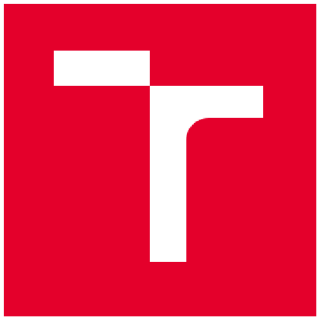


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY

DEPARTMENT OF MICROELECTRONICS

PROPRIETÁRNÍ KOMUNIKAČNÍ PROTOKOL PRO PŘENOS DAT MEZI FPGA A PC

PROPRIETARY COMMUNICATION PROTOCOL FOR DATA TRANSFER BETWEEN FPGA AND PC

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — David Beneš

AUTHOR

VEDOUCÍ PRÁCE — Ing. Vojtěch Dvořák

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Mikroelektronika a technologie**

Ústav mikroelektroniky

Student: David Beneš ***ID:*** 220860

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Proprietární komunikační protokol pro přenos dat mezi FPGA a PC

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s možnými způsoby přenosu dat mezi obvodem FPGA a PC pomocí sběrnice USB. Navrhněte komunikační protokol, který bude umožňovat posílat příkazy z PC do FPGA. Protokol bude podporovat základní operace jako zápis, čtení a zápis s kontrolou, a to jak pro jeden registr, tak pro celý paměťový blok. Dále musí podporovat autonomní odesílání telemetrických dat z obvodu FPGA a jejich příjem na straně PC. Proveďte implementaci modulu přijímače a vysílače paketů pro obvod FPGA v jazyce VHDL a ovládací funkce na straně PC v jazyce Python. Otestujte funkčnost přenosu dat na zvoleném cílovém obvodu a určete limit pro maximalní frekvenci přenosu telemetrie.

DOPORUČENÁ LITERATURA:

Dle pokynů vedoucího práce

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 2.6.2022

Vedoucí práce: Ing. Vojtěch Dvořák

doc. Ing. Jiří Háze, Ph.D.
předseda rady studijního programu

Abstrakt

Tato práce se zabývá návrhem a implementací komunikačního protokolu, který umožňuje přenášet data mezi PC a FPGA. Protokol podporuje funkce jako jsou zápis, čtení a zápis se čtením do paměti. Další funkcí je autonomní přenos dat získaných z telemetrie. V teoretické části této práce je popsán komunikační kanál, který je využit pro přenos paketů. V praktické části jsou nadefinovány jednotlivé pakety a protokol je zpracován v podobě knihovny na straně PC a v podobě modulu na straně FPGA.

Klíčová slova

FPGA, USB 2.0, komunikace s FPGA, komunikační protokol, Python

Abstract

This thesis deals with the design and implementation of a communication protocol which allows for data transfer between a PC and an FPGA. The designed protocol supports functions such as ´write´, ´read´ and ´write with a confirmation´ with a memory. Another supported function is autonomous transfer of telemetry data from an FPGA to a PC. Described in the theoretical part of the thesis, is the communication channel that is used for packet transfer. In the practical part are defined the packets and the protocol is implemented on PC in the form of a library and as a module on FPGA.

Keywords

FPGA, USB 2.0, communication with FPGA, communication protocol, Python

Bibliografická citace

BENEŠ, David. Proprietární komunikační protokol pro přenos dat mezi FPGA a PC [online]. Brno, 2022 [cit. 2022-05-07]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/142772. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky. Vedoucí práce Vojtěch Dvořák.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	David Beneš
VUT ID studenta:	220860
Typ práce:	Bakalářská práce
Akademický rok:	2021/22
Téma závěrečné práce:	Proprietární komunikační protokol pro přenos dat mezi FPGA a PC

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 2. června 2022 ----------

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Vojtěchu Dvořákovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: 2. června 2022 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

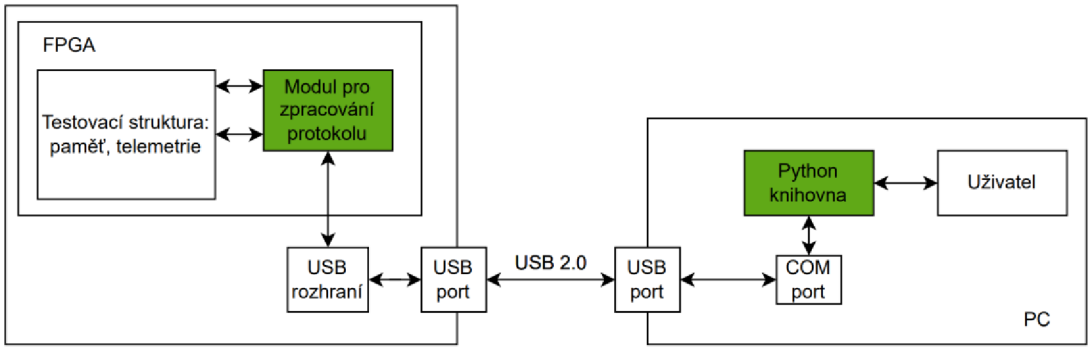
Tato bakalářská práce se zabývá možností propojení FPGA s počítačem a následné komunikaci mezi těmito objekty. Cílem této práce je navrhnout komunikační protokol, který bude umožňovat přenášet data mezi PC a FPGA pomocí rozhraní USB 2.0. Tento komunikační protokol by měl být schopen podporovat přístup a práci s pamětí v obvodu FPGA, a to ve smyslu zápisu a čtení jednotlivých slov. Dále by mělo být možné pomocí navrženého modulu v FPGA autonomně zpracovávat příchozí telemetrická data, která se následně odešlou do PC.

V rámci bakalářské práce bude nejprve představena struktura komunikační linky a následně popsány prvky nacházející se na komunikační trase. Konkrétně bude popsáno rozhraní USB na straně počítače, USB sběrnice včetně jejího protokolu a komunikační rozhraní na straně FPGA. V praktické části této práce bude na straně PC implementován protokol v podobě knihovny vytvořené v programovacím jazyce Python. Tato knihovna bude podporovat jednotlivé funkce pro práci s pamětí a bude umožňovat zachytit příchozí telemetrické pakety z testovaného zařízení. Na straně FPGA bude zpracování tohoto protokolu popsáno v jazyce VHDL. Klíčové pro testování aplikace na straně FPGA bude maximální možná frekvence telemetrie a parametry paketů, při kterých navržený modul zpracuje jednotlivá data správně.

Využití tohoto protokolu lze nalézt především pro testování aplikací, kde bude potřeba přistupovat k paměti a zároveň zpracovávat periodicky generující se data. To může být například měření EKG, teploty nebo určování polohy mechanického zařízení. Teoreticky lze i za pomoci paměti upravovat parametry měření, a tím rychleji kalibrovat daný měřící přístroj. Hlavní výhodou je především možnost pracovat s testovaným objektem v reálném čase a autonomní přenos dat měřícího zařízení.

1 STRUKTURA KOMUNIKAČNÍ LINKY

Jako u každého spojení dvou zařízení a výměny informací mezi nimi je potřeba definovat, jakým způsobem se budou data přenášet z bodu A do bodu B. V tomto případě se bude jednat o přenos dat mezi PC a FPGA. Pro přenos informací bude sloužit sběrnice USB 2.0. Propojení FPGA a PC pomocí USB ale není možné napřímo. Je to dáno především tím, že se jedná o asynchronní přenos. Napětí, na kterých pracuje USB 2.0 a FPGA navíc nejsou shodná [1]. Je tedy potřeba najít vhodný způsob, jak tyto dvě odlišná zařízení připojit. Na straně PC si vystačíme s jednoduchým USB portem, který lze ovládat pomocí příslušných ovladačů. Na straně FPGA je však třeba využít některé z komerčně dostupných rozhraní pro ovládání USB portu. Popisovanou strukturu komunikační linky je možné vidět na obrázku 1.1. V rámci bakalářské práce budou vyřešeny zvýrazněné prvky komunikační struktury. V případě PC se jedná o knihovnu pro zpracování komunikačního protokolu v jazyce Python, na straně FPGA to bude modul sepsaný v jazyce VHDL.



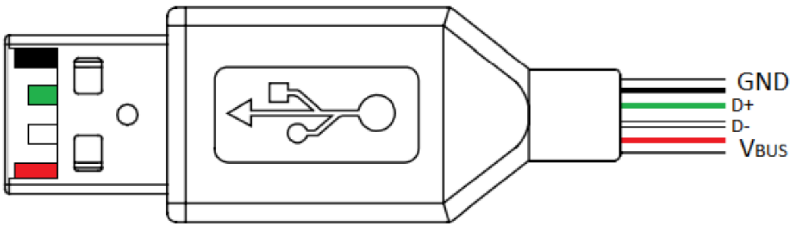
Obrázek 1.1 Struktura komunikační linky

1.1 Sběrnice USB 2.0

USB neboli „Universal Serial Bus“ je výsledkem myšlenky, která se snažila o jednoduché spojení počítače a periferií. Hlavní výhodou je okamžité napojení na dané zařízení bez nutnosti restartování PC (plug-and-play) [2]. Další důležitou vlastností, která bude využita v této aplikaci je i možnost přenášet data různou rychlostí díky vyrovnávacím pamětím, které se nacházejí na jednotlivých rozhraní, především pak vyrovnávací paměť na rozhraní FPGA-USB.

1.1.1 Mechanické a elektrické vlastnosti USB 2.0

USB kabel se skládá ze čtyř vodičů, dva signálové a dva napájecí. Napájecí vodiče jsou označovány jako V_{BUS} o potenciálu +5 V a GND o referenčním napětí. Vodiče jsou barevně odlišeny, a to tak, jak je vidět na obrázku 1.2, kde V_{BUS} je červený a zemnící vodič je černý. Tyto vodiče poskytují proud až 500 mA pro připojené periferie. [2]



Obrázek 1.2 Barevné rozložení vodičů v USB kabelu [2]

Dále se v USB kabelu nachází signálové vodiče. Jsou označovány jako D+ a D-, kde D+ je standardně zelený a D- bílý [2]. Tyto datové vodiče jsou navzájem inverzní kvůli odečtení rušivých signálů z vnějšího okolí. Vodiče jsou také zkrouceny, a to především kvůli vysokým přenosovým rychlostem, kdy se vodiče chovají jako vysílače. Kroucení toto vysílání účinně potlačuje. [3]

U USB 2.0 se setkáváme se třemi druhy rychlostí. *High Speed* (480Mbit/s), *Full Speed* (12Mbit/s) a *Low Speed* (1,5Mbit/s). Tyto rychlosti jsou definovány podle připojení datových vodičů k 3,3 *V* přes 1,5 *k*Ω odpor. *High Speed* režim, který je připojen stejně jako *Full Speed*, je řešen pomocí softwaru k dosažení vyšších rychlostí.[4]

1.1.2 USB 2.0 protokol

Komunikace mezi PC a koncovým zařízením je řízena souborem pravidel, která se jako celek nazývají protokol. Tyto pravidla definují, jakým způsobem je řízena výměna informací a jakým způsobem jsou skládány pakety pro přenos dat.

Pakety pro přenos dat se podle USB protokolu skládají z několika částí, které jsou skládány podle funkcí jednotlivých paketů. Každý paket začíná Synchronizačním polem (*SYNC field*). Jedná se o sekvenci pulzů o maximální hustotě, která zajišťuje synchronizaci hodin pro přesné čtení přicházejícího paketu.[2]

Dalším prvkem každého paketu je také jeho identifikace (PID). Tato část upozorňuje zařízení (jak PC, tak periferii), jaký formát bude paket mít a jakým způsobem má naložit s následujícími bajty. PID je kódováno pomocí osmi bitů. Čtyř identifikačních a čtyř ověřovacích – ty jsou inverzní vůči identifikačním bitům.[2]

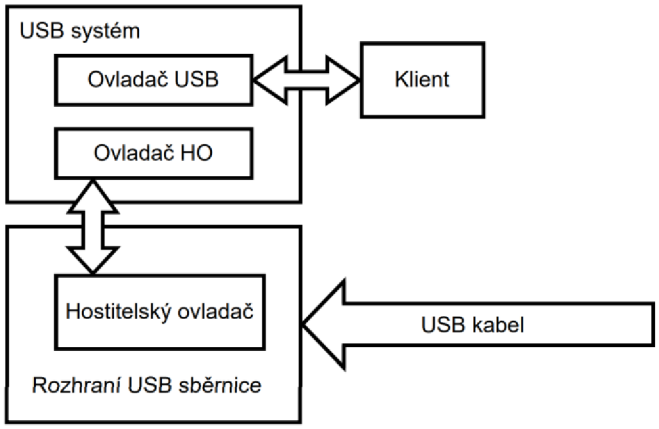
Pro přenos informací (dat) po USB lince se využívají čtyři druhy PID. DATA0, DATA1, DATA2 a MDATA. Toto dělení slouží pro jednotlivé režimy přenosu, které jsou již zmíněny v kapitole 0. Maximální velikost informací, které můžeme v datové části paketu přenášet je pro režim *High-speed* 1024 B [2]. Pakety jsou následně ukončovány pomocí algoritmu CRC, který ověřuje integritu zasílaných dat a EOP. Popisovaný paket můžeme vidět na obrázku 1.3 [5]. Jednotlivé bity jsou posílány na sériovou linku v řazení od nejméně významného bitu (LSb) až po nejvýznamnější bit (MSb), který je řazen jako poslední [2].

Synchronizační pole	PID	DATA	CRC	EOP
8 bitů(low/full)/32 bitů (high)	8 bitů	až 8 bajtů(low)/1023 bajtů (full)/ 1024 bajtů (high)	16 bitů	N/A

Obrázek 1.3 DATA paket, který odesílá data po sériové lince [5]

1.2 USB rozhraní – na straně PC

Na straně počítače se nachází dvě vrstvy, které překládají komunikaci mezi USB rozhraním a danou aplikací. Vrstva, která je na fyzické úrovni a nazývá se „Rozhraní USB sběrnice“ a vrstva, která je na úrovni operačního systému se nazývá „USB Systém“. Zjednodušený pohled na vrstvy je vyobrazen na obrázku 1.4, který vychází z [2].



Obrázek 1.4 Zjednodušený pohled na komunikaci s USB portem ze strany PC

1.2.1 Rozhraní USB sběrnice

Rozhraní USB sběrnice obsahuje několik fyzických prvků, které obsluhují jeden USB port. Jedním z důležitějších prvků je ´Hostitelský ovladač´. Tento prvek má několik úkolů:

- Řídí komunikaci na úrovní USB protokolu
- Balí data z vyšších vrstev do příslušných paketů
- Zpracovává vstupní a výstupní elektrické signály

Tato vrstva má také přístup do paměti operačního systému. Přenos dat mezi pamětí a USB sběrnicí je prováděn automaticky za pomoci „Hostitelského ovladače“ [2][6]

1.2.2 USB Systém

USB Systém obsahuje ve zjednodušeném pohledu dva důležité prvky. USB ovladač (USBD) a Ovladač HO (HCD), který má na starost ovládání výše zmíněného Hostitelského ovladače. Tato vrstva je napojena na více USB portů, se kterými může

komunikovat a zadávat jim instrukce.

HCD je ovladač, který popisuje ´Hostitelský ovladač´ na vyšší úrovni abstrakce a umožňuje tak ovladačům USBD univerzálnější komunikaci s USB portem. Je to dáno větším množstvím ´Hostitelských ovladačů´, kde by bylo neefektivní vytvářet pro každý vlastní USB ovladač. [2]

USBD je ovladač, který je součástí OS a umožňuje dané aplikaci pracovat s USB portem. Těchto ovladačů je několik. Například pro myš, klávesnici a nebo dedikovaný herní ovladač. Tyto ovladače mohou být upřesněny za pomocí softwaru, který poskytuje výrobce.

1.3 USB rozhraní – na cílovém zařízení

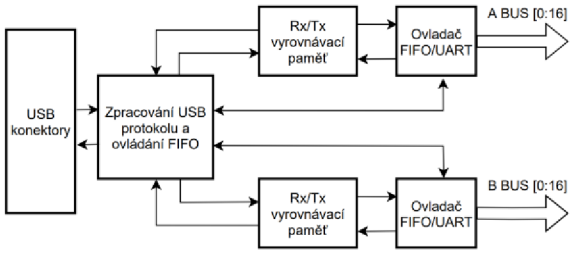
Pro připojení USB kabelu a zpracování signálů bude sloužit USB rozhraní. Tato zařízení zpravidla obsahují prvky, které se starají o řízení komunikace podle protokolu USB 2.0 a umožňují jednoduchým způsobem připojit externí zařízení k PC.

Všechna tato rozhraní mají společné rysy v podobě fyzických pinů pro USB kabel, zařízení pro překlad elektrických signálů na bity a zařízení pro zpracování USB protokolu. Pro předání bajtů do libovolného zařízení následně slouží buď paralelní rozhraní v podobě *First-In-First-Out bufferu* (FIFO) nebo *General-Purpose-Interface* (GPIF) a sériové rozhraní v podobě I^2C nebo USART.

Společností, které taková zařízení vyrábějí, je na trhu mnoho, ale mezi výraznější patří především společnosti „FTDI“ [7] a „Cypress Semiconductor“ [8]. K porovnání byly vybrány čipy FTDI FT2232H a Cypress FX2LP.

1.3.1 Čip FTDI FT2232H

Čip FT2232H je USB rozhraní, které je kompatibilní s USB 2.0 v režimu *High speed* a *Full Speed*. Obecně se skládá z prvků, které mají fyzické piny pro připojení USB kabelu. Dále se zde nachází i modul pro zpracování elektrických signálů a protokolu USB 2.0. Na obrázku 1.5 je ilustrována hlavní výhoda tohoto čipu, tedy dva kanály, které umožňují komunikovat se dvěma externími zařízení za pomocí jedné USB sběrnice. U tohoto čipu je ještě nutné zmínit, že jednotlivé kanály jsou omezeny na rychlost 40 MB/s a nevyužívají tak plnou šířku pásma USB 2.0 v režimu *High speed* [9].



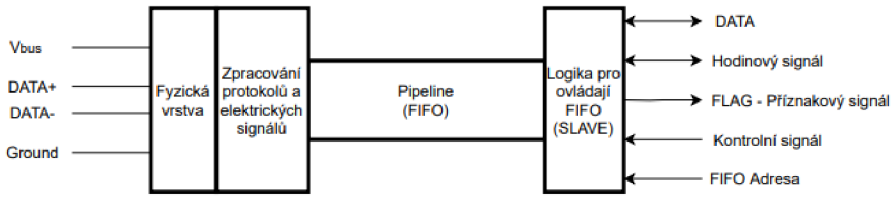
Obrázek 1.5 Obecný přehled prvků USB rozhraní FT2232H [9]

1.3.2 Čip Cypress FX2LP

Čip FX2LP na rozdíl od FT2232H umožňuje přenos data maximální rychlosti USB 2.0. Pro propojení komunikační cesty mezi USB a testovacím zařízením jsou zde využity prvky jako jsou *Serial interface engine*, dostatečně velké vyrovnávací paměti a mikrokontroler typu 8051. Tento čip také poskytuje paralelní a sériové rozhraní pro komunikaci PC a periferních zařízení.

Paralelní rozhraní jako jsou FIFO ve stavu *Slave* nebo GPIF, jsou schopna posílat data do periferního zařízení rychlostí, která odpovídá maximální rychlosti USB 2.0.

Rozhraní FIFO obsahuje sběrnici o 8 až 16 bitech pro přenos dat, která, může pracovat v synchronním i asynchronním režimu. Obsahuje také stavové signály, který informují periferní zařízení o stavu FIFO. Může buď indikovat plné nebo prázdné FIFO. [10]



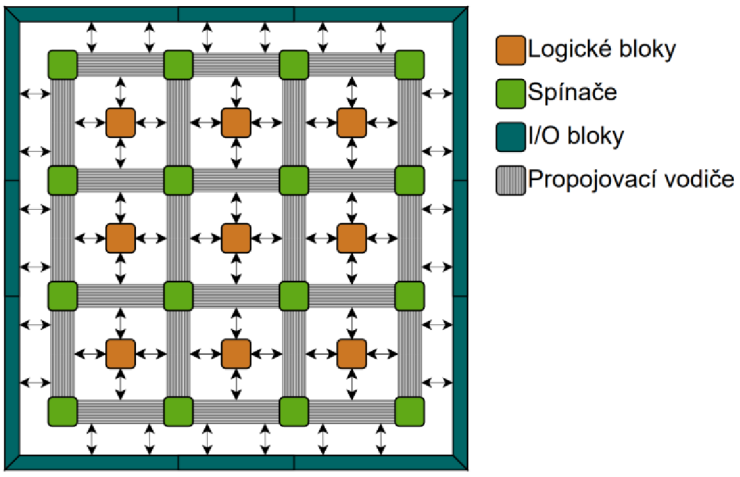
Obrázek 1.6 Cypress rozhraní [10]

2 OBVODY FPGA

FPGA neboli *Field Programmable Gate Array* je typ integrovaného obvodu. Jeho hlavní výhodou je možnost opakovatelného vytváření fyzických spojení mezi logickými bloky. Fyzická spojení se vytváří pomocí programovacích jazyků, a to nejčastěji za využití VHDL a jazyka Verilog (HDL – *Hardware Description Language*). Obvody FPGA představují kompromis mezi mikroprocesory a obvody ASIC.

Na rozdíl od procesoru, který pracuje sekvenčně, může FPGA provádět velké množství operací paralelně. To umožňuje efektivně zpracovávat například obrazové nebo digitální signály [12]. Obvody ASIC pak obvykle umožňují dosahovat ještě vyššího výpočetního výkonu v porovnání s obvody FPGA. Jejich vývoj je ale časově náročný a výroba malých sérií je velmi drahá. Další jejich nevýhodou je také nemožnost tento obvod přeprogramovat [11].

Obecně se FPGA architektura skládá ze čtyř prvků. Vstupně-výstupní bloky, spínače, propojovací síť a programovatelné logické bloky. Jednotlivé prvky jsou rozmístěny do dvourozměrné matice. Rozložení těchto prvků je ilustrováno na obrázku 2.1. Toto rozložení umožňuje návrháři nastavit propojení mezi jednotlivými logickými bloky.



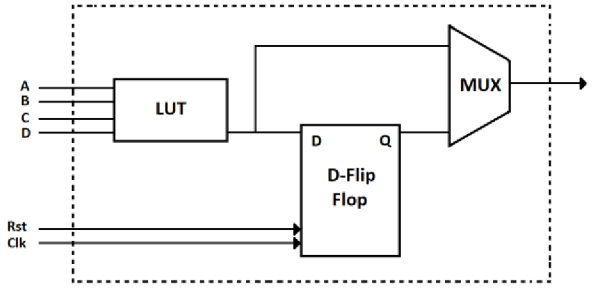
Obrázek 2.1 Architektura FPGA [12]

2.1 Programovatelné logické bloky

Logické bloky mají za úkol vytvářet logické, paměťové a aritmetické funkce v obvodech FPGA. Jednotlivé logické buňky obsahují tři základní prvky, náhledovou tabulku (LUT neboli *Look-up-Table*) a klopný obvod typu F (DFF) multiplexor. Ukázka struktury jedné logické buňky je na obrázku 2.2. níže [13].

Náhledová tabulka funguje na principu SRAM, kde je daná logická funkce uložena v paměti a vstupy se odkazují na danou paměťovou buňku. Výstup je následně vyveden

jako hodnota paměťové buňky. Tato metoda výrazně snižuje výpočetní čas, který by jinak byl potřeba na určení výsledné hodnoty. V závislosti na zvoleném obvodu FPGA jsou nejčastěji složeny ze čtyř nebo šesti-vstupových logických buněk. Klopný obvod typu D je binární registr, který ukládá logickou hodnotu po dobu jednoho hodinového cyklu. Tento prvek slouží k realizaci synchronní sekvenční části. Posledním doplňkovým prvkem v logické buňce je multiplexor [11][12].



Obrázek 2.2 Základní logická buňka FPGA

2.2 Propojovací sítě

Propojovací sítě slouží k propojení signálů mezi logickými bloky. Na tyto sítě jsou kladeny dva požadavky. Musí být flexibilní, tedy musí obsahovat spoustu programovatelných přepínačů, a signál přes tyto sítě musí mít co nejmenší zpoždění. Bohužel jdou tyto požadavky proti sobě, protože přepínače způsobují nezanedbatelné zpoždění. Z tohoto důvodu se používají dva typy sítí - sítě pro rozvod hodinového signálu a sítě pro logické signály.

Pro rozvod logických signálů se využívají propojovací vodiče. Existuje několik typů propojovacích vodičů, které se dělí podle toho, kolik logických bloků propojují. Jednoduché vodiče (*Single length*) propojují sousední logické bloky. Dvojité vodiče (*Double length*), které pojmou dva sloupce a dva řádky. A nakonec dlouhé vodiče (*Long length*), které obsáhnou celý sloupec nebo řádek logických bloků.

Sítě hodinového signálu jsou navrženy tak, aby zde nedocházelo k fázovým posuvům hodinového signály a obvod se následně nechoval nepředvídatelně. Proto jsou tyto sítě jen málo programovatelné a není jich příliš mnoho [11][12].

2.3 I/O bloky

Vstupně-výstupní bloky umožňují komunikace mezi vnitřní logikou FPGA a I/O (*In/Out*) piny. Každému I/O pinu přísluší jedna I/O buňka, která obsahuje tři základní signálové cesty. Vstupní cesty přenáší data z pinu do FPGA. Výstupní cesta zajišťuje přenos dat z vnitřní logiky FPGA na výstupní pin. A nakonec cesta ovládající třístavový výstup (stav vysoké impedance). Přenosové rychlosti na I/O pinech dosahují stovek Mb/s. [14]

3 NÁVRH KOMUNIKAČNÍHO PROTOKOLU

Tato kapitola se bude zabývat způsobem, kterým jsou strukturovány jednotlivé pakety protokolu pro komunikaci PC s FPGA. V zadání práce je specifikováno, že protokol musí podporovat operace pro zápis, čtení a zápis s kontrolou pro jeden registr a také pro celý paměťový blok. Dále bude muset zajišťovat odesílání telemetrických dat, která budou přicházet autonomně z obvodu FPGA.

Při návrhu komunikačního protokolu je nejprve zvolen obecný formát paketů, který je stejný pro zápis, čtení a zápis s kontrolou. Jednotlivé pakety obsahují vždy identifikační záhlaví, adresu v paměti a ukončovací část, která ohraničí daný paket. Na obrázku 3.1 je znázorněn obecný formát navržených paketů. Každý paket začíná záhlavím, které je definováno jako dvojice znaků ESC a CMD. ESC znak je kontrolovaná veličina, která nabývá hexadecimální hodnoty 0xFF. Pokud se tento znak objeví v komunikaci, je to znamení, že následuje příkazový nebo identifikační bajt. CMD je identifikátor paketu, který říká, jakým způsobem zpracovat následující informace.

Obrázek 3.1 Obecný formát paketů

Dále následuje část s názvem „ADRESA“, která je rozdělena do dvou částí o velikostech 1 B, kde první část obsahuje informaci o nejvýznamnějším bajtu (MSB). Druhá část naopak nese informaci o nejméně významném bajtu (LSB).

Prostor mezi adresou a ukončovací částí je vyhrazen pro dodatkové informace v případě čtení nebo pro datovou část paketu v případě přenosu informací. Poslední dva bajty jsou definovány jako ukončovací část paketu. Tato část je složena z kontrolního znaku „ESC“ a identifikací pro ukončení paketu „EOP“. „EOP“ znak nabývá hexadecimální hodnoty 0xF0. Hodnoty všech znaků jsou shrnuty v tabulce 3.1.

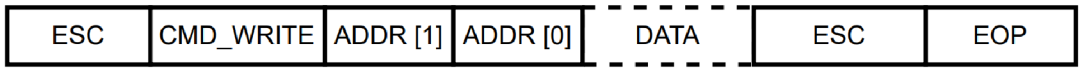
Tabulka 3.1 Výpis hexadecimálních hodnot použitých znaků

Znak	Hexadecimální hodnota	Popis
ESC	0xFF	ESC znak
CMD_WRITE	0x01	Příkaz pro zápis
CMD_READ	0x02	Příkaz pro čtení
CMD_WRITEREAD	0x03	Příkaz pro zápis s kontrolou
RPL_READ	0x20	Odpověď na CMD_READ
RPL_WRITEREAD	0x30	Odpověď na CMD_WRITEREAD
RPL_TELEMETRY	0x40	Identifikátor telemetrických dat
EOP	0xF0	Konec paketu

V tabulce 3.1 jsou vypsány všechny znaky, které jsou v paketech použity, jejich hexadecimální hodnoty a popis. Je vidět, že znaky ohraničující pakety mají horní polovinu bajtu nastavenou do 0xF. Identifikační znaky se dělí podle toho, jestli jde o příkaz nebo odpověď. Příkazové znaky mají obsazeny spodní polovinu bajtu, zatímco znaky s odpovědí horní část.

3.1 Zápis do paměti

Paket, který slouží pro zápis daných informací na určitou adresu v paměti, se skládá z hlavičky, adresy přenášených dat, dat samotných a ukončovací části. Hlavička obsahuje identifikátor paketu, který se nazývá CMD_WRITE a nabývá hexadecimální hodnoty 0x01. Následuje část pro přenos adresy o velikosti 2 B a část DATA, jejíž velikost je v tomto momentě omezena pouze pravidlem, že musí být násobkem velikosti slova, kde slovo je definováno jako 32 bitů nebo také 4 B. Výsledná možná maximální velikost datové části bude zjištěna až v implementaci protokolu na straně FPGA. Datová část, a tím samotný paket, je uzavřen ukončovací sekvencí znaků ESC a EOP. Popsaný paket je schematicky zobrazen na obrázku 3.2. níže.



Obrázek 3.2 Paket pro zápis informací do paměti

3.2 Čtení z paměti

Pro čtení z paměti, a to buď z jednoho registru nebo paměťového bloku slouží paket, který je vyobrazen obrázku 3.3. Tento paket se skládá z hlavičky, adresy, délky dat, které se mají přečíst a ukončovacího znaku. Hlavička paketu pro čtení obsahuje identifikátor CMD_READ, který nabývá hexadecimální hodnoty 0x02. Následuje adresa o velikosti 2 B a délka dat, která se mají přečíst. Paket je standardně ukončen sekvencí znaků ESC a EOP.

Obrázek 3.3 Paket pro čtení informací z paměti

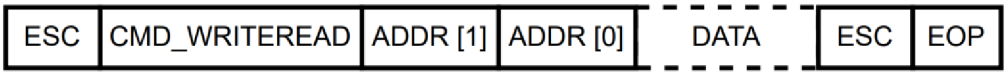
Pro čtení paměťové bloku bude v sekci DATAL nastaven počet adres, které se odešlou jako odpověď. Stejně jako u adresy je velikost této části 2 B. Čtení registrů probíhá od adresy ADRESA (včetně) až po adresu ADRESA + DATAL - 1. Odpověď s obsahem registru nebo paměťového bloku bude odeslána ve formátu paketu, jako na obrázku 3.4. Tento paket je podobně jako paket pro zápis složen z hlavičky, datové části a ukončovací části. Hlavička obsahuje identifikátor RPL_READ, který

nabývá hexadecimální hodnoty 0x20. Datová část tohoto paketu není v tuto chvíli žádným způsobem omezena. Je však možné předpokládat, že i maximální velikost dat, které je možné jedním paketem přenést bude omezena po implementaci protokolu do obvodu FPGA.

Obrázek 3.4 Paket s odpovědí pro žádost „CMD_READ“

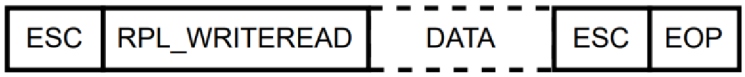
3.3 Zápis s kontrolou

Pro zápis do paměti s následnou kontrolou daného zápisu slouží paket, který je vyobrazen na obrázku 3.5. Paket obsahuje hlavičku, adresu, datovou část a ukončovací část. Hlavička obsahuje identifikátor CMD_WRITEREAD, který nabývá hexadecimální hodnoty 0x03. Adresa je standardně rozdělena do dvou bajtů a datová část je pro tuto chvíli neomezená. Paket se ukončuje sekvencí znaků ESC a EOP.



Obrázek 3.5 Paket pro zápis dat s očekávanou odpovědí

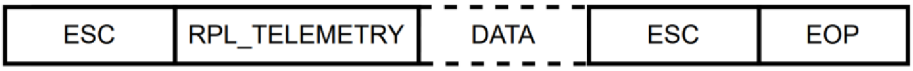
Na paket vyobrazený na obrázku 3.5 výše, očekáváme odpověď ve tvaru, který je vidět na obrázku 3.6. Hlavička tohoto kontrolního paketu obsahuje identifikátor ve tvaru RPL_WRITEREAD, který nabývá hexadecimální hodnoty 0x30. Zbytek paketu se standardně skládá z datové a ukončovací část.



Obrázek 3.6 Paket potvrzující správné zapsání do paměti

3.4 Telemetrické údaje

Pro odesílání telemetrických údajů z FPGA bude sloužit speciální paket, jehož formát je vyobrazen na obrázku 3.7. Paket se skládá z hlavičky, datové části a ukončovací části. Hlavička obsahuje identifikátor RPL_TELEMETRY, který nabývá hexadecimální hodnoty 0x40. Dále následuje datová část, která není prozatím žádným způsobem omezena. S omezením se počítá při implementaci, kdy se předpokládá, že maximální velikost dat narazí na limitace FPGA. Paket je ukončen sekvencí ESC a EOP.



Obrázek 3.7 Paket s daty z telemetrie

4 ZPRACOVÁNÍ PAKETŮ NA STRANĚ PC

Tato část práce popisuje implementaci protokolu, navrženého v kapitole 3, na straně počítače, a to v podobě knihovny vytvořené v programovacím jazyce Python. Ke komunikaci s USB ovladačem, konkrétně s ovladačem „Ports (COM & LPT)“ byla zvolena knihovna *pySerial* na níž byly vystavěny funkce, které jsou uvedeny v požadavku této bakalářské práce.

4.1 Python pySerial

Tato knihovna umožňuje na libovolný COM port zapisovat a zároveň z tohoto portu číst data, a to za pomoci jednoduché konstrukce. Pro otevření portů postačí jednoduchý zápis, který zároveň tento port definuje:

```
Fdevice = serial.Serial(PORT, BAUDRATE, …)
```

Pro zápis a čtení z daného portu zde poslouží funkce *write* a *read*:

```
Fdevice.write(bytes/bytearray)
Fdevice.read(SIZE = 1)
```

Funkce *Fdevice.write* umožňuje posílat data typu „*bytes*“ nebo „*bytearray*“ na zařízení, které je v našem případě definováno jako *Fdevice*. Pro případ čtení je zde funkce *Fdevice.read*, jejíž jediný parametr je počet bajtů, které musí přečíst ze sériové sběrnice.

Dalším důležitým faktorem, který je potřeba znát, je jakým způsobem funkce *Fdevice.write()* zpracovává bajtové pole (*bytearray*), respektive v jakém pořadí odesílá bajtové pole ke zpracování do USB ovladače. Na obrázku 4.1 je ukázán příklad paketu uloženého v bajtovém poli. Hodnoty ve hranatých závorkách vyjadřují pořadí v daném poli.

ESC	CMD_READ	ADDR [1]	ADDR [0]	DATAL [1]	DATAL [0]	ESC	EOP
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Obrázek 4.1 Příklad bajtového pole (bytearray)

Knihovna *pySerial* následně odesílá data způsobem, který je vyjádřen na obrázku 4.2. Je vidět, že jako první je odeslána položka pole s označením [0] a jako poslední se odešle poslední položka pole, v tomto případě [7]. Odesílání jednotlivých bitů je po sběrnici USB následně řízeno podle USB 2.0 protokolu 0.

EOP	ESC	DATAL [0]	DATAL [1]	ADDR [0]	ADDR [1]	CMD_READ	ESC	USB ovladač
[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]	

Obrázek 4.2 Řazení bajtů v případě odeslání do USB ovladače

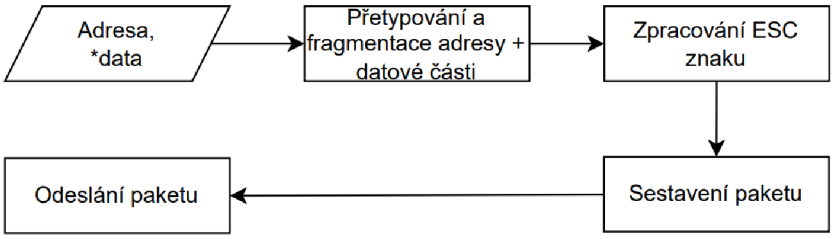
4.2 Implementace na straně PC

Pro zpracování protokolové části komunikace bude sloužit knihovna vytvořená v programovacím jazyce Python. Knihovna se nazývá „FPGA“ a skládá se ze tří skupin funkcí:

- hlavní funkce
- pomocné funkce a
- funkce sestavující pakety

Skupina hlavních funkcí obsahuje funkce pro zápis dat (*FPGA.write*), čtení dat (*FPGA.read*) a zápis s kontrolou (*FPGA.ctrl_write*). Pouze tyto funkce by měly být využívány uživatelem pro komunikaci s modulem FPGA.

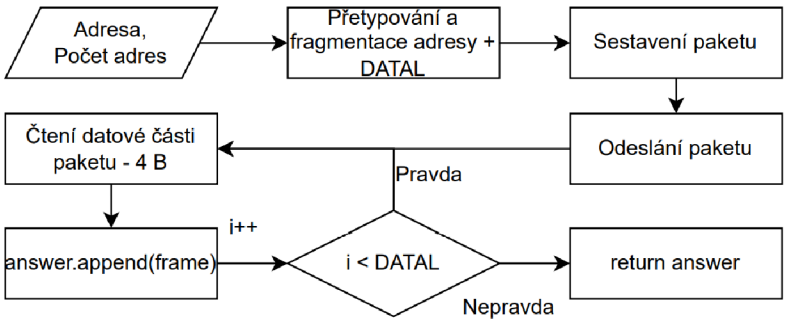
Funkce *FPGA.write(ADRESA, DATA)* při volání vyžaduje dva vstupní parametry - ADRESA a DATA. Parametr data je předáván jako libovolně velké pole, za pomocí konstrukce **data*. Oba tyto parametry jsou předávány jako celé číslo (*integer*). Ve funkci se následně tyto parametry převedou na formát, který lze poslat po sériové lince, tj. bajty (*bytes*) nebo bajtové pole (*bytearray*). V další části je jednoduchý algoritmus ze skupiny pomocných funkcí, který zpracuje ESC znaky v adresní i datové části paketu. Následně se sestaví požadovaný paket a odešle po sériové lince. Funkce pro sestavení paketů a pro zpracování ESC znaků bude popsána dále. Diagram popisující provedení funkce je znázorněn na obrázku 4.3.



Obrázek 4.3 Diagram pro funkci FPGA.write(ADRESA, DATA)

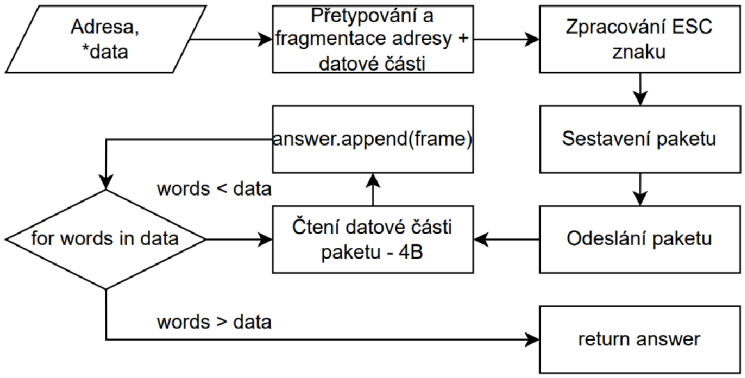
Funkce *FPGA.read(ADRESA, DATAL)* při volání akceptuje dva parametry. Parametr ADRESA a DATAL, a to ve formátu celého čísla (*integer*). ADRESA, podobně jako ve funkci *FPGA.write()*, bude převedena a rozdělena na dva samostatné bajty. Totéž je provedeno i s parametrem DATAL. Následně se tyto údaje ohraničí do definovaného paketu a jsou odeslány po USB sběrnici do cílového zařízení. Tato funkce, na rozdíl od funkce *write*, je blokující a očekává odpověď. Diagram popisující provedení funkce je znázorněn na obrázku 4.4.

V případ, kdy uživatel chce přečíst jen jeden registr na dané adrese, není parametr DATAL povinný, protože je ze základu nastaven na hodnotu jedna. V případě čtení paměťového bloku je nutné nastavit parametr DATAL na počet adres, které mají být přečteny.



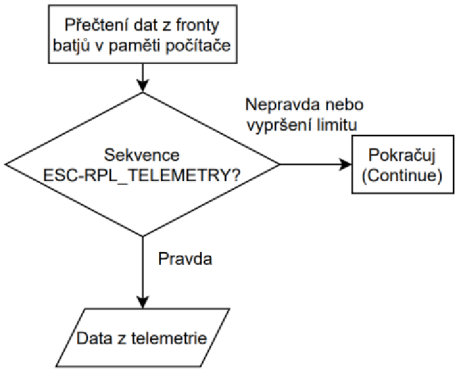
Obrázek 4.4 Diagram pro funkci FPGA.read(ADRESA, DATAL)

Funkce *FPGA.ctrl_write(ADRESA, DATA)* při volání přijímá stejné parametry jako funkce *FPGA.write()* a zároveň vrací přečtená data ze zapsané části podobně jako funkce *FPGA.read()*. ADRESA a DATA jsou ve formátu celého čísla (*integer*). Při volání funkce nejdříve dojde k přetypování a fragmentaci adresy, následně dochází k přetypování a ošetření ESC znaku v datové části a sestaví se požadovaný paket podle návrhu 1.3. Paket je následně odeslán a funkce čeká na odpověď. Diagram výsledné funkce je na obrázku 4.5.



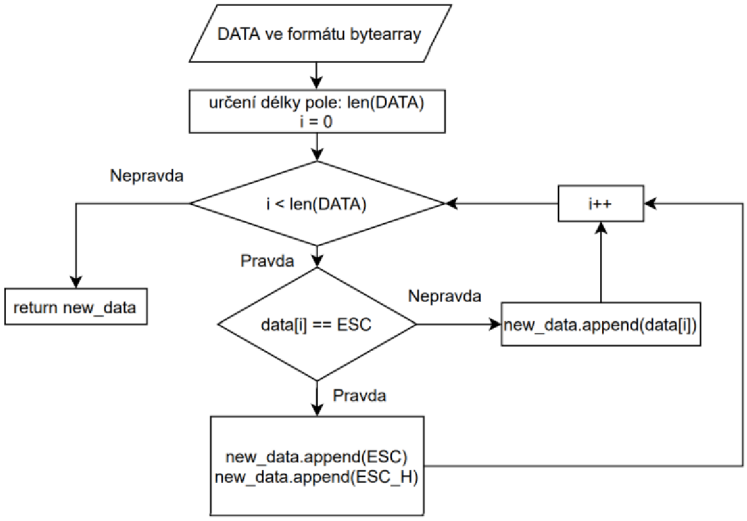
Obrázek 4.5 Diagram pro funkci FPGA.ctrl_write(ADRESA, DATA)

Pro příjem automaticky odesílaných dat z telemetrie je možné využít funkci *FPGA.telemetry_check()*. Tato funkce nepřijímá, žádné vstupní parametry. U této funkce se předpokládá, že bude volána periodicky a v případě nalezení paketu telemetrie v paměti počítače, uloží přečtená data. V případě, že tato funkce nic nenajde nebo ji vyprší časovač, se nic nezapíše a program se nezdrží jako v případě funkcí *FPGA.read()* a *FPGA.ctrl_write()*. Popisovaná funkce je vyobrazena na obrázku 4.6.



Obrázek 4.6 Diagram pro funkci FPGA.telemetry_check()

Pomocné funkce obsahují repetitivní části kódu, jako je například zpracování ESC znaku v datové části nebo fragmentace adresy na dva samostatné bajty. Na obrázku 4.7 je zobrazena funkce, která provádí zpracování ESC znaku v datové části komunikace. Jedná se o „*for*“ cyklus, který porovnává každý prvek pole s ESC znakem. V případě, že narazí na ESC znak, ho nahradí dvojicí znaků ESC – ESC_H a zapíše do nového řetězce, jinak zapíše do nového řetězce data původní. Funkce po ukončení vrací tento nový řetězec.



Obrázek 4.7 Diagram zpracování ESC znaku v datové části paketu

Fragmentace adresy nebo délky dat DATAL probíhá v další funkci, která dané celé číslo rozdělí na dva samostatné bajty. Výsledek operace je vrácen jako bajtové pole (*bytearray*). V následující ukázce je znázorněn způsob, jakým je řešena fragmentace. Nejvýznamnější bajt (MSB) je získán tím, že je na prvních 8 bitech použita maska a následně celé pole posunuto o jeden bajt do prava. Nejméně významný bajt (LSB) je získán logickým součinem 0x00FF a vstupními daty (viz.níže). Vypočtené fragmenty

jsou předány do pole, které funkce následně vrací. Obdobný způsob je využit i v případě převodu datové části paketu na bajty.

```
MSB_mask = 0xFF00
LSB_mask = 0x00FF
def fragment_data(data):                    #vstupní data ve formátu int
      MSB_fragment = MSB_mask & data
      MSB_fragment = MSB_fragment >> 8      #uložení MSB

      LSB_fragment = LSB_mask & data        #uložení LSB

      Fragmented_data = bytearray()         #uložení fragmentů do pole
      Fragmented_data.append(MSB_fragment)
      Fragmented_data.append(LSB_fragment)

      return Fragmented_data                #vrácení pole
```

Funkce, které jsou určeny k sestavování paketů jsou také odděleny z důvodu lepší přehlednosti kódu. Na začátku každé funkce se vytvoří proměnná s atributem „*bytearray()*“. K této proměnné je posléze možné přidávat data, a to pomocí metody „*append()*“ pro *integer* v rozmezí 0 <= x <256 nebo „*extend()*“ pro další bajtové pole.

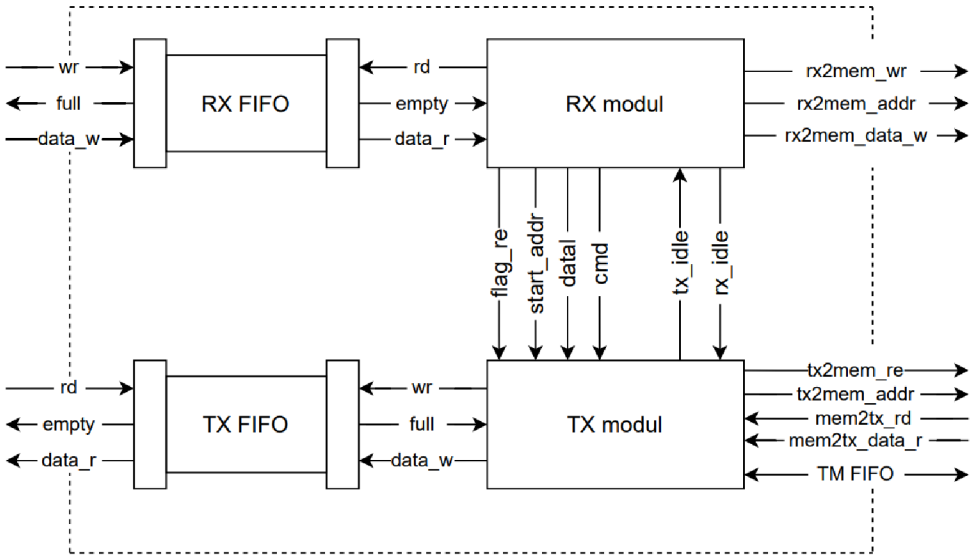
5 ZPRACOVÁNÍ PAKETŮ NA CÍLOVÉM ZAŘÍZENÍ

Modul zobrazený na obrázku 5.1 umožňuje příjem a zpracování příchozích paketů a v závislosti na typu paketu zápis nebo čtení dat z paměti. V případě dostupnosti dat z telemetrie umožňuje tento modul autonomně tato data zpracovávat a odesílat.

Příkazy v podobě řady bajtů přicházejí přes vstupní port (RX FIFO), se kterým lze komunikovat za pomoci tří signálů. Vstupními signály jsou *wr* a *data_w* a výstupním signálem je *full*, kde *wr* je povolovací signál pro zápis dat *data_w* o velikosti jednoho bajtu. Příznakový bit *full* upozorňuje na plné FIFO a v případě log. 1 nejsou přijímána další data. S výstupním portem (TX FIFO) lze také komunikovat za pomoci tří signálů. Výstupními signály jsou *empty* a *data_r* a vstupním signálem je *rd.* Pro čtení z tohoto portu je nutné nastavit příznakový bit *rd* do hodnoty log. 1. Data jsou následně vystavována na datový výstup *data_r* s frekvencí hodinového signálu. Příznakový bit *empty* upozorňuje na prázdné FIFO a čtená data nejsou v tomto případě validní.

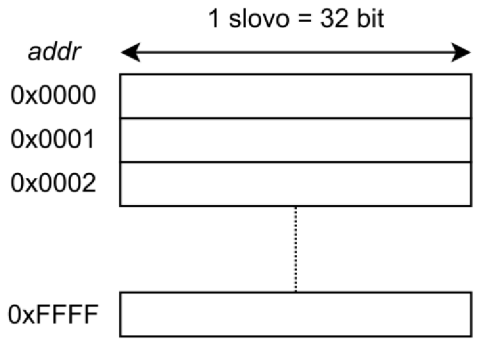
Pro zápis do paměti jsou z navrženého modulu vyvedeny tři signály, a to *wr, addr* a *data_w*. Příznakový bit *wr* značí, že data na signálech *addr* a *data_w* jsou validní, kde *addr* je adresa v paměti o velikosti dvou bajtů a *data_w* jsou data o velikosti jednoho slova, která se mají zapsat do dané paměťové buňky. Předpokládaná reprezentace paměti je znázorněna na obrázku 5.2.

V případě čtení z paměti jsou z modulu vyvedeny čtyři signály. Požadavek pro čtení je zajištěn signály *re* a *addr*, kde *addr* je adresa v paměti, ze které chceme číst. Tato adresa je validní v případě, že je signál *re* v log. 1. Data z paměti jsou následně přenesena pomocí signálů *rd* a *data_r,* kde jsou data z *data_r* o velikosti jednoho slova dostupná v případě, že je signál *rd* v log. 1.



Obrázek 5.1 Link control

V současné implementaci existuje omezení, kdy je přístup do paměti a odesílání či příjem dat v daném časovém okamžiku povolen jen jednomu z modulů. Tím je zajištěno, že nedojde ke konfliktu při přístupu do paměti či při komunikaci po lince USB. Informaci o aktuálním stavu si oba moduly předávají za pomoci dedikovaných signálů, přičemž přednost má ten, který spustí svoji činnost (opustí neaktivní stav *idle*) dříve.



Obrázek 5.2 Rozložení paměti

Vstupní signály pro telemetrii jsou na obrázku 5.1 souhrnně označeny jako TM FIFO. Jedná se opět o FIFO, ale s tím rozdílem, že v rámci jednoho hodinového taktu jsou vydávány čtyři bajty (jedno slovo). TM FIFO je obsluhováno v případě, že se zde nachází data, tedy že příznakový bit *empty* je nastaven do log. 0.

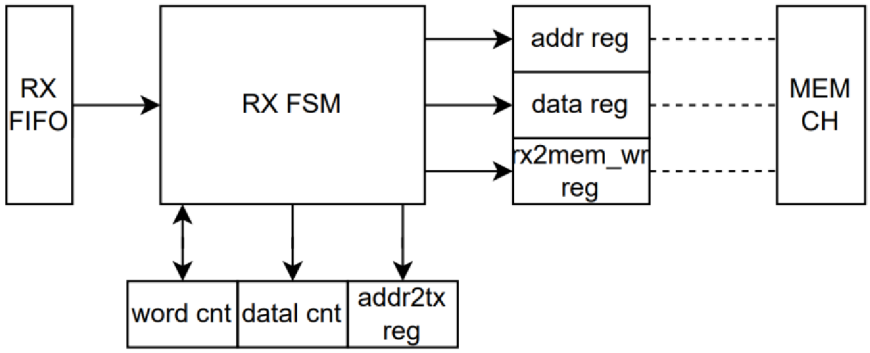
Jednotlivé moduly komunikují s FIFO za pomoci již zmíněných signálů *rd, empty* a *data_r* v případě čtení příchozích dat přijímacím modulem RX, nebo za pomoci signálů *wr, full* a *data_w* pro odesílací modul TX. V rámci komunikace mezi přijímacím a odesílacím modulem jsou předávány signály *flag_re, start_addr, datal* a *cmd,* které řídí proces čtení z paměti. Signál *start_addr* označuje počáteční adresu čtení a *datal* poskytuje informaci o počtu slov, která mají být přečtena. Příznak *cmd* dává informaci o zpracovávaném požadavku (čtení nebo zápis se čtením) a *flag_re* informuje o validitě dat na již zmíněných signálech.

5.1 Přijímací modul

Přijímací modul má za úkol zpracovat příchozí pakety a na základě formátu paketu zapisovat data do paměti nebo předat požadavek do odesílacího modulu. Ke zpracování paketů slouží stavový automat, jakožto jádro modulu a pomocné registry. Struktura přijímacího modulu je vyobrazen na obrázku 5.3. Registry pro komunikaci s pamětí jsou *addr* pro předání požadované adresy o velikosti dvou bajtů, *data* pro přenos slov a příznakový registr *rx2mem_wr*, který spustí požadované procesy pro zápis dat na danou adresu v paměti.

Dalším pomocným registrem je čítač pro určení definované velikosti slova *word cnt* v příchozích datech. Registr *datal cnt* slouží k uložení a předání informace o počtu slov,

která mají být v odesílacím modulu přečtena. Spolu s velikostí paměti, která má být přečtena, je v registru *addr2tx* předána i počáteční adresa v paměti.

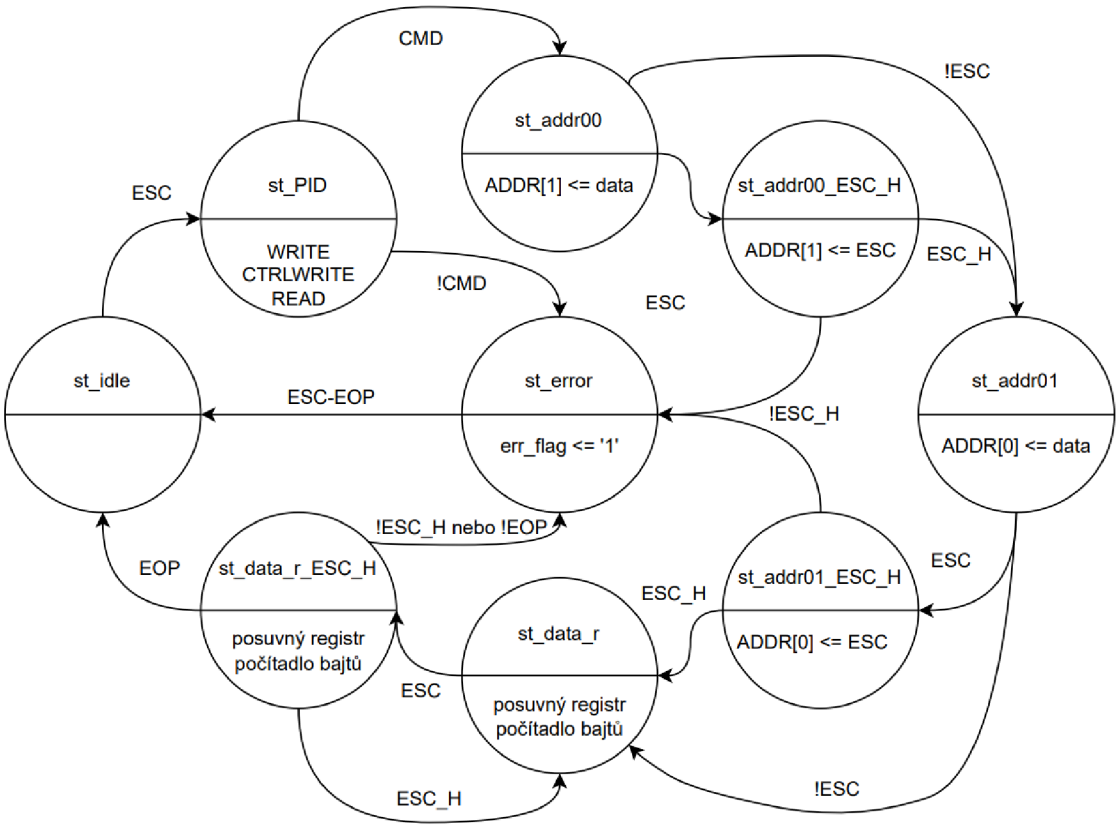


Obrázek 5.3 RX modul – registry

Stavový diagram navrženého stavového automatu vidíme na obrázku 5.4. Výchozím stavem je *st_idle*, kde automat vyčkává do doby, než jsou na vstupním FIFO (RX) dostupná data. Přechod do stavu *st_PID* je podmíněn přečtením ESC znaku, jinak automat setrvává ve stavu *st_idle*.

Ve stavu *st_PID* se očekává přečtení jednoho z definovaných znaků, které popisují, jak k následujícím datům přistupovat. Funkce automatu se po přečtení tohoto znaku následně dělí na tři možné větve: čtení, zápis a zápis se čtením. Tyto tři větve lze zjednodušit do formy, kterou vidíme na obrázku 5.4. V případě, že dojde k přečtení jiného než definovaného znaku, dojde k přechodu do chybového stavu *st_error*, kde automat vyčkává do doby, než se na vstupu objeví sekvence označující konec paketu ESC-EOP. Poté přejde automat do stavu *st_idle*. Po správné identifikaci paketu dojde k uložení adresy do registru *addr*. Tento proces je proveden ve dvou, respektive ve čtyřech stavech v případě výskytu ESC znaku v příchozí adrese. Jednotlivé bajty jsou postupně ukládány do registru *addr* vyobrazeném na obrázku 5.3.

Pokud je příchozí paket označen jako paket pro zápis, tedy s identifikačním bajtem CMD_WRITE nebo CMD_WRITEREAD dochází ke čtení a ukládání dat do posuvného registru *data* o velikosti jednoho slova. Jakmile je registr naplněn, dochází k předání dat a příslušné adresy do paměti. O kontrolu množství dat v registru *data* se stará čítač *word cnt*. S každým příchozím slovem se výchozí adresa v registru *addr* inkrementuje a dochází k zápisu do paměťového bloku. V případě výskytu ESC znaku v datové části přechází automat do stavu *st_data_r_ESC_H*, kde zkontroluje výskyt znaku ESC-H. Pokud je tento znak nalezen, uloží se do posuvného registru pouze znak ESC. Tento proces se opakuje do doby, dokud není zachycena ukončovací sekvence ESC-EOP.



Obrázek 5.4 Stavový diagram – Přijímací modul

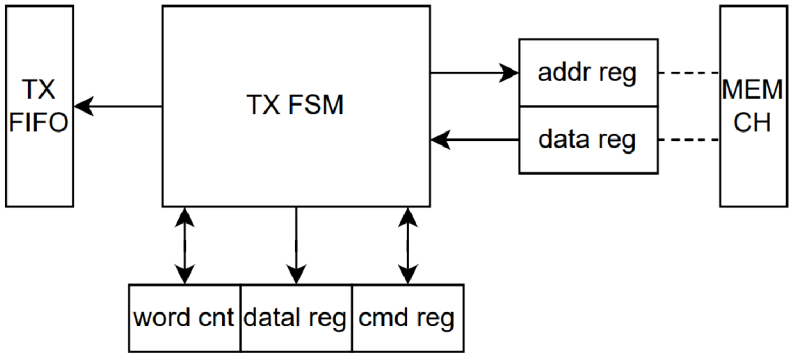
Další možností je, že je příchozí paket označen jako paket pro čtení. V tomto případě datová část paketu reprezentuje počet slov, která se mají přečíst z paměti a jsou uložena do registru *datal*. Po přečtení celého paketu dochází k předání informace o délce dat a adresy do odesílacího modulu. V případě příkazu CMD_WRITEREAD dochází k obdobné operaci, ale s rozdílem, že je délka dat odvozena z množství příchozích dat.

Stavový automat se vrací do stavu *st_idle* v případě, že se v příchozích datech objeví sekvence ESC-EOP. Tato kontrola probíhá ve stavu *st_data_r_ESC_H*.

5.2 Odesílací modul

Odesílací modul, zobrazený na obrázku 5.5, má za úkol plnit požadavky na čtení z paměti, přicházející z přijímacího modulu, balení odesílaných dat do příslušných paketů a autonomní zpracování a čtení telemetrických dat. Podobně jako u přijímacího modulu, se i zde nachází pomocné registry pro zpracování paketů. V případě čtení z paměti je příchozí počáteční adresa uložena do registru *addr*, kde se může následně dle potřeby inkrementovat. Data z jedné adresy v paměti jsou uložena do posuvného registru *data*, jehož obsah je následně zpracován stavovým automatem. Tento registr je společně se čtením z paměti sdílen i pro data přicházející z TM FIFO, u něhož je

předpoklad, že s hodinovým taktem poskytuje data o velikosti jednoho slova. Dalším pomocným blokem je čítač *word cnt*, který kontroluje počet bajtů v jednom slově v souvislosti s ESC znakem. Tento znak je v případě přečtení z paměti reprezentován sekvencí ESC-ESC_H. Registr *datal*, jehož hodnota je převzata z RX modulu, reprezentuje paměťový blok, který je potřeba přečíst. „*cmd*“ registr uchovává informaci o zpracovávaném paketu.

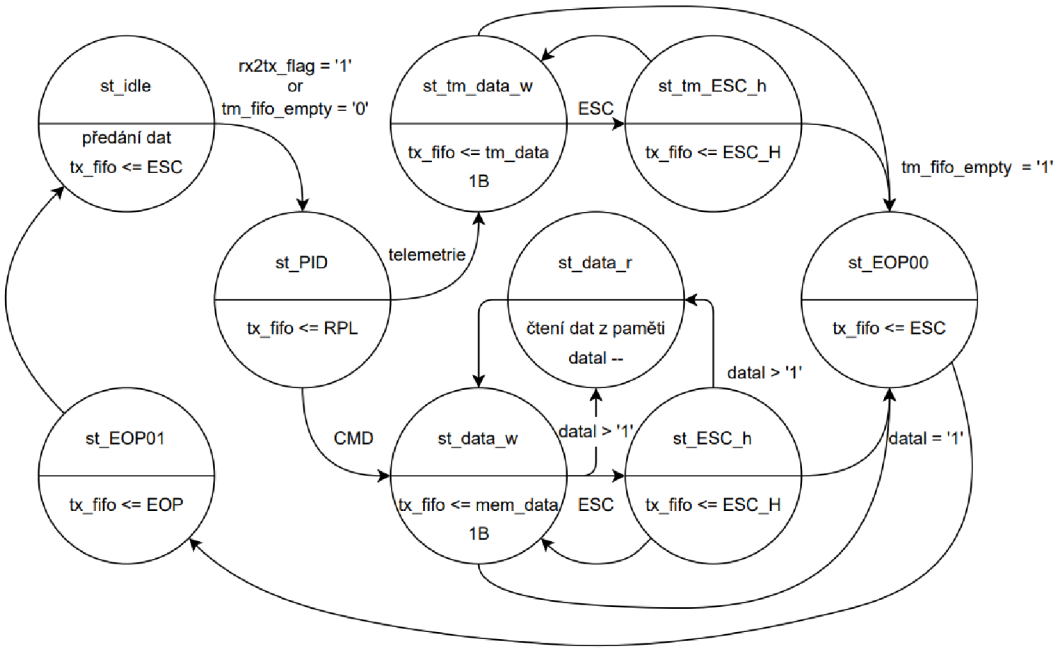


Obrázek 5.5 TX modul – registry

Odesílací stavový automat, jehož stavový diagram je zobrazen na obrázku 5.6, se skládá ze dvou větví – větev, která zajišťuje čtení z paměti a následné odeslání paketu a větev obsluhující autonomní odesílání telemetrie. Ve výchozím stavu se automat nachází ve *st_idle*, kde setrvává do doby, než bude příznakový registr z přijímacího modulu nastaven do hodnoty log. 1. Další možností, jak může automat přejít do dalšího stavu, je v případě zjištění dostupných dat telemetrie. Je předpokládáno, že rozhraní k telemetrii je implementováno za pomoci FIFO rozhraní, a dostupnost dat je zajištěna v případě, že je příznak *tm_fifo_empty* v log. 0. V případě reakce na signál od přijímacího modulu je ve stavu *st_idle* nejdříve formulován požadavek na čtení z paměti a následně je do registru *data* uložena hodnota dat. V momentě, kdy jsou data z paměti přečtena, se na výstupní FIFO (TX) zapíše ESC znak označující začátek paketu a automat přechází do stavu *st_PID*. V tomto stavu se na výstup zapíše RPL znak, a to na základě hodnoty registru *cmd*. Jednotlivé RPL znaky jsou definovány v tabulce 3.1.

V případě, že automat reaguje na požadavek od přijímacího modulu, přechází do stavu *st_data_w*, kde jsou jednotlivá slova v registru *data* z paměti rozložena na bajty a posílána do výstupního FIFO. Případný ESC znak je zpracován ve stavu *st_ESC_h*, kde po ESC znaku pošle na výstup i speciální znak ESC_H. Na základě hodnoty registru *datal*, je určen počet slov, která se mají z paměti přečíst. V případě, že je čteno více než jedno slovo, dochází po přečtení každého slova k inkrementaci registru *addr*. Požadavek na čtení je následně zpracován ve stavu *st_data_r*. Automat v tomto stavu setrvává do doby, než přijde potvrzení o validitě dat z paměti. Následně přechází do stavu *st_data_w* a proces se opakuje do doby, než je splněno zadání z přijímacího modulu. V momentě, kdy automat odešle poslední slovo, přechází do stavu

st_EOP00. Ve stavovém diagramu na obrázku 5.6 je znázorněno, že do ukončovacího stavu je možné přejít ze stavů *st_data_w* a *st_ESC_h*. Je to dáno možností výskytu ESC znaku na posledním odesílaném bajtu ve slově. Ve stavu *st_ESC00* a navazujícím *st_ESC01* je na výstup odeslána ukončovací sekvence ESC-EOP a automat se vrací do výchozího stavu *st_idle*.



Obrázek 5.6 Stavový diagram – Odesílací modul

Zpracování telemetrie zajišťují stavy *st_tm_data_w a st_tm_ESC_h*. Tyto stavy, podobně jako větev pro zpracování dat z paměti, využívají sdílený registr *data*, jehož obsah následně rozkládají na jednotlivé bajty a odesílají do výstupního FIFO. Případný ESC znak v příchozí telemetrii je ošetřen ve stavu *st_tm_ESC_h*, kde se na výstup zapíše náhradní sekvence ESC-ESC_H. Vzhledem k tomu, že telemetrická data přicházejí skrze FIFO rozhraní o velikosti buňky jednoho slova, je možné vynechat stav pro čtení, jako tomu bylo v případě první větve. Automat zpracovává příchozí data do doby, než je příznakový bit *tm_fifo_empty* v hodnotě log. 1, tedy v momentě, kdy je TM FIFO prázdné. Následně přechází automat do ukončovacího stavu *st_EOP00*, kde se tento paket ukončuje sekvencí ESC-EOP. Automat se vrací do výchozího stavu *st_idle*.

5.3 Výsledky implementace do FPGA

Syntéza a implementace modulu pro příjem dat v FPGA byla provedena v návrhovém prostředí „Vivado“ pro cílové FPGA Artix-7. Součástí navrženého modulu pro zpracování paketů, jsou také TX FIFO a RX FIFO pro příjem a odesílání dat přes rozhraní USB. Oba posuvné registry byly nastaveny na šířku slova 8 bitů a hloubku paměti 32 slov.

Dle výsledků syntézy, je pro implementaci tohoto modulu třeba 405 náhledových tabulek (LUT), z nichž 16 bylo využito jako paměťové bloky (*Distributed RAM*) a 260 klopných obvodů (DFF). Kritická cesta, která určuje maximální pracovní frekvenci, se nachází na rozhraní mezi RX FIFO a přijímacího modulu a kvůli zpoždění 11,73 *ns* je maximální možný kmitočet omezen na 85 MHz.

6 TESTOVÁNÍ APLIKACE

Testování aplikace bylo provedeno dvěma způsoby. V prvním případě je testován modul *Link_control* pomocí simulátoru digitálních obvodů, kde jsou postupně ověřeny jednotlivé případy, které se mohou v komunikaci vyskytnout. V druhém případě je otestováno obecné spojení komunikace mezi PC a FPGA a určeny limity přenosu z hlediska maximální rychlosti zpracování požadavku a maximální frekvence odesílané telemetrie v závislosti na velikosti datové části paketu a celkového počtu paketů.

6.1 Simulace modulu

Za pomoci simulačního nástroje ISim byly nejdříve otestovány základní funkce jednotlivých funkcí modulu *Link_control*. Jednotlivé testy a jejich výsledky jsou shrnuty v tabulce 6.1. Jedná se o zpracování základních paketů pro čtení, zápis a zápis se čtením. Dále jsou mezi základní testy zahrnuty testy zpracování telemetrie pro jedno nebo pro více slov. V druhé části tabulky jsou shrnuty obdobné testy, ale s tím rozdílem, že v datové části paketu jsou hodnoty odpovídající ESC znakům, které vyžadují ošetření.

Tabulka 6.1 Základní testy funkčnosti na úrovni simulátoru

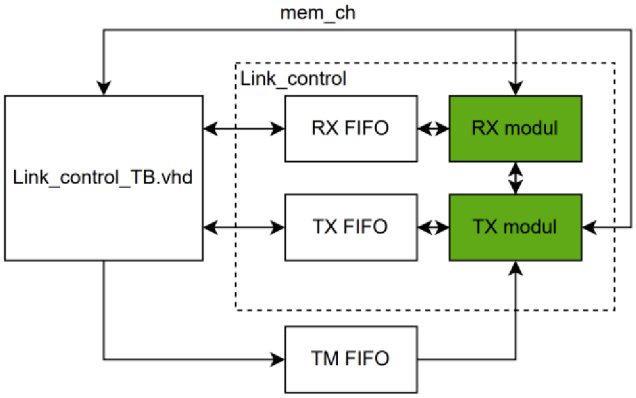
Základní funkčnost				
Č. testu	Název testu	Příkaz	Tvar paketu	Výsledek
1	Zápis (1 slovo)	CMD_WRITE	FF 01 00 50 12 34 56 78 FF F0	OK
2	Zápis (více slov)	CMD_WRITE	FF 01 00 50 12 34 56 78 AB CD EF FF FF F0	OK
3	Čtení (1 slovo)	CMD_READ	FF 02 00 50 00 01 FF F0	OK
4	Čtení (více slov)	CMD_READ	FF 02 00 50 00 02 FF F0	OK
5	Zápis se čtením (1 slovo)	CMD_CTRLWRITE	FF 03 00 50 11 22 33 44 FF F0	OK
6	Zápis se čtením (více slov)	CMD_CTRLWRITE	FF 03 00 50 AB CD EF FF 12 34 56 78 FF F0	OK
7	Telemetrie (1 slovo)	RPL_TELEMETRY	12 34 56 78	OK
8	Telemetrie (více slov)	RPL_TELEMETRY	AB CD EF FF 12 34 56 78	OK
ESC znaky				
Č. testu	Název testu	Příkaz	Tvar paketu	Výsledek
9	Zápis ESC paketu (1 slovo)	CMD_WRITE	FF 01 00 50 FF FF FF FF FF F0	OK
10	Zápis ESC paketu (více slov)	CMD_WRITE	FF 01 00 50 FF FF FF FF FF FF FF FF F0	OK
11	Čtení ESC paketu (1 slovo)	CMD_READ	FF 02 00 50 00 01 FF F0	OK
12	Čtení ESC paketu (více slov)	CMD_READ	FF 02 00 50 00 02 FF F0	OK
13	Zápis se čtením: ESC (1 slovo)	CMD_CTRLWRITE	FF 03 00 50 FF FF FF FF FF F0	OK
14	Zápis se čtením: ESC (více slov)	CMD_CTRLWRITE	FF 03 00 50 FF FF FF FF FF FF FF FF F0	OK
15	Telemetrie ESC paket (1 slovo)	RPL_TELEMETRY	FF FF FF FF	OK
16	Telemetrie ESC paket (více slov)	RPL_TELEMETRY	FF FF FF FF FF FF FF FF	OK

V první fázi simulace byly jednotlivé moduly otestovány samostatně. V případě přijímacího modulu bylo ověřeno, že jsou data z RX FIFO správně čtena a zpracovávána. Dále bylo ověřeno předání dat do odesílacího modulu a zápis do paměti, zejména pak pro zápis do paměťového bloku. V rámci ověření funkce pro zápis se čtením bylo ověřeno, že je správně určen počet slov, který byl zapsán do paměti. Odesílací modul byl otestován na čtení z paměti nebo paměťového bloku, a to pro případ kdy byla data dostupná ve stejném hodinovém taktu, ale i při zpoždění více hodinových taktů

při přístupu do paměti. Dále bylo otestováno zpracování dat z telemetrie a jejich autonomní odeslání.

V druhé fázi simulací, byly tyto bloky propojeny a byl vytvořen testovací soubor, který umožňuje zapisovat jednotlivé pakety na vstupní FIFO (RX). Tento modul je vyobrazen na obrázku 6.1, kde je součástí testovacího souboru generátor telemetrie a paměť, do které lze zapisovat nebo z ní číst. Jednotlivé pakety odpovídají formátu, který je zobrazen v tabulce 6.1. Klíčové testy se týkají především zpracování ESC znaku v různých částech datové části, tj. na začátku, uprostřed a na rozhraní dvou slov.

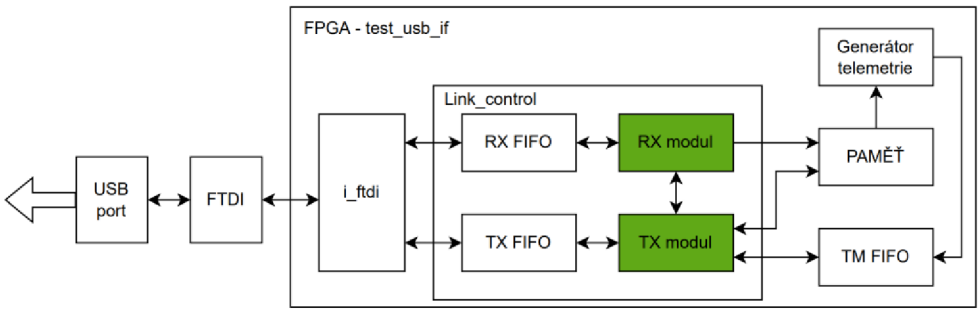
Všechny simulované případy dopadly podle předpokladu – v případě zápisu bylo zapsáno jedno slovo či paměťový blok. V případě čtení bylo přečteno jedno slovo, respektive paměťový blok. Zápis se čtením, který kombinuje předchozí dvě funkce také pracoval dle předpokladu. Data z telemetrie byla zpracována v momentě, kdy byla dostupná. Odesílací modul byl také otestován pro případ výskytu ESC znaků v datové části, kde na výstup správně zapsal sekvenci ESC-ESC_H na různých polohách datové části paketu. Simulační soubor a vizuální dokumentace k jednotlivým testům jsou dostupné v příloze.



Obrázek 6.1 Simulační struktura

6.2 Test aplikace na obvodu FPGA

V rámci zpracování protokolu na straně FPGA byly použity či nově vytvořeny další funkční bloky, jejichž zapojení v testovací aplikaci je zobrazeno na obrázku 6.2. Přijímací stavový automat je napojen na přijímací FIFO (RX), které je následně připojeno k ovladači čipu FTDI (*i_ftdi*). Obdobně je napojen i odesílací stavový automat (TX). Telemetrie je zde reprezentována jako FIFO (TM), které je ovládáno generátorem dat, jenž lze nastavit pomocí několika konfiguračních registrů. Tyto registry pak umožňují nastavit délku jednotlivých paketů telemetrie, periodu odesílání telemetrických paketů a jejich celkový počet pro daný test.



Obrázek 6.2 Testovací struktura

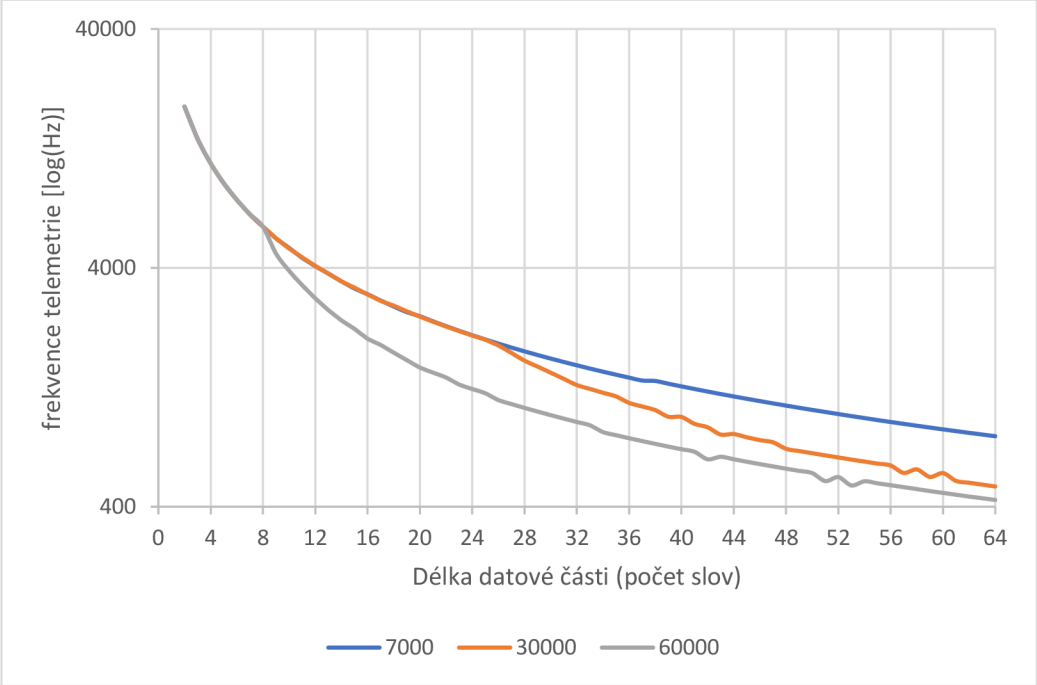
Pro testování implementace na reálném obvodu FPGA byla zvolena deska Nexys A-7, jejíž součástí je čip pro zpracování USB protokolu FTDI FT2232. Tento čip je ze strany FPGA ovládán pomocí ovladače, který je na obrázku 6.2 vyobrazen jako modul *i_ftdi*. USB rozhraní na straně počítače je ovládáno za pomoci knihovny *pySerial* (kapitola 0) na jejímž základě je postavena knihovna pro komunikaci s modulem *Link_control*. Prvním úkolem bylo najít maximální hodnotu parametru *Baudrate*, které dokáže FPGA generovat a při kterém je možné přenášet jednotlivé pakety. Tato hodnota byla stanovena na 2 *MBaud/s*.

Jedním z prvních testů, které byly provedeny, byl zápis numerické hodnoty o velikosti jednoho slova na danou adresu. Pro otestování funkčnosti byla adresa 0x0000 reprezentována na LED, které jsou součástí vývojové desky Artix-7. Tento test proběhl úspěšně – na diodách byla v binární hodnotě zobrazena velikost zapsaného čísla. Následně byla hodnota, která byla uložena do tohoto registru správně přečtena i na straně počítače za pomocí příkazu pro čtení. Obdobný test byl aplikován i na funkci pro zápis se čtením, která v podstatě tyto dvě předchozí funkce spojuje do jedné.

Dalším testem bylo určení maximální možné velikosti datové části pro funkce „Write“ a „Read“. V rámci tohoto testu byl vytvořen paket o velikosti datové části schopné přepsat celou paměť, tedy s o velikosti 256 *kB* a následně odeslán. Tento test proběhl pro zápis a čtení správně.

Hlavní test funkčnosti se týká především odesílání telemetrie, kdy bylo dle zadání potřeba zjistit, s jakou maximální frekvencí je možné naměřená data spolehlivě odesílat, dříve, než se začnou ztrácet. V první fázi testování se jednalo o zápis dat do jednotlivých registrů v paměti, které definovali vlastnosti telemetrie. Tyto registry nastavovaly počet odeslaných paketů, frekvenci, se kterou generátor telemetrie jednotlivá data zapisuje do TM FIFO a velikost datové části. Po ověření funkčnosti telemetrie byl v programovacím jazyce Python vytvořen skript, který hledal maximální frekvenci telemetrie v závislosti na velikosti datové části a počtu paketů, které jsou generovány. Základem algoritmu je opakované zapisování konfiguračních slov do definovaných registrů a čtení příchozí telemetrie do doby, než se začnou ztrácet data. V případě úspěšného přečtení je frekvence inkrementována o 30 Hz; v opačném případě je

frekvence snížena o stejnou hodnotu. Na základě tohoto algoritmu byly nalezeny limity odesílání telemetrie a výsledky shrnuty v grafu na obrázku 6.3, kde jednotlivé barvy reprezentují počet paketů, které byly odeslány v jednom cyklu. Je vidět, že s rostoucí délkou datové části, schopnost navrženého modulu zpracovávat a odesílat telemetrická data klesá. To se projevuje jako klesající maximální frekvence odesílání telemetrických dat z cílového zařízení. Maximální rychlosti, které telemetrie dosahuje je při malých délkách datové části paketu, a to 17 *kHz*. Rychlost přijímaní je v takových případech ovlivněna vysokou režií komunikace, kdy identifikační části paketu v případě přijímaní paketu o délce slova dva zabírají 30% kapacity komunikační linky. Režií je ovlivněna také rychlost přenosu, kdy maximální rychlosti 100 *kB/s* dosahují až pakety s délkou datové části deset a více slov. Dalším faktorem, který byl již zmíněn je počet paketů odeslaných v jednom cyklu. Se zvyšujícím se počtem paketů dochází ke snižování schopnosti zpracovávat data z telemetrie oproti původní modré křivce, která reprezentuje 7000 přijatých paketů. Z provedených testů lze zároveň usuzovat, že velikost TX FIFO (5 x 8 B) je optimální a při zvětšování jeho hloubky nedochází k výrazné změně maximální přenosové rychlosti. Všechna naměřená data jsou dostupná v příloze.



Obrázek 6.3 Limity zpracování telemetrie

Na tomto místě je nutné ještě podotknout, že pro tento test byla zvolena hloubka TM FIFO 64 slov, která efektivně limituje maximální délku telemetrických paketů a pro ověření rychlosti přenosu pro větší délky telemetrických paketů by bylo třeba nejprve zvětšit hloubku tohoto posuvného registru.

7 ZÁVĚR

Cílem této bakalářské práce bylo navrhnout proprietární komunikační protokol pro přenos dat mezi FPGA a PC. V návrhu komunikačního protokolu bylo zapotřebí definovat pakety pro zápis a čtení z daného registru nebo paměťového bloku v FPGA. Dále bylo zapotřebí definovat také pakety pro data přicházející periodicky z FPGA do PC. Tato periodicky se opakující se data mohou být například data z telemetrie. Formát jednotlivých paketů je vyobrazen na obrázcích 3.1– 3.7.

V průběhu tvorby knihovny funkcí pro komunikaci s FPGA bylo zapotřebí identifikovat, která z dostupných knihoven programovacího jazyka Python bude nejvíce vhodná pro danou aplikaci. Zvolená knihovna pro komunikaci v rámci sběrnice je *pySerial*, která umožňuje zapisovat a číst data z COM portu a tím zajistit základní komunikační kanál s FPGA. Druhou částí tvorby knihovny byl návrh funkcí, které zpracovávají uživatelské vstupy do jednotlivých paketů a odesílají data po sériové lince do cílového zařízení. Následně byly také vytvořeny funkce, které zajišťují příjem dat z cílového zařízení a extrahují z nich požadovaná data. Jednotlivými funkcemi jsou *write, read, ctrl_write* a *telemetry_check*.

V následujícím kroku byl implementován řadič protokolu na straně FPGA, který může být využit v libovolném projektu. Univerzálnosti pro různé projekty je dosaženo pomocí unifikovaného rozhraní pro připojení paměti cílového zařízení či telemetrického rozhraní. V textu práce byla popsána struktura a funkce takového řadiče, skládajícího se z přijímacího a odesílacího modulu a vstupního a výstupního FIFO, které slouží pro přenos dat mezi řadičem a ovladačem rozhraní USB.

V poslední části této práce jsou popsány metody testování jednotlivých bloků na úrovni simulace a testování pomocí vzorové aplikace pro určení limitu přenosu telemetrických dat. Výstupní graf z daného testování je na obrázku 6.3 a zobrazuje závislost maximální možné frekvence telemetrie na množství dat, která se musí v navrženém modulu zpracovat.

LITERATURA

[1] TRENZ ELECTRONIC GMBH. *Spartan-3E FPGA Industrial Micromodule: User Manual* [online]. 2011-10-04 [cit. 2021-12-08]. Dostupné z: https://shop.trenz-electronic.de/trenzdownloads/Trenz_Electronic/Modules_and_Module_Carriers/USB_OEM_Modules/TE0300_series/TE0300/documents/UM-TE0300.pdf

[2] *Universal Serial Bus Specification: USB 2.0* [online]. April 27, 2000 [cit. 2021-12-08]. Dostupné z: https://www.usb.org/document-library/usb-20-specification

[3] Boitan, A., Halunga, S., Bîndar, V. et al. *Compromising Electromagnetic Emanations of USB Mass Storage Devices. Wireless Pers Commun* (2020). https://doi.org/10.1007/s11277-020-07329-8

[4] SOBAN, Zbynek. *Universal Serial Bus: Speed Identification* [online]. 6. October 2003 [cit. 2021-12-09]. Dostupné z: https://hw-server.com/universal-serial-bus-usb-speed-identification

[5] HOPKINS, Jessica. *An Introduction to the USB Protocol* [online]. 28 July 2020 [cit. 2021-12-09]. Dostupné z: https://www.totalphase.com/blog/2020/07/about-the-usb-protocol-common-usb-bus-errors-and-how-to-troubleshoot-them/

[6] WANG, Shuangbao Paul. *Computer Architecture and Organization* [online]. [cit. 2021-12-09]. ISBN 978-981-16-5662-0. Dostupné z: https://doi.org/10.1007/978-981-16-5662-0

[7] *FTDI Chip* [online]. [cit. 2021-12-01]. Dostupné z: https://ftdichip.com/

[8] *Cypress: Semiconductor* [online]. [cit. 2021-12-01]. Dostupné z: https://www.cypress.com/

[9] *FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC Datasheet. FTDI Chip* [online]. 2019-05-27 [cit. 2021-12-09]. Dostupné z: https://ftdichip.com/wp-content/uploads/2020/07/DS_FT2232H.pdf

[10] *Getting Started with FX2LP™. Cypress* [online]. 2021-03-19 [cit. 2021-12-09]. Dostupné z: https://www.cypress.com/file/44956/download

[11] DANĚK, Martin. *Programovatelná hradlová pole – FPGA* [online]. 2016 [cit. 2021-12-09]. Dostupné z: https://automa.cz/cz/casopis-clanky/programovatelna-hradlova-pole-fpga-2006_02_30930_672/

[12] KOLÁŘ, Milan. *Architektura a návrh FPGA obvodů* [online]. 2010 [cit. 2021-12-09]. Habilitační práce. Technická univerzita v Liberci. Dostupné z: http://dspace.tul.cz/bitstream/handle/15240/39016/U_692_M.pdf

[13] *Configurable Logic Blocks (CLBs) on an FPGA* [online]. February 27, 2020 [cit. 2021-12-09]. Dostupné z: https://www.ni.com/documentation/en/labview-comms/latest/fpga-targets/configurable-logic-blocks/

[14] *Xilinx: Spartan-6 Family Overview* [online]. [cit. 2021-12-09]. Dostupné z: https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

PC	Osobní počítač
FPGA	Programovatelná hradlová pole
ASIC	Zákaznický integrovaný obvod
USB	Univerzální sériová sběrnice
PID	Identifikátor paketu
EOP	Konec paketu
ESC	Únikový znak
LSB	Nejméně významný bajt
LSb	Nejméně významný bit
MSB	Nejvýznamnější bajt
MSb	Nejvýznamnější bit
USBD	USB ovladač
HO	Hostielský ovladač USB portu
HCD	Ovladač pro řízení HO
FIFO	První dovnitř, první ven
I^2C	*Multi-master* počítačová sběrnice
USART	Synchronní/Asynchronní sériové rozhraní
RX	Přijímač
TX	Vysílač
VHDL	Programovací jazyk pro popis hardware
I/O	Vstup/výstup
MUX	Multiplexor
LUT	Vyhledávací tabulka
SRAM	Statická RAM
FSM	Stavový automat
TM	Telemetrie