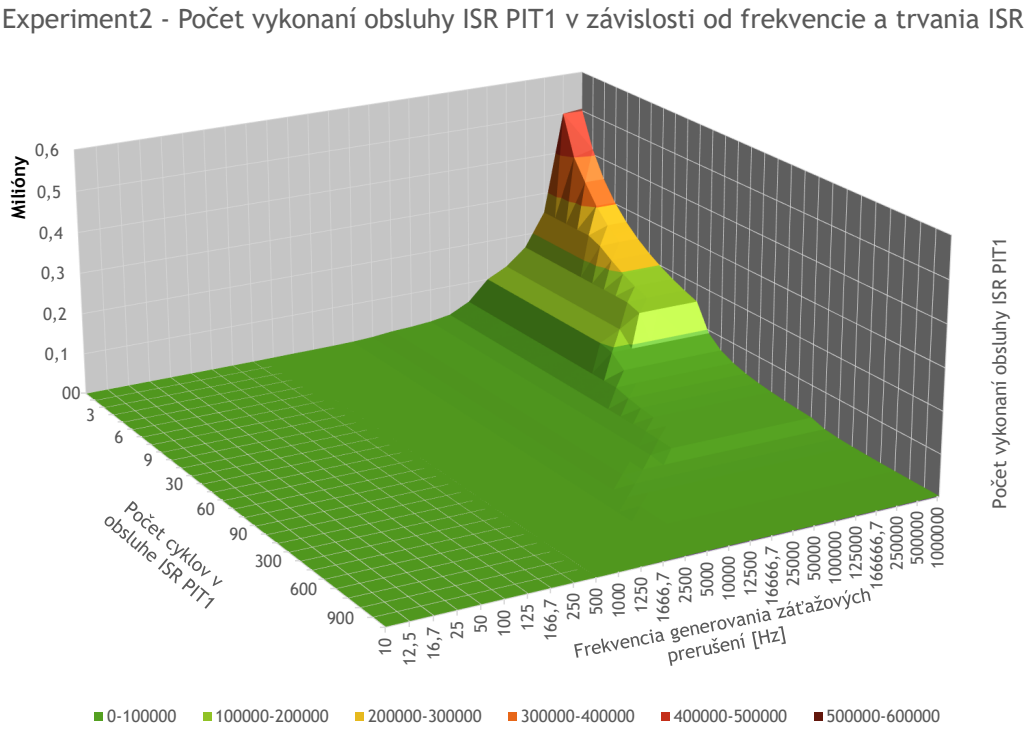
Obr. C.3: Počet iterácií hlavného programu v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1, pri menšom konfiguračnom kroku.



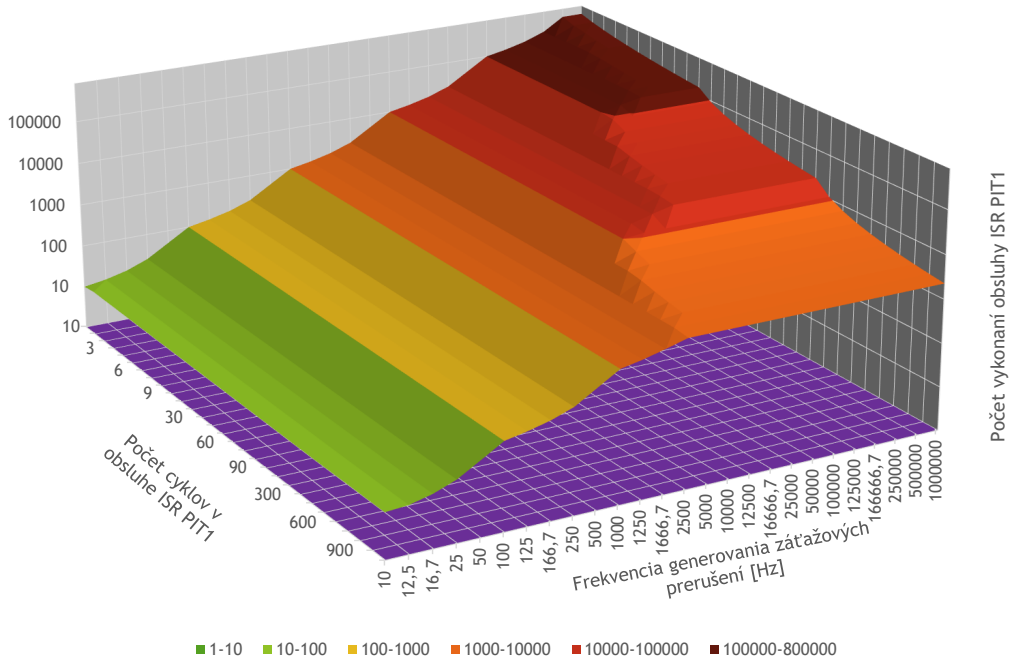
Obr. C.4: Počet taktov strávených obsluhou prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1, pri menšom konfiguračnom kroku.



Obr. C.5: Počet taktov strávených obsluhou prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1 v logaritmickom merítku, pri menšom konfiguračnom kroku.



Obr. C.6: Počet obslúžených prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1, pri menšom konfiguračnom kroku.

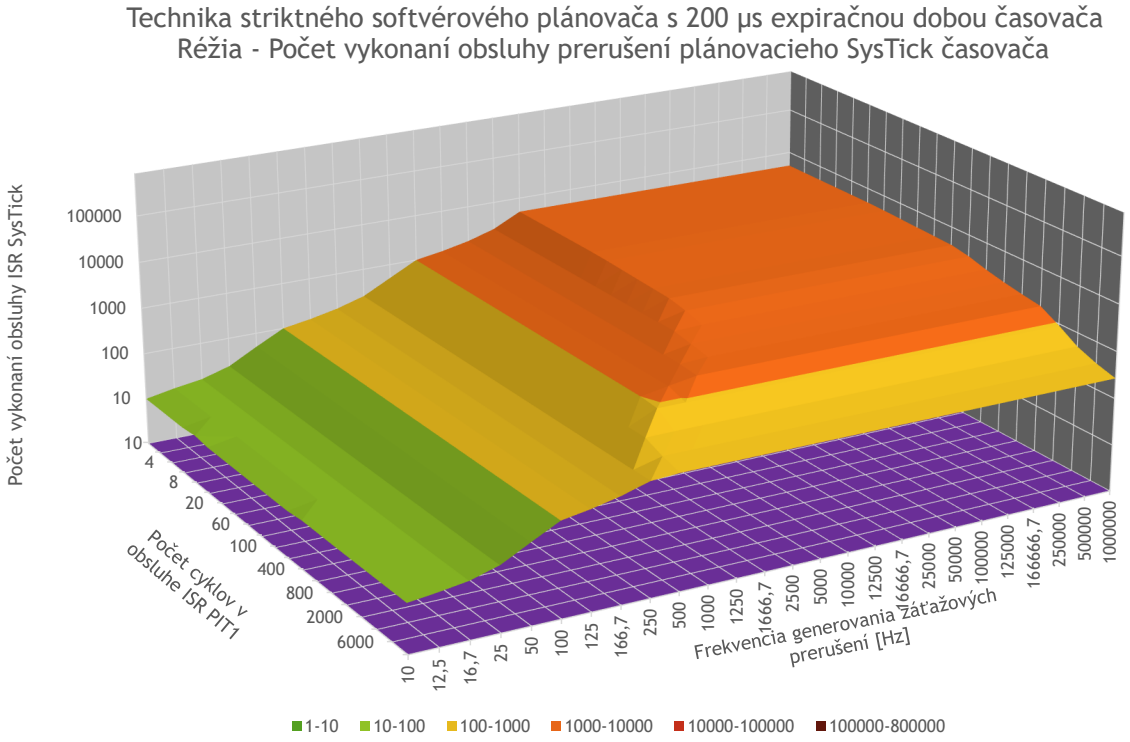


Obr. C.7: Počet obslúžených prerušení v závislosti od frekvencie generovania a dĺžky trvania ISR záťažového časovača PIT1 v logaritmickom merítku, pri menšom konfiguračnom kroku.

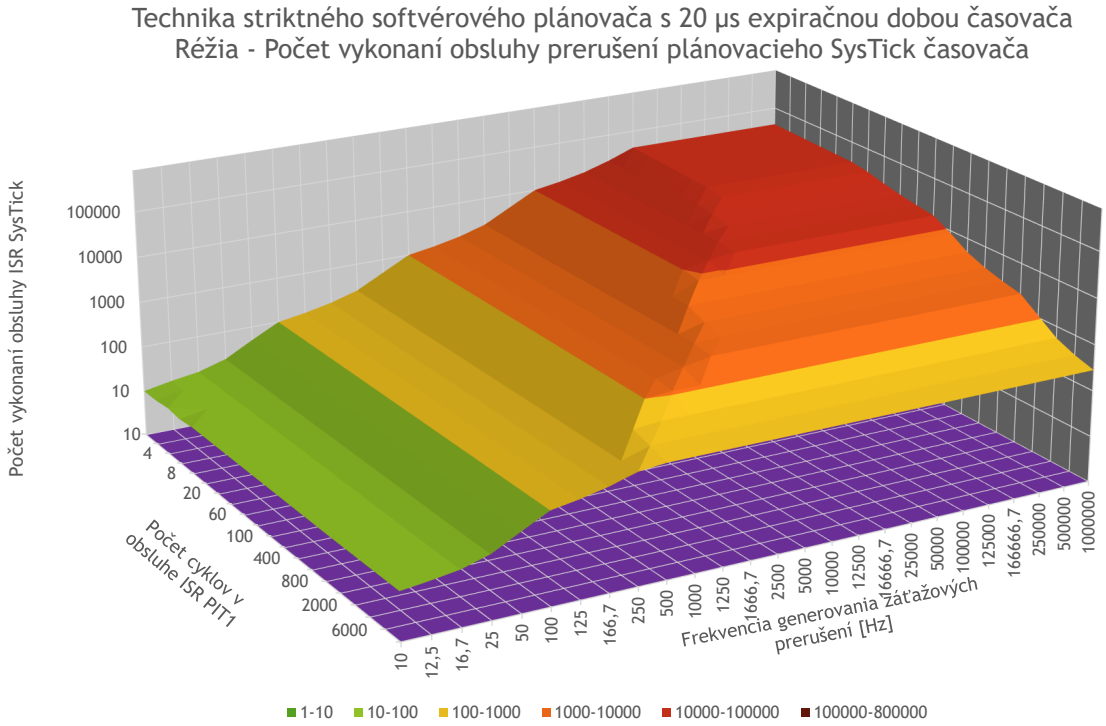
Príloha D

Réžia techniky striktného softvérového plánovača

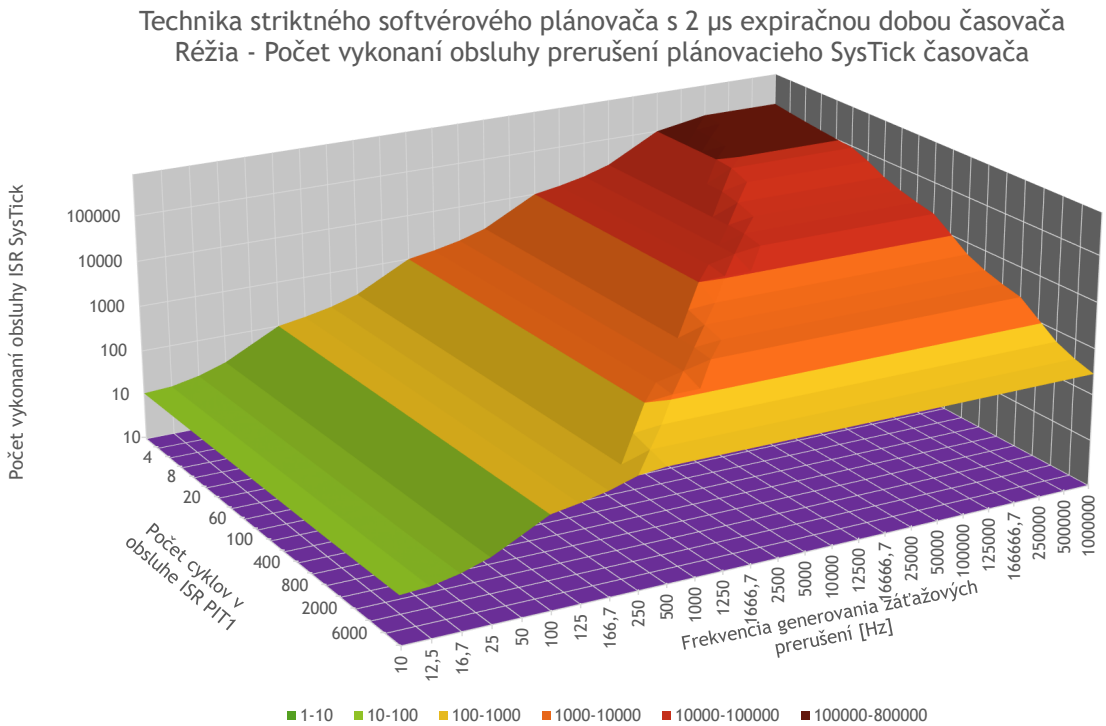
Grafy zobrazujúce počet vykonaných obslužných rutín jednorazového SysTick časovača, pre 200 μs, 20 μs a 2 μs, pri meraní réžie techniky striktného softvérového plánovača.



Obr. D.1: Počet obslúžených režijných prerušení SysTick pri technike striktného softvérového plánovača s 200 μs expiračnou dobou.



Obr. D.2: Počet obslúžených režijných prerušení SysTick pri technike striktného softvérového plánovača s 20 μs expiračnou dobou.

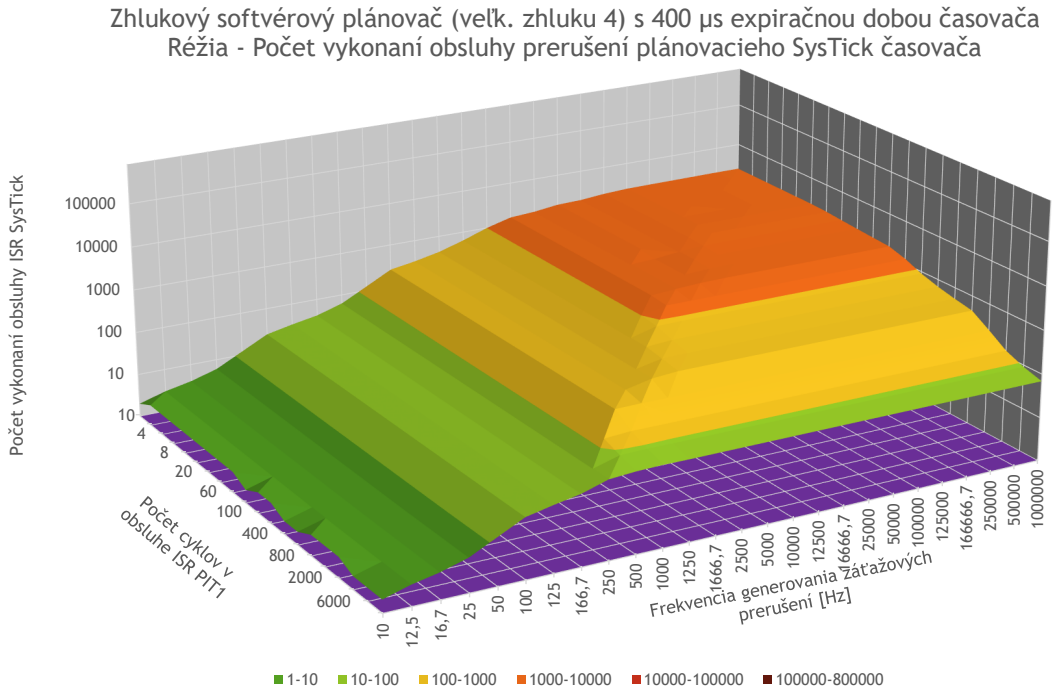


Obr. D.3: Počet obslúžených režijných prerušení SysTick pri technike striktného softvérového plánovača s 2 μs expiračnou dobou.

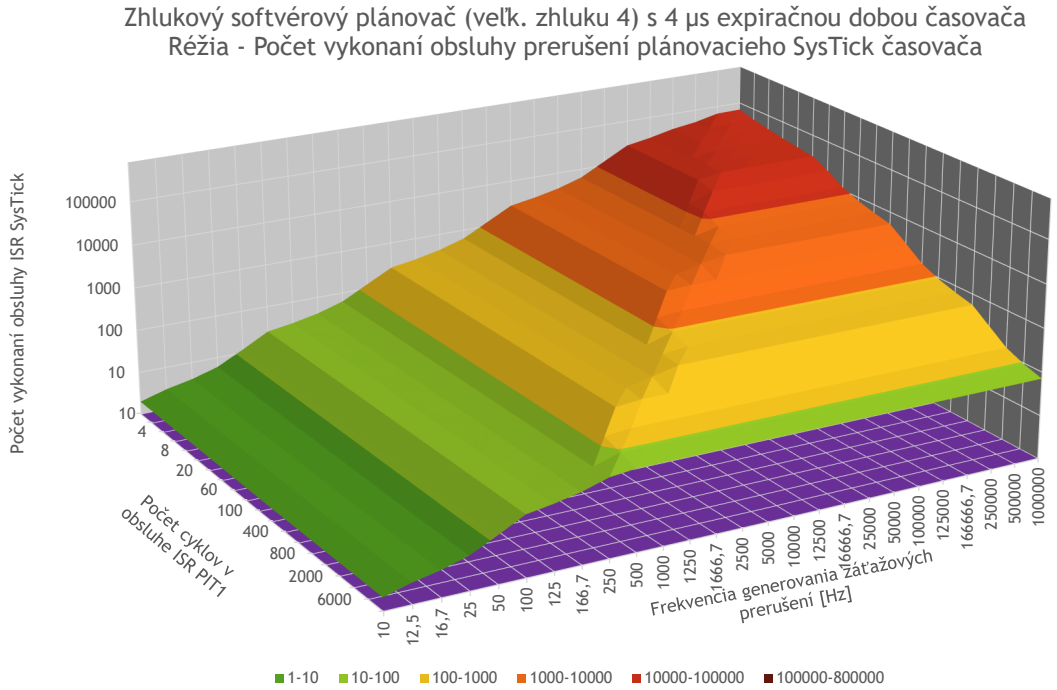
Príloha E

Réžia techniky zhlukového softvérového plánovača

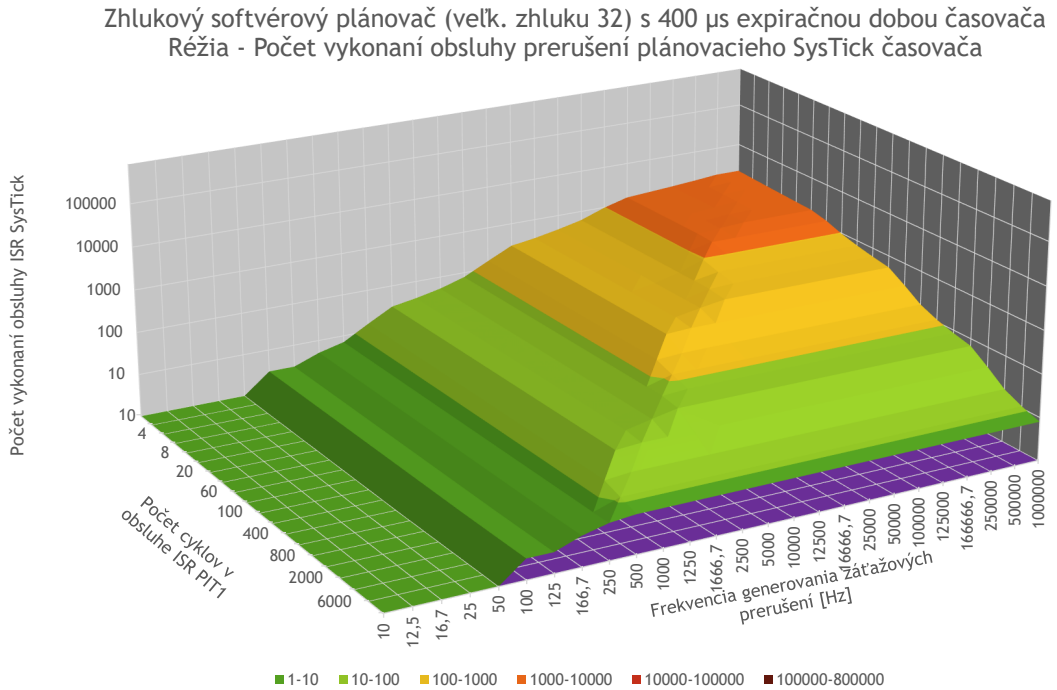
Grafy zobrazujúce počet vykonaných obslužných rutín jednorazového SysTick časovača, pre 400 μs a 4 μs, s maximálnou veľkosťou zhluku 4 a 32 prerušení, pri meraní réžie techniky zhlukového softvérového plánovača v logaritmickom merítku.



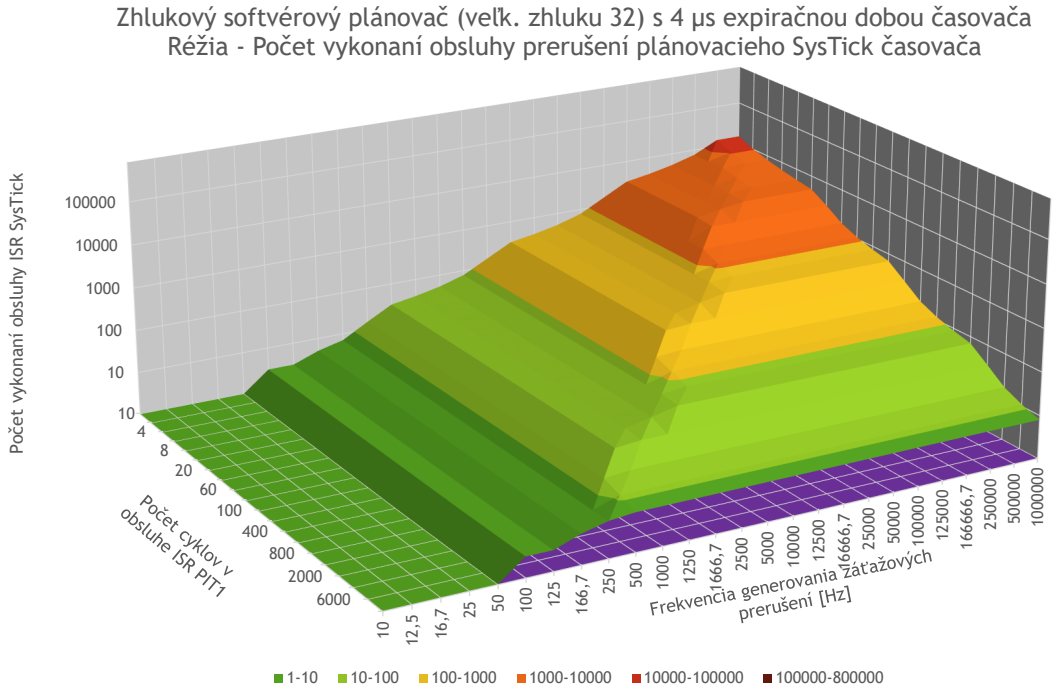
Obr. E.1: Počet obslúžených režijných prerušení SysTick časovača počas 1 s pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 400 μs expiračnou dobou SysTick časovača.



Obr. E.2: Počet obslúžených režijných prerušení SysTick časovača počas 1 s pri technike zhlukového softvérového plánovača s veľkosťou zhluku 4 a 4 μs expiračnou dobou SysTick časovača.



Obr. E.3: Počet obslúžených režijných prerušení SysTick časovača počas 1 s pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 400 μs expiračnou dobou SysTick časovača.



Obr. E.4: Počet obslúžených režijných prerušení SysTick časovača počas 1 s pri technike zhlukového softvérového plánovača s veľkosťou zhluku 32 a 4 μs expiračnou dobou SysTick časovača.

Príloha F

Porovnanie plánovacích techník

Nasledujúca tabuľka zobrazuje porovnanie efektívnosti plánovacích techník pre konkrétnu konfiguráciu PIT1 časovača. Časovač generuje záťažové prerušenia s frekvenciou 166 kHz a dĺžkou obslužnej rutiny 0 iterácií (95 taktov). Najlepšie výsledky pre danú konfiguráciu dosahuje technika zhlukového softvérového plánovača s veľkosťou zhluku 32.

Konfigurácia záťažových prerušení PIT1 166 kHz, 0 iterácií v ISR	Použitá technika		
	Striktný SW plánovač 800 μs	Zhlukový SW plánovač	
		veľkosť zhluku 4 4 ms	veľkosť zhluku 32 20 ms
Počet vykonaných prerušení PIT1	49 703	39 216	54 432
Počet taktov venovaných obsluhe ISR PIT1	3 827 131	2 901 982	3 980 339
Počet vykonaných iterácií v hlavnom cykle	2 547 204	2 892 145	2 816 220
Počet vykonaných ISR SysTick	49 116	9 803	1 700
Počet taktov venovaných obsluhe ISR SysTick	3 339 820	735 225	127 500

Tabuľka F.1: Porovnanie výsledkov implementovaných techník podľa počtu vykonaných ISR PIT1 pre frekvenciu generovania záťažových prerušení 166 kHz a 0 iterácií v tele obsluhy ISR.

Príloha G

Štatistiky vykonávaných meraní

Meranie:	**Čas merania:**	**Počet nameraných údajov:**
Experiment 0	(2x) 16 sec	64
Experiment 1	1 hod 9 min 20 sec	12 480
Experiment 2	8 min 40 sec	1 560
Experiment 2 (rozšírený počet)	12 min 34 sec	2 262
Technika striktného SW plánovača	1 hod 52 min 40 sec	20 280
Technika striktného SW plánovača - réžia	1 hod 52 min 40 sec	13 520
Technika zhlukového SW plánovača	3 hod 28 min	37 440
Technika zhlukového SW plánovača - réžia	3 hod 28 min	24 960

Tabuľka G.1: Čas vykonávaných meraní a počet nameraných výsledkov na platforme FIT-kit 3.0.