České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Hofman Martin

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: Implementace časem řízeného plánovače v distribuovaném bezpečnostně kritickém řídicím systému

Pokyny pro vypracování:

Spolehlivost a bezpečnost řídicích systémů vozidel a průmyslových zařízení vyžaduje nasazení deterministických komunikačních řešení a certifikovaných procesorových systémů, jako je obvod Texas Instruments TMS570LS1227ZWT. Úkolem práce je navrhnout řešení s hardwarově realizovaným rozvrhem vykonávání úloh na procesoru a přenosu zpráv po sběrnici CAN.

1) Seznamte se s procesorovým systémem dodaným průmyslovým partnerem. Jeho základem je obvod TMS570LS1227ZWT na bázi jádra Cortex-R4.
2) Navrhněte rozšíření knihoven RPP tak, aby bylo možné realizovat časování komunikace a úloh s využitím periferií časovače a CAN kontroléru nezávisle na hlavním procesoru.
3) Vyřešte synchronizaci rozvrhů mezi více jednotkami pro případ, kdy je jedna jednotka nadřazená. Při testování řešení se zaměřte na přesnost a spolehlivost časování a proveďte rozbor vlivu odchylek frekvencí oscilátorů jednotek.
4) Podrobně zdokumentujte navržené a otestované řešení a veškerá provedená měření a jejich výsledky.

Seznam odborné literatury:

[1] Jenkin, C.- Sojka, M. : Code generation for automotive rapid prototyping platform using Matlab/Simulink, RPP project documentation, ČVUT FEL, 2014
[2] Jeřábek, M.: FPGA Based CAN Bus Channels Mutual Latency Tester and Evaluation, diploma theses, ČVUT FEL, 2016
[3] TMS570LS12x/11x 16/32-Bit RISC Flash Microcontroller, Technical Reference Manual, Texas Instruments 2015

Vedoucí: Ing. Pavel Píša, Ph.D.

Platnost zadání do konce letního semestru 2017/2018

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

L.S.

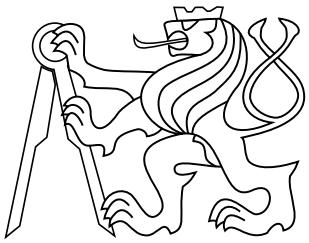
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 6.1.2017

diplomová práce

Implementace časem řízeného plánovače v distribuovaném bezpečnostně kritickém řídicím systému

Martin Hofman



Květen 2017

Ing. Pavel Píša Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra počítačů

Poděkování

Na tomto místě bych rád poděkoval vedoucímu diplomové práce Ing. Pavlu Píšovi, Ph.D. za odborné vedení, za pomoc a rady při zpracování této práce. Dále bych rád poděkovat Ing. Anně Minaevě za teoretické vedení práce. Dále bych chtěl poděkovat Ing. Michalu Sojkovi, Ph.D. za pomoc při tvorbě práce. Poté bych chtěl poděkovat všem z místnosti KN:G-203, kteří mi během práce pomohli. Dále bych rád poděkoval své rodině za podporu během celého studia. Nakonec bych chtěl poděkovat spolužáku za rady v průběhu studia.

Prohlášení

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

Abstrakt

Práce předkládá mechanismus pro přesné časování odesílaných zpráv na sběrnici CAN bez závislosti na hlavním procesoru a spouštění úloh na hlavním procesoru podle hardwarem stanoveného rozvrhu. V práci je využit obvod TMS570LS1227ZWT na bázi jádra Cortex-R4. V rámci práce jsou nejprve rozebrány jednotlivé periferie, které jsou vhodné k řešení daného úkolu. Poté je popsán návrh implementace, výběr a propojení periferií, které umožňují problém řešit. Vybrány byly periferie DCAN, DMA, VIM a N2HET. Dále práce popisuje způsob integrace do prostředí pro vývoj rychlý vývoj aplikací z prostředí Matlab/Simulink, založené na knihovně RPP, vyvíjené pro průmyslové partnery na na Katedře řídící techniky. Dále je popsané navržené aplikační rozhraní. Nakonec jsou uvedena měření a ověření vlastností implementované funkcionality.

Klíčová slova

TMS570LS1227ZWT; CAN; N2HET; Hardware obsluha CAN, Rozvrh, TMS570, Hercules, Texas Instruments

Abstrakt

This thesis presents mechnism for accurate timming of transmission of CAN messages without dependecy on main processor and hardware coordinated scheduling of CPU tasks. The chosen processor is TMS570LS1227ZWT based on the ARM Cortex-R4 architecture. This thesis analyze appropriate peripherals which can be combined to implement the project goals. The design and configuration with the most suitable peripherals selected (DCAN, DMA, VIM and N2HET) is described. Then the thesis describes integration into rapind prototyping platform providing support for application design in Matlab/Simulink environment for the target TMS570 platform. The environment is based on RPP library developed at CTU in Prague. Extension of the application interface is documented in the text. Finally, are listed the results of measurement and tests of added functionality.

Keywords

TMS570LS1227ZWT; CAN; N2HET; Hardware CAN, Schedule, TMS570, Hercules, Texas Instruments

Obsah

1 Úvod **1**

2 Analýza **3**
2.1 Úvod . . . 3
2.2 Možnosti řešení . . . 3
2.2.1 HW řešení . . . 3
2.2.2 SW řešení . . . 4
2.2.3 Kombinace HW+SW řešení . . . 4
2.3 Řešení vhodné pro platformu TMS570 . . . 4
2.3.1 Programovatelný časovač (High-End Timer (N2HET) Module) . . . 4
2.3.2 Řadič sběrnice CAN (Controller Area Network (DCAN) Module) . . . 6
2.3.3 Řadič přímého přístupu do paměti (Direct Memory Access (DMA) Module) . . . 7
2.3.4 Přenosová jednotka programovatelného časovače (High-End Timer Transfer Unit (HTU) Module) . . . 8
2.3.5 Vektorový správce přerušení (Vectored interrupt manager (VIM) Module) . . . 9
2.3.6 Free-Rtos . . . 10
2.3.7 Platforma pro rychlý modelově založený vývoj aplikací (Rapid Prototyping Platform (RPP)) . . . 11

3 Návrh **13**
3.1 Odesílání zpráv . . . 13
3.1.1 Odesílání jedné zprávy . . . 13
3.1.2 Odesílání více zpráv . . . 14
3.1.3 Režimy odesílání dat . . . 15
3.2 Oddělení odesílání zpráv od vykonávání rozvrhu . . . 16
3.3 Příjem zpráv . . . 17
3.3.1 Přijetí zprávy . . . 17
3.3.2 Přístup ke zprávám . . . 18
3.4 Synchronizace . . . 18
3.4.1 Master . . . 19
3.4.2 Slave . . . 19
3.5 Spouštění úloh . . . 22

4 Implementace **23**
4.1 Popis aplikačního rozhraní (API) . . . 23
4.1.1 Inicializace . . . 23
4.1.2 Změna dat . . . 26
4.1.3 Čtení přijatých zpráv . . . 26
4.1.4 Spouštění úloh . . . 26
4.1.5 Příklad konfigurace . . . 27
4.2 Integrace a změny rpp knihovny . . . 28
4.2.1 Přidané soubory . . . 28
4.2.2 N2HET program . . . 29
4.2.3 Změněné soubory . . . 29

5 Měření **31**
5.1 Měření na osciloskopu 31
5.1.1 Mód - data in schedule 31
5.1.2 Mód - data in message RAM 33
5.1.3 Srovnání přenosu DMA a DCANU 35
5.2 Měření na Xilinx Zynq 35
5.2.1 Odesílání zpráv 35
5.2.2 Měření příjmu zpráv 37
5.2.3 Měření přerušení 42
5.2.4 Měření synchronizace 43

6 Závěr **55**

Přílohy

A Obsah přiloženého CD **57**

B N2HET program **59**
B.1 Timer-WaitToSynchrMessageInSchedule.het 59
B.2 Timer-WaitToSynchPerm.het 61

Literatura **63**

Seznam obrázků

1 Zpřístupnění vykonávanému programu na NHETu (převzato z [3]) 4
2 Mapování požadavků z N2HETu na DMA a HTU(převzato z [3]) 5
3 Blokové schéma N2HETu(převzato z [3]) 6
4 Granularita DMA přenosu(převzato z [3]) 7
5 Blokové schéma VIM(převzato z [3]) 9
6 Architektura RPP(převzato z [4]) 11

7 Odeslaní jedné zprávy pomocí HTU 13
8 Odeslaní jedné zprávy pomocí DMA 14
9 Odeslaní více zpráv pomocí DMA 15
10 Odeslaní více zpráv pomocí DMA 16
11 Vykonávání rozvrhu 17
12 Příjem zpráv 17
13 Zpřístupnění zpráv 18
14 Rozvrh se synchronizací 19
15 Synchronizace slave jednotky 21
16 Spuštění úloh 22

17 Zapojení pro měření 31
18 Data in schedule - odeslání zprávy 32
19 Data in schedule - odeslání zprávy se zatížením 32
20 Data in message RAM - odeslání zprávy 33
21 Data in message RAM - odeslání zprávy se zatížením 34
22 DMA a DCAN přenos 35
23 Měření nepřesnosti odesílání zpráv 36
24 Měření nepřesnosti odesílání zpráv - při změnách teploty 36
25 Krátké měření 37
26 Meření příjmu zpráv 38
27 Odchylka 38
28 Doba reakce smyčky - různa data 39
29 Doba reakce smyčky - neměnící se data 40
30 Doba reakce smyčky - stejná data ve všech zprávách 41
31 Doba reakce smyčky - stejná data ve všech zprávách, se stejným ID 42
32 Čas spuštění přerušení 43
33 Hardwarová synchronizace 44
34 Hardwarová synchronizace po jednotlivých periodách 45
35 Hardwarová synchronizace po zahřátí master a slave jednotky 46
36 Hardwarová synchronizace po jednotlivých periodách při zahřátí 46
37 Synchronizace za využití přerušení 48
38 Synchronizace za využití přerušení po jednotlivých periodách 48
39 Synchronizace za využití přerušení při zahřátí 49
40 Synchronizace za využití přerušení při zahřátí po periodách 50
41 Synchronizace za využití přerušení a škálování 51
42 Synchronizace za využití přerušení a škálování při zahřátí po periodách . . . 52

Seznam tabulek

1 Registry DCANu 6

2 Prvky struktury cantt_config 24
3 Prvky struktury cantt_message 25
4 Prvky struktury cantt_update_message 26

5 Data in schedule výsledky měření 33
6 Data in schedule výsledky měření 34
7 Zprávy v grafu 35
8 Mininimální a maximální odchylka 39
9 Mininimální a maximální odchylka - různa data 39
10 Mininimální a maximální odchylka - neměnící se data 40
11 Mininimální a maximální odchylka - stejná data ve všech zprávách 41
12 Rozvrh - jedna synchronizační zpráva 44
13 Srovnání doby přijetí - hardwarová synchronizace 45
14 Srovnání doby přijetí - hardwarová synchronizace po zahřátí 47
15 Rozvrh - dvě synchronizační zprávy 47
16 Srovnání doby přijetí 49
17 Srovnání doby přijetí - zahřátí 50
18 Srovnání doby přijetí - škálování 51
19 Srovnání doby přijetí - škálování, zakřátí 52
20 Dlouhý rozvh 53
21 Statistiky vzdálednosti mezi zprávami 53

Zkratky

API	Application Programming Interface - Aplikační rozhraní
ARB	Arbitration
CMD	Command
DCAN	Controller Area Network - Řadič sběrnice CAN
DISS	Data in schedule slot - Data v položce rozvrhu
DMA	Direct memory access - Přímý přístup do paměti
FIQ	Fast Interrupt Requests - Rychlý požadavek na přerušení
HET	High-End timer - Programovatelný časovač
HTU	High-End Timer Transfer Unit - Přenosová jednotka programovatelného časovače
IRQ	Interrupt ReQuest - Požadavek na přerušení
MCTL	Message Controll
RAM	Random memory access - Paměť s přímým přístupem
RPP	Rapid prototyping platform - Platforma pro rychlý modelově založený vývoj aplikací
TTCAN	Time Triggered CAN - Časem spuštěný CAN
VIM	Vectored interrupt mannager - Vektorový správce přerušení

1 Úvod

Nároky na vestavěné systémy neustále rostou s rostoucími požadavky zákazníků. Konkrétně v automobilovém průmyslu se jedná o stále větší množství propojených jednotek, na kterých běží stovky až tisícovky většinou periodických úloh. Ty mezi sebou musí vyměňovat data podle pevně stanoveného rozvrhu. Na rozvrh jsou kladeny striktní požadavky tak, aby byla splněna časová omezení. Teoretickými základy a návrhem rozvrhu se zabývá předchozí část projektu [1] a tato práce navazuje praktickou implementací vykonávání takového rozvrhu.

Vzhledem k požadavkům na spolehlivost a bezpečnost řídicích systémů vozidel a průmyslových zařízení je nutné nasazení deterministických komunikačních řešení a certifikovaných procesorových systémů, jako je obvod Texas Instruments TMS570LS1227ZWT založený na bázi jádra Cortex-R4. V rámci této práce byla využita deska od firmy EATON založená na výše zmíněném obvodu.

Cílem této práce je navrhnout nové řešení s hardwarově realizovaným rozvrhem přenosů zpráv po sběrnici CAN a pro spouštění úloh na procesoru, jež rozšíří již navržené prostředí pro vývoj aplikací na dané rodině obvodů. Prostředí již zahrnuje podporu pro tvorbu aplikací z prostředí Matlab/Simulink. Základem prostředí Rapid prototyping platform je knihovna RPP.

Práce je rozdělena do čtyř částí zabývajících se analýzou, návrhem, implementací a měřením/ověřením implementované funkcionality.

V první analytické části práce jsou krátce rozebrány alternativy k vybranému řešení. Dále jsou v této části rozebrané jednotlivé periferie vybraného obvodu. Poté je popsána RPP knihovna a využitý operační systém Free-Rtos.

V druhé části zaobírající se návrhem, jsou detailně rozebrány principy fungování odesílání zpráv, příjmu zpráv, synchronizace a spouštění funkcí. Představené řešení využívá periferií DCAN, DMA, VIM a N2HET. Odesílání zpráv podle stanového rozvrhu je řešeno plně hardwarově za pomocí periferií a cyklicky rozvrh realizuje bez zásahu hlavního procesoru. Příjem zpráv je opět plně řešen v režii periferií. Synchronizace je realizovaná využitím synchronizačních zpráv, určujících jaká část rozvrhu se má na podřízené jednotce spustit. Přepínání jednotlivých částí rozvrhu je řešeno v rámci přerušení, kdy je potřeba spolupráce CPU. Pokud je tedy v rámci rozvrhu na nadřízené jednotce odesílána pouze jedna synchronizační zpráva je možné realizovat synchronizaci plně hardwarově bez přerušení. Spuštění úloh zajišťuje časovač N2HET spouštějící přerušení.

Ve třetí části práce je popsána implementace. V rámci popisu implementace je popsáno výsledné API výsledné knihovny a dále je popsán způsob integrace do RPP knihovny.

Ve čtvrté části jsou provedena měření a ověření implementované funkcionality. V rámci měření je nejprve rozebráno odesílání zpráv a příjem zpráv. Poté je probráno měření synchronizace dvou jednotek.

2 Analýza

2.1 Úvod

V rámci článku [1] jsou rozebrány teoretické základy a návrh rozvrhu spouštějícího úlohy a odesílajícího zprávy na více jednotkách. Cílem práce je na tento článek navázat praktickou realizací umožňující vykonávání takového rozvrhu za využití periferií obvodu TMS570LS1227ZWT. V rámci implementace lze vycházet ze standardu TTCAN (Time Triggered CAN) popsaném v normě ISO 11898-4 [2]. V TTCANu je rozvrh v rámci hyperporiody rozdělen na jednotlivé cykly. Tyto cykly jsou spouštěny na základě přenosu synchronizačních zpráv, jež obsahují informaci jaký cyklus se má spustit. Požadovaná granularita rozvrhu je 1 µs.

2.2 Možnosti řešení

Cíle práce je možné řešit několika způsoby. V této podkapitole jsou uvedeny možnosti řešení jako ukázka možných přístupů. Problém lze řešit buď plně hardwarově, softwarově nebo kombinací obou přístupů.

2.2.1 HW řešení

FPGA

Jedním z možných hardwarových řešení řešení je použít programovatelné hradlové pole. Jedná se integrovaný obvod, jehož bloky jsou tvořeny konfigurovatelnou maticí spojů a funkcionalita je konfigurována uživatelem. V našem případě je možné použít FPGA dvěma způsoby:

i Vytvořit celý CAN řadič v FPGA včetně stavového automatu řešícího časování.
ii Použít stávajícího řadiče CAN a programovatelný obvod využít k řízení řadiče a přenosu zpráv do řadiče

Výhody a nevýhody řešení:

\+ Přesnost v čase
\+ Flexibilita
– Cena
– Nutnost certifikace

Využití stávajícího řadiče

Pro realizaci je možné použít již existující řadič (např. integrovaný na mikrocontroléru) a realizovat časování zpráv s využitím dostupných hardwarových prostředků (DMA, časovače, atd.). Výhody a nevýhody řešení:

\+ Certifikované čipy
\+ Není nutné navrhovat HW
\+ Cena
– Na jeden čip (nepřenositelné)

2.2.2 SW řešení

Softwarově lze vysílání zpráv podle rozvrhu řešit 2 způsoby:

i Spuštění odeslání zprávy v Interruptu (Coretex-R umožňuje pro rychlejší odezvu použít místo IRQ FIQ)
ii Na úrovni tasku OS

Výhody a nevýhody řešení:

– Nejnižší přesnost v čase
– Nutný zásah procesoru
– Přerušení zhorší časování úloh vykonávaných na CPU

2.2.3 Kombinace HW+SW řešení

Další možností kombinace hardwarového a softwarového řešení. Vysílání zpráv podle rozvrhu lze řešit tím způsobem, že každou zprávu připraví procesor a o odeslání se postará časovač nebo jiná část integrovaného obvodu.

+ Certifikované čipy
+ Cena
– Na jeden čip (nepřenositelné)
– Nutný zásah procesoru

2.3 Řešení vhodné pro platformu TMS570

Protože práce navazuje na projekt [1] byla vybrána platforma daná požadavky zadavatele. Cílová platforma byla deska od firmy EATON, která je využívá obvod TMS570LS1227ZWT založen na architektuře ARM Cortex R4. Jedná se procesor speciálně navržený se ohledem na bezpečnost (dvě jádra vykonávající stejný program - jedno se zpožděním, mechanismus pro korekci chyb paměti pro SRAM a Flash, monitorování napětí,...). V rámci řešení je možné použít periferie popsané níže.

2.3.1 Programovatelný časovač (High-End Timer (N2HET) Module)

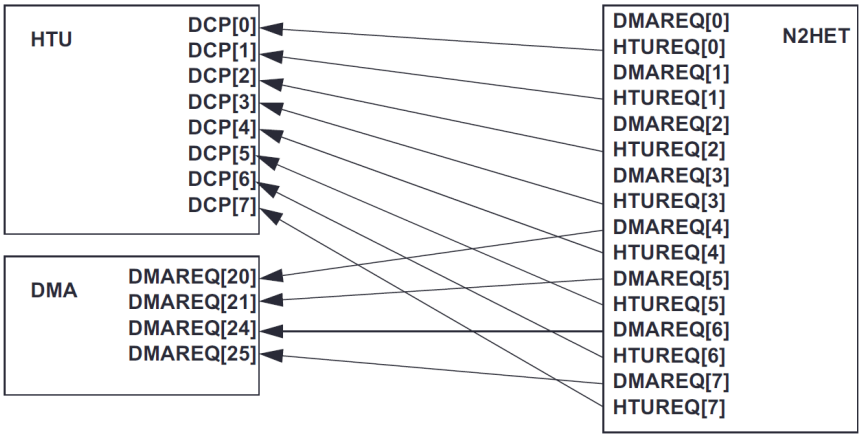
Jedná se o programovatelný časovač. Tento časovač je řízený programem sestávající se ze sekvence instrukcí. Modul rozlišuje 30 různých instrukcí. Každá instrukce je tvořena třemi částmi programovou, kontrolní a datovou. Program je spouštěn cyklicky od instrukce na drese 0. Cykly jsou spouštěny s periodou (LRP - Loop resolution period). Časovač dále umožňuje vysílat požadavky do jednotek HTU a DMA. Volitelně může vykonávání instrukce aktivovat přerušení na procesor.

N2HET Address	Host CPU or DMA Address Space				
Instruction	Program Field Address	Control Field Address	Data Field Address	Reserved Address	N2HET RAM Bank
000h	XX0000h	XX0004h	XX0008h	XX000Ch	A
001h	XX0010h	XX0014h	XX0018h	XX001Ch	B
002h	XX0020h	XX0024h	XX0028h	XX002Ch	C
003h	XX0030h	XX0034h	XX0038h	XX003Ch	D
004h	XX0040h	XX0044h	XX0048h	XX004Ch	A
:	:	:	:	:	:
03Fh	XX03F0h	XX03F4h	XX03F8h	XX03FCh	D
040h	XX0400h	XX0404h	XX0408h	XX040Ch	A
:	:	:	:	:	:
1FFh	XX1FF0h	XX1FF4h	XX1FF8h	XX1FFCh	D

Obrázek 1 Zpřístupnění vykonávanému programu na NHETu (převzato z [3])

Na obrázku 1 je zobrazeno mapování zprostředkující přístup k jednotlivým instrukcím a jejich položkám uloženým na RAM. Díky tomu je možné, jak přistupovat vykonávanému programu, tak měnit jeho vykonávání za pomoci CPU, DMA a HTU.

Vykonávání programu začíná vždy na první instrukci a je vykonáno v cyklech vždy končících opět na první instrukci. Program není striktně sekvenční, ale každá instrukce obsahuje položku jež určuje následující instrukci a některé instrukce obsahují i alternativní adresu instrukce, na kterou se přejde v případě splnění určité podmínky. Cyklus je vždy spouštěn jednou za periodu LRP a je tedy nutné, aby byl program vždy vykonán za kratší dobu, než je perioda cyklu. Frekvence vykovávání instrukcí je dána vydělením vstupní frekvence hodnotou děličky HR (High Resolution) a čas LRP je určený vydělením této frekvence hodnotou děličky LR (Loop Resolution).

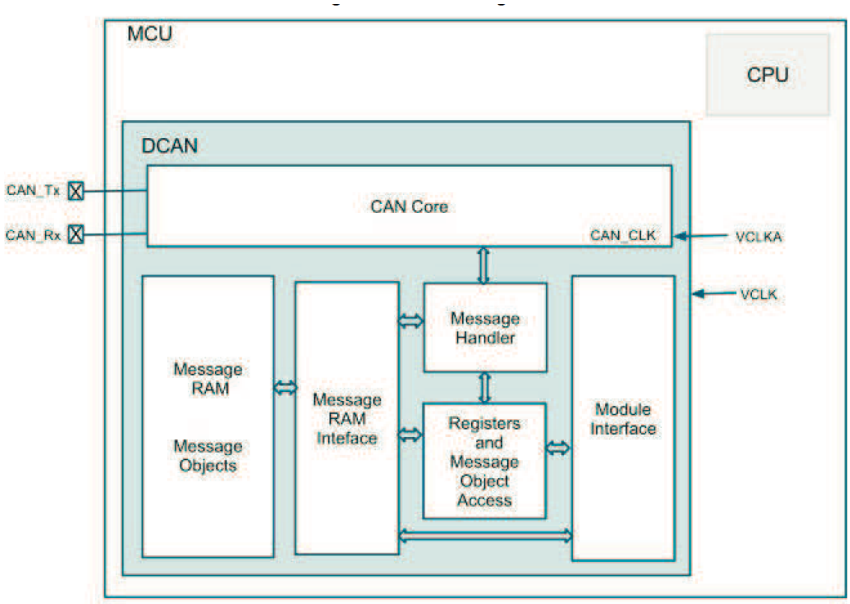


Obrázek 2 Mapování požadavků z N2HETu na DMA a HTU(převzato z [3])

N2HET umožňuje jak je znázorněno na obrázku 2 vysílat požadavky na jak na DMA tak na HTU. S tím že 4 kanály jsou vyhrazeny výhradně pro HTU a 4 kanály lze použít buď pro HTU nebo DMA.

2.3.2 Řadič sběrnice CAN (Controller Area Network (DCAN) Module)

Jedná se řadič zajišťující sériovou multimaster komunikaci podle standardu CAN-2.0B (ISO 11898). Použitý protokol využívá Non Return To Zero (NRZ) s bit stuffingem (vkládání bitu/bitů). Vysílané zprávy obsahují ID (11 nebo 29 bitů) a data (až 64 bitů). Vyslaná zpráva nemá stanovený cíl, takže filtrování zpráv probíhá na základě ID. Na následujícím obrázku 3 je znázorněno blokové schéma kontroléru.



Obrázek 3 Blokové schéma N2HETu(převzato z [3])

Integrovaná paměť RAM slouží k uložení až 64 zpráv z hlediska komunikace CAN se jedná o "message objekty". Přístup ke zprávám je řešený přes tři sady přístupových registrů. Data a identifikace zprávy jsou na základě zápisu požadavku přeneseny do/z přístupových registrů v nepřerušitelném cyklu čímž je zaručena konzistence. Pro manipulaci se zprávami a jejich čtení zprávami slouží 1 a 2 sada registrů. Třetí sada registrů slouží pouze k příjmu zpráv. K odeslání zprávy je nutné zapsat do následujících registrů:

Registr	Velikost[B]	Obsah registru
CMD	4	Číslo message objektu, bitové pole pro určení, které položky budou přeneseny z/do message objektu (např. data, ARB, MCTL), směr přenosu, požadavek na odeslání zprávy
ARB	4	ID zprávy
MCTL	4	Délka zprávy
DATA	8	Data zprávy

Tabulka 1 Registry DCANu

Registry jsou v paměti uspořádané ve stejném pořadí jako v tabulce 1, tedy v pořadí CMD, ARB, MCTL, DATA. Pro odeslání zprávy je nutné zapsat příkaz do CMD registru, který následně provede přenos do message objektu, kde nastaví požadavek na odeslání. Nevýhodou

tohoto uspořádaní je, že registr CMD je na nejnižší adrese a není možné do něj zapsat spolu s ARB, MCTL a DATA jedním přenosem DMA.

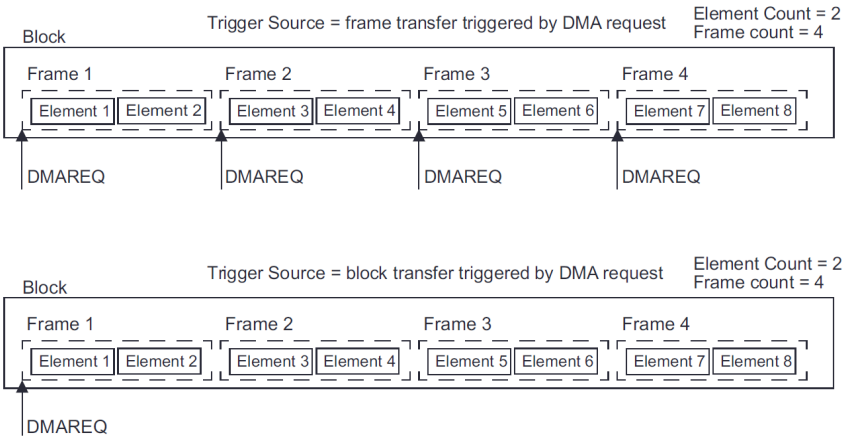
Rozhraní 3 je pouze pro čtení a je speciálně přizpůsobený pro spolupráci s DMA. Spolupráce s DMA probíhá tím způsobem, že je možné nakonfigurovat, jaký DMA kanál se má po přijetí zprávy spustit a dále které registry musí DMA přečíst než je povolené přepsat zprávu v rozhraní novou příchozí zprávou.

2.3.3 Řadič přímého přístupu do paměti (Direct Memory Access (DMA) Module)

DMA složí k přenosu dat v adresním prostoru mikrokontroléru. DMA umožňuje tři úrovně granularity:

i **Elementy** - Základní jednotka přenosu o velikosti 8, 16, 32 nebo 64 bitů. Přenos nelze přerušit
ii **Framy** - Jeden nebo více elementů přenesených v nedělitelné sekvenci
iii **Bloky** - Jeden nebo více framů přenesených, buď ma základě jednotlivých požadavků nebo v sekvenci na základě jednoho požadavku

Přenos může být realizován po framech nebo blocích. Je na znázorněno na obrázku 4 kdy každý request přenese buď jeden frame nebo celý blok.



Obrázek 4 Granularita DMA přenosu(převzato z [3])

Adresovat jednotlivé framy a elementy je možné ve 3 režimech:

i **Constant** - cílová a/nebo zdrojová adresa se nemění
ii **Post increment** - cílová a/nebo zdrojová adresa je post-inkrementována velikostí elementu
iii **Indexed** - cílová a/nebo zdrojová adresa je post-inkrementována hodnotou v registrech Element Index Offset Register a the Frame Index Offset Register

Přenos dat je realizován využitím kanálů. Každý kanál je popsán jedním řídím paketem (Channel Control Packet-CPP) z celkových šestnácti. Každý CPP je složen z devíti polí z nichž šest slouží k nastavení DMA a poslední tři slouží pouze pro čtení. Jednotlivá pole CPP jsou následující:

i **Initial Source Address** - počáteční zdrojová adresa určuje adresu, od které se zahájí přenos
ii **Initial Destination Address** - počáteční cílová adresa určuje počáteční adresu na níž bude zahájen přenos
iii **Initial Transfer Count** - obsahuje informace: počet elementů, počet framů

iv **Channel Configuration Word** - velikost čteného/zapisovaného elementu, typ přenosu (po blocích nebo framech), typ adresování zdroje/cíle, auto-inicializace (pokud je aktivována tak po přenesení nastaveného množství dat začne přenos znovu od začátku), jaký kanál bude spuštěn po dokončení přenosu

v **Element/Frame Index Pointer** - obsahuje informace o tom, jaký offset má být přičten ke zdrojovému/cílovému elementu/framu

vi **Current Source Address** - obsahuje aktuální zdrojovou adresu kanálu, hodnota je pouze pro čtení

vii **Current Destination Address** - obsahuje aktuální cílovou adresu kanálu, hodnota je pouze pro čtení

viii **Current Transfer Count** - obsahuje informace, o tom kolik elementů zbývá přenést v rámci bloku

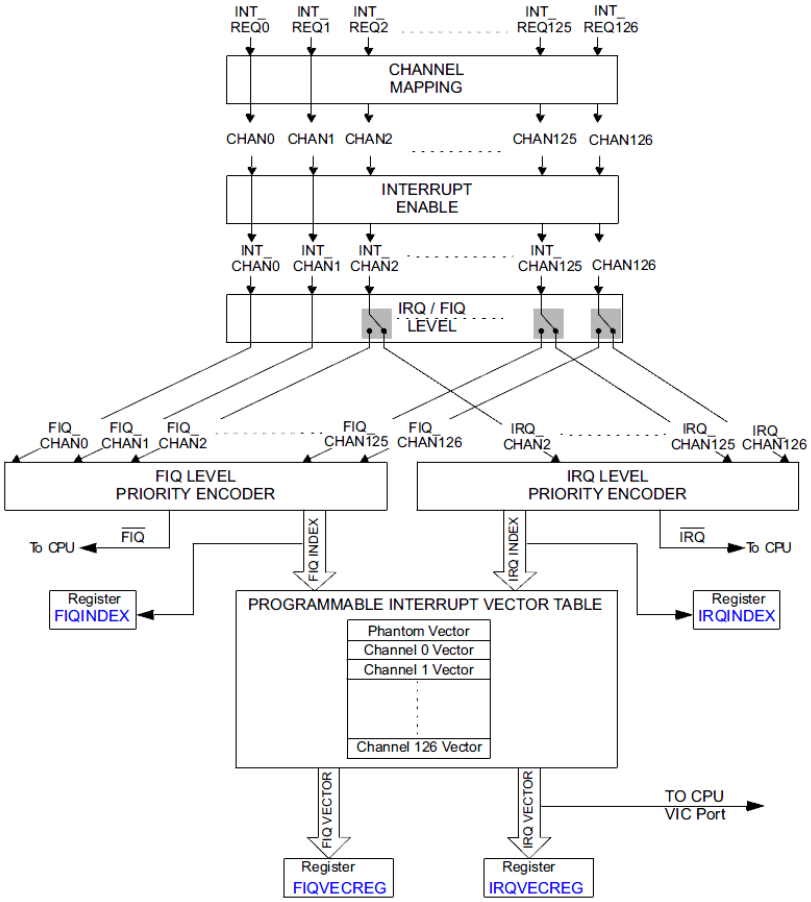
Pro každý kanál lze zvolit jeden ze zdrojů požadavků (requestů). Princip činnosti je takový, že po requestu na určitý řádek dojde k postupnému přenosu dat v rámci přiřazených kanálů. Při aktivaci více kanálů naráz jsou kanály postupně obslouženy podle priority od toho s nejnižším číslem po ten nejvyšším. V případě, že je na daný kanál navázán přenos jiného kanálu, je po ukončení přenosu tohoto kanálu spuštěn kanál navazující.

2.3.4 Přenosová jednotka programovatelného časovače (High-End Timer Transfer Unit (HTU) Module)

Jedná se o jednodušší verzi DMA přímo určenou pro přenos mezi pamětí N2HETu a hlavní pamětí. Umožňuje přenášet pouze souvislé bloky paměti v jednom nebo více requestech. Na každý request lze navázat pouze jeden přenos.

2.3.5 Vektorový správce přerušení (Vectored interrupt manager (VIM) Module)

Jedná se modul poskytující nástroje pro prioritizaci a kontrolu zdrojů přerušení. Na obrázku 5 je zobrazen blokový diagram VIMu.



Obrázek 5 Blokové schéma VIM(převzato z [3])

Jak je z obrázku 5 patrné tak zpracování požadavku na přerušení probíhá v několika krocích. Při přijetí požadavku na přerušení jako první dochází k mapování přijatého požadavku na kanál přerušení. Toto mapování určuje jaký kanál odpovídá přijatému požadavku. Poté dojde ke kontrole jestli je přerušení povoleno na daném kanálu. Dalším krokem je rozhodnutí jestli se jedná o IRQ (Interrupt ReQuest) nebo FIQ (Fast Interrupt Requests). Poté dojde k vybrání požadavku s nejvyšší prioritou nastavení registrů INDEX a VECREG. Nakonec dojde k odeslání požadavku na procesor.

2.3.6 Free-Rtos

Jedná se o real-time operační systém pro vestavěné systémy. Tento operační systém nabízí podporu pro 35 architektur. Převažující většina jeho kódu je implementována v jazyce C. V assembleru jsou napsány pouze platformě specifické funkce. Free-Rtos obsahuje API pro řízení a plánování úloh. API dále nabízí možnost použití mutexů, časovačů, front a dalších funkcí.

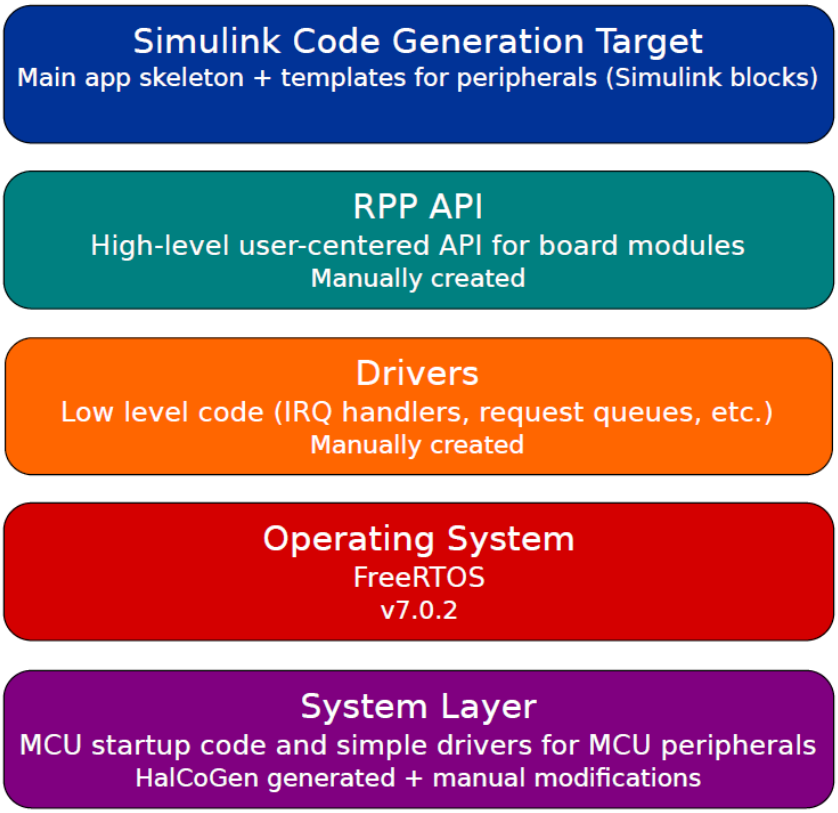
V rámci této práce je pro nás důležitá možnost synchronizace úlohy z přerušení s využitím semaforů. Systém pro tuto synchronizaci nabízí binární semafor. Na následujících řádcích je uveden příklad užití.

```
1  SemaphoreHandle_t xSemaphore = NULL;
2
3  void task( void * parameters )
4  {
5          xSemaphore = xSemaphoreCreateBinary();
6      for( ;; )
7      {
8          if( xSemaphoreTake( xSemaphore, LONG_TIME ) == pdTRUE )
9          {
10                 //Akce
11             ...
12         }
13     }
14 }
15
16 void isr( void * parameters )
17 {
18         static signed BaseType_t xHigherPriorityTaskWoken=pdFALSE;
19     xSemaphoreGiveFromISR( xSemaphore, &xHigherPriorityTaskWoken );
20     portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
21 }
```

Příklad obsahuje dvě procedury. První procedura task obsahuje inicializaci semaforu a následné čekání, až dojde k uvolnění semaforu, které umožní vykonat konkrétní akci. Druhá procedura v obslužné rutině přerušení uvolňuje semafor.

2.3.7 Platforma pro rychlý modelově založený vývoj aplikací (Rapid Prototyping Platform (RPP))

Jedná se knihovnu vyvíjenou na FEL ČVUT. Knihovna je vytvořena pro desku navrženou a vyráběnou firmou EATON založenou na architektuře ARM Cortex R4 TMS570LS1227ZWT. Knihovna slouží k obsluze periferií a je využívána v rámci prostředí pro generování kódu v jazyku C ze Simulinkových modelů. Knihovna je složena z několika vrstev, zobrazených na obrázku 6. Návrh vrstev je takový, že každá vrstva poskytuje rozhraní a volá pouze nižší vrstvu.



Obrázek 6 Architektura RPP(převzato z [4])

V rámci práce budou rozšířeny vrstvy RPP API, System a Drivers. Popis jednotlivých vrstev se nachází v následujícím seznamu.

i **System layer** - obsahuje soubory jež obsahují definice typů, hodin a mapování přerušení
ii **Operating System** - obsahuje Free Rtos
iii **Drivers** - obsahuje ovladače pro práci s periferiemi
iv **RPP API** - jedná se nejvyšší vrstvu knihovny obsahují sadu funkcí pro každou periferii
vi **Simulink Code Generation Target** - tato vrstva generuje ze Simulinkových modelů kód v jazyce C využívající RPP API

3 Návrh

3.1 Odesílání zpráv

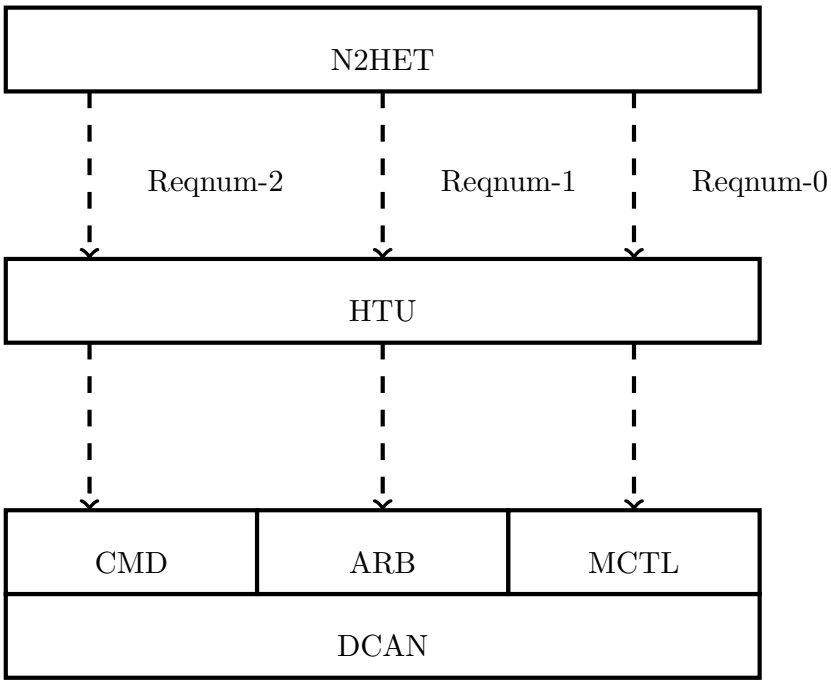
Vzhledem k nutnosti měřit přesně čas a na základě času spouštět další aktivity na platformě TMS570 je pro tyto účely předurčena periferie N2HEt. Další součástí byl CANový kontrolér pro realizaci komunikace po sběrnici CAN. Bylo nutné rozhodnout, jakými prostředky zajistit jejich vzájemnou spolupráci. Tento problém je podrobně rozebrán v následujících odstavcích.

3.1.1 Odesílání jedné zprávy

Prvním krokem v rámci odesílání zprávy bylo implementovat rozesílání jedné zprávy a to posléze rozšířit pro odesílání více zpráv.

Odesílání zprávy pomocí HTU

Jako první možnost se nabízelo využití HTU, protože se jedná o jednotku přímo určenou pro přenosy mezi N2HETem a hlavní pamětí.



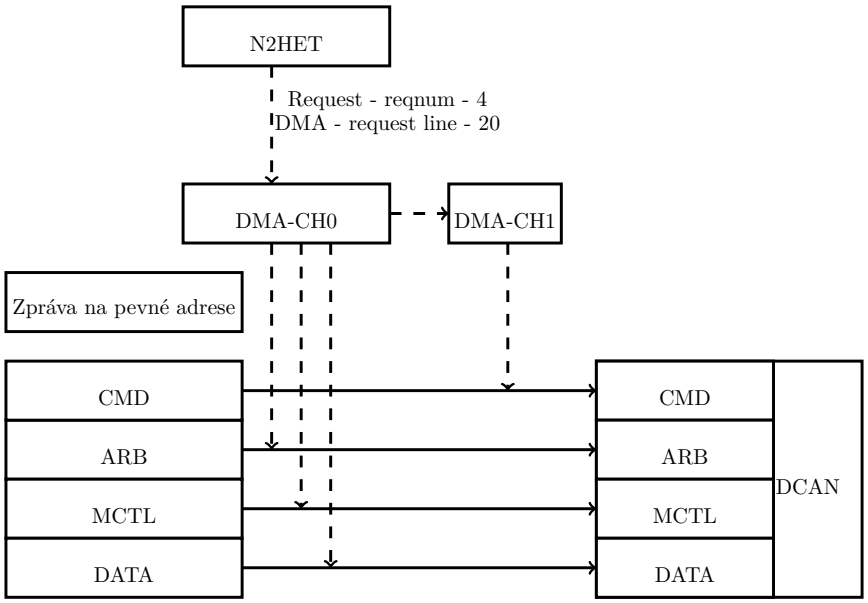
Obrázek 7 Odeslaní jedné zprávy pomocí HTU

Princip: Na obrázku 7 je znázorněn princip činnosti. Princip je takový že v N2HETu běží čítač, který v pevně stanoveném čase přeteče, čímž vyvolá tři požadavky na HTU. Tyto požadavky spustí přenos dat z N2HETu do DCANu konkrétně do registrů MCTL, ARB a CMD, čímž zahájí odeslání zprávy. Pro jednoduchost přenosů se nebral v úvahu přenos dat.

Zhodnocení řešení: Použití HTU bylo zavrženo, protože neumožňuje více přenosů v rámci jednoho požadavku. Dalším důvodem je možnost přenášet data pouze mezi N2HETem a hlavní pamětí.

Odesílání zprávy pomocí DMA

Další alternativa k HTU se nabízí DMA. Výhodou DMA je možnost řetězit přenosy a že data v rámci přenosu nemusí být přenášena přes N2HET. Toho je využito v následujícím řešení. Toto řešení obsahuje navíc jednu zprávu na pevné adrese v paměti jež obsahuje hodnoty registrů CMD, ARB, MCTL a DATA, tyto hodnoty jsou přeneseny pomocí 2 kanálů. Kanály jsou použity dva, protože registr CMD se nenachází v paměti přímo za ostatními registry a nebylo by možné do něj zapsat jedním přenosem posledního elementu v rámci bloku.



Obrázek 8 Odeslaní jedné zprávy pomocí DMA

Princip: Na obrázku 8 je znázorněn princip činnosti. V N2HETu běží čítač, který v pevně stanoveném čase přeteče, čímž vyvolá požadavek na DMA. DMA nejprve spustí kanál 0 a ten přenese hodnoty ARB, MCTL a DATA ze zprávy na pevné adrese v paměti do interface registru DCAN. Po dokončení přenosu se zahájí přenos kanálu 1, který je navázaný na kanál 0, tento kanál přenese hodnotu CMD do odpovídajícího registru DCANu a tím se zahájí přenos zprávy do message RAM a její následné odeslání.

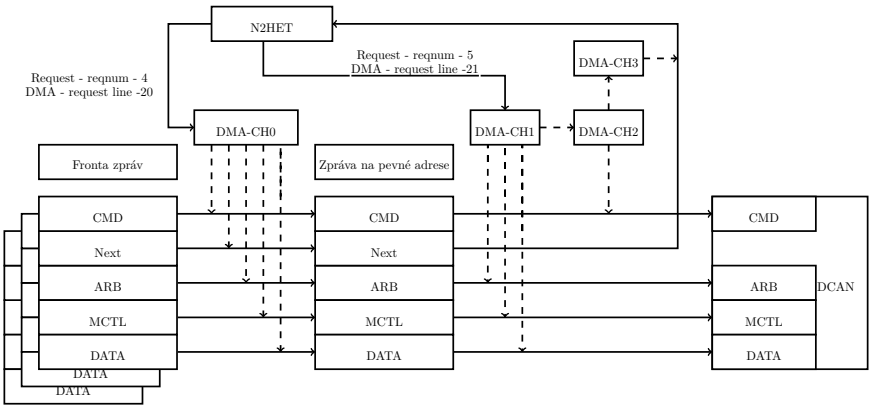
Zhodnocení řešení: Toto řešení bylo potvrzeno jako vhodná volba, neboť umožňuje odeslat zprávu jedním požadavkem. V následující části bude rozšířeno o možnost vysílání více zpráv.

3.1.2 Odesílání více zpráv

Dalším krokem po implementaci odesílání jedné zprávy bylo rozšíření mechanizmu odesílání zprávy o více zpráv.

Odesílání zprávy s využitím fronty zpráv

Jedná se o rozšíření předchozí verze o frontu zpráv určených k odeslání. Navíc je zde přidána zpětná vazba do N2HETu, která umožňuje měnit hodnotu, do které bude čítač čítat.



Obrázek 9 Odeslaní více zpráv pomocí DMA

Princip: Na obrázku 9 je zobrazen princip činnosti. Po přetečení čítače v N2HETu dojde k odeslání dvou požadavků na DMA. První požadavek spustí kanál 0, který přenese jednu zprávu z fronty zpráv na pevné místo v paměti. Druhý požadavek spustí přenos prvním kanálem, jenž přenese hodnoty registrů ARB, MCTL a DATA do interface registru DCANu. Poté se spustí druhý kanál, jenž je navázaný na první a ten provede přenos hodnoty do CMD registru, tím se zahájí přenos zprávy do message RAM DCANu a její následné odeslání. Nakonec je proveden přenos realizovaný kanálem tři, jež je navázaný na kanál dva, tento přenos přenese do čítače N2HETu nový čas, do kterého bude čítač čítat a tím je určený interval, za který bude odeslána následující zpráva.

Zhodnocení řešení: Toto řešení úspěšně implementuje rozesílání více zpráv dle stanoveného rozvrhu. Jeho nevýhodou je, že pokud se zpráva s určitým identifikátorem vyskytuje v rámci periody rozvrhu vícekrát, tak je při jejich změně nutné změnit data ve výskytech zprávy s daným identifikátorem v rozvrhu.

3.1.3 Režimy odesílání dat

Vzhledem k nutnosti měnit data ve zprávách obnáší uložení dat přímo v rozvrhu komplikace a nárůst spotřebované paměti. Problém nastane pokud se zpráva často v rozvrhu opakuje. Protože v tom případě pokud se zpráva nachází v rámci periody rozvrhu vícekrát, tak se vyskytuje i vícekrát v rámci pole zpráv. Což má za následek, že při nutnosti změnit data ve zprávě k danému ID, je nutné změnit data ve všech výskytech zprávy v rámci pole zpráv. Proto byly navrženy dva módy mající za cíl tento problém řešit.

Data in schedule

Diagram: Je stejný jako výše (obrázek 9).

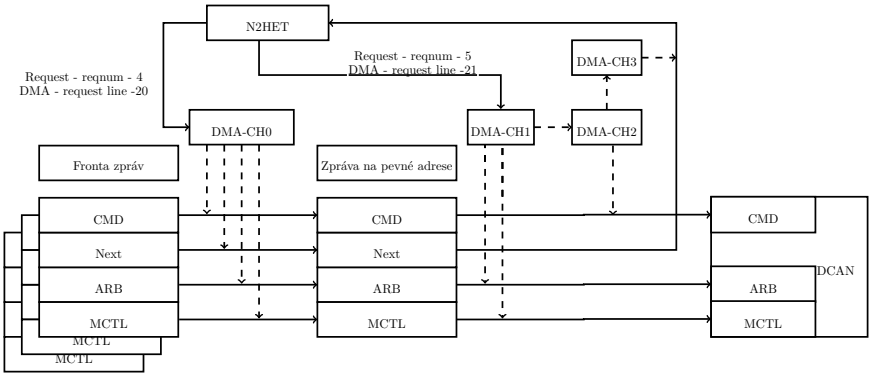
Princip: Princip a diagram jsou stejné jako výše (obrázek 9) s tím rozdílem, že tento mód je rozšířen o možnost uložit data konkrétní zprávy do vyhrazeného message objektu. To, kam se mají data uložit určuje flag DISS (Data in schedule slot), pokud je nastaven na hodnotu 1, tak se použijí data z rozvrhu, pokud je nastaven na 0, tak se jen spustí požadavek na odeslání dat již připravených v daném message objektu. Běžně jsou data DMA kanálem přenesena do registru rozhraní D-CAN. Pokud je však nastavený příznak DISS nastaven na jedna tak nedojde k přenosu do konkrétního message objektu.

Princip změny dat:

i DISS=0 - data se mění v message objektu

ii DISS=1 - data se změní u prvního výskytu dané zprávy v rozvrhu, vzdálené od aktuálního času alespoň o hodnotu makra CANTT_MESSAGE_MIN_SPACE definovanou jako počet zvýšení hodnoty čítače.

Data in message ram



Obrázek 10 Odeslaní více zpráv pomocí DMA

Princip: Princip činnosti je stejný jako výše zmíněný mód Data in schedule pouze s tím rozdílem, že při DMA přenosu se nepřenáší datová položka zprávy. Graficky je to znázorněno na obrázku 10. Data zprávy jsou uložena přímo v message objektu. Vzhledem k tomu, že každá zpráva má svůj message objekt, tak je počet zpráv limitovaný na 64. Hodnota flagu DISS je ignorována.

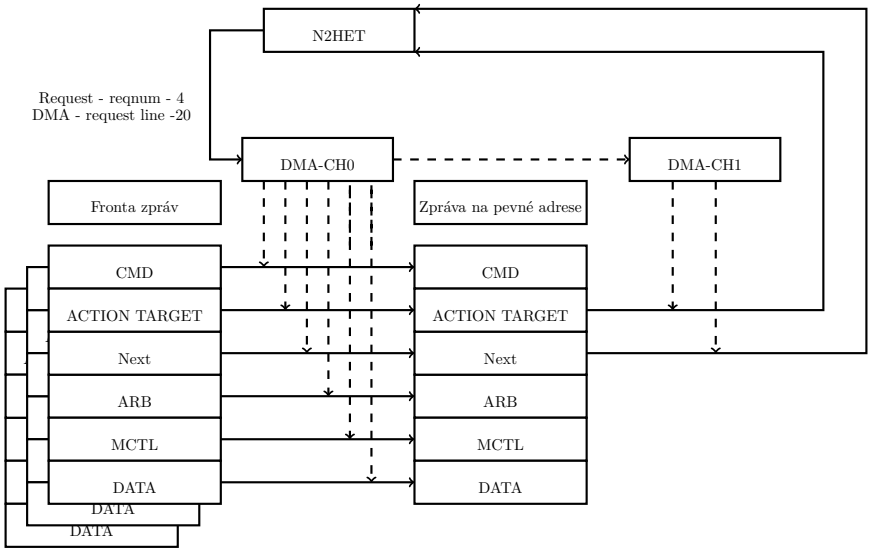
Změna dat: Ke změně dat dojde tím že, se změní data přímo v konkrétním message objektu odpovídajícímu dané zprávě.

Možné vylepšení: Možnost další optimalizace je, že by došlo k nastavení hodnot registrů ARB a MCTL konkrétního message objektu při počáteční inicializaci. To by vyžadovalo rozšíření DCANového driveru o funkce umožňující nastavení daných registrů. Poté by bylo možné odstranit ze zpráv hodnoty zmíněných registrů čímž by se ušetřilo 64 bitů na zprávu v rámci rozvrhu.

3.2 Oddělení odesílání zpráv od vykonávání rozvrhu

Motivací pro oddělení vykonávání rozvrhu od odeslání zprávy je nutnost vykonávat i jiné operace v rámci rozvrhu. Těmito operacemi může být například vyvolání přerušení, ale i odesílání zpráv přes více CANových kontrolérů.

Princip: Na obrázku 11 je zobrazen princip vykonávání rozvrhu. Rozdílem oproti předchozího návrhu je, že zde přibyla položka ACTION_TARGET, jejíž hodnota je přenesena do N2HETu. Tato hodnota po přenesení do N2HETu mění vykonávání programu tím, že u instrukce realizující čítač mění číslo následující instrukce, která se má vykonat po dosažení přiřazeného odstupu od předchozí akce. Sekvence instrukcí, které začínají na odkazovaných adresách, pak vykonají tu činnost, která je danou položkou rozvrhu požadovaná.

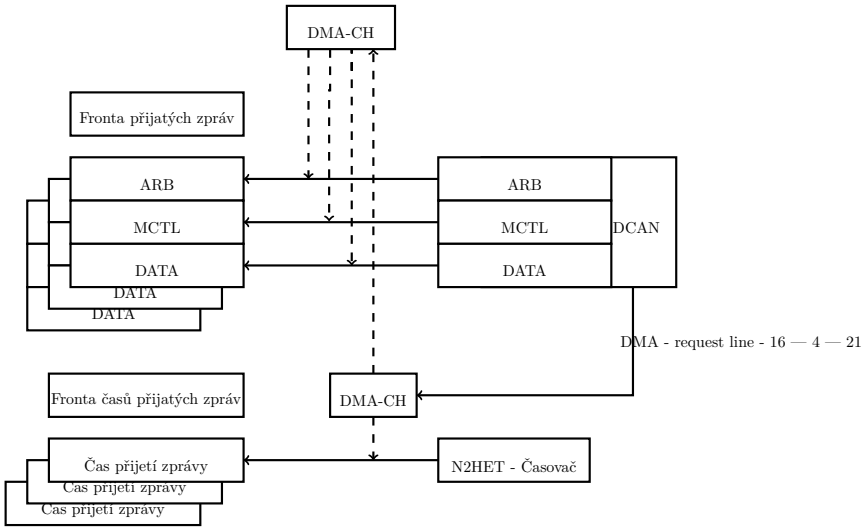


Obrázek 11 Vykonávání rozvrhu

3.3 Příjem zpráv

3.3.1 Přijetí zprávy

Příjem zpráv je realizován pomocí 3 rozhraní DCAN kontroléru. Toto rozhraní je speciálně navrženo pro spolupráci s DMA. DCAN je možné nakonfigurovat tak, že po přijetí zprávy vyšle požadavek na DMA a do rozhraní 3 připraví danou zprávu. U přijaté zprávy na rozhraní 3 je možné nastavit, které části zprávy musí být přetečené předtím, než zpráva může být přepsána další přijatou zprávou.



Obrázek 12 Příjem zpráv

Princip: Příchozí zprávy jsou zpracovávané v následujících krocích (graficky znázorněno na obrázku 12):

i CAN kontrolér přijme zprávu
ii CAN kontrolér vyšle požadavek na DMA
iii DMA převede aktuální čas z N2HETu do fronty časů přijatých zpráv

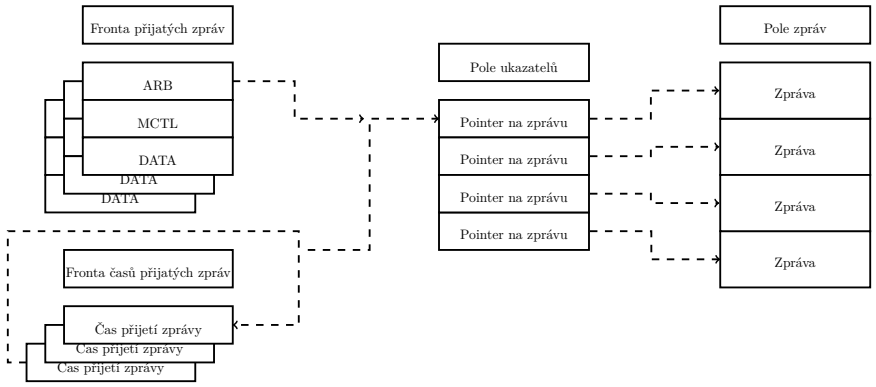
iv DMA přenese obsah přijaté zprávy z třetího interfacu DCANu do fronty přijatých zpráv

Zprávy a časy jsou do front přidávány cyklicky za sebou.

3.3.2 Přístup ke zprávám

Čtení zpráv z fronty přijatých zpráv využívá toho, že struktura instrukce čítače v N2HETu inkrementuje pouze horních 25 bitů a dolních 7 nechává nulových. Je tedy možné použít nultý bit pro označení, že daná zpráva byla přečtena tím, že bude nastaven na 1. Na začátku je tedy nutné označit všechny zprávy jako přečtené, aby nedošlo k jejich přečtení. Potom je možné detekovat přijetí zprávy přečtením nultého bitu, který je v takovém případě nastaven na 0. Tento mechanizmus také umožňuje detekci přetečení fronty zpráv a to tím způsobem, že poslední přečtená zpráva bude mít hodnotu nultého bitu 0. Díky tomuto návrhu je možné přistupovat k přijatým zprávám jako k datové struktuře - frontě.

Zpřístupnění zpráv úlohám a funkcím běžícím na procesoru je realizováno vykopírováním zprávy z příchozí fronty do paměťových lokací vyhrazených pro jednotlivé přijímané identifikátory zpráv. Lokace pro přijatou zprávu je určena polem ukazatelů, které je indexované identifikátorem zprávy. Toto řešení je použitelné v případě, že zprávy používají standardní velikost 11 bitů pro ID. V případě, že je použito rozšířené 29 bitové ID, tak není možné umístit pole pro mapování lokací k identifikátorům do paměti ($32 * 2^{29} = 17179869184$ bitů). Tento problém by se dal vyřešit tím, že by pole ukazatelů obsahovalo pouze určité zprávy ale setříděné, což by znamenalo možnost použít binární vyhledávání a vyhledávat konkrétní zprávu v logaritmickém čase. V rámci diplomové práce je implementováno pouze řešení pro standardní velikost ID zprávy.



Obrázek 13 Zpřístupnění zpráv

Na obrázku 13 je zobrazen výsledný princip zpřístupnění zpráv. Využije se konstrukcí přestavených v předchozích odstavcích. Nejprve dojde k přečtení zprávy z fronty a poté pomocí indexace podle jejího ID dojde k nalezení místa v paměti kam se má uložit a dojde k jejímu uložení. Toto zpracování přijatých zpráv je provedeno vždy před spuštěním nějaké funkce na procesoru, které je popsáno níže.

3.4 Synchronizace

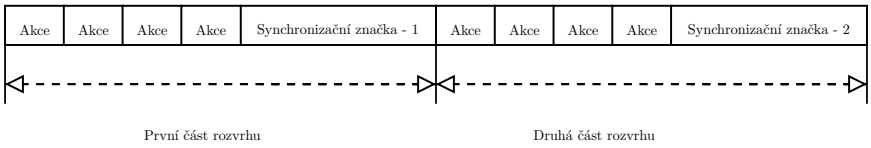
Pro synchronizaci byla zvolena master-slave architektura. K synchronizaci je použita zpráva s určitým ID, jejíž data určují, jaká část rozvrhu se má aktuálně spustit.

3.4.1 Master

Jedná se o standardně fungující jednotku komunikace, lišící se od ostatních, tím že v určitém čase odesílá synchronizační zprávy. Synchronizační zpráva využívá standardní délku ID (11 bitů) a hodnota ID zprávy se rovná 1. ID musí být dlouhé 11 bitů, protože N2HET, jež na slave jednotce ověřuje hodnotu ID, umožňuje hlavně 25 bitové operace. Dále synchronizační zpráva obsahuje v datové části číslo určující jaká část rozvrhu se má na slave jednotkách spustit.

3.4.2 Slave

Slave jednotka se odlišuje od master jednotky tím, že ve rozvrhu obsahuje jednu nebo více akcí s hodnotou action target nastavenou na hodnotu makra CANTT_ACTION_TARGET_WAIT_FOR_NOTIFICATION_MESSAGE. Příjem synchronizační zprávy je možný pouze na jednom DCAN kontroléru a to na tom, který je v konfiguraci vybrán jako can 1 (viz. kapitola Implementace). Je ale možné, aby slave na jedné síti byl master na druhé.



Obrázek 14 Rozvrh se synchronizací

Jak je znázorněno na obrázku 14, tak jednotlivé synchronizační značky se nachází vždy na konci odpovídající části rozvrhu. Po přijetí synchronizační zprávy dojde v přerušení k nastavení DMA na přenos části rozvrhu, odpovídající hodnotě v synchronizační zprávě.

Detekce přijaté zprávy je realizována v rámci N2HETu, ale vzhledem k jeho konstrukci je nutné, aby synchronizační zpráva používala standardní velikost ID (11 bitů), protože N2HET umožňuje pouze 25 bitové operace. Detekce synchronizační zprávy je dostupná ve dvou verzích:

i První verze reaguje na přijatou zprávu ve chvíli přijetí dané zprávy. DMA je tedy přenastaveno okamžitě po jejím přijetí což může způsobit, že pokud synchronizační zpráva přijde brzy, tak se část rozvrhu nevykoná.

ii Druhá verze reaguje na synchronizační zprávu, až poté co v rámci rozvrhu dojde k položce mající jako action target CANTT_ACTION_TARGET_WAIT_FOR_NOTIFICATION_MESSAGE.

V rámci přerušení jednou za cyklus je již na procesoru řešené škálování časové základny. Časová základna je škálovaná tím způsobem, že na začátek N2HETového kódu jsou umístěny následující příkazy:

```
1 ST01    ADD { src1=T,src2=IMM,dest=T,next=ST02,data=16777216};
2 ST02    BR { next=ST00,cond_addr=L00,event=C};
```

Příkaz ST01 realizuje 25 bitový akumulátor, který přičítá hodnotu položky data a druhý příkaz realizuje podmínku, která pokud dojde k přetečení při předchozí operaci pokračuje ve vykonávání programu. Měněním datové položky v prvním příkazu je tedy možné dělit frekvenci čítače, který spouští jednotlivé úkony. Výsledná frekvence je tedy rovna:

$$(frekvenceN2HETu)/(2^{25}/(datovapolozkavprikazuST01))$$

Změna frekvence čítání čítače je realizována v rámci přerušení při němž dochází k přepojování rozvrhu, protože v této chvíli známe délku rozvrhu a skutečný čas běhu. Nová hodnota přičítaná v rámci první instrukce je upravovaná tak, aby s určitou mírou filtrace (potlačení šumu) sledovala hodnotu vypočítanou z intervalu mezi dvěma příjmy synchronizační značky měřenému proti hodinám slave jednotky. PI regulátor byl zvolen namísto PID regulátoru, protože derivační složka prudce reaguje na změny což nechceme, jelikož velké změny znamenají možnou chybu (výpadek zprávy) a nejsou u krystalového oscilátoru předpokládané.

V rámci následujícího odvozování budeme brát v úvahu tři čítače a to:

i **Čítač 1**: Čítač čítající před škálováním
ii **Čítač 2**: Čítač reprezentující škálování
iii **Čítač 3**: Čítač čítající po škálováním

Formálně je tedy frekvence čítače 3 na slave jednotce f_{cs} rovna:

$$1. \quad f_{cs} = (f_{hs} \cdot sf)/2^N$$

kde f_{hs} je vstupní frekvence N2HETu neboli frekvence čítače 1, sf je hodnota uložená v datové položce instrukci ST01 a hodnota N je počet bitů v čítači 2.

Frekvence čítače 3 na master jednotce f_{cm} je rovna

$$2. \quad f_{cm} = 1/2 \cdot f_{hm}$$

kde f_{hm} je vstupní frekvence N2HETu. Vstupní frekvence je dělena dvěma, což je defaultní hodnota škálování na master jednotce. Frekvenci čítače lze také rozvést následujícím způsobem:

$$3. \quad f_{cm} = h_{cyc}/T_{cyc}$$

kde h_{cyc} je rovno hodnotě periodě rozvrhu a T_{cyc} je rovno periodě přetečení čítače 2.

Z rovnice 3. dostáváme:

$$4. \quad T_{cyc} = h_{cyc}/f_{cm}$$

Hodnota čítače 3 na slave jednotce je rovna:

$$5. \quad h_{hs} = T_{cyc} \cdot f_{hs}$$

Úpravou získáváme:

$$6. \quad f_{hs} = h_{hs}/T_{cyc}$$

Dosazením rovnice 4. do 6. dostáváme:

$$7. \quad f_{hs} = h_{hs}/(h_{cyc}/f_{cm})$$

Po úpravě:

$$8. \quad f_{hs} = (h_{hs} \cdot f_{cm})/h_{cyc}$$

Dosazením z rovnice 2. do rovnice 8. dostáváme:

$$9. \quad f_{hs} = (n_{hs} \cdot 1/2 \cdot f_{hm})/h_{cyc}$$

Cílový stav je dosažení toho, že je frekvence čítačů 3 na slave jednotce shodná s frekvencí čítače 3 na master jednotce schodná:

$$10. \quad \hat{f_{cs}} = f_{cm}$$

Po dosazení z rovnice 1. a 2. dostáváme:

$$11. \quad (f_{hs}.\hat{sf})/2^N = 1/2 \cdot f_{hm}$$

Po úpravě:

$$12. \quad \hat{sf} = 2^{N-1} \cdot f_{hm}/f_{hs}$$

Dosazením rovnice 9. do rovnice 12. dostáváme optimální hodnotu sf:

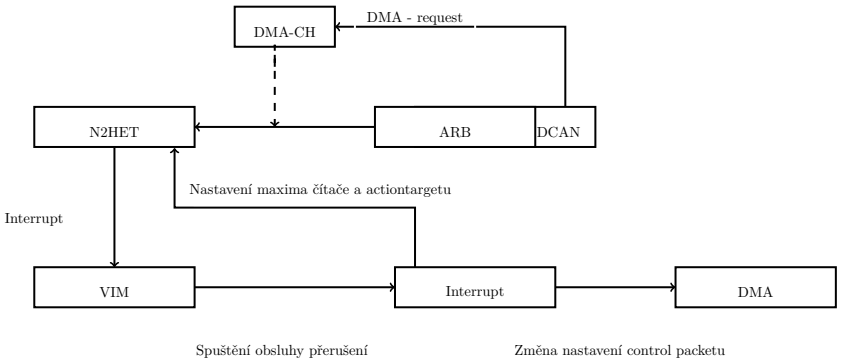
$$13. \quad \hat{sf} = 2^N \cdot h_{cyc}/h_{hs}$$

Výsledná rovnice pro postupné přiblížení se frekvence časování rozvrhu na slave jednotce hodinám na master jednotce:

$$sf_{k+1} = sf_k + (\hat{sf} - sf_k) \cdot k_i$$

kde k_i určující integrační složku a sf_k je hodnota sf v čase k. V PI regulátoru je využito pouze I, které zaručuje vynulovat statickou odchylku, při velmi malé hodnotě konstanty. Díky pomalému najíždění k optimální hodnotě a minimální rychlosti změn frekvence oscilátorů je dynamika systému bezpečná.

Výsledný princip činnosti slave jednotky



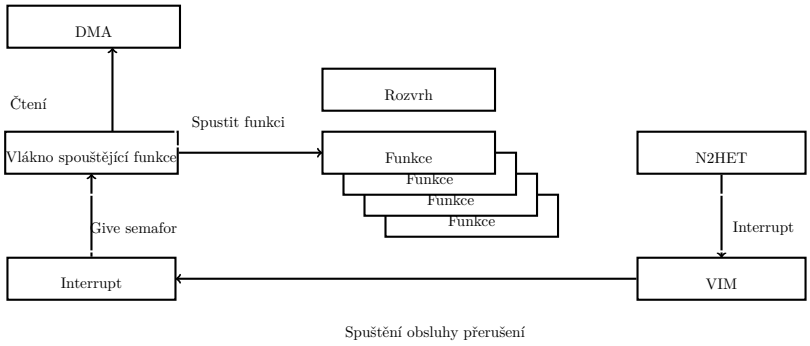
Obrázek 15 Synchronizace slave jednotky

Princip činnosti je zobrazen na obrázku 15. Po přijetí zprávy dojde k odeslání požadavku na DMA a tím dojde k zahájení přenosu z třetího rozhraní DCANu do N2HETu. N2HET ověří jestli se jedná o synchronizační zprávu a to tím způsobem, že ověří ID přijaté zprávy. Pokud zpráva není synchronizační, tak ji ignoruje. V případě, že se jedná o synchronizační zprávu, tak N2HET vyvolá požadavek na přerušení a vynuluje čítač pro měření intervalu do další akce. Po vyvolání požadavku na přerušení VIM spustí obsluhu přerušení na procesoru. V rámci obsluhy přerušení dojde k přenastavení DMA, na část rozvrhu určenou v přijaté synchronizační zprávě. Dále procesor změní cílovou akci, maxima čítače a upraví hodnotu škálování v N2HETu.

Jak vyplývá z konstrukce synchronizace, tak pokud rozvrh obsahuje pouze jednu synchronizační zprávu a nevyužívá škálování, tak lze takový rozvrh realizovat bez přerušení a tedy plně hardwarově.

3.5 Spouštění úloh

V rámci rozvrhu bylo požadované spouštět určité funkce. Tyto jsou zavolány v případě, že je položka ACTION_TARGET nastavena na CANTT_ACTION_TARGET_INTERRUPT.



Obrázek 16 Spuštění úloh

Princip: Princip činnosti je zobrazen na obrázku 16. V čase určeném ke spuštění funkce dojde k přetečení čítače v N2HETu, což vyvolá požadavek na přerušení. Po vyvolání požadavku na přerušení VIM spustí obsluhu přerušení. V rámci obsluhy přerušení dojde k uvolnění semaforu a následnému spuštění úlohy, jež vykoná příslušné funkce. Vzhledem k tomu, že během prodlevy před spuštěním obslužné rutiny a úlohy na procesoru mohlo dojít k dalším požadavkům, tak dojde ke spuštění všech úloh od poslední spuštěné až do aktuální pozice DMA.

4 Implementace

4.1 Popis aplikačního rozhraní (API)

API umožňuje inicializaci periferií TMS570 a následné spuštění nebo zastavení odesílání zpráv, spouštění funkcí dle rozvrhu. Dále umožňuje změnu dat ve zprávách. K inicializaci slouží funkce *cant_init* a provádí se pomocí struktur *cantt_config* a *cant_message*. Ke změně dat slouží funkce *cantt_update_message_data* a datová struktura *cantt_update_data*.

4.1.1 Inicializace

Struktura *cantt_config* obsahuje konfiguraci jednotlivých periferií.

```
1  struct cantt_config{
2          enum cantt_mode mode;
3          struct rpp_can_config can_config;
4          uint8_t can_controller1;
5          uint8_t can_controller2;
6          uint64_t can_message_objects_receive_1;
7          uint64_t can_message_objects_receive_2;
8          uint8_t can_interface_register_number_transmission_message;
9          uint8_t can_interface_register_number_update_message;
10         uint8_t het_module;
11         uint32_t het_clk;
12         uint32_t period;
13         uint32_t number_of_messages;
14         struct cantt_message* messages;
15         struct cantt_read_message **received_messages_mapping;
16 };
```

Popis jednotlivých prvků struktury se nachází v tabulce 2

Prvek struktury	Účel	Přípustné hodnoty
mode	použitý mód ukládání dat zpráv	data_in_schedule, data_in_message_ram
can_config	konfigurace CANu pro rpp knihovnu	struktura rpp_can_config
can_controller1	určuje, jaký CAN kontrolér bude použitý pro odesílání zpráv, příjem zpráv a příjem synchronizačních zpráv	1,2,3
can_controller2	určuje, druhý CAN kontrolér bude použitý pro odesílání zpráv, příjem zpráv	1,2,3
can_message_objects_receive_1	určuje, které message objekty na can_controller1 budou po příjmu použity pro přenesení do 3 interfacu	64 bitové číslo
can_message_objects_receive_2	určuje, které message objekty na can_controller2 budou po příjmu použity pro přenesení do 3 interfacu	64 bitové číslo
can_interface_register-_number_transmission- _message	určuje, který interface registr bude použit pro odesílání zpráv	1,2
can_interface_register-_number_update- _message	určuje, který interface registr bude použit pro změnu dat zprávy	1,2
het_module	určuje N2HET modul použitý jako čítač	1,2
het_clk	vstupní frekvence N2HETU	frekvence v Hz
period	perioda rozvrhu	čas v µs
number_of_messages	počet zpráv v rozvrhu	
messages	ukazatel na rozvrh se zprávami k odeslání	
received_messages_mapping	mapování přijatých zpráv	

Tabulka 2 Prvky struktury cantt_config

Struktura cantt_message reprezentuje jednu zprávu v rámci rozvrhu.

```
1 struct cantt_message{
2         uint8_t action_target;
3         uint8_t flags;
4         uint32_t hw_object;
5         uint32_t id;
6         uint32_t time;
7         uint8_t data_lenght;
8         union {
9                 uint8_t data_8[4*CANTT_SIZE_OF_DATA_32];
```

```
10                  uint32_t data_32[CANTT_SIZE_OF_DATA_32];
11          };
12          void (*task)(void);
13  };
```

Popis jednotlivých prvků struktury se nachází v tabulce 3:

Prvek struktury	Účel	Přípustné hodnoty
action_target	cíl požadavku/prerušení N2HETu	makra s prefixem CANTT_ACTION_TARGET_
flags	příznaky zprávy	Bit 0 DISS (Data in schedule slot) - 0 vysílací data jsou uloženy v message RAM - 1 vysílací data jsou uloženy v rozvrhu - pro nastavení stačí udělat logický součet příznaků (flags) dané zprávy dané zprávy s hodnotou makra CANTT_FLAG_WITH_DATA_MASK Bity 1-7 – nepoužity
hw_object	hw objekt použitý pro odeslání dané zprávy	1-64
type	typ zprávy	RPP_CAN_STANDARD, RPP_CAN_EXTENDED, RPP_CAN_MIXED
id	id zprávy	29 bitová hodnota pro Extended Frame, 11 bitová hodnota pro Standard Frame
time	čas odeslání zprávy v µs	25 bitová hodnota. první zpráva se odesílá v čase 0 bez ohledu na hodnotu v ní uloženou
data_lenght	délka dat zprávy	0 až 8 bytů
data_8, data_32	data zprávy	
task	funkce, která se má vykonat	ukazatel na funkci bez vstupních výstupních hodnot

Tabulka 3 Prvky struktury cantt_message

V následujícím seznamu se nachází seznam možných action targetů a jejich popis:

i CANTT_ACTION_TARGET_NOP - neprovede žádnou akci
ii CANTT_ACTION_TARGET_CAN - odešle zprávu
iii CANTT_ACTION_TARGET_PIN_UP - nastaví pin 20 N2HETu na logickou hodnotu 1
iv CANTT_ACTION_TARGET_PIN_DOWN - nastaví pin 20 N2HETu na logickou hodnotu 0
v CANTT_ACTION_TARGET_COPY_TIME - vykopíruje aktuální čas (po škálování) z N2HETu, je možné k hodnotě přistoupit pomocí funkce cantt_get_het_copied_time
vi CANTT_ACTION_TARGET_WAIT_FOR_NOTIFICATION_MESSAGE - časovač počká na přijmutí synchronizační zprávy
vii CANTT_ACTION_TARGET_INTERRUPT - spustí přerušení uvolňující servisní vlákno spouštějící úlohu předepsanou rozvrhem

4.1.2 Změna dat

Změna dat využívá strukturu *cantt_update_data* a funkci *cantt_update_message_data*.

```
struct cantt_update_data{
        uint32_t id;
        union {
                uint8_t data_8[4*CANTT_SIZE_OF_DATA_32];
                uint32_t data_32[CANTT_SIZE_OF_DATA_32];
        };
};
```

Popis jednotlivých prvků struktury se nachází v tabulce 4:

Prvek struktury	Účel
id	id zprávy
data_8, data_32	data zprávy

Tabulka 4 Prvky struktury cantt_update_message

4.1.3 Čtení přijatých zpráv

Přístup ke frontě přijatých zpráv zajišťuje funkce cantt_read_message. Funkce má jeden vstupní a jeden výstupní parametr. Vstupním parametrem je číslo CANového kontroléru, který přijal zprávu. Výstupním parametrem je ukazatel na strukturu cantt_read_message. Ve struktuře je uloženo id přijaté zprávy, délka zprávy, data a čas přijetí zprávy. Funkce vrací jako návratovou hodnotu úspěšnost operace.

```
struct cantt_read_message{
        uint32_t id;
        uint8_t dlc;
        uint8_t data[8];
        uint32_t time;
};
```

Tato funkce je interně využita pro mapování přijatých zpráv do pole přijatých zpráv popsaném v podkapitole 3.2.2 Přístup ke zprávám v kapitole Návrh. Toto mapování probíhá vždy před spuštěním úlohy na procesoru.

4.1.4 Spouštění úloh

Pro spuštění spouštění úloh je nutné spustit funkci cantt_iterrupt_run_tasks jako separátní task. Tento task nainicializuje semafor pro synchronizaci s přerušením a je tedy nutné ho spustit dříve, než dojde k vyvolání přerušení spouštějící funkce. Funkce se spouští následujícím způsobem:

```
void my_task2(void *p) {
        ...
        cantt_iterrupt_run_tasks();
}
void main(void) {
        ...
        xTaskCreate(my_task2, FREERTOS_TASK_NAME("my_task2");
        ...
}
```

4.1.5 Příklad konfigurace

```
        struct cantt_message me[]={
                {.action_target=CANTT_ACTION_TARGET_INTERRUPT,
                .task=task1,.time=0},
                {.action_target=CANTT_ACTION_TARGET_INTERRUPT,
                .task=task2,.time=500},
                {.action_target=CANTT_ACTION_TARGET_CAN,
                .flags=CANTT_FLAG_WITH_DATA_MASK,.id=10,.hw_object=1,
                .time=1000,.data_lenght=8,.data_32={0,1}},
                {.action_target=CANTT_ACTION_TARGET_INTERRUPT,
                .task=task3,.time=8000},
                {.action_target=CANTT_ACTION_TARGET_WAIT_FOR_NOTIFICATION_MESSAGE,
                .id=1,.time=10000},
                {.action_target=CANTT_ACTION_TARGET_INTERRUPT,
                .task=task4,.time=10100},
                {.action_target=CANTT_ACTION_TARGET_CAN,
                .flags=CANTT_FLAG_WITH_DATA_MASK,.id=20,.hw_object=2,
                .time=11000,.data_lenght=8,.data_32={0,2}},
                {.action_target=CANTT_ACTION_TARGET_INTERRUPT,
                .task=task5,.time=15000},
                {.action_target=CANTT_ACTION_TARGET_CAN,
                .flags=CANTT_FLAG_WITH_DATA_MASK,.id=30,.hw_object=3,
                .time=16000,.data_lenght=8,.data_32={0,3}},
                {.action_target=CANTT_ACTION_TARGET_WAIT_FOR_NOTIFICATION_MESSAGE,
                .id=2,.time=19000}
        };

        struct cantt_config configuration= {
                .mode=data_in_schedule,
                .can_controller1=CANTT_CAN_CONTROLLER_1,
```

```
30                  .can_controller2=CANTT_CAN_CONTROLLER_2,
31                  .can_interface_register_number_transmission_message
32                          =CANTT_CAN_INTERFACE_REGISTER_1,
33                  .can_interface_register_number_update_message
34                          =CANTT_CAN_INTERFACE_REGISTER_2,
35                  .can_message_objects_receive_1=1<<5,
36                  .can_message_objects_receive_2=1<<5,
37                  .can_config=can_config,
38                  .het_clk=80000000,
39                  .het_module=CANTT_HET_MODULE_1,
40                  .period=20000,
41                  .number_of_messages=10,
42                  .messages=me,
43                  .received_messages_mapping=received_messages_pointers
44          };
45          cantt_init(&configuration);
46
47          cantt_run();
48
49          cantt_iterrupt_run_tasks();
```

Tento příklad obsahuje konfiguraci slave jednotky. Rozvrh je rozdělen na dvě části dvěma čekáními na synchronizační zprávy. V první části rozvrhu dojde ke spuštění funkcí task1 v čase 0 µs, task2 v čase 500 µs a funkce task3 v čase 8000 µs. Dále dojde k odeslání zprávy v čase 1000 µs. Druhá část rozvrhu navazuje na první a obsahuje dvě spuštění funkce a dvě zprávy. Perioda celého rozvrhu 20 000 µs. Použitý mód je *data_in_schedule* a data ve zprávách ukládá přímo v rozvrhu. K odesílání používá první interface registr a k úpravě hodnoty dat využívá druhý interface registr. Jako časovač využívá časovač N2HET 1. Hodiny vstupující do N2HETu mají kmitočet 80 MHz. Pro příjem zpráv pomocí DMA jsou použity message objekty 4 na obou CANových kontrolérech. Příklad nebere v úvahu konfiguraci RPP knihovny.

4.2 Integrace a změny rpp knihovny

V této podkapitole je popsáno jaké soubory byly do RPP knihovny přidány a jaké byly upraveny.

4.2.1 Přidané soubory

V následujícím seznamu najdete soubory a jejich popis o než byla knihovna rozšířena:

i **rpp/cantt_dma.h, rpp/cantt_dma.c** - Jedná se o rozšíření již existujícího DMA driveru od TI. Rozšiřuje možnosti konfigurace o možnost nakonfigurovat control packet a přiřadit ho ke konkrétnímu kanálu. Dále rozšiřuje možnosti čtení aktuálního stavu DMA, a to o možnost číst aktuální cílovou a zdrojovou adresu.

ii **rpp/cantt_het.h, rpp/cantt_het.c** - Jedná se o driver pro N2HET. Tento driver umožňuje inicializaci konkrétního N2HETu. V rámci inicializace je nahrán program a je

provedena konfigurace (povolení přerušení, povolení requestů, časování,...). Dále driver umožňuje spuštění a zastavení vykonávání programu s tím že při zastavení dojde k dokončení aktuálně vykonávané instrukce. Driver dále umožňuje přístup k RAM N2HETu a tím měnit vykonávání programu. V rámci přístupu k programu uloženému na RAM N2HETu je možné měnit maximální hodnotu čítače spouštějícího úlohy, měnit action target a upravovat škálování. Dále driver umožňuje přístup k příznakům přerušení. V rámci hlavičkového souboru jsou uvedena makra jež odkazují na důležité části programu s nimiž je nutné manipulovat s pomocí DMA. V případě změny programu je nutné pro jeho správnou součinnost s ostatními periferiemi nutné změnit hodnoty těchto maker.

iii **rpp/cantt.h rpp/cantt.c** - Jedná se knihovnu umožňující konfiguraci jednotlivých periferií pomocí odpovídajících driverů. Dále knihovna obsahuje obsluhy přerušení.

iv **drv/can.h** - Tento hlavičkový soubor obsahuje hodnoty, jež bylo nutné sdílet mezi driverem CANu a konfigurací DMA.

v **drv/het.h** - Tento soubor obsahuje hodnoty, jež bylo nutné sdílet mezi N2HETovými drivery.

4.2.2 N2HET program

Program v rámci N2HETu je implementován v assembleru. Tento program je následně převeden do struktur v jazyce C. Tyto vygenerované struktury je nutné přidat do driveru pro N2HET.

Jak bylo popsáno v kapitole 3 Návrh, tak jsou k dispozici dvě verze programu pro N2HET v následujících souborech (okomentovaný zdrojový kód se nachází v příloze B):

i **Timer-WaitToSynchPerm.het** - soubor obsahující N2HETový program psaný v asseblerových instrukcích, který reaguje vždy na přijatou synchronizační zprávu

ii **Timer-WaitToSynchrMessageInSchedule.het** - soubor obsahující N2HETový program psaný v asseblerových instrukcích, jež reaguje na synchronizační zprávu pouze v případě, že je nastaven odpovídající action target.

4.2.3 Změněné soubory

V rámci implementace bylo nutné některé soubory obsažené v knihovně upravit. V následujícím seznamu jsou uvedeny soubory jež byli v rámci rpp knihovny upraveny:

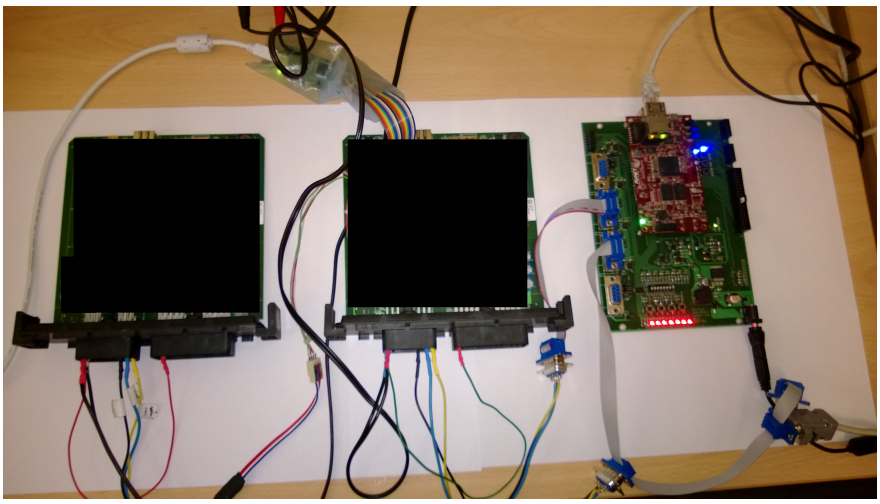
i **rpp/can.h rpp/can.c** - Jedná se driver pro CAN. Tento driver byl rozšířen o možnost nastavit data v konkrétním message objektu a o možnost konfigurace třetího interface konkrétního CAN kontroléru pro příjem zpráv.

ii **sys/sys_startup.c** - Tento soubor obsahuje inicializaci desky. Tento soubor byl rozšířen o povolení přerušení z N2HETu a nastavení odpovídajícího přerušení v tabulce přerušení.

ii **sys/sys_vim.h** - Tento soubor je rozšířen o include funkce realizující přerušení z N2HETu.

5 Měření

K měření byl použit osciloskop Agilent DSO3202A a Xilinx Zynq SoC s integrovaným FPGA ([5]). Zynq realizuje hardwarový a softwarový systém pro přesné měření latence pro sběrnici CAN, a to s rozlišením na mikrosekundy. Na obrázku 17 je fotografie zapojení. Vzhledem k tomu že návrh desek není veřejný, byly desky začerněny. Na obrázku se nacházejí dvě desky master a slave. Deska umístěná vpravo je měřící přístroj Zynq. Mater a slave jednotky mají propojeny CAN kontroléry trojicí drátů (černý, žlutý, modrý).



Obrázek 17 Zapojení pro měření

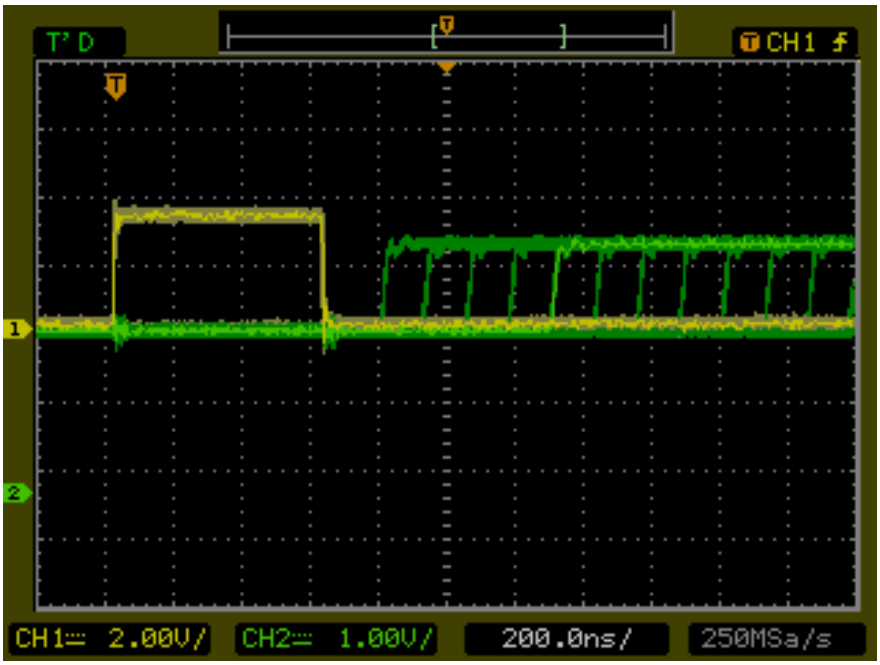
5.1 Měření na osciloskopu

Na následujících obrázcích 18, 19, 20, 21 jsou znázorněny grafy z osciloskopu. Žlutý průběh reprezentuje logickou hodnotu vyvedenou na výstupní pin N2HETu, který je nastaven na logickou hodnotu 1 v okamžiku, kdy má být zpráva přenesená a je znovu nastaven pomocí DMA na hodnotu 0 po ukončení DMA přenosu. Zelený průběh zobrazuje odeslání prvního bitu zprávy, přímo na sběrnici CAN pro odeslání zprávy. Osciloskop byl nastavený na nekonečný dosvit, takže zobrazoval všechny průběhy do jednoho grafu.

5.1.1 Mód - data in schedule

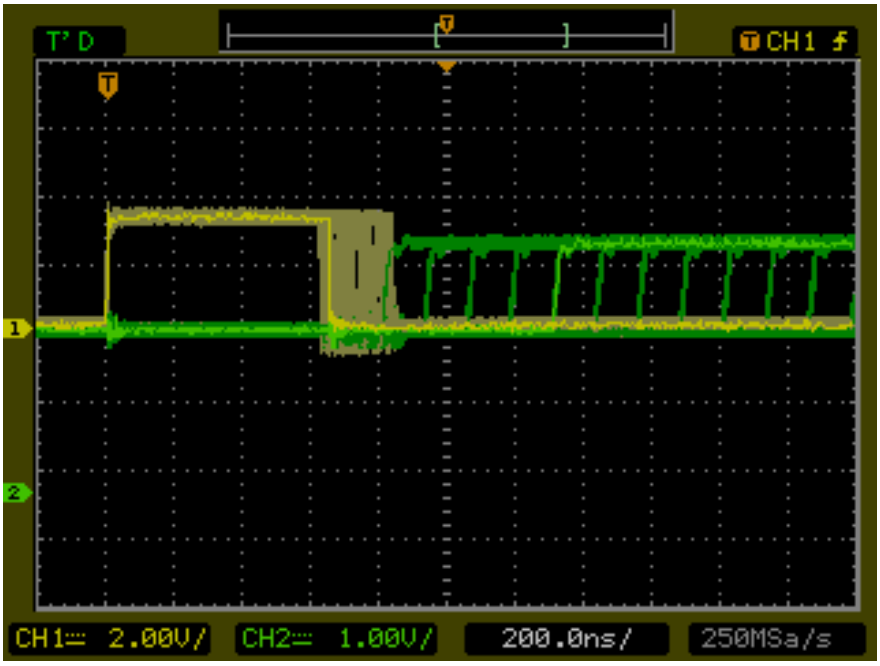
Bez zátěže

Na obrázku 18 je žlutě zobrazena doba přenosu pomocí DMA, v případě procesor není zatížen. Jak je z obrázku patrné, tak obdélník zobrazuje dobu přenosu je neustále na jednom místě, což je dle očekávání, neboť DMA nevyužívá nikdo kromě našeho přenosu z N2HETu. Zelený průběh reprezentující začátek přenosu z DCANu. Z obrázku 18 je vidět, že přenos začíná vždy v určitých časech, jedná se nejspíše o to, že se přenos po zapsání do message objektu zahájí pouze v čase, když k němu dojde takt vnitřního stavového automatu DCANu pro periodickou obsluhu objektu.



Obrázek 18 Data in schedule - odeslání zprávy

Se zátěží



Obrázek 19 Data in schedule - odeslání zprávy se zatížením

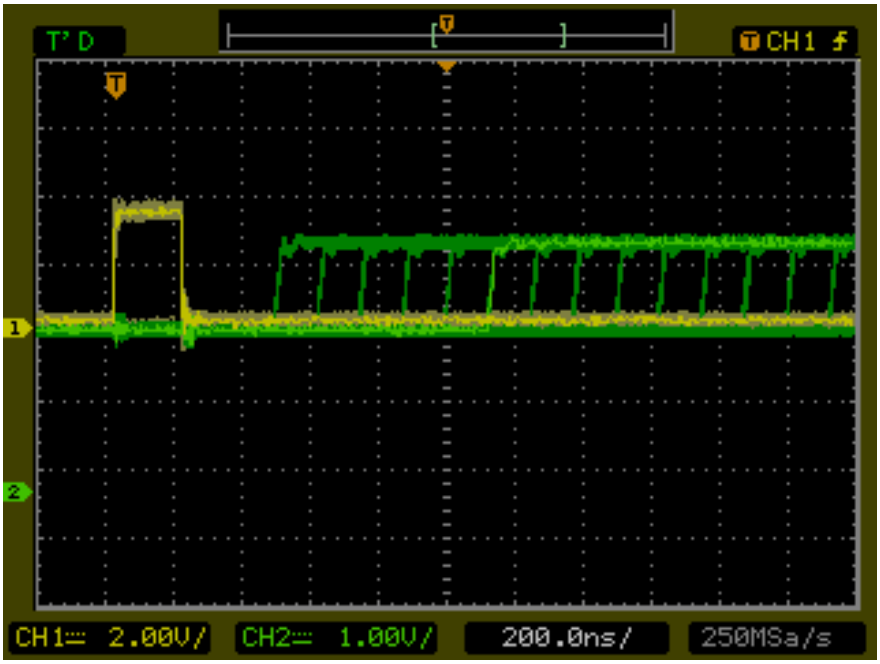
K příkladu výše byla přidána zátěž procesoru v podobě neustálého kopírování dat v rámci paměti. Tímto kopírováním došlo k zatížení procesoru a sběrnice, což se projevilo rozkmitem doby dokončení přenosu v rámci DMA na obrázku 19. Výsledky měření jsou v následující tabulce 5.

Periferie	Minimální čas	Maximální čas	Jitter
DMA - doba přenosu	600 ns	850 ns	250 ns
CAN - zahájení přenosu	820 ns	1.840 µs	1 µs

Tabulka 5 Data in schedule výsledky měření

5.1.2 Mód - data in message RAM

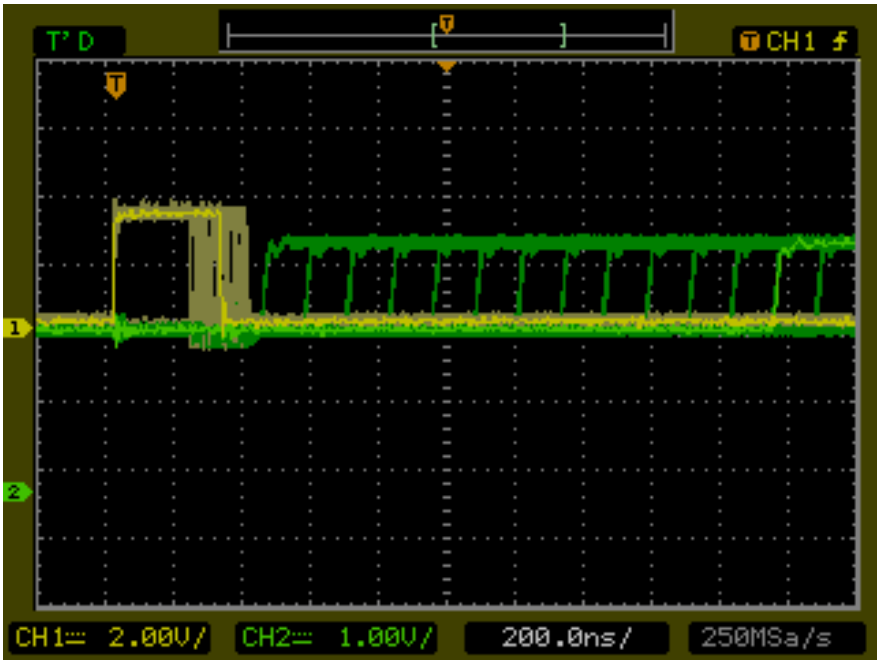
Bez zátěže



Obrázek 20 Data in message RAM - odeslání zprávy

Na obrázku 20 je zobrazen ekvivalent obrázku 18 s tím rozdílem, že byl použit mód data in message RAM.

Se zátěží



Obrázek 21 Data in message RAM - odeslání zprávy se zatížením

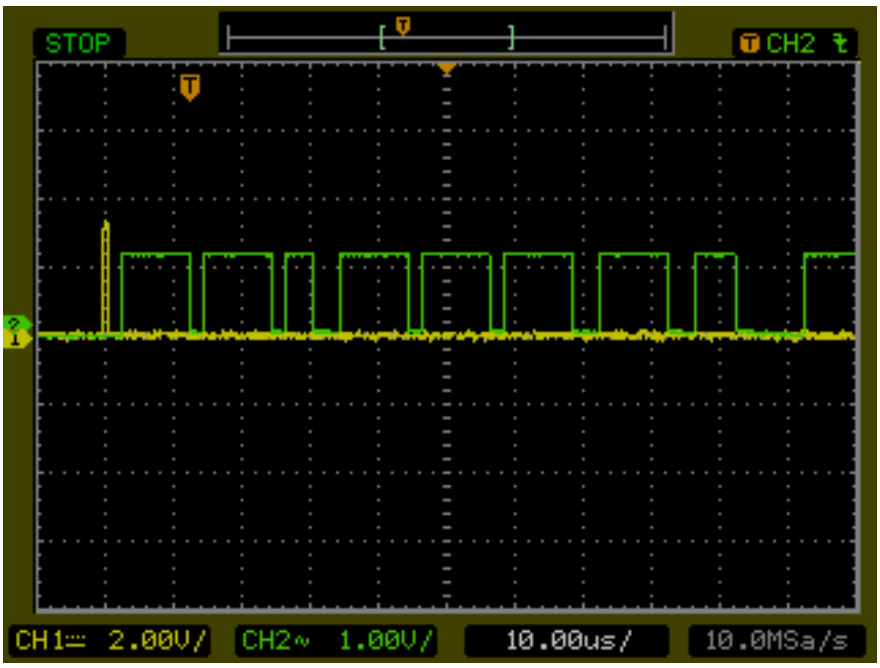
Na obrázku 21 je zobrazen ekvivalent obrázku 19 s tím rozdílem, že byl použit mód data in message RAM. Výsledky měření jsou v následující tabulce 6.

Periferie	Minimální čas	Maximální čas	Jitter
DMA - doba přenosu	224 ns	338 ns	114 ns
CAN - zahájení přenosu	460 ns	2.5 µs	2.04 µs

Tabulka 6 Data in schedule výsledky měření

Dle očekávání je mód Data in message RAM rychlejší, neboť DMA realizuje přenos kratší o 2 bajty a to hned 2 krát. Poprvé je to během přenosu zprávy z pole zpráv na pevnou adresu a podruhé během přenosu ze zprávy na pevné adrese do DCANu.

5.1.3 Srovnání přenosu DMA a DCANU



Obrázek 22 DMA a DCAN přenos

Na obrázku 22 je zobrazený žlutý průběh reprezentující dobu přenosu DMA a zelený průběh zobrazující přenos zprávy. Z obrázku je patrné, že přenos pomocí DMA trvá jen nepatrný okamžik (stovky ns) oproti přenosu zprávy, který může trvat až 300 µs.

5.2 Měření na Xilinx Zynq

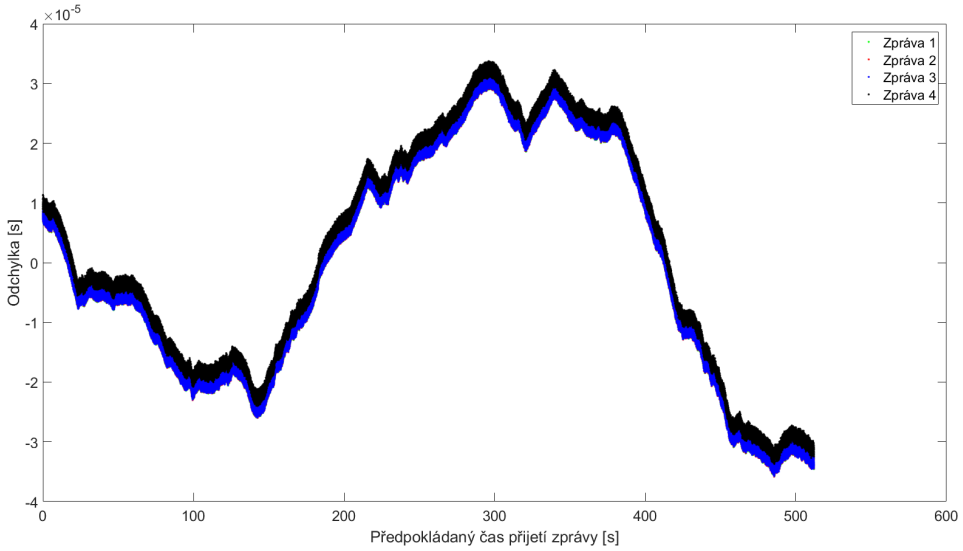
5.2.1 Odesílání zpráv

V následujících grafech jsou jednotlivé zprávy rozlišeny barevně a to dle následující tabulky. Perioda rozvrhu byla zvolena 10 000 µs. Data pro graf byla přepočítána z naměřených dat pomocí lineární regrese a následně odečtena od očekávaného času přijetí.

ID	Čas odeslání[us]	Barva
1	0	zelená
2	500	červená
3	1000	modrá
4	5000	černá

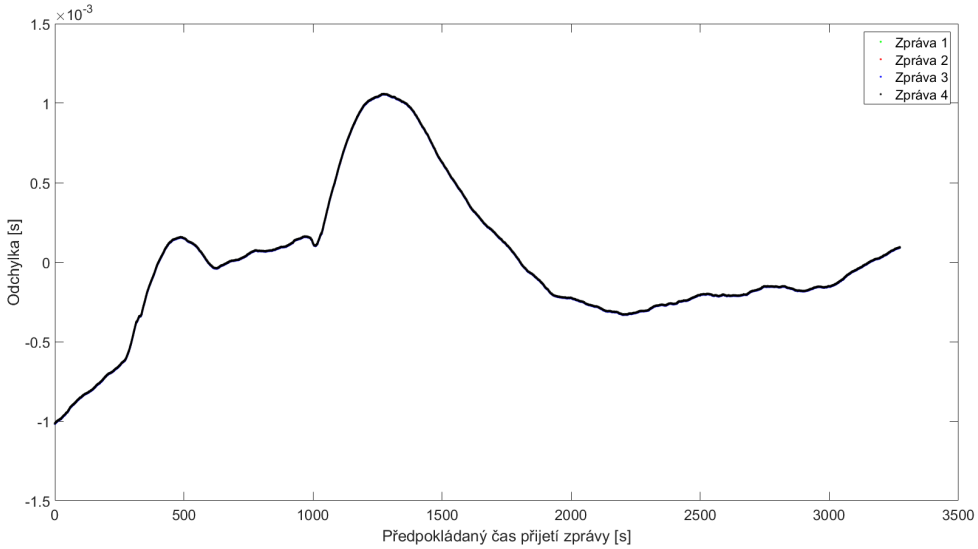
Tabulka 7 Zprávy v grafu

Dlouhé měření



Obrázek 23 Měření nepřesnosti odesílání zpráv

Na obrázku 23 je zobrazené měření trvající přes 500 s. Nejvyšší rozdíl proti očekávanému času je 35.8 µs. Chyba může být způsobena vzájemným driftem frekvence oscilátorů TMS570 a měřícího přístroje.



Obrázek 24 Měření nepřesnosti odesílání zpráv - při změnách teploty

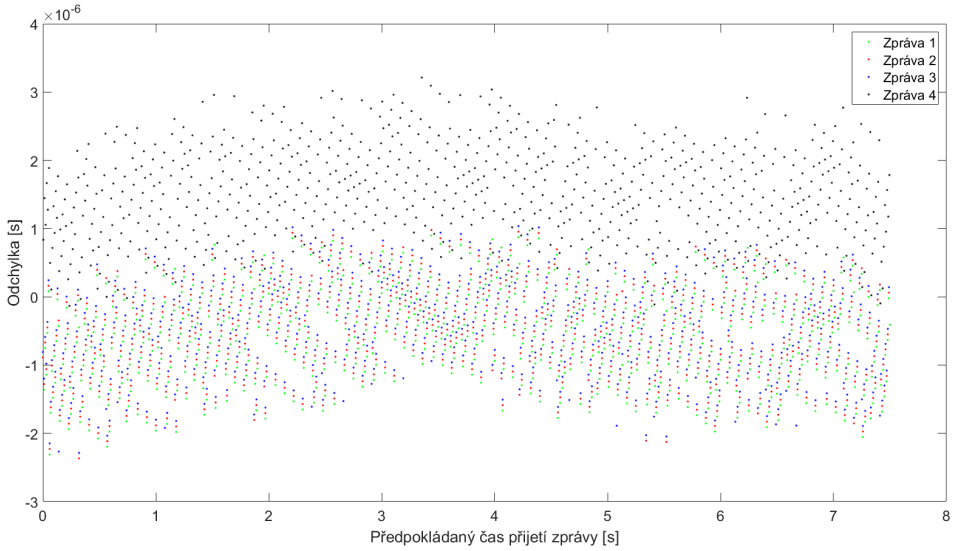
Na obrázku 24 je zobrazen rozdíl proti očekávanému času při zvýšení teploty. Nejprve byla zvýšena teplota odesílateli zpráv procesoru TMS570 a poté byla zvýšena teplota příjemci zpráv systému na bázi Zynq. V grafu na obrázku 24 je každé zvýšení teploty reprezentováno zvýšením odchylky od očekávaného času z čehož vyplývá, že jednomu zařízení se kmitočet při zahřátí zvyšuje a druhému se snižuje. Nejvyšší rozdíl očekávaného proti očekávanému času přesáhl milisekundu.

Vliv teploty na oscilátory je tedy velký. Pro periodu cyklu 10 msec, ale změna není až tak

velká - 3 µs. Pokud by ale byl cyklus například 1 s, tak by již změna jeho délky činila 300 µs, což již koliduje s délkou okna pro jednu zprávu.

Pokud by se tedy časování při použití více jednotek pouze s cyklem synchronizovalo a nekorigovalo by se rozložení zpráv v čase, tak by to pro větší délky cyklů jen s jednou synchronizační značkou mohl nastat problém. Pak přístup TTCANu (Time Triggered CAN), kdy je hyperperioda přes více cyklů vychází lépe. Je potřeba změřené údaje brát v potaz i při návrhu protokolu a synchronizace.

Krátké měření



Obrázek 25 Krátké měření

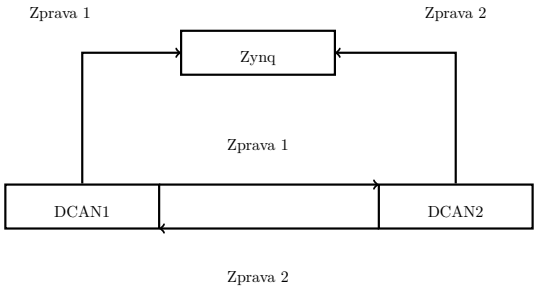
Na obrázku 25 je zobrazena chyba pouze z krátkého výseku dat, aby byli vidět jednotlivé zprávy. Z obrázku je patrné, že zprávy 1,2 a 4 jsou chybou blízko sebe, zatímco zpráva 3 má chybu rozdílnou. Je to způsobeno tím, že zprávy 3 to má k ostatním zprávám čase rozvrhu nejdál.

5.2.2 Měření příjmu zpráv

Způsob měření

Měření probíhalo tím způsobem, že se na daném mikrokontroléru vykonával rozvrh rozesílající zprávy z jednoho CAN kontroléru. Druhý CAN kontrolér tyto zprávy přijímal a program vykonávaný na TMS570 ve smyčce neustále testoval jestli nepřišla zpráva a po jejím přijetí odeslal odpověď. Grafické znázornění naleznete na obrázku 26. Jednotlivé zprávy byly vysílány v časech 100 µs,1 ms,2 ms a 5 ms s délkou periody 10 ms. Konkrétně měření probíhalo v následujících krocích:

1. DCAN1 odešle zprávu jejíž data obsahují čas, kdy by měla být zpráva odeslaná. Tato hodnota je pevně daná v µs.
2. Zynq přijme zprávu a zaznamená obsah a čas přijetí zprávy
3. DCAN2 přijme zprávu a odešle zprávu obsahující čas přijetí zprávy a čas začátku cyklu N2HETu.
4. Zynq přijme zprávu a zaznamená obsah a čas přijetí zprávy



Obrázek 26 Meření příjmu zpráv

Způsob zpracování výsledků

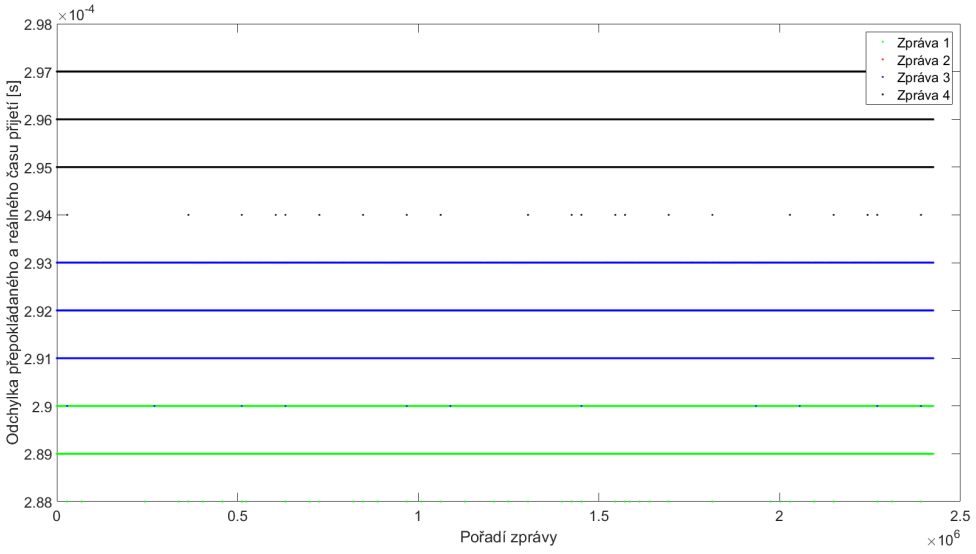
Z dat je možné získat 2 druhy výsledků:

i. Z dat naměřených na Zynqu - odečtením času přijetí zpráv lze získat dobu reakce smyčky realizující odeslání odpovědi na přijatou zprávu.

ii. Z dat obsažených ve zprávách - odečtením předpokládaného času z první zprávy v absolutní hodnotě od rozdílu začátku cyklu a času přijetí první zprávy na desce, lze získat odchylku od předpokládaného času přijetí zprávy.

Výsledky

Na obrázku 27 se nachází odchylka od očekávaného času přijetí zprávy. Jak je z obrázku patrné, tak se zpoždění pohybuje pod hranicí 300 µs, což je přibližně doba potřebná k odeslání zprávy. Mezi jednotlivými úrovněmi se nachází rozmezí 1 µs. V rámci každé úrovně je odchylka menší než 1 µs, což je již mimo rozsah měřícího přístroje. Existence jednotlivých oddělených úrovní je způsobena nejspíše tím, že k odeslání zprávy dochází až když k message objektu dojde takt DCANu.



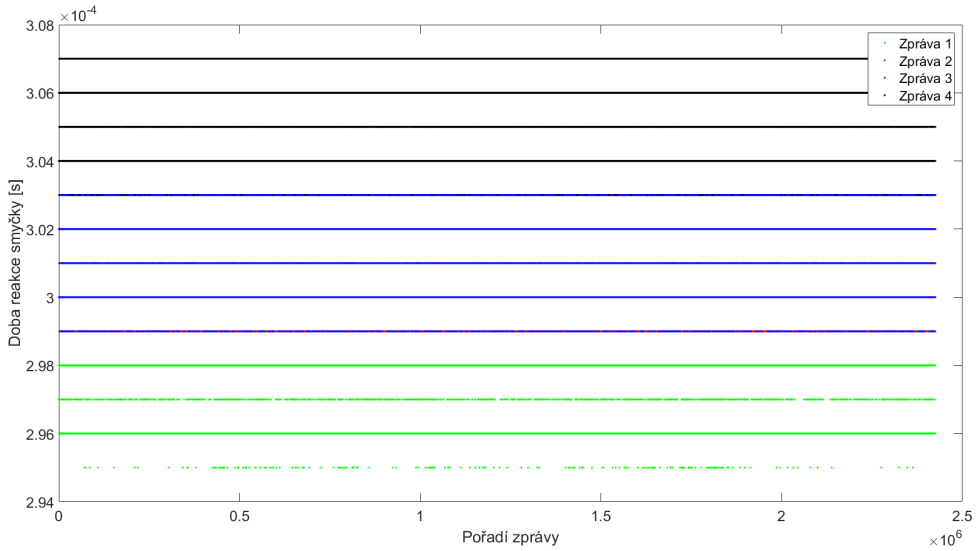
Obrázek 27 Odchylka

V tabulce 8 je zobrazena maximální a minimální hodnota odchylky pro zprávy z obrázku 27. Každá odchylka má rozdíl mezi maximální a minimální hodnotou rovnou 3 µs.

Zpráva	Min [µs]	Max [µs]
1	290	293
2	290	293
3	294	297
4	288	291

Tabulka 8 Mininimální a maximální odchylka

Na obrázku 28 je zobrazena doba odpovědi smyčky čekající na přijetí zprávy. Rozmezí mezi jednotlivými úrovněmi je 1 µs.



Obrázek 28 Doba reakce smyčky - různa data

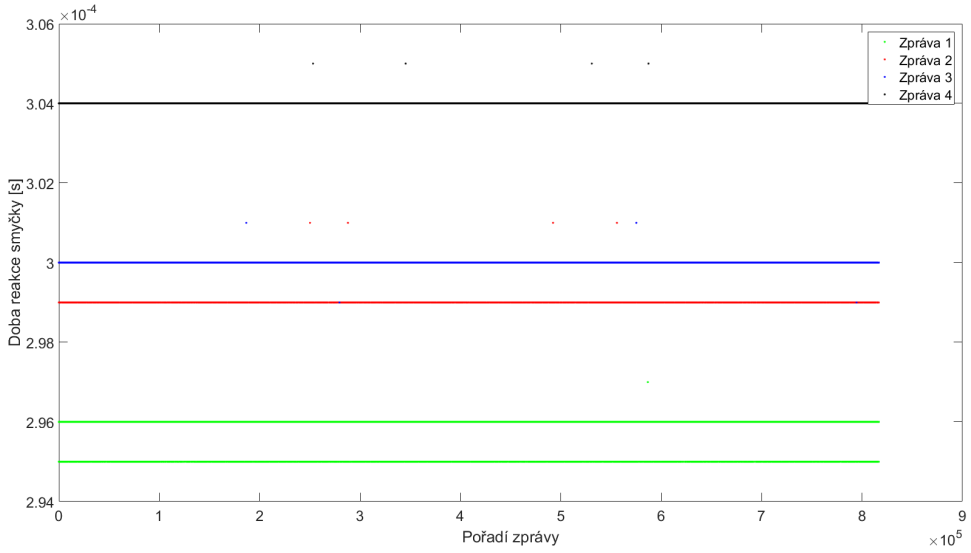
V tabulce 9 je zobrazena maximální a minimální hodnota pro zprávy z obrázku 28. Pro každou zprávu je rozsah reakce smyčky 4 µs.

Zpráva	Min [µs]	Max [µs]
1	299	303
2	299	303
3	303	307
4	295	299

Tabulka 9 Mininimální a maximální odchylka - různa data

To, že jednotlivé zprávy na obrázcích 27, 28 tvoří různé úrovně je způsobeno tím, že měnící se data mají vliv na bit stuffing, což způsobuje, že je zpráva různě dlouhá a její přijetí tedy trvá různou dobu.

Na obrázku 29 je zobrazeno měření s tím rozdílem, že v kroku 3 jsou odesílána v každé zprávě stejná data.



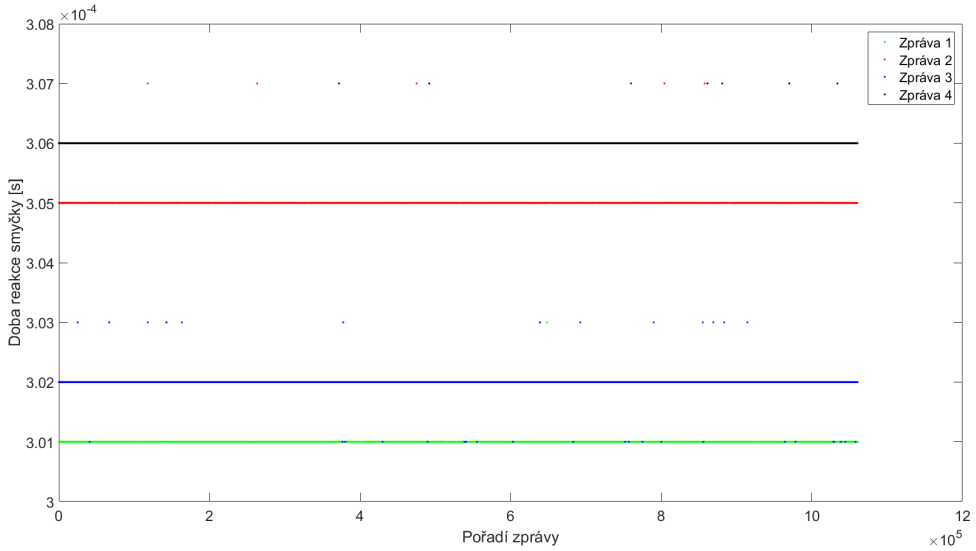
Obrázek 29 Doba reakce smyčky - neměnící se data

V tabulce 10 je zobrazena maximální a minimální hodnota pro zprávy z obrázku 29.

Zpráva	Min [µs]	Max [µs]
1	299	301
2	299	301
3	304	305
4	295	297

Tabulka 10 Mininimální a maximální odchylka - neměnící se data

Na obrázku 30 je zobrazeno měření s tím rozdílem, že v krocích 1 a 3 jsou odesílána v každé zprávě stejná data.



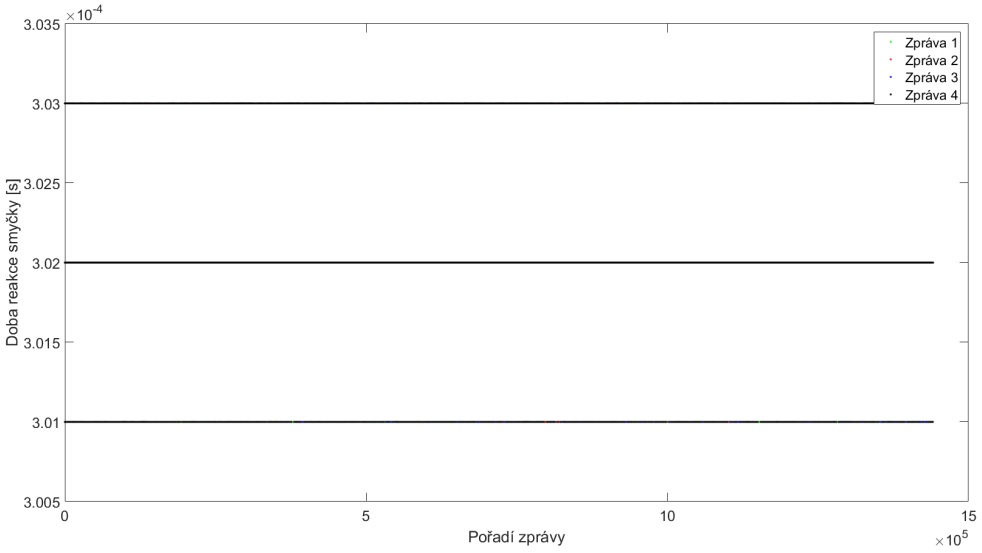
Obrázek 30 Doba reakce smyčky - stejná data ve všech zprávách

V tabulce 11 je zobrazena maximální a minimální hodnota pro zprávy z obrázku 30.

Zpráva	Min [µs]	Max [µs]
1	305	307
2	301	303
3	306	307
4	301	303

Tabulka 11 Mininimální a maximální odchylka - stejná data ve všech zprávách

Na obrázku 31 je zobrazeno měření s tím rozdílem, že v krocích 1 a 3 jsou odesílána v každé zprávě stejná data a se stejným ID. Minimální doba reakce smyčky a maximální s liší u každé zprávy o 2 µs, každá zpráva se tedy vyskytuje ve třech úrovních. Mezera mezi jednotlivými úrovněmi je 1 µs.



Obrázek 31 Doba reakce smyčky - stejná data ve všech zprávách, se stejným ID

Zhodnocení výsledků

Z naměřených dat zobrazených na obrázcích 29, 30 a 31 je patrné, že rozdíly mezi jednotlivými zprávami byly způsobeny bit stuffingem. Na obrázku 31 tvoří zprávy 3 úrovně mezi, každou z nich je odstup 1 µs. V rámci každé úrovně se vyskytují všechny zprávy. Mezery mezi úrovněmi a jejich existence je nejspíše způsobena tím, že k odeslání zprávy dochází až když k message objektu dojde takt DCANu.

5.2.3 Měření přerušení

Čas spuštění přerušení

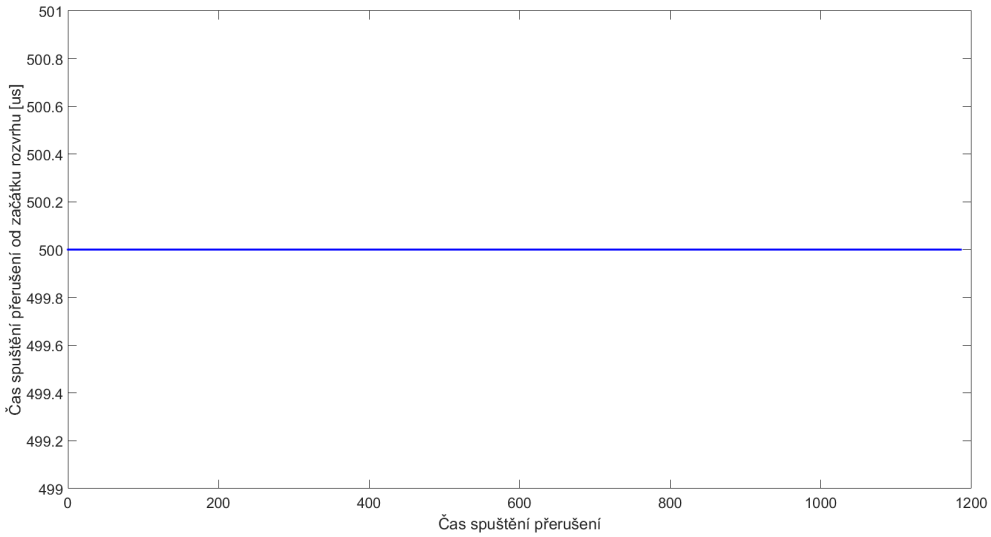
V tomto měření bylo cílem zjistit, jak rychle po spuštění přerušení v rámci N2HETu dojde ke spuštění přerušení. Měření probíhalo v následujících krocích.

i N2HET vykopíruje aktuální čas z N2HETu, čímž získáme záznam, kdy začala perioda rozvrhu

ii N2HET vyvolá přerušení, jež zaznamená aktuální čas z N2HETu a poté uvolní semafor

ii Task jež ve uvolněn v kroku ii. odešle zprávu, jež v horních 4 bytech obsahuje čas spuštění periody rozvrhu a v dolních 4 bytech obsahuje čas spuštění přerušení.

Rozvrh tedy obsahoval v čase 0 µs vykopírování času z N2HETu a v čase 500 µs spuštění přerušení.



Obrázek 32 Čas spuštění přerušení

Na obrázku 32 je jsou zobrazeny časy spuštění přerušení. Jak je z obrázku patrné, tak ke každému spuštění docházelo přesně v čase 500 µs.

Ověření spouštění funkcí

Ověření toho, že dochází ke spouštění funkcí, bylo otestováno na rozvrhu obsahujícím spuštění dvou funkcí a odeslán jedné zprávy. Rozvrh je vykonáván v následujících krocích:

i Dojde k inkrementaci nultého byte ve zprávě
ii Dojde k odeslání zprávy
ii Dojde k inkrementaci prvního byte ve zprávě

Data v odesílané zprávě jsou nastavena na 0. Postupně dochází ke spouštění funkcí, přičemž v rámci zprávy přijaté ne Zynqu je první byte vždy o jedna nižší než nultý byte.

Ověření přístupu k přijatým zprávám

K ověření toho, že přijaté zprávy zprávy jsou zpřístupněny spouštěným funkcím došlo za pomoci dvou jednotek. První master jednotka byla stejná jako v podkapitole "Ověření spouštění funkcí"s tím, že odesílaná zpráva měla ID 1 a byla tedy brána jako synchronizační pro druhou slave jednotku. Rozvrh slave probíhal v následujících krocích:

i Slave jednotka přijme synchronizační zprávu
ii Dojde ke spuštění funkce, jež zkopíruje data z přijaté synchronizační zprávy do zprávy k odeslání v rámci rozvrhu
ii Dojde k odeslání zprávy s ID 2

V rámci měření na Zynqu tedy zpráva s ID 2 obsahovala vždy stejná data jako zpráva z ID 1.

5.2.4 Měření synchronizace

V této podkapitole je popsáno jak probíhalo měření přesnosti synchronizace rozvrhu na síti obsahující jednu master a jednu slave jednotku. Podkapitola je rozdělena na několik části, první část obsahuje měření synchronizace za využití pouze hardwarové podpory, druhá obsahuje měření za využití přerušení, tedy přepínání rozvrhu za pomocí DMA a třetí přidává

navíc škálování časové základny. Nakonec je provedeno srovnání synchronizace bez a se škálováním.

Hardwarová synchronizace

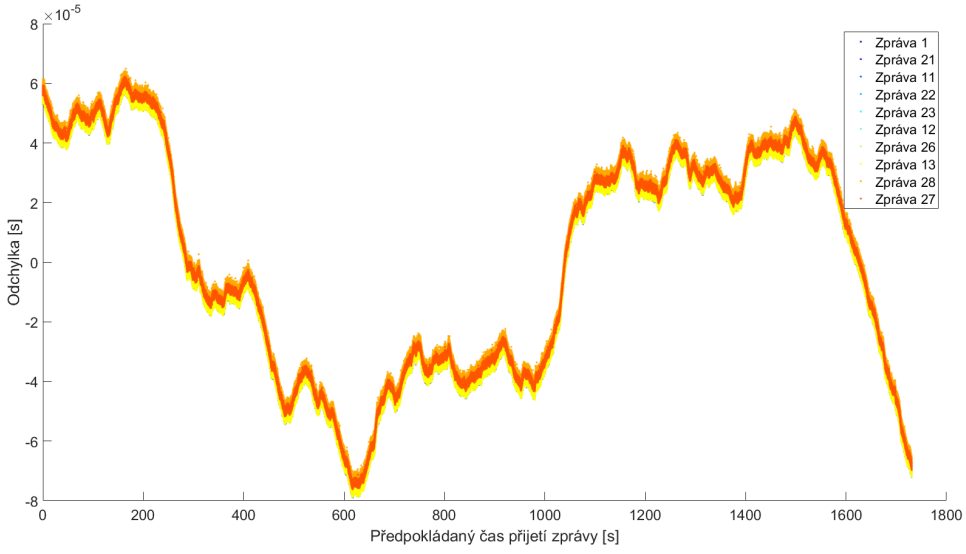
V tomto měření bylo použito pouze hardwarové synchronizace, tedy přerušení byla vypnuta. Vzhledem k tomu, že v případě ztráty synchronizační zprávy by došlo k ”posunutí”rozvrhu na slave jednotce, byl zvolen rozvrh s jednou synchronizační zprávou. Rozvrh na síti je zobrazen v tabulce 12:

Id zprávy	Čas odeslání [µs]	Jednotka
1	0	Master
21	750	Slave
11	2000	Master
22	2750	Slave
23	3550	Slave
12	4000	Master
26	4750	Slave
13	8000	Master
28	8550	Slave
27	9250	Slave

Tabulka 12 Rozvrh - jedna synchronizační zpráva

Perioda rozvrhu byla 10 000 µs.

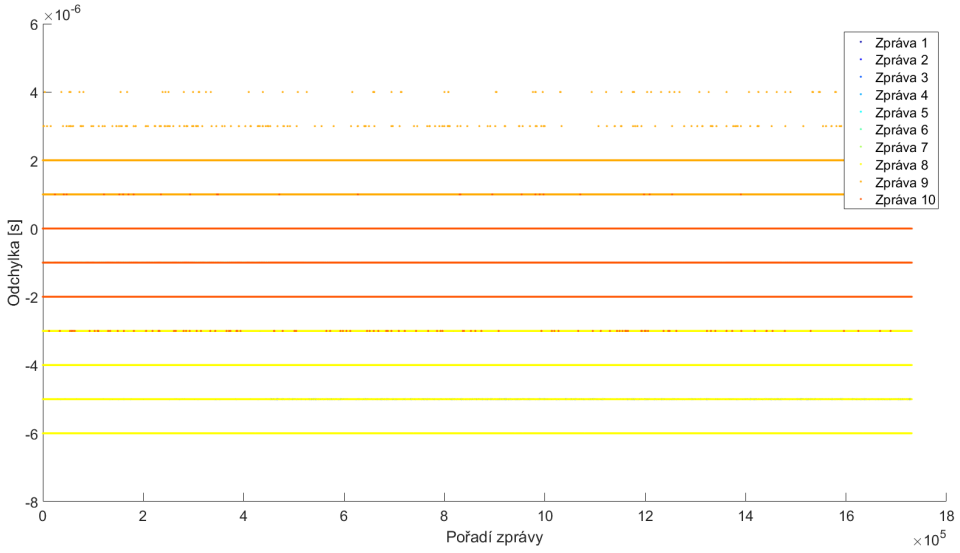
V rámci měření bylo nejprve provedeno měření přesnosti synchronizace bez zahřívání jednotlivých desek a poté se zahřátím jednotlivých desek.



Obrázek 33 Hardwarová synchronizace

Na obrázku 33 je zobrazena přesnost času přijetí zpráv rozvrhu. Na vodorovné ose je zobrazen předpokládaný čas přijetí zpráv a na svislé ose je zobrazena odchylka od předpokládané času přijetí. Nejvyšší hodnota odchylky je rovna 78 µs, takto velká chyba je nejspíše způsobena rozdílností oscilátorů na měřícím přístroji a master and slave jednotkách. Pokud vezmeme v potaz každou periodu rozvrhu zvlášť, a tím vyeliminujeme vliv rozdílnosti

oscilátorů mezi jednotlivými periodami, dostáváme graf na obrázku 34.



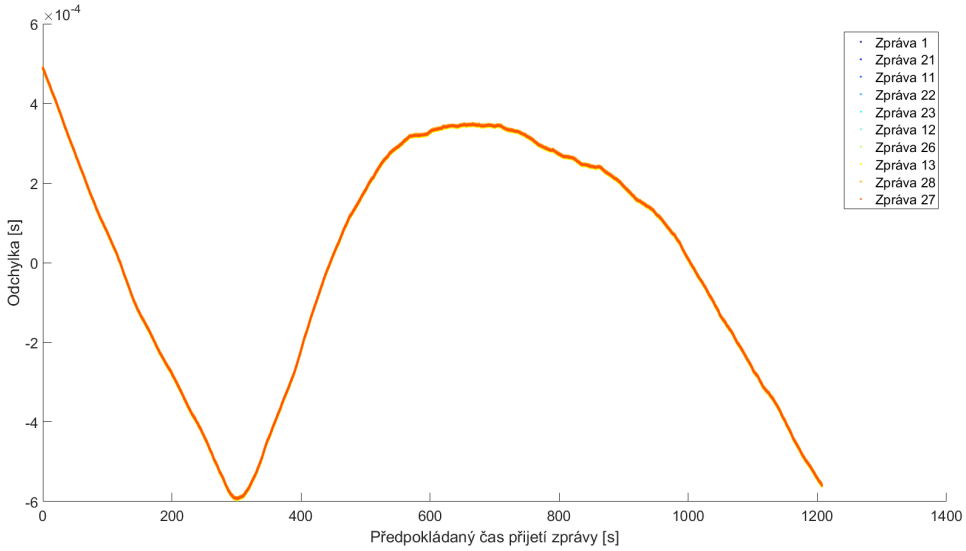
Obrázek 34 Hardwarová synchronizace po jednotlivých periodách

Jak je z obrázku 34 patrné tak nejvyšší chyba v rámci jednotlivých period dosahuje maximálně 6 µs. Chyby pro jednotlivé zprávy jsou znázorněny v tabulce 13.

Id zprávy	Minimální chyba [µs]	Maximální chyba [µs]	Rozdíl [µs]
1	0	0	0
21	-2	2	4
11	-6	-2	4
22	-2	2	4
23	-2	2	4
12	-4	0	4
26	-2	2	4
13	-6	-2	4
28	-1	4	5
27	-3	1	4

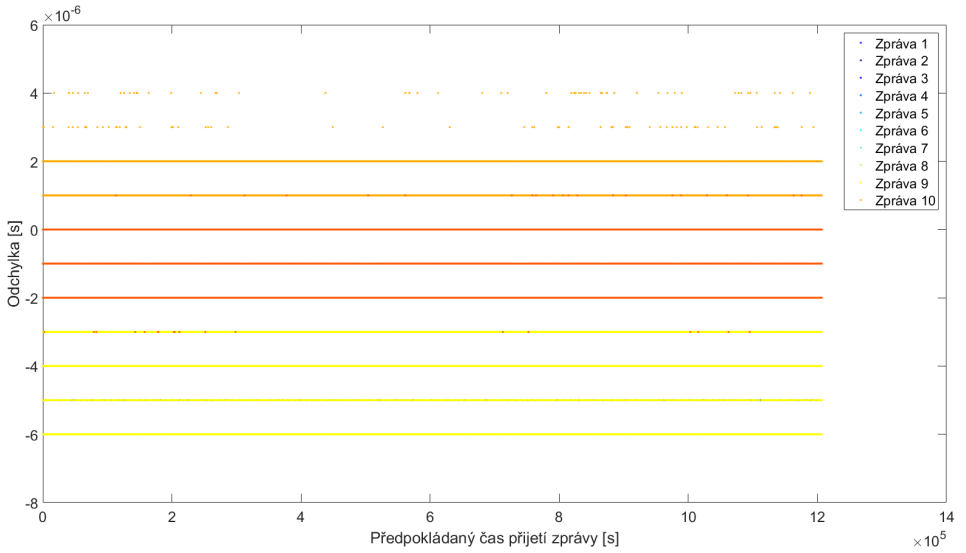
Tabulka 13 Srovnání doby přijetí - hardwarová synchronizace

Z tabulky je patrné že mezi nejnižším časem přijetí zprávy a nejvyšším časem přijaté zprávy je maximálně 5 µs.



Obrázek 35 Hardwarová synchronizace po zahřátí master a slave jednotky

Na obrázku 35 je zobrazena přesnost času přijetí zpráv rozvrhu. Na vodorovné ose je zobrazen předpokládaný čas přijetí zpráv a na svislé ose je zobrazena odchylka od předpokládané času přijetí. V rámci tohoto měření došlo nejprve k zahřátí master jednotky po 5 minutách (300 sekundách) běhu. Zahřívání trvalo 2 minuty a po 5 minutách byla zahřáta slave jednotka. Nejvyšší dosažená chyba je 596 µs. Na obrázku 36 je zobrazena chyba pokud vezme v úvahu každou periodu zvlášť. Jak je z obrázku patrné tak chyba se pohybuje v rozmezí 6 µs.



Obrázek 36 Hardwarová synchronizace po jednotlivých periodách při zahřátí

Chyby pro jednotlivé zprávy při zahřátí jsou znázorněny v tabulce 14. Jak je z tabulky patrné, tak zahřátí nemělo na chybu v rámci periody velký vliv, pouze u zprávy 12 došlo ke zhoršení o 1 µs.

Id zprávy	Minimální chyba [µs]	Maximální chyba [µs]	Rozdíl [µs]
1	0	0	0
21	-2	2	4
11	-6	-2	4
22	-2	2	4
23	-2	2	4
12	-4	1	5
26	-2	2	4
13	-6	-2	4
28	-1	4	5
27	-3	1	4

Tabulka 14 Srovnání doby přijetí - hardwarová synchronizace po zahřátí

Z uvedených grafů a tabulek je jasně vidět že chyba v rámci periody je malá, ale postupně se nasčítá. Problémem je rozdílná frekvence oscilátorů master jednotky a měřícího přístroje. V rámci jednotlivých period je chyba v řádu mikro sekund, jak na master, tak na slave jednotce. S ohledem i na předchozí měření se již dostáváme na limity daného hardwaru.

Synchronizace s využitím přerušení

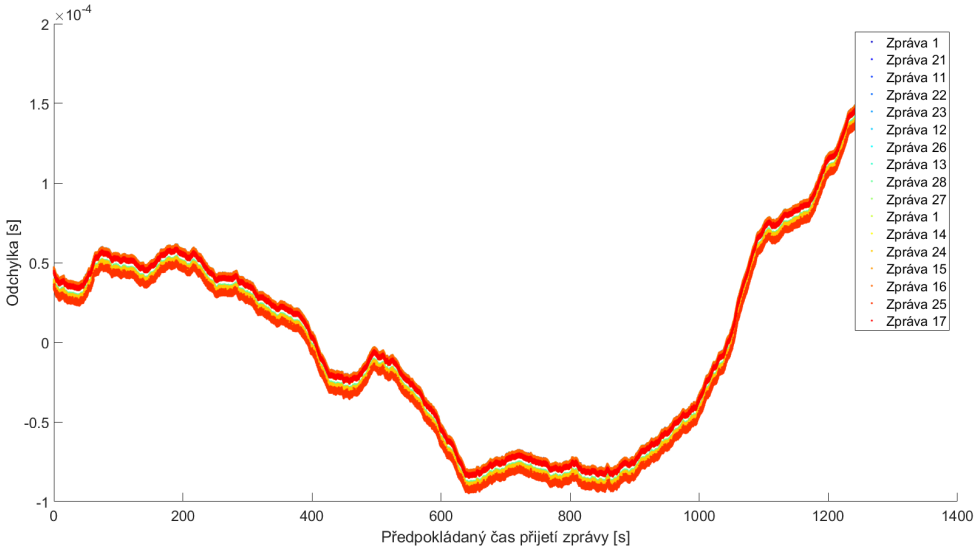
V rámci tohoto měření bylo využito přerušení v jehož rámci docházelo k přenastavení DMA na určitou část rozvrhu a následně ke škálování časové základny. Rozvrh na síti je zobrazen v tabulce 15:

Id zprávy	Čas odeslání [µs]	Jednotka
1	0	Master
21	750	Slave
11	2000	Master
22	2750	Slave
23	3550	Slave
12	4000	Master
26	4750	Slave
13	8000	Master
28	8550	Slave
27	9250	Slave
1	10000	Master
14	12000	Master
24	12750	Slave
15	14000	Master
16	16000	Master
25	12500	Slave
17	18000	Master

Tabulka 15 Rozvrh - dvě synchronizační zprávy

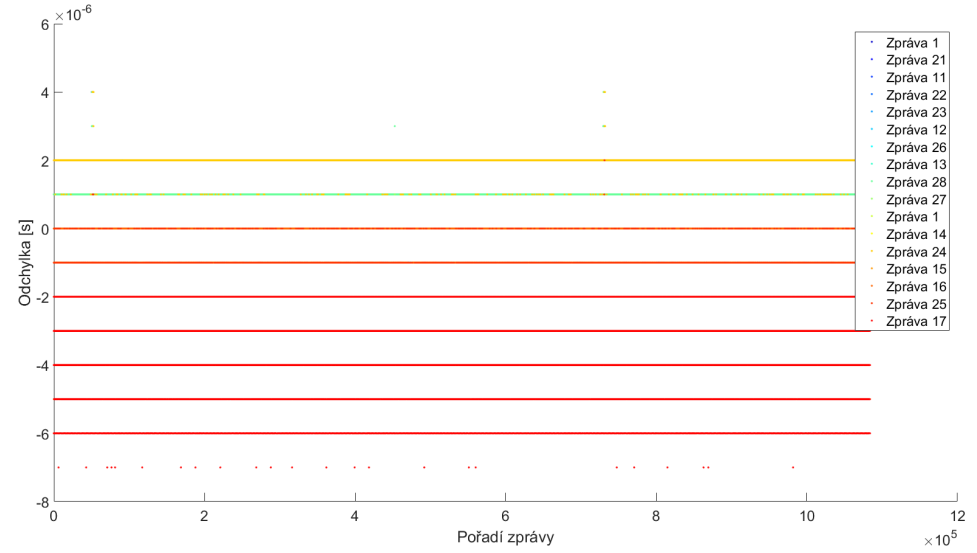
Rozvrh byl rozdělen na dvě části dlouhé 10 000 µs a celá perioda rozvrhu byla 20 000 µs.

V rámci měření bylo nejprve provedeno měření přesnosti synchronizace bez zahřívání jednotlivých jednotek komunikace a poté se zahřátím.



Obrázek 37 Synchronizace za využití přerušení

Na obrázku 37 je zobrazena odchylka od přepodkládaného času přijetí zprávy. Zpráva 1 se nachází v legendě grafu dvakrát, první je pro první synchronizační zprávu v čase 0 a druhá pro zprávu v čase 10 000 µs. V grafu opět narážíme na problém s rozdílností oscilátorů master jednotky a měřícího přístroje. V rámci grafu je patrné rozdělení na dvě úrovně přičemž dolní úroveň odpovídá slave jednotce a horní úroveň odpovídá master jednotce. Vznik dvou úrovní může znamenat, že spouštění vykonávání rozvrhu po přijetí synchronizační zprávy za pomocí přerušení je rychlejší než spouštění vykonávání rozvrhu za pomoci DMA, jež je použito ke spuštění rozvrhu za pomoci hardwarové synchronizace. V případě že vezmeme v úvahu jednotlivé periody a části rozvrhu zvlášť a tím vyeliminujeme rozdílnosti oscilátorů mezi jednotlivými periodami dostáváme graf na obrázku 38.



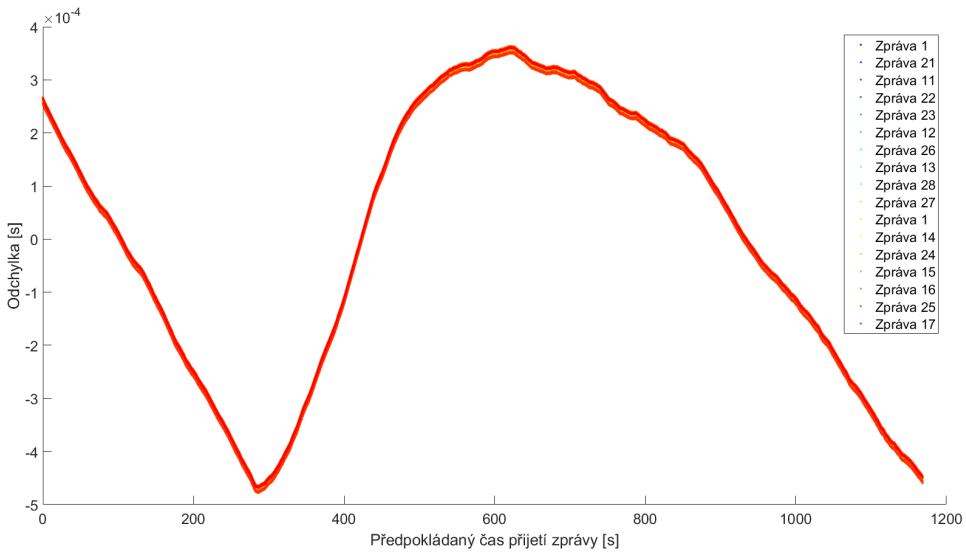
Obrázek 38 Synchronizace za využití přerušení po jednotlivých periodách

Vezeme-li absolutní časy v rámci každé části rozvrhu zvlášť, dostáváme tabulku 16.

Id zprávy	Minimální doba přijetí [µs]	Maximální doba přijetí [µs]	Rozdíl [µs]
1	0	0	0
21	748	754	6
11	1994	1998	4
22	2748	2754	6
23	3548	3554	6
12	3996	4000	4
26	4748	4752	4
13	7994	7998	4
28	8550	8554	4
27	9247	9252	5
1	0	0	0
14	1994	1998	4
24	2748	2754	6
15	3994	3998	4
16	5996	6000	4
25	7246	7252	6
17	7993	7998	5

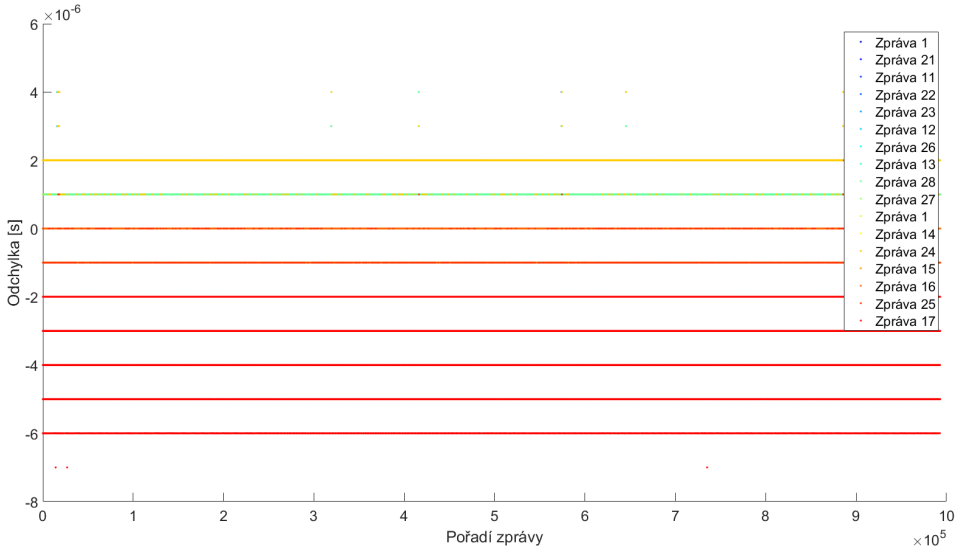
Tabulka 16 Srovnání doby přijetí

Z tabulky 16 je patrné, že existenci dvou úrovní způsobily chyby master jednotky, jež měla větší chybu, než v předchozích měřeních.



Obrázek 39 Synchronizace za využití přerušení při zahřátí

Na obrázku 39 je zobrazena chyba v případě, že po pěti minutách dojde k zahřátí master jednotky na dvě minuty a po dalších pěti minutách dojde k zahřátí slave jednotky. Nejvyšší dosažená chyba je 478 µs. Na obrázku 40 je znázorněna chyba v rámci jednotlivých period.



Obrázek 40 Synchronizace za využití přerušení při zahřátí po periodách

Vezeme-li absolutní časy v rámci každé části rozvrhu zvlášť, dostáváme tabulku 17. Jak je z tabulky patrné, tak zahřátí nemělo na chybu v rámci periody velký vliv, pouze u zpráv 12 a 28 došlo ke zhoršení o 1 µs.

Id zprávy	Minimální chyba [µs]	Maximální chyba [µs]	Rozdíl [µs]
1	0	0	0
21	748	754	6
11	1994	1998	4
22	2748	2754	6
23	3548	3554	6
12	3996	4001	5
26	4748	4752	4
13	7994	7998	4
28	8549	8554	5
27	9247	9252	5
1	0	0	0
14	1994	1998	4
24	2748	2754	6
15	3994	3998	4
16	5996	6000	4
25	7246	7252	6
17	7993	7998	5

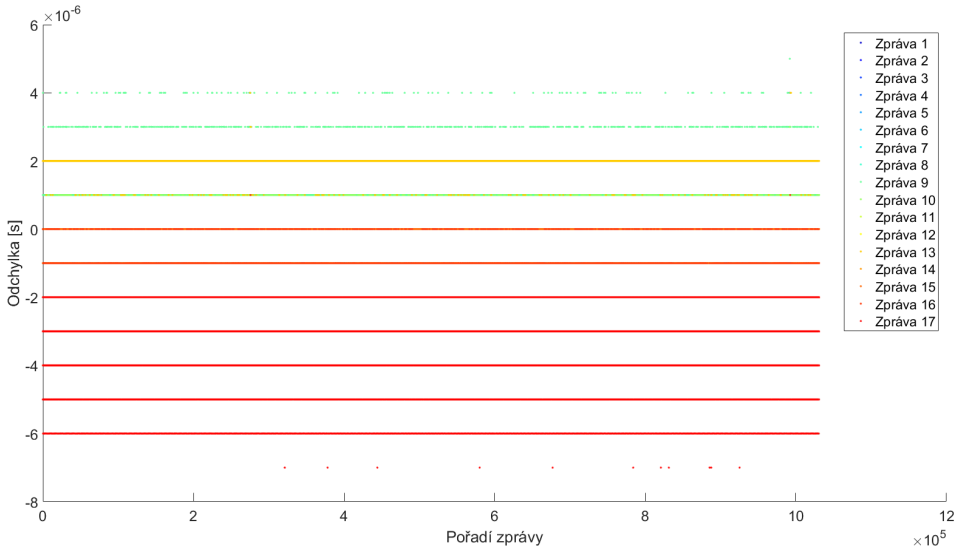
Tabulka 17 Srovnání doby přijetí - zahřátí

Z uvedených grafů a tabulek je jasně vidět, že chyba v rámci periody je malá, ale postupně se nasčítá. Problémem je rozdílná frekvence oscilátorů master jednotky a měřícího přístroje. V rámci jednotlivých period je chyba v řádu mikro sekund jak na master tak na slave jednotce. S ohledem i na předchozí měření se již dostáváme na limity daného hardwaru.

Synchronizace s využitím přerušení a škálováním časové základny

V rámci tohoto měření byl použit stejný rozvrh jako při měření bez škálování. S tím rozdílem že byl použit PI regulátor z kapitoly 3 Návrh. Nastavení parametrů regulátoru bylo: $k_i = 0.01$.

Pokud vezeme v úvahu chyby v rámci každé části rozvrhu zvlášť, dostáváme graf na obrázku 41.



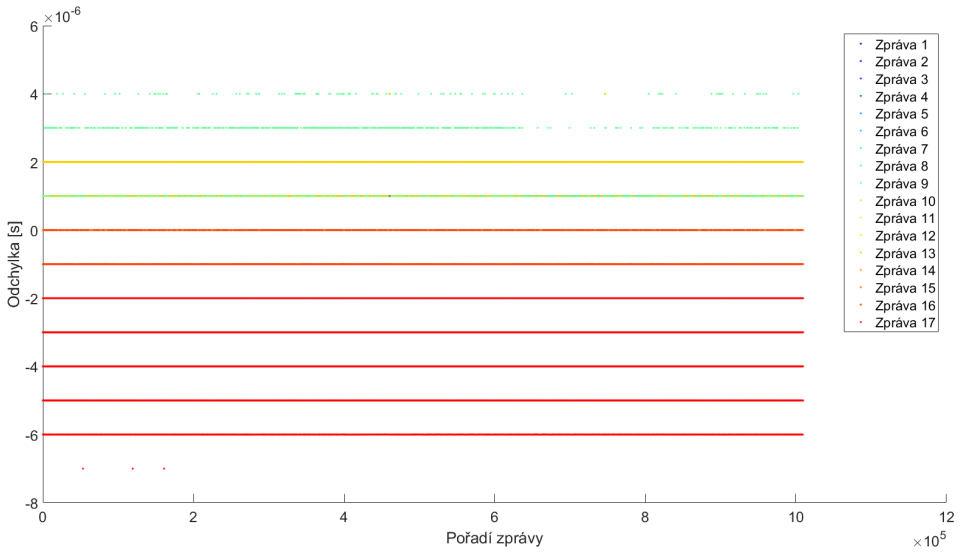
Obrázek 41 Synchronizace za využití přerušení a škálování

V následující tabulce 18 jsou uvedeny minimální a maximální hodnoty příjmu času příjmu zpráv v rámci periody.

Id zprávy	Minimální doba přijetí [µs]	Maximální doba přijetí [µs]	Rozdíl [µs]
1	0	0	0
21	748	754	6
11	1994	1998	4
22	2748	2754	6
23	3548	3554	6
12	3996	4000	4
26	4748	4754	6
13	7994	7998	4
28	8549	8555	6
27	9247	9253	6
1	0	0	0
14	1994	1998	4
24	2748	2754	6
15	3994	3998	4
16	5996	6000	4
25	7246	7251	5
17	7993	7998	5

Tabulka 18 Srovnání doby přijetí - škálování

Na obrázku 42 je znázorněna chyba pro jednotlivé zprávy, přičemž je každá část rozvrhu brána zvlášť.



Obrázek 42 Synchronizace za využití přerušení a škálování při zahřátí po periodách

V následující tabulce 19 jsou uvedeny minimální a maximální hodnoty příjmu času příjmu zpráv v rámci periody, při zahřátí master a slave jednotky.

Id zprávy	Minimální doba přijetí [µs]	Maximální doba přijetí [µs]	Rozdíl [µs]
1	0	0	0
21	748	754	6
11	1994	1998	4
22	2748	2754	6
23	3548	3554	6
12	3996	4000	4
26	4748	4754	6
13	7994	7998	4
28	8549	8554	5
27	9247	9253	6
1	0	0	0
14	1994	1998	4
24	2748	2754	6
15	3994	3998	4
16	5996	6000	4
25	7246	7251	5
17	7993	7998	5

Tabulka 19 Srovnání doby přijetí - škálování, zakřátí

Jak při srovnání tabulek 19, 18, 17 a 16 patrné tak při vzdálenosti mezi synchronizačními zprávami 10 000 µs, se vliv škálování snižuje rozcházení master a slave jednotky pouze o 1 µs a to i při zahřátí. Vliv teploty je tedy malý v pokud je mezi synchronizačními zprávami vzdálenost 10 000 µs a škálování časové základny není třeba.

Srovnání synchronizace se škálováním a bez škálování na delším rozvrhu

Jak je patrné z předchozích měření, tak pro vzdálenost mezi synchronizačními zprávami 10 000 µs nemá škálování velký vliv, proto bylo provedeno měření s větší vzdáleností mezi synchronizačními zprávami. Rozvrh na síti je zobrazen v tabulce 20:

Id zprávy	Čas odeslání [µs]	Jednotka
1	0	Master
11	18000	Master
21	19250	Slave

Tabulka 20 Dlouhý rozvh

Rozvrh obsahuje jednu synchronizační zprávu a celá periody rozvrhu je dlouhá 100 ms. Aby se ukázal vliv škálování je brán v úvahu změřený čas mezi zprávami 11 a 21. Jednu z pozorovaných zpráv vysílá master a druhou slave jednotka, takže lze pozorovat, jak se proti sobě obě jednotky posouvají. V tabulce 21 jsou uvedeny statistiky v případě postupného zahřátí obou desek.

Použito škálování	Min [µs]	Max [µs]	Průměr [µs]	Rozptyl [µs]
Ne	1250	1256	1252.021	1.521
Ano	1250	1256	1252.455	1.42

Tabulka 21 Statistiky vzdálednosti mezi zprávami

Jak je z naměřených hodnot patrné tak, vliv teploty je velmi malý a na jednotky nemá téměř vliv a škálování to již nemůže příliš zlepšit vzhledem k režii při odesílání a příjmu zpráv zjištěných při předchozích měřeních.

6 Závěr

Cílem práce bylo navrhnout a naimplementovat hardwarovou realizaci rozvrhu podle [1], umožňující časování komunikace a úloh za využití periferií nezávisle na hlavním procesoru obvodu TMS570LS1227ZWT. Dále bylo nutné vyřešit synchronizaci jednotek za předpokladu, že je jedna jednotka nadřízená. Práce je rozdělena do čtyř částí zabývajících se analýzou, návrhem, implementací a měřením/ověřením implementované funkcionality.

V rámci realizace byly použity periferie N2HET, DCAN, VIM a DMA. Odesílání zpráv bylo realizováno za pomoci periferií N2HET, DMA a DCAN. Časovač N2HET byl využit pro určení času odeslání zprávy a DCAN kontrolér byl použit pro odesílání zpráv. Jejich vzájemnou spolupráci zajišťuje DMA. Příjem zpráv byl řešen za použití stejných periferií jako odesílání zpráv. Pro přijetí zpráv byl použit DCAN kontrolér, konkrétně jeho třetí interface speciálně navržený pro spolupráci s DMA. Pro měření času příjmu zprávy byl použit časovač N2HET. Pro příjem a odesílání zpráv je možné použít jeden N2HET časovač a dva DCAN kontroléry. V rámci synchronizace je použita master and slave architektura. K synchronizaci jsou použity zprávy s ID 1 a datovou položkou obsahující informace, pro nastavení části rozvrhu, která se má na slave jednotce spustit. Nastavení části rozvrhu, která má být v daném cyklu na slave jednotce spuštěná, se využívá obslužné rutiny přerušení na slave jednotce. Pokud je synchronizační zpráva v rámci rozvrhu jenom jedna, tak je možné řešit i veškerou synchronizaci komunikace na slave jednotce bez zásahu procesoru. Spouštění funkcí je realizováno také časovačem N2HET který kromě měření času vyvolává požadavek na přerušení pro procesor, jež uvolní semafor blokující úlohu, jež provede vykonání odpovídající funkce.

V rámci měření je rozebráno odesílání zpráv, příjem zpráv a synchronizace. Z měření vyplývá že, vysílání zpráv podle rozvrhu probíhá s přesností lepší než 6 µs při neuvažování chyby oscilátoru. Při uvažování chyby oscilátoru v případě odstupu synchronizačních zpráv 10 ms je nutný odstup zpráv odesílaných různými jednotkami minimálně 12 µs. Vzhledem k tomu, že teplota má malý vliv na rozcházení master a slave jednotky a tomu že se dostáváme na limity daného hardwaru, tak škálování nemá velký efekt. Určení a zaznamenání času přijetí zprávy je možné s přesností lepší než 3 µs.

Budoucí možná rozšíření jsou popsána v kapitole zabývající se návrhem. Jedná se o rozšíření příjmu o možnost používat 29 bitové ID. Další možností rozšíření je odstranění přebytečných položek ze zpráv odesílaných v módu Data in message RAM a jejich nastavení na DCAN kontroléru již při inicializaci. Již nad rámec rozboru předloženého v této práci je ošetření situací, kdy úloha na procesoru není dokončená do vypršení rozvrhem předepsaného časového limitu.

Příloha A

Obsah přiloženého CD

Zde se nachází obsah přiloženého CD. Seznam souboru a adresářů. Včetně informací o tom, co v nich hledat.

- Implementace/ - Zdrojové kódy přidané funkcionality. Knihovna je vzhledem k licenci připojena pouze v binární podobě.
- Text/ - Tento dokument v PDF i se zdrojovým kódem systému LaTeX.
- readme.txt - Textový soubor s informacemi o disku.

Příloha B

N2HET program

B.1 Timer-WaitToSynchrMessageInSchedule.het

```
;Free running counter - before scaling
ST00   CNT { next=ST01,reg=NONE,max=0x1FFFFFF};
;Scaling accumulator
ST01   ADD { src1=T,src2=IMM,dest=T,next=ST02,data=16777216};
;If overflow occurs, continues in program
ST02   BR { next=ST00,cond_addr=L00,event=C};

;Free running counter - after scaling
L00   CNT { next=L01,reg=NONE,max=0x1FFFFFF};
;Main counter which measure time between two actions, after overflow sends request to DMA channel 0
L01   CNT { next=S00,reqnum=4,request=GENREQ,reg=A,max=0x5};

;CANTT_ACTION_TARGET_CAN
;If main counter overflows, sends request to DMA channel 2, which starts transmission of message from CAN 1
M00   BR { next=ST00,reqnum=5,request=GENREQ,cond_addr=ST00,event=Z};
;If main counter overflow, sends request to DMA channel 4, which starts transmission of message from CAN 2
M01   BR { next=ST00,reqnum=6,request=GENREQ,cond_addr=ST00,event=Z};

;CANTT_ACTION_TARGET_PIN_UP
;If main counter overflows, sets pin 21 to high
P00   ECMP { next=ST00,hr_lr=HIGH,en_pin_action=ON,cond_addr=ST00,pin=22, action=PULSEHI,reg=A,data=0};

;CANTT_ACTION_TARGET_PIN_DOWN
;If main counter overflows, sets pin 21 to low
P01   ECMP { next=ST00,hr_lr=HIGH,en_pin_action=ON,cond_addr=ST00,pin=22, action=PULSELO,reg=A,data=0};

```

```
27 ;CANTT_ACTION_TARGET_COPY_TIME
28 ;If main counter overflows, sends request to DMA channel 1, which copy value
   ↪ from Free running counter - after scaling into memory
29 T00   BR { next=ST00,reqnum=7,request=GENREQ,cond_addr=ST00,event=Z};
30
31 ;CANTT_ACTION_TARGET_WAIT_FOR_NOTIFICATION_MESSAGE
32 ;Clears counter
33 S00   MOV32 { next=S01,remote=4,type=REGTOREM,reg=S,data=0};
34 ;Loads received id from instruction into register B
35 S01   MOV32 { next=S02,remote=3,type=IMTOREG,reg=B,data=5};
36 ;Check if received message in register B have 11 bit ID with value 1
37 S02   SUB { src1=B,src2=IMM,dest=NONE,next=S03,data=16779264};
38 ;If received message was synchronization, sends interrupt request to VIM
39 S03   BR { next=ST00,reqnum=4,request=GENREQ,cond_addr=S04,event=Z,irq=ON};
40 ;Clears received message ID in instruction S01
41 S04   MOV32 { next=ST00,remote=11,type=REGTOREM,reg=S,data=1};
42
43 ;CANTT_ACTION_TARGET_INTERRUPT
44 ;If main counter overflows, sends interrupt request to VIM
45 I00   BR { next=ST00,cond_addr=ST00,event=Z,irq=ON};
```

B.2 Timer-WaitToSynchPerm.het

```
1  ;Free running counter - before scaling
2  ST00   CNT { next=ST01,reg=NONE,max=0x1FFFFFF};
3  ;Scaling accumulator
4  ST01   ADD { src1=T,src2=IMM,dest=T,next=ST02,data=16777216};
5  ;If overflow occurs, continues in program
6  ST02   BR { next=ST00,cond_addr=L00,event=C};
7
8  ;Free running counter - after scaling
9  L00   CNT { next=L01,reg=NONE,max=0x1FFFFFF};
10 ;Main counter which measure time between two actions, after overflow sends
   ↪ request to DMA channel 0
11 L01   CNT { next=S00,reqnum=4,request=GENREQ,reg=A,max=0x5};
12
13 ;CANTT_ACTION_TARGET_CAN
14 ;If main counter overflows, sends request to DMA channel 2, which starts
   ↪ transmission of message from CAN 1
15 M00   BR { next=S01,reqnum=5,request=GENREQ,cond_addr=S01,event=Z};
16 ;If main counter overflow, sends request to DMA channel 4, which starts
   ↪ transmission of message from CAN 2
17 M01   BR { next=S01,reqnum=6,request=GENREQ,cond_addr=S01,event=Z};
18
19 ;CANTT_ACTION_TARGET_PIN_UP
20 ;If main counter overflows, sets pin 21 to high
21 P00   ECMP { next=S01,hr_lr=HIGH,en_pin_action=ON,cond_addr=ST00,pin=22,
   ↪ action=PULSEHI,reg=A,data=0};
22
23 ;CANTT_ACTION_TARGET_PIN_DOWN
24 ;If main counter overflows, sets pin 21 to low
25 P01   ECMP { next=S01,hr_lr=HIGH,en_pin_action=ON,cond_addr=ST00,pin=22,
   ↪ action=PULSELO,reg=A,data=0};
26
27 ;CANTT_ACTION_TARGET_COPY_TIME
28 ;If main counter overflows, sends request to DMA channel 1, which copy value
   ↪ from Free running counter - after scaling into memory
29 T00   BR { next=S01,reqnum=7,request=GENREQ,cond_addr=S01,event=Z};
30
31 ;CANTT_ACTION_TARGET_WAIT_FOR_NOTIFICATION_MESSAGE
32 ;Clears counter
33 S00   MOV32 { next=S01,remote=4,type=REGTOREM,reg=S,data=0};
```

```
34 ;Loads received id from instruction into register B
35 S01   MOV32 { next=S02,remote=3,type=IMTOREG,reg=B,data=5};
36 ;Check if received message in register B have 11 bit ID with value 1
37 S02   SUB { src1=B,src2=IMM,dest=NONE,next=S03,data=16779264};
38 ;If received message was synchronization, sends interrupt request to VIM
39 S03   BR { next=ST00,reqnum=4,request=GENREQ,cond_addr=S04,event=Z,irq=ON};
40 ;Clears received message ID in instruction S01
41 S04   MOV32 { next=ST00,remote=11,type=REGTOREM,reg=S,data=1};
42
43 ;CANTT_ACTION_TARGET_INTERRUPT
44 ;If main counter overflows, sends interrupt request to VIM
45 I00   BR { next=S01,cond_addr=S01,event=Z,irq=ON};
```

Literatura

[1] Hanzalek Z. Minaeva A. Akesson B. a Dasari D. *Time-Triggered Co-Scheduling of Computation and Communication with Jitter Requirements*. 2017.

[2] *ISO 11898-4 Road vehicles - Controller area network (CAN) - Time-triggered communication*. 1. srp. 2004.

[3] *TMS570LS12x/11x 16/32-Bit RISC Flash Microcontroller Technical Reference Manual*. URL: `http://www.ti.com/lit/ug/spnu515b/spnu515b.pdf` (cit. 13. 05. 2017).

[4] Sojka M. Jenkin C. *Code generation for automotive rapid prototyping platform using Matlab/Simulink, RPP project documentation*. 2014.

[5] Jerabek M. *FPGA Based CAN Bus Channels Mutual Latency Tester and Evaluation*. URL: `http://rtime.felk.cvut.cz/can/F3-BP-2016-Jerabek-Martin-Jerabek-thesis-2016.pdf` (cit. 13. 05. 2017).

[6] *HET IDE Tutorials, User's Guide*. URL: `http://www.ti.com/lit/ug/spnu485c/spnu485c.pdf` (cit. 13. 05. 2017).

[7] *FreeRTOS documentation*. URL: `http://www.freertos.org/RTOS.html` (cit. 13. 05. 2017).

[8] *TMS570LS1227 16- and 32-Bit RISC Flash Microcontroller*. URL: `http://www.ti.com/lit/ds/symlink/tms570ls1227.pdf` (cit. 13. 05. 2017).

[9] *System Performance Improvements by Leveraging the High-End Timer Transfer Unit on Hercules ARM Safety MCUs*. URL: `http://www.ti.com/lit/an/spna130a/spna130a.pdf` (cit. 13. 05. 2017).

[10] *Sine Wave Generation Using PWM With Hercules N2HET and HTU*. URL: `http://www.ti.com/lit/an/spna217/spna217.pdf` (cit. 13. 05. 2017).

[11] *How to Create A HALCoGen-Based Project For CCS*. URL: `http://www.ti.com/lit/an/spna121b/spna121b.pdf` (cit. 13. 05. 2017).