**Závěrečná zpráva**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | **Jméno** | | **Datum** |
| Autor: | Petr | Čechura | dd.mm.yyyy |
|  |  |  |  |

1. document change log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Verze | Datum | Autor | Strany | Popis změn |
| 1.0 |  |  | Všechny | První verze |
|  |  |  |  |  |

1. Obsah

[1 Úvod 4](#_Toc86057407)

[2 Aplikovatelné a Odkazované dokumenty 5](#_Toc86057408)

[2.1 Seznam aplikovatelných dokumentů 5](#_Toc86057409)

[2.2 Seznam odkazovaných dokumentů 5](#_Toc86057410)

[3 Definice a seznam zkratek 6](#_Toc86057411)

[3.1 Definice 6](#_Toc86057412)

[3.2 Psaní čísel 6](#_Toc86057413)

[3.3 Jednotky 6](#_Toc86057414)

[3.4 Zkratky 6](#_Toc86057415)

[4 Představení projektu 7](#_Toc86057416)

[5 Plán vývoje 8](#_Toc86057417)

[6 Popis Návrhu 9](#_Toc86057418)

[7 Verifikační Plán 10](#_Toc86057419)

[7.1 Verifikační matice 10](#_Toc86057420)

[7.2 Popis verifikačního prostředí 10](#_Toc86057421)

[7.3 Verifikační testy 11](#_Toc86057422)

[8 Výsledky implementace 13](#_Toc86057423)

1. Seznam Obrázků

[Obrázek 4‑1 Blokové schéma AAU 7](#_Toc86057323)

[Obrázek 5‑1 Vývojový diagram 8](#_Toc86057324)

[Obrázek 4‑1 Blokové schéma pomocné aritmetické jednotky 9](#_Toc86057325)

[Obrázek 4‑1 Blokové schéma verifikačního prostředí 10](#_Toc86057326)

1. Seznam Tabulek

[Table 2‑1 Applicable Documents 5](#_Toc86057327)

[Table 2‑2 Reference Documents 5](#_Toc86057328)

[Table 2‑2 Verifikační matice 10](#_Toc86057329)

# Úvod

# Aplikovatelné a Odkazované dokumenty

Všechny dokumenty sloužící jako zadávací dokumentace nebo odkazované v tomto textu jsou uvedeny v tabulkách dole. Pokud není známá přesná verze, je použito datum vydání dokumentu (měsíc/rok). V případě že není známé ani datum vydání, je jako verze uvedena hodnota 0.

## Seznam aplikovatelných dokumentů

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ref. | Název | | Číslo dokumentu |  | Verze |
|  | |  | - |  | - |
|  | | - | - |  | - |

Table 2‑1 Seznam aplikovatelných dokumentů

## Seznam odkazovaných dokumentů

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ref. | Název | | Číslo dokumentu |  | Verze |
|  | | Pomocná aritmetická jednotka  Semestrální projekt | 1 |  | - |
|  | | Requirement Specification | 2 |  | - |

Table 2‑2 Seznam odkazovaných dokumentů

# Definice a seznam zkratek

*Formální část dokumentu, doporučujeme inspirovat se obsahem podobné kapitoly ve specifikaci požadavků.*

## Definice

*Cokoliv je třeba definovat.*

## Psaní čísel

*Formát zápisu čísel.*

## Jednotky

Pouze jednotky ze soustavy SI jsou použity v textu.

## Zkratky

|  |  |
| --- | --- |
| AAU | Auxiliary Arithmetic Unit |
| FSM | Finite state machine |
|  |  |
|  |  |
| TBC | To Be Confirmed |
| TBD | To Be Defined |
|  |  |
| - | - |

# Představení projektu

Výsledkem projektu je architektura samostatné výpočetní jednotky, která komunikuje přes rozhraní SPI a je schopna provádět operace **sčítání** a **násobení**.

Jednotka obsahuje následující vstupy a výstupy:

* **CS\_b** *(vstup)* – Signál, kterým *master* zahajuje komunikaci překlopením do log. 0. Pokud komunikace neprobíhá, signál musí být v log. 1.
* **MOSI** *(vstup)* – Datový signál, obsahující bity pro provedení aritmetických operací.
* **MISO** *(výstup)* – Datový signál, obsahující výsledky aritmetických operací.
* **SCLK** *(vstup)* – Synchronizační signál, určující frekvenci komunikace; při náběžné hraně jednotka posílá bit do MISO a *master* jej přijímá, při sestupné hraně *master vysílá bity do jednotky a ta je přijímá.*
* **clk** *(vstup)* – Hodinový signál, který je pro všechny části architektury stejný.
* **reset** *(vstup)* – Resetovací signál, společný pro všechny části architektury.

Komunikace probíhá ve formě paketů, kde každý paket obsahuje dva rámce. Protože je využita jak náběžná, tak sestupná hrana signálu SCLK, součástí jednoho rámce jsou bity pro provedení operací (jdoucí do jednotky) a výsledek operace s čísly z předchozího paketu (jdoucí do *master*).

Čísla jsou ve formátu desetinných čísel s dvojkovým doplňkem a s pevnou řádovou čárkou na  
8. bitu. Protože jeden rámec má velikost 16 bitů, maximální číslo, které je možné vyjádřit pomocí jednoho rámce, je 127.9960938; nejmenší pak -128. Pokud výsledek některé z operací převyšuje maximální rozsah rámce, jednotka jej automaticky zaokrouhlí.

Architektura je navržena pro FPGA firmy Spartan3, konkrétně model . Hodinový signál má   
frekvenci 50 MHz.



Obrázek 4‑1 Blokové schéma AAU

*A následně stručné shrnutí, co bude obsahem této zprávy.*

# Plán vývoje

Vývoj výpočetní jednotky začal definicí všech požadavků a shromážděním doporučených postupů pro co největší eliminaci hrubých chyb, které by vedly k významným opravám v kódu. Návrh architektury se do velké míry řídil instrukcemi z dokumentace RD01.

Nejprve byl navržen funkční blok pro zpracování příchozího signálu dle standardu SPI (**SPI\_Interface)**. Ten obsahuje obvody pro zpracování asynchronního signálu a ošetření možných chyb, které při přenosu mohou vzniknout. Následovala samostatná verifikace SPI rozhraní, kterou bylo ověřeno, že jednotka je schopna zpracovat příchozí signál a z něj vytvořit paralelní slovo.

Druhým funkčním blokem je **PKT\_Control**, jehož součástí je stavový automat o 4 stavech, který má řídit průběh komunikace a adekvátně předávat data z SPI rozhraní do výpočetní jednotky (a naopak) podle současného stavu. Po návrhu znovu následovala verifikace, součástí které byla simulace všech možných situací, které v jednotce mohou nastat.

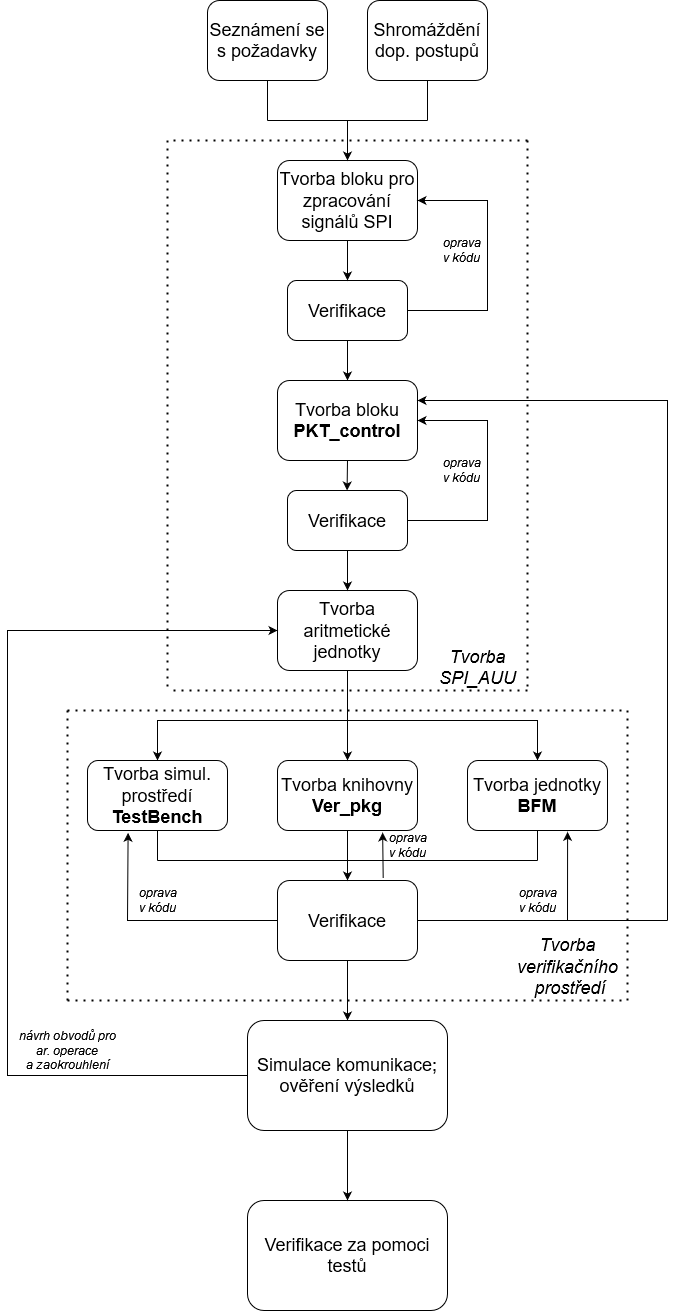
Třetí funkční blok je samotná aritmetická jednotka (**Aritmetical\_unit**), ve které jsou prováděny aritmetické operace a která řídí i proces zaokrouhlování. Výsledek je pak poslán zpět do SPI rozhraní, aby byl v dalším paketu vysílán do řídící jednotky. Pro usnadnění verifikace byla aritmetická jednotka v raných fázích zjednodušena tak, aby předávala stále stejný rámec – obvody pro výpočet a zaokrouhlování výsledku byly vytvořeny až ke konci vývoje (*viz dále*).

Tři funkční bloky dohromady tvoří kompletní výrobek, který bylo potřeba otestovat. K tomu bylo vytvořeno verifikační prostředí, jehož součástí je **TestBench**, ve kterém jsou v samostatném procesu prováděny testy. Testy využívají procedury z vlastní knihovny **Ver\_pkg**, která byla vytvořena pro přehlednost. Procedury jsou celkem tři (***SendRightPacket*, *SendWrongPacket*, *SendFrame***) a simulaci komunikace s testovanou jednotkou (**SPI\_AUU**) provádějí za pomoci bloku **BFM**, který podle pokynů z bloku **TestBench** vysílá do jednotky datové signály podle standardu SPI.

Po vytvoření funkčního verifikačního prostředí (podoba se několikrát změnila), schopného odeslat  
a přijmout paket, byl proveden první test, a sice odeslání platného paketu a příjem odpovědi. Pro správnou funkci byla nutná drobná úprava v architektuře jednotky (problém byl v datovém toku výsledků aritmetických operací uvnitř jednotky).

Jakmile přenos probíhal bez problémů, pozornost byla zaměřena na aritmetickou jednotku, tedy správnost výsledků a ošetření zaokrouhlování. Pro snadnější simulaci byl vytvořen v jazyce Python skript (**Converter.py)**, který převádí binární číslo do dekadické podoby s pevnou řádovou tečkou a s dvojkovým doplňkem.

Po ověření, že jednotka je schopna reagovat na platné rámce a odeslat výpočetně správné výsledky, byly za pomoci vytvořených procedur provedeny další testy, kterými byla verifikace dokončena.



Obrázek 5‑1 Vývojový diagram

# Popis Návrhu

Jednotka se skládá ze tří hlavních funkčních bloků:

* **SPI\_Interface (**rozhraní SPI),
* **PKT\_Control** (řadič paketů)
* **Aritmetical unit** (aritmetická jednotka)

Diagram

Description automatically generated

Obrázek 4‑1 Blokové schéma AAU

## SPI\_Interface

Blok, představující nejnižší vrstvu komunikace, který je navržen tak, aby zpracovával vstupní signály (tj. převáděla je na paralelní slovo) a vysílal výstupní signály do řídící jednotky podle standardu SPI. Součástí jsou i obvody pro detekci chyb přenosu; na případnou chybu reaguje blok **PKT\_Control** (*viz dále*).

Diagram

Description automatically generated

Obrázek 4‑2 Blokové schéma funkčního bloku SPI\_Interface

Celkem obsahuje 5 funkčních bloků:

* **Ris\_fall\_detector** – Detektor náběžné a sestupné hrany; použit pro detekci změny signálu SCLK a CS\_b.
* **ShiftInOut** – Podle náběžných a sestupných hran SCLK a CS\_b řídí funkci serialiseru a deserialiseru.
* **Deserialiser** – Registr, převádějící vstupní signál na paralelní slovo, které lze dále zpracovat v FPGA.
* **Serialiser** – Registr, obsahující data (výsledky aritmetických operací) k odeslání.
* **ErrorHandle** – Podle počtu náběžných a sestupných hran SCLK zjišťuje, zda přenos probíhá bez chyby. Zároveň detektuje začátek a konec komunikace.

## PKT\_Control

Zprostředkovává komunikaci mezi rozhraním SPI a aritmetickou jednotkou, představuje řídící blok celé komunikace. Součástí je stavový automat a časovač, který je použit pro měření času 1 ms; pokud není do této doby přijat druhý rámec, celý paket je považován za neplatný (*viz* **požadavky**).

Diagram

Description automatically generated

Obrázek 4‑3 Blokové schéma funkčního bloku PKT\_Control

Stavový automat má celkem 4 stavy:

* **S\_awaiting\_fr1** – Výchozí stav při neprobíhající komunikaci. Při začátku komunikace posílá výsledek součtu z předchozího paketu do rozhraní SPI a přechází v další stav.
* **S\_getting\_fr1** – Jednotka přijímá první rámec. Vyskytla-li se chyba, dochází k návratu do výchozího stavu. Bezchybný rámec je poslán do AU, do serialiseru je vyslán součin z předchozího paketu a nastává přechod do dalšího stavu.
* **S\_awaiting\_fr2** – Jednotka čeká na druhý rámec. Časovač měří dobu 1 ms; trvá-li čekání déle, stavový automat přechází do výchozího stavu.
* **S\_getting\_fr2** – Jednotka přijímá druhý rámec. Je-li přijat bez chyby, předává se dál do aritmetické jednotky a automat přechází do výchozího stavu. Při výskytu chyby se automat vrací do předchozího stavu a očekává opravu.

Diagram

Description automatically generated

Obrázek 4‑4 Stavový automat ve funkčním bloku PKT\_Control

## Aritmetical\_unit

Poslední funkční blok přijímá vstupní data pro výpočet součtu a součinu, výsledky pak dává na výstup. Součástí jsou rovněž kombinační obvody, které automaticky zaokrouhlí výsledek v případě přetečení nebo podtečení. Blok neobsahuje samostatné komponenty, je tedy napsán kompletně v jednom souboru.

Diagram

Description automatically generated

Obrázek 4‑5 Blokové schéma funkčního bloku Aritmetical\_unit

# Verifikační Plán

*Verifikační plán by měl definovat, jakým způsobem chcete ověřit požadavky kladené na návrh.   
Jiný způsob bude pro požadavky, které mají být ověřeny kontrolou dokumentace ([R]), simulací ([S]) či analýzou ([A]).*

*Při verifikaci splnění požadavku kontrolou dokumentace budou například uvedeny odkazy na jednotlivé kapitoly závěrečné zprávy, které splnění požadavku dokumentují. Při verifikaci pomocí simulace je pak třeba představit základní myšlenku simulace – tedy testování je prováděno pomocí testovacích vektorů, které jsou aplikovány na návrh a následně je vyhodnocena jeho odezva. V případě verifikace požadavku analýzou je pak třeba uvést, jakou analýzou je takový požadavek ověřen.*

## Verifikační matice

*Formální záležitost, ale zřejmě nejdůležitější část dokumentace týkající se verifikace. Matice obsahuje všechny požadavky aplikované na návrh, způsob jejich ověření a konstatování, zda byl požadavek úspěšně ověřen.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| REQ ID | Název | Ver. metoda | | | Verifikace požadavku | Splněno |
| **R** | **S** | **A** |
| REQ\_AAU\_G\_001 | Cílová technologie | A | - | - | Kapitola 8 | A/N |
| REQ\_AAU\_G\_002 | Synchronní návrh | A | - | - | Kapitola 8 |  |
| REQ\_AAU\_G\_003 | Výstupní signály | A | - | - | Kapitola 6.1 | A |
| REQ\_AAU\_G\_004 | Vstupní signály | A | - | - | Kapitola 6.1 | A |
| REQ\_AAU\_G\_005 | Bezpečná implementace FSM | A | A | - | Kapitola 8 |  |
| REQ\_AAU\_G\_006 | Dokumentace | A | - | - |  |  |
| REQ\_AAU\_F\_010 | Výpočetní jednotka | A | - | - | Kapitola 6.3 |  |
| REQ\_AAU\_F\_011 | Formát čísel | - | A | - | Test *tc\_spi\_001* | A |
| REQ\_AAU\_F\_012 | Zaokrouhlování čísel | - | A | - | Test tc\_spi\_005 | A |
| REQ\_AAU\_F\_013 | Přetečení operací | - | A | - | Test tc\_spi\_005 | A |
| REQ\_ AAU\_I\_020 | Frekvence SPI | - | A | - | Test *tc\_spi\_001* | A |
| REQ\_ AAU\_I\_021 | Pořadí bitů | - | A | - |  |  |
| REQ\_ AAU\_I\_022 | Nekompletní rámec | - | A | - | Testy tc\_spi\_003 a tc\_spi\_004 | A |
| REQ\_ AAU\_I\_023 | Reset komunikace | - | A | - | Test tc\_spi\_002 | A |
| REQ\_ AAU\_I\_024 | Formát paketů | - | A | - | Test tc\_śpi\_001 | A |

Table 2‑2 Verifikační matice

## Popis verifikačního prostředí

*Tato kapitola popisuje strukturu verifikačního prostředí a funkci jeho jednotlivých komponent. Podobně jako při popisu návrhu je nejprve představena celková struktura a popis předávání dat mezi jednotlivými komponentami. Následně jsou jednotlivé komponenty blíže představeny.*

Verifikační prostředí se skládá ze tří samostatných modulů:

* **TestBench** – Hlavní platforma, ve které jsou prováděny testy pomocí procedur z **Ver\_pkg**. Obsahuje dva samostatně běžící procesy:
* **BFM** – Ve své podstatě simulovaný řadič sběrnice, který je na požádání schopen odeslat a přijmout rámec podle standardu SPI. Komunikuje s modulem **TestBench**, od kterého přijímá data, které má odeslat, a kterému přijaté rámce také předává.
* **Ver\_pkg** – Knihovna, obsahující jak procedury pro provádění testů, tak vlastní datové typy pro snadnější manipulaci s daty.

Diagram

Description automatically generated

Obrázek 4‑1 Blokové schéma verifikačního prostředí

### TestBench

Řídící komponenta celého verifikačního prostředí; jsou v ní obsaženy všechny ostatní komponenty a pro realizaci testů jsou použity procedury z Ver\_pkg. Skládá se ze dvou samostatných procesů:

* **TESTCASE\_P** – Na začátku jsou definovány proměnné pro požadované testy (rámce, velikost rámců a zpoždění). Samotný proces začíná zapnutím hodinového signálu (clk\_en), pak následují jednotlivé testy. Na konci je hodinový signál vypnut a signalizován konec testování (test\_end) pro **SPI\_BFM**.
* **clk\_P** – Proces pro generování hodinového signálu, který je distribuován do testované jednotky.

Kromě procesů obsahuje TestBench pomocné signály:

* pro propojení BFM a testované jednotky SPI\_AUU
  + *s\_mosi*
  + *s\_miso*
  + *s\_sclk*
  + *s\_cs\_b*
* pro spojení s BFM v rámci odeslání a přijmu rámců
  + bfm\_com
  + bfm\_results

### BFM

Komponenta, která na základě příkazů, které obdrží ve formě signálu *bfm\_com*, posílá do výpočetní jednotky rámce s definovanou hodnotou a délkou.

Komunikace začíná, jakmile komponenta obdrží náběžnou hranu signálu *bfm\_com.start*; po odeslání rámce jsou signály, vedoucí do jednotky, vráceny do výchozích hodnot, a proces znovu čeká na náběžnou hranu *bfm\_com.start*. Proces končí, jakmile je signál *test\_end* v log. 1.

Odpověď z jednotky posílá komponenta formou signálu *bfm\_rep*.

Průběh odesílání a příjmu rámců probíhá podle standardu SPI, tj. s náběžnou hranou SCLK jsou data přijata a se sestupnou jsou odeslána. Pokud je délka rámce (*bfm\_com.fr\_size*) větší než 16, řadič posílá do jednotky požadovaný rámec (který má fixní délku 16) a po jeho odeslání posílá  
log. 0.

### Ver\_pkg

Knihovna, kterou používá primárně TestBench pro realizaci testů, obsahuje ale také definice datových typů *record* pro snazší komunikaci mezi moduly.

Datové typy:

* **Bfm\_com** – Sdružuje signály, které TestBench posílá do BFM.
  + **Frame** – Obsahuje paralelní slovo rámce, který je potřeba odeslat. Má pevně danou velikost 16 bitů.
  + **Fr\_size** – Definuje velikost odeslaného rámce. Může obsahovat jakoukoliv hodnotu.
  + Start – Signalizuje start přenosu.
  + **Reset** – Jediný signál, který není připojen do BFM, ale do AAU. Je použit pro resetování jednotky.
  + **Sclk\_period** – Definuje periodu signálu SCLK pro požadovaný přenos.
* **Bfm\_rep** – Sdružuje signály, které BFM posílá do TestBench.
  + **Result** – Výsledný rámec, který BFM získal z výpočetní jednotky.
  + **Done** – Signalizace, že byl obdržen kompletní rámec a je možné jej zpracovat.

Procedury:

* **SendRightPacket** – Procedura k odeslání platného packetu, tj. bez zpoždění a s validními délkami rámců. Začíná resetováním testované jednotky následuje za použití procedury SendFrame odesílání jednotlivých rámců, což je doplněno komentáři pro snadné sledování průběhu a výsledků. Odeslány jsou celkem 4 rámce: 2 obsahují data k odeslání, další dva jsou prázdné a jsou odeslány jen pro obdržení výsledků.
* **SendWrongPacket** – Na rozdíl od předchozí procedury umožňuje odeslat neplatný packet, tj. definovat délku a zpoždění. Protože procedura počítá s tím, že jeden z rámců nebude přijat (protože je neplatný), okamžitě posílá opravený platný rámec. Proto je možné definovat hodnoty třech rámců, aby bylo u porovnání výsledků možné poznat, které rámce jednotka přijala a které ne. Dohromady je tedy odesláno 5 rámců: 3 s daty, 2 pro příjem.
* **SendFrame** – Obsahuje odeslání jednoho rámce a ověření zda byl odeslán.

## Verifikační testy

*V této kapitole jsou uvedeny všechny testy pro ověření funkčnosti návrhu (a splnění požadavků). Popis každého testu by se měl skládat z následujících částí:*

* *Název testu*
* *Stručný popis testu – jakou vlastnost má ověřit, do jakých stavů je DUT uvedeno*
* *Seznam požadavků, které tento test ověřuje*
* *Detailní popis testovací procedury – v případě přímých testů jsou to pak jednotlivé kroky.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Název testu** | Ověření funkčnosti při odeslání platného paketu pro různé frekvence SCLK. | **Číslo testu.:** | tc\_spi\_001 |
| **Popis testu** | Účelem je ověřit, že při odeslání paketu se správnou délkou rámců a s hodnotami, u kterých nehrozí přetečení, jednotka paket adekvátně vyhodnotí a odešle zpět správné výsledky. Test je proveden pro frekvence 1 MHz, 100 kHz a 10 kHz. | | |
| **Reflektované požadavky** | REQ\_AAU\_I\_022 | | |
| **Postup testu** | 1. Nastavení frekvence SCLK na 1 MHz. 2. Reset DUT 3. BFM: Odeslán platný rámec s hodnotou 0001110000111110. 4. Zpoždění 200 µs. 5. BFM: Odeslán platný rámec s hodnotou 1111111111000011. 6. BFM: Přijat rámec s hodnotou 0001110000000001. 7. BFM: Přijat rámec s hodnotou 1111100101000101. 8. Nastavení frekvence SCLK na 100 kHz. 9. Opakování bodů 2. – 7. 10. Nastavení frekvence SCLK na 10 kHz. 11. Opakování bodů 2. – 7. 12. Ověření výsledků:   **Sčítání**   * 0001110000111110 + 1111111111000011 =   0001110000000001   * 28.2421875 - 0.23828125 = 28.00390625   **Násobení**   * 0001110000111110 ∙ 1111111111000011 =   1111100101000101   * 28.2421875 ∙ (-0.23828125) = -6.73046875 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Název testu** | Detekce zpožděného druhého rámce s následnou opravou | **Číslo testu.:** | tc\_spi\_002 |
| **Popis testu** | Účelem je ověřit, že při zpoždění druhého rámce jednotka celý paket ignoruje a čeká na správný paket. Frekvence SCLK je 1 MHz. | | |
| **Reflektované požadavky** | REQ\_AAU\_I\_022 | | |
| **Postup testu** | 1. Reset DUT 2. BFM: Odeslán platný rámec s hodnotou 0001110000111110. 3. Zpoždění 1100 µs. 4. BFM: Odeslán platný rámec s hodnotou 1111111111000011. 5. Zpoždění 100 µs. 6. BFM: Odeslán platný rámec s hodnotou 0000001100011100. 7. BFM: Přijat rámec s hodnotou 0000001011011111. 8. BFM: Přijat rámec s hodnotou 1111111101000010. 9. Ověření výsledků:   **Sčítání**   * 1111111111000011 + 0000001100011100 =   0000001011011111   * -0.23828125 + 3.109375 = 2.87109375   **Násobení**   * 1111111111000011 ∙ 0000001100011100 =   1111111101000010   * -0.23828125 ∙ 3.109375 = -0.7421875 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Název testu** | Detekce chybného prvního rámce | **Číslo testu.:** | tc\_spi\_003 |
| **Popis testu** | Účelem je ověřit, zda jednotka ignoruje neplatný první rámec a přijme teprve až ten správný. Nejprve je ověřeno, že jednotka ignoruje příliš krátký rámec, poté je stejná procedura zopakována pro příliš dlouhý rámec. Frekvence SCLK je 1 MHz. | | |
| **Reflektované požadavky** | REQ\_AAU\_I\_022 | | |
| **Postup testu** | 1. Reset DUT 2. BFM: Odeslán neplatný rámec s hodnotou 0000001110000111110  (délka 19 bitů). 3. Zpoždění 500 µs. 4. BFM: Odeslán platný rámec s hodnotou 1111111111000011. 5. Zpoždění 100 µs. 6. BFM: Odeslán platný rámec s hodnotou 0000001100011100. 7. BFM: Přijat rámec s hodnotou 0000001011011111. 8. BFM: Přijat rámec s hodnotou 1111111101000010. 9. Reset DUT 10. BFM: Odeslán neplatný rámec s hodnotou 1110000111110 (délka 12 bitů) 11. Zpoždění 500 µs. 12. BFM: Odeslán platný rámec s hodnotou 1111111111000011. 13. Zpoždění 100 µs. 14. BFM: Odeslán platný rámec s hodnotou 0000001100011100. 15. BFM: Přijat rámec s hodnotou 0000001011011111. 16. BFM: Přijat rámec s hodnotou 1111111101000010. 17. Ověření výsledků:   **Sčítání**  1111111111000011 + 0000001100011100 =  0000001011011111  -0.23828125 + 3.109375 = 2.87109375  **Násobení**  1111111111000011 ∙ 0000001100011100 =  1111111101000010  -0.23828125 ∙ 3.109375 = -0.7421875 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Název testu** | Detekce chybného druhého rámce | **Číslo testu.:** | tc\_spi\_004 |
| **Popis testu** | Účelem je ověřit, zda jednotka ignoruje neplatný druhý rámec a přijme teprve až ten správný. Nejprve je ověřeno, že jednotka ignoruje příliš krátký rámec, poté je stejná procedura zopakována pro příliš dlouhý rámec. Frekvence SCLK je 1 MHz. | | |
| **Reflektované požadavky** | REQ\_AAU\_I\_022 | | |
| **Postup testu** | 1. Reset DUT 2. BFM: Odeslán platný rámec s hodnotou 0001110000111110. 3. Zpoždění 500 µs. 4. BFM: Odeslán neplatný rámec s hodnotou 0001111111111000011 (délka 19 bitů). 5. Zpoždění 100 µs. 6. BFM: Odeslán platný rámec s hodnotou 0000001100011100. 7. BFM: Přijat rámec s hodnotou 0001111101011010. 8. BFM: Přijat rámec s hodnotou 0101011111010000. 9. Reset DUT 10. BFM: Odeslán platný rámec s hodnotou 0001110000111110. 11. Zpoždění 500 µs. 12. BFM: Odeslán neplatný rámec s hodnotou 1111111000011 (délka 12 bitů). 13. Zpoždění 100 µs. 14. BFM: Odeslán platný rámec s hodnotou 0000001100011100. 15. BFM: Přijat rámec s hodnotou 0001111101011010. 16. BFM: Přijat rámec s hodnotou 0101011111010000. 17. Ověření výsledků:   **Sčítání**  0001110000111110 + 0000001100011100 =  0001111101011010  28.2421875 + 3.109375 = 31.3515625  **Násobení**  0001110000111110 ∙ 0000001100011100 =  0101011111010000  28.2421875 ∙ 3.109375 = 87.8125 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Název testu** | Ověření schopnosti zaokrouhlit výsledky při přetečení nebo podtečení | **Číslo testu.:** | tc\_spi\_005 |
| **Popis testu** | Účelem je ověřit, zda je jednotka schopna zaokrouhlit výsledky sčítání a násobení, a to směrem nahoru i dolů. Nejprve jsou odeslány pakety s hodnotami 14.9453125 a 127.97265625, což vyústí v zaokrouhlení na hodnotu 127.99609375, následně pak pakety s hodnotami (-125.5234375, -112.015625) a (14.9453125, -24.02734375), které povedou pro zaokrouhlení dolů. Frekvence SCLK je 1 MHz. | | |
| **Reflektované požadavky** | REQ\_AAU\_I\_022 | | |
| **Postup testu** | 1. Reset DUT 2. BFM: Odeslán platný rámec s hodnotou 0000111011110010. 3. Zpoždění 200 µs. 4. BFM: Odeslán platný rámec s hodnotou 0111111111111001. 5. BFM: Přijat rámec s hodnotou 0111111111111111. 6. BFM: Přijat rámec s hodnotou 0111111111111111. 7. Reset DUT 8. BFM: Odeslán platný rámec s hodnotou 1000001001111010. 9. Zpoždění 200 µs. 10. BFM: Odeslán platný rámec s hodnotou 1000111111111100. 11. BFM: Přijat rámec s hodnotou 1000000000000000. 12. BFM: Přijat rámec s hodnotou 0111111111111111. 13. Reset DUT 14. BFM: Odeslán platný rámec s hodnotou 0000111011110010. 15. Zpoždění 200 µs. 16. BFM: Odeslán platný rámec s hodnotou 1110011111111001. 17. BFM: Přijat rámec s hodnotou 1111011011101011. 18. BFM: Přijat rámec s hodnotou 1000000000000000. | | |

# Výsledky implementace

*V závěrečné kapitole zprávy by měly být uvedeny výsledky implementace pro cílový obvod FPGA.*

*Kromě základních informací (využité zdroje, maximální pracovní frekvence) je nutné také uvést informace o tom, jakým způsobem byly výsledky získány. Jedná se především o použité nástroje včetně jejich verze, nastavení pro implementaci (konfigurace syntetizéru, implementační strategie pro P&R) a také omezení implementace (tzv. constraints), pokud jsou použity.*

*V tomto projektu jsou vstupní signály asynchronní, v takovém případě je žádoucí výslednou implementaci analyzovat z hlediska zpoždění na vstupních a výstupních signálech a dokázat dostatečnou časovou rezervu při zpracování signálů sběrnice SPI.*