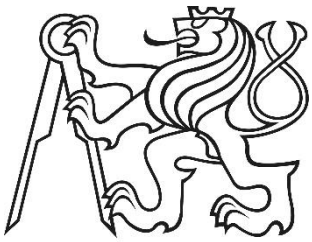
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra mikroelektroniky



Návrh rozšíření existujícího řadiče CAN o standard FD (flexible data rate)

Extension of existing CAN controller for FD standard

Jméno: Bc. Karel Fitz

Vedoucí práce: Doc. Ing. Jiří Jakovenko Ph.D.

Praha 2017

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Fitz** Jméno: **Karel** Osobní číslo: **383032**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra mikroelektroniky**

Studijní program: **Komunikace, multimédia a elektronika**

Studijní obor: **Elektronika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Návrh rozšíření existujícího řadiče CAN o standard FD

Název diplomové práce anglicky:

Extension of the Existing CAN Controller for FD Standard

Pokyny pro vypracování:

1) Prostudujte problematiku a specifikaci řadiče CAN 2.0 a CAN FD (flexible data rate)
2) Navrhněte vylepšení jednotlivých funkčních bloků existujícího řadiče CAN o rozšíření FD standardu s využitím jazyka VHDL
3) Realizované úpravy ověřte simulací v programu NC Sim
4) Funkční ověření proveďte na hradlovém poli FPGA
5) K funkčnímu ověření využijte externí mikropočítač připojený sběrnicí AHB lite.

Seznam doporučené literatury:

[1] ISO 11898-1:2015 Road vehicles. Controller area network (CAN). Data link layer and physical signalling, ISBN 978-0580830303
[2] CAN Specification Version 2.0. Robert Bosch GmbH, 1991

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jiří Jakovenko Ph.D., katedra mikroelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **03.02.2017** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **10.09.2018**

Podpis vedoucí(ho) práce	Podpis vedoucí(ho) ústavu/katedry	Podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání	Podpis studenta
24.5.2017	

Čestné Prohlášení

Prohlašuji, že jsem tuto diplomovou prací vytvořil samostatně, a to jen s pomoci vedoucího práce a konzultantů. K vyhotovení jsem použil pouze literaturu uvedenou v seznamu referencí a vědomosti, kterých jsem nabyl v průběhu studia.

V Praze dne 26.5 2017 …….

podpis

Poděkování

Chtěl bych poděkovat panu Doc. Ing. Jiřímu Jakovenkovi Ph.D., vedoucímu mé diplomové práce, za vedení, cenné rady, připomínky a také za čas, který mi věnoval. Dále bych chtěl poděkovat zúčastněným zaměstnancům společnosti ASICentrum za cenné, praktické rady v průběhu řešení obtíží a otázek, na které jsem v průběhu práce narazil, stejně jako za rady z praxe, které jsou postaveny na zkušenostech. Mé poděkování rovněž patří mé rodině a přátelům za podporu a pomoc během studia.

Abstract

The CAN FD controller is a device controlling the data transmission (communication) on CAN-bus. It stores received messages, transmits messages and communicates with the host system, which is usually a micro-controller.

At first it is necessary for successful development of this project to be familiar with CAN-bus protocol specification 2.0B. At second it is needed to be familiar with the last version of CAN FD from ISO 11898 document released on the end of 2015.

This work is divided into 2 parts. The first part is focused on theoretical explanation of all necessary information. These are the steps and rules of IC development, basics of standard CAN 2.0 and standard of CAN FD. In the second part are described all steps, which are necessary to achieve the main goal of this thesis, which is an extended CAN driver working with both standards CAN 2.0 and also with CAN FD. At the end of this document there are results of simulation and also verification, which are made by using FPGA board and industrial CAN analyzer. These results prove that the designed CAN driver works fine according to the standard and specification.

Keywords: *bus driver, CAN bus, CAN FD, FPGA, Integrated Circuit, Digital Design*

Anotace

CAN FD kontrolér je zařízení řídící výměnu dat a komunikaci na CAN sběrnici. Toto zařízení ukládá relevantní přijaté zprávy, vysílá zprávy a také komunikuje s řídícím systémem, kterým je ve většině případu mikrokontrolér.

Pro úspěšný vývoj tohoto projektu je nezbytné se nejprve seznámit s protokolem sběrnice CAN. Toto nejprve pro případ protokolu CAN 2.0. Následně po detailním pochopení je potřeba se seznámit se standardem CAN FD z konce roku 2015.

Tato práce je dělena do dvou částí. V první části práce je věnován prostor teoretickému rozboru, zejména problematice postupu návrhu integrovaných obvodů, dále pak je věnován prostor problematice standardu CAN 2.0 a jejího přiblížení. Na závěr první časti je prostor věnovaný standardu CAN FD. Druhá část práce je věnována popisu realizace, tedy popisu všech potřebných kroků pro realizaci rozšíření exitujícího řadiče CAN o standard FD. Tato část zahrnuje také popis simulační části a následně i validace. Na závěr je shrnuta validace, která byla provedena měřením, ale i průmyslovým analyzátorem. Tato validace je pak důkazem správné funkce navrženého řadiče.

Klíčová slova: *řadič, sběrnice CAN, CAN FD, FPGA, HDL, integrovaný obvod, digitální návrh*

Obsah

ÚVOD

Tato práce obsahuje nezbytné informace a popis kroků, které jsou potřebné pro vytvoření rozšíření existujícího řadiče CAN („Controller Area Network“) o standard FD („Flexible Data-Rate“), který je určen pro FPGA („Field Programmable Gate Array” – programovatelné hradlové pole). Vše je provedeno tak, aby funkce odpovídala dokumentu ISO 11898. Toto rozšíření je aplikováno na dodaný existující řadič, pracující dle BOSCH specifikace CAN 2.0, který je navržený pro práci s 8 bitovým procesorem.

Práce je rozdělena do dvou částí, v první části je věnována pozornost teorii návrhu integrovaných obvodů, teorii okolo standardu CAN 2.0 a také standardu CAN FD. Druhá část se zabývá realizací řadiče CAN na základě dodaného existujícího řadiče, který pracuje dle specifikace CAN 2.0. V této části je věnován prostor také výsledkům simulací. Na závěr je dokumentována validace a validace návrhu s následným shrnutím a diskuzí výsledků.

Hlavním cílem teto práce je návrh a následný test digitálního bloku kontroléru CAN FD, psaném v jazyce VHDL („VHSIC Hardware Description Language“). Toto bude vytvořeno na základě existujícího dodaného řadiče (2.0) s 8 bitovým rozhraním psaném v jazyce VHDL. Rozhraní je obměněno s použitím komunikačního protokolu AHB-Lite („Advanced High performance Bus - Lite“). Ověření funkce je provedeno nejprve formou simulace v prostředí NCSim a následně je návrh validován na FPGA desce měřením a také pomocí průmyslového analyzátoru.

Jelikož je CAN standard v praxi hojně využíván, dosahují dnes v mnoha použitích sběrnice založené na standardu CAN 2.0 horních limitních parametrů z pohledu objemu přenesených dat za jednotku času. Z tohoto důvodu byl vyvinut standard CAN FD. Motivací se stalo zdokonalení existujícího řadiče na takovou úroveň, kdy bude schopen provozu v obou variantách a bude tak umožněn vyšší objem přenesených dat na sběrnici mezi zařízeními a rovněž kompatibilita s novými i starými zařízeními. Umožní to například tvorbu rychlejších systémů, ve kterých řídící členy dostanou odezvu na událost rychleji a s možností mnohem většího počtu podrobností a informací, které může nový CAN FD obsáhnout v jedné zprávě.

1 Teoretický rozbor

První část této kapitoly je věnována rozboru problematiky návrhu integrovaných obvodů, která byla rozdělena dle chronologických fází návrhu. V druhé části kapitoly je vyčleněn prostor pro popis BOSCH specifikace CAN 2.0. V poslední kapitole této části je popsán CAN Flexible Data-Rate dle ISO dokumentu.

1.1 Postup a pravidla návrhu integrovaných obvodů

Specifikace vlastností

Specifikace obsahuje souhrn vlastností a parametrů systému. Tento souhrn vytváří svými požadavky zadavatel projektu a designer/vývojář pak vychází z těchto faktů při návrhu. Specifikace je většinou odvozena od normy týkající se problematiky nebo také může vycházet z programu ve vyšším programovacím jazyce jako je například C++. [9]

Systémový a logický návrh

V této části návrhu je stanovena hierarchie, uspořádání a vzájemné propojení bloků. V praxi se tato část odvíjí od zkušeností návrhářů, jelikož míra dokonalosti je závislá na množství jejich zkušeností a úsudku.

Pro logický návrh se dříve používala grafická metoda, tedy kreslení schémat logických hradel a jejich propojení. Tento způsob se dnes prakticky nevyužívá. Je to hlavně vlivem míry obtížnosti a složitosti navrhovaných systémů. Dnes se pro návrh používá paralelní popisovací jazyk HDL (Hardware Description Language), a to pro všechny úrovně složitostí. Nejpoužívanější jsou jazyky Verilog a VHDL, ve kterém je také navrhován a realizován systém řadiče CAN.

Simulace funkce

Nezbytnou součástí je ověření funkce navrženého systému. Jde tedy o ověření správné funkce logického návrhu pomocí simulací. V jednodušších případech postačí zobrazení signálu v simulátoru a v jeho grafickém prostředí. Ve složitějších případech je potřeba vytvořit takzvaný „self-check Test Bench“, což je ve své podstatě emulátor reálných podmínek prostředí navrhovaného systému, který se stará i o kontroly sama sebe. Tento je vytvářen také v jazyce HDL.

Syntéza, rozmístění a propojení

Syntéza je automatický proces převádějící logický návrh na síť logických hradel se správným propojením. Před provedením syntézy je třeba dbát návrhových pravidel a také zvolit určitá nastavení této syntézy. Jde o volbu cílové technologie, tedy programovatelné obvody FPGA („Field Programmable Gate Array“ - programovatelné hradlové pole), CPLD („Complex Programmable Logic Device“) nebo cílovou technologii pro zákaznický obvod ASIC. Výsledný

syntetizovatelný návrh je nazýván RTL („Register - Transfer Level“). V případě programovatelných obvodů je třeba převod logických hradel na reálné propojení hradel daného programovatelného obvodu. Tento proces je nazýván „place and route“.

Verifikace

Po provedení syntézy je další v pořadí opětovná kontrola funkce, přičemž v tuto chvíli je již do simulací zahrnuto reálné časové zpoždění hradel a propojovacích vodičů mezi hradly. V případě kdy tyto simulace odpovídají simulacím před syntézou a splňuji požadavky specifikace je přistoupeno k validaci.

Validace

Validace je poslední krok v procesu návrhu integrovaných obvodů. Jde o reálné ověření funkce systému po vložení výsledné syntézy do cílové technologie. Pod pojmem validace si lze obecně představit kontrolu, tedy například měřením s porovnáním naměřených výsledku s výsledky simulací a specifikací.

1.2 Jazyk Verilog

Verilog je jeden z jazyků pro návrh a modelování integrovaných obvodů. Jedná se o jazyk HDL (Hardware Description Language). Tento jazyk umožnuje design, verifikaci a realizaci analogových, digitálních nebo smíšených obvodů s různou úrovní abstrakce. Jedním z cílů návrhářů tohoto jazyka byla co největší podobnost syntaxe na programovací jazyk C.

1.3 Jazyk VHDL

Druhý ze dvou nejrozšířenějších a nejpoužívanějších jazyků pro popis hardware, je jazyk VHDL. Tento jazyk spadá také do takzvané HDL (Hardware Description Language) kategorie.

Zkratka VHDL je zkráceninou „Very High Speed Integrated Circuits Hardware Description Language“. Jazyk VHDL byl původně vyvinut na ministerstvu obrany USA za účelem řízení paralelních procesů. Jazyk VHDL byl později standardizován pro paralelní návrh integrovaných obvodů, konkrétně v roce 1987 s označením IEEE Std 1076-1987, později IEEE 1076-2008. [7]

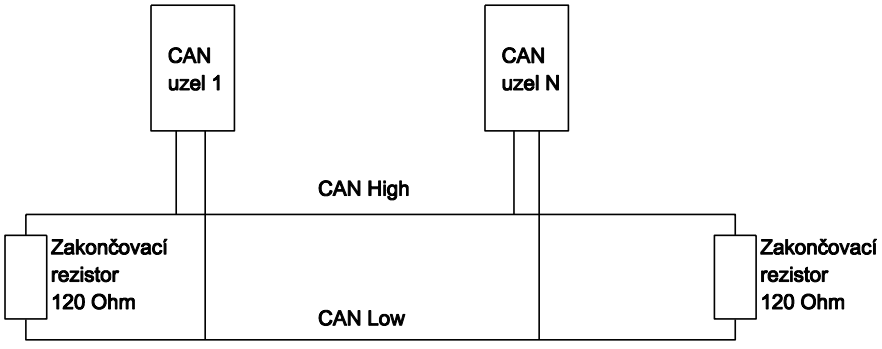
Výhodou jazyka je nezávislost návrhu na cílové technologii. Další vlastností jsou 2 typy procesů, kterými je možné popsat chování úloh a zejména způsob jejich provedení. Jde o Paralelní a Sekvenční způsob vykonání úlohy, jinak řečeno procesy souběžné (paralelní vykonání) a postupné (sekvenční vykonání). [9]

1.4 CAN Standard

CAN („Controller Area Network“) je sériový protokol vytvořený pro co nejbezpečnější komunikaci v reálném čase tak, aby byl schopen udržet velmi nízkou procentuální hodnotu chybovosti, a to i v zarušeném prostředí. Sběrnice byla navržena jako multimaster, což umožňuje vzájemnou komunikaci například mikrokontrolérů a jiných zařízení bez použití řídícího zařízení či počítače. Sběrnice je využívána nejčastěji pro vzájemnou komunikaci vnitřních řídících jednotek či sítě různých senzorů v automobilu. Sběrnice CAN se také používá například v oblasti průmyslové automatizace. Díky jeho nezávislosti na médiu a velmi vysoké spolehlivosti i v těch nejhorších přenosových podmínkách je dnes velmi rozšířenou sběrnicí. Jako další příklady použití lze zmínit například zemědělské stroje, autobusy, nákladní automobily, kombajny ale i lodě a další.

Sběrnice CAN byla vyvinuta firmou Robert Bosch GmbH a oficiálně představena v roce 1986 takzvaným společenstvím Automobilových inženýrů (Society of Automotive Engineers - SAE) na konferenci v Detroitu v USA. První CAN kontrolér byl vyroben o rok později společností Intel a Philips. O další rok později, tedy 1988 již byla CAN sběrnice poprvé použita v Automobilu BMW řady 8.[9]

Bosch již publikoval více verzí CAN specifikace. První byla CAN 2.0 v roce 1991. Tato je dělena na více částí. První se označuje jako CAN 2.0A a vyznačuje se použitím 11bitového identifikátoru. Verze CAN 2.0B pak používá identifikátor o délce 29bitů. V roce 1993 pak mezinárodní organizace pro standardizaci (ISO) vydala k této problematice ISO dokument pod označením ISO 11898. Dalším publikovaným CAN protokolem od Bosch byl CAN FD nebo Flex Data-Rate. Následně bylo zjištěno, že protokol byl sice vydán, ale obsahuje chyby. Tyto chyby pak byly eliminovány ve vydaném standardizovaném ISO 11898 dokumentu na konci roku 2015. Tuto chybu našla právě organizace ISO při procesu standardizace protokolu od firmy BOSCH, za účelem vydání nového standardizovaného ISO dokumentu. CAN FD se vyznačuje jinou skladbou rámce, která umožňuje přenos většího počtu datových bajtů a také možnost přepnutí na vyšší bitovou rychlost v oblasti takzvané datové fáze. CAN FD je vytvořen tak, aby byl schopen koexistovat na sběrnici s řadiči dle staršího standardu.



obr. 1 Principiální uspořádání sítě CAN [4]

Principiální uspořádání sítě CAN dle normy IEEE 11898 je vidět na obr. 1. Sběrnici je nutno na obou koncích ukončit ukončovacím odporem. Tyto odpory slouží jako zabezpečení

proti odrazům, tedy jako impedanční přizpůsobení. Jejich hodnota je stanovena na 120 Ohm, kde pro maximální rychlost 1Mbit/s je uvažována maximální délka vedení 40m. V ideálním případě lze ke sběrnici připojit libovolné množství uzlů/stanic. Z důvodu zachování použitelných dynamických ale i statických vlastností sběrnice je třeba stanovit konečný počet, který může být například 30 stanic.

1.5 CAN 2.0

Protokol je definován normou ISO 11898, který popisuje fyzickou vrstvu a specifikaci CAN 2.0A. Specifikace 2.0B, která byla vytvořena později má jisté odlišnosti, které byly udělány za účelem umožnění funkce ve větších sítích. Rozdíly jsou ve fyzické vrstvě a také v linkové vrstvě. Specifikace 2.0B implementuje názvy jako standardní a extended identifikátor, kde rozdíl mezi nimi je v bitové délce. Standardní identifikátor je kratší než identifikátor rozšířený/extended.

Jak už bylo zmíněno, protokol je vytvořen jako multi-master což znamená, že zde neplatí žádná pravidla "Master - Slave" ale jen nastaveni priorit pro Arbitraci. Toto má za následek, že žádný chybový uzel (uzel, který vykazuje chyby) neovlivní ostatní uzly připojené na sběrnici, které jsou v pořádku.

Specifikace 2.0 stanovuje 2 komplementární úrovně na sběrnici: dominantní a recesivní. Tyto mohou byt interpretovány jako logická 0 pro dominantní úroveň a logická 1 pro úroveň Recesivní. Pravidlo "agresivní nuly" říká, že v případě jednoho uzlu, který vyšle dominantní úroveň na sběrnici je hodnota na sběrnici, navzdory vysílání jakéhokoliv počtu recesivních hodnot jakýmikoliv uzly, stažena k nule, tedy na dominantní hodnotu.

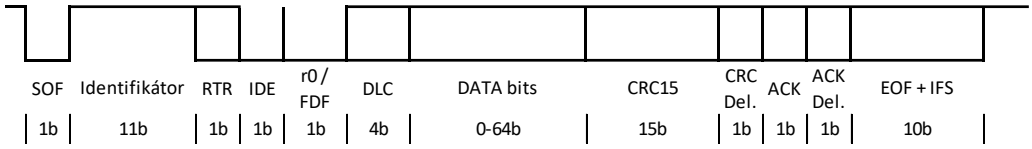
Všechny zprávy lze rozdělit do 4 druhů, kde první pár je pro přijímání a vysílaní zprávy, druhý pár je pak pro síťový management/řízení. Zprávy jsou přijímány všemi připojenými uzly, které jsou v módu přijímač přítomny na sběrnici. Součástí každé zprávy je identifikátor, který obsahuje smysl zprávy. Ten je využit na přijímačích k rozhodnutí, zda má zprava pro daný uzel smysl, tedy jestli je určena pro něj a má být uložena a následně vyhodnocována, nebo nikoliv. Toto je zkráceně nazváno jako filtrace. V případě kolize dvou zpráv vysílaných současně je za pomoci arbitrace zaručena volba zprávy s vyšší prioritou, která bude vysílána jako první.

1.5.1 Druhy zpráv

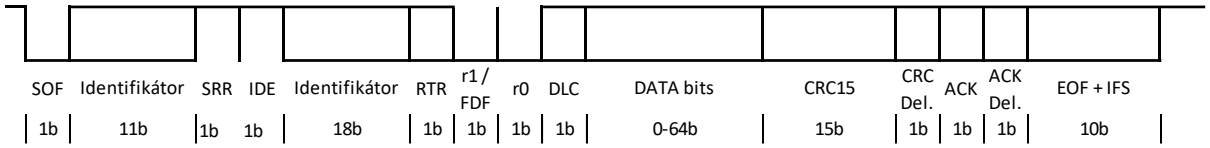
Jak již bylo zmíněno, uzly mezi sebou komunikují pomocí 4 druhů zpráv. Díky vynalézavému bitovému uspořádaní je umožněno uzlům pracujícím dle starší specifikace 2.0A koexistovat na sběrnici s uzly pracujícími dle novější specifikace 2.0B. Uzly s 2.0B specifikaci mohou však komunikovat se staršími uzly na 2.0A pouze pomocí zpráv se standardním formátem (standardní identifikátor).

Existují: Data Frame (datový rámec), Remote frame (žádost o data), Error Frame (chybový rámec), Overload Frame (rámec přetížení) a rámec Interframe space.

Datový rámec dle standardu CAN 2.0 A (Identifikátor = 11bitů)

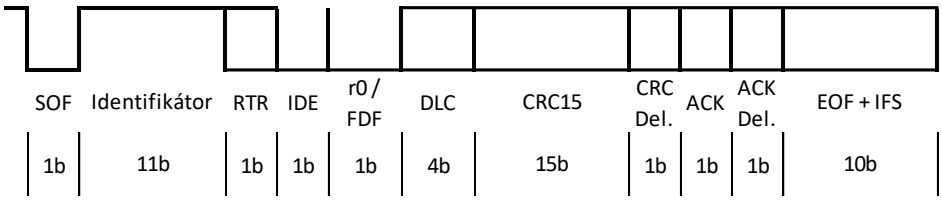


A) **Datový rámec dle standardu CAN 2.0 B (Identifikátor = 29bitů)**



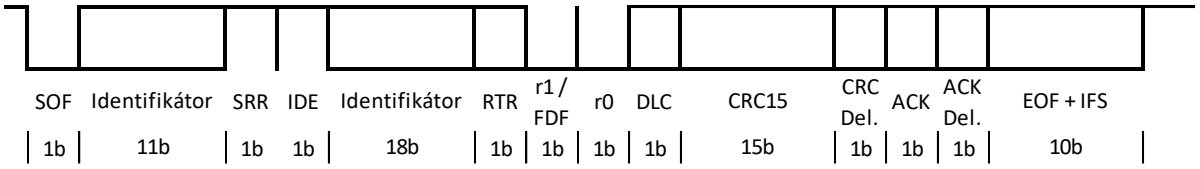
obr. 2 Datový rámec dle specifikace CAN 2.0

Remote rámec dle standardu CAN 2.0 A (11bit identifikátor)



B)

Remote rámec dle standardu CAN 2.0 B (Identifikátor = 29bitů)



obr. 3 Remote rámec dle specifikace CAN 2.0

1.5.2 Kontrola přístupu

Přenosová pravidla jsou dle multi-master systému jednoduchá a jasna. V případě, že uzel začne vysílat a těsně potom chce začít vysílat uzel druhy, tak první uzel vyhrává sběrnici pro sebe a všechny ostatní uzly musí "naslouchat", tedy být v přijímacím módu.

Existuje případ, kdy dva uzly začnou vysílat v tu samou chvíli. Nejprve je třeba zmínit, že každý uzel kontroluje hodnotu vyslanou na sběrnici s hodnotou, která na sběrnici opravu je. Řešení této situace je zaručeno arbitrací, a sice zpráva s nižší hodnotou identifikátoru vyhrává arbitraci. Když si situaci přiblížíme, nastane stav, kdy druhý uzel detekuje na sběrnici dominantní hodnotu, zatímco vyslal hodnotu recesivní, ihned potom se uzel přepíná do přijímacího módu, ve kterém setrvává, až dokud není zprava úspěšně poslána. Pak má druhý uzel další možnost získat pro sebe sběrnici a následně vyslat zprávu.

1.5.3 Zabezpečení přenášené zprávy a detekce chyb

Pro dosažení maximálního zabezpečení přenosu jsou implementovány účinné mechanismy v každém CAN uzlu. Všechny pak spolu koexistují.

Detekce chyby:

- Monitoring (vysílací uzel kontroluje vyslanou hodnotu s hodnotou, která na sběrnici opravdu je)
- Cyklická redundantní kontrola - CRC
- Bit stuffing
- Kontrola rámců zpráv

Jako první lze zmínit **Bit stuffing**. Tento mechanismus vkládá ve vysílacím uzlu komplementární bit („Stuff bit - SB“) za každou detekovanou posloupností 5ti po sobě jdoucích stejných bitů. Jakmile uzel detekuje absenci stuff bitu na místě, kde má SB být, začne ihned generovat chybovou zprávu „Stuff Error“. Bit-stuffing je použit pouze u pole Arbitrace, Kontrolního pole, Datového pole a CRC sekvence. Sestupné hrany, které těmito stuff bity vznikají (z Recesivní úrovně na Dominantní) jsou použity pro synchronizaci uzlu.

Jiný mechanismus je pak neustálá kontrola vysílacího uzlu, zda vyslaná hodnota souhlasí s hodnotou, která je na sběrnici. V případě nerovnosti je generována chybová zprava **BIT Error.** Výjimkou jsou pole jako ACK, nebo pole Arbitrace při vysílání.

Form Error chyba je generována v případě, kdy není dodržena rámcová struktura. Toto může nastat v oddělovacích polích v tzv. delimitrech, které musejí mít vždy recesivní hodnotu bitu.

Další kontrolní mechanismus je založen na CRC sekvenci, která je výsledkem kalkulace vysílacího uzlu a je použita v každé zprávě. Přijímající uzel vypočítává svou vlastní CRC sekvenci na základě přijatých hodnot zprávy s následným porovnáním s CRC sekvencí vysílače, jenž je ve zprávě. V případě neshody je generována chyba CRC error. Bližší popis CRC sekvence je následující: rámcová sekvenční kontrola je odvozena od cyklického redundantního kódu, který je ideální pro použití rámců s počtem bitu menším než 127 bitu. Fundamentální část CRC výpočtu je polynom, který je dělen a následně nazýván v anglické literatuře “polynomial”. Koeficienty jsou dány dělením bitové posloupnosti (zbavené tzv. STUFF bitů) z vybraných polí, tedy “Start of Frame”, “Arbitration Field”, “Control Field“ a také z pole Data Field.

$$X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1$$

Tento polynomial je dělen generátor – polynomem [2]:

Zbytek po dělení tímto polynomem je potřebná CRC sekvence, která je vyslána vysílačem a kontrolována na přijímači, zda je vypočtená hodnota na přijímači shodná s přijatou CRC sekvencí.

Na konci každé zprávy vysílač odesílá recesivní ACK bit a očekává dominantní hodnotu bitu na sběrnici, kterou vkládají přijímače v případě, že zprávu přijali v pořádku. V případě že k tomuto nedochází, tedy vysílač detekuje pouze recesivní hodnotu v poli pro ACK bit, je to známka nepřijetí zprávy přijímačem a je generována chybová zpráva ACK Error. Poté se vysílání opakuje.

1.5.4 Detekce chyb a jejich signalizace

Každá zjištěná chyba je signalizována vysíláním specifické chybové zprávy okamžitě po detekci. Výjimkou je CRC chyba, která je posílána až po ACK oddělovači. Každý uzel má 2 čítače pro definici v jakém stavu se uzel nachází, ty závisí na chybách. Čítače jsou pojmenovány jako "Transmit Error Counter" – TEC a "Receive Error Counter" – REC, tedy čítač chyb vysílače (TEC) a čítač chyb přijímače (REC). Jakmile je detekována chyba, je inkrementován příslušný čítač. Naopak při úspěšném odeslání či úspěšném přijetí zprávy je příslušný čítač dekrementován.

Existuji 3 stavy uzlů, ve kterých se mohou nacházet v závislosti na chybách, respektive na chybových čítačích. V případě, že je chybový čítač pod hodnotou 128, nachází se uzel v **Aktivním** módu. Jakmile čítač překročí hodnotu 128, je stav uzlu změněn na **Pasivní.** Uzel se může z pasivního modu vrátit do módu aktivního, ale jen v případě splnění podmínky poklesu hodnoty čítače pod hodnotu 128. Naopak v případě, že hodnota chybového čítače vzroste nad hodnotu 256, je uzel přepnut do stavu **Buss-OFF** a odpojuje se od sběrnice. Aby se uzel stal opět aktivní ze stavu Buss-OFF musí oba čítače být nulové a musí být detekováno 128 výskytů posloupnosti 11ti po sobě jdoucích recesivních bitu na sběrnici.

1.5.5 Synchronizace a časováni

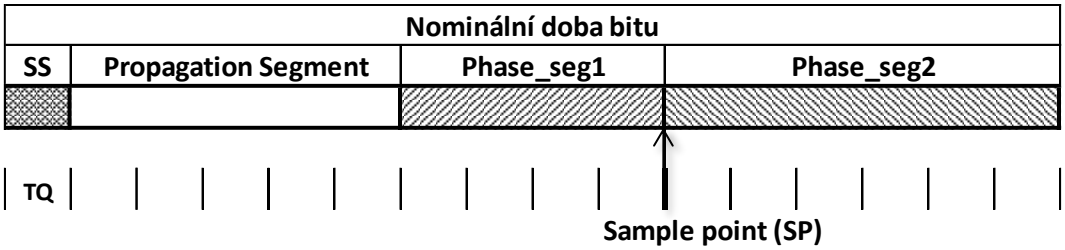
Dle specifikace CAN 2.0 existuji 2 typy synchronizace. První je nazvána „**Hard synchronizace"**. Ta je prováděna vždy na počátku rámce, a to na sestupnou hranu, která se označuje jako tzv. SOF (Start of Frame) tedy při změně z recesivní na dominantní hodnotu. To ale jen v případě, že předchozí stav sběrnice byl „buss-free", tedy volná sběrnice. Druhý typ synchronizace je nazván „**Resynchronizace"**. Tato resynchronizace se provádí na každou spadovou hranu v průběhu CAN zprávy, s výjimkou hrany první „SOF".

Časování založené na standardu CAN 2.0 stanovuje rámec tzv. Nominální doby bitu, který se skládá ze 4 nepřekrývajících se časových segmentů. To lze vidět na obr. 4. Zmíněné 4 segmenty jsou: „Synchronization segment", „Propagation time segment", „Phase segment 1" a „Phase segment 2". V poli synchronizačního segmentu by v ideálním případě měla byt umístěna spádová hrana. Na hranici mezi Fázovým segmentem 1 a 2 se nachází bod vzorkování, tzv. „Sample point" (SP), toto je také vidět na obr. 4. V průběhu synchronizace je vzorkovací bod (SP) posouván dle umístění aktuální hrany. Toho je docíleno odečtem hodnoty tzv. Synchronizačního skoku z anglického „Synchronization Jump Width" (SJW), (často zkracováno pouze na JW) od hodnoty Phase segmentu 2. To ale jen v případě, že se hrana nachází za bodem vzorkování a počet časových úseků (TQ) mezi hranou a synchronizačním segmentem není menší než SJW. V opačném případě, kdy se hrana vyskytuje před bodem vzorkování (SP), je hodnota synchronizačního skoku přičtena k hodnotě Phase segmentu 1. Zmíněný synchronizační skok - „Synchronization Jump Width" (SJW) je obvykle odvozen od frekvence hodin. Základní jednotka je brána jako základní časový úsek „Time Quantum" (TQ), což je celistvý dělitel

nominální doby přenosu 1 bitu. Zkracováni nesmí byt nikdy provedeno s výsledkem kratšího časového segmentu než je tzv. „Information Processing Time“ (IPT), volně přeloženo jako Doba zpracováni. Resynchronizaci lze uplatnit jen jedenkrát za dobu Nominálního bitu.

Délka zmíněných segmentu je následující

Synchronizační segment	---	1TQ
Propagační segment	---	1 až 8TQ
Fázový segment	---	1 až 8TQ
Fázový segment 2	---	ta vyšší hodnota z Fázový segment1 a IPT
Doba zpracování (IPT)	---	minimálně 2 TQ



obr. 4 Rozložení nominálního bitu [2]

1.6 CAN FD

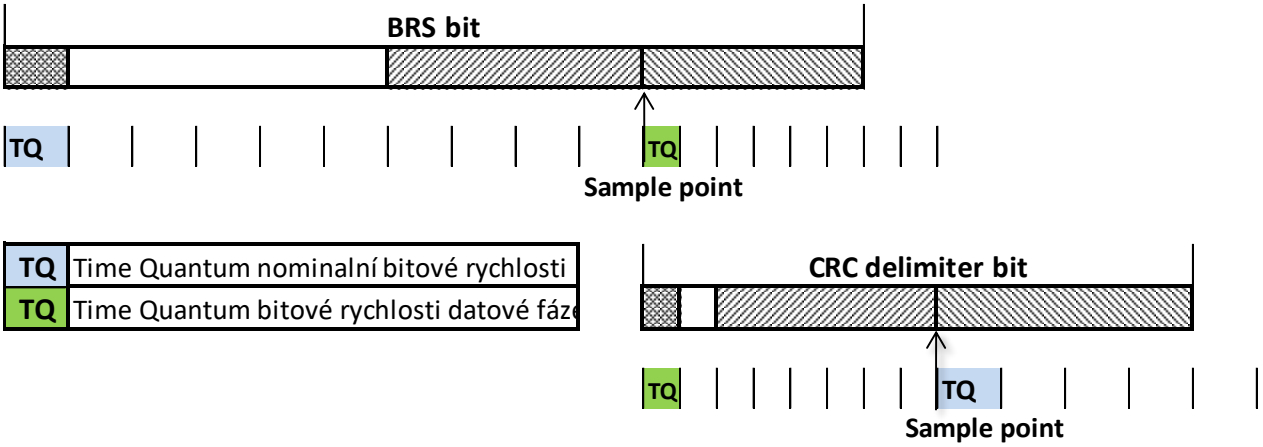
CAN FD – „Flexible Data Rate” je vystavěn na základech CAN 2.0. Z tohoto důvodu budou zmíněny pouze odlišnosti a změny mezi CAN 2.0 a CAN FD. Popis bude omezen jen na nezbytně nutné informace, resp. detaily potřebné k vysvětlení a popisu problematiky CAN FD. Všechny nezbytné další informace jsou popsány ve specifikaci, resp. v ISO dokumentu.

1.6.1 Rozdíly a odlišnosti CAN FD a CAN 2.0

Standardní CAN 2.0 formát zprávy umožňuje přenos dat o maximální velikosti 8Bajtů na zprávu s maximální bitovou rychlostí až 1Mbit/s. Toto je ve zprávě standardu „CAN Flexible Data Rate“ rozšířeno na 64Bajtů dat s maximální bitovou rychlosti dosahující až 8 Mbit/s.

- Jako první změnu lze zmínit jeden z přidaných bitů pojmenovaný jako „FDF“ (FD Format indicator). Tento bit nese informaci o použitém standardu zprávy, zdali se jedná o zprávu standardu CAN 2.0 (Dominantní bit), nebo o CAN FD (Recesivní bit).
- Druhá změna je přidaný bit s názvem „res“ který je pro rezervní účely do budoucna. Vyskytuje se vždy po FDF bitu, ale jen ve zprávách formátu FD. Tento bit je vždy v Dominantní úrovni. Hodnota a umístění bitu je použita pro více mechanismů, které budou zmíněny později.
- Dále se zavádí pojem Datová fáze, což odpovídá rozmezí polí/bitů od BRS bitu po

bit CRC delimiter, ale jen ve zprávách formátu CAN FD.



obr. 5 Přechody mezi bitovými rychlostmi 61[5]

- Ihned za res bitem je umístěn bit BRS „Bit Rate Switch“. Tento bit nese informaci o rychlosti sběrnice, respektive o to, zda má dojít ke změně bit rate pro datovou část FD rámce zprávy. V případě shodné rychlosti cele zprávy je bit dominantní úrovně, naopak při odlišné rychlosti v datové fázi je bit recesivní úrovně. V FD zprávách je bit rate datové části rámce měněna vzhůru a zpět. Tyto změny jsou prováděny v místě vzorkovacího bodu (SP) bitu BRS, (recesivní hodnota bitu) a pak na konci rámce v CRC delim. bitu, viz obr. 5.
- Další změnou je ESI bit („Error State Indicator“), který následuje v FD zprávě za bitem BRS. Tento bit přenáší informaci o Error stavu vysílače. Tedy konkrétně Error Active (Dominantní bit), Error Passive (Recesivní bit).

Indikátor délky datového pole DLC („Data Length Code“) je v porovnání se standardem CAN 2.0 shodný v rozmezí 0 - 8 Bajtů dat. Odlišnost je pak v rozšíření resp. v použití všech zbylých kombinací 4bitového DLC slova na kombinace pro 12, 16, 20, 24, 32, 48 a 64 Bajtů dat. Nové kombinace a jejich význam je vidět níže na obr. 6.

DLC	1001	1010	1011	1100	1101	1110	1111
Byte	12	16	20	24	32	48	64

obr. 6 Obrázek s tabulkou nových DLC kombinací [5]

$CRC15 \qquad C599_{16} \qquad X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1$

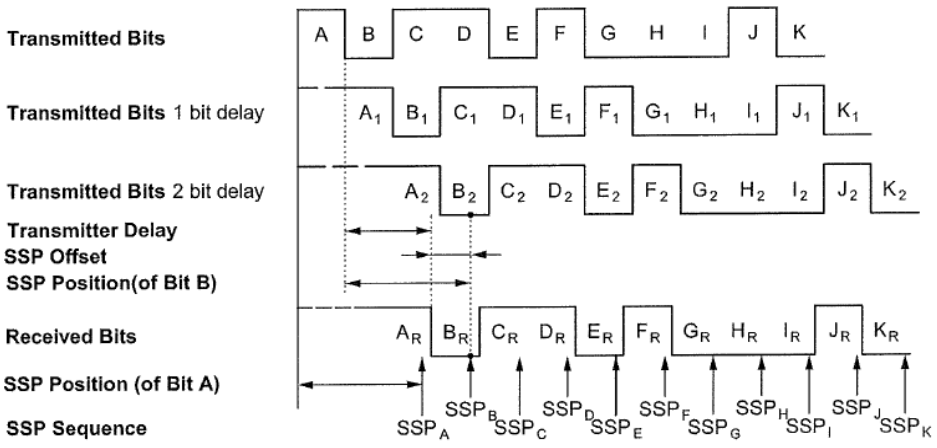
$CRC17 \qquad 3685B_{16} \qquad X^{17} + X^{16} + X^{14} + X^{13} + X^{11} + X^6 + X^4 + X^3 + X^1 + 1$

$CRC21 \qquad 302899_{16} \qquad X^{21} + X^{20} + X^{13} + X^{11} + X^7 + X^4 + X^3 + 1$

Tyto polynomy pro CRC výpočet jsou dané konstanty uvedené v dokumentu ISO. [5]

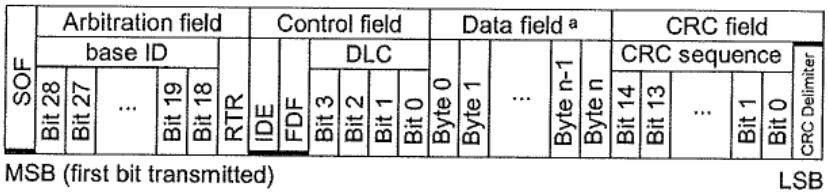
Vlivem většího počtu datových bajtů ve zprávách dle nového standardu je pro dodržení bezpečnosti zprávy nutná změna i ve výpočtu CRC sekvence. CRC výpočet je různý dle množství dat. Jsou zavedeny 2 nové CRC polynomy pro výpočet CRC sekvence FD formátu. Jsou-li data v rozmezí délek 0-16 Bajtů je použit pro výpočet polynom CRC17. V případě, že je délka dat větší než 16Bajtů, je použit polynom CRC21. Pro 0-8 Bajtů je použít polynom dle formátu zprávy, tedy v klasickém CAN 2.0 CRC15, v FD zprávě pak CRC17.

- CRC pole je ve zprávě dle FD formátu značně odlišné od klasického CRC pole ve standardu CAN2.0.
    - První změnou je 4bitové pole nazvané v překladu jako počet stuff bitů – „Stuff count“. Jde o informaci o počtu stuff bitů ve zprávě od SOF bitu po poslední datový bit. Tato informace je přepočtena přes modulo8 a kódována 3bitovým Grayovým kódem. Na pozici 4. Bitu je umístěn bit Parity.
    - Druhou změnou je vkládaní takzvaných Fixních stuff bitu do celého CRC pole. Tyto fixní stuff bity jsou umístěny za každým 4. bitem a to s komplementární hodnotou k bitu poslednímu. Tento Fixní stuff bit je vkládán od prvního bitu CRC pole. Tedy před polem stuff count je vždy fixní stuff bit, dále jsou už v pořadí na každém pátém místě. Jelikož je použit na prvním místě CRC pole FSB, nesmí dojit k vložení standardního stuff bitu na poslední místo Datového rámce, tedy za poslední data bit.
    - V klasickém CAN 2.0 formátu jsou standardní stuff bity vkládány i v CRC poli. Naproti tomu v CAN FD formátu standardní stuff bity v CRC poli povoleny nejsou.
- Z důvodu eliminace problémů na vyšší bitové rychlosti (bitrate), nebo přechodem mezi rychlou a pomalejší bitovou rychlostí, je zavedeno pár následujících mechanismů.
    - Je zavedena druhá „Hard synchronizace“, a to v okamžiku spádové hrany mezi FDF a res bitem. To je z důvodu eliminace problémů s větší fázovou chybou, než je maximální schopnost korekce s menším SJW v FD části. Tu je snazší eliminovat standardním SJW “pomalé” bitové rychlosti. Tento mechanismus je prováděn v každém FD rámci/zprávě.



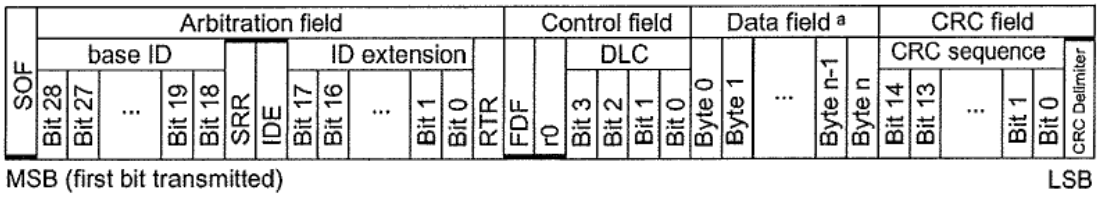
obr. 7 Použití SSP a jeho umístění (viz [5])

- V době Datové fáze, tedy v oblasti s vyšší bitovou rychlostí, by bylo výrazným problémem zpožděním mezi TX a RX, které jsou u zařízení běžné. Toto zpoždění je takzvaně "zpoždění vlivem šíření" z anglického "Propagation delay" a může nabývat hodnot jako například 230ns. To je více než je doba jednoho bitu na sběrnici v data fázi, například při nastavení na bitové rychlosti na 8 Mbit/s, což by způsobilo bit Error, respektive chybu bitu. Z tohoto důvodu má uživatel možnost povolit/použít mechanismus takzvaného "Transmitter delay compensation", tedy kompenzaci zpoždění vysílacího uzlu. Tento mechanismus zavádí druhotný bod vzorkování (SSP) posunutý od standardního SP o zpoždění a nastavitelný offset. Toto zpoždění je měřeno na spádové hraně z FDF na res bit čítačem, který čítá počet minimálních/FD TQ od doby vyslání dominantní hodnoty (res bit) na TX, do doby než je tato dominantní hodnota přijata na RX. V tomto případě je nutné vypnout standardní kontrolu bit erroru a použít kontrolu SSP hodnoty. Celá situace je vidět na obr. 7.



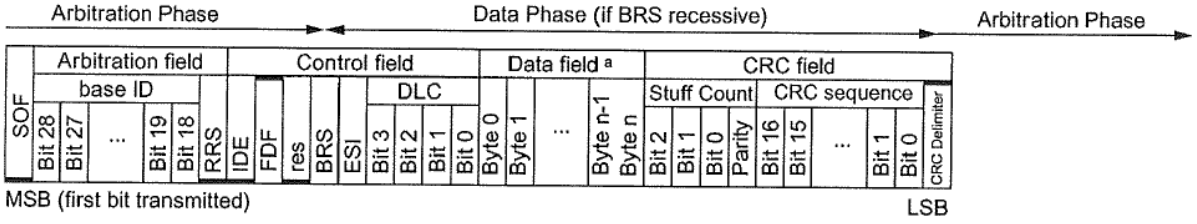
obr. 8 Rámec zprávy dle CAN 2.0 se standardním identifikátorem [5]

Podoba rámce zprávy dle standardu CAN 2.0 je vidět na obr. 8 výše. Jedná se o rámec zprávy, která má použít standardní identifikátor (11bitů)



obr. 9 Rámec zprávy dle CAN 2.0 s rozšířeným identifikátorem [5]

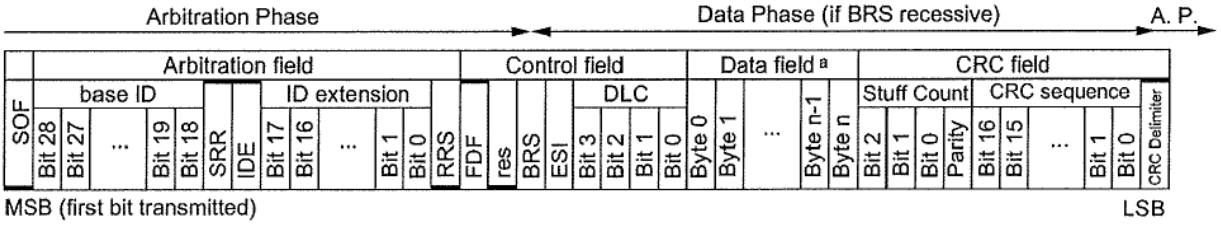
Druhý obrázek je také rámec zprávy standardu CAN 2.0, ale jak je z jeho délky vidět, jde o rámec s rozšířeným identifikátorem. Znamená to použití namísto 11bitů, identifikátor o délce 29 bitů. Situace je vidět na obr. 9.



obr. 10 Rámec zprávy dle CAN FD se standardním identifikátorem [5] (do 16 data Bajtů)

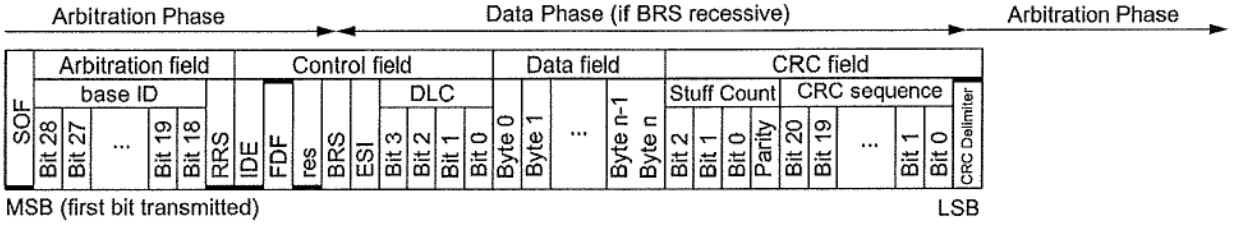
Uspořádání rámce dle standardu CAN FD s použitím standardního identifikátoru (11 bitů) je vidět na obr. 10. Jedná se o typ zprávy, kdy je použita CRC sekvence pro menší počet dat, než

je 17 datových Bytů. Je to vidět na číslování bitů pole CRC.



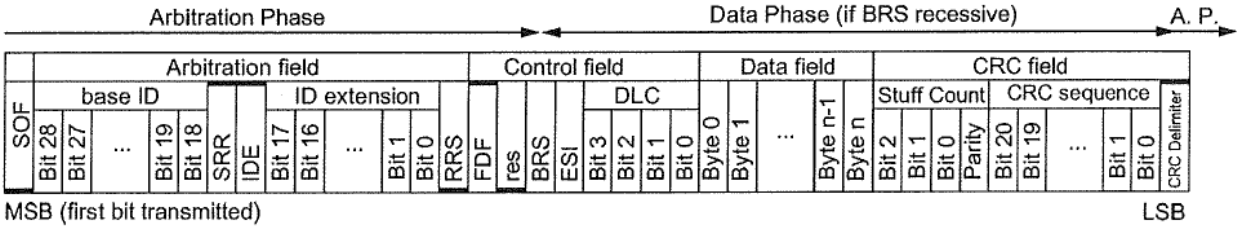
obr. 11 Rámec zprávy dle CAN FD s rozšířeným identifikátorem [5] (do 16 data Bajtů)

Další možností uspořádání rámce zprávy pro standard CAN FD je rámec s rozšířeným identifikátorem. Jde opět o zprávu používající mechanismus CRC s polynomem CRC17. To znamená, že maximální počet datových bajtu v tomto rámci zprávy je 16. Situace je vidět na obr. 11.



obr. 12 Rámec zprávy dle CAN FD se standardním identifikátorem [5] (17 až 64 data Bajtů)

Předposlední možnost je rámec zprávy standardu CAN FD se standardním 11bitovým identifikátorem. Odlišnost od předchozí varianty je v počtu použitých datových Bajtů. V případě použití 17 až 64 datových Bajtů, je použit polynom CRC21 pro výpočet CRC sekvence. To je vidět na číslování bitů pole CRC na obr. 12.



obr. 13 Rámec zprávy dle CAN FD se rozšířeným identifikátorem [5] (17 až 64 data Bajtů)

Poslední kombinace uspořádání rámce dle standardu CAN FD je použití rozšířeného identifikátoru a zároveň použití počtu přenášených Bajtů v rozmezí 17 až 64. Toto je vidět na obr. 13.

Po zhlédnutí všech kombinací rámců zpráv si lze povšimnout, že rámec je ve všech případech identický, co se prvních 14 bitů týče, tedy od SOF bitu po IDE bit. Dále je již rozdíl v použitém druhu identifikátoru a následně použitém standardu. Hlavním dalším rozdílem je kontrolní pole, kde v FD rámci přibyly 4 nové bity. Další změna je v poli CRC Field, do kterého je vloženo v případě FD rámce pole stuff count. Rámce se také odlišují podle počtu datových bajtů ve zprávě, a tím se mění i použitá CRC sekvence.

2 Realizace řadiče CAN

Tato následující část pojednává o realizaci CAN FD řadiče na základě existujícího CAN 2.0 řadiče. V této části jsou popsány kroky a úpravy potřebné k dosažení celku fungujícího řadiče, schopného fungovat na sběrnici používající standard CAN 2.0, ale rovněž i standard CAN FD. To vše je psáno v jazyce VHDL pro FPGA.

V první části práce bylo třeba analyzovat dodaný řadič (CAN 2.0). Následně byla potřebná modifikace na dnešní používané softwarové zázemí, tedy simulační software NCSim pro simulaci signálů v čase a ověření správné funkce. Dalším krokem bylo ověření funkce tohoto řadiče ve zmíněném simulátoru s otestováním co nejvíce CAN pravidel, které standard 2.0 definuje. Toto zahrnuje tvorbu testu a předchozí nastavení prostředí. Po úspěšném otestování je dalším krokem rozšířeni bloku jádra designu o nezbytné části realizující komptabilitu s CAN FD standardem. Jedním z kroku je také úprava registrové mapy z 8bitové na 32bitovou registrovou mapu.

2.1 Dodaný řadič CAN 2.0

Jak již bylo zmíněno, dodaný CAN řadič pracující podle specifikace standardu CAN 2.0, byl vytvořen se schopností komunikace s 8bitovým mikrokontrolérem po 8bitové sběrnici. Verifikace a simulace byla provedena simulátorem ModelSim, který je starším produktem společnosti Mentor Graphics.

K dodanému projektu jsem vytvořil podpůrné procesy a funkce, za účelem správného nastavení řadiče a úspěšnou realizaci funkčních simulací. Tyto byly použity po úspěšném nastavení prostředí.

2.1.1 Nastavení Digitálního prostředí

Nejprve je potřeba nastavení takzvaného digitálního prostředí projektu. Toto zahrnuje nastavení správného jména projektu, nastavení správného pracovního adresáře, volba a nastavení simulátoru (NCSim) a dalších potřebných vlastností s pomoci existujících skriptů.

2.1.2 Testy správného chování designu

Testy designu byly původně provedeny v ModelSim simulátoru, a to za účelem ověření správné funkce jádra designu. Druhotným účelem bylo hlubší seznámení se s designem a také s NCSim simulátorem.

Původní testy, vytvořené k ověření původního řadiče byly vytvořeny formou „do files“. V těchto testech bylo s pomocí přiřazování hodnoty konkrétním signálům na daný čas simulováno, krom jiného vysílání a příjem zpráv. Informace z těchto testů byly použity do určité míry pro tvorbu testů nových v NCSim simulátoru.

Testováno je například: mechanismus detekce chyb, které jsou STUFF, CRC nebo FORM error, nebo ověření bit Stuffing mechanismu v CRC poli. Dále také ověření Datového rámce se standardním a rozšířeným identifikátorem s 0 - 8 B dat, obdobně Remote rámec s 0Bajty dat.

2.1.3 Nezbytné kroky pro nastavení kontroléru

Jakmile je nastaveno prostředí, vytvořen projekt a vše je správně propojeno (signály mezi jednotlivými bloky) musel jsem vytvořit procesy a funkce pro nastavení samotného řadiče s pomocí jeho registrů, vytvořil jsem také procesy a procedury pro následné testování. Například procedury pro správnou komunikaci přes 8bit sběrnici s registry pro čtení příchozích zpráv, čtení druhu nastaveného přerušení, nebo čtení chybové zprávy a rovněž zápis zmíněných i dalších registrů. Důvodem je fakt, že vytvořený interface je připraven pro komunikaci, resp. obecně pro práci s procesorem 8051.

SEND BYTE, READ BYTE procedury

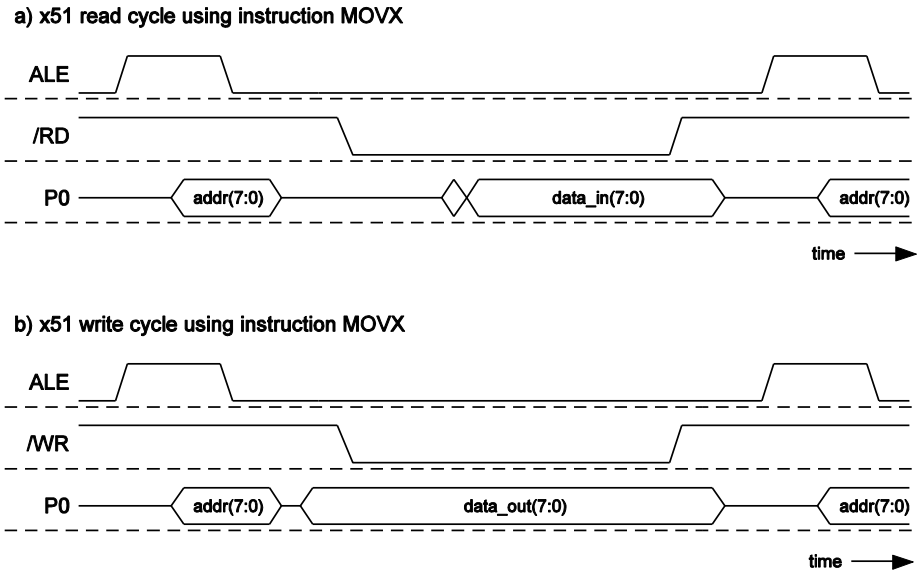
Procedury byly nazvány dle své funkce „SEND BYTE“ a „READ BYTE“. Tyto procedury jsou umístěny hierarchicky na nejnižší úrovni a slouží k použití existujícího x51 interface s 8bit sběrnici pro nastavení/čtení, respektive kontrolu registru radiče za účelem nastavení správného chování řadiče.

Tvar příkazu je: SEND_BYTE (adresa, data) obdobné je to při čteni. Adresy jsou pojmenované, samonapovídající, předdefinované konstanty nesoucí jméno registru nebo jeho části.

Tento interface je založen na cca 7 signálech, které jsou: CLK, RES, CS, ALE, RD, WR a DATA (7:0). Zpráva putující po sběrnici má tvar typického tzv.“ read/write cycle“ MOVX instrukce, což je vidět na obr. 14 viz níže.

Procedury pro odeslání zprávy

Jakmile je kontrolér správně nastaven, vytvoříme „Send message“ proceduru. Tato procedura simuluje rámec zprávy CAN na sběrnici. Toto je pak v simulátoru zobrazeno jako příjem zprávy. Obdobně pak při vysílání s rozdílem, že u vysílání jsou pouze naplněny specifické registry identifikátorem, daty apod. s následným povolením vysílání zprávy.



obr. 14 Průběh čtení a zápisu MOVX instrukce (procesory x51)

Příjem a odesílání zprávy

Rozdělíme-li procedury a funkce na úrovně, je v nejvyšší úrovni procedura pro odesílání a procedura pro příjem CAN zprávy. Obě používají proceduru CRC generátor na nižší úrovni, která na základě vstupních dat vygeneruje potřebně informace jako je CRC sekvence, délka dat (DLC) a počet ID bitů pro stanovení typu identifikátoru standardní/rozšířený. Na základě těchto dat je později sestaven rámec zprávy.

Procedura odesílání Standardní & Extended zprávy

Zde je vytvořen vektor s posloupností bitů, které odpovídají chtěnému významu jednotlivých polí a dohromady CAN zprávě. Tvorba vektoru je dělena dle typu identifikátoru zprávy. Do vektoru jsou na závěr doplněny stuff bity a pak je bit po bitu odeslán hned, jakmile je sběrnice volná.

Procedury pro kontrolu chyby & přerušení

Pro kontrolu stavu uzlu jsou vytvořeny procedury pro čtení registru s informací o stavu. Read (READ_BYTE). Procedura kontroly chyby čte registr chybového stavu a po dekódování jakéhokoliv erroru (nenulová hodnota konkrétních bitu) informuje uživatele zprávou o typu chyby. Obdobně dochází ke kontrole registru přerušeni s následnou dekódovanou informaci pro uživatele.

Procedura kontroly stavu uzlu

Tato procedura používá všechny zmíněné procedury a funkce na konci testu pro zjištění stavu uzlu. Konkrétně tedy, zda byla zpráva vyslána/přijata v pořádku, zda byla detekována chyba případně, jaká chyba to byla, a to i v závislosti na přerušení. Na závěr je porovnána rovnost bloku dat z registru s daty v testu. Například při odesílání, zda byla testem odeslaná data správně přijata a uložena do příchozího registru dat.

2.2 Návrh rozšíření o CAN FD

Jelikož je CAN standard hojně využíván, dosahují dnes často sběrnice založené na standardu CAN 2.0 horních limitních parametrů z pohledu objemu přenesených dat. Z tohoto důvodu byl vyvinut standard CAN FD. Motivací se stalo zdokonalení existujícího řadiče na úroveň, kdy bude schopen provozu v obou variantách a bude tak umožněn vyšší objem přenesených dat na sběrnici mezi zařízeními a rovněž kompatibilita s novými i starými zařízeními.

Ze všeho nejdříve bylo třeba si rozmyslet tzv. dekompozici. Je velmi důležité mít vše setříděno a rozděleno do úrovní a bloků. Zabrání to budoucím problémům s úpravami čehokoli v projektu. Toto nám pak dává možnost najít ten správný blok a to správné místo s konkrétní funkcí, jenž je třeba změnit. A to změna pouze specifické věci bez dlouhého hledání a chápání více částí okolo, než je nezbytně nutné.

Nezbytným krokem je také hluboké porozumění existujícího designu řadiče na standardu CAN 2.0. Zvláště tyto části:

- část s časováním (BTL) z důvodu možnosti rozšíření a modifikace časování (bit timing) a také přepínání mezi rychlostmi v FD standardu a klasickým CAN 2.0
- bit stuffing (BSL) pro možnost rozšíření o fixní stuff bity uvnitř CRC pole a možnost omezit, respektive nastavit pravidla standardního stuffingu.
- bit timing procesor (BSL) za účelem modifikace stavového automatu o nové stavy a rovněž rozšíření pravidel pro nastavování signálů a přechodů mezi stavy, rozšíření CRC mechanismu, nemluvě o pravidlech komunikace jádra s interfacem.

Dalším krokem je také úvaha nad úpravou designu tak, aby bylo dosaženo cíle s co nejmenším počtem zásahů a změn v designu. Jedním z takových rozhodnutí bylo například přebudování registrové mapy z 8bitové na 32bitovou s přístupem přes AHB-Lite sběrnici pomocí 8bitových slov.

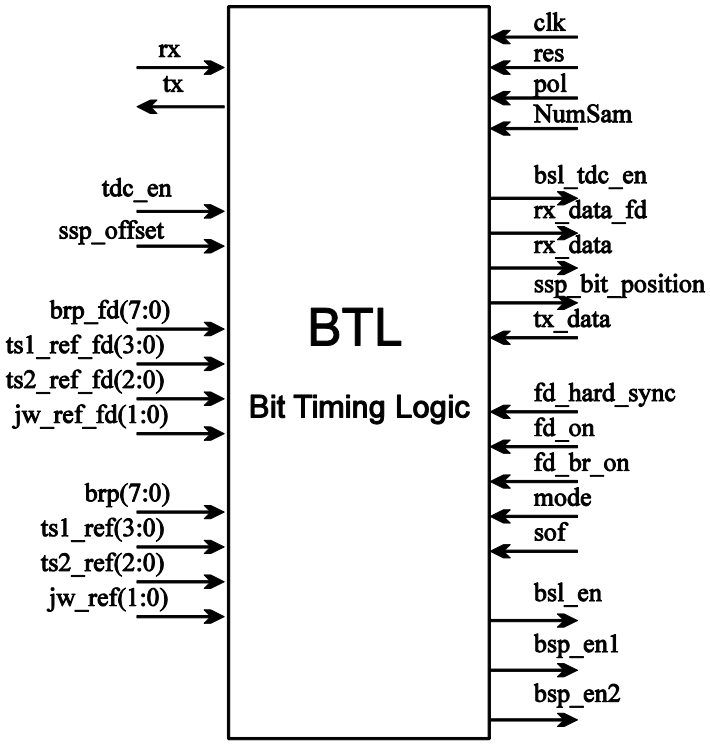
2.2.1 úpravy části jádra - Kernel

Jak již bylo zmíněno, za účelem umožnění funkce dle CAN FD standardu, je třeba rozšířit logiku realizující jádro řadiče. Následující část je o nových pravidlech, komponentech, procesech či principech vkládaných do již existujícího bloku jádra (blok Kernel). To vše je rozděleno dle bloků, ze kterých se jádro skládá.

Tedy blok BSP (Bit Stream Processor), který je řídící a hlavní části jádra. Jeho činnost je založena na stavovém automatu, pomocí kterého je, jak název napovídá, skládán, či dekódován bitový tok. Tento „bit-Stream“ je upravován dále blokem BSL (Bit Stuffing Logic), který se stará o operace s doplňkovými bity tzv. stuff bity. V závislosti na nich je, při monitoringu sběrnice a dané fázi přenosu, ovládán stavový automat bloku BSP, který je povolován či deaktivován v závislosti na aktuálním bitu (vkládaném bitu), (Zda se má daný bit zpracovávat či nikoliv). Blok BSL rovněž vyhodnocuje a detekuje chyby, jako jsou chyba bitu či chyba Stuff mechanismu. Poslední blok jádra, blok BTL (Bit Timing Logic) realizuje časování uzlu na sběrnici, čímž je krom jiného hlavně myšlena synchronizace uzlu na připojenou sběrnici.

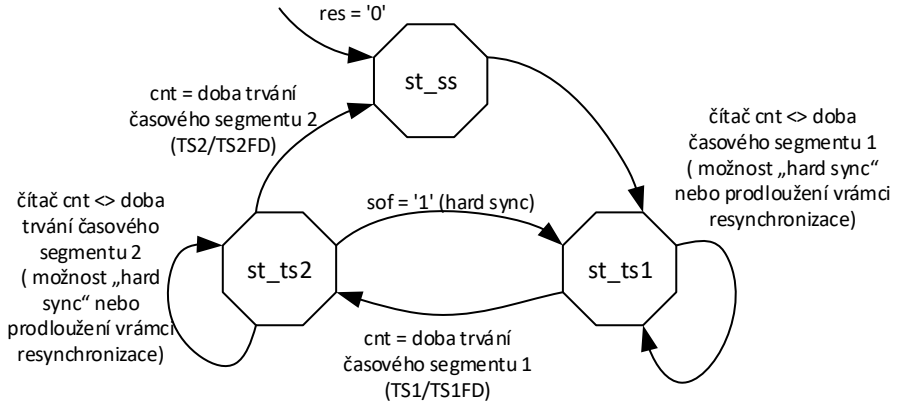
Vysvětlení bude co možná nejpodrobnější a nejjednodušší s cílem snadného a rychlého pochopení nezasvěcené osoby, s předpokladem znalostí základních pravidel standardu CAN 2.0 a CAN FD, které jsou zmíněny v kapitole: 1.5 a 1.6.

BTL ("Bit Timing Logic") a editace



obr. 15 Blok BTL

Jako první z pohledu příchozího bitu, ze sběrnice do jádra designu, je v cestě blok BTL. Tento blok se stará o veškeré časování a synchronizaci řadiče na sběrnici. Jde tedy o nastavení bitových rychlostí dle standardu a možností CAN 2.0, ale také standardu CAN FD. Tento blok rovněž realizuje stavovým automatem (viz obr. 16) sestavení tzv. Nominálního bitu, tedy rámce bitu. Ten, jak již bylo zmíněno v teoretické části, se skládá z: Synchronizačního segmentu (SS), časového segmentu 1 (TS1) a časového segmentu 2 (TS2). Dále také realizuje vzorkovaní hodnoty RX v bodě SP. Zmíněný stavový automat pracuje dle obr. 16. Tento automat je řízen hodnotou čítače *cnt*. Tedy nastane-li stav, kdy je hodnota čítače rovna nastavené hodnotě velikosti daného segmentu (ss, ts1, ts2) je čítač nulován a automat přechází do následujícího stavu. Velikost segmentů je dělitelná na časové úseky tzv. TQ. Výjimkou je tvrdá synchronizace, po které automat přechází vždy do stavu časového segmentu 1 (TS1). V případě vysílání je na přelomu stavů TS2 a SS přenášena na sběrnici odezva na poslední přijatý bit. Obdobný průběh je při přijímání, jak již bylo uvedeno v teoretické části. Tedy na rozhraní stavu TS1 a TS2 je umístěn (SP) bod vzorkování, v jehož okamžik je vzorkována hodnota sběrnice. Ta je následně poskytnuta ke zpracování. Toto zpracování je ale vykonáváno až v povolený okamžik, který je zajištěn generováním 3 řídících signálů: *bsl_en*, *bsp_en1* a *bsp_en2*.



obr. 16 Stavový automat bloku BTL [2]

Jak je vidět na obr. 16, čítač předděličky čítá do nastavených hodnot registru (SS, TS1, TS2). Jakmile je dosažena hodnota registru je hodnota čítače nulována a v okamžiku průchodu nulou je generován pulz signálu enable či enable_FD, který odpovídá informaci o uplynutí doby TQ a začátku nového. Tento signál je pak použit k aktivaci zbylých částí bloku. V případě resynchronizace je v závislosti na umístění hrany signálu vůči stavu automatu zkracován segment 2, či prodlužován časový segment 1 o nastavenou hodnotu skoku JW (JW_FD). V tento okamžik (např. zkrácení TS2) je také nulována hodnota čítače a posunut stav automatu. Obdobně při „hard synchronizaci“ je restartován čítač předděličky a nulován čítač.

Blok BTL si z registrů odebírá informaci o uživatelem zvolené konfiguraci řadiče. Konkrétně jde o *brp* definující nastavení předděličky, tedy výsledného taktu TQ, dále pak *ts1_ref, ts2_ref, jw_ref* pro definici parametrů délky segmentů tvořící nominální bit dle klasického standardu CAN 2.0. Obdobně pro standard CAN FD jsou pak přidány signály *brp_fd, ts1_ref_fd, ts2_ref_fd, jw_ref_fd* pro definici podoby bitu v datové fázi FD bitu.

Dalším rozšířením je signál *tdc_en,* který z registru nese informaci o volbě uživatele, zda je v módu vysílače použit mechanismus kompenzace zpoždění (TDC – „Transmitter Delay Compensation“) či nikoliv. Poslední z nových signálů vedoucí do nastavitelných registrů je signál *ssp_offset*, který slouží k uživatelem definovanému posunu sekundárního bodu vzorkovaní (SSP) vůči původnímu Sample Pointu.

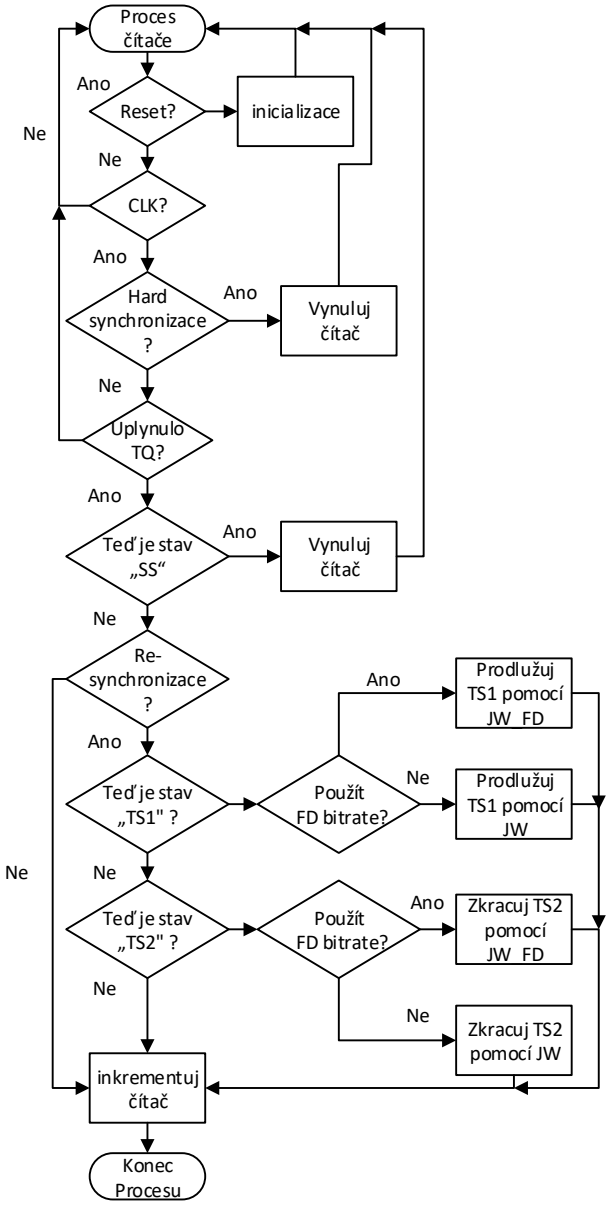
Přepínání bitové rychlosti („Bit Rate Switch“)

Přepínání bitové rychlosti je prováděno v okamžik vzorkovacího bodu BRS bitu za podmínky, že je vzorkovaná hodnota bitu recesivní. Zpět na klasickou, nastavenou rychlost se bitová rychlost vrací v okamžiku vzorkovacího bodu bitu CRC delimiter. K řízení tohoto přepínání jsou použity signály *fd_on* nesoucí informaci o druhu zprávy, *fd_br_on* signál upozorňující na potenciální přepnutí bitové rychlosti jakmile dojde na vzorkovací bod BRS bitu. Dále také vnitřní signál *fd_br_en*, který slouží jako přepínač v podmínkách procesů k rozlišení použití časových segmentů klasického CANu či použití časovacích segmentů pro CAN FD. Tento signál se ve zmíněných vzorkovacích bodech (BRS a CRC delim.) nastavuje dle hodnoty vzorku.

Rozlišení formátu zprávy

Pro jasnou definici, zda se jedná v daný okamžik o standardní zprávu dle CAN 2.0 standardu, nebo o zprávu dle CAN FD standardu, je použit signál *fd_on*, který se v bloku BSP při navzorkovaném FDF bitu nastaví do log1 (recesivní), nebo do log 0 (dominantní). Signál *fd_br_on,* který slouží jako potenciální informace o přepnutí bit rate je využit v bloku BTL k povolení přepnutí v následujícím vzorkovacím okamžiku (BRS bit, či CRC delim.) Pro jasnou definici, kdy má docházet k přepnutí bitové rychlosti (bitrate), je v bloku BTL sestavena podmínka ze všech výše zmíněných signálů v kombinaci s hodnotou navzorkované sběrnice. Pomocí těchto podmínek je pak rozlišováno použití nastavení pro bit FD a klasický bit.

Vývojové diagramy (BTL)



V této části jsou popsány stěžejní procesy bloku BTL, jejich logika a funkce se samotným účelem.

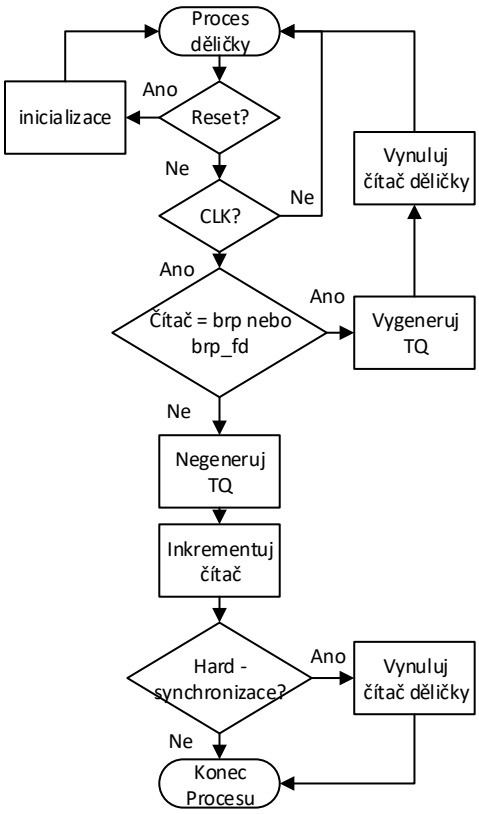
Proces čítače bloku BTL. Na obr. 17 je zobrazena logika procesu realizující funkci čítače v bloku BTL. Jak je vidět z obrázku, je vždy nejprve kontrolována hodnota signálu reset a hodnota hodin. V případě výskytu impulzu na reset signálu je provedena inicializace všech signálu do výchozích hodnot. Další v pořadí je v tomto procesu kontrola, zda není splněna podmínka pro vykonání „Hard synchronizace“ s případným nulováním čítače při splnění. Další kontrola v pořadí je kontrola výskytu, resp. uplynutí doby TQ. Pokud je podmínka splněna, je kontrolován stav, ve kterém se aktuálně stavový automat nachází. V případě synchronizačního segmentu je čítač vždy nulován, jelikož jeho délka je 1TQ. Pokud se nejedna o stav ss, kontroluje se, zda jde o resynchronizaci. Pokud ano, je vyjasněno v jakém stavu se automat nachází. Následně je vyjasněno, zda jde o klasickou bitovou rychlost či o bitovou rychlost FD zprávy v datové fázi. Po vyjasnění je provedena resynchronizace zkrácením (TS2) či prodloužením (TS1) segmentu s pomocí synchronizačního skoku, který je volen dle typu bitu (klasický / FD). Pokud se nejedná o resynchronizaci, nebo je resynchronizace již vykonána, je pouze inkrementován čítač.

obr. 17 Vývojový diagram procesu čítače bloku BTL

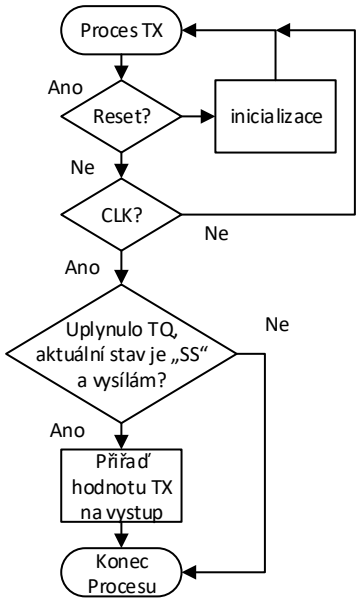
Proces předděličky bloku BTL. Na obr. 18 je vidět logika procesu předděličky bloku BTL. Po úvodní kontrole hodnoty reset signálu a případné inicializaci při splnění podmínky je kontrolována hodnota hodinového signálu.

Další stěžejní kontrola je rovnost čítače předděličky a nastavené hodnoty registru předděličky pro klasický či FD bitrate. V případě rovnosti čítač dosáhl velikosti časového segmentu TQ a je generován pulz na signálu enable reprezentující TQ a povolující činnost jiných procesů bloku BTL. Zároveň se po vygenerování TQ čítač vrací na hodnotu nula, od které čítá při časovém impulzu o 1 vzhůru.

V případě nerovnosti čítače a registru předděličky, je ponechán signál enable (TQ) v nule a čítač je inkrementován. Na závěr je kontrolováno, zda nedošlo k okamžiku resynchronizace, kdy se případně nuluje čítač. V opačném případě je cyklus uzavřen.



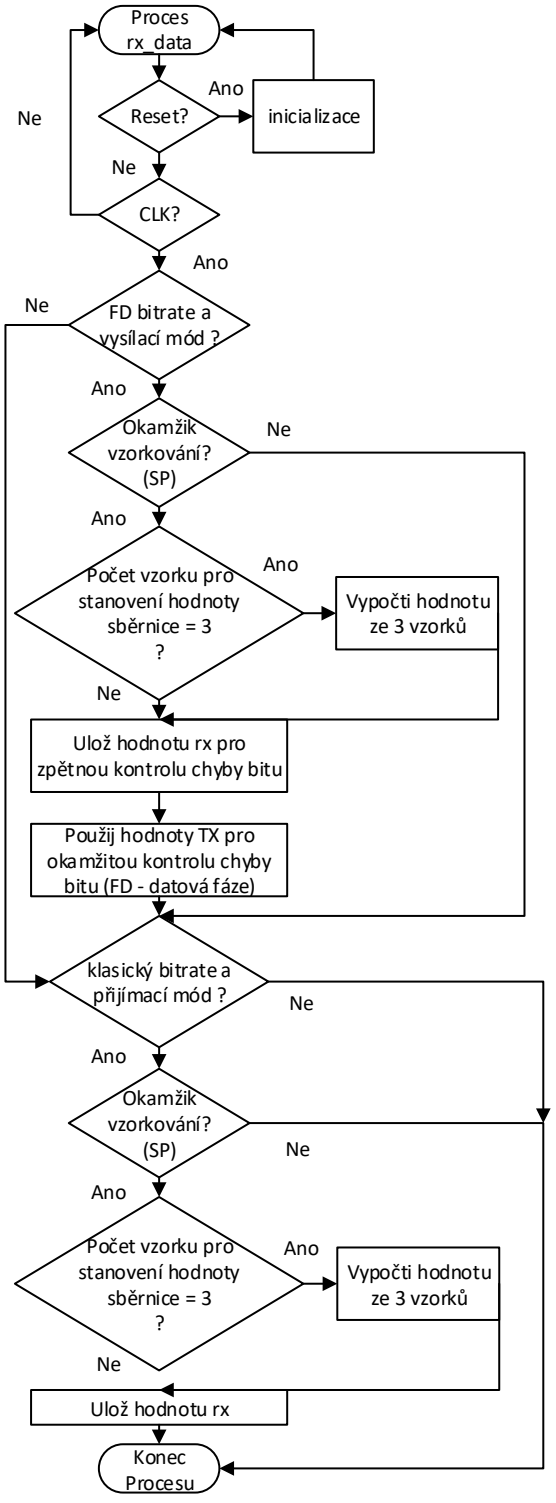
obr. 18 Vývojový diagram procesu děličky bloku BTL



Proces vysílání bloku BTL. Na obr. 19 vidíme logiku procesu pro vysílání bloku BTL. Na úvod je opět kontrolován reset s případným provedením inicializace signálů v případě splnění podmínky. Následně je kontrolován signál CLK.

Stěžejní podmínka vysílacího procesu je kontrola 3 faktů. Tedy kontrola, zda uplynul okamžik TQ a zároveň, zda je aktuální stav stavového automatu Synchronizační segment a samozřejmá kontrola, zda je řadič v módu vysílače. Pokud jsou tyto 3 podmínky splněny, je ten správný okamžik pro přiřazení bitu pro vyslání na výstupní port TX.

obr. 19 Vývojový diagram procesu vysílání bloku BTL



Proces přijímání bloku BTL. Na obr. 20 je vidět logika procesu pro příjem zprávy, respektive hodnoty bitu sběrnice CAN. Na úvod jako ve všech procesech, je při splnění reset podmínky provedena inicializace všech signálů. Dále v případě CLK impulzu je kontrolován typ aktuální používané bitové rychlosti (bitrate) a mód v jakém se řadič nachází.

Pro případ vysílaní a bitové rychlosti rámce FD v datové fázi je vyčkáno na okamžik vzorkování (SP) a následně dle nastavení registru NumSam je stanovena vzorkovaná hodnota sběrnice. Ta je vypočtena ze tří vzorků hodnot sběrnice (NumSam = 1), nebo je použit pouze jeden vzorek sběrnice (NumSam = 0). Z důvodu zpětné kontroly je hodnota TX ukládána. Zároveň je ukládána reálná hodnota RX na sběrnici. Tato hodnota RX je dále umístěna na výstupní port bloku BTL pro další zpracování blokem BSL a později pak BSP.

Ve druhém případě, pokud se jedná o přijímací mód řadiče a klasický bitrate, je postup obdobný. Je vyčkáváno na vzorkovací okamžik (SP) a následně dle hodnoty registru NumSam je vypočtena hodnota sběrnice ze 3 vzorků, nebo je použita hodnota jednoho vzorku. Zde se pak hodnota RX posílá na výstup bloku ke zpracování blokem BSL.

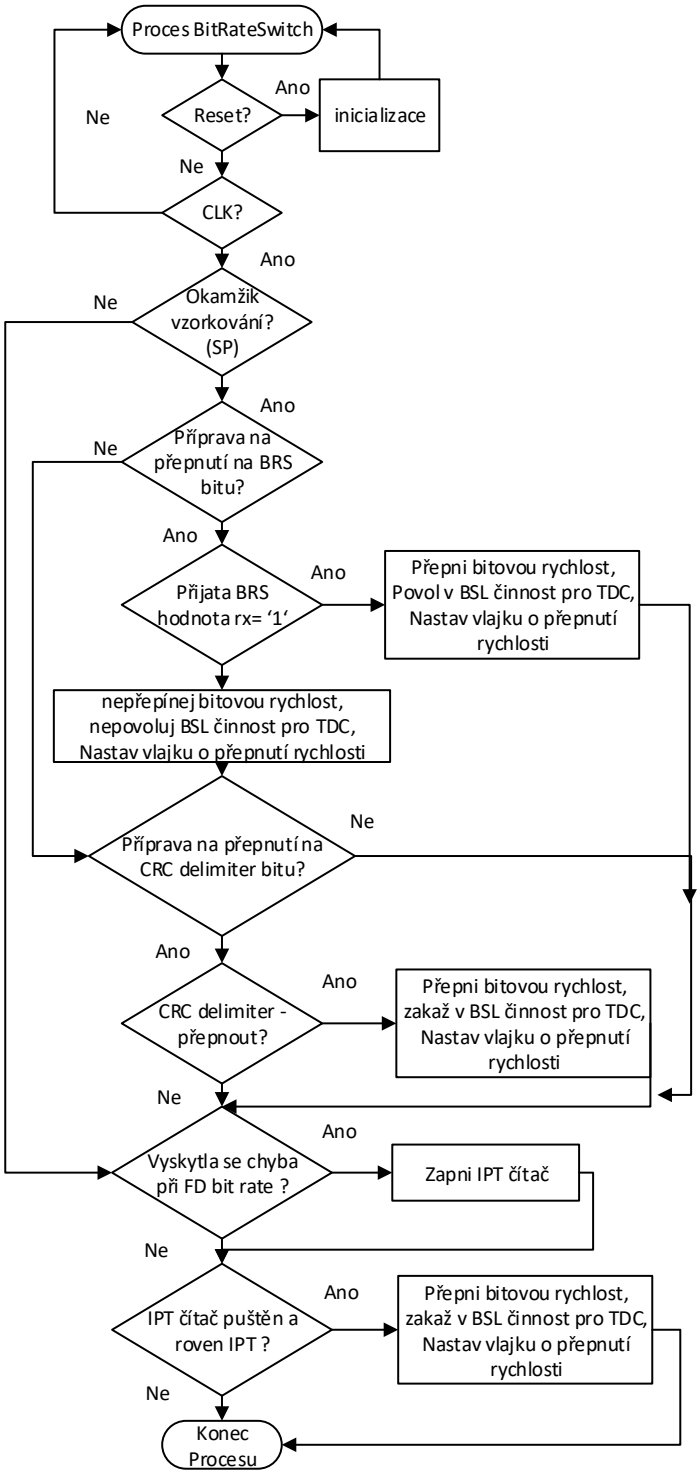
obr. 20 Vývojový diagram procesu přijímaní bloku BTL

Proces přepínání bitové rychlosti BRS v bloku BTL.

Na obr. 21 je vidět logika procesu pro přepínání bitové rychlosti. Nejprve je začátek věnován kontrole resetu s případným inicializováním proměnných a signálů. Dále je kontrolována hodnota signálu CLK, tedy hodin.

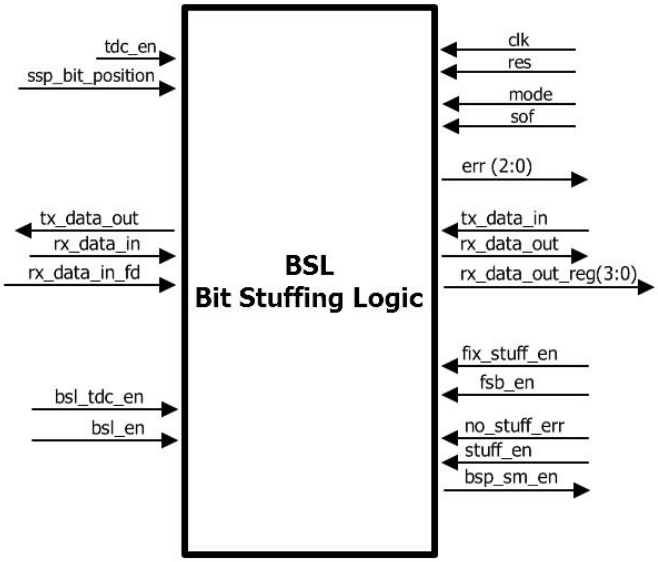
Jelikož přepínání bitové rychlosti je prováděno v okamžiku vzorkovacího bodu (SP), je dalším krokem kontrola, zda nastal tento okamžik. Dále je kontrolován signál z bloku BSP informující o potenciálním přepnutí – tedy pokyn k přípravě přepnutí na BRS bitu. Dále je pak kontrolována hodnota sběrnice. V případě, že se jedná o recesivní hodnotu, jde o BRS bit uvozující přepnutí na vyšší bitovou rychlost FD zprávy, což se také vykoná. V případě dominantní hodnoty nedochází k přepínání na FD bitovou rychlost, ale je nastaven pouze signál informující o proběhnutí pokusu přepnutí na BRS bitu.

Další kontrolou v pořadí je kontrola signálu z BSP informující o potenciálním přepnutí z FD bit rate na klasickou bitovou rychlost v okamžiku SP bitu CRC delimiter. Následně, když se dospěje k tomuto bitu, je bitová rychlost přepnuta na klasickou rychlost, a to pouze v případě používání FD bitrate. Další kontrola je k detekci chyby, a tedy k následnému přepnutí FD bitové rychlosti na klasickou, ale až po uplynutí doby IPT, k čemuž je poslední kontrola kontrolující čítač IPT, zda dovršil této hodnoty a může/má dojít k přepnutí na klasickou bitovou rychlost.



obr. 21 Vývojový diagram procesu Přepínání bitové rychlosti BRS v bloku BTL

BSL (“Bit Stuffing logic”) a editace



obr. 22 Blok BSL

Druhý blok v pořadí v cestě příchozího bitu je blok BSL (Bit Stuffing Logic), viz blokové schéma na obr. 22. Jde o blok jádra designu starajícího se kromě hlavního úkolu tzv. Bit stuffingu, také o detekci a vyhodnocení chyb. Chod tohoto bloku není řízen na rozdíl od zbylých dvou žádným stavovým automatem, ale pouze vytvořenými podmínkami. Jak již bylo zmíněno “bit stuffing” je mechanismus vkládání komplementárního bitu po 5 stejných, po sobě jdoucích bitech, za účelem vytvoření sestupné hrany použité pro synchronizaci (v případě módu vysílače). Obdobně pak na přijímací straně tento blok zbavuje tzv. „bit stream“ (bitový tok) těchto doplňkových bitů a dále předává již upravenou zprávu pro BSP blok. V okamžiku Stuff bitu vypíná blok BSL aktivační signál *bsp_sm_en*, pomocí něhož je provedeno řízení povolení zpracování příchozího bitu blokem BSP.

Novými vstupními signály pro tento blok jsou *fsb_en, fix_stuff_en, tdc_en, bsl_tdc_en, sof, ssp_bit_position* a upraven byl signál *err*. Signál *fsb_en* slouží společně s *fix_stuff_en* k přesnému načasování operace s fixními stuff bity v oblasti CRC pole vyskytující se pouze v FD zprávě.

Modulo 8 & Gray code(3bit)				
Stuff count				
SC2	SC1	SC0	Parity	DEC
0	0	0	0	0
0	0	1	1	1
0	1	1	0	2
0	1	0	1	3
1	1	0	0	4
1	1	1	1	5
1	0	1	0	6
1	0	0	1	7

obr. 23 Pravdivostní tabulka pole Stuffcount

Fixní Bit Stuffing

Signál *fsb_en* slouží k povolení a předběžné informaci o následné potřebě vkládat/eliminovat fixní stuff bity (FSB). Signál *fix_stuff_en* pak přesně v daný bit uvozuje vkládání/odebrání FSB na dané bitové pozici. Pozice bitu je čítána bitovým čítačem *bit_cnt*, který počítá od hodnoty 0 do hodnoty 4. Tento čítač je povolován právě zmíněnými signály. Pracuje tedy jen v oblasti

pole CRC Field v FD zprávě. V případě každé nulové hodnoty čítače se v okamžiku povolení činnosti bloku BSL signálem *bsl_en* v log 1, vkládá FSB, je-li uzel v módu vysílače. Obdobně tomu je v případě, že je uzel v módu přijímače. To znamená, že v tento okamžik je aktuální bit (FSB) z posloupnosti bitů, jenž je dále vyhodnocována blokem BSP, odebrán.

Bity FSB jsou umístěny na každém pátém místě v CRC poli, a to pouze ve zprávě typu FD. Počet těchto doplňkových bitů je různý. Liší se dle použité CRC kalkulace, respektive dle množství datových bajtů ve zprávě. Jak je vidět na obr. 24, tak v CRC15, tedy v klasickém CAN 2.0 standardu se FSB nepoužívá. Naopak při použití CRC17 či CRC 21 se již používá. Délka CRC posloupnosti je, jak název napovídá 17 či 21 bitů, a to odpovídá při rozdělení po 4 bitech, počínaje polem stuff count, na 6 fixních stuff bitů pro sekvenci CRC17 a 7 těchto FSB bitů pro sekvenci CRC21.

	CRC FIELD																															
CRC type / bit number		Stuff count				CRC sequence																										
	1.FSB	SC2	SC1	SC0	Parity	2.FSB	0	1	2	3	3.FSB	4	5	6	7	4.FSB	8	9	10	11	5.FSB	12	13	14	15	6.FSB	16	17	18	19	7.FSB	20
CRC15																																
CRC17	FSB	Y	Y	Y	Z	FSB	x	x	x	x	FSB	x	x	x	x	FSB	x	x	x	x	FSB	x	x	x	x	FSB	x					
CRC21	FSB	Y	Y	Y	Z	FSB	x	x	x	x	FSB	x	x	x	x	FSB	x	x	x	x	FSB	x	x	x	x	FSB	x	x	x	x	FSB	x

obr. 24 Fixní bit stuffing v poli CRC Field - rozložení

Je třeba vzít v potaz výjimku prvního bitu, který je FSB bitem vždy, a proto je nutné omezit standardní bit stuffing tak, aby nedošlo k situaci, kde budou vedle sebe stuff bit z Datového pole a Fixní stuff bit z pole CRC.

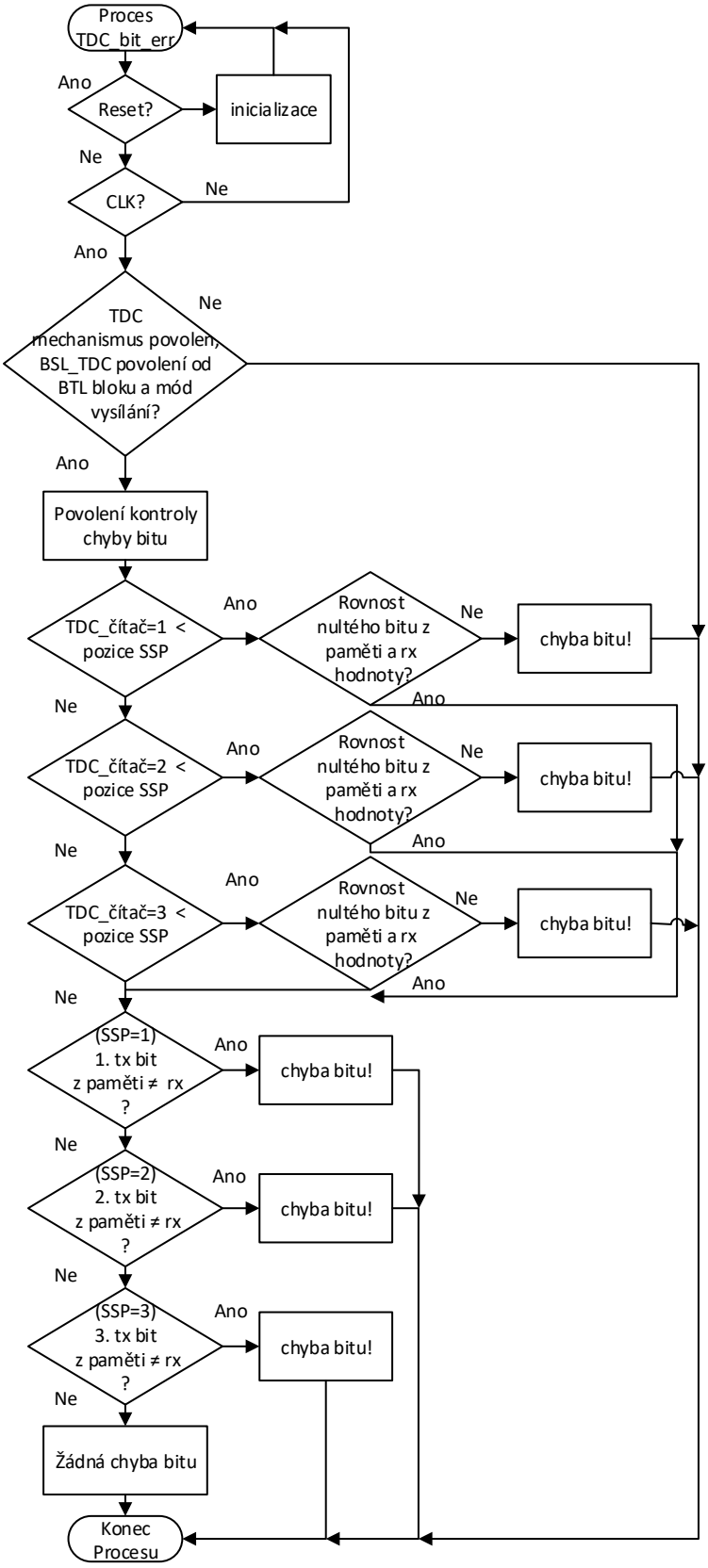
Pole stuff count, jak bylo v teoretické části zmíněno, viz.1.6.1 přenáší informaci o počtu použitých standardních stuff bitů, která je kódovaná metodou modulo8 a dále kódována 3bit Grayovým kódem s přidáním bitu parity (modulo odpovídá zbytku po dělení, v našem případě odpovídá zbytku po dělení číslem 8). Tabulka kombinací pole stuff count je vidět na obr. 23.

Kontrola chyb

Tento blok, jak již bylo zmíněno, se stará také o kontrolu chyb. Jednou z doplněných kontrol je kontrola hodnoty FSB bitu. Na straně přijímače se kontroluje FSB a také předchozí bit, tedy kontrola komplementarity FSB bitu k bitu předchozímu. V případě chyby, tedy že jsou tyto dva po sobě jdoucí bity shodné hodnoty, je vyslána chybová zpráva FORM erroru.

Vstupní signál *tdc_e*n povoluje v bloku činnost procedur tykajících se kontroly bit erroru v případě vysílače majícího zpoždění mezi vyslanou a přijatou hodnotou (TX a RX). V BSL bloku je zřízen registr vyslaných hodnot, který uchovává 7 posledních vyslaných bitů na TX. Tato bitová paměť slouží ke kontrole zpožděného přijatého bitu za účelem vyhodnocení bit erroru. Volba bitu z tohoto paměťového registru závisí na měření a výpočtu v bloku BTL, který hodnotou signálu *ssp_bit_position* udává pořadové umístění kontrolovaného bitu z paměti. Tedy řekněme „ukazatel“ na bit, který se použije pro kontrolu bit erroru s právě přijatou hodnotou.

Stávající kontroly standardního bit stuffingu byly ponechány beze změn. Jde o kontrolu posloupnosti 6 po sobě jdoucích stejných hodnot bitů. Za tímto účelem je zřízen registr příchozích hodnot, který v případě obsahu „000000“ nebo „111111“ signalizuje „stuff error“.



obr. 25 Vývojový diagram kontroly chyby bitu s povoleným TDC (BSL)

Vývojové diagramy (BSL)

Proces kontroly chyby bitu s TDC mechanismem při vysílání v bloku BSL. (obr. 25)

Na úvod je kontrolován signál reset s případnou inicializací signálů a proměnných. Následně je kontrolován signál hodin CLK.

Hlavní podmínkou je kontrola povolení TDC mechanismu v registru, pak povolení mechanismu kontroly chyby bitu od bloku BTL a následně kontrola nastavení řadiče na vysílání. Při splnění je povolen signál pro povolení kontroly chyby bitu.

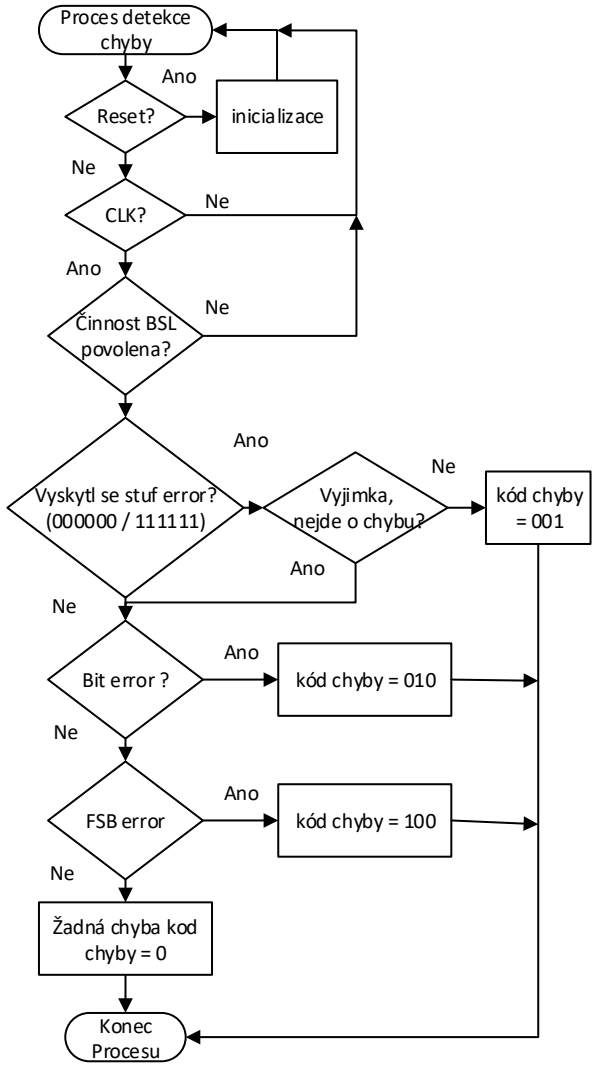
Následující kontrolní blok je zde za účelem kontroly chyby prvních 3 bitů, tedy prvních bitů po přepnutí rychlosti (až 3 bity DLC pole), než je v posuvném registru umístěn správný bit na místě odpovídající změřenému zpoždění (počet bitů zpoždění). V případě nerovnosti dané kontroly je vyhodnocena chyby bitu.

Kontrolní blok na konci je pro kontrolu zbytku zprávy v okamžik, kdy je již posuvný registr vyslaných hodnot správně naplněn pro kontrolu zpožděného bitu s konkrétním bitem dle naměřené a vypočtené hodnoty SSP. V případě nedetekování žádné chyby není nastavována žádná chyba.

Proces detekce chyby v bloku BSL. Na obr. 26 je vidět logika procesu detekce chyby v bloku BSL. Po úvodní kontrole reset signálu a inicializaci je při hodinovém pulzu kontrolováno povolení činnosti bloku BSL.

V případě povolení činnosti bloku BSL jsou kontrolovány podmínky jednotlivých druhů chyb. První v pořadí je kontrolována chyba doplňkových bitů, tedy selhání bit stuffing mechanismu. Podmínka reaguje na posloupnost 6 stejných hodnot na sběrnici. Následně je ověřeno, zda se nejedná o výjimku, jako tomu je například v případě pole End of Frame. Pokud ne, je nastaven kód chyby na binární hodnotu 1. Další v pořadí je kontrola bit erroru, tedy kontrola hodnoty vyslaného a přijatého bitu, V případě selhání je nastaven kód chyby 2. Třetí kontrolou je kontrola mechanismu Fixních stuff bitů. Jde o kontrolu, zda je bit na pozici fixního stuff bitu komplementární k bitu předchozímu. V případě selhání je nastaven kód chyby 4.

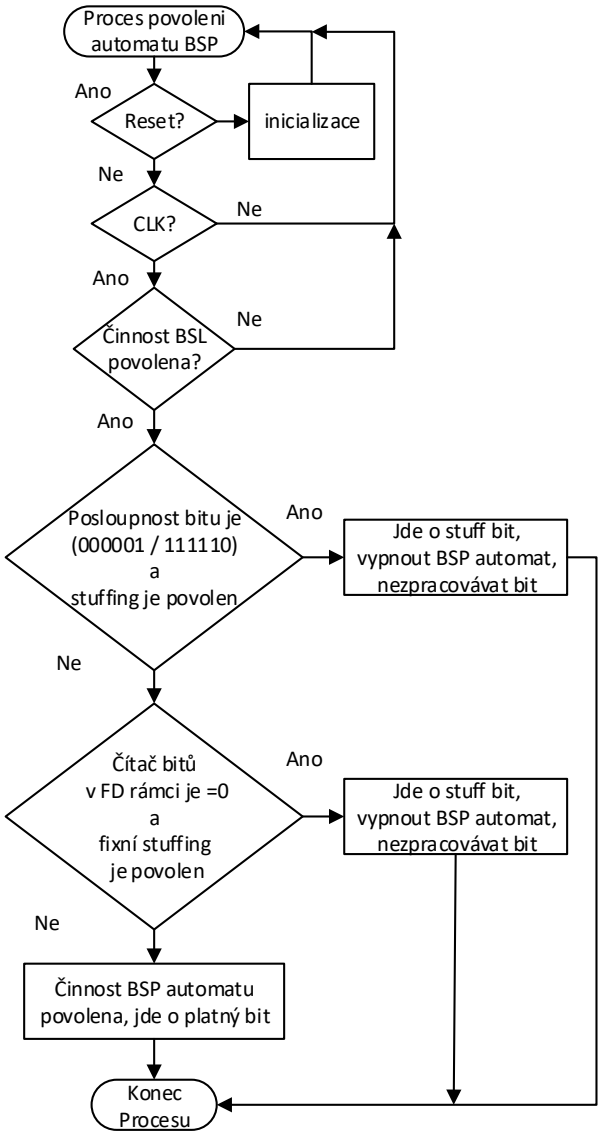
Pokud není detekována žádná chyba je nastaven, respektive ponechán kód chyby na nule.



obr. 26 Vývojový diagram procesu detekce chyby

Podmínečné přiřazení hodnoty signálu mimo proces pro realizaci stuffingu a fixního bit stuffingu v bloku BSL. Na obr. 27 je vidět logika provedení stuffing mechanismu. Použitím podmíněného přiřazení hodnoty signálu, tedy zápisu „WHEN“, která vykoná přiřazení hodnoty při splnění podmínky, nebo přiřadí hodnoty defaultní umístěnou za „ELSE“, je realizováno nastavení hodnoty signálu pro vysílání. Tento signál dává na výstup platná data a v případě výskytu stuff bitu, či fixního stuff bitu se použije poslední vyslaný bit po jeho znegování jako doplňkový stuff bit, nebo FSB.

obr. 27 Vývojový diagram podmínky pro stuffing mimo proces v bloku BSL



Proces řízení stavového automatu bloku BSP v bloku BSL. Na obr. 28 je vidět logika procesu pro řízení automatu v bloku BSP umístěná v bloku BSL.

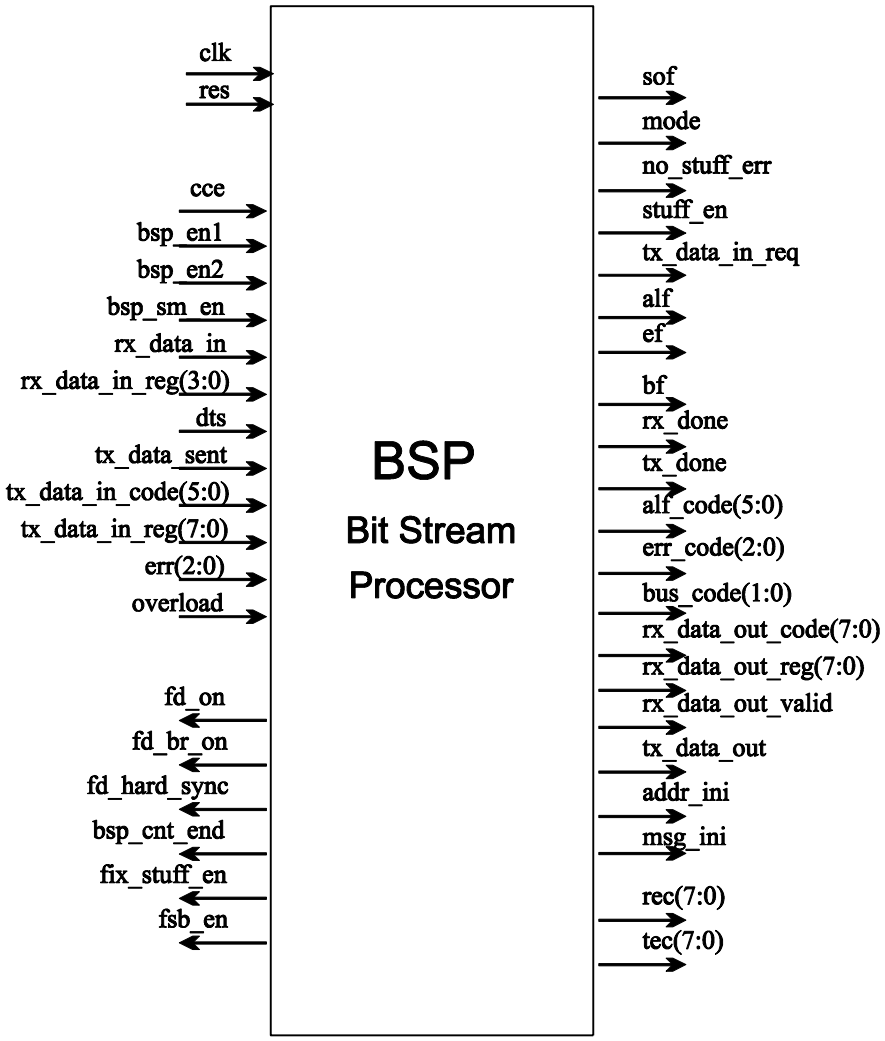
Na úvod je provedena inicializace po resetu. Dále jsou monitorovány hodiny a povolení činnosti bloku BSL.

V případě povoleného mechanismu stuffingu a detekce posloupnosti 5ti stejných bitů následovaných komplementárním bitem, tedy (000001 nebo 111110), je poslední bit posloupnosti stuff bit. Pro tento případ se nastavuje řídící signál *bsp_sm_en* do logické nuly, čímž se deaktivuje činnost stavového automatu bloku BSP. To má za následek nezpracování aktuálního bitu, jimž je stuff bit. V opačném případě je automat ponechán v provozu.

Další kontrolou je kontrola výskytu Fixního stuff bitu, který jak bylo v teoretické části a specifikaci zmíněno, je umístěn za každým 4. bitem, tedy na každém pátém místě CRC pole. Toto místo je hlídáno čítačem, který v okamžiku výskytu FSB prochází nulou. To při povoleném mechanismu fixního stuffingu způsobí, že je stavový automat bloku BSP deaktivován.

obr. 28 Vývojový diagram procesu řízení automatu BSP v bloku BSL

BSP (“Bit Stream processor”) a editace



obr. 29 Blok BSP

Tento blok, jak název napovídá, je řídící blok jádra starající se o správné uspořádání posloupnosti bitů tvořící tzv. „bit stream“. Jinak řečeno stará se o kódování a dekódování zprávy na sběrnici. Tyto posloupnosti bitů pak odpovídají rámcům zprávy dle standardů CAN. Toto vykonává s pomocí stavových automatů a nastavených pravidel. Rovněž je tento blok klíčový, co se komunikace a interface týká. Tedy podávání informací o přijaté zprávě společně s podáním vybraných vlastností a DAT s aktivačním signálem pro platná data. Obdobně v případě vysílání je komunikace s interface například o žádostech o další data k odeslání po předchozím naplnění a nastavení registrů pro vysílání. Módy řadiče jsou rozlišovány signálem *mode*, který je v případě vysílání v logické jedničce a v logické nule při příjmu. V případě vysílání je pomocí bloku BTL a BSL generován příslušný bit na sběrnici, a to v okamžiku synchronizačního segmentu bitu.

Pro správné řízení dalších dvou bloků BSL a BTL jsou zřízeny řídící/informační či aktivační signály, jako je například signál *fd_on*. Tento ve stavu *st_fdf*, tedy při příjmu FDF bitu, nastavuje hodnotu *fd_on* signálu v závislosti na hodnotě signálu nesoucí navzorkovanou hodnotu

sběrnice (*rx_data_in*). V případě logické jedničky na *rx_data_i*n je nastaven signál *fd_on* do log úrovně jedna, naopak při logické nule je nastaven signál na log. nulu.

Dalším novým, řídícím signálem je signál *fd_br_on*, který informuje s předstihem jednoho bitu blok BTL o potenciálním přepínání bitové rychlosti v okamžiku bodu vzorkování (SP) nadcházejícího bitu tak, aby nedocházelo k žádnému zpoždění při přepnutí vlivem vyhodnocování apod.

Nový signál je také *fd_hard_sync*, který je nastaven v logické jedničce v rozmezí přechodu mezi FDF a res bitem. Toto je za účelem povolení druhé „hard" synchronizace vykonávané v okamžiku detekce spádové hrany z FDF na res bit.

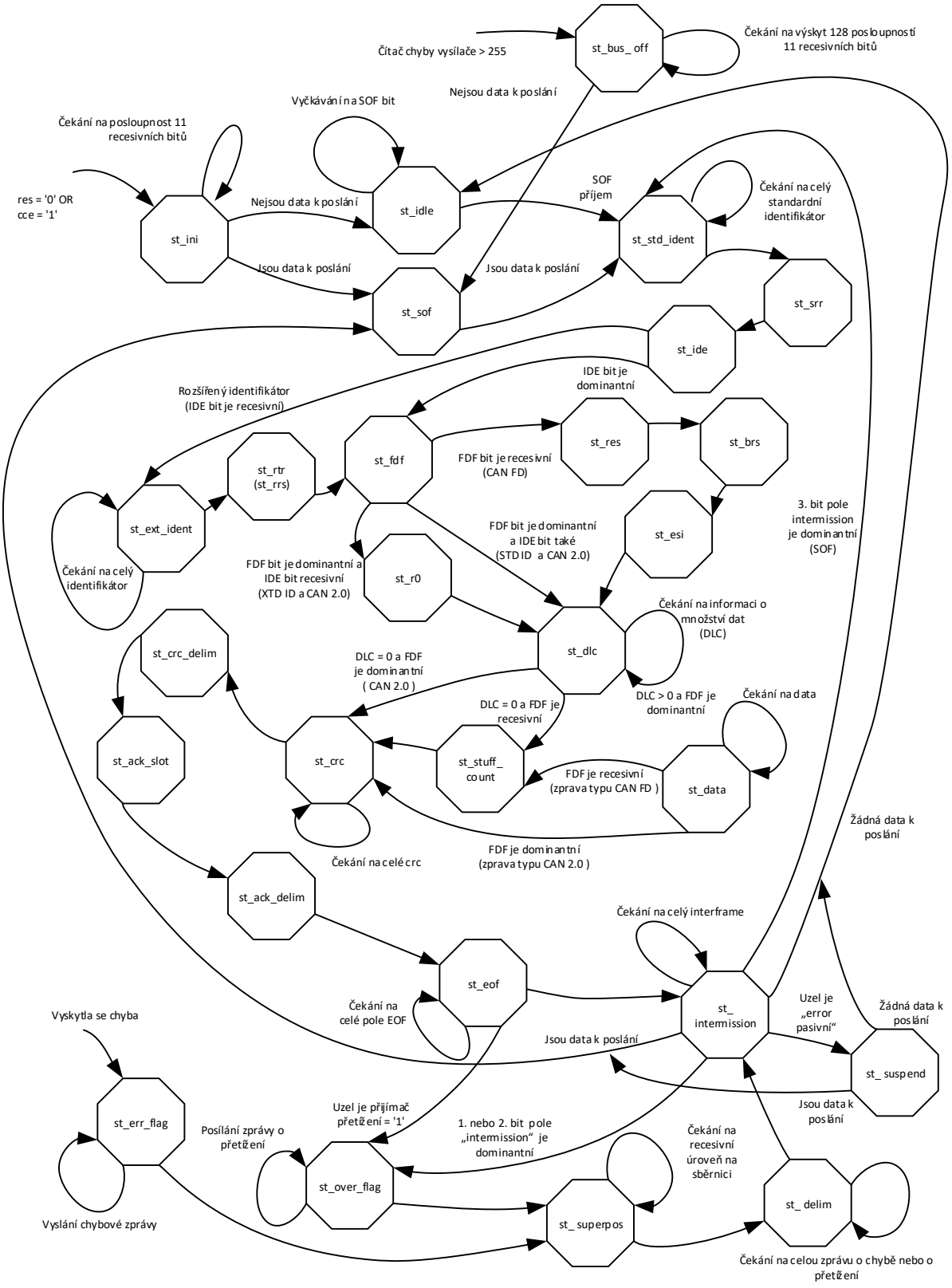
Signál *bsl_fs_en* slouží k řízení bloku BSL, a sice řízení Fixních Stuff bitů, ať už vkládání při vysílání či jejich eliminaci v příchozí zprávě. Tento signál je nastaven do logické jedničky o bit dříve, než se zdá, že je potřeba. Opak je pravdou, jedná se o další případ předčasného nastavení za účelem vykonání včasné operace. Zde jde o zachycení okamžiku prvního fixního stuff bitu na pozici prvního bitu pole „stuff count". K tomu dopomáhá signál *fix_stuff_en*, který již přesně v onen okamžik uvozuje činnost týkající se FSB.

Signál použitý pro přesné načasování zpětného přechodu z bitové rychlosti datová fáze FD rámce na nominální bitovou rychlost, a to v okamžiku vzorkovacího bodu (SP) pole CRC delimiteru, je *bsp_cnt_end*.

Stavový automat

Povolení činnosti stavového automatu, má na starost signál *bsp_sm_en*. Při signálu bsp_en1 bylo provedeno vyhodnocení příchozího bitu. Při signálu bsp_en2 je pak generována odezva na příchozí bit. Automat bylo třeba rozšířit o nové stavy a nová pravidla spjata s těmito novými stavy automatu. Jedná se o *st_fdf* (FDF bit), *st_res* (res bit), *st_brs* (BRS bit), *st_esi* (ESI bit), *st_stuff_count* (Stuff count posloupnost) a také *st_rrs* (RRS bit). Automat je založen na tzv. CASE struktuře. Touto strukturou je definován následující stav automatu. Např. BRS bit následuje ihned po res bitu, tudíž stav *st_brs* je nastaven jako hodnota *next_state* v případě, že aktuální stav *curr_state* je *st_res*. Další součástí je definování podmínek pro přechod do dalšího stavu, případně přechod do stavu jiného, než je obvyklé. Například z bitu FDF se dle jeho hodnoty přechází při recesivní úrovni na stav *st_r0* (CAN 2.0) či při dominantní úrovni na *st_res* (FD).

Stavový automat je vlivem existence stavů, které mají délku více než 1 bit, odvíjející se od čítačů. Pro detekci více případu, například volné sběrnice či inicializace (*ini_cnt*) či 128 výskytů 11 stejných recesivních bitů po sobě, slouží čítač (*bus_off_cnt*). Nejdůležitějším čítačem z hlediska stavů automatu je čítač *cnt*, použitý k realizaci vícebitových stavů automatu. Pro detekci konce stavu je použit signál *cnt_end* nastavovaný do log1 v případě posledního bitu daného stavu. Ten je použit pro správný přechod mezi stavy, kdy je čítač nulován pomocí signálu *clr_cnt*.



obr. 30 Stavový diagram stavového automatu bloku BSP

Časování připojování front zpráv

Pro časování správné práce s frontami zpráv je v bloku BSP sada signálů. Jsou to *msg_ini, rx_data_out_valid, rx_data_out_reg(7:0), rx_data_out_code(5:0)* a *rx_done*.

Informaci o zahájení příjmu zprávy dodává signál *msg_ini*, který je nastaven v okamžiku přechodu stavového automatu do stavu se standardním identifikátorem *st_std_ident*. Posuvný registr *rx_data_out_reg(7:0)* je plněn identifikátorem a následně datovými bity. Po naplnění tohoto posuvného registru je generován informační signál o naplnění, resp. pokyn k vyčtení platných dat *rx_data_out_valid*. K informování o ukončení přenosu je generován signál *rx_done* a ve stejný okamžik je řídící registr *rx_data_out_code(5:0)* naplněn informacemi a vlastnostmi dané zprávy.

Pro správné časování práce s frontami, v případě vysílání zpráv, jsou připraveny signály: *addr_ini, dts, tx_data_in_req, tx_data_in_reg(7:0), tx_data_in_code(5:0)* a *tx_done*.

Podobně jako u příjmu je i u vysílání signál informující o okamžiku přenosu, zde konkrétně uvozuje vhodnou chvíli pro vysílání signál *addr_ini*. Dalším krokem je splnění podmínek, kterými jsou signál *dts* informující o tom, že jsou data k odeslání („data to send“- *dts*) je nastaven do log 1, a zároveň signál *tx_data_in_code(5:0)* musí být naplněn řídícími informacemi dané zprávy. V neposlední řadě musí být signál *tx_data_in_reg(7:0)* naplněn nejvyšším Bajtem standardního identifikátoru dané zprávy, která má být odeslána. Tento registr je pak dále plněn v závislosti na fázi přenosu, kdy je vyžádán přísun dalších dat vždy s předstihem, tedy před vysláním celého Bajtu včetně posledního bitu, a to pomocí signálu *tx_data_in_req*. Po úspěšném odeslání je nastaven signál *tx_done* do logické 1 a signál *dts*. V případě že není připravena další zpráva pro vysílání, přechází do log 0.

Řídící signály

Dále jsou vytvořeny procesy pro nastavování řídících signálů, jako jsou *fd_on, fd_hard_sync, fd_br_on* a další. V tomto procesu se mění hodnota signálu v případě splnění podmínky. Tyto procesy nesplňují podobu standardní IF/ELSE struktury, ale používají pouze IF a ELSIF, jelikož bylo potřeba docílit toho, aby se hodnota z logické 0 změnila na logickou 1 jen a pouze v konkrétních případech, stejně tak i v případě opačném. V jiných případech je třeba, aby signál v této úrovni setrvával a neměnil se. Například signál *fd_on* se nastaví do log 1 okamžitě, jakmile je ve stavu FDF navzorkována recesivní hodnota. V této úrovni (log. 1) pak setrvává až do stavu *st_ack* nebo v případě chyby do stavu *st_err_flag*.

Čítač bitů

Čítač „*cnt*“ v bloku BSP počítá jednotlivé bity. Jelikož některé stavy, jako například DLC, DATA, stuff count a další, jsou delší než jeden bit, jsou zavedeny proměnné a konstanty definující referenční hodnotu počtu bitů daného stavu, tedy například DLC má bity 4. U referenční hodnoty počtu bitu stavu *st_data* je toto číslo nastavováno dle DLC hodnoty. Jakmile čítač dosáhne posledního bitu daného stavu, je okamžitě nastaven takzvaný ukončující signál *ctn_end* signalizující poslední bit stavu. Zároveň je čítač nulován. Tento *cnt_end* signál následně figuruje v procesu, jenž řídí přechod na další stav, tedy překlopení hodnoty *next_state* do *curr_state*.

CRC výpočet

Proces starající se o výpočet CRC byl značně rozšířen. Jelikož je potřeba paralelní výpočet všech tří CRC kombinací (CRC15, CRC17, CRC21) do doby než je na základě bitů DLC a DAT zvolena jen jedna z kombinací, jsou vytvořeny 3 procesy, které se liší v použitém řídícím polynomu pro výpočet CRC sekvence a ve výchozí hodnotě pro výpočet. Jsou zde zavedeny 2 řídící signály použité pro CRC procesy. První je signál *crc_cnt_en* povolující použití bitu k výpočtu CRC sekvence. Druhý signál *crc_shift_en* je pak použit pro povolení vysouvání, respektive čtení bitů (výsledné CRC sekvence) bit po bitu.

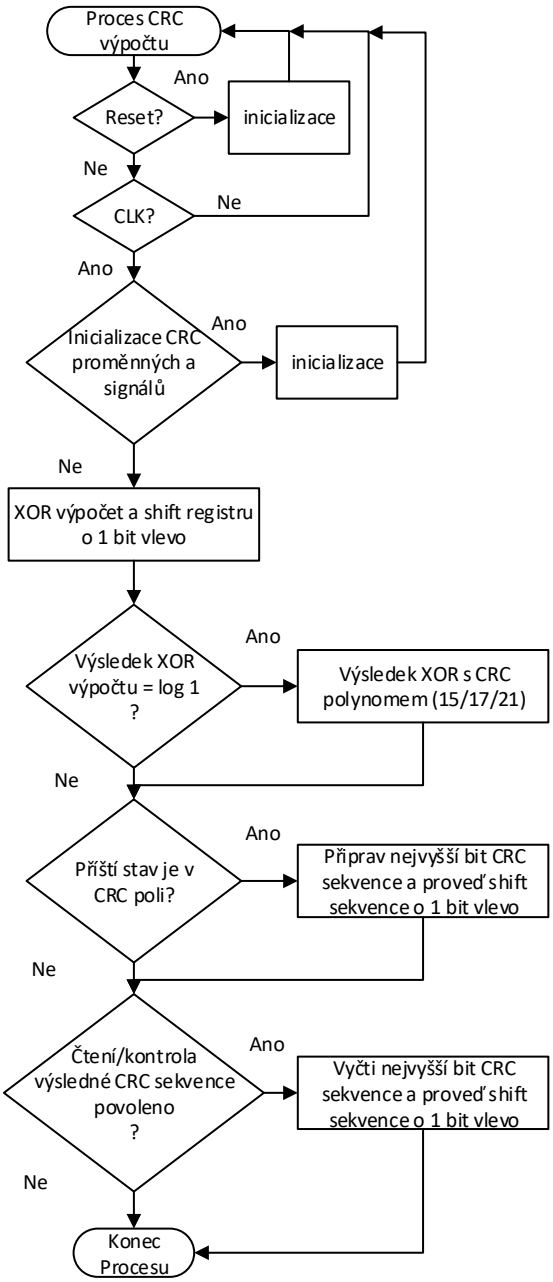
Jde-li o mód přijímače, pak se v procesu, který kontroluje CRC chyby, tyto bity komparují s aktuálním bitem CRC sekvence, která je přijímána. V případě vysílače je vysouvaný bit přímo vysílán. Dojde-li k detekci rozdílných bitů CRC, tedy neshody sekvence, nedochází k okamžitému generování chybové zprávy, ale jen nastavení signálu *crc_err* do log 1. Dále se vyčkává až na dokončení zprávy, tedy jakmile dojde na stav ACK. Pak po ACK delimiteru je vyslána chybová zpráva CRC erroru.

Řízení fix-stuffingu

V bloku BSP jsou generovány hodnoty signálu *fsb_en* a *fix_stuff_en*. Tyto signály jsou řídící pro blok BSL a také pro část tykající se kontroly fixních stuff bitů. Signál *fsb_en* je z důvodu docílení včasné akce nastaven o bit dříve do logické jedničky, tedy ve stavu *st_res*. Následně pak přesně ve správnou chvíli, když je již blok BSL díky *fsb_e*n připraven, dodá blok BSP signálem *fix_stuff_en* pokyn k vkládání/vyjmutí FSB bitů při shodě s čítačem bitů v BSP. Obdobně jsou hodnoty signálů nulovány, aby k vkládání/vyjmutí nedocházelo mimo specifikací stanovený rozsah.

Vývojové diagramy (BSP)

V této části je věnován prostor popisu hlavních procesů formou vývojových diagramů.



obr. 31 Vývojový diagram procesu výpočtu CRC sekvence v bloku BSP

Proces výpočtu CRC sekvence v bloku BSP. Na obr. 31 je viděl vývojový diagram logiky výpočtu CRC sekvence, která je umístěna v bloku BSP.

Po kontrole reset signálu a případné inicializaci po resetu, je monitorována úroveň hodinového signálu. Dále je kontrolováno, zda jde o novou zprávu, a tedy zda je třeba provést inicializaci potřebných signálů pro CRC proces.

Pokud je vše připraveno je provedena logická operace XOR (Exclusive OR) aktuálního bitu z bitové posloupnosti, která je vysílána či přijímána. Pokud je výsledek této operace logická jednička, je následně provedena další operace XOR CRC polynomu a aktuálního bitu dané zprávy.

Další kontrolou je, zda je příštím stavem CRC pole. Pokud je tomu tak, bude probíhat vyčítání sekvence. Ta bude bit po bitu kontrolována s CRC přijatým bitem, který odpovídá bitu CRC sekvence vysílače. (Toto v případě příjmu zprávy.) V případě vysílání bude docházet pouze ke čtení vypočtené sekvence bit po bitu, což je následováno vždy posunem registru s touto sekvencí o jeden bit vlevo. Čtený bit vždy z nejvyšší pozice, tedy z pozice bitu MSB.

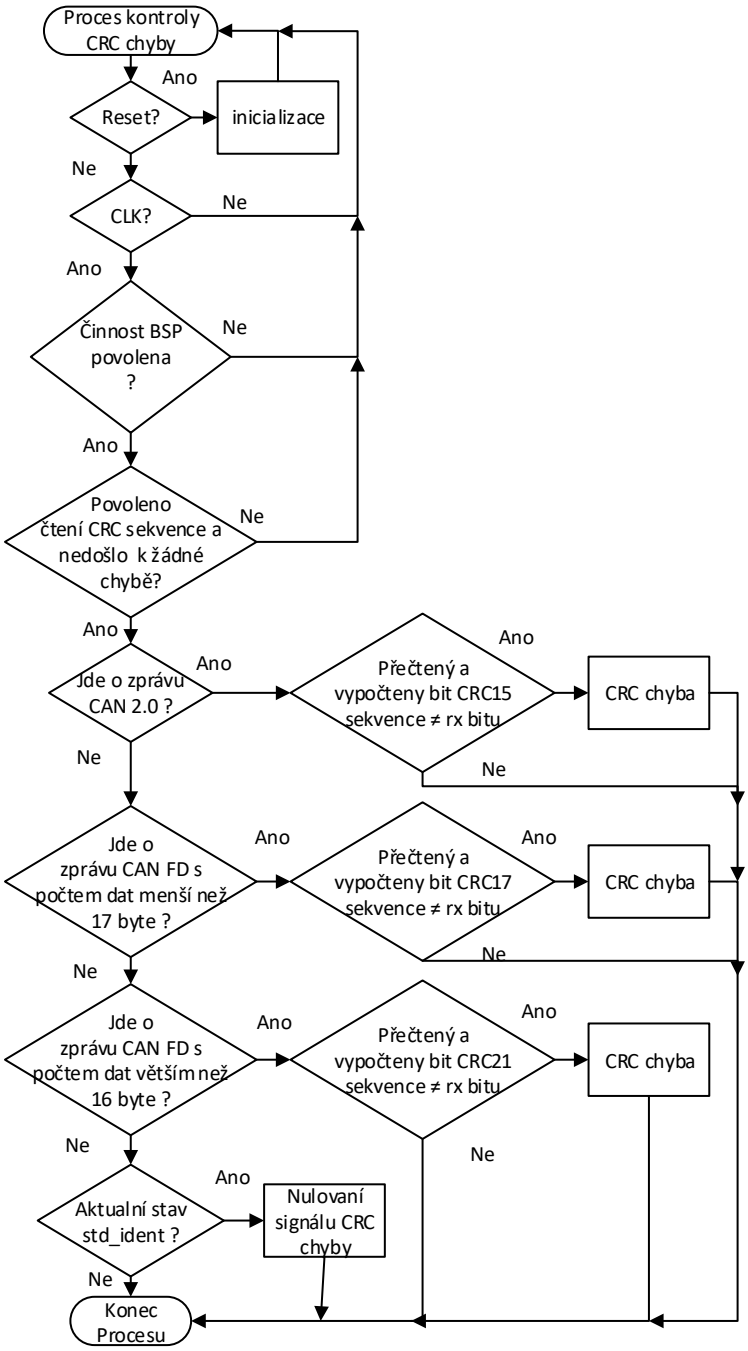
Tento proces je téměř identický pro výpočet všech tří sekvencí CRC, tedy CRC15, CR17 a CRC21. Jediným rozdílem je rozdílná inicializační hodnota proměnné pro výpočet sekvence a samozřejmě řídící polynom. Z důvodu potřeby připravenosti CRC sekvence jsou, jak je specifikací určeno, počítány všechny 3 CRC sekvence současně. Z tohoto důvodu je tento proces v kódu použit 3krát s modifikací polynomu a inicializačních hodnot. V okamžiku rozhodnutí o výběru CRC sekvence na základě DLC kódu ze 3 připravených je vše připraveno.

Proces kontroly CRC chyby v bloku BSP. Na obr. 32 je vidět logika procesu kontroly CRC chyby.

Po resetu řadiče a inicializaci signálu a proměnných je monitorována hodnota hodinového signálu a také kontrolováno povolení činnosti bloku BSP.

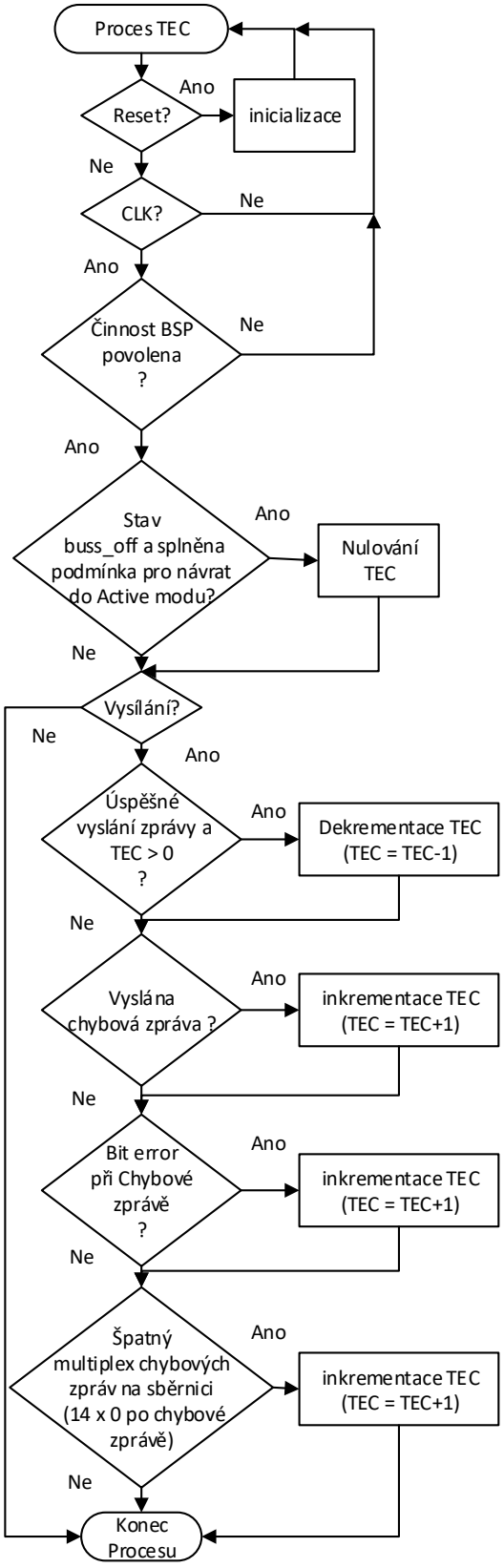
Pokud je činnost povolena, je kontrolována podmínka rozlišující druh zprávy. Jedná-li se o zprávu klasického CAN 2.0 standardu, je provedena kontrola bitu s bitem sekvence CRC15. V případě nerovnosti je generován chybový signál informující i výskytu CRC chyby.

V případě, že se jedná o zprávu standardu CAN FD, je kontrolována ještě hodnota DLC, tedy počet datových bajtů za účelem určení použité CRC sekvence, která se tímto parametrem řídí. Pokud je hodnota počtu datových bajtů pod číslem 17, je použita metoda CRC17, a tedy kontrolovaný bit je porovnáván s hodnotou CRC sekvence vypočtené dle polynomu pro CRC17. Jedná-li se o opačný případ, tedy o FD zprávu s počtem datových bajtů, který je vyšší než 16, tak je použita sekvence vypočtená dle polynomu pro CRC21.



obr. 32 Vývojový diagram detekce CRC chyby v BSP

Posledním blokem je kontrola stavu automatu, a sice pokud je aktuální stav mimo CRC pole, konkrétně v poli standardního identifikátoru, tak je vynulován signál nesoucí informaci o případném výskytu chyby CRC.



obr. 33 Vývojový diagram procesu Transmit error counteru

Proces čítače chyb při vysílání v bloku BSP. Na obr. 33 je vidět vývojový diagram procesu čítače chyb vysílače, který je umístěn v bloku BSP.

Po úvodní kontrole resetu a případné inicializaci proměnných a signálů, je monitorována hodnota hodinového signálu. V případě, že je povolena činnost bloku BSP, je vývoj procesu následující.

První je provedena kontrola módu řadiče. V případě, že se nachází v *buss_off* módu a zároveň je splněna podmínka pro návrat do aktivního módu, je čítač chyb nulován. V jiném případě je kontrolována činnost, tedy zda se vysílá, či ne. V případě vysílání jsou kontrolovány 4 podmínky. Nejprve se kontroluje, zda je hodnota čítače chyb TEC nenulová, a v případě úspěšného vysílání je čítač dekrementován, tedy snížen o jedničku. V případě vyslání chybové zprávy je naopak čítač zvětšen, inkrementován. Vyskytne-li se chyba při samotné chybové zprávě, odpovídá to chybě bitu a je tedy opět navýšena hodnota čítače o 1. Poslední z kontrolovaných možností je situace, kdy dojde ke špatnému seskupení chybových zpráv na sběrnici po odeslání chybové zprávy. Následek tohoto případu je opět inkrementace, tedy navýšení čítače TEC o 1.

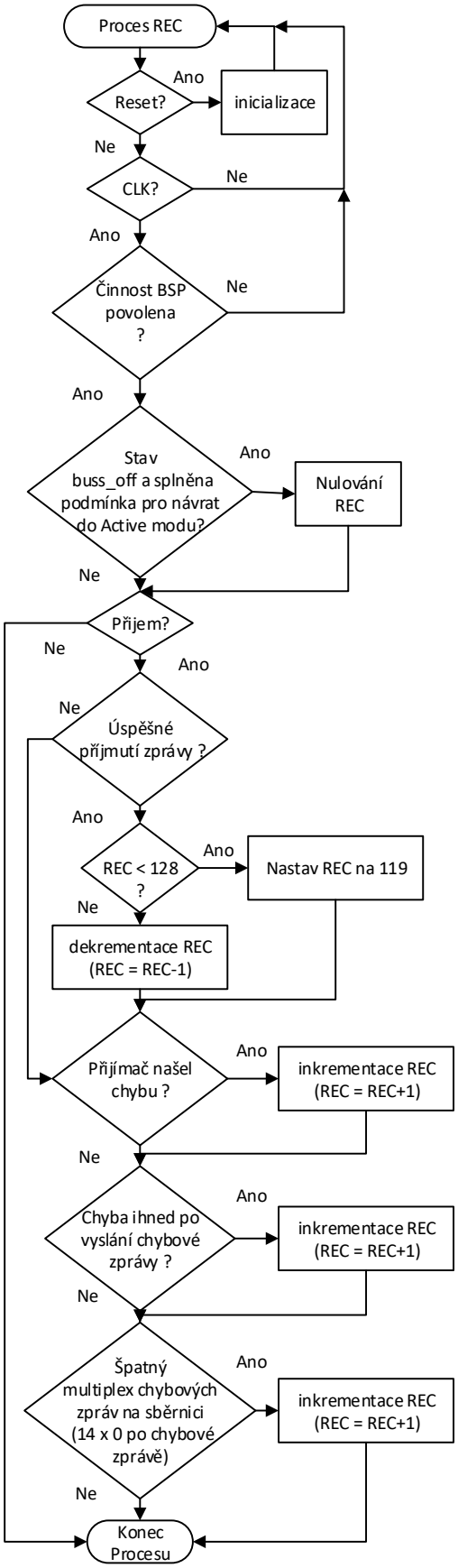
Proces čítače chyb při příjmu.

Na obr. 34 je vidět vývojový diagram procesu čítače chyb při příjmu, který je umístěn v bloku BSP. Na úvod jako většina procesů je kontrolován reset signál s případným provedením inicializace všech potřebných signálů a proměnných. Dále pak je sledována hodnota hodinového signálu.

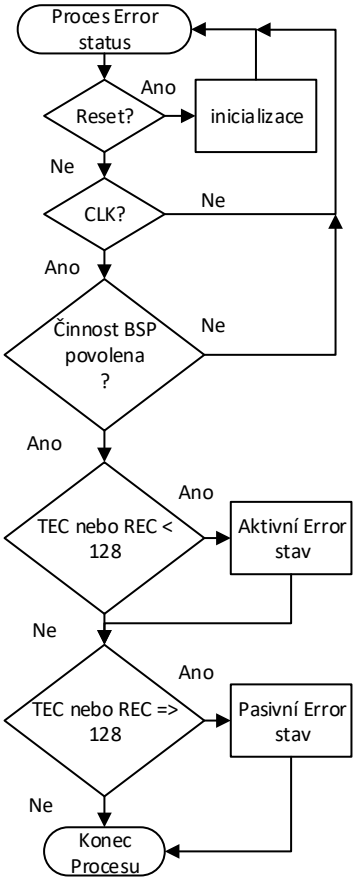
Následně, v případě povolení činnosti bloku BSP je kontrolován stav řadiče. Pokud se řadič nachází ve stavu *buss_off* a je zároveň splněna podmínka pro návrat do aktivního módu, tak je čítač nulován. V jiném případě, či po předchozím kroku, je kontrolováno, zda se jedná o mód příjmu zprávy.

V případě, že jde o mód příjmu a dále pak, že byla úspěšně přijata zpráva, bude čítač snížen o jedničku. To ale jen v případě, že je jeho hodnota větší než 128 V opačném případě je nastaven na hodnotu 119. V případě detekce chyby při příjmu je prováděno navýšení (inkrementace) čítače chyb REC o jedničku.

Další kontrola je prověření výskytu chyby ihned po vyslání chybové zprávy. Pokud je podmínka splněna, je opět čítač zvětšen o hodnotu 1. Poslední kontrolou je detekce špatného uspořádání chybových zpráv na sběrnici. Při splnění této podmínky je opět navýšena hodnota čítače o jedničku.



obr. 34 Vývojový diagram procesu Receive error counter



obr. 35 Vývojový diagram procesu error status

Proces nastavování chybového stavu.

Na obr. 35 Je vidět proces nastavení chybového stavu řadiče v závislosti na chybových čítačích. Tento proces je umístěn v bloku BSP.

Po úvodním resetu, inicializaci proměnných a signálů je kontrolována hodnota hodinového signálu.

Pokud je povolena činnost bloku BSP, jsou kontrolovány hodnoty chybových čítačů (TEC a REC) pracujících dle vývojových digramů uvedených výše. Konkrétně je tedy kontrolována hodnota čítače chyb vysílače (TEC) a také hodnota čítače chyb přijímače (REC). V případě, že je hodnota jednoho z nich menší než 128 a odpovídá to aktuálnímu módu, tak je nastaven Aktivní error status.

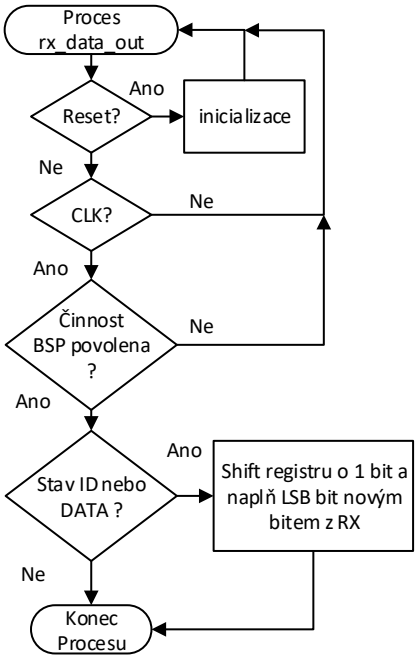
Další kontrolou je opět hodnota těchto dvou čítačů. Tentokrát jde o kontrolu, zda nejde o situaci Pasivního error stavu. To je v případě, že při odpovídajícím módu je hodnota čítače rovna nebo větší než číslo128.

Proces registru přijatých hodnot.

Logika procesu s registrem přijatých hodnot je vidět na obr. 36.

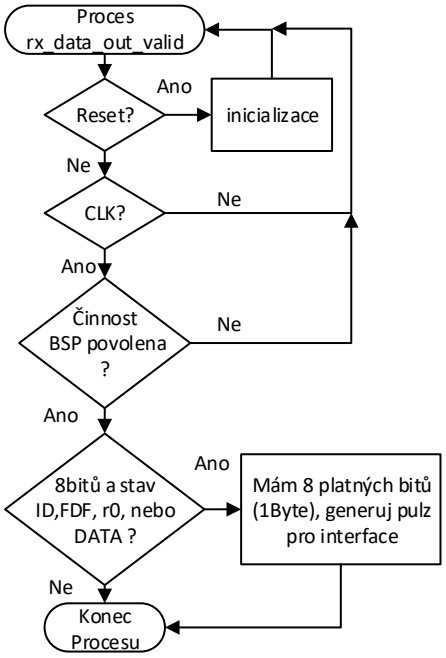
Na úvod je vždy po resetu provedena inicializace signálů a proměnných. Dále je monitorována hodnota hodinového signálu.

Pokud je povolena činnost bloku BSP, je kontrolována jediná podmínka procesu. Tato podmínka obsahuje kontrolu aktuálního stavu stavového automatu. Konkrétně zda se automat nachází ve stavu s identifikátorem nebo ve stavu s daty. Pokud je tato podmínka splněna, tak je proveden takzvaný shift registru o jednu pozici vlevo. Jde o bitový posun, kdy se na uvolněnou pozici LSB vsune nový přijatý bit a zároveň je použita hodnota MSB před posunutím, a to pro poskytnutí bitu dalšímu bloku.



obr. 36 Vývojový diagram procesu registru přijatých hodnot

Proces logiky signálu *rx_data_out_valid* v bloku BSP.



obr. 37 Vývojový diagram potvrzení platných dat v příchozím registru

Logika procesu se signálem *rx_data_out_valid* je vidět na obr. 37.

Po zapnutí řadiče je proveden reset, v tomto okamžiku jsou inicializovány všechny signály a proměnné do výchozích hodnot. Následně je pak kontrolována také hodnota hodinového signálu.

V případě povolení činnosti bloku BSP, je v tomto procesu kontrolována jediná podmínka. Tato podmínka ověřuje stav automatu, a v případě že tento stav odpovídá identifikátoru, FDF bitu, bitu r1 či datovému poli, tak je po přijetí 8 bitů nastaven signál *rx_data_out_valid* do logické jedničky. Tento slouží jako informace pro blok interface o naplnění registru hodnot platnými daty a má tedy dojit k jejich vyčtení do bloku přijímacího FIFA a vyprázdnění registru za účelem přípravy pro další platná data. V okamžiku úspěšného vyčtení je signál navrácen do logické nuly.

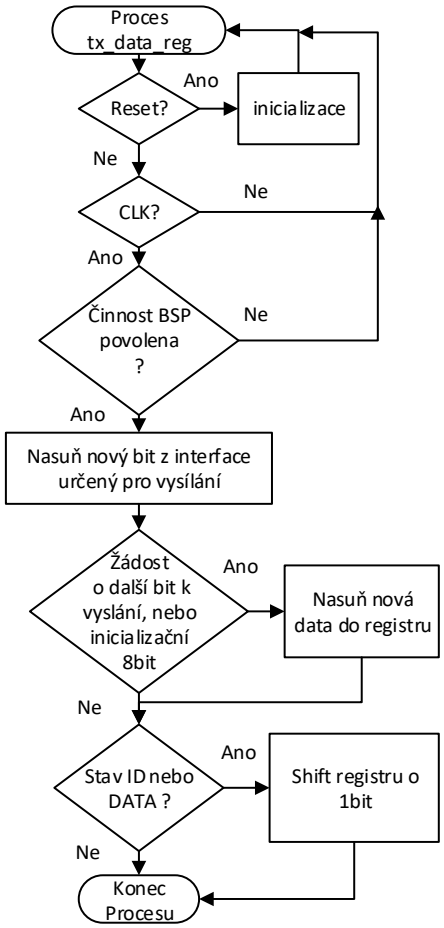
Proces logiky registru vysílaných hodnot.

Logika procesu s registrem vysílaných hodnot je vidět na obr. 38.

Po resetu je provedena inicializace, tedy nastavení výchozí hodnoty signálů a proměnných. V další fázi je kontrolován hodinový signál.

Pokud je povolena činnost bloku BSP, tak je připraven další bit k vyslání nasunutím do registru. Další v pořadí je kontrola, zda již došlo k vyslání a je potřeba dalších hodnot. Tato kontrola je prováděna dříve, než je registr vysílaných hodnot zcela vyprázdněn, tak aby nasouvaná data byla připravena dříve a naplnění registru proběhlo bez problémů.

Také je monitorován stav stavového automatu, tedy v případě, že je aktuálním stavem oblast identifikátoru nebo oblast dat, pak je registr přijatých hodnot posunut o bit vlevo přičemž MSB bit, který je vysunut z registru, je vysílán jako platný bit daného pole, tedy identifikátor nebo datový bit.



obr. 38 Vývojový diagram procesu vysílacího registru dat

2.2.2 Paměťový prostor – Registry

Po úpravě registrů z 8bitových na 32bitové a zvážení všech potřebných jednotlivých registrů je vygenerována následující registrová mapa, která vyžaduje paměťový prostor 256 Bajtů a je umístěna na adresách 0x00 až 0x1F. To je vidět ve sloupci označeném „Offset“. V rozmezí adres 0xA0 až 0xC4 je prostor registrů přerušení (tento prostor není popsán v této práci z důvodu jeho nevyužití). Rozbor a popis smyslu jednotlivých registrů následuje na dalších stranách.

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**CANCtrl0**	En			TxMsgProcEn			RxMsgFiltEn	RxMsgProcEn			Pol	NumSam			JW [1:0]			TS2 [2:0]			TS1 [3:0]				BRP [7:0]							
	reset value	0			1			1	1			0	1			1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x04	**CANCtrl1**			SSPOffset [5:0]									TDCEn			JWFD [1:0]			TS2FD [2:0]			TS1FD [3:0]				BRPFD [7:0]							
	reset value			0	0	0	0	0	0				0			1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x08	**CANStat**											ARBLost [5:0]						TEC [7:0]								REC [7:0]							
	reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**CANRxFIFO**	RxFIFOFull	RxFIFOEmpty	RxFIFOLimitMore							RxFIFONumBytes [6:0]							RxFIFOFlush								RxFIFOLimit [6:0]							
	reset value	0	0	0							0	0	0	0	0	0	0	0								0	0	0	0	0	0	0	0
0x10	**CANTxFIFO**	TxFIFOFull	TxFIFOEmpty		TxFIFOLimitLess						TxFIFONumBytes [6:0]							TxFIFOFlush								TxFIFOLimit [6:0]							
	reset value	0	0		0						0	0	0	0	0	0	0	0								0	0	0	0	0	0	0	0
0x14	**CANFIFOData**	TxRxFIFOData [31:0]																															
	reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**CANMask0Std**																	Mask0Stdf1 [2:0]								Mask0Stdf0 [7:0]							
	reset value																	1	1	1						1	1	1	1	1	1	1	1
0x1C	**CANMask0Ext**	Mask0Extf3 [4:0]								Mask0Extf2 [7:0]								Mask0Extf1 [7:0]								Mask0Extf0 [7:0]							
	reset value	1	1	1	1	1				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Kontrolní registr číslo 0

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**CANCtrl0**	En			TxMsgProcEn			RxMsgFiltEn	RxMsgProcEn			Pol	NumSam			JW [1:0]			TS2 [2:0]			TS1 [3:0]				BRP [7:0]							
	reset value	0			1			1	1			0	1			1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	register type	RW			RW			RW	RW			RW	RW			RW			RW			RW				RW							

BRP „Baud Rate Prescaler“ Dělička přenosové rychlosti sběrnice CAN
Minimální dělící poměr je 1:2 (00000001 b)

TS1 „Timing Segment 1“ Časový segment 1
Počet časových úseků (TQ) realizujících časovací segment 1

TS2 „Timing Segment 2“ Časový segment 2
Počet časových úseků (TQ) realizujících časovací segment 2

JW „Jump Width“ Šířka synchronizačního skoku

NumSam Počet vzorkování sběrnice pro stanovení hodnoty jednoho bitu (SP)

Pol Polarita dominantní úrovně sběrnice CAN

RxMsgProcEn Povolení Filtrace příchozích zpráv
Při log. 0 jsou přijímány všechny příchozí zprávy

RxMsgFiltEn Povolení přijímacího procesu

TxMsgProcEn Povolení vysílacího procesu

En Centrální povolení funkce CAN kontroléru

Kontrolní registr číslo 1

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x04	**CANCtrl1**			SSPOffset [5:0]									TDCEn			JWFD [1:0]			TS2FD [2:0]			TS1FD [3:0]				BRPFD [7:0]							
	reset value			0	0	0	0	0	0				0			1	1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	register type			RW									RW			RW			RW			RW				RW						

BRPFD Dělička přenosové rychlosti sběrnice CAN – část FD
Minimální dělící poměr je 1:2 (00000001 b)

TS1FD „Timing Segment 1 “ Časový segment 1 – část FD
Počet časových úseků (TQ) realizujících časovací segment 1

TS2FD „Timing Segment 2“ Časový segment 2 – část FD
Počet časových úseků (TQ) realizujících časovací segment 2

JWFD „Jump Width“ Šířka synchronizačního skoku – část FD
Počet časových úseků (TQ) použitých k resynchronizaci

TDCEn „Transmit Delay Compensation“ Kompenzace zpoždění vysílače
Povolení použití mechanismu (log 1)

SSPOffset offset SSP – sekundárního vzorkovacího bodu (SP)
Uživatelem nastavitelný offset použitý pro výpočet pozice SSP

Status registr CAN kontroléru

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x08	**CANStat**											ARBLost [5:0]						TEC [7:0]								REC [7:0]							
	reset value											0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	register type											RO						RO								RO							

REC „Receive Error Counter Register“ Registr čítače chyb při příjmu

TEC „Transmit Error Counter Register“ Registr čítače chyb při vysílání

ARBLost „Arbitration Lost Code Register“
Registr uchovávající informaci o bitu, při kterém došlo ke ztrátě arbitru.

Kontrolní a status registr bloku RX FIFO

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0C	**CANRxFIFO**	RxFIFOFull	RxFIFOEmpty	RxFIFOLimitMore							RxFIFONumBytes [6:0]							RxFIFOFlush									RxFIFOLimit [6:0]							
	reset value	0	0	0							0	0	0	0	0	0	0	0									0	0	0	0	0	0	0	0
	register type	RO	RO	RO							RO							W1									RW							

RxFIFOLimit Nastavitelná mez pro naplnění Rx FIFO registru

RxFIFOFlush Bit pro vyprázdnění obsahu bloku Rx FIFO

RxFIFONumBytes Počet obsazených/platných bitu v bloku Rx FIFO

RxFIFOLimitMore Signální bit - překročení nastaveného horního limitu počtu bitů v bloku Rx FIFO hodnotou registru RxFIFOLimit

RxFIFOEmpty Signální bit informující o prázdném bloku RxFIFO

RxFIFOFull Signální bit informující o dosažení kapacity bloku RxFIFO

Kontrolní a status registr bloku TX FIFO

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x10	**CANTxFIFO**	TxFIFOFull	TxFIFOEmpty		TxFIFOLimitLess						TxFIFONumBytes [6:0]							TxFIFOFlush									TxFIFOLimit [6:0]						
	reset value	0	0		0						0	0	0	0	0	0	0	0									0	0	0	0	0	0	0
	register type	RO	RO		RO						RO							W1									RW						

TxFIFOLimit Nastavitelná mez pro naplnění Tx FIFO registru

TxFIFOFlush Bit pro vyprázdnění obsahu bloku Tx FIFO

TxFIFONumBytes Počet obsazených/platných bitu v bloku Tx FIFO

TxFIFOLimitLess Signalizace detekování méně bitů, než je v TxFIFOLimit nastaveno pro blok Tx FIFO

TxFIFOEmpty Signální bit prázdného bloku TxFIFO

TxFIFOFull Signální bit informující o dosažení kapacity bloku TxFIFO

TX/RX FIFO blok

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x14	**CANFIFOData**	TxRxFIFOData [31:0]																															
	reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	register type	WORO																															

TxRxFIFOData CAN TX/RX FIFO DATA

Registr obsahující odchozí zprávu v případě TX FIFO

Byte[0] (7:0) "code register" obsahující FDF bit, IDE bit, rtr a DLC(3:0)

Byte[1] (15:8) obsahuje STD ID (28:21)

Byte[2] (23:16) obsahuje STD ID (20:18) zarovnáno k levé straně a doplněno nulovými bity (4:0)

Byte[3] (31:24) obsahuje XTD ID jen v případě použití XTD ID, v jiném případě jde o první datový Bajt

Byte[4] (7:0) obsahuje XTD ID jen v případě použití XTD ID, v jiném případě jde o datový Bajt

Byte[5-63] (7:0) obsahuje Datové Bajty

Registr obsahující příchozí zprávu v případě RX FIFO

Byte[0] (7:0) "code register" obsahující ESI bit, IDE bit, rtr a DLC(3:0)

Zbylé Bajty dodržují stejný systém obsahu jako při odesílání.

Registr Standardní masky

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x18	**CANMask0Std**																	Mask0Stdf1 [2:0]								Mask0Stdf0 [7:0]							
	reset value																	1	1	1						1	1	1	1	1	1	1	1
	register type																	RW								RW							

Mask0Stdf0 „Standard Identifier Mask Register 0“ - pole bitů (10:3)

Mask0Stdf1 „Standard Identifier Mask Register 0“ - pole bitů (2:0)

Registr Rozšířené Masky

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1C	**CANMask0Ext**	Mask0Extf3 [4:0]								Mask0Extf2 [7:0]								Mask0Extf1 [7:0]								Mask0Extf0 [7:0]							
	reset value	1	1	1	1	1				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	register type	RW								RW								RW								RW							

Mask0Extf0 „Extended Identifier Mask Register 0“ – pole bitů (28:21)
Maska rozšířeného identifikátoru nejvyšších 8bitů

Mask0Extf1 „Extended Identifier Mask Register 0“ – pole bitů (20:13)
Maska rozšířeného identifikátoru

Mask0Extf2 „Extended Identifier Mask Register 0“ – pole bitů (12:5)
Maska rozšířeného identifikátoru

Mask0Extf3 „Extended Identifier Mask Register 0“ – pole bitů (4:0)
Maska rozšířeného identifikátoru nejnižších 5 bitů

2.2.3 Interface a AHB-Lite

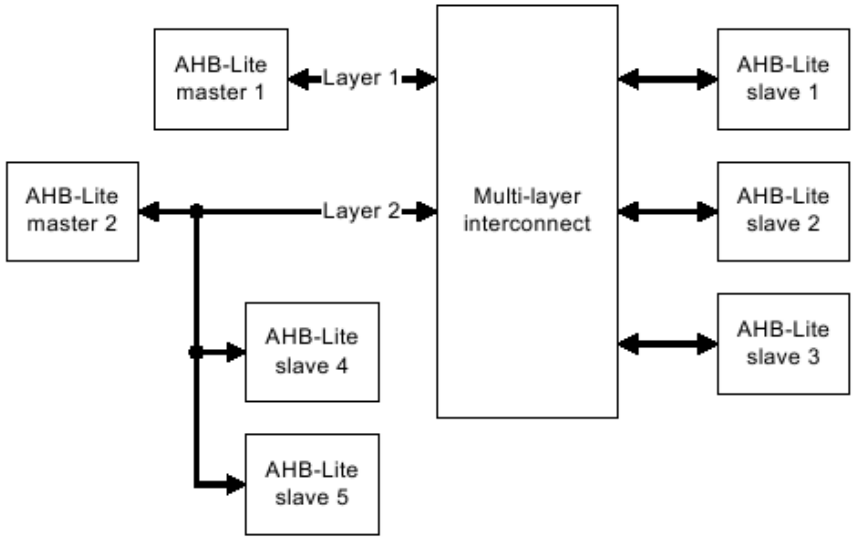
Nutné změny byly třeba udělat také s blokem interface. Jednou ze změn je náhrada původní x51 sběrnice pro komunikaci s mikrokontrolérem, jenž byla založena na tvaru MOVX instrukce za novější sběrnici pracující dle protokolu AHB-Lite.

AHB-Lite

Sběrnicový protokol AHB-Lite je součástí tzv. AMBA 3 architektury. Tedy „Advanced Microcontroller Bus Architecture“ 3, což je sběrnicový protokol pro propojení mezi vnitřními funkčními bloky SoC (Systém on Chip). Tento protokol je dílem autora mikroprocesorového jádra ARM. AHB-Lite je specifikace určité podmnožiny funkcí z úplného AMBA AHB protokolu. Nabízí všechny základní funkce potřebné pro systém AMBA AHB slave - master, obdobně i pro multi úrovňové AMBA propojení. Ve většině případu pro zařízení, které jsou vybaveny plným AMBA AHB rozhraním, je provedena efektivnější implementace použitím protokolu AMBA AXI rozhraní.[5]

Více-úrovňová AHB-Lite

Jelikož je AHB-Lite rozhraní/sběrnice typu takzvaně “single master”, je v případě typu “multimaster” rozhraní potřeba použít komponentu, která oddělí navzájem všechny master jednotky. Každý master blok je umístěn na svou vlastní úroveň, tzn. propojovací komponenta musí vytvořit víceúrovňové propojení, ve kterém jsou master jednotky mezi sebou izolovány, ale dokáží mezi sebou sdílet přístup k slave jednotkám. Arbitrace slave jednotky je prováděna propojovací komponentou.[5]



obr. 39 Přiklad víceúrovňového AHB-Lite uspořádání, blokový diagram [5]

AHB-Lite komunikace mezi řídícím mikrokontrolérem a blokem interface je v našem případě definována jako 8bitová. Používá následující signály HRESET, HCLK, HADDR, HWRITE, HDATA. Možnosti sběrnice dle protokolu jsou od 8bitové až po 256bitové, ale i více.

Z důvodu použití AHB-Lite sběrnice jsem upravil také procedury a funkce pro nastavení registrů řadiče. Tedy funkce pro nastavení správného chování řadiče. Tyto byly ponechány 8bitové, ale upraveny na zmíněný protokol.

Blok Interface

FIFO paměť

Nezbytnou součástí je paměť. Druhy paměti lze rozlišovat podle mnoha kritérií. Podle názvu naší použité paměti připadá v úvahu rozdělení podle způsobu výběru datových položek. Paměť je typu FIFO, tedy z anglického „First In, First Out“, v překladu „první dovnitř, první ven“, což znamená, že data která přišla do zásobníku jako první, jsou také jako první ze zásobníku vyčteny/vyjmuty.

Jiným druhem je například paměť typu LIFO, častěji nazýván jako zásobník, pro který platí, že poslední příchozí data jsou také první, která jsou ze zásobníku vyjmuty, respektive vyčteny. Název je z anglického „Last In, First Out“, tedy poslední dovnitř, první ven.

Upraveny musely být také bloky front FIFO. Na rozdíl od původní logiky věci, kde bylo jedno FIFO pro příjem a druhé FIFO pro odesílání, je nyní použit jeden 32bitový registr, který dle módu řadiče volí přijímací, nebo vysílací frontu FIFO. Tyto FIFO bloky jsou již rozděleny na RX a TX, tedy přijímací a vysílací. Přístup do tohoto registru je možný více způsoby, například přes jeho nejnižší adresu tedy zápis po 8 bitech. Je tedy přistupováno vždy na stejnou adresu, data se posunují po naplnění 8bitů o 1Bajt dále.

Registrová mapa

Registrová mapa viz kapitola 2.2.2 je v porovnání s původní 8bitovou registrovou mapou zcela změněna. Hlavním rozdílem je úprava jednotlivých registrů na 32bitové. Stručné shrnutí a použití výsledných registrů nové registrové mapy je následující.

Na nejnižší adrese začíná kontrolní registr0 (CANCtrl1), který slouží k nastavení hodnoty bit rate předděličky, timing segmentu1 & 2, šířky synchronizačního skoku SJW, povolení filtrace zpráv dle identifikátoru, nastavení masky a povolení vysílacího procesu. Téměř nakonec prvního 32bitoveho registru je umístěn bit „Enable“ který zastaví veškeré hodiny obvodu, tento bit se používá pouze při prvotním nastavení registrů řadiče, pak se bit nastaví zpět do log 1 a dále se s ním již nemanipuluje.

Obdobně je uspořádán kontrolní registr1 (CANCtrl1), který je určen nejen pro definování hodnot pro časování části zprávy v FD formátu. Tedy počínaje předděličkou bitové rychlosti pro datovou fázi zprávy FD, pak časovými segmenty 1 & 2 pro definici nominální velikosti bitu, šířkou synchronizačního skoku pro synchronizaci, povolení mechanismu kompenzace zpoždění vysílače (TDC) a nastavení velikosti offsetu druhého vzorkovacího bodu (SSP).

Další registr (CANStat) nese informace o stavu řadiče. První Bajt má informaci o velikosti čítače počtu chyb přijímače (REC), druhý Bajt obsahuje informaci o velikosti čítače počtu chyb vysílače (TEC), třetí necelý Bajt pak kódovou informaci o místě ztráty arbitru v případě, že k tomu dojde.

Registr příchozí fronty (CANRxFIFO) má v prvním Bajt možnost nastavit horní mez plnosti fronty/paměti. Má také ovládací bit pro vyprázdnění fronty, necelý Bajt pak pro aktuální obsazenost nebo také naplněnost fronty a poslední tři bity registru jsou indikátory dosažení nastaveného limitu, dále informace o prázdném zásobníku a poslední je indikátor plného zásobníku.

Analogicky je sestaven registr odchozí fronty (CANTxFIFO).

Další v pořadí je registr obsahující DATA (CANFIFOData). Tento Registr je schopen pojmout zprávu FD standardu o maximální velikosti 64Bajt a k tomu další Bajty s identifikátorem (standardní či rozšířený 11/29bitů) a necelým Bajtem nesoucím informaci o počtu datových Bajtů, druhu identifikátoru, ESI bit a RTR bit.

Předposlední registr (CANMask0Std) je určen pro standardní masku, respektive masku identifikátoru zprávy, pro filtraci zpráv na sběrnici, a tedy výběru relevantní zprávy. Tedy 11 nejnižších bitů.

Poslední registr (CANMask0Ext) je pro nastavení masky rozšířeného identifikátoru zprávy, která má být filtrem přijata. Tento má 29 bitů.

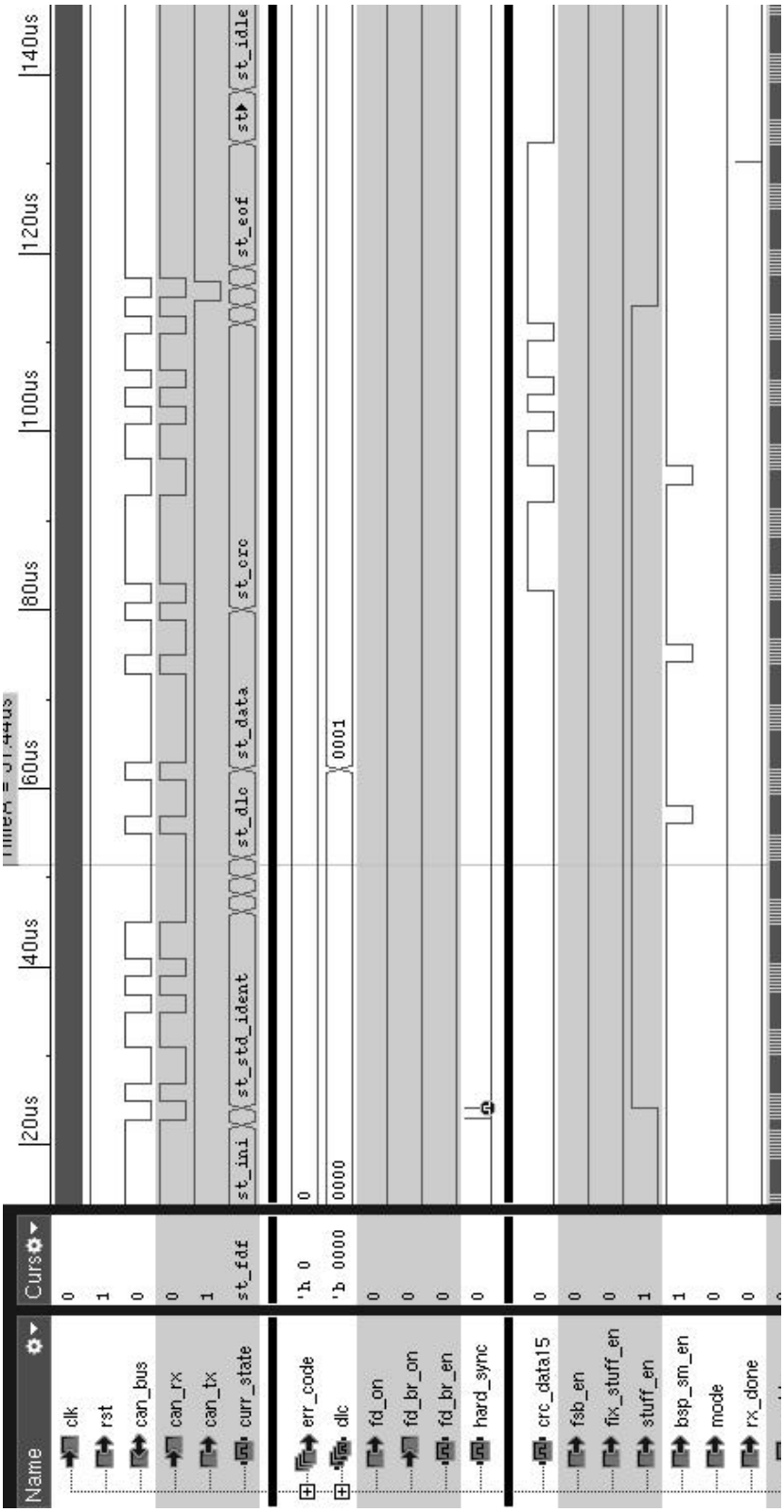
Dále jsou vytvořeny také registry přerušení, které jsou pak uspořádány podle významu. Těmto registrům zde není věnován prostor, jelikož nebyly využity.

2.2.4 Simulace funkce

Po rozšíření logiky všech bloků dle úvahy, která vznikla na základě specifikace, také po sestavení celého designu, propojeni všech bloků nezbytnými signály, je další nezbytnou součástí simulace chování designu za účelem ověření správné funkce.

Vytvořil jsem test, který simuluje zprávu na sběrnici. Tímto je simulován příchod posloupnosti bitů odpovídající určitému typu zprávy se zvolenými parametry. Takto je ověřeno správné chování designu. U testu se předpokládá například úspěšné přijetí zprávy s potvrzením ACK bitem ze strany přijímače a následné ověření shody přijatých hodnot (DATA, ID a další, které se ukládají do registrů) a parametrů, které jsou nastaveny v testu.

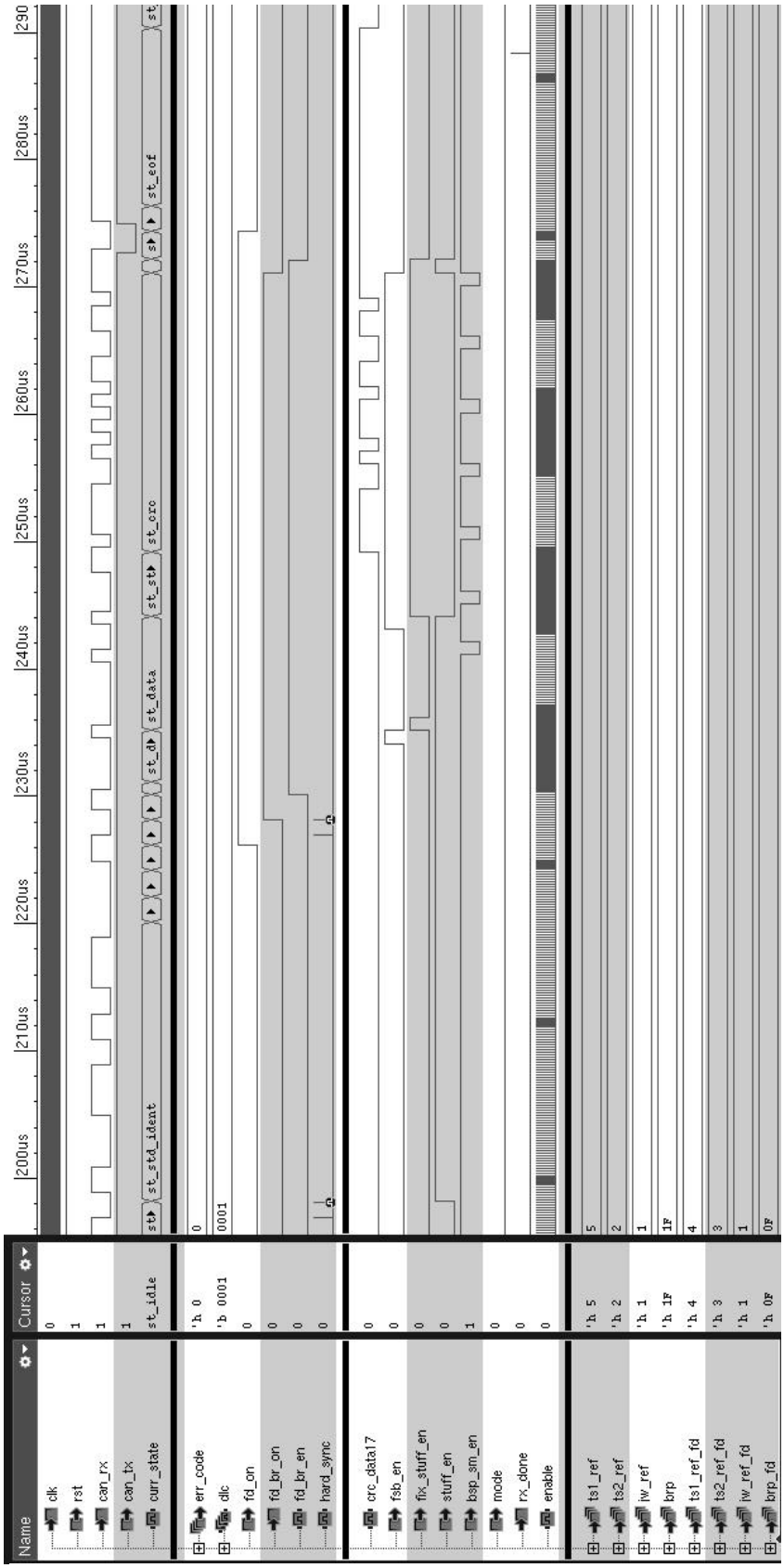
Na obr. 40 je vidět výsledek simulace, tedy rámec zprávy dle standardu klasického CAN 2.0, konkrétně na řádku, který odpovídá signálu s názvem *can_rx*. Řádek se signálem *curr_state* pak zobrazuje aktuální nastavený stav stavového automatu bloku BSP. V tomto případě jde o přijímanou zprávu se standardním identifikátorem (11bitů) a 1 Bajtem dat. Lze si



obr. 40 Simulace přijmu CAN 2.0 zprávy (STD ID, 1 data Bajt)

také všimnout signálu *stuff_en*, povolující mechanismus stuff bitů, který je dle specifikace CAN 2.0 správně nastaven do logické jedničky i v oblasti CRC pole, tedy povoluje zmíněný mechanismus i ve stavu *st_crc*. Kurzor je vložen v pozici bitu FDF, kde se v tento okamžik vyskytuje na sběrnici logická nula, jelikož se nejedná o zprávu v FD formátu, ale zprávu dle CAN 2.0

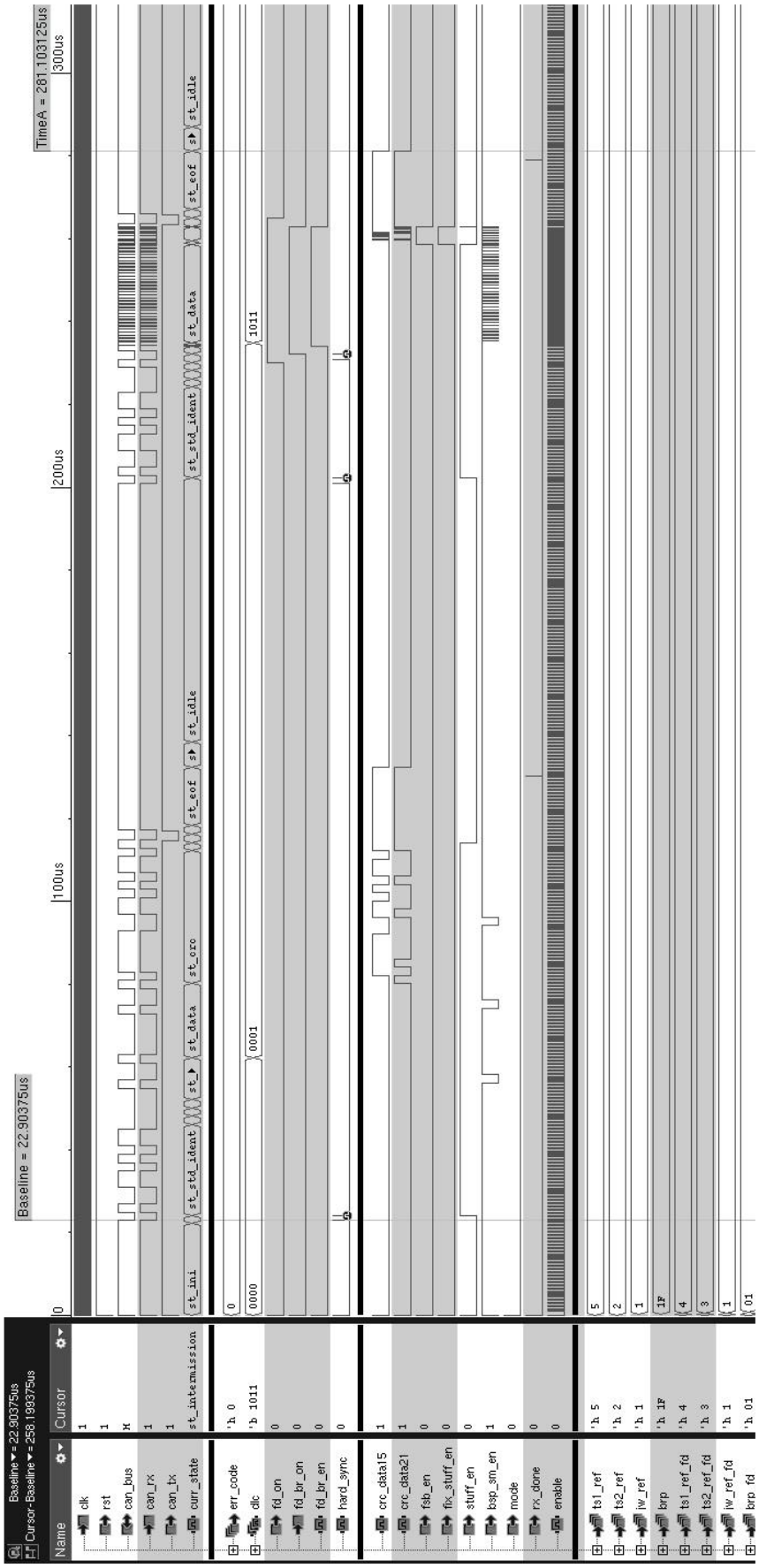
Na obr. 41 je vidět simulační výsledek 2. testu, konkrétně rámec zprávy dle nové specifikace CAN FD. Jde o přijímanou zprávu (RX) se standardním (11bitů) identifikátorem a 1 Bajtem dat. Uspořádání signálů na obrázku je obdobné jako v předchozím případě. Lze si všimnout například signálu *fd_on* nastavujícího se do log 1, v okamžiku, kdy se stavový automat nachází ve stavu *st_fdf* a signál *can_rx* v tu chvíli vykazuje logickou úroveň log. 1. Rovněž je vidět, že dochází k druhé „hard synchronizaci“ v oblasti spádové hrany při přechodu z bitu FDF na bit res. Toto je vidět na signálu *hard_sync*. Dále si lze všimnout, že vypočtena crc sekvence (crc_data17) neodpovídá přesně oblasti zprávy crc pole. Je to z důvodu, že jsou do



obr. 41 Simulace přijmu CAN FD zprávy (STD ID, 1 data Bajt)

sekvence doplněny FSB bity, jejichž umístění je vidět na signálu *bsp_sm_en*. Ten, když se nachází v úrovni logické nuly, jde o doplňkový bit (zobrazuje fixní i standardní stuff bity). Tento signál, jak název napovídá, povoluje chod stavového automatu bloku BSP. To znamená, že v logické nule přikazuje bloku BSP, aby aktuální bit nezpracovával, jelikož se jedná o stuff bit či fixní stuff bit a nikoli o platný bit pole crc či jiného pole.

Na obr. 42 je vidět porovnání standardní zprávy s 1 datovým Bajtem se zprávou standardu FD s 20 datovými Byty. Pro srovnání délek rámců zprávy byly použity kurzory pro lepší odečet začátku a konce. Je vidět, že při konfiguraci bitové rychlosti datové části FD rámce na 8 Mbit/s a zbytku zprávy a klasické zprávy CAN 2.0 na 500kbit/s, je tato zpráva stejně dlouhá nebo dokonce kratší než standardní zpráva. Přičemž druhá zpráva (FD) přenesla o 19 Bajtů více dat. Toto bylo hlavním cílem a důvodem vytvoření CAN FD standardu, jelikož pro dnešní účely již datový tok standardu CAN 2.0 často přestává být dostatečný.



obr. 42 Porovnání CAN 2.0 a CAN FD zprávy

2.3 Validace

Dalším krokem byla kontrola kódu tzv. „code review“ za účelem maximálního zjednodušení a snadné čtivosti s rychlým pochopením člověka neznalého problematiky. Následně pak připadá v úvahu syntéza.

Po zkompletování designu s mikroprocesorovým jádrem („proprietární“), které je v našem případě nejpodobnější známému jádru ARM, je návrh syntetizován a následně vložen do desky s FPGA.

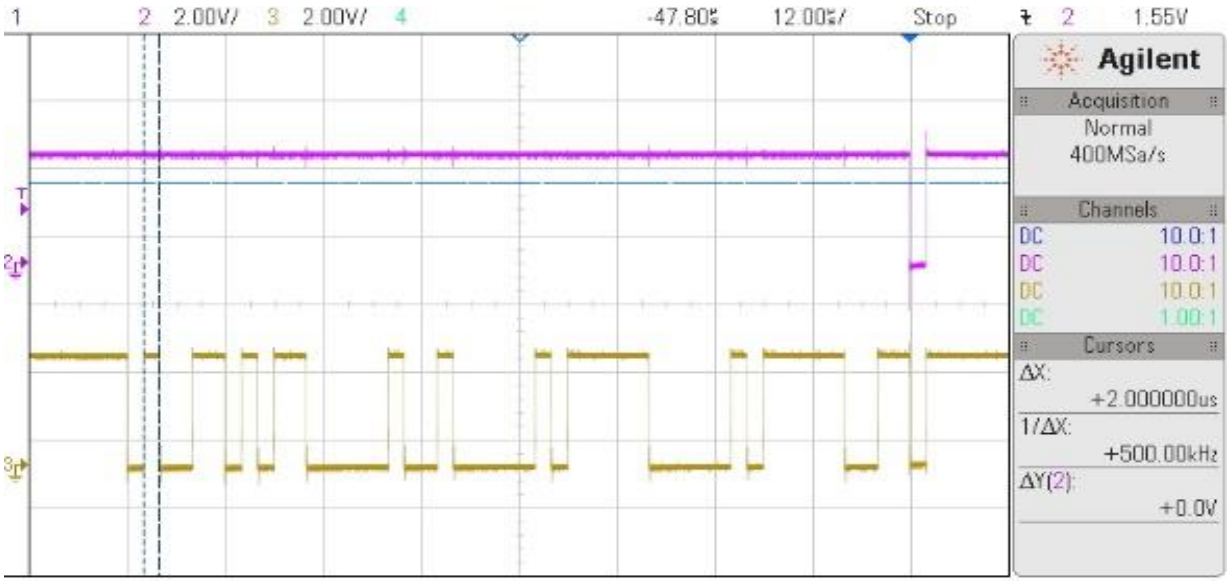
Následujícím krokem je seznámení se s prostředím pro programování, resp. vytváření programu pro mikrokontrolér. Použité prostředí je založeno na ECLIPSE. Umožňuje používat knihovny, hlavičkové soubory tzv. “header file“ a mnohé další. Mezi nezbytnou schopnost patří také DEBUGGER. Je tedy umožněno vytvořený kód po „build“ operaci nahrát do mikrokontroléru a následně po spuštění například krokovat jednotlivé řádky kódu s okamžitou kontrolou hodnot v paměti, odpovídajících registrům dříve zmíněné registrové mapy, například zpočátku programu, kdy probíhá nastavení po resetu, lze provést kontrolu kontrolního registru 0 a 1.

Jakmile je programem ověřena funkce a odladěno debuggerem, je dalším krokem ověření měřením a následně ověření průmyslovým analyzátorem - testerem.

2.3.1 Ověření měřením

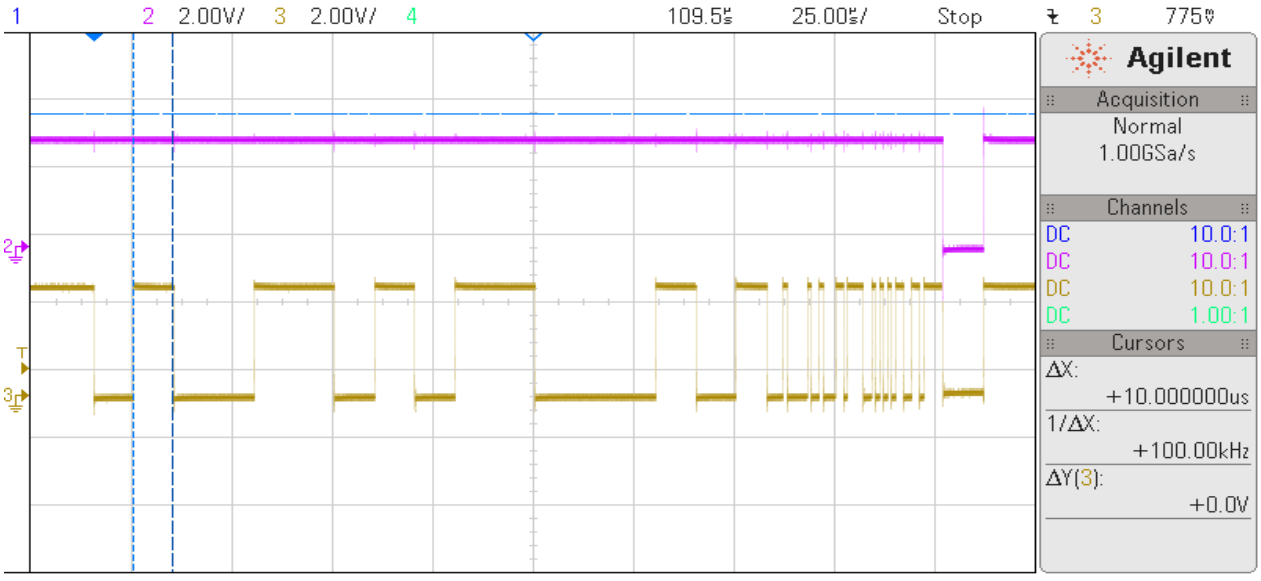
Za účelem zjištění reálného chování řadiče s mikroprocesorem je provedeno měření. K tomuto měření byl použit Osciloskop Agilent. Po nastavení trigeru na hranu signálu, nastavení přibližného časového okna na 100us a amplitudy/napětí na dílek cca 1V, je zpuštěna funkce „single“. Tato funkce po zachycení trigerovací podmínky uloží hodnoty zvoleného vstupu o nastaveném množství, tedy o určité časové délce a výsledek zobrazí na obrazovce. Tímto způsobem je možné zachytit a zobrazit rámec vysílané zprávy, který lze následně přiblížit, analyzovat, dekódovat a porovnat se simulací, zda je vše v pořádku.

Měření vyslané zprávy (CAN 2.0, standardní identifikátor, 1datový Bajt) na sběrnici zmíněným způsobem je vidět na obr. 43. V dolní části obrázku je průběh vyslané zprávy. Současně je měřen výstup druhého uzlu, který je vidět v horní časti obrázku. Je na něm pouze jediný impulz umístěný na konci zprávy, a to v oblasti ACK pole, kdy uzel potvrzuje dominantním bitem (ACK bit) úspěšné přijetí zprávy. S pomocí delta kurzorů v ose X je měřena šířka jednoho bitu ve zprávě. Výsledek měření je vidět v pravé části obrázku (2us). To odpovídá nastavení bitové rychlosti sběrnice na 500kbit/s, tedy velikosti TQ 0,2us, to při nastavení SS+TS1+TS2= 10 odpovídá 10 TQ na bit, tedy 2us na bit.



obr. 43 Zpráva CAN 2.0 měřena OSC na sběrnici (1 data Bajt)

Měření vyslané zprávy (CAN FD, standardní identifikátor, 1datový Bajt) na sběrnici zmíněným způsobem je vidět na obr. 44. V dolní části obrázku je vidět průběh zprávy typu CAN FD. Je dobře patrné, že první část zprávy obsahuje bity o mnohem větší šířce (délce), než je tomu v části druhé. Pro zajímavost jsou použity delta kurzory v ose X, jejichž rozdíl je vidět v pravé části obrázku (10us). Toto nastavení odpovídá bitové rychlosti 100kbit/s v oblasti arbitrace, a 1Mbit/s pak v oblasti datové fáze rámce. Na potvrzovacím bitu ACK je patrné, že je tato oblast již opět v módu 100kbit/s.



obr. 44 Zpráva CAN FD měřena OSC na sběrnici (1 data Bajt)

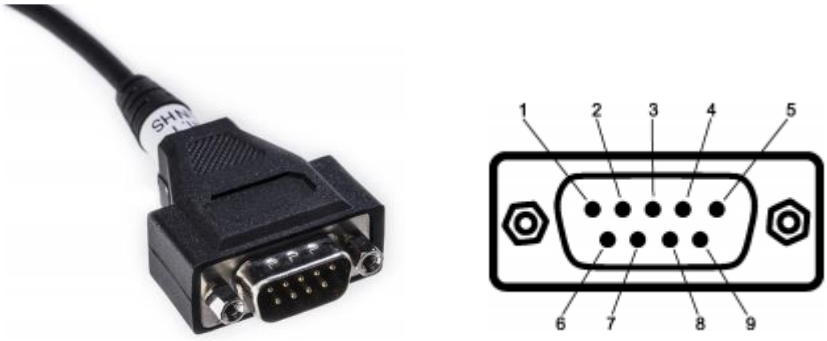
2.3.2 Ověření průmyslovým analyzátorem

Za účelem testování správného chování byl zadavatelem práce pořízen průmyslový analyzátor Kvaser Leaf Pro HS v2. Podoba zařízení viz obr. 45. Tento analyzátor umožňuje generování zpráv ve standardu CAN 2.0 s nastavitelnými parametry, jako jsou identifikátor (standardní (11b) nebo rozšířený (29b)) nebo bitrate, tedy bitovou rychlost sběrnice. Dále také umožňuje stanovit množství dat a mnohé další. Tento průmyslový analyzátor rovněž podporuje standard CAN FD, tedy umožňuje generovat, či přijímat zprávy s obdobným nastavením jako tomu je v předchozím případě.

obr. 45 Průmyslový analyzátor Kvaser Leaf Pro HS v2

Toto zařízení společnosti Kvaser, která je švédského původu, disponuje širokou škálou příkladů, předdefinovaných funkcí a procedur, a také rychlou a kvalitní podporou pro zákazníka.

Zmíněný tester disponuje 9 pinovým DSUB CAN konektorem. Zapojené piny jsou: 2 – CAN_L, 3 – GND, 7 - CAN_H a 5 – SHIELD. Ostatní piny jsou nezapojeny. Podoba, tvar konektoru a uspořádání pinu viz obr. 46.



obr. 46 Uspořádání DSUB CAN konektoru

K analyzátoru je volně stažitelný software (drivery ale i SDK). Jednoduchý SDK pojmenovaný TRX umožňuje v mnoha programovacích jazycích psát program pro specifické

chování. Například vysílání CAN FD zprávy každou vteřinu či reakci na přijatou zprávu s konkrétním identifikátorem například odesláním jiné zprávy. Zároveň lze doplnit řádky informující uživatele o aktuálním stavu uzlu v řádkovém výpisu, tedy například zda přijal zprávu a jaké jsou její parametry (ID, DATA apod.).

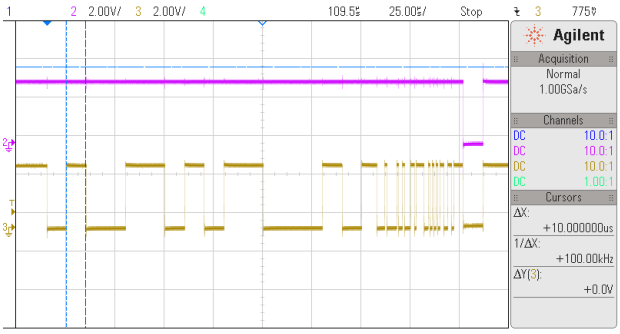
V tomto prostředí bylo sestaveno několik základních programů. Konkrétně jsou to tyto:

1. Odesílání zprávy dle klasického CAN 2.0 standardu se standardním identifikátorem a jedním Bajtem dat.
2. Odesílání zprávy dle klasického CAN 2.0 standardu s rozšířeným identifikátorem a jedním Bajtem dat.
3. Odesílání zprávy CAN FD standardu se standardním identifikátorem a 12 Bajty dat.
4. Odesílání zprávy CAN FD standardu s rozšířeným identifikátorem a 12 Bajty dat.
5. Přijmutí jakékoliv zprávy dle klasického CAN 2.0 standardu.
6. Přijmutí jakékoliv zprávy dle CAN FD standardu.

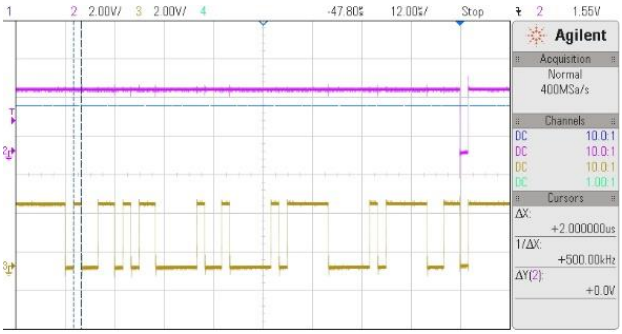
Zprávy generované těmito programy byly po připojení ke sběrnici měřeny osciloskopem pro ověření správného nastavení vlastností těchto zpráv a správné funkce analyzátoru. Měřený výstup, tedy zprávy z průmyslového analyzátoru při shodném nastavení jako u řadiče v FPGA se kompletně shodují. Viz kapitola 2.3.3.

2.3.3 Shrnutí Validace

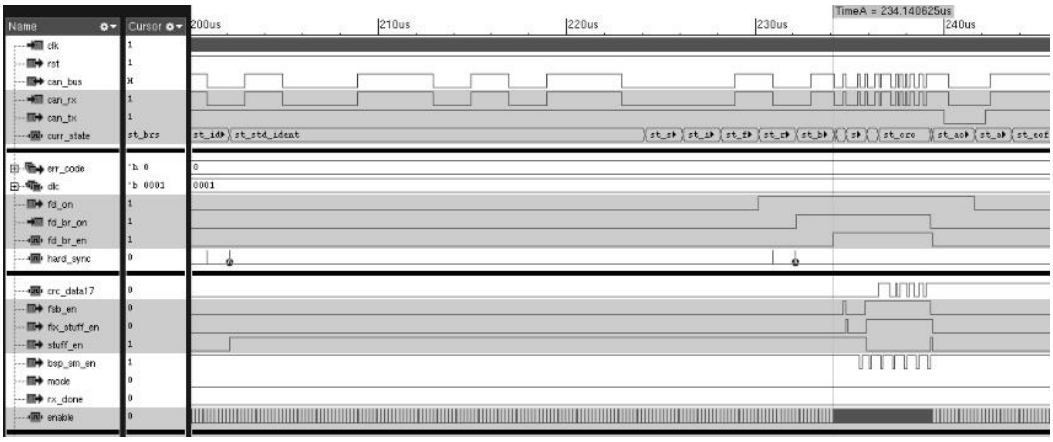
Výsledky validace je možné stanovit v okamžiku porovnání výsledků simulací a výsledků měření. V našem případě by byla Validace umožněna i bez srovnání se simulacemi, jelikož je použit průmyslový analyzátor. Tedy v případě, že odeslané zprávy navrženým řadičem na sběrnici jsou úspěšně přijaty analyzátorem a přijatá data a parametry této zprávy odpovídají vyslané zprávě, je ověřena správná funkce vysílacího módu řadiče. Obdobně pro mód přijímání jsou generovány různé případy rámce zprávy na analyzátoru a je pozorováno přijmutí na straně navrženého řadiče. V případě, že se opět shodují parametry a obsah zprávy přijaté a vyslané, pak je tímto způsobem potvrzena správná funkce řadiče. Úspěch je také vidět již v případě vyslání ACK bitu bez jakékoli chybové zprávy.



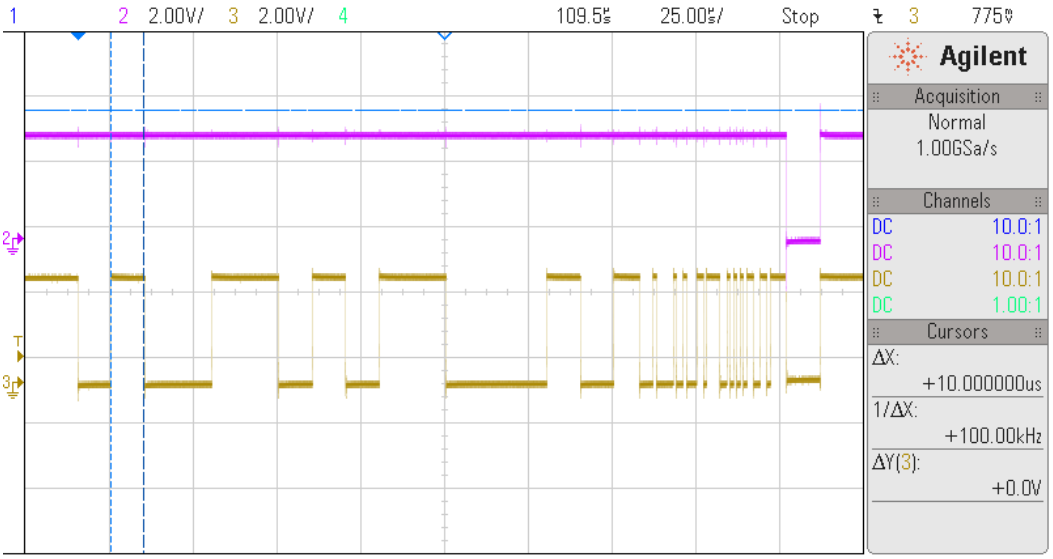
obr. 47 Výstup průmyslového analyzátoru, zpráva CAN FD (1 data Bajt)



obr. 48 Výstup průmyslového analyzátoru, zpráva CAN 2.0 (1 data Bajt)



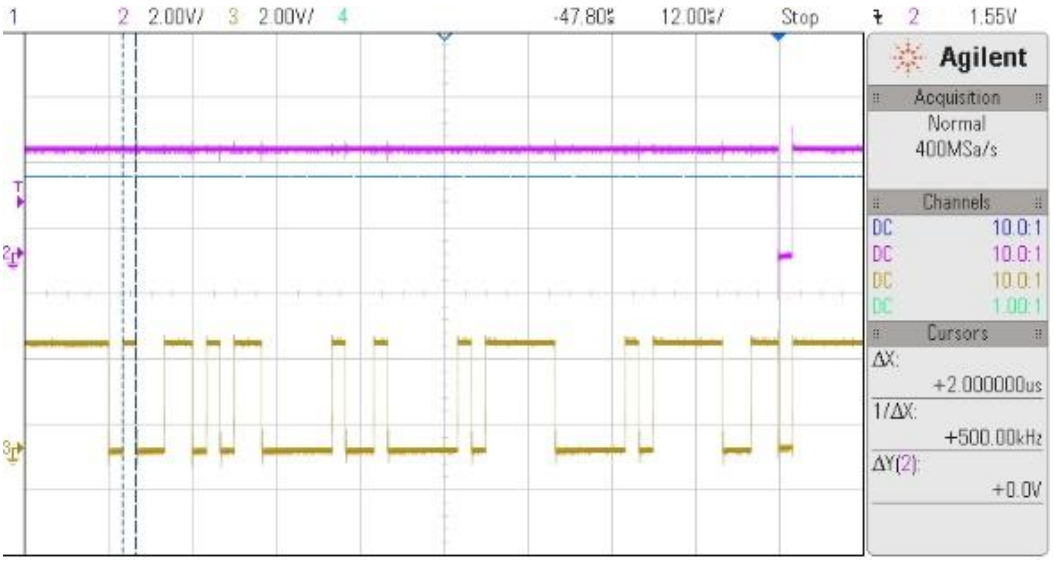
obr. 49 Simulace zprávy CAN FD standardu, 1 data Bajt STD ID



obr. 50 Měřená zpráva CAN FD standardu, 1 data Bajt a STD ID

Na této stránce je srovnání CAN FD zprávy, která byla simulována (obr. 49) a CAN FD zprávy, která byla řadičem vyslána a změřena (obr. 50). Při hlubším prostudování je vidět, že se zprávy shodují ve všech bitech rámce zprávy. Jde tedy o úspěšné potvrzení správné funkce navrženého řadiče

obr. 51 Simulace zprávy CAN 2.0 standardu, 1 data Bajt a STD ID



obr. 52 Měření zprávy CAN 2.0 standardu, 1 data Bajt a STD ID

Obdobný výsledek jako je na předchozí stránce je vidět na obr. 51 a obr. 52. Jedná se v obou případech o stejnou zprávu sestavenou dle standardu CAN 2.0. Použit je standardní identifikátor hodnoty 1238hex a náhodně zvolené hodnoty datového bajtu 3hex. Při hlubším studiu obou obrázku je závěrem shoda simulačního případu (obr. 51) a měřeného případu (obr. 52) zprávy.

Jedná se tedy o úspěšný návrh, jelikož se Validace návrhu řadiče shoduje s výsledky simulací. Druhotným ověřením funkce je správná, bezproblémová komunikace na sběrnici s průmyslovým analyzátorem, což je důkazem správného chování řadiče.

3 Závěr

Sběrnice CAN je v praxi hojně využívána z mnoha důvodů. Jedním z hlavních důvodů je její odolnost v rušivém prostředí. Uplatnění našla hlavně v průmyslu, ať už v automobilovém, leteckém či lodním.

Sběrnice (CAN 2.0) dnes dosahují v mnoha použitích horních limitních parametrů, například z pohledu objemu přenesených dat v čase. Z tohoto důvodu byl vyvinut standard CAN FD. Aktuálním tématem se tedy stalo zdokonalení existujícího řadiče na takovou úroveň, kdy bude schopen provozu v obou variantách a bude tak umožněn vyšší objem přenesených dat na sběrnici mezi zařízeními a rovněž bude dodržena kompatibilita s novými i starými zařízeními. Umožní to například tvorbu rychlejších systémů, ve kterých řídící členy dostanou odezvu na událost rychleji a s možností mnohem většího počtu podrobností a informací, které může nový CAN FD obsáhnout v jedné zprávě.

Cílem této práce bylo navržení a realizace rozšíření existujícího řadiče CAN o standard FD, který již byl psán v popisovacím jazyce VHDL. Tento jazyk má být použit i pro realizaci zmíněného rozšíření. Zároveň byla cílem úprava interního komunikačního rozhraní mezi řadičem a mikrokontrolérem z rozhraní pro mikroprocesory x51 na protokol AHB-Lite. Jeden z cílů práce je rovněž ověření funkce simulacemi, ale také ověření na desce FPGA.

Výsledkem této práce je funkční blok CAN2.0 / CAN FD řadiče. Jeho správná funkce byla úspěšně ověřena simulací, kde byla testována všechna pravidla a správná funkce rozšířených, ale i původních mechanismů. Mezi nově implementované mechanismy tvořící rozšíření dodaného řadiče patří: mechanismus TDC – Transmitter Delay Compensation, vzorkování pomocí SSP, Stuffcount uvnitř rozšířeného pole CRC Field a v neposlední řadě rozšířené pole Data Field. Simulacemi bylo ověřeno maximum pravidel a správné chování řadiče. Jde například o vysílání a příjem zpráv na všech, protokolem stanovených rychlostech až do 8Mbit/s pro CAN FD. Komunikace a samotné nastavení řadiče probíhá přes interní AHB-Lite sběrnici, přes kterou je řadič spojen s hostitelským mikrokontrolérem.

Vše, co bylo použitým hardwarem umožněno, bylo také ověřeno v rámci Validace. Ta probíhala na FPGA desce v laboratoři. Nejprve byla vytvořena sada testů pro ověření správného chování řadiče měřením, následně pak byl použit průmyslový CAN analyzátor firmy KVASER. S jeho pomocí bylo ověřeno, že řadič komunikuje skrze sběrnici s uzlem druhým. Jinak řečeno si stanice takzvaně rozumí v obou standardech, tedy CAN 2.0 i CAN FD. Vysílání a příjem zpráv dle CAN 2.0 bylo ověřeno v rozmezí 100kbit/s a maximální bitové rychlosti 1Mbit/s. V případě rozšíření byla provedena validace až do rychlosti 2Mbit/s, což je již rozsah rychlosti pouze pro CAN FD a zároveň laboratorní maximum použitého hardwaru. Validace tedy potvrdila výsledky simulací.

Práce na tomto projektu byla nejprve postavena na kompletním a hlubokém porozumění CAN 2.0 specifikace firmy BOSCH. Vzápětí byl věnován čas pro porozumění všem blokům a jejich funkcím či procedurám existujícího řadiče. Zmíněný řadič byl vytvořen v popisovacím jazyku VHDL („Very High Speed Integrated Circuits Hardware Description Language“). Vytvořením podpůrných funkcí a procesů, obecně tedy „Test bench“ a simulováním funkce existujícího řadiče byla ověřena znalost standardu CAN 2.0 a utužena zručnost s prostředím, simulátorem a dalšími nástroji. Dalším nezbytným krokem bylo dokonalé porozumění normě popisující CAN FD. V neposlední řadě byl věnován prostor principu sběrnice

AHB-Lite. Všechny tyto kroky byly nezbytnou přípravou před samotnou tvorbou rozšíření řadiče, které byly z hlediska zkušeností pro mne velkým přínosem.

V druhé části, tedy při samotné realizaci rozšíření jednotlivých bloků řadiče o funkce, procedury a pravidla v jazyku VHDL, byla nutná implementace rozšíření, jako jsou pole Data Field z 8 na 64 Bajtů, rozšíření pole CRC Field o Stuffcount pole a také o další dvě možnosti výpočtu CRC sekvence, která závisí na množství dat (CRC17 a CRC21). Dále také bylo potřeba implementovat mechanismus TDC („Transmitter Delay Compensation“) používaný pro kompenzaci nezanedbatelného zpoždění vysílače. Tento mechanismus zahrnuje také vytvoření druhého bodu vzorkování tzv. SSP („Secondary Sample Point“) s veškerými pravidly upravujícími kontroly chyb atd. Jedním z rozšíření byla také úprava původní interní komunikace (sběrnice) mezi řadičem a mikrokontrolérem ze sběrnice pro procesory typu x51 na standard AHB-Lite používaný dnes například v mikroprocesorech s jádrem ARM.

V průběhu celé práce bylo kromě jiného zdokonaleno mé chápání paralelního prostředí či přijmutí mnoha užitečný a důležitých návyků a pravidel pro digitální návrh integrovaných obvodů.

Do budoucna by bylo určitě vhodné pořídit průmyslový analyzátor schopný realizace CAN zpráv všech druhů, se všemi různými variantami nastavení. Toto za účelem plného otestování navrženého řadiče. Ten již bude finančně mnohem hůře dostupný. Dalším možným krokem by byla také realizace testovacího mechanismu pro ověření TDC, tedy kompenzace zpoždění vysílače. Možným krokem při testování chyb by bylo například použití dlouhého vedení a připojení většího počtu stanic/uzlů k této sběrnici.

4 Reference

[1] Florian Hartwich, Robert Bosch GmbH, “CAN with Flexible Data-Rate,” 2012.

[2] Robert Bosch GmbH, “CAN Specification 2.0,” 1991.

[3] Internetové stránky: http://www.can-cia.org, CAN in Automation, 2006.

[4] Wilfried Voss, ”A Comprehensible Guide to Controller Area Network”, 2008.

[5] ISO 11898-1:2015(E), “Road vehicles – Controller area network(CAN)”, 2015

[6] ARM Limited, “AMBA 3 AHB-Lite Protocol Specification”, 2006

[7] Internetové stránky: http://www.electronicdesign.com

[8] Sedlaček R. - přednášky a cvičení Aplikace programovatelných hradlových polí, ČVUT FEL Praha, 2016
http://measure.feld.cvut.cz/vyuka/predmety/A0B38APH

[9] Hazdra P. – přednášky a cvičení Návrh systémů VLSI, ČVUT FEL Praha, 2016
https://moodle.fel.cvut.cz/courses/A0M34NSV

Seznam použitých zkratek

ACK	„Acknowledge bit“ : Potvrzovací bit
AHB	“Advanced High performance Bus”: Sběrnicový protokol
AHB – Lite	“Advanced High performance Bus -Lite”: Sběrnicový protokol
AMBA	“Advanced Microcontroller Bus Architecture”: Sběrnicový protokol
ARM	“Acorn RISC Machine”: typ mikroprocesorového jádra
ASIC	“Application Specific Integrated Circuit”: Zákaznický obvod
AXI	“Advanced eXtensible Interface”: Sběrnicový protokol
BRS	“Bit Rate Switch”: Přepnutí bitové rychlosti
BSL	“Bit Stuffing Logic”: Blok jádra řadiče
BSP	“Bit Stream Processor”: Blok jádra řadiče
BTL	“Bit Timing Logic”: Blok jádra řadiče
CAN	“Controller Area Network”: Sběrnicový protokol
CAN_H	“CAN High”: jeden vodič sběrnice CAN
CAN_L	“CAN Low”: jeden vodič sběrnice CAN
CAN FD	“Controller Area Network Flexible Data-Rate”: Sběrnicový protokol
CPLD	“Complex Programmable Logic Device”: Typ programovatelného obvodu
CRC	“Cyclic Redundancy Check”: kontrolní metoda
ESI	“Error Status Indicator”: indicator chybového stavu vysilače
EOF	“End Of Frame”: ukončovací blok CAN zprávy
DLC	“Data Length Code”: informace o množství dat
FDF	“FD format indicator”: indikace FD formátu
FIFO	“First In First Out”: Paměťová struktura
FPGA	“Field Programmable Gate Array”: Typ programovatelného obvodu
FSB	“Fixed Stuff Bit”: fixní doplňkový bit

IEEE	“Institute of Electrical and Electronics Engineers”: Organizace
ISO	“International Organization for Standardization” : Organizace
IPT	“Information Processing Time”: doba zpracování informace
RTL	“Register - Transfer Level”
REC	“Receive Error counter”: čítač chyb přijímače
SOF	“Start of Frame”: počáteční bit zprávy CAN
TDC	“Transmitter Delay Compensation”: kompenzace zpožděni vysilače
TEC	“Transmitter Error Counter”: čítač chyb vysílače
TS1	“Timing Segment 1”: Pole TQ tvořící tvar bitu
TS2	“Timing Segment 2”: Pole TQ tvořící tvar bitu
SJW	“Synchronization Jump Width”
SS	“Synchronization Segment”: Pole TQ tvořící tvar bitu
SSP	“Secondary Sample Point”: Druhotný vzorkovací okamžik
SP	“Sample Point”: vzorkovací okamžik
TQ	“Time Quantum”: časový úsek
VHSIC	“Very High Speed Integrated Circuit”: druh Integrovaného Obvodu
VHDL	“VHSIC Hardware Description Language”: jazyk pro popis HW
IC	“Integrated Circuit”: Integrovaný obvod

Seznam obrázků