



# 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

## FORMALNÍ VERIFIKACE RISC-V PROCESORU S VYUŽITÍM QUESTA PROPCHECK

FORMAL VERIFICATION OF RISC-V PROCESSOR WITH QUESTA PROPCHECK

### DIPLOMOVÁ PRÁCE

MASTER'S THESIS

### AUTOR PRÁCE

AUTHOR

Bc. Adrián Javor

### VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Vojtěch Dvořák

BRNO 2020



# Diplomová práce

magisterský navazující studijní obor **Mikroelektronika**

Ústav mikroelektroniky

**Student:** Bc. Adrián Javor

**ID:** 173666

**Ročník:** 2

**Akademický rok:** 2019/20

## NÁZEV TÉMATU:

### Formální verifikace RISC-V procesoru s využitím Questa PropCheck

## POKYNY PRO VYPRACOVÁNÍ:

1. Nastudujte instrukční sadu RISC-V, techniky a dostupné nástroje pro verifikaci hardwaru
2. Analyzujte možnosti užití dostupných nástrojů pro formální verifikaci
3. Proveďte návrh využití zvoleného nástroje pro formální verifikaci RISC-V procesoru
4. Implementujte návrh z bodu 3
5. Analyzujte výsledky a zhodnoťte přínos formálních technik oproti funkční verifikaci

## DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

**Termín zadání:** 3.2.2020

**Termín odevzdání:** 1.6.2020

**Vedoucí práce:** Ing. Vojtěch Dvořák

**Konzultant:** Ing. Tomáš Vaňák

**doc. Ing. Lukáš Fucik, Ph.D.**  
předseda oborové rady

## UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být 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.

# ABSTRAKT

Témou tejto diplomovej práce je Formálna verifikácia RISC-V procesoru s využitím Questa PropCheck pomocou formálnych tvrdení jazyka SystemVerilog Assertions. Teoretická časť práce je venovaná architektúre RISC-V, popisu vybraných komponentov procesora Codix Berkelium 5 určených na formálnu verifikáciu, popisu komunikačného protokolu AHB-lite, formálnej verifikácii, jej metódam a nástrojom. Praktickú časť tvorí návrh verifikačného plánu vybraných komponentov, ich následná formálna verifikácia, analýza výsledkov a zhodnotenie prínosu formálnych techník.

# KLÚČOVÉ SLOVÁ

Formálna verifikácia, RISC-V, Questa PropCheck, kontrola modelu, SystemVerilog assertions

# ABSTRACT

The topic of this master thesis is Formal verification of RISC-V processor with Questa PropCheck using SystemVerilog assertions. The theoretical part writes about the RISC-V architecture, furthermore, selected components of Codix Berkelium 5 processor used for formal verification are described, communication protocol AHB-lite, formal verification and its methods and tools are also studied. Experimental part consists of verification planning of selected components, subsequent formal verification, analysing of results and evaluating a benefits of formal technics.

# KEYWORDS

Formal verification, RISC-V, Questa PropCheck, model checking, SystemVerilog assertions

## **BIBLIOGRAFICKÁ CITÁCIA**

JAVOR, A. *Formální verifikace RISC-V procesoru s využitím Questa PropCheck*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky, 2020. 42 s., 18 s. příloh. Diplomová práce. Vedúci práce: Ing. Vojtěch Dvořák.

## **POĎAKOVANIE**

Ďakujem vedúcemu práce Ing. Vojtěchovi Dvořákovi za pedagogickú pomoc a cenné rady pri vytváraní verifikačného plánu. Taktiež ďakujem Ing. Tomášovi Vaňákovi za pripomienky k formálnym tvrdeniam a odbornú pomoc.

# OBSAH

<b>Úvod</b>	<b>1</b>
<b>1 RISC-V</b>	<b>2</b>
1.1 Inštrukčná sada .....	2
1.2 Štandardné rozšírenia.....	2
1.3 Formát inštrukcii.....	3
1.4 Codix Berkelium 5.....	4
1.4.1 Dátová pamäť L1 cache .....	5
1.4.2 Modul pre násobenie a delenie celočíselných hodnôt .....	7
1.4.3 Dekodér.....	8
<b>2 Protokol AMBA 3 AHB-LITE</b>	<b>11</b>
2.1 Obecná štruktúra .....	11
2.2 Prenos.....	12
2.2.1 Druhy prenosov.....	13
2.2.2 Druhy transakcii.....	14
2.2.3 Odpoveď prenosu.....	15
<b>3 Formálna verifikácia hardwaru</b>	<b>16</b>
3.1 Overovanie modelu.....	16
3.1.1 SystemVerilog Assertions.....	18
3.1.2 Okamžité formálne tvrdení .....	18
3.1.3 Súbežné formálne tvrdení .....	19
3.1.4 Verifikácia založená na formálnych tvrdeniach .....	22
Questa PropCheck .....	22
3.2 Dokazovanie viet .....	23
3.2.1 Automatické dokazovanie viet.....	23
<b>4 Návrh verifikačného plánu</b>	<b>26</b>
4.1 Dekodér.....	26
4.2 Modul pre násobenie a delenie celočíselných hodnôt .....	26
4.3 Dátová pamäť cache typu L1 .....	27

4.3.1	Rozhranie typu slave.....	27
4.3.2	Rozhranie typu master .....	28
<b>5</b>	<b>Priebeh verifikácie a vyhodnotenie</b>	<b>29</b>
5.1	Nastavenie Questa PropCheck.....	29
5.2	Výsledky verifikácie pomocou Questa PropCheck .....	30
5.2.1	Dekodér.....	30
5.2.2	Modul pre násobenie a delenie celočíselných hodnôt .....	32
5.2.3	Dátová pamäť typu L1 cache.....	35
5.3	Zhodnotenie prínosov formálnej verifikácie.....	40
	<b>Záver</b>	<b>42</b>
	<b>Literatura</b>	<b>43</b>
	<b>Zoznam obrázkov</b>	<b>45</b>
	<b>Zoznam tabuliek</b>	<b>46</b>
<b>A</b>	<b>Popis dekódovaných riadiacich signálov dekodéra</b>	<b>48</b>
<b>B</b>	<b>Zoznam RISC-V inštrukcií</b>	<b>51</b>
B.1	Inštrukcie vykonávané v module pre násobenie a delenie celočíselných hodnôt	51
B.2	Zoznam inštrukcií dekodéru aj s ich požadovanými hodnotami dekódovaných signálov.....	52
<b>C</b>	<b>Dátová pamäť typu L1 cache</b>	<b>55</b>
C.1	Zoznam konfigurácií naplánovaných na kontrolu .....	55
C.2	Zoznam požiadaviek na rozhraní typu slave.....	55
C.3	Zoznam požiadaviek na rozhraní typu master .....	56
C.4	Zoznam doplnených vlastností použitých ako predpoklady na rozhraní typu slave	59
C.5	Zoznam doplnených vlastností použitých ako predpoklady na rozhraní typu master	63
<b>D</b>	<b>Výsledky verifikácie</b>	<b>64</b>

# ÚVOD

Dnešný rýchly vývoj zložitých hardwarových návrhov vyžaduje spoľahlivé verifikačné metódy. Medzi najznámejšie metódy formálnej verifikácie patrí kontrola modelu. Kontrola modelu overuje správnosť hardwarového návrhu s ohľadom na požadované správanie sa. Je uskutočňovaná kontrolou stavového diagramu, ktorý reprezentuje hardwarový návrh. Overuje funkčnosť stavového diagramu podľa špecifikácie správania sa vyjadrenej vo vzorcach temporálnej logiky alebo konečného stavového automatu. Verifikačné nástroje pracujúce na metóde kontroly modelu nám umožňujú spoľahlivo potvrdiť 100% správnosť hardwarového návrhu vzhľadom na špecifikáciu. V prípade, ak návrh neprejde verifikáciou, sú tieto nástroje schopné vygenerovať protipríklad, ktorý definuje za akých podmienok návrh nespĺňa požadovanú špecifikáciu, čo predstavuje veľmi dôležitú spätnú väzbu pre úpravu hardwarového návrhu.[1]

Cieľom tejto diplomovej práce bolo zostavenie verifikačného plánu a následná formálna verifikácia vybraných komponentov procesora Codix Berkelium 5 od spoločnosti Codaip, založeného na architektúre RISC-V (redukovaná inštrukčná sada piatej generácie). V prvej kapitole teoretickej časti práce je rozobratá architektúra RISC-V a popis vybraných komponentov procesora. Nasledujúca kapitola pojednáva o komunikačnom protokole AHB-lite. Tretia kapitola sa venuje formálnej verifikácii, jej metódam a nástrojom. V štvrtej kapitole je navrhnutý verifikačný plán vybraných komponentov procesora Codix Berkelium 5 určených na formálnu verifikáciu pomocou nástroja Questa PropCheck. V poslednej kapitole je rozobraté nastavenie nástroja, priebeh verifikácie a vyhodnotenie dosiahnutých výsledkov spolu s porovnaním formálnej verifikácie s funkčnou verifikáciou založenou na princípe simulácie.

# 1 RISC-V

RISC-V je otvorený, voľne dostupný súbor inštrukcií (ISA) ktorý vznikol v roku 2010 na Kalifornskej univerzite v Berkeley. Pôvodne bol vytvorený za účelom vzdelávania a výskumu počítačových architektúr. RISC-V ISA prináša novú úroveň voľného, rozšíriteľného softwaru a slobody hardwarovej architektúry v oblasti navrhovania a inovácie výpočtovej techniky na ďalších 50 rokov. V roku 2015 bola založená nezisková organizácia RISC-V Foundation ktorej cieľom je vybudovať otvorenú spoluprácu v oblasti inovácie softwaru a hardwaru založeného na RISC-V ISA. Jej členovia sa podieľajú aj na vývoji špecifikácie RISC-V ISA a jej rozšíreniach. Výhodou RISC-V je ISA ktorý nie je zameraný na jeden typ mikroarchitektúry a implementačnú technológiu (ASIC, FPGA). Ďalšou výhodou je podpora veľkého množstva voliteľných rozšírení a režimov adresovania (32, 64, 128 bitových). Aktuálne sú podporované tri privilegované módy a to : strojový (machine), superužívateľský (supervisor) a užívateľský (user).[2,3,4]

## 1.1 Inštrukčná sada

RISC-V ISA je definovaný ako základný súbor inštrukcií pre celočíselné operácie, ktorý musí byť prítomný v každej implementácii s možnosťou doplnenia veľkým množstvom rozšírení. Každá základná celočíselná inštrukčná sada je charakterizovaná šírkou celočíselných registrov, odpovedajúcou veľkosťou adresného priestoru a počtom celočíselných registrov. Obsahuje celočíselné výpočtové inštrukcie, inštrukcie na zápis, čítanie a riadenie toku (skoky). Primárne existujú dve základné celočíselné varianty a to RV32I a RV64I, poskytujúce 32 a 64 bitový adresný priestor. Existuje však aj experimentálne rozšírenie RV128I poskytujúce 128 bitový adresný priestor. Základné celočíselné inštrukčné sady používajú na vyjadrenie znamienkových celočíselných hodnôt dvojkový doplnok.[4]

## 1.2 Štandardné rozšírenia

Za účelom podpory všeobecnejšieho vývoja softwaru bola definovaná sada štandardných rozšírení poskytujúcich celočíselné násobenie a delenie, atomické operácie a aritmetiku s jednoduchou a dvojistou presnosťou pohyblivej rádovej čiarky. Ide o nasledujúce štandardné rozšírenia :

- M (Multiplication) – pridáva inštrukcie násobenia a delenia hodnôt nachádzajúcich sa v celočíselných registroch
- A (Atomic) – pridáva inštrukcie atomického čítania, zápisu a modifikácie

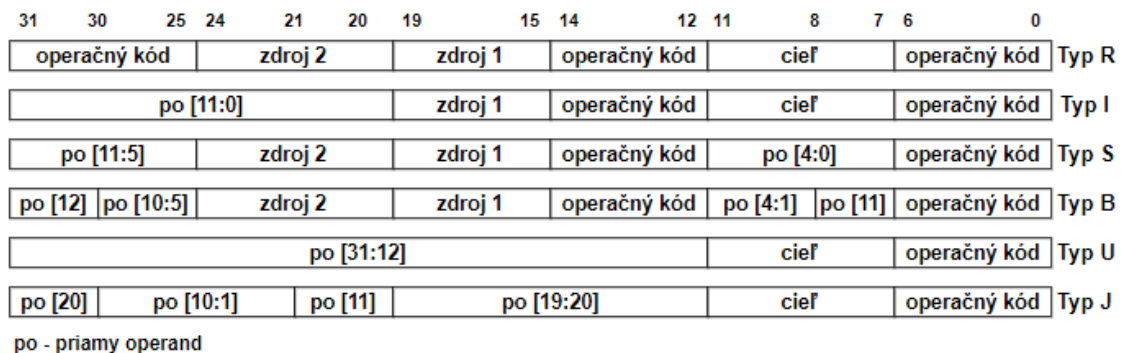
pamäte pre vnútro procesorovú synchronizáciu

- F (Float) - pridáva registre, výpočtové inštrukcie, inštrukcie načítania a uloženia pre čísla s jednoduchou presnosťou pohyblivej rádovej čiarky
- D (Double) – má obdobný charakter ako F so zvýšením jednoduchovej presnosti pohyblivej rádovej čiarky na dvojitú.
- C (Compressed) - pridáva užší 16-bitový formát bežných inštrukcií.[4]

V súčasnosti existujú aj viaceré rozšírenia tie však v práci nebudú rozoberané, nakoľko práca je zameraná na kombináciu RV32IMC.

### 1.3 Formát inštrukcií

Pre základnú inštrukčnú sadu RV32I existujú štyri hlavné formáty inštrukcií s označením R, I, S a U, ktoré sú zobrazené na Obrázku 1.1. Všetky základné inštrukcie majú pevnú dĺžku 32 bitov a musia byť v pamäti zarovnané na 4 bajty. Pozície zdrojových a cieľových registrov sú vo všetkých formátoch na rovnakom mieste za účelom zjednodušenia dekódovania. Priame operandy sú vždy znamienkovo rozšírené a umiestnené smerom k najvýznamnejšiemu bitu inštrukcie za účelom zníženia hardwarovej zložitosti. Výnimkou je 5-bitový priamy operand (immediate) používaný v inštrukciách pracujúcich s kontrolnými a stavovými registrami (tzv. control status registers CSR). Znamienkový bit všetkých priamych operandov sa nachádza na 31. bite inštrukcie pre urýchlenie znamienkového rozšírenia. Existujú ešte ďalšie dve varianty formátov inštrukcií s označením B a J založené na manipulácii s priamymi operandmi.[4]



Obrázok 1.1 Základné formáty inštrukcií RISC-V (prevzaté z [4])

Jediný rozdiel medzi formátom B a S je v automatickom doplnení nuly na najnižšiu bitovú pozíciu pri formáte B. Tento formát sa používa pre podmienené skoky. Umožňuje tak doskočiť ďalej a zjednodušiť kontrolu zarovnania adresy. Namiesto posunu všetkých bitov kódovaného priameho operandu doľava, zostávajú všetky bity okrem druhého najvýznamnejšieho na svojich miestach. Druhý najvýznamnejší bit sa nachádza na uvoľnenej pozícii nultého bitu. Týmto spôsobom zostáva maximálny počet

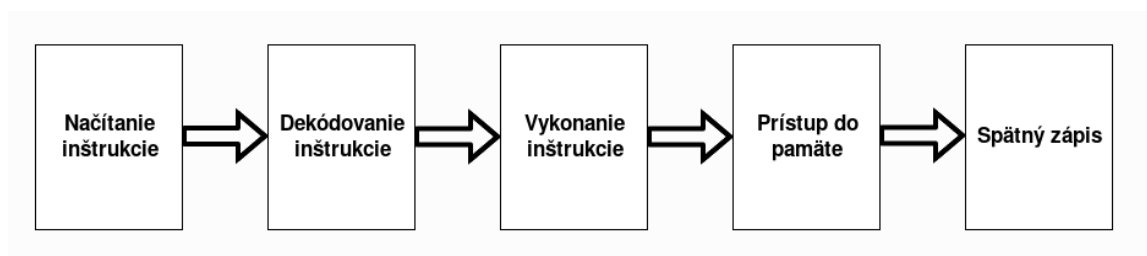
bitov na fixných pozíciách, čo opäť zjednodušuje dekódovanie. Podobne tomu je aj s formátmi U a J, kde U kóduje vrchné bity priameho operandu a J spodné.[4,5]

## 1.4 Codix Berkelium 5

Codix Berkelium 5 (Bk 5) predstavuje 32 bitový procesor patriaci do rodiny procesorov Codix Berkelium spoločnosti Codasip. Táto rodina procesorov je založená na otvorenej inštrukčnej sade RISC-V. Veľká výhoda oproti tradičným pevne konfigurovaným jadrám je jeho úplná konfigurovateľnosť a rozšíriteľnosť. Bk 5 disponuje nasledujúcimi parametrami :

- Inštrukčnú sadu RV32-IMC
- Podpora rozšírenia F (voliteľné)
- Paralelná násobička (voliteľné)
- Statická alebo dynamická predikcia skokov
- Podporovaný strojový (machine) privilegovaný mód
- Pamäť L1 cache a podpora pre cache operácie (voliteľné)
- IEEE-1149.1 JTAG Debug Logic s 4 až 8 hardwarovými zarážkami (breakpoints)
- Jednotka správy napájania (Power Management Unit) (voliteľné)
- Vnútorň radič prerušenia
- Null-pointer výnimka (voliteľné)
- AMBA 3 AHB-lite rozhranie [6]

Bk 5 je založený na spracovaní inštrukcii formou 5-stupňového zret'azenia v poradí (in-order execution) tak ako je znázornené na Obrázku 1.2. Skladá sa z nasledujúcich funkčných blokov: prediktor skokov, programový čítač, dekodér, pamäť programu, pamäť dát, aritmeticko-logická jednotka, paralelná násobička celočíselných hodnôt, modul pre násobenie a delenie celočíselných hodnôt, jednotka pre načítanie a ukladanie, registrové pole, jednotka pre spracovanie výsledku. Nasledujúce kapitoly popisujú komponenty, ktoré boli vybrané do verifikačného plánu.[6]

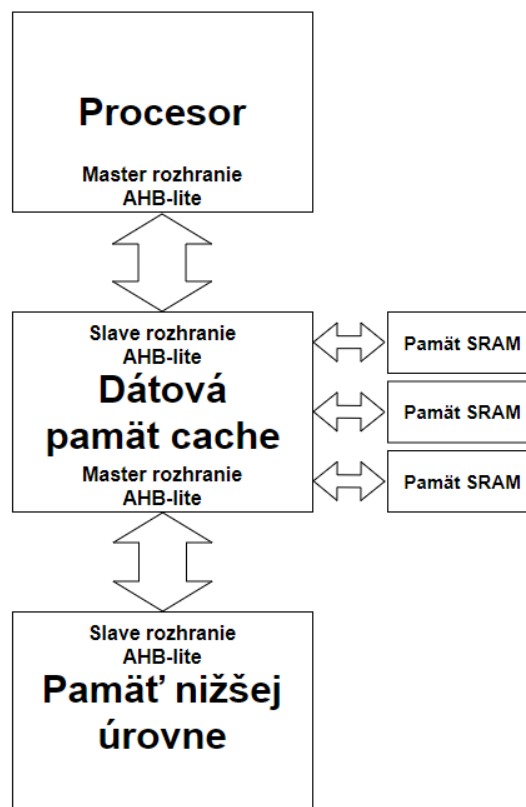


Obrázok 1.2 Spracovanie inštrukcii procesoru Bk 5 - 5-stupňové zret'azenie v poradí (prevzaté z [6])

### 1.4.1 Dátová pamäť L1 cache

Problémové miesto výkonu procesoru je rýchlostná priepasť medzi procesorom a pamäťou. Jednou z efektívnych a osvedčených techník vyrovnania tejto rýchlostnej priepasti je pridanie pamäte cache. V našom prípade sa jedná o primárnu pamäť cache označovanú ako L1. Tento typ cache pamäte býva integrovaný priamo na čipe procesoru a je realizovaný pomocou pamäte SRAM.[7,8]

Do verifikačného plánu bola zahrnutá dátová pamäť typu L1 cache, ktorá je navrhnutá ako synchronná. Dátová pamäť typu L1 cache má dve hlavné rozhrania typu AHB-lite (tento protokol bude podrobne popísaný v kapitole 3). Slave rozhranie, ktoré prijíma požiadavky od procesoru, a master rozhranie pripojené k pamäti nižšej úrovne viz. Obrázok 1.3. Okrem spomínaných dvoch hlavných rozhraní má dátová pamäť typu L1 cache ešte 3 rozhrania typu SRAM. Pomocou týchto rozhraní sú pripojené tri synchronne pamäte, ktoré dátová pamäť typu L1 cache využíva pre svoju potrebu. Tieto rozhrania nebudú v práci rozoberané, nakoľko nie sú súčasťou verifikačného plánu.[6]

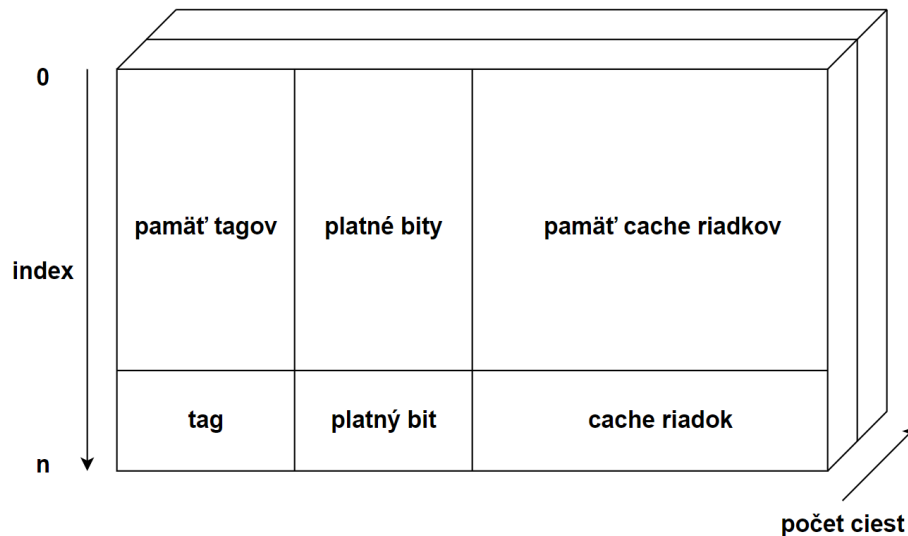


Obrázok 1.3 Zjednodušená schéma zapojenia dátovej pamäte typu L1 cache (upravené podľa [9])

Na konfiguráciu dátovej pamäte typu L1 cache môžeme použiť nasledujúce parametre:

- RST\_ACT\_LEVEL (aktívna úroveň resetovacieho signálu)
- RST\_SYNC (použitie asynchrónneho alebo synchrónneho resetu)
- CSU\_BASE\_ADDR (základná adresa kontrolnej stavovej jednotky)
- DCACHE\_IS\_LITTLE (endianita dátovej pamäte typu L1 cache)
- DCACHE\_SIZE (veľkosť dátovej pamäte typu L1 cache)
- DCACHE\_LINE\_SIZE (veľkosť cache riadku)
- DCACHE\_NUM\_WAYS (počet ciest)
- DCACHE\_ENABLE\_BURST (povolenie dávkových transakcií na rozhraní typu master)
- DCACHE\_MASTER\_ADDR\_BITS (adresná bitová šírka na rozhraní typu master)
- DCACHE\_MASTER\_DATA\_BITS (dátová bitová šírka na rozhraní typu master)
- DCACHE\_ADDR\_BITS (adresná bitová šírka na rozhraní typu slave)
- DCACHE\_DATA\_BITS (dátová bitová šírka na rozhraní typu slave)[9]

Dátová pamäť typu L1 cache obsahuje kontrolnú stavovú jednotku. V tejto jednotke sa nachádzajú pamäťovo mapované registre, ktoré sa používajú ako počítadlá a registre obsahujúce základné informácie. Dáta v dátovej pamäti typu L1 cache sú uložené v cache riadkoch, ktoré predstavujú najmenšie bloky dátových slov. Každému cache riadku je priradený tag a platný bit (valid bit). Tag reprezentuje časť adresy pre dáta uložené v hlavnej pamäti a platný bit rozhoduje o platnosti uloženého tagu. Riadok na ktorom je namapovaný pamäťový blok je označený časťou adresy nazývanou index. Dátová pamäť typu L1 cache je rozdelená do jednotlivých úsekov ktoré umožňujú aby bol pamäťový riadok namapovaný na rozličné cache riadky. Takáto cache pamäť sa označuje ako N-cestne asociatívna (N-way associative cache). Parameter N rozhoduje o počte miest, kde môžu byť dáta pre daný index (jeden cache riadok) uložené. Požadované dáta sú vyberané z pamäte na základe offsetu. Tento offset obsahuje časť fyzickej adresy a určuje adresu dát v cache riadku. Tzv. Dirty bit signalizuje ak došlo k zmene cache riadku od posledného načítania z hlavnej pamäte, ktorá je alokovaná do dátovej pamäte typu L1 cache. Organizácia dátovej pamäte typu L1 cache je znázornená na Obrázku 1.4.[6]



Obrázok 1.4 Organizácia dátovej pamäti typu L1 cache[6]

Pri prehľadávaní dát umiestnených v dátovej pamäti typu L1 cache sa porovnávajú uložené tagy s tagom získaným z fyzickej adresy. Tagy sú uložené v pamäti tagov na indexe vypočítaného z fyzickej pamäti. V prípade, ak sa hodnota tagu získaného z fyzickej adresy zhoduje s tagom umiestneným v nejakej z ciest v dátovej pamäti typu L1 cache a zároveň je nastavený platný bit, potom hovoríme o tzv. cache hit a dáta z príslušného cache riadku sú prístupné. Cache riadok je umiestnený na vypočítanom indexe a na rovnakej ceste v ktorej bol nájdený tag. V opačnom prípade hovoríme o tzv. cache miss a dáta musia byť získavané z pamäte nižšej úrovne.[6]

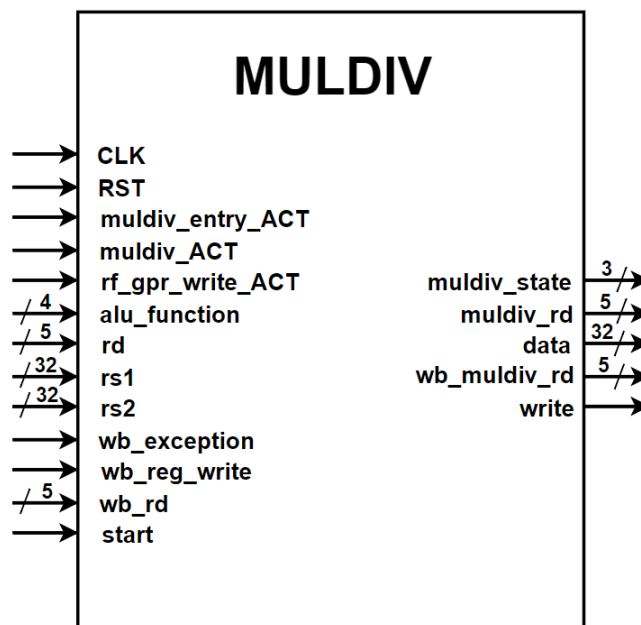
## 1.4.2 Modul pre násobenie a delenie celočíselných hodnôt

Modul pre sériové násobenie a delenie celočíselných 32-bitových hodnôt je označovaný v implementácii ako MULDIV. Tento modul je pre obe operácie spoločný. Všetky operácie, ktoré umožňuje implementovaný modul pre násobenie a delenie sú kvôli prehľadnosti popísané v prílohe v kapitole B.1. Prehľad jednotlivých vstupov a výstupov modulu je znázornený na Obrázku 1.5. Vstupy a výstupy modulu pre násobenie a delenie celočíselných hodnôt sú :

- *muldiv\_entry\_ACT*, *muldiv\_ACT*, *rf\_gpr\_write\_ACT* – aktivačné signály informujúce modul že procesor je v prevádzke.
- *rs1* - prvý vstupný operand operácie.
- *rs2* - druhý vstupný operand operácie.
- *rd* - vstup určujúci cieľový register výpočtu.
- *alu\_function* - operácia ktorá sa má previesť v module.
- *start* - modul môže začať počítať.
- *wb\_reg\_write* - procesor nie je pripravený prijať vypočítanú hodnotu.
- *wb\_exception* - procesor nie je pripravený prijať vypočítanú hodnotu lebo spracováva výnimku.

- *wb\_rd* - adresa registra do ktorého momentálne procesor očakáva zápis.
- *wb\_muldiv\_rd* - adresa registra do ktorého sa snaží zapísať modul výsledok. Súčasťou úspešného zápisu výsledku do registra je zhoda signálov *wb\_rd* a *wb\_muldiv\_rd*.
- *muldiv\_rd* - adresa registra do ktorého sa snaží zapísať modul výsledok. S rozdielom oproti signálu *wb\_muldiv\_rd*, že je pripojený do inej časti procesoru.
- *data* - výsledok výpočtu modulu.
- *write* - modul je pripravená predať výsledok výpočtu procesoru.
- *muldiv\_state* - informácia v ktorom stave sa nachádza modul.[10]

Popis jednotlivých stavov modulu pre násobenie a delenie celočíselných hodnôt nebude v práci rozoberaný nakoľko sa pri verifikácii nevyužívajú s výnimkou stavu ready. V stave ready sa nachádza modul v prípade že je pripravený na prevedenie operácie násobenia alebo delenia. Je signalizovaný hodnotou 0.

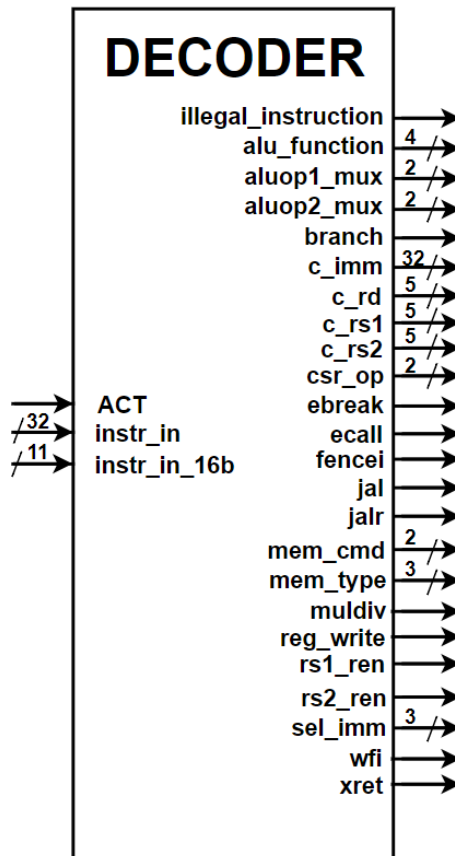


Obrázok 1.5 Schéma muldiv modulu

### 1.4.3 Dekodér

Dekodér implementovaný v procesore Codix Berkelium 5 je založený výlučne na kombinačnej logika. Dekodér disponuje troma vstupmi *ACT*, *instr\_in* a *instr\_in\_16b*. Vstup *ACT* je aktivačný signál a informuje dekodér že procesor je v prevádzke. 32-bitový signál *instr\_in* tvorí vstup pre 32-bitový a 16-bitový formát inštrukcií. Do vstupného 11-bitového signálu *instr\_in\_16b* sa privádza 2. až 12. bit komprimovaných inštrukcií. Pomocou tohoto signálu sa dekodujú zdrojové registre, cieľový register alebo priamy operand z komprimovanej inštrukcie. Popis výstupných dekódovaných riadiacich signálov využívaných pri verifikácii sa nachádza kvôli prehľadnosti v prílohe v kapitole A. Prehľad všetkých vstupov a výstupov dekodéru je

znázornený na Obrázku 1.6.[10]



Obrázok 1.6 Schéma dekodéru

Výstupné dekódované riadiace signály sú:

- *alu\_function* - signál vyberajúci operáciu použitú v aritmeticko-logickej jednotke alebo v module pre násobenie a delenie. V prípade že inštrukcia nevyžaduje aritmeticko-logickú jednotku a ani modul pre násobenie a delenie nastaví sa do 0 čo predstavuje súčet.
- *csr\_op* - signál vyberajúci operáciu použitú s kontrolným a stavovým registrom.
- *aluop1\_mux* - kontrolný signál multiplexoru na vstupe prvého operandu aritmeticko-logickej jednotky.
- *aluop2\_mux* - kontrolný signál multiplexoru na vstupe druhého operandu aritmeticko-logickej jednotky.
- *rs1\_ren* - povolenie čítania z prvého zdrojového registra.
- *rs2\_ren* - povolenie čítania z druhého zdrojového registra.
- *sel\_imm* - výber dekódovania priameho operandu z 32-bitovej inštrukcie.
- *c\_rs1* - adresa prvého zdrojového registra z 16-bitovej inštrukcie.
- *c\_rs2* - adresa druhého zdrojového registra z 16-bitovej inštrukcie.

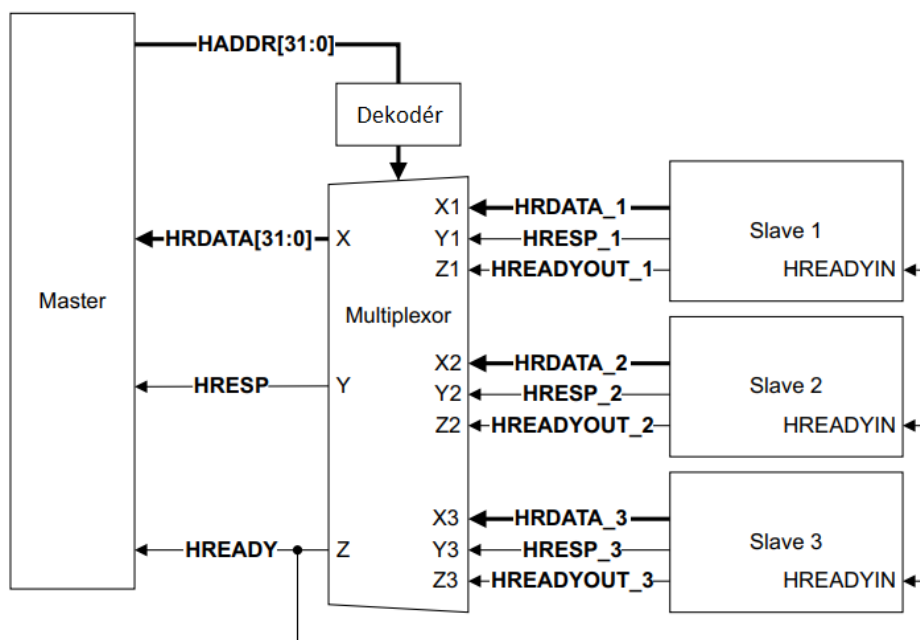
- *c\_rd* - adresa cieľového registra z 16-bitovej inštrukcie.
- *c\_imm* - výber dekódovania priameho operandu z 16-bitovej inštrukcie.
- *reg\_write* - inštrukcia zapisuje do registru.
- *mem\_type* - šírka prístupu do pamäti (bajt, polslovo, slovo). V prípade že inštrukcia nevyžaduje prístup do pamäti nastaví sa na hodnotu 0 čo predstavuje veľkosť bajt.
- *mem\_cmd* - typ prístupu do pamäti (čítanie, zápis).
- *muldiv* - inštrukcia pracuje s modulom pre násobenie a delenie.
- *ebreak* - inštrukcia vracajúca riadenie debuggovacímu prostrediu.
- *ecall* - inštrukcia volajúca funkciu operačného systému.
- *fencei* - inštrukcia synchronizácie inštrukcií zápisu do pamäti.
- *jal* - inštrukcia priameho skoku.
- *jalr* - inštrukcia nepriameho skoku.
- *branch* - inštrukcia podmieneného skoku.
- *wfi* - inštrukcia čakajúca na vznik prerušenia.
- *xret* - inštrukcia návratu z obsluhy výnimky.
- *illegal\_instruction* – nerozpoznanie inštrukcie.[10]

## 2 PROTOKOL AMBA 3 AHB-LITE

AHB-Lite (Advanced High-Performance Bus Lite) je odľahčená verzia vysoko výkonnej zbernice AHB spadajúca do otvoreného štandardu mikrokontrolerových zbernicových rozhraní AMBA 3 (Advanced Microcontroller Bus Architecture 3) spoločnosti ARM. Táto zbernica adresuje požiadavky vysoko výkonného syntetizovateľného návrhu. Podporuje jeden obvod typu master a poskytuje operácie s veľkou šírkou pásma. Najčastejším obvodom typu slave býva pri protokole AHB-Lite interná pamäť zariadenia, externá pamäť rozhrania a periférie s veľkou šírkou pásma.[11]

### 2.1 Obecná štruktúra

Obecná štruktúra rozhrania AHB-Lite je znázornená na Obrázku 2.1.



Obrázok 2.1 Príklad obcej štruktúry AHB-Lite s 3 obvodymi typu slave a 32-bitovou dátovou zbernicou (prevzaté z [11])

Protokol AHB-Lite definuje 32-bitovú adresnú zbernicu signálom  $HADDR [31:0]$ . Druh transakcie sa nastavuje pomocou signálu  $HBURST [2:0]$ . Existuje 8 druhov transakcií ktoré sú uvedené v štandarde v kapitole 3.5. Šírku prenášaného slova udáva signál  $HSIZE [2:0]$ . Všetky kombinácie širok prenášaných slov sú popísané v štandarde v kapitole 3.4. Signál  $HTRANS [1:0]$  udáva typ prenosu ktorý môže byť: IDLE, BUSY, NONSEQ, SEQ. Podrobný popis jednotlivých typov prenosu je znázornený v štandarde v kapitole 3.2. Šírka dátovej zbernice zápisu  $HWDATA$  a dátovej zbernice čítania  $HRDATA$  môže nadobudnúť hodnoty 8, 16, 32,

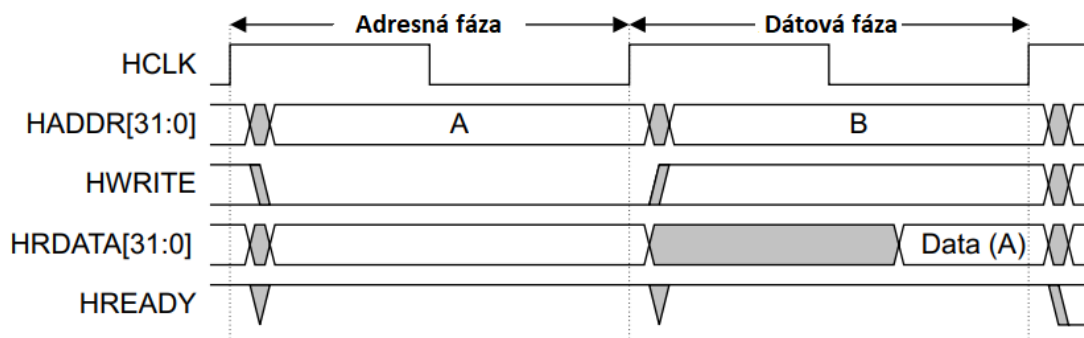
64, 128, 256, 512 alebo 1024 bitov. Smer prenosu určuje signál HWRITE ktorý v log. 1 indikuje zápis a v log. 0 čítanie. Prenosovú odpoveď udáva signál HRESP a ukončenie prenosu a začiatok nového prenosu signál HREADYOUT. Podrobný popis jednotlivých signálov sa nachádza v štandarde protokolu AHB-Lite.[11]

## 2.2 Prenos

Obvod typu master začína prenos nastavením adresy a kontrolných signálov. Tieto signály poskytujú informáciu o adrese, smere prenosu, šírke prenášaného slova, ochrane a prístupu k dátam, typu transakcie a indikujú zamknutú transakciu. Dátová zbernica zápisu posiela dáta od obvodu typu master smerom k obvodu typu slave a dátová zbernica čítania posiela dáta od obvodu typu slave smerom k obvodu typu master. Každý prenos pozostáva z 2 fáz :

- **Adresnej fázy** - jeden cyklus pre adresu a kontrolné signály
- **Dátovej fázy** - jeden alebo viac cyklov pre dáta[11]

Príklad zjednodušeného prenosu čítania bez čakacieho cyklu je znázornený na Obrázku 2.2.



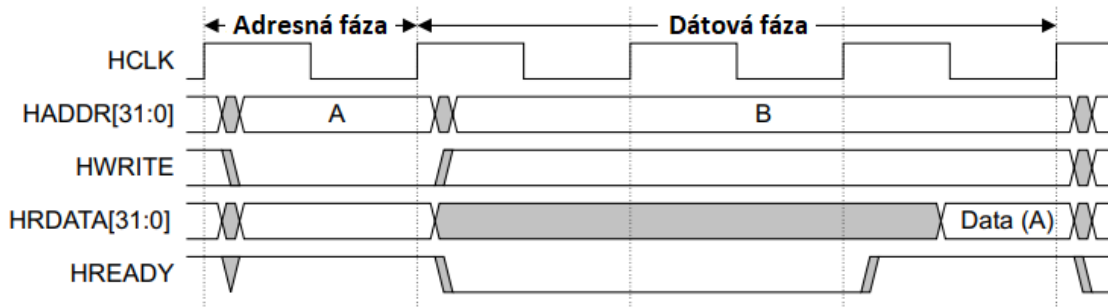
Obrázok 2.2 Príklad zjednodušeného prenosu čítania bez čakacieho cyklu (prevzaté z [11])

Zobrazený zjednodušený prenos čítania bez čakacieho cyklu pozostáva z 3 krokov:

1. Obvod typu master vystaví adresu HADDR a kontrolné signály na zbernicu, po náběžnej hrane hodinového signálu HCLK.
2. Obvod typu slave prijme adresu a kontrolné signály počas nasledujúcej náběžnej hrany signálu HCLK.
3. Po prijatí adresy a kontrolných informácií o prenose obvodom typu slave, tento obvod môže vystaviť príslušnú odpoveď v podobe signálu HREADY a dát HRDATA. Odpoveď je následne prijatá obvodom typu master na tretej náběžnej hrane hodinového signálu HCLK.[11]

Adresná fáza akéhokoľvek prenosu nastáva počas dátovej fázy predchádzajúceho prenosu. Takéto prekryvanie adresnej a dátovej fázy je podstatou

zreťazenia zbernice umožňujúce vysoko výkonnú prevádzku, zatiaľ čo je poskytnutý dostatočný čas pre obvod typu slave aby poskytol odpoveď na prenos. Obvod typu slave môže vložiť do každej dátovej fázy prenosu čakací cyklus pomocou vystavenia signálu HREADY do log.0, aby mal dostatok času na dokončenie. Rozšírenie dátovej fázy prenosu má vedľajší efekt rozšírenia adresnej fázy nasledujúceho prenosu tak ako zobrazuje Obrázku 2.3. [11]



Obrázok 2.3 Príklad prenosu čítania s dvoma čakajúcimi cyklami (prevzaté z [11])

## 2.2.1 Druhy prenosov

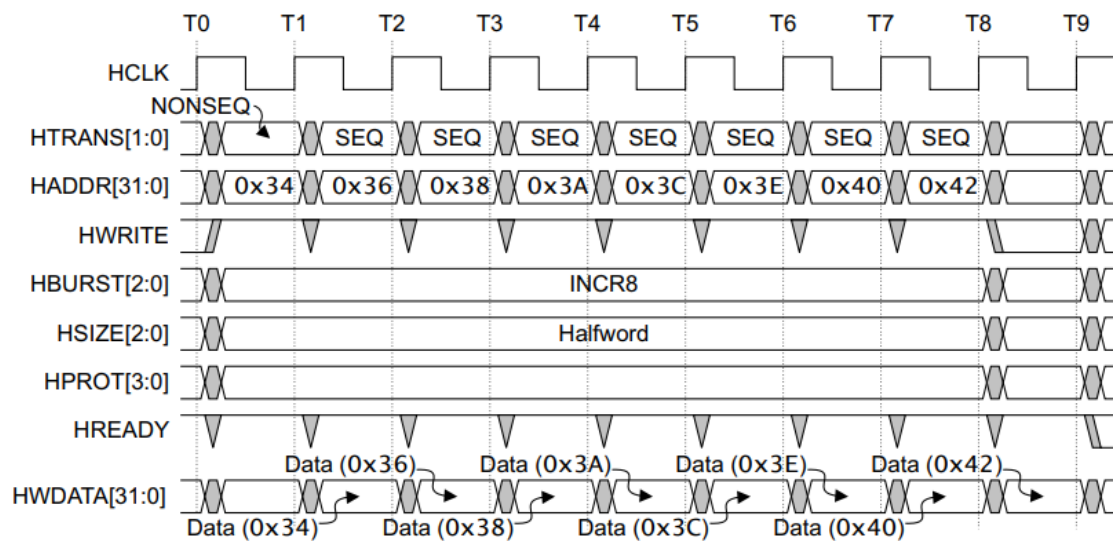
Prenosy môžu byť klasifikované do 4 rôznych typov, ktoré určuje signál HTRANS [1:0] :

- **IDLE** (0b00) Obvod typu master vystaví signál HTRANS na hodnotu IDLE v prípade, že nie je potrebné prenášať žiadne dáta. Obvod typu slave musí vždy vystaviť odpoveď HTRANS na hodnotu OKAY na takýto typ prenosu bez vloženia čakacieho cyklu.
- **BUSY** (0b01) Tento typ prenosu nastavuje obvod typu master v prípade, ak pokračuje v dávke, ale ďalší prenos momentálne nemôže zahájiť. Výnimkou je inkrementujúca dávka o nedefinovanej dĺžke, pri ktorej môže byť tento prenos ako posledný cyklus v dávke s informáciou o ukončení dávky. Adresa a kontrolné signály musia počas tohoto prenosu niesť informácie o ďalšom prenose. Podobne ako pri prenose IDLE musí obvod typu slave vystaviť odpoveď HTRANS na hodnotu OKAY na takýto typ prenosu bez vloženia čakacieho cyklu.
- **NONSEQUENTIAL** (0b10) Indikuje transakciu o jednom prenose, alebo prvý prenos v dávke. Adresa a kontrolné signály nesúvisia s predchádzajúcim prenosom.
- **SEQUENTIAL** (0b11) Indikuje zvyšné prenosy v dávke, pričom adresa súvisí s predchádzajúcim prenosom a kontrolné signály sú identické s predchádzajúcim prenosom.[11]

## 2.2.2 Druhy transakcií

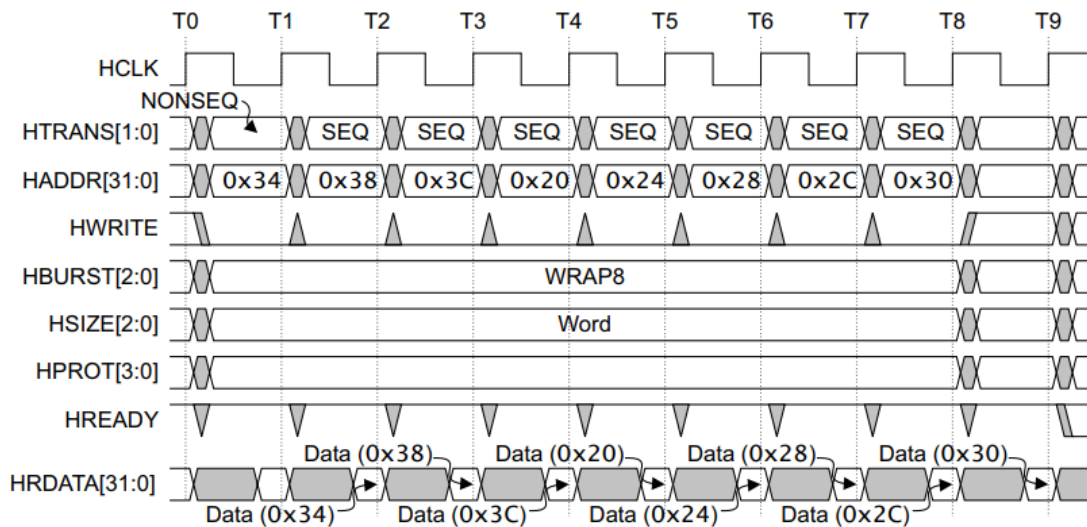
AHB-lite definuje nasledujúce transakcie: inkrementujúcu dávku v 4, 8 alebo 16 prenosoch, zaobalenú dávku v 4, 8 alebo 16 prenosoch, inkrementujúcu dávku nedefinovanej dĺžky a transakciu o jednom prenose. Každý prenos musí mať zarovnanú adresu na základe šírky prenášaných dát.[11]

**Inkrementujúca dávka (INCR):** Nasledujúca adresa prenosu v dávke je inkrement predchádzajúcej adresy. Veľkosť inkrementu je daná signálom HSIZE. Obvod typu master nemôže nikdy začať prenos, ktorého inkrementujúca dávka presahuje adresný priestor jeden kilobajt. Príklad inkrementujúcej dávky o 8 prenosoch zápisu a šírkou prenášaného slova 2 bajty (HSIZE = 0b001) je znázornený na Obrázku 2.4. [11]



Obrázok 2.4 Príklad inkrementujúcej dávky o 8 prenosoch zápisu a šírkou prenášaného slova 2 bajty (prevzaté z [11])

**Zaobalená dávka (WRAP):** Pre celú dávku sa vytvorí adresný priestor vypočítaný ako súčin počtu prenosov v dávke a šírky jedného prenosu. Adresa prenosu môže takto vytvoreným adresným priestorom pretečť. Príklad zaobalenej dávky o 8 prenosoch čítania a šírkou prenášaného slova 4 bajty (HSIZE = 0b010) je znázornený na Obrázku 2.5. V znázornenom príklade je zobrazené pretečenie vytvoreného adresného priestoru o veľkosti 32 bajtov, v momente keď po prenose z adresy 0x3C nasleduje prenos s adresou 0x20.[11]



Obrázok 2.5 Príklad zaobalenej dávky o 8 prenosoch čítania a šírkou prenášaného slova 4 bajty (prevzaté z [11])

### 2.2.3 Odpoveď prenosu

Po zahájení prenosu obvodom typu master, obvod typu slave riadi ako bude prenos pokračovať. Obvod typu master nemôže po zahájení prenosu tento prenos zrušiť bez príslušnej odpovede od obvodu typu slave. Odpoveď indikujúca stav prenosu je poskytovaná obvodom typu slave pomocou signálu HRESP:

- **OKAY** (0b0) Prenos bol úspešne dokončený alebo obvod typu slave vyžaduje ďalšie cykly pre dokončenie žiadosti. Signál HREADY indikuje či bol prenos dokončený alebo čaká.
- **ERROR** (0b1) Počas prenosu došlo k chybe. Chybový stav vyžaduje odpoveď v 2 cykloch, s nastavením signálu HREADY do log.1 v druhom cykle.[11]

Úplná prenosová odpoveď závisí od kombinácií signálov HRESP a HREADY zobrazených v Tabuľke 2.1[11]

Tabuľka 2.1 Úplná prenosová odpoveď (prevzaté z [11])

HRESP	HREADY	
	0b0	0b1
0b0	Čakajúci prenos	Úspešne ukončený prenos
0b1	Prvý cyklus chybnej odpovede	Druhý cyklus chybnej odpovede

## 3 FORMÁLNA VERIFIKÁCIA HARDWARU

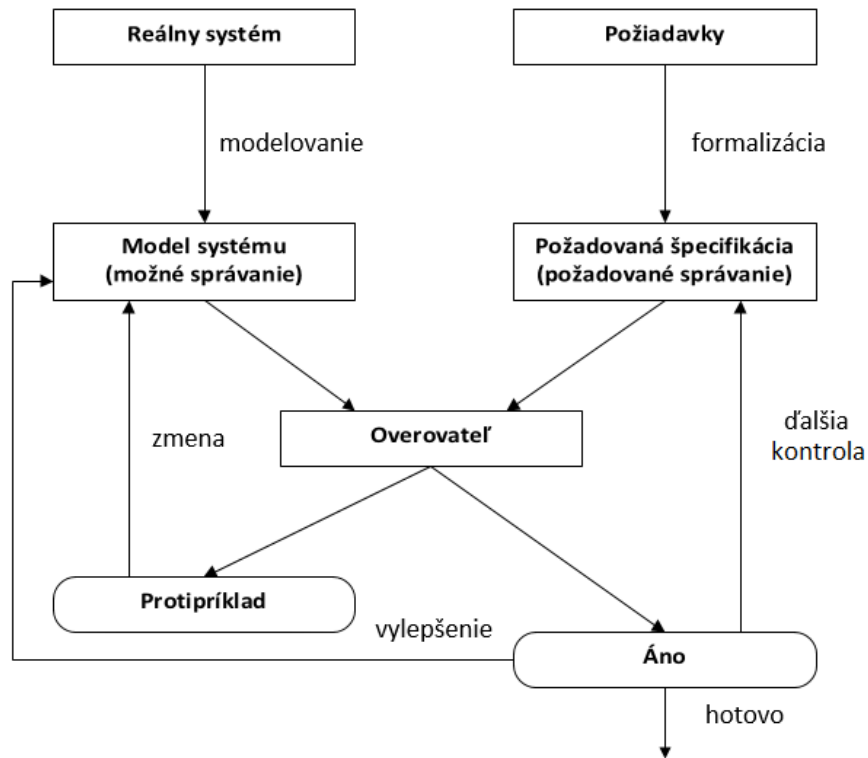
Verifikácia je neoddeliteľnou súčasťou procesu navrhovania hardwarových systémov. Jej cieľom je overenie funkcionality a odhaľovanie možných chýb v navrhovanom hardwarovom produkte. Verifikačné tímy používajú pre tento účel niekoľko techník. V priemysle je často používanou metódou simulácia, najmä kvôli jej ľahkej adaptácii a poskytovaniu dobrej predstavy o správaní systému. S narastajúcou veľkosťou hardwarového návrhu však vzrastá aj čas simulácie, preto verifikačné tímy hľadajú spôsob ako znížiť počet vektorov potrebných na overenie systému pri zachovaní prijateľnej miery pokrytia. Formálna verifikácia je potenciálne veľmi rýchla oproti simulácii, pretože nepotrebuje vyhodnotiť každý možný stav, aby dokázala že daná logika spĺňa skupinu vlastností za všetkých podmienok. Avšak výkon formálnej verifikácie vo veľkej miere závisí na type logiky na akú sa uplatňuje a od spôsobu akým sa aplikuje. Formálne techniky, ktoré dokážu overiť že návrh je v súlade s jeho špecifikáciou alebo normou, sú skvelým doplnkom verifikačného postupu. Nástroje využívajúce formálne techniky sú založené na matematických metodikách a dokážu zaručiť správnosť jednotlivých častí architektúry. Formálna verifikácia poskytuje metódy a techniky ktoré matematicky dokazujú správnosť systému, ako napríklad overovanie modelov (model checking), dokazovanie viet (theorem proving) a overovanie ekvivalenciou (equivalence checking).[12,13]

### 3.1 Overovanie modelu

Na začiatku 80-tych rokov minulého storočia Clarke a Emerson navrhli kontrolu modelu (tzv. model checking), metódu pre automatickú a algoritmickú verifikáciu súbežných systémov s konečným počtom stavov. Nezávisle na tom navrhli Quielle a Sifakis v podstate tu istú metódu. Kontrola modelu je automatizovaná technika, ktorá pre daný model s konečným počtom stavov nejakého systému a formálnou vlastnosťou, systematicky kontroluje, či je táto vlastnosť dodržaná pre daný stav v kontrolovanom modeli. Pri kontrole modelu sa temporálna logika (tzv. temporal logic TL) používa na určenie správneho správania sa systému. Na nájdenie správnych časových vzorcov sa používa efektívny a flexibilný postup vyhľadávania v grafe súbežného systému s konečným počtom stavov.[14,15]

Overovanie modelu je v súčasnosti jednou z najpoužívanejších a najznámejších formálnych techník. Jadrom kontroly modelu je jeho schopnosť automaticky overiť vlastnosti systému s konečným počtom stavov pre všetky možné systémové správania. Vlastnosti, ktoré sa majú preskúmať musia byť presne a jednoznačne definované. Kontrola modelu je schopná zistiť protipríklady, preto je vhodná aj na odhaľovanie a opravu chýb v prípade že daný model nespĺňa špecifikovanú požiadavku. Aj v prípade keď je požadované správanie systému uspokojené, je možné model vylepšiť a znovu použiť kontrolu modelu. Obrázok 3.1 znázorňuje všeobecný príklad kontroly modelu a

zahŕňa všetky kroky, ktoré táto technika sprevádza.[16]



Obrázok 3.1 Verifikačná metodológia kontroly modelu (prevzaté z [16])

Vlastnosti, ktoré sa majú skúmať, môžu byť špecifikované pomocou logického výpočtového stromu (tzv. computational tree logic CTL). CTL je špecifikačný jazyk pre systémy s konečným počtom stavov, ktorý je schopný uvažovať o slede udalostí. Problém s kontrolou modelu sa znižuje na kontrolu toho, že pre daný model  $M$  je počiatkový stav  $s \in S$ , kde  $S$  je množina všetkých stavov modelu a CTL vzorec  $\varphi$ ,  $M, s \models \varphi$  je potvrdený.[16]

Na začiatku 90-tych rokov sa začalo uvažovať o priemyselnej uplatniteľnosti kontroly modelu. Toto podnietili že v roku 1998 technická komisia pre formálne overovanie v organizácii Accellera sa začala usilovať o štandardizáciu časovej logiky pre použitie v hardwarovom priemysle. Požiadavky boli zhromaždené od priemyselných používateľov kontroly modelu v hardware, ako napríklad aké veci je potrebné vyjadriť ľahšie ako mohli v existujúcej temporálnej logike. Toto úsilie nakoniec vyústilo do dvoch popredných IEEE štandardov, PSL (Property Specification Language, IEEE Standard 1850) a SVA (SystemVerilog Assertions, IEEE Standard 1800). PSL aj SVA sú založené na lineárnej paradigme (súbor princípov, na ktorých je založený programovací jazyk) a najmä na temporálnej logike. Poskytujú tiež lokálne premenné, ktoré možno považovať ako mechanizmus pre deklarovanie kvantifikovaných premenných a obmedzovanie (constrain) ich správania.[15]

### 3.1.1 SystemVerilog Assertions

Formálne tvrdenia jazyka SystemVerilog (SystemVerilog Assertions) sa primárne používajú na overenie správania návrhu. Môžu sa však použiť na poskytnutie informácií o funkčnom pokrytí návrhu a označenie vstupných stimulov ktoré nespĺňajú predpokladané požiadavky. Takéto tvrdenia môžu byť kontrolované dynamicky simuláciou alebo staticky pomocou samostatného nástroja na kontrolu vlastností tzv. formálneho verifikačného nástroja, ktorý dokazuje či daný návrh spĺňa danú špecifikáciu. Formálne tvrdenie sú spojené s príkazmi ktoré udávajú, ktorá verifikačná funkcia má byť vykonaná. Jedná sa o nasledujúce príkazy :

- **Assert** – špecifikuje vlastnosť ako pravidlo pre návrh, ktoré má byť kontrolované aby sa overila platnosť tejto vlastnosti.
- **Assume** – špecifikuje vlastnosť ako predpoklad pre prostredie. Simulátory kontrolujú či táto vlastnosť platí, zatiaľ čo formálne nástroje používajú túto vlastnosť ako informáciu pre generovanie vstupných stimulov.
- **Cover** – sleduje vyhodnotenie vlastnosti pre pokrytie.[17,18]

Existujú dva základné druhy formálnych tvrdení a to okamžité a súbežné.[18]

### 3.1.2 Okamžité formálne tvrdení

Okamžité (immediate) formálne tvrdenie je test výrazu vykonaného v okamžiku, keď sa vykoná príkaz v procedurálnom bloku. Výraz nie je temporálny a je interpretovaný rovnako ako výraz podmienky procedurálneho príkazu if. Inými slovami, ak sa výraz vyhodnotí na X, Z alebo 0 je interpretovaný ako nepravdivý a príkaz formálneho tvrdenia sa považuje za neúspešný (false). V opačnom prípade sa výraz interpretuje za pravdivý a príkaz formálneho tvrdenia za úspešný (pass). Existujú dva druhy okamžitých formálnych tvrdení.

- **Jednoduché okamžité** (simple immediate) - akcie, ktoré sa vykonávajú pri neplatnosti alebo platnosti formálneho tvrdenia sú vykonávané okamžite po vyhodnotení tvrdenia.
- **Oneskorené okamžité** (deferred immediate) - akcie, ktoré sa vykonávajú pri neplatnosti alebo platnosti formálneho tvrdenia sú oneskorené až do neskoršieho časového obdobia, čím je zaručená určitá úroveň ochrany proti neúmyselným viacnásobným spusteniam pri prechodných hodnotách alebo zákmite (glitch).[18]

### 3.1.3 Súbežné formálne tvrdení

Súbežné (concurrent) formálne tvrdenia popisujú správanie, ktoré je spojené s časom. Na rozdiel od okamžitých tvrdení je vyhodnocovací model založený na hodinách, teda vyhodnocovanie formálnych tvrdení sa uskutočňuje iba pri prítomnosti hodinového signálu. Za účelom overenia správneho správania systému a prispôsobenia sa čo najbližšie sémantike založenej na cykloch, by mala byť hodinová udalosť (clock event) bez zákmitov a s jedným prechodom v ktoromkoľvek časovom kroku. Ak sa hodinová udalosť v priebehu časového kroku zmení viackrát, výsledné správanie nie je definované. Súbežné formálne tvrdenia používajú vzorkované hodnoty výrazov.[17,18]

Booleovské výrazy sú základné stavebné bloky súbežných formálnych tvrdení. Sú vyhodnocované ako true (pravda) alebo false (nepravda). V súbežných formálnych tvrdeniach sa môžu vyskytovať na dvoch miestach:

- Vo výrazoch vlastností (properties) alebo sekvencií (sequences). V tomto prípade musia byť vyhodnocované v rámci vzorkovaných hodnôt všetkých premenných.
- Ako argument v deaktiváčnom príkaze formálneho tvrdenia `disable iff`. Hodnoty výrazov v deaktiváčnej podmienke sú vyhodnocované na základe aktuálnych hodnôt premenných.[18,19]

**Sekvencie** (sequences) poskytujú schopnosť tvoriť a manipulovať so sekvenčným správaním, definujú vzťahy medzi Booleovskými výrazmi v čase. Jednotlivé sekvencie môžu byť kombinované logicky alebo sekvenčne za účelom vytvorenia komplexnejšej sekvencie. Príklad jednoduchej sekvencie ktorá pri každej nábežnej hrane hodinového signálu *clk* kontroluje či je buď signál *a* alebo signál *b* v log.1 je uvedený na Obrázku. 3.2.[19,20]

```
sequence sekvencia_1;  
    @(posedge clk) a || b;  
endsequence
```

Obrázok 3.2 Príklad jednoduchej sekvencie

**Vlastnosti** (properties) zohrávajú kľúčovú úlohu pri formálnych tvrdeniach jazyka SystemVerilog, pretože tvoria telo súbežných formálnych tvrdení. Sú vyhodnocované ako true alebo false. Môžu byť vytvorené z ostatných vlastností, sekvencií a Booleovských výrazov. Vlastnosti definujú správanie návrhu. Aby bolo možné použiť toto správanie vo verifikácii, musia sa vlastnosti použiť spolu s vyššie spomínanými príkazmi `assert`, `assume`, `cover` alebo `restrict`. Výraz v príkaze `disable iff` sa nazýva deaktiváčna podmienka. Táto klauzula umožňuje špecifikáciu resetu.

V prípade, ak je táto deaktivovaná podmienka pravdivá kedykoľvek od začiatku vyhodnocovania vlastnosti, je celkové vyhodnocovanie výsledku tejto vlastnosti deaktivované. Takáto deaktivácia vyhodnocovania vlastnosti nemá vplyv na jeho výsledok.[19]

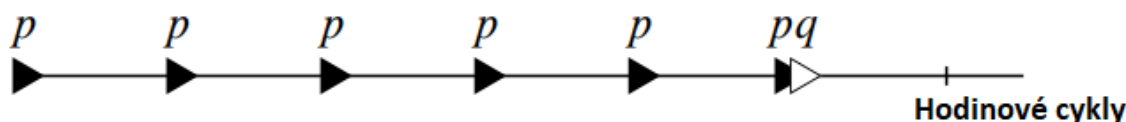
Jazyk SystemVerilog definuje súbor **operátorov** sekvencií a vlastností. Súbor týchto vlastností je znázornených v Tabuľke 3.1. od najvyššej priority po najnižšiu aj s príslušnou asociatívitou.[18]

Tabuľka 3.1 Súbor operátorov sekvencií a vlastností aj s príslušnou asociatívitou (prevzaté z [18])

Operátory sekvencií	Operátory vlastností	Asociatívitá
[*], [=], [->]		-
##		zľava
throughout		sprava
within		zľava
intersect		zľava
	not, nexttime, s_nexttime	-
and	and	zľava
or	or	zľava
	iff	sprava
	until, s_until, until_with, s_until_with, implies	sprava
	->,  =>, #-#, #=#	sprava
	always, s_always, eventually, s_eventually, if-else, case, accept_on, reject_on, sync_accept_on, sync_reject_on	-

Popis jednotlivých operátorov nebude v práci uvedený nakoľko sa vo verifikácii nevyužívajú s výnimkou operátora until\_with, nenásledného opakovania [=] a operátorov implikácie (|->, |=>).

Operátor until\_with patrí medzi prekrývajúce sa operátory vlastností. Uvažujme o vlastnosti  $p$  until\_with  $q$ . Táto vlastnosť nadobudne hodnotu true iba v prípade ak signál  $p$  bude v log. 1 v každom cykle hodinového signálu až po (a vrátane) prvý hodinový cyklus v ktorom signál  $q$  bude mať hodnotu log. 1 tak ako je zobrazené na Obrázku 3.3[19]



Obrázok 3.3 Príklad použitia vlastnosti until\_with (prevzaté z [19])

Operátor nenásledného opakovania (non-consecutive repetition) sa používa v prípade kontroly počtu pravdivých Booleovských výrazov, medzi ktorými môže byť oneskorenie jedného alebo viacerých hodinových cyklov.[19]

Operátor implikácie sa používa pri kontrole vlastnosti, ktorá je vykonávaná podmiennečne pri zhode sekvenčného predchodcu. Výsledkom implikácie je buď pravda alebo nepravda. Operand na ľavej strane sa nazýva predchodca (antecedent), zatiaľ čo pravý operand následník (consequent). Implikácia je pravdivá v prípade ak je pravdivý následník po dokončení jeho predchodcu. Existujú dve varianty operátoru implikácie:

- **Prekrývajúca sa implikácia (overlapping implication):**  $| \rightarrow$ . V prípade prekrývajúcej sa implikácie začína kontrola následníka v okamihu ako je splnený predchodca. Vlastnosť uvedená na Obrázku 3.4 sleduje či je signál  $a$  (predchodca) v log. 1 pri danej nábežnej hrane hodinového signálu  $clk$ . V prípade ak sa bude nachádzať v log. 1 začne vlastnosť kontrolovať, či je pri rovnakej nábežnej hrane hodinového signálu zároveň aj signál  $b$  (následník) v log. 1.

```
property priklad_1;
    @(posedge clk) a |-> b;
endproperty
```

Obrázok 3.4 Príklad použitia prekrývajúcej sa implikácie vo vlastnosti

- **Neprekrývajúca sa implikácia (nonoverlapping implication):**  $| \Rightarrow$ . V prípade splnenia predchodcu začína vyhodnocovanie nasledovníka v ďalšom hodinovom cykle. Vlastnosť uvedená na Obrázku 3.5 sleduje či je signál  $a$  (predchodca) v log. 1 pri danej nábežnej hrane hodinového signálu  $clk$ . V prípade ak sa bude nachádzať v log. 1 začne vlastnosť kontrolovať, či je pri ďalšej nábežnej hrane hodinového signálu hodnota signálu  $b$  (následník) log. 1.[21]

```
property priklad_2;
    @(posedge clk) a |=> b;
endproperty
```

Obrázok 3.5 Príklad použitia neprekrývajúcej sa implikácie vo vlastnosti

Vzorkovacie funkcie sú systémové funkcie jazyka SystemVerilog využívané pre prístup k vzorkovaným hodnotám výrazu. Tieto funkcie zahŕňujú schopnosť pristupovať k aktuálnej vzorkovanej hodnote, vzorkovanej hodnote v minulosti alebo zistiť zmeny vo vzorkovanej hodnote výrazu. Jedná sa o nasledujúce funkcie:

- **\$sampled**: táto funkcia vráti vzorkovanú hodnotu argumentu. Jediné zmysluplné využitie je v klauzule `disable iff`, pretože deaktivácia podmienka nie je štandardne vzorkovaná.
- **\$rose**: vráti `true` v prípade ak najmenej významný bit výrazu prejde do log. 1.
- **\$fell**: vráti `true` v prípade ak najmenej významný bit výrazu prejde do log. 0.
- **\$stable**: vráti `true` v prípade, ak je momentálna hodnota výrazu rovnaká ako hodnotou z predchádzajúceho hodinového cyklu.
- **\$changed**: vráti `true` v prípade, ak je momentálna hodnota výrazu odlišná od hodnoty z predchádzajúceho hodinového cyklu.
- **\$past**: funkcia na získavanie vzorkovaných hodnôt výrazu z minulých hodinových cyklov. Štandardne poskytuje hodnotu výrazu z predchádzajúceho hodinového cyklu.[18]

### 3.1.4 Verifikácia založená na formálnych tvrdeniach

Jedným zo spôsobov riešenia zvýšenej komplexnosti návrhu je doplnenie tradičných metód funkčnej verifikácie verifikáciou založenou na formálnych tvrdeniach (angl. Assertion-Based Verification, ABV). V súčasnosti sa verifikácia založená na formálnych tvrdeniach úspešne používa na viacerých úrovniach abstrakcie pri návrhu a verifikácii. Od vysokoúrovňových formálnych tvrdení nachádzajúcich sa v testbenchoch, ktoré sú na transakčnej úrovni, až po formálne tvrdenia na úrovni implementácie, syntetizovateľné do emulácie a hardwaru. S príchodom štandardizovaných jazykov založených na formálnych tvrdeniach a ich knižníc sa priemysel v poslednom čase stal svedkom zvýšeného záujmu o prijatie týchto techník.[22]

#### Questa PropCheck

Questa PropCheck je formálne založená aplikácia určená na analýzu vlastností návrhu v kontexte funkčnej verifikácie. Formálna analýza vykonáva statickú verifikáciu formálnych tvrdení návrhu. Vlastnosti (properties) formálnych tvrdení sú popísané pomocou SVA alebo PSL. Môžu to byť však aj inštancie kontrolovačov formálnych tvrdení (assertion checkers) nachádzajúcich sa v knižnici OVL (Open Verification Library). Na analýzu konkrétneho formálneho tvrdenia používa analyzátor niekoľko formálnych analyzujúcich strojov (engines), aby sa pokúsil nájsť riešenie. Questa PropCheck je preto schopný vyčerpávajúco odhaliť akékoľvek chyby návrhu, ktoré sa môžu vyskytnúť bez potreby špecifického stimulu na detekciu chyby.[23]

## 3.2 Dokazovanie viet

Dokazovanie viet sa vo veľkej miere spolieha na logiku vyššieho rádu a použitie matematických štruktúr, podľa ktorých sa vytvoria vzorce korešpondujúce so správaním systému. Overovacím procesom je vyhodnocovanie takto vytvorených vzorcov. Táto metóda sa môže uplatniť na projekty akejkoľvek komplexnosti, pretože nepracuje so stavmi ale so vzorcami. Vyžaduje to však vysoký stupeň znalosti obvodu určenému k verifikácii (DUV design under verification) a logiky pre používateľa na vykonanie verifikácie. Tento proces je ťažké automatizovať, pretože zahŕňa zložité vzorce. Používateľ musí vykonávať systematicky dokazovanie, vytvárať vzorce, vkladať ich do nástroja a analyzovať výsledky.[16]

### 3.2.1 Automatické dokazovanie viet

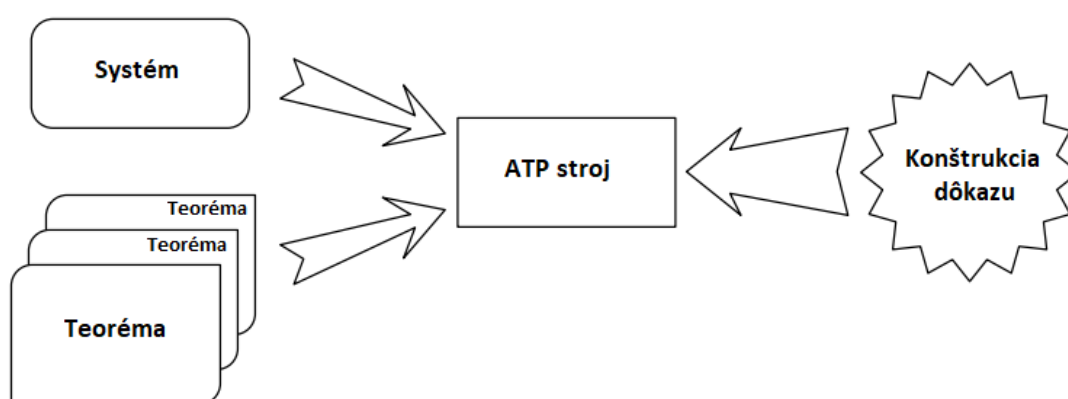
Jedná sa o proces zisťovania či dané tvrdenie (teoréma) je alebo nie je platné za určitých predpokladov (axiómach) pri určitej teórii použitím nástrojov ktoré pomáhajú urýchliť proces dokazovania. Stupeň automatizácie môže byť väčší alebo menší, v závislosti na použití formalizmu popisujúceho teorémy a model systému. Existujú rozličné formalizmy alebo formálne logiky používané na popis teorém viz. Tabuľke 3.2.[16]

Tabuľka 3.2 Popis formalizmov pri automatickom dokazovaní viet (prevzaté z [16])

Formalizmy	Stupeň automatizácie	Výraznosť	Rozhodnuteľnosť
Výroková logika	Vysoký	Malá	Rozhodnuteľná
Časová logika	Vysoký	Malá	Rozhodnuteľná
Predikovaná logika prvého rádu	Čiastočný	Stredná	Polorozhodnuteľná
Logika vyššieho rádu	Malý	Vysoká	Ani polorozhodnuteľná

V poradí s narastajúcou výraznosťou tu zaraďujeme výrokovú logiku, časovú logiku, predikátovú logiku prvého rádu a logiku vyššieho rádu. Spektrum pokryté vyššie uvedenými formalizmami siaha od atomických výrokov spojených spolu s výrokovými spojkami (výroková logika) až po kvantifikáciu predikátov a funkcií s predikátmi a funkciami pre argumenty (logika vyššieho rádu). Takéto zvyšovanie výraznosti má inverzný vplyv na automatizáciu. Čím výraznejšia je logika, tým viac klesá stupeň možnej automatizácie. Z toho vyplýva, že nástroje pre logiku vyššieho rádu zvyčajne pracujú iteratívne, kde je užívateľ zodpovedný za poskytnutie jednoduchších dôkazov ako pomôcok pri verifikačnom procese. Ďalšou charakteristikou z týchto formalizmov je rozhodnuteľnosť, s ktorou je tiež spojený stupeň výraznosti. Teória je rozhodnuteľná, ak procedúra môže určiť rozhodnuteľnosť. Okrem toho môže byť teória polorozhodnuteľná, ak pri pokuse o určenie jej rozhodnuteľnosti procedúra nikdy nevráti odpoveď, aj keď môže byť teória platná.[16]

Základná funkcionalita má dva kroky. Prvý krok sa dosiahne použitím adekvátneho formalizmu, ktorý je sám o sebe formálnym jazykom. Každý formalizmus má svoje vlastné axiómy predstavujúce východiskový bod pre overovateľa. Napriek tomu vlastné axiómy môžu pomôcť dokazovateľovi postupovať rýchlejšie s procesom rozlíšenia v ďalšom kroku. Druhý krok je vykonávaný buď samostatným dokazovateľom alebo iteratívne s vedením používateľa, v závislosti od úrovne automatizácie, ktorá je daná formalizmom. V oboch prípadoch má dokazovateľ k dispozícii niekoľko techník k slúžiacich na popis teorém a systému. Popis systémovej štruktúry nástrojov pre automatické dokazovanie viet (ATP Automated Theorem Proving) je zobrazený na Obrázku 3.6.[16]



Obrázok 3.6 Systémová štruktúra nástrojov pre ATP (prevzaté z [16])

Typickými predstaviteľmi nástrojov využívajúcich ATP sú Prototypový Verifikačný Systém (PVS Prototype Verification System), Isabell a B Metóda.[16]

### Prototypový Verifikačný Systém (PVS)

Prototypový verifikačný systém je interaktívny overovateľ pre logiku vyššieho rádu a poskytuje užívateľovi dva jazyky. Prvý určený na popis definícií a teorém a druhý na dokazovanie teorém. Na popis systému a teorém podľa [16], PVS ponúka základné typy ako Booleovský typ, celé čísla a reálne ako aj neinterpretované typy používateľov. Konštruktory zahrňujúce funkcie, sety, n-tice, záznamy, enumerované typy a induktívne definované abstraktné dátové typy. Špecifikácie sú usporiadané v parametrizovaných teóriách, ktoré môžu zahŕňať predpoklady, definície, axiómy a teorémy. Samotné parametre môžu zahŕňať konštanty, typy a inštalácie teórii. Poskytujú tiež obvyklé aritmetické a logické operátory, funkčné aplikácie, lambda abstrakciu a kvantifikátory pre výrazy. PVS poskytuje primitívne postupy dedukcie ktoré napríklad zahŕňajú výrokové a kvantifikačné pravidlá, indukciu, prepisovanie, zjednodušovanie pomocou rozhodovacích postupov na kontrolu rovnosti a lineárnych aritmetických údajov. Vyššie uvedené postupy dedukcie sú používané iteratívne pod vedením užívateľa.[16]

## **Isabell**

Isabell je generický iteračný asistent, ktorý dokáže pracovať s logikou vyššieho rádu. Isabell ponúka špecifikačné nástroje pre dátové typy, indukčné definície a rekurzívne funkcie s komplexným vzorom porovnávania. Dôkazy sú generované pomocou jazyka Isar, ktorý slúži ako človekom čitateľný výstup. Obsahuje tiež nástroje na urýchľovanie dokazovacieho procesu. Čo sa týka teórie, Isabell vlastní veľkú knižnicu formálne overenej matematiky, zahrnujúcu teóriu elementárnych čísel, analýzu, algebru a teóriu množín. Má tiež mnoho príkladov formálneho overovania, zahrňujúcich aplikáciu matematiky a informatiky.[16]

## 4 NÁVRH VERIFIKAČNÉHO PLÁNU

Verifikačné plánovanie je dôležitou a neoddeliteľnou súčasťou overovania bez ohľadu na veľkosť systému. Približne 70% času návrhového cyklu sa vynakladá na overovanie. Pri správne nastavenom verifikačnom pláne sa niektoré z problémov, ktoré by sa mohli vyskytnúť v neskorších fázach môžu riešiť skôr. Verifikačný plán možno opísať ako súbor cieľov, ktoré je potrebné overiť. Takéto plánovanie by malo byť nezávislé od použitých nástrojov. Poskytuje to flexibilitu pri rozhodovaní o správnom nástroji na dosiahnutie cieľov. Súčasťou verifikačného plánu je aj analýza špecifikácie. [24]

Zostavený verifikačný plán bol zameraný na procesor Codix Berkelium 5 firmy Codasip s využitím formálneho nástroja Questa PropCheck a použitím jazyka SystemVerilog assertions. Cieľom práce bolo zamerať sa na komponenty, ktoré je obtiažné overiť pomocou bežných techník využívajúcich simuláciu. Na základe toho boli vybrané 3 moduly: modul pre násobenie a delenie celočíselných hodnôt, dekodér a dátová pamäť typu L1 cache.

### 4.1 Dekodér

Pri dekodéri bolo naplánované zadefinovať všetky inštrukcie so základným 32-bitovým formátom ktoré sú implementované v procesore pri konfigurácii RV32IMC podľa špecifikácie RISC-V. Následne pomocou formálnych tvrdení kontrolovať požadované dekódované riadiace signály pri všetkých kombináciách jednotlivých inštrukcií. Zoznam týchto inštrukcií aj s ich požadovanými dekódovanými signálmi je uvedený v prílohe v kapitole B.2. Tieto požiadavky boli vytvorené na základe architektúry dekodéra a špecifikácie RISC-V.

### 4.2 Modul pre násobenie a delenie celočíselných hodnôt

Cieľom verifikačného plánu pri module pre násobenie a delenie celočíselných hodnôt je overenie správneho výsledku operácii násobenia, delenia a zvyšku po delení pri všetkých kombináciách dvoch 32-bitových vstupných operandov. Výnimkou sú kombinácie pri ktorých by došlo k deleniu nulou. Prehľad všetkých inštrukcií modulu pre násobenie a delenie celočíselných hodnôt je popísaný v prílohe v kapitole B.1.

### 4.3 Dátová pamäť cache typu L1

Verifikačný plán pri dátovej pamäti cache bol vytvorený za účelom kontroly dodržania pravidiel komunikácie protokolu AHB-lite. Kontrola týchto požiadaviek bola naplánovaná na rozhranie typu slave, pomocou ktorého je dátová pamäť typu L1 cache pripojená k procesoru. Rovnako aj na rozhranie typu master, pomocou ktorého je dátová pamäť typu L1 cache pripojená k pamäti nižšej úrovne. Všetky požiadavky boli vytvorené na základe špecifikácie protokolu AHB-lite. Typ transakcie a veľkosť šírky prenášaného slova na rozhraní typu master závisí od nastavenia parametrov DCACHE\_LINE\_SIZE a DCACHE\_MASTER\_DATA\_BITS. Na rozhraní typu master môže byť vždy iba jeden druh transakcie a šírky prenášaného slova. Prenos typu BUSY sa na tomto rozhraní nevyužíva. Na rozhraní typu slave typ transakcie nezávisí od parametrov a maximálna šírka prenášaného slova je daná parametrom DCACHE\_DATA\_BITS. Na základe týchto skutočností bola naplánovaná kontrola pravidiel protokolu AHB-lite pri 11 konfiguráciách uvedených v prílohe v kapitole C.1. Výber konfigurácií bol zameraný na použitie rozličných šírok prenášaného slova na rozhraní typu slave a master, rozličných typoch transakcií na rozhraní typu master a rozličných typoch resetu aj s ich aktivačnou úrovňou. Parametre ktoré pri jednotlivých konfiguráciách neboli zmenené sú nasledovné:

- CSU\_BASE\_ADDR - Adresa bola nastavená na hodnotu 0.
- DCACHE\_NUM\_WAY - Počet ciest bol nastavený na hodnotu 4.
- DCACHE\_SIZE - Veľkosť dátovej pamäte typu L1 cache bola nastavená na 16 kilobajtov.
- DCACHE\_IS\_LITTLE - Endianita dátovej pamäte typu L1 cache bola nastavená na malú.
- DCACHE\_ADDR\_BITS a DCACHE\_MASTER\_ADDR\_BITS - Adresné bitové šírky oboch rozhraní boli nastavené na hodnotu 32 bitov.

#### 4.3.1 Rozhranie typu slave

Pri rozhraní typu slave bola naplánovaná kontrola pozostávajúca z nasledujúcich požiadaviek:

- Správna šírka dátovej zbernice určenej na čítanie (HRDATA).
- Hodnota signálu HREADYOUT v log. 1 počas resetu.
- Správnej odpovede na typ prenosu IDLE a BUSY.
- Správne nastavenie úplnej chybnej prenosovej odpovedi.
- Vkladanie čakacích cyklov iba do dátovej fázy prenosu.

Prehľadný popis jednotlivých požiadaviek ktoré sa majú kontrolovať na rozhraní typu slave je uvedený v prílohe v kapitole C.2.

### 4.3.2 Rozhranie typu master

Pri rozhraní typu master bola naplánovaná kontrola pozostávajúca z nasledujúcich požiadaviek:

- Správna šírka dátovej zbernice určenej na zápis (HWDATA).
- Hodnota signálu HTRANS v IDLE počas resetu.
- Validná hodnota adresy a kontrolných signálov obvodu typu master počas resetu.
- Šírka prenášaného slova nesmie byť väčšia ako šírka dátovej zbernice.
- Hodnota signálu HTRANS iba v IDLE alebo NONSEQ po prenose typu IDLE alebo transakcii o jednom prenose.
- Podržanie hodnoty adresy a kontrolných signálov obvodu typu master počas adresnej fázy pokiaľ neodpovie obvod typu slave.
- Podržanie hodnoty dát určených na zápis počas dátovej fázy pokiaľ neodpovie obvod typu slave.
- Zarovnanie adresy pri všetkých druhoch prenosov podľa šírky prenášaného slova.
- Správny počet prenosov typu NONSEQ a SEQ pri zaobalenej dávke o 4, 8 a 16 prenosoch.
- Správne počítanie adresy pri všetkých kombináciách zaobalených dávok a širok prenášaného slova.

Prehľadný popis jednotlivých požiadaviek ktoré sa majú kontrolovať na rozhraní typu master je uvedený v prílohe v kapitole C.3.

## 5 PRIEBEH VERIFIKÁCIE A VYHODNOTENIE

V tejto kapitole je popísané nastavenie a spustenie nástroja pre verifikáciu, výsledky verifikácie a zhodnotenie prínosov formálnej verifikácie.

### 5.1 Nastavenie Questa PropCheck

Ako prvé bol vytvorený skript s názvom **run** obsahujúci 2 príkazy slúžiace na spustenie procesu formálnej verifikácie a zobrazenia výsledkov formálnej analýzy pomocou užívateľského rozhrania nástroja. Príklad takéhoto skriptu určeného na verifikáciu dátovej pamäte typu L1 cache aj s popisom príkazov je znázornený na Obrázku 5.1.

```
# Spustenie nástroja Questa Propcheck v konzolovom režime,  
# s výstupným adresárom s názvom log a pomocou spúšťacieho  
# skriptu run_dcachе.do.  
qverify -c -od log -do qft_files/run_dcachе.do  
  
# Načítanie databázy vygenerovanej pomocou formálnej analýzy  
# užívateľským rozhraním nástroja Questa PropCheck.  
qverify log/formal_verify.db
```

Obrázok 5.1 Súbor **run** určený na spustenie nástroja Questa PropCheck pri verifikácii dátovej pamäte typu L1 cache

V nasledujúcom kroku bol vytvorený spúšťací skript napísaný v skriptovacom jazyku tcl pre nástroj Questa Propcheck. Príklad spúšťacieho skriptu nástroja pre verifikáciu dátovej pamäte typu L1 cache aj s popisom príkazov je uvedený na Obrázku 5.2. Jednotlivé spúšťacie skripty sa pri rozličných moduloch návrhu od seba líšia. Tieto zmeny sú rozobraté v nasledujúcich podkapitolách.

```

# Vytvorenie pracovnej knižnice.
vlib work

# Kompilácia zdrojových súborov návrhu v jazyku VHDL podľa verzie
# VHDL-2008.
vcom -2008 -f qft_files/flist.vh

# Kompilácia zdrojových súborov návrhu v jazyku Verilog.
vlog -f qft_files/flist.vl

# Kompilácia súboru obsahujúceho modul v ktorom sú napísané
# formálne tvrdenia v jazyku SystemVerilog. Definovanie nástroja
# že tento modul je pripojený k návrhu pomocou direktívy bind.
vlog -sv -mfcu -cuname sva_bind ../src/properties/dcache_bind.sv

# V prípade výskytu chyby, ukončenie analýzy nástroja s chybou.
onerror {exit 1}

# Zadefinovanie hodinového signálu.
netlist clock CLK -period 20

# Kompilácia formálneho modelu návrhu a logiky
# vlastností (properties).
formal compile -d top_dcache -cuname sva_bind

# Spustenie formálnej analýzy nástroja s inicializačnou
# sekvenciou, s maximálnou dobou behu 2 hodiny, behom na jednom
# jadre procesora a vypnutím automatického obmedzovania signálov
formal verify -init qft_files/init.seq -timeout 2h -jobs 1 \
-auto_constraint_off

# Ukončenie s nulovým návratovým kódom.
exit 0

```

Obrázok 5.2 Spúšťací skript **run\_dcache.do** nástroja Questa PropCheck určený na verifikáciu dátovej pamäte typu L1 cache

Za účelom sprehľadnenia boli formálne tvrdenia oddelené od súborov s návrhom a to vytvorením samostatných súborov. Tieto samostatné súbory obsahujú moduly, v ktorých sa nachádzajú formálne tvrdenia a sú prepojené s modulmi návrhu pomocou direktívy **bind**. Všetky formálne tvrdenia boli vytvorené v jazyku SystemVerilog Assertion.

## 5.2 Výsledky verifikácie pomocou Questa PropCheck

Formálna verifikácia bola vykonaná na nasledovných komponentoch: dekodér, modul pre násobenie a delenie celočíselných hodnôt a dátová pamäť typu L1 cache.

### 5.2.1 Dekodér

Pri dekodéri bol upravený spúšťací skript **run\_decoder.do** nástroja Questa PropCheck. Definovanie hodinového signálu *CLK* spolu s inicializačnou sekvenciou bolo odstránené z dôvodu výlučne kombinačnej logiky dekodéru. Aktivačný signál *ACT* bol počas priebehu verifikácie obmedzený na hodnotu log. 1. Celkovo bolo vytvorených 56 oneskorených okamžitých formálnych tvrdení definovaných pomocou príkazu **assert final**. Pre každú inštrukciu so základným 32-bitovým formátom implementovanú v procesore pri konfigurácii RV32IMC bolo vytvorené jedno formálne tvrdenie. Tieto formálne tvrdenia museli byť umiestnené do procedurálneho bloku **always\_comb**. Jednotlivé formálne tvrdenia sú spustené

v prípade, že sa na vstupe dekodéru *instr\_in* objaví na definovanej pozícii operačný kód, ktorý patrí danej inštrukcii. Následne takto spustené formálne tvrdenie porovnáva hodnoty 20 výstupných dekodovaných riadiacich signálov s očakávanými hodnotami. Príklad vytvoreného oneskoreného okamžitého formálneho tvrdenia pre inštrukciu súčtu je zobrazený na Obrázku 5.3.

```

always_comb begin
  if (instr_in [6:0] == 'h33 && instr_in [14:12] == 'h0 && instr_in [31:25] == 'h0) begin
    add_instruction: assert final (
      illegal_instruction == 0      &&
      alu_function        == ALU_ADD &&
      aluop1_mux          == ALUOP1_RS1 &&
      aluop2_mux          == ALUOP2_RS2 &&
      branch              == 0      &&
      csr_op              == CSR_NONE &&
      ebreak              == 0      &&
      ecall               == 0      &&
      fencei             == 0      &&
      jal                 == 0      &&
      jalr                == 0      &&
      mem_cmd             == CMD_NONE &&
      mem_type            == MEM_BYTE &&
      muldiv              == 0      &&
      reg_write           == 1      &&
      rs1_ren             == 1      &&
      rs2_ren             == 1      &&
      sel_imm             == IMM_Z   &&
      wfi                 == 0      &&
      xret                == 0
    );
  end
end

```

Obrázok 5.3 Oneskorené okamžité formálne tvrdenie pre inštrukciu súčtu

Z celkového počtu 56 oneskorených okamžitých formálnych tvrdení sa podarilo úspešne overiť 54. Dve formálne tvrdenia boli neúspešné. Konkrétne sa jednalo o inštrukcie **fence.i** a **ecall**. Nakoľko tieto inštrukcie nepracujú s aritmeticko-logickou jednotkou procesora, očakávaná hodnota signálu *alu\_function* bola jeho implicitná čo predstavuje 0 ktorá je definovaná ako súčet (ADD). Skutočná hodnota signálu *alu\_function* bola 6 ktorá je definovaná ako logický súčet (OR). Tento rozdiel nepredstavoval funkčnú chybu v procesore. Vygenerované protipríklady týchto formálnych tvrdení sú uvedené v prílohe v kapitole D. Nástroj Questa PropCheck dokázal vyhodnotiť všetkých 56 oneskorených okamžitých formálnych tvrdení za 1 sekundu. Zhrnutie výsledku a štatistika procesu verifikácie dekodéru nachádzajúce sa vo vygenerovanom výstupnom súbore **formal\_verify.log** je zobrazené na Obrázku 5.4.

```

-----
Property Summary                               Count
-----
Assumed                                         0
Proven                                          54
Inconclusive                                   0
Fired                                           2
-----
Total                                           56
-----

----- Process Statistics -----
Elapsed Time                                   0 s
----- Orchestration Process -----
----- pcjavor:1599 -----
CPU Time                                       0 s
Peak Memory                                   0.2 GB
----- Engine Processes -----
----- pcjavor:1605 -----
CPU Time                                       1 s
Peak Memory                                   0.3 GB
CPU Utilization                               0 %
-----

Message Summary
-----
Count  Type      Message ID      Summary
-----
      1  Info    formal-210      Design is purely
combinational.

Final Process Statistics: Peak Memory 0.29GB, Cumulative CPU Time
1s, Elapsed Time 0s

```

Obrázok 5.4 Zhrnutie výsledku verifikácie dekodéru z výstupného súboru **formal\_verify.log**

## 5.2.2 Modul pre násobenie a delenie celočíselných hodnôt

Pri module pre násobenie a delenie celočíselných hodnôt bol upravený spúšťačí skript **run\_muldiv.do** nástroja Questa PropCheck. Inicializačná sekvencia pozostávala z piatich hodinových cyklov počas ktorých bol resetovací signál *RST* aktívny v log. 0 a ostatné vstupné signály mali hodnotu 0. Po inicializačnej sekvencii boli počas priebehu verifikácie obmedzené niektoré vstupné signály za účelom zmenšenia stavového priestoru modulu a neustáleho počítania modulu. Jedná sa o nasledujúce signály:

- *RST* na hodnotu log.1 v ktorej nie je aktívny.
- *muldiv\_entry\_ACT* na hodnotu log. 1.
- *muldiv\_ACT* na hodnotu log. 1.
- *rf\_gpr\_write\_ACT* na hodnotu log. 1.
- *start* na hodnotu log. 1.
- *wb\_exception* na hodnotu log. 0.
- *wb\_reg\_write* na hodnotu log. 0.
- *wb\_rd* a *rd* na adresu 1.

Celkovo bolo vytvorených 8 súbežných formálnych tvrdení. Pre každú operáciu modulu pre násobenie a delenie celočíselných hodnôt bolo vytvorené jedno súbežné formálne tvrdenie. Cieľom týchto súbežných formálnych tvrdení bolo overenie

správnosti počítania jednotlivých operácií modulu pri všetkých kombináciách 32-bitových vstupných operandov *rs1* a *rs2*. Výnimkou boli operácie delenia a zvyšku po delení, pri ktorých bol druhý operand *rs2* rovný 0. Príklad vlastnosti, z ktorej bolo vytvorené súbežné formálne tvrdenie pre operáciu *mulhu* je na Obrázku 5.5. Jednotlivé vlastnosti operácií sú spustené v prípade, ak sa modul nachádza v stave *ready*, ktorý signalizuje výstup *muldiv\_state* s hodnotou 0 a zároveň sa nachádza na vstupe *alu\_function* požadovaná operácia. V momente kedy je táto vlastnosť spustená sa dopredu vypočíta očakávaná hodnota výsledku operácie z operandov, ktoré sa momentálne nachádzajú na vstupe *rs1* a *rs2* pomocou operácie jazyka SystemVerilog. Takto vypočítaná hodnota očakávaného výsledku sa uloží do lokálnej premennej. Následne vlastnosť čaká 1 až 40 hodinových cyklov počas ktorých musí modul vyrátať výsledok operácie a vystaviť signál *write* do log. 1, čo signalizuje že modul je pripravený predať výsledok. Zároveň sa v tejto chvíli porovná vopred vypočítaný očakávaný výsledok operácie, ktorý je uložený v lokálnej premennej s výsledkom operácie modulu. Všetky vlastnosti majú v deaktivovanej podmienke pridaný resetovací signál *RST* v log. 0. V prípade operácií delenia a zvyšku po delení sa v ľavej časti operácie implikácie tzv. predchodcovi nachádza porovnanie druhého operandu *rs2* s 0. V prípade že sa druhý operand rovná 0 takáto vlastnosť ani nebude spustená.

```

property mulhu;
  bit [63:0] mulhu_result;
  @(posedge CLK) disable iff(~RST)
  (muldiv_state == 0 && alu_function == ALU_MULHU, mulhu_result = (rs1 * rs2)) |->
  ##[1:40] write == 1 && data == mulhu_result[63:32]
endproperty

```

Obrázok 5.5 Vlastnosť kontrolujúca správnosť počítania operácie *mulhu*

Na začiatku bola nastavená maximálna doba behu verifikácie 60 minút počas ktorých nástroj Questa PropCheck nedokázal vyhodnotiť jednotlivé súbežné formálne tvrdenia. Následne bola doba behu verifikácie zvýšená na 12 hodín. Ani po 12 hodinách nedokázal nástroj Questa PropCheck vyhodnotiť tieto súbežné formálne tvrdenia. Zhrnutie výsledku a štatistika procesu verifikácie modulu pre násobenie a delenie celočíselných hodnôt s dobou behu verifikácie 12 hodín je zobrazené na Obrázku 5.6.

```

-----
Property Summary                               Count
-----
Assumed                                         0
Proven                                          0
Inconclusive                                   8
  Analysis Incomplete                          8
Fired                                          0
-----
Total                                           8
-----

----- Process Statistics -----
Elapsed Time                                   43199 s
----- Orchestration Process -----
----- pcjavor:6698 -----
CPU Time                                       2 s
Peak Memory                                   0.3 GB
----- Engine Processes -----
----- pcjavor:6704 -----
CPU Time                                       43162 s
Peak Memory                                   5.8 GB
CPU Utilization                               99 %
-----

Message Summary
-----
-----
Count  Type      Message ID      Summary
-----
      1  Info      formal-205      Using initialization sequence
file to initialize design state.
      1  Info      formal-206      Timeout reached.

Final Process Statistics: Peak Memory 5.82GB, Cumulative CPU Time
43164s, Elapsed Time 43199s

```

Obrázok 5.6 Zhrnutie výsledku verifikácie modulu pre násobenie a delenie celočíselných hodnôt po 12 hodinách z výstupného súboru **formal\_verify.log**

Na základe tohto výsledku bol uskutočnený experiment, v ktorom bola nastavená maximálna doba behu verifikácie 2 hodiny a oba operandy mali pevne definovanú hodnotu. Hodnota operandu *rs1* bola 0x84444444 a hodnota *rs2* bola 0x8fffffff. Toto obmedzenie hodnôt oboch operandov bolo vykonané pomocou príkazu **assume**, tak ako je uvedené na Obrázku 5.7.

```
operands_assumption: assume property (@(posedge CLK) (rs1 == 'h84444444 && rs2 == 'h8fffffff));
```

Obrázok 5.7 Obmedzenie hodnôt vstupných operandov *rs1* a *rs2* pomocou príkazu **assume**

Tento experiment dokázal úspešne overiť všetkých 8 súbežných formálnych tvrdení pri obmedzení oboch vstupných operandov na pevne definovanú hodnotu tak ako je zobrazené na Obrázku 5.8. Počas vyhodnocovania jednotlivých súbežných formálnych tvrdení mal nástroj Questa PropCheck problém s ich zložitou, ktorá sa odzrkadlila na čase za aký ich vyhodnotil a označením zdravia analyzujúceho stroja (engine). Zdravie analyzujúceho stroja je označené farbou hviezdíčky. Zelená farba signalizuje že nástroj nemal s vyhodnotením daného formálneho tvrdenia žiaden

problém a žltá a červená farba signalizuje že počas vyhodnocovania narazil na ťažkosti.[25]

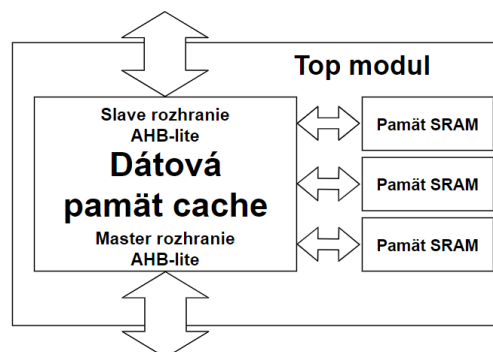
Name	Health	Radius	Time
bind_muldiv.mul_assertion	★ 10		1h 36m 37s
bind_muldiv.mulh_assertion	★ 10		31m 6s
bind_muldiv.mulhu_assertion	★ 10		12m 44s
bind_muldiv.mulhsu_assertion	★ 10		31m 51s
bind_muldiv.div_assertion	★ 10		13m 23s
bind_muldiv.divu_assertion	★ 10		6m 57s
bind_muldiv.rem_assertion	★ 10		21m 55s
bind_muldiv.rem_u_assertion	★ 10		13m 53s
bind_muldiv.operands_assumption			

Obrázok 5.8 Zobrazenie výsledku verifikácie modulu pre násobenie a delenie celočíselných hodnôt pri obmedzení vstupných operandov užívateľským rozhraním nástroja

Z uvedených výsledkov je zrejmé, že nástroj Questa PropCheck nedokáže overiť súbežné formálne tvrdenia, ktoré by potvrdili správnosť počítania modulu pre násobenie a delenie celočíselných hodnôt pri všetkých kombináciách vstupných operandov vzhľadom na zložitosť týchto súbežných formálnych tvrdení.

### 5.2.3 Dátová pamäť typu L1 cache

Prvý krok pozostával z pripojenia dátovej pamäte typu L1 cache k 3 synchronným pamätiam pomocou rozhrania SRAM a následného zapuzdrenia do Top modulu tak, aby sa na tomto module nachádzalo iba rozhranie AHB-lite typu master a slave. Toto zapuzdrenie bolo vykonané z dôvodu správneho fungovania dátovej pamäte typu L1 cache. V opačnom prípade by nástroj Questa PropCheck generoval náhodne hodnoty vstupných signálov rozhrania SRAM. Zjednodušené zapojenie je znázornené na Obrázku 5.9.



Obrázok 5.9 Zapuzdrenie dátovej pamäte typu L1 cache s pripojenými pamätami SRAM do Top modulu

Inicializačná sekvencia pozostávala z piatich hodinových cyklov, počas ktorých bol resetovací signál *RST* v aktívnej úrovni, vstupný signál *DCACHE\_MASTER\_HREADY* na master rozhraní v log. 1 a ostatné vstupné signály mali hodnotu 0. Z dôvodu obmedzenia vlastností na čo najmenší počet bol vytvorený univerzálny modul obsahujúci tieto vlastnosti ktorého inštancia bola vytvorená na rozhraní typu master ako aj slave. Tento modul bol vytvorený na základe architektúry dátovej pamäte typu L1 cache ako aj pravidiel protokolu AHB-lite. Keďže master rozhranie dátovej pamäte typu L1 cache je vytvorené pre jeden obvod typu slave, nenachádza sa tu signál *HSEL* a *HREADYOUT*. Na základe toho bol signál *HSEL* pripojený do log. 1, signál *HREADY* do log. 0 a signál *HREADYOUT* k signálu *DCACHE\_MASTER\_HREADY* v inštancii modulu obsahujúceho vlastnosti pripojeného k rozhraniu typu master. Takéto zapojenie je znázornené na Obrázku 5.10. Pri takejto kombinácii zapojenia preberá signál *HREADYOUT* funkciu signálu *HREADY*.

```

ahb_lite_checker #(
    .RST_ACT_LEVEL(RST_ACT_LEVEL),
    .RST_SYNC(RST_SYNC),
    .ADDR_SIZE(DCACHE_MASTER_ADDR_BITS),
    .DATA_SIZE(DCACHE_MASTER_DATA_BITS),
    .TYPE_OF_ITF(1'b0)
) ahb_lite_master_checker(
    .CLK(CLK),
    .RST(RST),
    .HADDR(DCACHE_MASTER_HADDR),
    .HBURST(DCACHE_MASTER_HBURST),
    .HMASTLOCK(DCACHE_MASTER_HMASTLOCK),
    .HPROT(DCACHE_MASTER_HPROT),
    .HSIZE(DCACHE_MASTER_HSIZE),
    .HTRANS(DCACHE_MASTER_HTRANS),
    .HWDATA(DCACHE_MASTER_HWDATA),
    .HWRITE(DCACHE_MASTER_HWRITE),
    .HREADY(1'b0),
    .HSEL(1'b1),
    .HRDATA(DCACHE_MASTER_HRDATA),
    .HREADYOUT(DCACHE_MASTER_HREADY),
    .HRESP(DCACHE_MASTER_HRESP)
);

```

Obrázok 5.10 Inštancia modulu obsahujúceho vlastnosti na rozhraní typu master

V tomto module bol vytvorený nový parameter *TYPE\_OF\_ITF* ktorý v log. 0 značí pripojenie k rozhraniu typu master a v log. 1 k rozhraniu typu slave. Na základe takéhoto parametru sa jednotlivé vlastnosti definujú ako súbežné formálne tvrdenia pomocou príkazu **assert** a ako predpoklady pomocou príkazu **assume** podľa typu rozhrania. V prípade, že sú definované ako predpoklady, slúžia tieto vlastnosti ako pravidlá pre generovanie vstupných stimulov nástrojom Questa PropCheck v súlade s protokolom AHB-lite. Na každú požiadavku boli vytvorené jednotlivé vlastnosti. Takto vytvorené vlastnosti boli následne rozšírené o vlastnosti, ktoré dopĺňali predpoklady určené na správne generovanie vstupných stimulov do dátovej pamäte typu L1 cache. Zoznam doplnených vlastností použitých ako predpoklady je uvedený v prílohe v kapitole C.4 pre rozhranie typu slave a C.5 pre rozhranie typu master. Tieto

doplnené vlastnosti pridávali pravidla použitia transakcií INCR, INCR4, INCR8, INCR16, typ prenosu BUSY a definovali hodnoty vstupných signálov dátovej pamäte typu L1 cache počas resetu. Pravidlá pre kontrolu adresy pri transakcii WRAP4, WRAP8 a WRAP16 boli vytvorené nanovo, pre každú kombináciu šírky prenášaného slova a druhu transakcie. Celkovo tak bolo vytvorených 67 pravidiel, ktoré boli navrhnuté tak, aby zohľadňovali možnosť výberu aktivačnej úrovne resetovacieho signálu *RST* podľa parametru *RST\_ACT\_LEVEL* a použitia synchronného alebo asynchronného resetu podľa parametru *RST\_SYNC*. Počet aplikovaných súbežných formálnych tvrdení a predpokladov na obidve rozhrania pri jednotlivých konfiguráciách je uvedený v Tabuľke 5.1. Prehľad jednotlivých konfigurácií dátovej pamäte typu L1 cache je uvedený v prílohe v kapitole C.1.

Tabuľka 5.1 Prehľad aplikovaných súbežných formálnych tvrdení a predpokladov na jednotlivých rozhraniach a pri rôznych konfiguráciách

Číslo konfigurácie	Rozhranie typu slave		Rozhranie typu master	
	Počet súbežných formálnych tvrdení	Počet predpokladov	Počet súbežných formálnych tvrdení	Počet predpokladov
1	8	29	11	7
2	8	22	11	7
3	8	26	11	7
4	8	29	11	7
5	8	35	12	7
6	8	29	12	7
7	8	22	12	7
8	8	26	12	7
9	8	22	12	7
10	8	32	12	7
11	8	38	12	7

Ako prvé bolo povolené generovanie hodnoty resetovacieho signálu *RST* počas procesu formálnej verifikácie a nastavená maximálna doba behu verifikácie 2 hodiny. Výsledky formálnej verifikácie všetkých konfigurácií dátovej pamäte typu L1 cache pri generovaní resetovacieho signálu *RST* sú uvedené v Tabuľke 5.2.

Tabuľka 5.2 Výsledky formálnej verifikácie dátovej pamäte typu L1 cache pri zapnutom generovaní hodnoty resetovacieho signálu *RST*.

Číslo konfigurácie	Celkový počet súbežných formálnych tvrdení	Počet dokázaných súbežných formálnych tvrdení	Počet neúspešne dokázaných súbežných formálnych tvrdení	Počet nedokázaných súbežných formálnych tvrdení	Počet prázdnych súbežných formálnych tvrdení	Čas [min]
1	19	19	0	0	0	48
2	19	17	0	1	1	120
3	19	18	0	1	0	120
4	19	19	0	0	0	92
5	20	19	0	1	0	120
6	20	20	0	0	0	14
7	20	18	0	1	1	120
8	20	20	0	0	0	13
9	20	18	0	1	1	120
10	20	20	0	0	0	11
11	20	20	0	0	0	40

V prípade konfigurácií 2, 7 a 9 nástroj Questa PropCheck vyhodnotil súbežné formálne tvrdenie **continious\_error\_2\_cycle\_response\_assertion** kontrolujúce správne nastavenie dvoch po sebe idúcich chybných odpovedí na rozhraní typu slave ako prázdne (vacuous). Takéto označenie znamená že neexistuje žiadny vstupný stimul na základe ktorého by bol predchodca (ľavá strana operátora implikácie) pravdivý po inicializačnej sekvencii. Pravdepodobne to je spôsobené tým, že pri 2 a viac chybných odpovediach za sebou, daná konfigurácia dátovej pamäte typu L1 cache vkladá medzi jednotlivé chybné odpovede čakacie cykly na rozhraní typu slave. Tieto konfigurácie sú charakteristické dátovou šírkou rozhrania typu slave 8 bitov. Pri konfiguráciách 2, 3, 5, 7 a 9 ktoré sú charakteristické asynchrónnym resetom, nástroj Questa PropCheck nestihol vyhodnotiť všetky súbežné formálne tvrdenia za čas 2 hodiny. Pri všetkých týchto konfiguráciách vyhodnotil súbežné formálne tvrdenie **hwdata\_stable\_assertion** kontrolujúce podržanie dát na dátovej zbernici obvodu typu master v prípade predĺženej dátovej fázy zápisu ako nedokázané. V prípade konfigurácie 2 a 3 označil nástroj Questa Propcheck 5 a 4 súbežných formálnych tvrdení ako dokázaných ale pravdepodobne prázdnych (possibly-vacuous). Príčinou takéhoto označenia bol nedostatočný čas formálnej verifikácie za ktorý nástroj Questa PropCheck nestihol skontrolovať či dané súbežné formálne tvrdenie nie je prázdne. Výsledok formálnej verifikácie dátovej pamäte typu L1 cache pri zapnutom generovaní hodnôt resetovacieho signálu *RST* konfigurácie 2 je na Obrázku 5.11 a konfigurácie 3 je na Obrázku 5.12.

Name	Health	Radius	Time
...ahb_lite_slave_checker.slave_if.continuous_error_2_cycle_response_assertion	★ 7		35s
...platform_ahb_t.ahb_lite_master_checker.master_if.hwdata_stable_assertion	● 10	253 @ CLK	1h 40m 21s
...orm_ahb_t.ahb_lite_slave_checker.slave_if.error_2_cycle_response_assertion	★ 10		1h 43m 44s
...lite_master_checker.master_if.wider_write_transfer_than_data_bus_assertion	★ 7		1h 33m 3s
...ahb_lite_master_checker.master_if.request_stable_until_response_assertion	★ 7		1h 50m 17s
...b_t.ahb_lite_master_checker.master_if.wrap4_number_of_transfer_assertion	★ 12		1h 53m 52s
...hb_t.ahb_lite_master_checker.master_if.wrap_size_8_addr_check_assertion	★ 10		1h 39m 28s
...e_slave_checker.slave_if.asyn_reset.hreadyout_during_asyn_reset_assertion	★ 7		34s
...che_platform_ahb_t.ahb_lite_slave_checker.slave_if.busy_response_assertion	★ 7		34s
...ache_platform_ahb_t.ahb_lite_slave_checker.slave_if.idle_response_assertion	★ 7		34s
...latform_ahb_t.ahb_lite_slave_checker.slave_if.extend_data_phase_assertion	★ 7		35s
...dasip_dcache_platform_ahb_t.ahb_lite_slave_checker.hwdata_size_assertion	★ 7		34s
...odasip_dcache_platform_ahb_t.ahb_lite_slave_checker.hrdata_size_assertion	★ 7		34s
...e_master_checker.master_if.asyn_reset.htrans_during_asyn_reset_assertion	★ 7		34s
...r_checker.master_if.asyn_reset.master_signals_during_asyn_reset_assertion	★ 7		34s
...lite_master_checker.master_if.wider_read_transfer_than_data_bus_assertion	★ 7		34s
...latform_ahb_t.ahb_lite_master_checker.master_if.htrans_after_idle_assertion	★ 10		38s
...asip_dcache_platform_ahb_t.ahb_lite_master_checker.hwdata_size_assertion	★ 7		34s
...asip_dcache_platform_ahb_t.ahb_lite_master_checker.hrdata_size_assertion	★ 7		34s

Obrázok 5.11 Zobrazenie výsledku formálnej verifikácie užívateľským rozhraním nástroja pri druhej konfigurácii dátovej pamäte typu L1 cache a pri zapnutom generovaní hodnôt resetovacieho signálu *RST*

Name	Health	Radius	Time
...b_lite_master_checker.master_if.hwdata_stable_assertion	● 12	253 @ CLK	1h 49m 3s
...r.master_if.wider_write_transfer_than_data_bus_assertion	★ 7		1h 22m 3s
...ecker.master_if.request_stable_until_response_assertion	★ 7		1h 25m 46s
...r_checker.master_if.wrap8_number_of_transfer_assertion	★ 12		1h 27m 54s
...ter_checker.master_if.wrap_size_8_addr_check_assertion	★ 10		1h 32m 25s
...ve_if.asyn_reset.hreadyout_during_asyn_reset_assertion	★ 7		1m 13s
...t.ahb_lite_slave_checker.slave_if.busy_response_assertion	★ 7		1m 14s
..._t.ahb_lite_slave_checker.slave_if.idle_response_assertion	★ 7		1m 14s
..._slave_checker.slave_if.error_2_cycle_response_assertion	★ 10		1m 17s
...cker.slave_if.continuous_error_2_cycle_response_assertion	★ 7		1m 14s
...lite_slave_checker.slave_if.extend_data_phase_assertion	★ 7		1m 14s
...orm_ahb_t.ahb_lite_slave_checker.hwdata_size_assertion	★ 7		1m 13s
...tform_ahb_t.ahb_lite_slave_checker.hrdata_size_assertion	★ 7		1m 13s
...master_if.asyn_reset.htrans_during_asyn_reset_assertion	★ 7		1m 13s
...f.asyn_reset.master_signals_during_asyn_reset_assertion	★ 7		1m 13s
...r.master_if.wider_read_transfer_than_data_bus_assertion	★ 7		1m 14s
...lite_master_checker.master_if.htrans_after_idle_assertion	★ 10		1m 17s
...rm_ahb_t.ahb_lite_master_checker.hwdata_size_assertion	★ 7		1m 13s
...rm_ahb_t.ahb_lite_master_checker.hrdata_size_assertion	★ 7		1m 13s

Obrázok 5.12 Zobrazenie výsledku formálnej verifikácie užívateľským rozhraním nástroja pri tretej konfigurácii dátovej pamäte typu L1 cache a pri zapnutom generovaní hodnôt resetovacieho signálu *RST*

Na základe predchádzajúcich výsledkov bolo v nasledujúcom experimente zavedené obmedzenie hodnoty resetovacieho signálu *RST* na neaktívnu úroveň po inicializačnej sekvencii vymazaním argumentu **auto\_constraint\_off** z príkazu **formal verify** v spúšťačom skripte. Boli odstránené všetky súbežné formálne tvrdenia a predpoklady kontrolujúce a nastavujúce signály počas resetu. V prípade že by neboli súbežné formálne tvrdenia kontrolujúce signály počas resetu odstránené, nástroj

Questa PropCheck by ich označil ako prázdne. Maximálna doba behu verifikácie bola zmenšená na 60 minút. Výsledky formálnej verifikácie všetkých konfigurácií pri generovaní resetovacieho signálu *RST* sú uvedené v Tabuľke 5.3.

Tabuľka 5.3 Výsledky formálnej verifikácie dátovej pamäte typu L1 cache pri obmedzení resetovacieho signálu *RST* počas formálnej verifikácie.

Číslo konfigurácie	Celkový počet súbežných formálnych tvrdení	Počet dokázaných súbežných formálnych tvrdení	Počet neúspešne dokázaných súbežných formálnych tvrdení	Počet nedokázaných súbežných formálnych tvrdení	Počet prázdnych súbežných formálnych tvrdení	Čas [min]
1	16	16	0	0	0	8
2	16	15	0	0	1	26
3	16	16	0	0	0	10
4	16	16	0	0	0	12
5	17	17	0	0	0	12
6	17	17	0	0	0	11
7	17	16	0	0	1	24
8	17	17	0	0	0	4
9	17	16	0	0	1	13
10	17	17	0	0	0	10
11	17	17	0	0	0	10

Pri obmedzení resetovacieho signálu *RST* na neaktívnu úroveň po inicializačnej sekvencii nástroj Questa PropCheck dokázal za pomerne krátku dobu vyhodnotiť všetky súbežné formálne tvrdenia okrem súbežných formálnych tvrdení kontrolujúcich signály počas resetu ktoré boli odstránené. Podobne ako v predchádzajúcom experimente pri konfigurácii 2, 7 a 9 označil nástroj Questa PropCheck súbežné formálne tvrdenie **continuous\_error\_2\_cycle\_response\_assertion** kontrolujúce správne nastavenie dvoch po sebe idúcich chybných odpovedí ako prázdne. Zvyšné súbežné formálne tvrdenia sa podarilo úspešne overiť. Súbežné formálne tvrdenia ktoré pri konfigurácii 2 a 3 v predchádzajúcom experimente nástroj Questa PropCheck vyhodnotil ako pravdepodobne prázdne boli pri tomto experimente označené ako dokázane a zároveň neprázdne.

### 5.3 Zhodnotenie prínosov formálnej verifikácie

Jednou z veľkých výhod formálnej verifikácie je fakt, že nepotrebuje od užívateľa vytvorené testovacie vektory na overenie 100% správnosti daného správania sa tak, ako tomu je v prípade použitia funkčnej verifikácie založenej na princípe simulácii. Táto vlastnosť formálnej verifikácie výrazne skraca čas, ktorý je potrebný na overenie predkladaného návrhu. Použitie funkčnej verifikácie, ktorá je

založená na simulácii, častokrát neumožňuje overiť dané správanie sa pri prejdení celého stavového priestoru návrhu pomocou vytvorených testovacích vektorov. Dôvodom je časová náročnosť samotnej realizácie simulácie. Na druhej strane aj formálna verifikácia je obmedzená veľkosťou stavového priestoru návrhu a náročná na výpočtový výkon.

V prípade metódy kontroly modelu založenej na verifikácii pomocou formálnych tvrdení je overenie kombinačnej logiky veľmi rýchle a jednoduché. To sa potvrdilo pri verifikácii dekodéru pomocou oneskorených okamžitých formálnych tvrdení. V prípade dosiahnutia rovnakých výsledkov pri verifikácii dekodéru pomocou simulácie by bola doba takejto verifikácie omnoho dlhšia ako doba formálnej verifikácie. Táto skutočnosť predstavuje obrovskú výhodu oproti funkčnej verifikácii kombinačnej logiky pomocou simulácie. Metóda kontroly modelu je tiež veľmi vhodná na overovanie pravidiel komunikačných protokolov. Pri overovaní komunikačných protokolov je dôležité zadať predpoklady pre generovanie správnych vstupných stimulov nástroja, podľa pravidiel daného protokolu, poprípade aj hodnôt vstupných signálov počas resetu. V opačnom prípade by došlo ku generovaniu vstupných stimulov, ktoré porušujú pravidlá protokolu a návrh by sa mohol začať chovať podľa neočakávaného správania. Medzi výhody patrí aj možnosť nastavenia generovania resetu v priebehu formálnej verifikácie. Pri nastavení generovania resetu počas formálnej verifikácie sa v niektorých prípadoch môže čas potrebný na overenie jednotlivých pravidiel protokolu niekoľkonásobne zvýšiť, zvlášť v prípadoch, ak sa jedná o asynchrónny typ resetu. K týmto prípadom došlo pri formálnej verifikácii pravidiel protokolu dátovej pamäti typu L1 cache. Pri overovaní aritmetických operácií však už táto metóda formálnej verifikácie zlyhala, nakoľko došlo k nežiadúcemu nárastu časového trvania samotného overovania. Jednalo sa o overenie funkčnosti jednotlivých operácií násobenia, delenia a zvyšku po delení celočíselných hodnôt pri všetkých kombináciách 32-bitových operandov. Hlavným problémom bola zložitosť jednotlivých súbežných formálnych tvrdení v kombinácii s veľkou latenciou operácií. Pri takomto type overovania by bolo vhodnejšie použiť funkčnú verifikáciu založenú na simulácii.

# ZÁVER

Predložená diplomová práca pojednáva o formálnej verifikácii procesora Codix Berkelium 5 založeného na architektúre RISC-V od spoločnosti Codasip. V teoretickej časti diplomovej práce je rozobraná architektúra RISC-V a popis vybraných komponentov implementovaných v procesore určených na formálnu verifikáciu. Jedná sa o komponenty - dekodér, modul pre násobenie a delenie celočíselných hodnôt a dátová pamäť typu L1 cache. Ďalšia časť teórie je venovaná formálnej verifikácii, jej metódam ako overovanie modelu a dokazovaniu viet. Taktiež je tu rozobratý popis formálnych nástrojov využívajúcich tieto metódy.

Cieľom tejto diplomovej práce bolo vytvorenie verifikačného plánu vybraných komponentov a následné použitie formálnej verifikácie na ich overenie pomocou nástroja Questa PropCheck s využitím metódy kontroly modelu a použitím formálnych tvrdení jazyka SystemVerilog Assertions. Pri kontrole dekódovania všetkých 56 inštrukcií s 32-bitovým formátom pri konfigurácii RV32IMC, použitím oneskorených okamžitých formálnych tvrdení boli odhalené dve chyby. Jedna pri inštrukcii fence.i a druhá pri ecall. Pri verifikácii zameranej na kontrolu funkcionality všetkých operácií nachádzajúcich sa v module pre násobenie a delenie celočíselných hodnôt nástroj Questa PropCheck nebol schopný overiť zadané súbežné formálne tvrdenia z dôvodu časovej náročnosti procesu overovania. Posledný komponent predstavovala dátová pamäť typu L1 cache, na ktorej bola prevedená kontrola pravidiel protokolu AHB-lite pri jej 11 konfiguráciách formou súbežných formálnych tvrdení. Celkovo bolo vytvorených 67 vlastností popisujúcich správanie protokolu AHB-lite. Pri kontrole tohto typu boli uskutočnené dva experimenty. Prvý experiment pozostával z kontroly pravidiel protokolu AHB-lite pri generovaní resetu počas formálnej analýzy a druhý s obmedzením resetu na neaktívnu úroveň po inicializačnej sekvencii. Prvý experiment preukázal, že mal nástroj Questa PropCheck s vyhodnotením všetkých súbežných formálnych tvrdení problém, konkrétne pri konfiguráciách obsahujúcich asynchrónny reset a s nastavenou maximálnou dobou verifikačného behu 2 hodiny. Pri druhom experimente však dokázal všetky súbežné formálne tvrdenia vyhodnotiť za pomerne krátku dobu. Kontrola pravidiel protokolu AHB-lite dátovej pamäte typu L1 cache nepreukázala porušenie týchto pravidiel. V závere tejto práce sú zhodnotený prínosy použitej techniky formálnej verifikácie oproti funkčnej verifikácii založenej na princípe simulácie.

# LITERATURA

- [1] CHOCKLER, H, A IVRII, A MATSLIAH, S MORAN a Z NEVO. *Incremental formal verification of hardware*. 2011 *Formal Methods in Computer-Aided Design (FMCAD)* [online]. IEEE, 2011, s. 135-143 [cit. 2019-12-10]. ISBN 9781467308960. Dostupné z: <https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/document/6148887>
- [2] RISC-V History. [online]. [cit. 2019-12-10]. Dostupné z: <https://riscv.org/risc-v-history/>
- [3] RISC-V Foundation. [online]. [cit. 2019-12-10]. Dostupné z: <https://riscv.org/risc-v-foundation/>
- [4] WATERMAN, A., LEE, Y., AVIZIENIS, R. *The RISC-V Instruction Set Manual Volume I: Unprivileged*. [online]. [cit. 2019-12-10]. Dostupné z: <https://riscv.org/specifications/>
- [5] BARTÁK, Jiří. *Model procesoru RISC-V*. Vysoké učení technické v Brně. Fakulta informačních technologií, Ústav počítačových systémů, 2015, 56 listov, Diplomová práce. Vedúci práce: Ing. Šimková Marcela
- [6] Codasip: *Codix Berkelium Bk5 Datasheet, Version 1.6.0, Document Version 1.2.0*. Oct 2017.
- [7] Cache paměti, [online]. [cit. 2020-4-20]. Dostupné z: <https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/CACHE.HTML>
- [8] DUBOIS, Michel, Murali ANNAVARAM a Per STENSTRÖM. *Parallel computer organization and design*. Cambridge: Cambridge University Press, 2012, xvii, 542 s. : il. ISBN 978-0-521-88675-8.
- [9] Codasip: *Interná špecifikácia pamäti cache*. 2020.
- [10] Codasip: *Codix Berkelium RISC-V Compliant Processor IP*. 2020.
- [11] ARM Limited.: *AMBA® 3 AHB-Lite Protocol Specification* [online]. [cit. 2020-4-20]. Dostupné z: [http://eecs.umich.edu/courses/eecs373/readings/ARM\\_IHI0033A\\_AMBA\\_AHB-Lite\\_SPEC.pdf](http://eecs.umich.edu/courses/eecs373/readings/ARM_IHI0033A_AMBA_AHB-Lite_SPEC.pdf)
- [12] GRIMM, Tomáš, Djones LETTNIN a Michael HÜBNER. A Survey on Formal Verification Techniques for Safety-Critical Systems-on-Chip. *Electronics* [online]. MDPI, 2018, 7(6), 81 [cit. 2019-12-10]. DOI: 10.3390/electronics7060081. Dostupné z : [https://primo.lib.vutbr.cz/permalink/f/1pt3lf4/TN\\_doaj\\_soai\\_doaj\\_org\\_article\\_10e99deb2a0b41e69f48e80b0beef1c9](https://primo.lib.vutbr.cz/permalink/f/1pt3lf4/TN_doaj_soai_doaj_org_article_10e99deb2a0b41e69f48e80b0beef1c9)
- [13] Formal verification. [online]. [cit. 2019-12-10]. Dostupné z : <https://www.techdesignforums.com/practice/guides/formal-verification-guide/>
- [14] KONNOV, Igor, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (eds): *Handbook of model checking* ; Springer International Publishing AG, Cham, Switzerland, 2018, xxiv+1210 pp, ISBN 978-3-319-10574-1 (Hardcover, 2.13 kg), ISBN 978-3-319-10575-8 (eBook, PDF). Springer, 2019, 31(4), 455 [cit. 2019-12-10]. DOI: 10.1007/s00165-019-00486-z. ISSN 0934-5043
- [15] BAIER, Christel a Joost-Pieter KATOEN. *Principles of model checking*. Cambridge ; London: MIT Press, 2008, xvii, 975 s. ISBN 978-0-262-02649-9.

- [16] ČAUŠEVIĆ, Aida. *Formal Approaches to Service-oriented Design: From Behavioral Modeling to Service Analysis* [online]. 2011 [cit. 2019-12-10]. ISBN 9789174850123. ISSN 1651-9256 Dostupné z : [https://primo.lib.vutbr.cz/permalink/f/1pt3lf4/TN\\_swepuboi:DiVA.org:mdh-12166](https://primo.lib.vutbr.cz/permalink/f/1pt3lf4/TN_swepuboi:DiVA.org:mdh-12166)
- [17] SystemVerilog Assertions Tutorial, [online]. [cit. 2019-12-10]. Dostupné z : <https://www.doulos.com/knowhow/sysverilog/tutorial/assertions/>
- [18] IEEE *Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language*. New York, USA: IEEE, 2018. DOI: 10.1109/IEEESTD.2018.8299595.
- [19] CERNY, Eduard. *The power of assertions in SystemVerilog*. New York ; London: Springer, 2010, xvii, 544 s. : il. ISBN 978-1-4419-6599-8.
- [20] ŠIMKOVÁ, M., KAJAN, M. *Verifikace číslicových obvodů: využití metod a nástrojů formální verifikace*. V Brně: Vysoké učení technické, Fakulta informačních technologií, Ústav počítačových systémů, 2013. 25 s.
- [21] VIJAYARAGHAVAN, Srikanth a Meyyappan RAMANATHAN. *A practical guide for SystemVerilog assertions*. New York: Springer, 2005, xxv, 334 s. : il. ISBN 978-0-387-26049-5.
- [22] Assertion-Based Verification, [online]. [cit. 2019-12-5]. Dostupné z : <https://verificationacademy.com/courses/assertion-based-verification>
- [23] Questa PropCheck, [online]. [cit. 2019-11-27]. Dostupné z : <https://verificationacademy.com/products/questa-propcheck>
- [24] Verification Planning for Core based Designs, [online]. [cit. 2019-12-1]. Dostupné z : <https://www.design-reuse.com/articles/16141/verification-planning-for-core-based-designs.html>
- [25] Mentor: *Questa PropCheck User Guide: Software Version 2019.4*

# ZOZNAM OBRÁZKOV

Obrázok 1.1 Základné formáty inštrukcií RISC-V (prevzaté z [4]) .....	3
Obrázok 1.2 Spracovanie inštrukcii procesoru Bk 5 - 5-stupňové zreťazenie v poradí (prevzaté z [6]).....	4
Obrázok 1.3 Zjednodušená schéma zapojenia dátovej pamäte typu L1 cache (upravené podľa [9]) .....	5
Obrázok 1.4 Organizácia dátovej pamäti typu L1 cache[6] .....	7
Obrázok 1.5 Schéma muldiv modulu .....	<b>Chyba! Záložka není definována.</b>
Obrázok 1.6 Schéma dekodéru .....	<b>Chyba! Záložka není definována.</b>
Obrázok 2.1 Príklad obecnej štruktúry AHB-Lite s 3 obvodymi typu slave a 32-bitovou dátovou zbernicou (prevzaté z [11]) .....	11
Obrázok 2.2 Príklad zjednodušeného prenosu čítania bez čakacieho cyklu (prevzaté z [11]) .....	12
Obrázok 2.3 Príklad prenosu čítania s dvoma čakajúcimi cyklami (prevzaté z [11]) ....	13
Obrázok 2.4 Príklad inkrementujúcej dávky o 8 prenosoch zápisu a šírkou prenášaného slova 2 bajty (prevzaté z [11]) .....	14
Obrázok 2.5 Príklad zaobalenej dávky o 8 prenosoch čítania a šírkou prenášaného slova 4 bajty (prevzaté z [11]).....	15
Obrázok 3.1 Verifikačná metodológia kontroly modelu (prevzaté z [16]).....	17
Obrázok 3.2 Príklad jednoduchej sekvencie.....	19
Obrázok 3.3 Príklad použitia vlastnosti until_with (prevzaté z [19]).....	21
Obrázok 3.4 Príklad použitia prekrývajúcej sa implikácie vo vlastnosti.....	21
Obrázok 3.5 Príklad použitia neprekrývajúcej sa implikácie vo vlastnosti.....	21
Obrázok 3.6 Systémová štruktúra nástrojov pre ATP (prevzaté z [16]).....	24
Obrázok 5.1 Súbor <b>run</b> určený na spustenie nástroja Questa PropCheck pri verifikácii dátovej pamäte typu L1 cache .....	29
Obrázok 5.2 Spúšťač skript <b>run_dcachе.do</b> nástroja Questa PropCheck určený na verifikáciu dátovej pamäte typu L1 cache .....	30
Obrázok 5.3 Oneskorené okamžité formálne tvrdenie pre inštrukciu súčtu.....	31
Obrázok 5.4 Zhrnutie výsledku verifikácie dekodéru z výstupného súboru <b>formal_verify.log</b> .....	32
Obrázok 5.5 Vlastnosť kontrolujúca správnosť počítania operácie mulhu .....	33
Obrázok 5.6 Zhrnutie výsledku verifikácie modulu pre násobenie a delenie celočíselných hodnôt po 12 hodinách z výstupného súboru <b>formal_verify.log</b> .....	34
Obrázok 5.7 Obmedzenie hodnôt vstupných operandov <i>rs1</i> a <i>rs2</i> pomocou príkazu <b>assume</b> .....	34

Obrázok 5.8	Zobrazenie výsledku verifikácie modulu pre násobenie a delenie celočíselných hodnôt pri obmedzení vstupných operandov užívateľským rozhraním nástroja .....	35
Obrázok 5.9	Zapúzdrenie dátovej pamäte typu L1 cache s pripojenými pamäťami SRAM do Top modulu .....	35
Obrázok 5.10	Inštancia modulu obsahujúceho vlastnosti na rozhraní typu master .....	36
Obrázok 5.11	Zobrazenie výsledku formálnej verifikácie užívateľským rozhraním nástroja pri druhej konfigurácii dátovej pamäte typu L1 cache a pri zapnutom generovaní hodnôt resetovacieho signálu <i>RST</i> .....	39
Obrázok 5.12	Zobrazenie výsledku formálnej verifikácie užívateľským rozhraním nástroja pri tretej konfigurácii dátovej pamäte typu L1 cache a pri zapnutom generovaní hodnôt resetovacieho signálu <i>RST</i> .....	39
Obrázok 0.1	Vygenerovaný protipríklad formálneho tvrdenia inštrukcie <i>ecall</i> .....	64
Obrázok 0.2	Vygenerovaný protipríklad formálneho tvrdenia inštrukcie <i>fence.i</i> .....	65

## Zoznam tabuliek

Tabuľka 2.1	Úplná prenosová odpoveď (prevzaté z [11]).....	15
Tabuľka 3.1	Súbor operátorov sekvencií a vlastností aj s príslušnou asociativitou (prevzaté z [18]).....	20
Tabuľka 3.2	Popis formalizmov pri automatickom dokazovaní viet (prevzaté z [16])..	23
Tabuľka 5.1	Prehľad aplikovaných súbežných formálnych tvrdení a predpokladov na jednotlivých rozhraniach a pri rôznych konfiguráciách.....	37
Tabuľka 5.2	Výsledky formálnej verifikácie dátovej pamäte typu L1 cache pri zapnutom generovaní hodnoty resetovacieho signálu <i>RST</i> .....	38
Tabuľka 5.3	Výsledky formálnej verifikácie dátovej pamäte typu L1 cache pri obmedzení resetovacieho signálu <i>RST</i> počas formálnej verifikácie.....	40
Tabuľka 0.1	Zoznam operácií určujúcich signálom <i>alu function</i> .....	48
Tabuľka 0.2	Zoznam operácií používaných s kontrolnými a stavovými registrami určujúcich signálom <i>csr_op</i> .....	48
Tabuľka 0.3	Výber prvého operandu aritmeticko-logickej jednotky pomocou signálu <i>aluop1_mux</i> .....	49
Tabuľka 0.4	Výber druhého operandu aritmeticko-logickej jednotky pomocou signálu <i>aluop1_mux</i> .....	49
Tabuľka 0.5	Výber typu priameho operandu pomocou signálu <i>sel_imm</i> .....	49
Tabuľka 0.6	Výber typu prístupu do pamäte pomocou signálu <i>mem_cmd</i> .....	49
Tabuľka 0.7	Výber šírky prístupu do pamäte pomocou signálu <i>mem_type</i> .....	50

Tabuľka 0.8	Zoznam inštrukcií násobenia a delenia aj s ich požadovanými dekodovanými signálmi.....	52
Tabuľka 0.9	Zoznam inštrukcií pracujúcich s kontrolnými a stavovými registrami aj s ich požadovanými dekodovanými signálmi.....	52
Tabuľka 0.10	Zoznam aritmetických a logických inštrukcií aj s ich požadovanými dekodovanými signálmi.....	52
Tabuľka 0.11	Zoznam inštrukcií zápisu a čítania z pamäte aj s ich požadovanými dekodovanými signálmi.....	53
Tabuľka 0.12	Zoznam inštrukcií skokov aj s ich požadovanými dekodovanými signálmi .....	54
Tabuľka 0.13	Zoznam zvyšných inštrukcií aj s ich požadovanými dekodovanými signálmi.....	54
Tabuľka 0.14	Naplánovaná kontrola jednotlivých konfigurácií dátovej pamäte typu L1 cache .....	55
Tabuľka 0.15	Vytvorené požiadavky protokolu AHB-lite na rozhraní typu slave.....	55
Tabuľka 0.16	Vytvorené požiadavky protokolu AHB-lite na rozhraní typu master .....	56
Tabuľka 0.17	Prehľad doplnených vlastností použitých ako predpoklady na rozhraní typu slave .....	59
Tabuľka 0.18	Prehľad doplnených vlastností použitých ako predpoklady na rozhraní typu master.....	63

# A POPIS DEKÓDOVANÝCH RIADIACICH SIGNÁLOV DEKODÉRA

Tabuľka 0.1 Zoznam operácií určujúcich signálom *alu function*

<i>alu_function</i> [3:0]	Označenie operácie aritmeticko-logickej jednotky	Popis operácie aritmeticko-logickej jednotky	Označenie operácie modulu pre násobenie a delenie celočíselných hodnôt
0b0000	ALU_ADD	súčet	ALU_MUL
0b0001	ALU_SL	logický posun vľavo	ALU_MULH
0b0010	-	-	-
0b0011	-	-	-
0b0100	ALU_XOR	exkluzívny logický súčet	ALU_DIV
0b0101	ALU_SR	logický posun vpravo	ALU_DIVU
0b0110	ALU_OR	logický súčet	ALU_REM
0b0111	ALU_AND	logický súčin	-
0b1000	ALU_EQ	rovnosť	-
0b1001	ALU_NE	nerovnosť	-
0b1010	ALU_SUB	rozdiel	-
0b1011	ALU_SRA	aritmetický posun vpravo	-
0b1100	ALU_LT	menší než	ALU_MULHU
0b1101	ALE_GE	väčší než	-
0b1110	ALU_LTU	menší než (beznamienkové porovnanie)	ALU_MULHSU
0b1111	ALU_GEU	väčší než (beznamienkové porovnanie)	-

Tabuľka 0.2 Zoznam operácií používaných s kontrolnými a stavovými registrami určujúcich signálom *csr\_op*

<i>csr_op</i> [1:0]	Označenie	Popis
0b00	CSR_NONE	nevykoná sa nič
0b01	CSR_WRITE	zápis
0b10	CSR_SET	nastaví vybrané bity do log.1
0b11	CSR_CLEAR	nastaví vybrané bity do log. 0

Tabuľka 0.3 Výber prvého operandu aritmeticko-logickej jednotky pomocou signálu *aluop1\_mux*

<i>aluop1_mux</i> [1:0]	Označenie	Popis
0b00	ALUOP1_NONE	nevybere sa žiadny operand
0b01	ALUOP1_RS1	použije sa prvý zdrojový register
0b10	ALUOP1_PC	použije sa hodnota programového počítadla
0b11	ALUOP1_IMM_CSR	použije sa hodnota priameho operandu pri operácii s kontrolným a stavovým registrom

Tabuľka 0.4 Výber druhého operandu aritmeticko-logickej jednotky pomocou signálu *aluop2\_mux*

<i>aluop2_mux</i> [1:0]	Označenie	Popis
0b00	ALUOP2_NONE	nevybere sa žiadny operand
0b01	ALUOP2_LINK	použije sa veľkosť inštrukcie v bajtoch
0b10	ALUOP2_RS2	použije sa druhý zdrojový register
0b11	ALUOP2_IMM	použije sa hodnota priameho operandu

Tabuľka 0.5 Výber typu priameho operandu pomocou signálu *sel\_imm*

<i>sel_imm</i> [2:0]	Označenie
0b000	IMM_S
0b001	IMM_SB
0b010	IMM_U
0b011	IMM_UJ
0b100	IMM_I
0b101	IMM_Z (nevybere sa žiadny priamy operand)
0b110	-
0b111	-

Tabuľka 0.6 Výber typu prístupu do pamäte pomocou signálu *mem\_cmd*

<i>mem_cmd</i> [1:0]	Označenie	Popis
0b00	CMD_NONE	nevykoná sa nič
0b01	CMD_READ	čítanie z pamäte
0b10	CMD_WRITE	zápis do pamäte
0b11	-	-

Tabuľka 0.7 Výber šírky prístupu do pamäte pomocou signálu *mem\_type*

<i>mem_type</i> [2:0]	Označenie	Popis
0b000	MEM_BYTE	bajt
0b001	MEM_HALF	poloslovo
0b010	MEM_WORD	slovo
0b011	-	-
0b100	MEM_UBYTE	bezznamienkový bajt
0b101	MEM_UHALF	bezznamienkové poloslovo
0b110	-	-
0b111	-	-

## B ZOZNAM RISC-V INŠTRUKCIÍ

### B.1 Inštrukcie vykonávané v module pre násobenie a delenie celočíselných hodnôt

Násobenie:

- **MUL** - operácia násobenia dvoch celočíselných hodnôt, pri ktorej výsledok má dvojnásobnú bitovú šírku ako je šírka registra, a do cieľového registra sa zapíše spodná polovica výsledku
- **MULH** - operácia násobenia dvoch znamienkových celočíselných hodnôt, pri ktorej výsledok má dvojnásobnú bitovú šírku ako je šírka registra. Do cieľového registru sa zapíše vrchná polovica výsledku.
- **MULHU** - operácia násobenia dvoch bezznamienkových celočíselných hodnôt, pri ktorej výsledok má dvojnásobnú bitovú šírku ako je šírka registra. Do cieľového registru sa zapíše vrchná polovica výsledku.
- **MULHSU** - operácia násobenia znamienkového celočíselného čísla a bezznamienkového celočíselného čísla, pri ktorej výsledok má dvojnásobnú bitovú šírku ako je šírka registra. Do cieľového registru sa zapíše vrchná polovica výsledku.

Delenie:

- **DIV** - operácia delenia dvoch znamienkových celočíselných hodnôt so zaokrúhlením smerom k nule
- **DIVU** - operácia delenia dvoch bezznamienkových celočíselných hodnôt so zaokrúhlením smerom k nule
- **REM** - do cieľového celočíselného registra zapíše hodnotu zvyšku po delení dvoch znamienkových celočíselných hodnôt
- **REMU** - do cieľového celočíselného registra zapíše hodnotu zvyšku po delení dvoch bezznamienkových celočíselných hodnôt

## B.2 Zoznam inštrukcií dekodéru aj s ich požadovanými hodnotami dekódovaných signálov

Tabuľka 0.8 Zoznam inštrukcií násobenia a delenia aj s ich požadovanými dekódovanými signálmi

Inštrukcia	<i>alu_function</i> [3:0]	<i>muldiv</i>	<i>aluop1_mux</i> [1:0]	<i>aluop2_mux</i> [1:0]	<i>rs1_ren</i>	<i>rs2_ren</i>	<i>sel_imm</i> [2:0]	<i>reg_write</i>	ostatné signály
mul	0b0000 (ALU_MUL)	0b1	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
mulh	0b0001 (ALU_MULH)	0b1	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
mulhu	0b1100 (ALU_MULHU)	0b1	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
mulhsu	0b1110 (ALU_MULHSU)	0b1	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
div	0b0100 (ALU_DIV)	0b1	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
divu	0b0101 (ALU_DIVU)	0b1	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
rem	0b0110 (ALU_REM)	0b1	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
remu	0b0111 (ALU_REMU)	0b1	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0

Tabuľka 0.9 Zoznam inštrukcií pracujúcich s kontrolnými a stavovými registrami aj s ich požadovanými dekódovanými signálmi

Inštrukcia	<i>csr_op</i> [1:0]	<i>aluop1_mux</i> [1:0]	<i>rs1_ren</i>	<i>sel_imm</i> [2:0]	<i>reg_write</i>	ostatné signály
csrw	0b01 (CSR_WRITE)	0b01 (ALUOP1_RS1)	0b1	0b101 (IMM_Z)	0b1	0b0
csrrs	0b11 (CSR_SET)	0b01 (ALUOP1_RS1)	0b1	0b101 (IMM_Z)	0b1	0b0
csrrc	0b10 (CSR_CLEAR)	0b01 (ALUOP1_RS1)	0b1	0b101 (IMM_Z)	0b1	0b0
csrrwi	0b01 (CSR_WRITE)	0b11 (ALUOP1_IMM_CSR)	0b0	0b101 (IMM_Z)	0b1	0b0
csrrsi	0b11 (CSR_SET)	0b11 (ALUOP1_IMM_CSR)	0b0	0b101 (IMM_Z)	0b1	0b0
csrrci	0b10 (CSR_CLEAR)	0b11 (ALUOP1_IMM_CSR)	0b0	0b101 (IMM_Z)	0b1	0b0

Tabuľka 0.10 Zoznam aritmetických a logických inštrukcií aj s ich požadovanými dekódovanými signálmi

Inštrukcia	<i>alu_function</i> [3:0]	<i>aluop1_mux</i> [1:0]	<i>aluop2_mux</i> [1:0]	<i>rs1_ren</i>	<i>rs2_ren</i>	<i>sel_imm</i> [2:0]	<i>reg_write</i>	ostatné signály
addi	0b0000 (ALU_ADD)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
slti	0b1100 (ALU_LT)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
sltiu	0b1110 (ALU_LTU)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0

xori	0b0100 (ALU_XOR)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
ori	0b0110 (ALU_OR)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
andi	0b0111 (ALU_AND)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
slli	0b0001 (ALU_SL)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
srli	0b0101 (ALU_SR)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
srai	0b1011 (ALU_SRA)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
add	0b0000 (ALU_ADD)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
sub	0b1010 (ALU_SUB)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
sll	0b0001 (ALU_SL)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
slt	0b1100 (ALU_LT)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
sltu	0b1110 (ALU_LTU)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
xor	0b0100 (ALU_XOR)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
srl	0b0101 (ALU_SR)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
sra	0b1011 (ALU_SRA)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
or	0b0110 (ALU_OR)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0
and	0b0111 (ALU_AND)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b101 (IMM_Z)	0b1	0b0

Tabuľka 0.11 Zoznam inštrukcií zápisu a čítania z pamäte aj s ich požadovanými dekódovanými signálmi

Inštrukcia	<i>mem_cmd</i> [1:0]	<i>mem_type</i> [2:0]	<i>aluop1_mux</i> [1:0]	<i>aluop2_mux</i> [1:0]	<i>rs1_ren</i>	<i>rs2_ren</i>	<i>sel_imm</i> [2:0]	<i>reg_write</i>	ostatné signály
lb	0b01 (CMD_READ)	0b000 (MEM_BYTE)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
lbu	0b01 (CMD_READ)	0b100 (MEM_UBYTE)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
lh	0b01 (CMD_READ)	0b001 (MEM_HALF)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
lhu	0b01 (CMD_READ)	0b101 (MEM_UHALF)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
lw	0b01 (CMD_READ)	0b010 (MEM_WORD)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b0	0b100 (IMM_I)	0b1	0b0
sb	0b10 (CMD_WRITE)	0b000 (MEM_BYTE)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b1	0b000 (IMM_S)	0b0	0b0
sh	0b10 (CMD_WRITE)	0b001 (MEM_HALF)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b1	0b000 (IMM_S)	0b0	0b0
sw	0b10 (CMD_WRITE)	0b010 (MEM_WORD)	0b01 (ALUOP1_RS1)	0b11 (ALUOP2_IMM)	0b1	0b1	0b000 (IMM_S)	0b0	0b0

Tabuľka 0.12 Zoznam inštrukcií skokov aj s ich požadovanými dekódovanými signálmi

Inštrukcia	<i>alu_function</i> [3:0]	<i>aluop1_mux</i> [1:0]	<i>aluop2_mux</i> [1:0]	<i>rs1_ren</i>	<i>rs2_ren</i>	<i>sel_imm</i> [2:0]	<i>reg_write</i>	<i>branch</i>	<i>jal</i>	<i>jalr</i>	ostatné signály
jal	0b0000 (ALU_ADD)	0b10 (ALUOP1_PC)	0b01 (ALUOP2_LINK)	0b0	0b0	0b011 (IMM_UJ)	0b1	0b0	0b1	0b0	0b0
jalr	0b0000 (ALU_ADD)	0b10 (ALUOP1_PC)	0b01 (ALUOP2_LINK)	0b1	0b0	0b100 (IMM_I)	0b1	0b0	0b0	0b1	0b0
beq	0b1000 (ALU_EQ)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b001 (IMM_SB)	0b0	0b1	0b0	0b0	0b0
bne	0b1001 (ALU_NE)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b001 (IMM_SB)	0b0	0b1	0b0	0b0	0b0
blt	0b1100 (ALU_LT)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b001 (IMM_SB)	0b0	0b1	0b0	0b0	0b0
bge	0b1101 (ALU_GE)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b001 (IMM_SB)	0b0	0b1	0b0	0b0	0b0
bltu	0b1110 (ALU_LTU)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b001 (IMM_SB)	0b0	0b1	0b0	0b0	0b0
bgeu	0b1111 (ALU_GEU)	0b01 (ALUOP1_RS1)	0b10 (ALUOP2_RS2)	0b1	0b1	0b001 (IMM_SB)	0b0	0b1	0b0	0b0	0b0

Tabuľka 0.13 Zoznam zvyšných inštrukcií aj s ich požadovanými dekódovanými signálmi

Inštrukcia	<i>aluop1_mux</i> [1:0]	<i>aluop2_mux</i> [1:0]	<i>sel_imm</i> [2:0]	<i>reg_write</i>	<i>ecall</i>	<i>ebreak</i>	<i>wfi</i>	<i>xret</i>	<i>fencei</i>	ostatné signály
lui	0b00 (ALUOP1_NONE)	0b11 (ALUOP2_IMM)	0b010 (IMM_U)	0b1	0b0	0b0	0b0	0b0	0b0	0b0
auipc	0b10 (ALUOP1_PC)	0b11 (ALUOP2_IMM)	0b010 (IMM_U)	0b1	0b0	0b0	0b0	0b0	0b0	0b0
mret	0b00 (ALUOP1_NONE)	0b00 (ALUOP2_NONE)	0b101 (IMM_Z)	0b0	0b0	0b0	0b0	0b1	0b0	0b0
wfi	0b00 (ALUOP1_NONE)	0b00 (ALUOP2_NONE)	0b101 (IMM_Z)	0b0	0b0	0b0	0b1	0b0	0b0	0b0
fence.i	0b00 (ALUOP1_NONE)	0b00 (ALUOP2_NONE)	0b101 (IMM_Z)	0b0	0b0	0b0	0b0	0b0	0b1	0b0
ebreak	0b00 (ALUOP1_NONE)	0b00 (ALUOP2_NONE)	0b101 (IMM_Z)	0b0	0b0	0b1	0b0	0b0	0b0	0b0
ecall	0b00 (ALUOP1_NONE)	0b00 (ALUOP2_NONE)	0b101 (IMM_Z)	0b0	0b1	0b0	0b0	0b0	0b0	0b0

# C DÁTOVÁ PAMÄŤ TYPU L1 CACHE

## C.1 Zoznam konfigurácií naplánovaných na kontrolu

Tabuľka 0.14 Naplánovaná kontrola jednotlivých konfigurácií dátovej pamäte typu L1 cache

Číslo konfigurácie	Aktívna úroveň resetu	Typ resetu	Povolenie dávkovej transakcie na rozhraní typu master	Veľkosť cache riadku [Bajt]	Dátová šírka pre rozhranie typu slave [Bit]	Dátová šírka pre rozhranie typu master [Bit]	Typ transakcie na rozhraní typu master	Šírka prenášaného slova na rozhraní typu master [Bit]
1	0	Synchrónny	Nie	16	32	32	SINGLE	32
2	0	Asynchrónny	Áno	4	8	8	WRAP4	8
3	1	Asynchrónny	Áno	8	16	8	WRAP8	8
4	1	Synchrónny	Áno	16	32	8	WRAP16	8
5	1	Asynchrónny	Áno	16	128	16	WRAP8	16
6	0	Synchrónny	Áno	16	32	16	WRAP8	16
7	0	Asynchrónny	Áno	32	8	16	WRAP16	16
8	0	Synchrónny	Áno	16	16	32	WRAP4	32
9	0	Asynchrónny	Áno	64	8	64	WRAP8	64
10	1	Synchrónny	Áno	256	64	128	WRAP16	128
11	1	Synchrónny	Áno	256	256	256	WRAP8	256

## C.2 Zoznam požiadaviek na rozhraní typu slave

Tabuľka 0.15 Vytvorené požiadavky protokolu AHB-lite na rozhraní typu slave

Názov požiadavku	Popis požiadavku
HRDATA_SIZE	Šírka dátovej zbernice určenej na čítanie HRDATA musí byť široká 8, 16, 32, 64, 128, 256, 512 alebo 1024 bitov.
HREADYOUT_DURING_ASYNC_RESET	Počas asynchrónneho resetu musí obvod typu slave vystaviť signál HREADYOUT do log. 1.
HREADYOUT_DURING_SYN_RESET	Počas synchrónneho resetu musí obvod typu slave vystaviť signál HREADYOUT do log. 1.
BUSY_RESPONSE	Obvod typu slave musí na prenos druhu BUSY odpovedať bez vloženia čakacieho cyklu a s odpoveďou OKAY.
IDLE_RESPONSE	Obvod typu slave musí na prenos druhu IDLE odpovedať bez vloženia čakacieho cyklu a s odpoveďou OKAY.

ERROR_2_CYCLES_RESPONSE	Chybná odpoveď prenosu obvodu typu slave vyžaduje 2 cykly. V prvom cykle musí obvod typu slave vystaviť signál HRESP do log. 1 (ERROR) a signál HREADYOUT do log. 0. V druhom cykle musí podržať signál HRESP v log. 1 a signál HREADYOUT vystaviť do log. 1.
CONTINUOUS_ERROR_2_CYCLE_RESPONSE	V prípade že obvod typu master pokračuje v prenose aj po chybnjej odpovedi a druhý prenos je tiež s chybnou odpoveď bez čakacieho cyklu, obvod typu slave musí dodržať nasledovný postup. V prvom cykle obvod typu slave vystavil signál HRESP do log. 1 (ERROR) a signál HREADYOUT do log. 0. V druhom cykle podržal signál HRESP v log. 1 a vystavil signál HREADYOUT do log. 1. V momente keď podrží signál HRESP v log. 1 aj tretí cyklus musí zase v tomto cykle vystaviť signál HREADYOUT do log. 0 a v štvrtom cykle podržať signál HRESP v log. 1 a vystaviť HREADYOUT do log. 1.
EXTENDED_DATA_PHASE	Obvod typu slave môže vložiť čakacie cykly iba do dátovej fázy prenosu.

### C.3 Zoznam požiadaviek na rozhraní typu master

Tabuľka 0.16 Vytvorené požiadavky protokolu AHB-lite na rozhraní typu master

Názov požiadavku	Popis požiadavku
HWDATA_SIZE	Šírka dátovej zbernice určenej na zápis HWDATA musí byť široká 8, 16, 32, 64, 128, 256, 512 alebo 1024 bitov.
HTRANS_DURING_ASYNC_RESET	Počas asynchrónneho resetu musí obvod typu master vystaviť signál HTRANS na typ IDLE.
HTRANS_DURING_SYNC_RESET	Počas synchronného resetu musí obvod typu master vystaviť signál HTRANS na typ IDLE.
MASTER_SIGNALS_DURING_ASYNC_RESET	Počas asynchrónneho resetu musí obvod typu master zabezpečiť aby adresa a kontrolne signály mali validnu hodnotu.
MASTER_SIGNALS_DURING_SYNC_RESET	Počas synchronného resetu musí obvod typu master zabezpečiť aby adresa a kontrolne signály mali validnu hodnotu.

WIDER_WRITE_TRANSFER_THAN_HWDATA	Obvod typu master nesmie nikdy zahájiť prenos zápisu ktorého šírka prenášaného slova indikovaná signálom HSIZE je väčšia ako šírka dátovej zbernice zápisu HWDATA.
WIDER_READ_TRANSFER_THAN_HRDATA	Obvod typu master nesmie nikdy zahájiť prenos čítania ktorého šírka prenášaného slova indikovaná signálom HSIZE je väčšia ako šírka dátovej zbernice čítania HRDATA.
HTRANS_AFTER_IDLE	Signál HTRANS môže po prenose typu IDLE nadobudnúť iba typ IDLE alebo NONSEQ.
HTRANS_AFTER_SINGLE_BURST	Obvod typu master nemôže vystaviť signál HTRANS na typ BUSY ani SEQ ako nasledujúci typ prenosu po transakcii o jednom prenose. Po transakcii o jednom prenose môže signál HTRANS nadobudnúť iba typy IDLE alebo NONSEQ.
REQUEST_STABLE_UNTIL_RESPONSE	V prípade prenosu NONSEQ alebo SEQ obvod typu master nemôže bez odpovede obvodu typu slave zmeniť adresu a kontrolné signály po tom ako už vystavil požiadavku. Obvod typu master musí držať adresu a kontrolné signály na rovnakej hodnote až pokiaľ obvod typu slave nevystaví buď signál HRESP alebo HREADYOUT do log. 1.
HWDATA_STABLE	V prípade predĺženej dátovej fáze zápisu prenosu NONSEQ alebo SEQ, obvod typu master musí držať hodnotu dát určených na zápis na rovnakej hodnote až kým obvod typu slave nevystaví buď signál HRESP alebo HREADYOUT do log. 1.
ALIGNED_ADDRESS	Adresy prenosov IDLE, NONSEQ a SEQ musia byť zarovnané podľa šírky prenášaného slova indikovaného signálom HSIZE.
WRAP4_NUMBER_OF_TRANSFERS	Kontrola počtu prenosov pri zaobalenej dávke o 4 prenosoch. V prípade že obvod typu master nezruší transakciu po chybnjej odpovedi od obvodu typu slave, je táto transakcia tvorená 1 prenosom typu NONSEQ a 3 prenosmi typu SEQ.

WRAP8_NUMBER_OF_TRANSFERS	Kontrola počtu prenosov pri zaobalenej dávke o 8 prenosoch. V prípade že obvod typu master nezruší transakciu po chybnej odpovedi od obvodu typu slave, je táto transakcia tvorená 1 prenosom typu NONSEQ a 7 prenosmi typu SEQ.
WRAP16_NUMBER_OF_TRANSFERS	Kontrola počtu prenosov pri zaobalenej dávke o 16 prenosoch. V prípade že obvod typu master nezruší transakciu po chybnej odpovedi od obvodu typu slave, je táto transakcia tvorená 1 prenosom typu NONSEQ a 15 prenosmi typu SEQ.
WRAP_SIZE_8_ADDR_CHECK	Kontrola adresy jednotlivých prenosov pri zaobalenej dávke o 4, 8 alebo 16 prenosoch a šírke prenášaného slova 8 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku.
WRAP_SIZE_16_ADDR_CHECK	Kontrola adresy jednotlivých prenosov pri zaobalenej dávke o 4, 8 alebo 16 prenosoch a šírke prenášaného slova 16 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku.
WRAP_SIZE_32_ADDR_CHECK	Kontrola adresy jednotlivých prenosov pri zaobalenej dávke o 4, 8 alebo 16 prenosoch a šírke prenášaného slova 32 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku.
WRAP_SIZE_64_ADDR_CHECK	Kontrola adresy jednotlivých prenosov pri zaobalenej dávke o 4, 8 alebo 16 prenosoch a šírke prenášaného slova 64 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku.
WRAP_SIZE_128_ADDR_CHECK	Kontrola adresy jednotlivých prenosov pri zaobalenej dávke o 4, 8 alebo 16 prenosoch a šírke prenášaného slova 128 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku.
WRAP_SIZE_256_ADDR_CHECK	Kontrola adresy jednotlivých prenosov pri zaobalenej dávke o 4, 8 alebo 16 prenosoch a šírke prenášaného slova 256 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku.

## C.4 Zoznam doplnených vlastností použitých ako predpoklady na rozhraní typu slave

Tabuľka 0.17 Prehľad doplnených vlastností použitých ako predpoklady na rozhraní typu slave

Názov vlastnosti	Popis vlastnosti
SET_MASTER_SIGNALS_DURING_ASYNC_RESET	Počas asynchrónneho resetu majú všetky vstupné signály hodnotu 0 okrem signálu HREADY ktorý je v log. 1.
SET_MASTER_SIGNALS_DURING_SYN_RESET	Počas synchronného resetu majú všetky vstupné signály hodnotu 0 okrem signálu HREADY ktorý je v log. 1.
MULTIPLEXING_HREADY	V prípade že je signál HSEL v log. 1, hodnota signálu HREADY musí byť rovnaká ako hodnota signálu HREADYOUT.
BUSY_TERMINATES_UNDEFINED_LENGTH_BURST	Iba v prípade inkrementujúcej transakcie nedefinovanej dĺžky môže po prenose typu BUSY nasledovať prenos NONSEQ alebo IDLE. Jedná sa o efektívne ukončenie inkrementujúcej transakcie o nedefinovanej dĺžke.
BUSY_REFLECT_NEXT_TRANSFER	Pri prenose typu BUSY musí adresa a kontrolné signály niesť hodnotu nasledujúceho prenosu.
INCR4_NUMBER_OF_TRANSFERS	Kontrola počtu prenosov pri inkrementujúcej dávke o 4 prenosoch. V prípade že obvod typu master nezruší transakciu po chybnjej odpovedi od obvodu typu slave, je táto transakcia tvorená 1 prenosom typu NONSEQ a 3 prenosmi typu SEQ.
INCR8_NUMBER_OF_TRANSFERS	Kontrola počtu prenosov pri inkrementujúcej dávke o 8 prenosoch. V prípade že obvod typu master nezruší transakciu po chybnjej odpovedi od obvodu typu slave, je táto transakcia tvorená 1 prenosom typu NONSEQ a 7 prenosmi typu SEQ.

<p>INCR16_NUMBER_OF_TRANSFERS</p>	<p>Kontrola počtu prenosov pri inkrementujúcej dávke o 16 prenosoch. V prípade že obvod typu master nezruší transakciu po chybnjej odpovedi od obvodu typu slave, je táto transakcia tvorená 1 prenosom typu NONSEQ a 15 prenosmi typu SEQ.</p>
<p>ADDR_CHECK_INCR_BURST</p>	<p>Kontrola adresy pri inkrementujúcej dávke. Adresa nasledujúceho prenosu v dávke musí byť inkrementovaná adresa predchádzajúceho prenosu o veľkosť šírky prenášaného slova.</p>
<p>CHECK_1KB_ADDR_BOUNDARY_INCR</p>	<p>Obvod typu master nemôže zahájiť inkrementujúcu dávku v prípade že by počas nej adresa prekročila hranicu 1 kilobajtu.</p>
<p>WRAP4_SIZE_8_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 4 prenosoch a veľkosti prenášaného slova 8 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 32 bitov.</p>
<p>WRAP4_SIZE_16_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 4 prenosoch a veľkosti prenášaného slova 16 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 64 bitov.</p>
<p>WRAP4_SIZE_32_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 4 prenosoch a veľkosti prenášaného slova 32 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 128 bitov.</p>

<p>WRAP4_SIZE_64_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 4 prenosoch a veľkosti prenášaného slova 64 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 256 bitov.</p>
<p>WRAP4_SIZE_128_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 4 prenosoch a veľkosti prenášaného slova 128 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 512 bitov.</p>
<p>WRAP4_SIZE_256_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 4 prenosoch a veľkosti prenášaného slova 256 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 1024 bitov.</p>
<p>WRAP8_SIZE_8_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 8 prenosoch a veľkosti prenášaného slova 8 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 64 bitov.</p>
<p>WRAP8_SIZE_16_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 8 prenosoch a veľkosti prenášaného slova 16 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 128 bitov.</p>
<p>WRAP8_SIZE_32_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 8 prenosoch a veľkosti prenášaného slova 32 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 256 bitov.</p>

<p>WRAP8_SIZE_64_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 8 prenosoch a veľkosti prenášaného slova 64 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 512 bitov.</p>
<p>WRAP8_SIZE_128_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 8 prenosoch a veľkosti prenášaného slova 128 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 1024 bitov.</p>
<p>WRAP8_SIZE_256_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 8 prenosoch a veľkosti prenášaného slova 256 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 2048 bitov.</p>
<p>WRAP16_SIZE_8_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 16 prenosoch a veľkosti prenášaného slova 8 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 128 bitov.</p>
<p>WRAP16_SIZE_16_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 16 prenosoch a veľkosti prenášaného slova 16 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 256 bitov.</p>
<p>WRAP16_SIZE_32_ADDR_CHECK</p>	<p>Kontrola adresy pri zaobalenej dávke o 16 prenosoch a veľkosti prenášaného slova 32 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 512 bitov.</p>

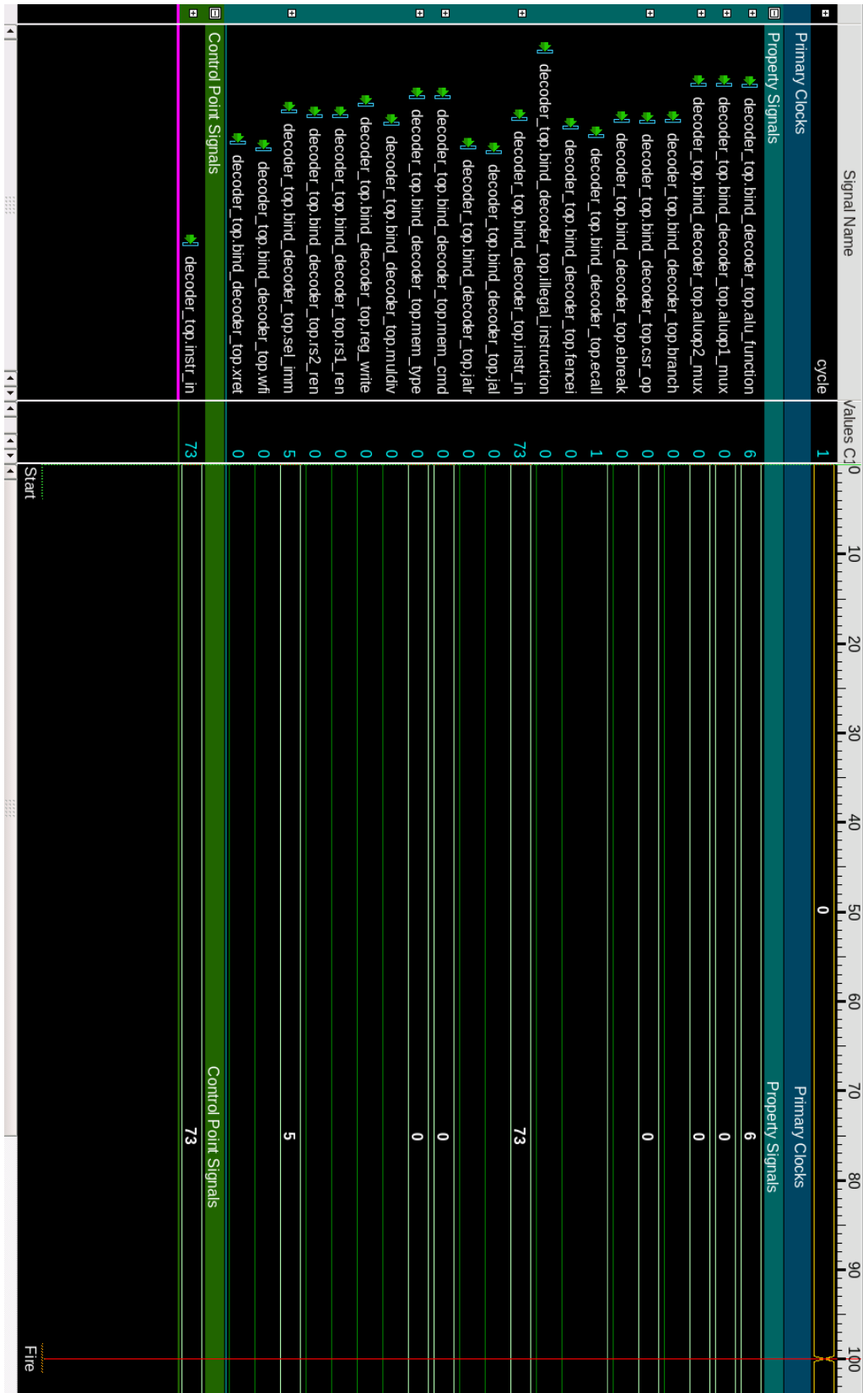
WRAP16_SIZE_64_ADDR_CHECK	Kontrola adresy pri zaobalenej dávke o 16 prenosoch a veľkosti prenášaného slova 64 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 1024 bitov.
WRAP16_SIZE_128_ADDR_CHECK	Kontrola adresy pri zaobalenej dávke o 16 prenosoch a veľkosti prenášaného slova 128 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 2048 bitov.
WRAP16_SIZE_256_ADDR_CHECK	Kontrola adresy pri zaobalenej dávke o 16 prenosoch a veľkosti prenášaného slova 256 bitov. Adresy jednotlivých prenosov nesmú prekročiť adresný priestor vytvorený pre celú dávku. Takýto adresný priestor ma veľkosť 4096 bitov.

## C.5 Zoznam doplnených vlastností použitých ako predpoklady na rozhraní typu master

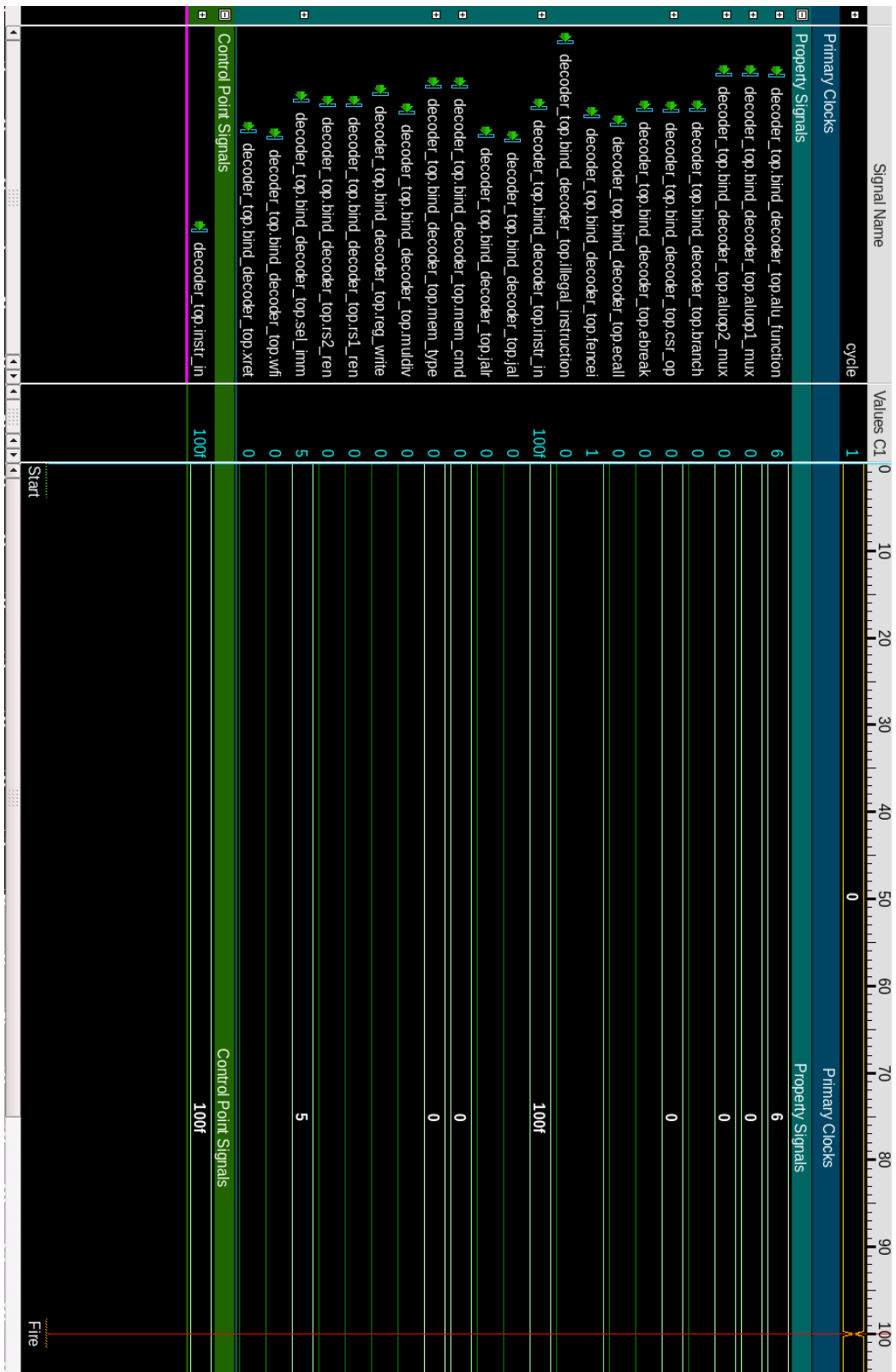
Tabuľka 0.18 Prehľad doplnených vlastností použitých ako predpoklady na rozhraní typu master

Názov vlastnosti	Popis vlastnosti
SET_SLAVE_SIGNALS_DURING_ASYNC_RESET	Počas asynchrónneho resetu majú vstupné signály HRESP a HRDATA hodnotu 0.
SET_SLAVE_SIGNALS_DURING_SYNC_RESET	Počas synchronného resetu majú vstupné signály HRESP a HRDATA hodnotu 0.

# D VÝSLEDKY VERIFIKÁCIE



Obrázok 0.1 Vygenerovaný protipríklad formálneho tvrdenia inštrukcie ecall



Obrázok 0.2 Vygenerovaný protipríklad formálneho tvrdenia inštrukcie fence.i