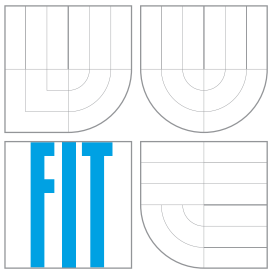
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

FUNKČNÍ VERIFIKACE ROBOTICKÉHO SYSTÉMU POMOCÍ UVM

FUNCTIONAL VERIFICATION OF ROBOTIC SYSTEM USING UVM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. STANISLAV KRAJČÍR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARCELA ŠIMKOVÁ

BRNO 2015

Abstrakt

Jedním z aktuálně nejvíce využívaných přístupů pro verifikaci hardwarových systémů je funkční verifikace. Tato diplomová práce se zabývá tvorbou verifikačního prostředí s využitím metodiky UVM (Universal Verification Methodology) pro ověření korektnosti řídicí jednotky robotického systému s cílem odstranění funkčních chyb z její implementace. Teoretická část práce popisuje základní informace z oblasti funkční verifikace, metody tvorby verifikačního prostředí, jazyk SystemVerilog a problematiku zajištění odolnosti systémů proti poruchám. Následující část práce se zaměřuje na návrh verifikačního prostředí, jeho implementaci a na tvorbu testů sloužících k ověření korektnosti řídicí jednotky. V závěru práce jsou diskutovány a zhodnoceny dosažené výsledky verifikace.

Abstract

One of the currently most used approaches for verification of hardware systems is functional verification. This master thesis describes design and implementation of a verification environment using UVM (Universal Verification Methodology) methodology for verifying the correctness of the robot controller in order to eliminate functional errors and faults of its implementation. The theoretical part of the thesis describes the basic information about functional verification, methodologies for creating verification environments, the SystemVerilog language and fault tolerance methodologies. The next part of thesis focuses on the design of the verification environment, its implementation and the creation of tests used to verify the correctness of the robot controller. Results of verification are discussed and evaluated in the conclusion of this work.

Klíčová slova

funkční verifikace, SystemVerilog, UVM, odolnost proti poruchám

Keywords

functional verification, SystemVerilog, UVM, fault tolerance

Citace

Stanislav Krajčír: Funkční verifikace robotického systému
pomocí UVM, diplomová práce, Brno, FIT VUT v Brně, 2015

Funkční verifikace robotického systému pomocí UVM

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pani Ing. Marcely Šimkové. Další informace mi poskytl konzultant Ing. Jakub Podivínsky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Bc. Stanislav Krajčír
25. mája 2015

Poděkování

Chci poděkovat vedoucí své diplomové práce, Ing. Marcele Šimkovej za odbornou pomoc, cenné rady, a konzultace spojené s vedením práce. Děkuji také konzultantovi této práce, Ing. Jakubovi Podivínskému za konzultace a odbornou pomoc.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

V období posledných rokov sa výpočtové systémy stali priam až neoddeliteľnou súčasťou ľudských životov. Technologické napredovanie spolu s trendom rastu hustoty integrácie hradiel tranzistorov nám dovoľuje vytvárať čoraz komplexnejšie systémy, ktoré nachádzajú široké uplatnenie. Medzi tieto zariadenia patria aj špecializované systémy určené na využitie v extrémnych podmienkach. Spolu s nárastom zložitosti týchto zariadení sa do popredia dostávajú požiadavky na posúdenie ich korektnosti, spoľahlivosti a zabezpečenia ich správnej funkcionality. Medzi takéto špecializované systémy patria zariadenia vysielané do vesmíru, kde sú vystavené extrémnym podmienkam vo forme kozmického žiarenia. Z hľadiska efektívneho využívania zdrojov je veľmi dôležité mať možnosť overiť funkčnosť a mieru odolnosti týchto zariadení proti poruchám ešte predtým, než im budú vystavené v skutočnom prostredí. Jedným zo spôsobov je využitie softvérovej simulácie. V rámci simulačného prostredia sa do zariadenia injektujú poruchy s cieľom overenia miery odolnosti daného zariadenia proti poruchám. Cieľom tejto práce je prezentovať možnosti urýchlenia a zefektívnenia celého uvedeného procesu s využitím funkčnej verifikácie. Ide o inovatívne riešenie, nakoľko tento spôsob využitia funkčnej verifikácie nebol zatiaľ nikde realizovaný.

Táto práca sa zaoberá tvorbou verifikačného prostredia za účelom funkčnej verifikácie riadiacej jednotky robotického systému. Riadiaca jednotka, ktorá je predmetom overenia, bola vytvorená s ohľadom na zabezpečenie proti poruchám. Výsledky tejto práce budú priamo použité v rámci výskumu odolnosti systémov proti poruchám, ktorý prebieha na Ústave počítačových systémov Fakulty informačných technológií VUT v Brne. Tvorba verifikačného prostredia tvorí jednu z dielčich činností, ktorých cieľom je v budúcnosti vytvorenie plne automatizovaného systému na overovanie odolnosti riadiacej jednotky proti poruchám, ktoré budú do nej umelo injektované.

Text práce je rozdelený do niekoľkých kapitol. Kapitola 2 sa venuje motivácii k tvorbe tejto práce a predstavuje čitateľovi celkový koncept výskumného projektu a jasne vymedzuje oblasť, ktorej sa v rámci konceptu venuje samotná práca. Kapitola 3 obsahuje teoretické informácie, ktoré čitateľovi objasnia základné pojmy z oblasti funkčnej verifikácie, návrhu verifikačného prostredia, jazyka SystemVerilog a problematiku zabezpečenia odolnosti systémov proti poruchám. Kapitola 4 sa venuje predstaveniu základnej funkcionality riadiacej jednotky robotického systému, ktorá je predmetom verifikácie. Obsahuje popis jednotlivých blokov, z ktorých je jednotka zostavená a objasňuje ich funkcionalitu. V rámci kapitoly 5 je uvedený návrh samotného verifikačného prostredia. Kapitola 6 obsahuje implementáciu verifikačného prostredia a návrh testov. Súčasťou kapitoly je overenie funkčnosti prostredia a analýza dosiahnutých výsledkov. Koncept budúceho využitia verifikačného prostredia ako súčasti automatizovaného systému na overovanie odolnosti riadiacej jednotky proti poruchám je predstavený v kapitole 7. Kapitola 8 obsahuje záverečné zhrnutie práce.

Kapitola 2

Motivácia

Táto kapitola slúži čitateľovi k objasneniu dôvodov tvorby tejto práce a predstavuje základný koncept činnosti jej návrhu a následnej implementácie.

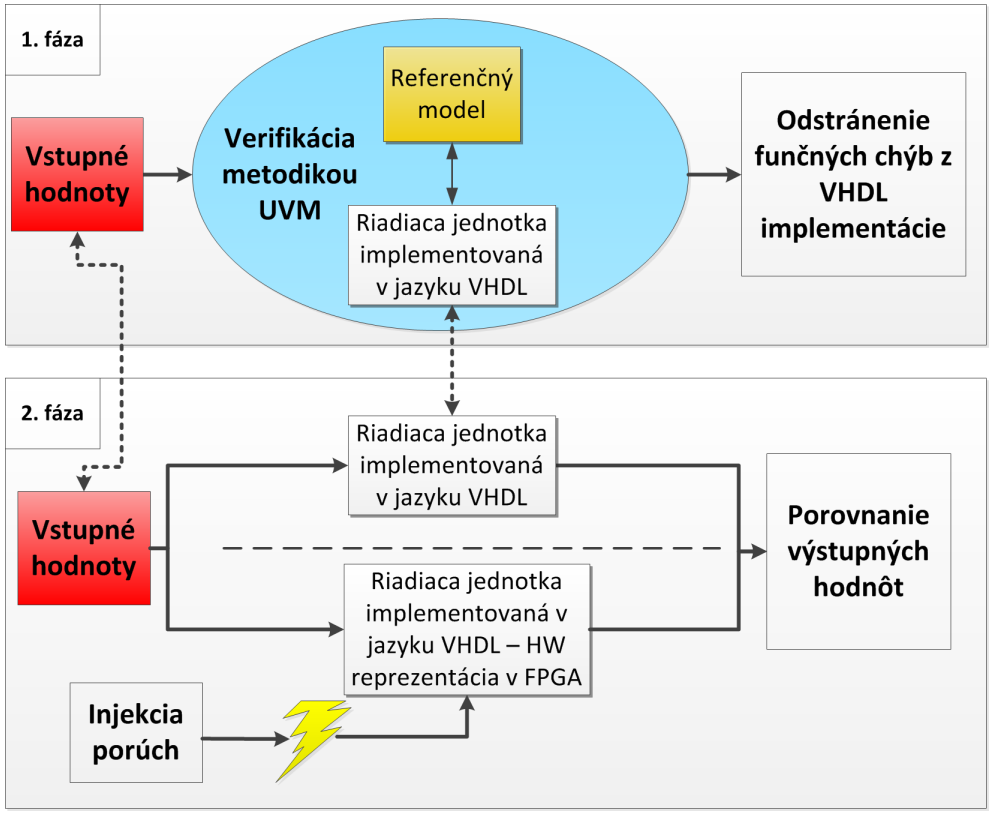
Systémy, ktoré sa vysielajú do vesmíru bývajú vystavené najrôznejším formám kozmického žiarenia. Cieľom rôznych výskumných organizácií je zabezpečenie správnej funkcionality týchto zariadení aj v takýchto extrémnych podmienkach. Hľadaním odpovedí na otázky, ktoré riešia úlohy tohto typu sa zaoberá oblasť zabezpečenia odolnosti systémov proti poruchám.

Kozmické žiarenie (protóny, neutróny a častice alfa), ktorému sú systémy vo vesmíre vystavené, obsahuje veľké množstvo energie, ktoré je schopné ovplyvniť fungovanie elektromechanických zariadení. V rámci skúmania odolnosti systémov proti poruchám je veľmi dôležité overiť aké účinky môže žiarenie na zariadenie mať a aké chyby môže spôsobiť. Z hľadiska efektívneho využívania materiálov a finančných zdrojov je veľmi dôležité mať možnosť overiť mieru odolnosti systému proti poruchám predtým, než bude týmto poruchám systém vystavený v reálnom nasadení.

Existuje niekoľko využívaných prístupov k overeniu miery odolnosti systému proti poruchám. Jedným z nich je vytvorenie podobných podmienok ako sú vo vesmíre tu na Zemi. Zariadenie je v špeciálnej miestnosti vystavené týmto podmienkam, pričom sa sleduje ich vplyv na jeho funkčnosť. Na základe získaných informácií sa zariadenie alebo jeho časti upravujú tak, aby boli schopné týmto extrémnym podmienkam odolať. Toto riešenie je však finančne nákladné. Druhým možným prístupom je vytvorenie softvérovej simulácie, v rámci ktorej sa do daného zariadenia umelo injektujú rôzne typy porúch. Tento spôsob je momentálne predmetom výskumu, ktorý prebieha na Ústave počítačových systémov (ďalej len ÚPSY) Fakulty informačných technológii VUT v Brne. V rámci výskumnej skupiny Diagnostika bola na ÚPSY vytvorená riadiaca jednotka robotického systému určeného pre samočinný pohyb v bludisku [9, 8].Táto riadiaca jednotka bola navrhnutá tak, že obsahuje mechanizmy, ktoré zabezpečujú odolnosť proti poruchám (napr. ide o mechanizmy *Triple Modular Redundancy - TMR*, *N-Modular Redundancy - NMR* a *Duplexné zabezpečenie*). Podrobný popis zabezpečovacích mechanizmov sa nachádza v kapitole 3.5. V rámci simulačného prostredia sa do riadiacej jednotky injektujú poruchy a sleduje sa, ako na ne bude riadiaca jednotka reagovať. Tento spôsob overovania je z časového hľadiska zdĺhavý, pretože je potrebné manuálne spustiť celý simulačný proces a sledovať, či robot plní svoju funkciu a správne prechádza bludiskom zo štartovacej pozície do cieľovej pozície.

Keďže sledovanie vplyvu porúch na riadiacu jednotku robota v rámci procesu overovania odolnosti jednotky proti poruchám v simulácii bolo časovo zdĺhavé, vzišiel návrh celý proces zautomatizovať, a tým pádom ho výrazne zrýchliť a zefektívniť. Pre realizáciu toh-

to návrhu sme sa rozhodli použiť prístup, ktorý na základe našich informácií zatiaľ nebol za týmto účelom nikdy použitý. Jedná sa o využitie funkčnej verifikácie. Výhody využitia tohto prístupu budú uvedené v ďalšej časti tejto práce. Cieľom tejto práce je vytvoriť verifikačné prostredie, pričom v rámci práce budú využité už existujúce časti výskumu z ÚPSY. Schematický princíp riešenia s využitím verifikačného prostredia je znázornený na obrázku 2.1.



Obr. 2.1: Princíp fungovania verifikačného prostredia.

Samotný proces je možné rozdeliť na dve hlavné fázy:

- **Prvá fáza** tvorí základ tejto práce (návrh a implementácia verifikačného prostredia). Jej cieľom je overiť korektnosť riadiacej jednotky robota. Za týmto účelom bude vytvorené verifikačné prostredie v jazyku SystemVerilog s využitím metodiky UVM (*Universal Verification Methodology*). Funkcionalita riadiacej jednotky bude overovaná voči funkcionalite tzv. referenčného modelu. Referenčný model je súčasťou verifikačného prostredia a implementuje rovnakú činnosť ako riadiaca jednotka. Jeho implementácia je nezávislá na implementácii riadiacej jednotky, vychádza iba z rovnakej špecifikácie funkcionality. Na základe vstupných hodnôt, ktoré budú aplikované ako na vstup riadiacej jednotky, tak aj na vstup referenčného modelu, budeme overovať korektnosť jednotky za účelom odstránenia funkčných chýb z jej implementácie. Funkčná verifikácia disponuje tzv. analýzou pokrytia (angl. *coverage analysis*), ktorá nám poskytuje informácie o tom, ktoré časti riadiacej jednotky boli počas verifikácie preverené. Na základe tejto informácie vieme z množiny vstupných hodnôt vybrať iba tie hodnoty, pomocou ktorých je dosiahnuté úplné pokrytie riadiacej jednotky. Ako metriku pokrytia je možné zvoliť napr. pokrytie kódu. Táto vyselektovaná

množina vstupných hodnôt je pre nás veľmi dôležitá, nakoľko bude tvoriť základ pre druhú fázu. Prvá fáza bude bežať kompletne v softvérovom simulačnom prostredí.

- **Druhá fáza** priamo nadväzuje na výsledky prvej fázy a bude v budúcnosti realizovaná v rámci výskumu na ÚPSY. Samotná realizácia druhej fázy nie je predmetom tejto práce. Nakoľko však priamo nadväzuje na výsledky prvej fázy, tak je tu uvedená z dôvodu, aby čitateľ získal celkový prehľad o riešení problematiky, ktorý je predmetom výskumu na ÚPSY. Jej cieľom je verifikácia riadiacej jednotky, ktorá bude umiestnená v hardvérovom obvode FPGA (*Field-Programmable Gate Array*), do ktorej budú umelo injektované poruchy. Ako referenčný model bude využitá riadiaca jednotka robota, ktorú sme si pripravili v prvej fáze. Teda táto riadiaca jednotka bude odladená a nebude obsahovať funkčné chyby. Vstupy bude tvoriť vyselektovaná množina vstupných hodnôt o ktorých vieme, že poskytujú úplné pokrytie. Tieto vstupné hodnoty budú privedené na vstup referenčnej riadiacej jednotky, ktorá bude fungovať v softvérovej simulácii a na vstup riadiacej jednotky, ktorá bude umiestnená v obvode FPGA, teda bude realizovaná v hardvéri. Do hardvérovej jednotky zároveň budú injektované poruchy a výstupy medzi týmito dvomi jednotkami budú porovnávané. Na základe porovnania bude možné určiť, či injektovaná porucha ovplyvnila funkčnosť riadiacej jednotky.

Kapitola 3

Teoretický úvod

Táto kapitola sa venuje úvodu do problematiky funkčnej verifikácie a overovania odolnosti systémov proti poruchám. Obsahuje základné informácie, ktorých objasnenie je nevyhnutné k pochopeniu širších súvislostí, ktoré budú predstavené v rámci návrhu a implementácie samotného verifikačného prostredia robotického systému.

3.1 SystemVerilog

SystemVerilog je programovací jazyk, ktorý radíme do skupiny tzv. HDVL jazykov [11], teda jazykov pre popis a verifikáciu hardvérových obvodov (angl. *Hardware Description and Verification Language).* Jazyk SystemVerilog je štandardizovaný (aktuálna verzia štandardu je IEEE 1800-2012 [2]) a vznikol rozšírením jazyka Verilog o konštrukcie, ktoré umožnili zvýšiť efektívnosť procesu simulácie a verifikácie. Medzi tieto konštrukcie patria napríklad:

- **nové dátové typy:**
  - fronty (angl. *queues*), dynamické a asociatívne polia: umožňujú zníženie pamäťových nárokov, vstavaná podpora pre funkcie vyhľadávania a zoraďovania,
  - dvoj-stavové typy (angl. *two state types*): zníženie pamäťových nárokov a zvýšenie výkonnosti simulátora,
  - triedy a štruktúry: podpora pre abstraktné dátové štruktúry,
  - reťazce: vstavaná podpora pre využitie textových reťazcov,
  - vymenované (angl. *enumerated*) dátové typy: umožňujú jednoduchší zápis a porozumenie zapísaného kódu.
- **objektovo-orientované programovanie** (angl. *object-oriented programming*). Využitie princípov objektovo-orientovaného programovania ako napr. dedičnosť, zapúzdrenie, návrhové vzory, polymorfizmus, atď. umožňuje zvýšenie produktivity pri verifikácii rozsiahlejších obvodov najmä vďaka možnosti opätovného využitia predtým vytvorených verifikačných komponentov.
- **spolupráca s ostatnými programovacími jazykmi.** SystemVerilog umožňuje volanie funkcií, ktoré boli zapísané v iných programovacích jazykoch (primárne v jazykoch C a C++). Využíva pritom rozhranie nazývané SystemVerilog DPI (angl. *Direct Programming interface*). Hodnoty argumentov a výsledky funkcií je možné prenášať priamo. Rovnako tak môžu funkcie z iných programovacích jazykov volať funkcie z jazyka

SystemVerilog, čím môžu získať prístup k simulačnému času a udalostiam. Pre volanie funkcií z jazyka SystemVerilog je potrebné, aby boli dané funkcie preložené do formy tzv. zdieľaných knižníc (pre operačné systémy Linux a Unix - *„Shared Objects - .so“*, pre operačné systémy od spoločnosti Microsoft - *„Dynamic Link Library - .dll“*). Preklad musí byť vykonaný na rovnakej architektúre a verzii operačného systému, na ktorej bude spustené verifikačné prostredie. V rámci odovzdávania vstupných a výstupných hodnôt medzi jazykom SystemVerilog a programovacím jazykom C/C++ je potrebné dodržať typovú kompatibilitu premenných. Tabuľka 3.1 obsahuje dátové typy jazyka SystemVerilog, ktoré sú priamo kompatibilné s dátovými typmi jazyka C. Jednotlivé funkcie, ktoré budú volané z jazyka SystemVerilog je potrebné deklarovať. V rámci deklarácie sa rozlišujú dva typy funkcií: „**čisté**“ (angl. *pure*) a „**kontextové**“ (angl. *context*). „Kontextové“ funkcie zachovávajú kontext v rámci jednotlivých volaní. Umožňujú teda uchovávať informácie napr. s využitím statických a globálnych premenných. „Čistá“ funkcia musí spĺňať nasledujúce vlastnosti:

1. Návratová hodnota závisí iba od vstupných parametrov funkcie.
2. Typ návratovej hodnoty nesmie byť typu „*void*“.
3. V prípade, že nie je potrebná návratová hodnota danej funkcie alebo ak pre dané vstupné argumenty funkcie existuje už vypočítaná hodnota, funkcia nemusí byť z prostredia jazyka SystemVerilog vyvolaná.
4. Nesmie priamo alebo nepriamo vykonávať žiadne z nasledujúcich operácií:
   (a) Manipulácia so súbormi (vytváranie, otváranie, čítanie, zápis, ...),
   (b) Čítanie alebo zápis do systémových premenných (angl. *enviroment variables*), zdieľanej pamäte alebo sieťových socketov.
   (c) Čítanie alebo zápis globálnych a statických premenných.

SystemVerilog	C
byte	char
int	int
longint	long int
shortint	short int
real	double
shortreal	float
chandle	void*
string	char*

Tabuľka 3.1: Typová kompatibilita premenných jazyka SystemVerilog a jazyka C.

3.2 Simulácia a jednoduché testovacie prostredie obvodu

Simulácia a jednoduché testovacie prostredie (angl. *testbench*) sú jednou zo základných metód využívaných pre overovanie korektnosti číslicových obvodov. Číslicový systém, ktorý je predmetom overenia, je v prvom kroku prevedený na softvérový model. Nad daným softvérovým modelom sa následne vykonávajú experimenty za účelom zistenia, či sa v danom modeli nenachádzajú chyby. Vstupné stimuly sú vytvárané priamo verifikačným inžinierom,

v rámci testovacieho prostredia. Výstupné hodnoty zo simulačného prostredia sú následne kontrolované a porovnávané s referenčnými hodnotami, ktoré sa očakávajú na výstupe. Tento spôsob overovania funkčnosti dokáže odhaliť skryté chyby, ktoré sa v obvode nachádzajú, nedokáže však potvrdiť, že obvod už žiadne chyby neobsahuje. To znamená, že nedokazuje korektnosť overovaného číslicového systému.

S narastajúcou zložitosťou verifikovaných systémov rastie aj počet vstupných stimulov, ktoré je potrebné manuálne vytvoriť a aplikovať na testovaný obvod. Simulácia zložitých systémov je preto časovo náročnou úlohou, kedy je verifikačný inžinier obmedzený výkonom simulačného softvéru, ktorý je závislý od dostupných hardvérových prostriedkov.

Na základe vyššie uvedených poznatkov je možné konštatovať, že simulácia sa využíva iba pre verifikáciu menších častí verifikovaných obvodov.

3.3 Funkčná verifikácia

Funkčná verifikácia je založená na simulácii, pričom využíva ďalšie prídavné techniky za účelom zefektívnenia verifikačného procesu a rozšírenia jednoduchého testovacieho prostredia. Medzi tieto techniky patrí:

- **generovanie náhodných vstupných stimulov** *(*angl. *constrained-random stimulus generation)*: Pri verifikácii obvodu je proces tvorby verifikačných scenárov veľmi náročnou úlohou najmä pri verifikácii väčších obvodov. Stimuly sú generované na základe sady obmedzujúcich podmienok *(*angl. *constraints)*, ktoré slúžia pre špecifikáciu hodnôt, ktoré môžu stimuly nadobúdať. Týmto spôsobom je možné sa zamerať na špecifickú oblasť obvodu, ktorú je potrebné verifikovať.

- **verifikácia riadená pokrytím** (angl. *coverage-driven verification*). Počas jednotlivých simulačných behov sú zaznamenávané štatistiky, ktoré sa následne využívajú k interpretácii toho, ktoré stavy verifikovaného systému boli preverené. Existuje viac typov pokrytí, ktoré sa v štatistikách sledujú: funkčné pokrytie (angl. *functional coverage*), pokrytie kódu (angl. *code coverage*), pokrytie ciest (angl. *path coverage*), pokrytie konečných automatov (angl. *FSM coverage*) a iné.

- **verifikácia založená na formálnych tvrdeniach** (angl. *assertion-based verification*). Pravdivosť formálneho tvrdenia je vyhodnocovaná počas celého simulačného behu v každom kroku. Tieto formálne tvrdenia môžu byť použité za účelom kontroly rôznych prvkov obvodu, ako napr. vstupno-výstupných rozhraní, kontroly kríženia hodinových signálov atď. Tieto formálne tvrdenia plnia monitorovaciu úlohu a medzi prínosy ich využitia patrí napr. okamžité rozpoznanie miesta v obvode, kde bolo formálne tvrdenie porušené.

- **samo-kontrolné mechanizmy** (angl. *self-checking mechanisms*). Podľa špecifikácie verifikovaného systému sú vstupné stimuly transformované na výstupné nezávisle od vykonanej implementácie. Následne sa vykoná automatická kontrola reálnych výstupov verifikovaného systému s predpokladanými výstupmi. Medzi techniky samokontrolných mechanizmov patrí využitie referenčného modelu (anglické označenie *scoreboarding, offline checking*).

3.4 Proces verifikácie

Verifikácia je komplexný proces, ktorého cieľom je zabezpečiť, aby verifikovaný obvod (angl. *DUT - Design under test*) vo výsledku spĺňal všetky požiadavky, ktoré boli na neho kladené v jeho špecifikácii. Je to proces, ktorý typicky prebieha súbežne s návrhom obvodu.

3.4.1 Verifikačný plán

Verifikačný plán je dokument, ktorý obsahuje postupy a kroky, ktoré je potrebné vykonať v procese verifikácie [3]. Špecifikuje, ako má byť verifikovaný obvod overený už v počiatočnej fáze návrhu a pomáha odhaľovať chyby, ktoré mohli nastať v prvotných fázach návrhu verifikačných postupov. Určuje metriky ukončenia verifikácie a špecifikuje zdroje, ktoré bude potrebné pre verifikáciu zabezpečiť (výpočtové, ľudské, odhad ceny). Samotný verifikačný plán obsahuje nasledujúce časti:

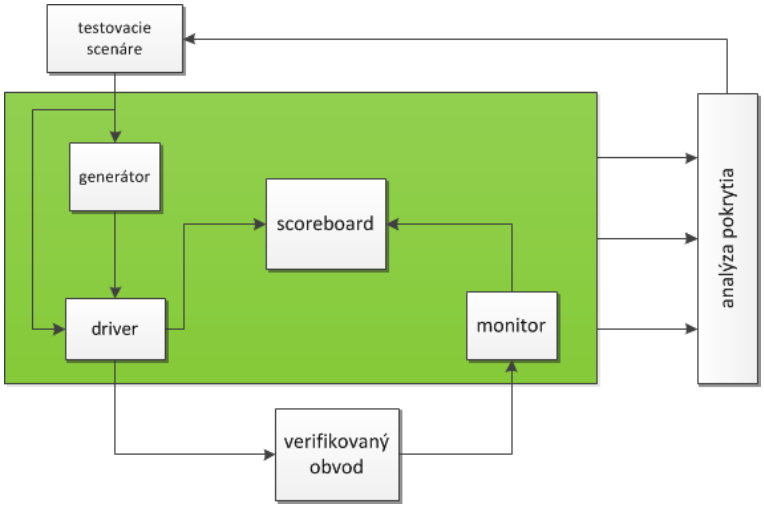
- **Verifikačné testy aplikované na návrh obvodu.** Testy sú určené na verifikáciu funkčnosti návrhu podľa špecifikácie. Sú to testy, ktoré majú byť aplikované na najvyššiu úroveň (angl. *top-level*) a na jednotlivé pod-bloky návrhu.
- **Verifikačné prostredie pre verifikovanú jednotku.** Obsahuje definície použitých verifikačných jazykov, štruktúru verifikačného prostredia (tiež označované v literatúre ako *„testbench“*, ide o pokročilú verziu testovacieho prostredia) a špeciálne inštrukcie. Táto štruktúra zahŕňa jednotlivé komponenty verifikovanej jednotky (na úrovni rozhraní) a balíky (angl. *packages*) na deklaračnej úrovni.
- **Validačné prostredie pre verifikovanú jednotku.** Obsahuje popis verifikovaného a očakávaného správania obvodu, postupy pre hlásenie chýb a typy detekovaných chýb.

3.4.2 Tvorba verifikačného prostredia

Úlohou verifikačného prostredia je určiť, či verifikovaný obvod spĺňa korektnosť vzhľadom k špecifikácii. Postup tvorby je nasledovný:

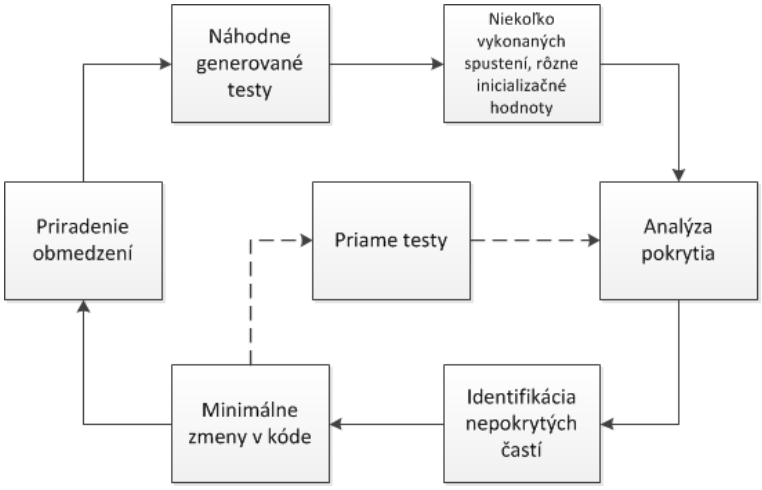
1. Vytvorenie vstupných stimulov. (V literatúre sa často stimul označuje ako vstupná transakcia, pretože verifikácia pracuje na transakčnej úrovni. Transakcia je v podstate dátová štruktúra, ktorá zaobaluje stimul pre verifikačné prostredie.)
2. Aplikácia stimulov na verifikovaný obvod.
3. Uloženie výstupných hodnôt.
4. Porovnanie skutočných výstupov s referenčnými výstupmi.
5. Zhodnotenie postupu vzhľadom k cieľom verifikácie.

Hlavné časti verifikačného prostredia znázorneného na nižšie uvedenom diagrame (obr. 3.1) sú nasledujúce:



Obr. 3.1: Verifikačné prostredie.

- **Testovacie scenáre** - obsahujú informácie o obmedzujúcich podmienkach (angl. *constraints*) pre generátor.
- **Generátor** - vytvára stimuly a odosiela ich do jednotky driver.
- **Driver** - stimuly, ktoré obdržal delí na konkrétne signály, ktoré odosiela na vstup verifikovaného obvodu a na vstup jednotky Scoreboard.
- **Monitor** - z jednotlivých výstupov verifikovanej jednotky vytvára transakcie (vyššie dátové štruktúry), ktoré odosiela na vstup jednotky Scoreboard.
- **Scoreboard** - implementuje referenčný model, ktorý sa v literatúre označuje aj ako „prediktor“. Referenčný model spracováva vstupné hodnoty z jednotky „driver“ podobným spôsobom, ako verifikovaný obvod (DUT) a vytvára referenčné výstupy. Následne porovnáva výstupné hodnoty z verifikovaného obvodu (DUT) získané z jednotky „monitor“ s týmito referenčnými výstupmi. V prípade, ak sa nezhodujú, hlási chybu.
- **Analýza pokrytia** - zbiera a analyzuje informácie o vykonaných verifikačných behoch, na základe ktorých generuje správu o pokrytí. Na základe tejto správy sa vytvárajú ďalšie testovacie scenáre, ktoré slúžia k testovaniu tých častí obvodu, ktoré neboli ešte verifikované. Princíp analýzy pokrytia je znázornený na nasledujúcom obrázku (3.2).



Obr. 3.2: Analýza pokrytia.

3.4.3 Verifikačné metodiky

Verifikačné metodiky špecifikujú ako vytvárať znovu použiteľné a jednoducho rozšíriteľné verifikačné prostredie v jazyku SystemVerilog. Umožňujú použitie predpripravených komponentov verifikačného prostredia, ktoré sú definované v podobe knižníc. Medzi najznámejšie verifikačné metodiky patria:

- **Verification Methodology Manual (VMM)** - jedna z prvých úspešne využívaných metodík. Obsahovala veľký počet odporúčaní pre tvorbu znovu použiteľného verifikačného prostredia v jazyku SystemVerilog. Bola vytvorená spoločnosťou Synopsys. Využívala výhody objektovo-orientovaného prístupu, funkčného pokrytia, sady obmedzení, atď., k vytváraniu škálovateľných a znovu použiteľných verifikačných prostredí. Prínos metodiky VMM zohral dôležitú úlohu v neskoršom vytvorení metodiky UVM.

- **Open Verification Methodology (OVM) [5]** - knižnica objektov a funkcií určených na generovanie vstupných stimulov, zberu dát a kontrolu celého procesu verifikácie. Umožňuje rýchle a jednoduché vytváranie znovu použiteľných verifikačných prostredí s vyžitím komunikácie na úrovni transakcií a funkčného pokrytia. Bola vytvorená spoločnosťami Mentor Graphics a Cadence. Metodika OVM prispela veľkým podielom k vývoju jej nasledovníka - metodiky UVM.

- **Universal Verification Methodology (UVM) [1]** - knižnica jazyka SystemVerilog, ktorá je distribuovaná pod licenciou *„Open Source“*. Ide o najnovšiu metodiku, ktorá umožňuje jednoduché vytváranie znovu použiteľných komponentov verifikačného prostredia s využitím generovania náhodných stimulov a funkčného pokrytia. Metodika UVM stavia na úspešnom nasadení metodík VMM a OVM. Jej hlavným cieľom je zlepšiť proces znovu použitia simulačných prostredí, zabezpečiť vyššiu prenositeľnosť verifikačného kódu a vytvoriť priestor pre nasadenie univerzálneho verifikačného a vysoko kvalitného verifikačného IP (angl. *Intellectual Property*).

3.5 Overovanie odolnosti systémov proti poruchám

Pre zabezpečenie odolnosti systému proti poruchám je na začiatku nutné definovať, čo je to spoľahlivosť systému. Spoľahlivosť nie je možné vyjadriť ako jednu vlasnosť, ktorá by zahrňovala niekoľko rôznych hľadísk, preto sa definuje pomocou nasledujúcich dielčich atribútov [7]:

- **dostupnosť** (angl. *availability*) - pohotovosť k vykonaniu správnej služby,
- **spoľahlivosť** (angl. *reliability*) - nepretržitosť poskytovania správnej služby,
- **zabezpečenosť** (angl. *safety*) - absencia katastrofických následkov pre užívateľa a prostredie,
- **dôvernosť** (angl. *security)* - absencia nepovolených odhalení informácie,
- **integrita** (angl. *integrity*) - absencia nevhodných zmien stavu systému,
- **udržiavateľnosť** (angl. *maintainability*) - schopnosť vykonávať opravy a modifikácie systému.

S pojmom spoľahlivosť úzko súvisia nasledujúce pojmy:

- **porucha** (angl. *fault*) - jav, ktorý vedie k narušeniu schopnosti objektu vykonávať požadovanú funkciu. Za vznikom poruchy stojí spravidla nejaká vonkajšia príčina.
- **chyba** (angl. *error*) - jav, ktorý je väčšinou dôsledkom poruchy, ale nie každá porucha musí spôsobiť chybu. Chyba sa prejavuje produkovaním nesprávnych výstupov systému.

Systém sa označuje ako odolný proti poruchám vtedy, pokiaľ je schopný správne vykonávať svoju funkciu aj v prítomnosti porúch technického vybavenia alebo chyby v programe. Nakoľko termín „správne vykonávať svoju funkciu“ je možné chápať rôzne, obvykle sa funkčnosť považuje za správnu, pokiaľ spĺňa nasledujúce tri podmienky [6]:

- spracovanie dát nebolo zastavené ani zmenené v dôsledku poruchy,
- výsledok je správny,
- výsledok bol získaný v predpísanej dobe.

Pokiaľ systém nespĺňa všetky tri uvedené podmienky, ale len niektoré z nich (napr. výsledok je správny, ale bol dodaný oneskorene), býva označovaný ako „čiastočne odolný proti poruchám“. Vždy však musí byť splnená prvá podmienka - požiadavka na zachovanie funkčnosti systému.

Pre zaistenie odolnosti systému proti poruchám rozlišujeme nasledujúce dva základné typy techník [4]:

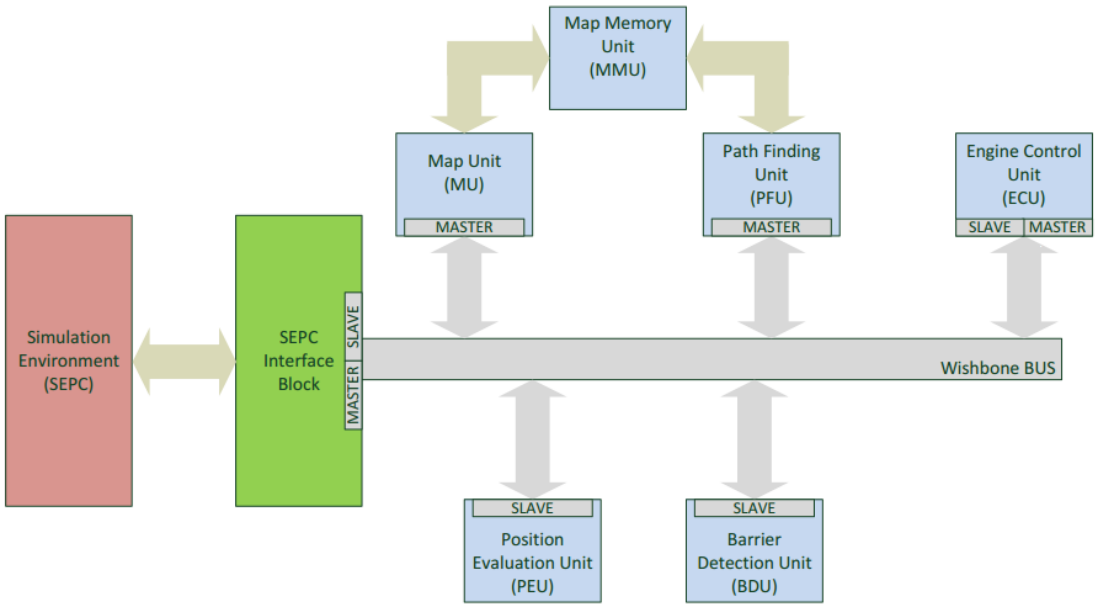
- **pasívna redundancia** - využíva maskovanie porúch, ktoré sa vyskytli v systéme. Obvykle nie sú využívané žiadne techniky pre detekciu a opravu porúch. Najčastejším príkladom je technika označovaná ako „TMR“ (angl. Triple *Modular Redundancy*), ktorá spočíva v trojnásobnom implementovaní rovnakých častí systému a za výsledok sa volí hodnota s najväčším výskytom. Obdobne sa využíva technika „NMR“ (angl. *N - Modular Redundancy*), ktorá využiva N-násobné implementovanie rovnakých častí systému a následne sa vyberie hodnota s najväčším výskytom.

- **aktívna redundancia** - využíva možnosť detekcie a opravy porúch. Pri tomto prístupe je vykonávané testovanie systému za behu a o prípadnom výskyte poruchy je systém okamžite informovaný. Následne je možné poruchu lokalizovať a izolovať, aby neovplyvňovala ostatné časti systému. Zistenú poruchu je možné následne opraviť a ovplyvnená časť systému je schopná opäť plniť svoju pôvodnú funkciu.

Kapitola 4

Riadiaca jednotka robota

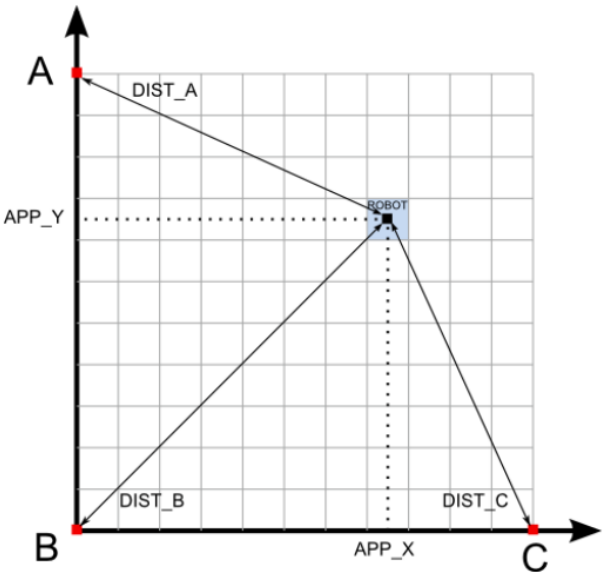
Účelom tejto práce je vytvorenie verifikačného prostredia pre riadiacu jednotku robota. Riadiaca jednotka robota určeného pre samočinný pohyb v bludisku tvorí exemplárny systém, ktorý je určený pre testovanie a overovanie metodík pre zaistenie odolnosti proti poruchám. Bloková schéma riadiacej jednotky sa nachádza na obrázku 4.1. Táto kapitola popisuje základnú činnosť riadiacej jednotky.



Obr. 4.1: Bloková schéma riadiacej jednotky robota.[8]

4.1 Výpočet aktuálnej pozície

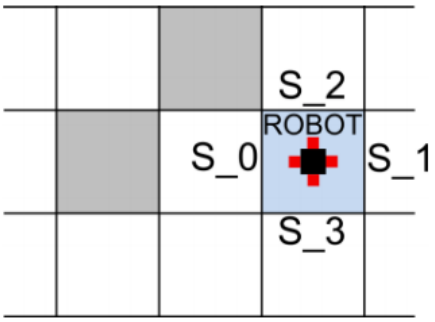
Blok pre vyhodnotenie pozície (*Position Evaluation Unit - PEU*) slúži pre výpočet pozície robota na mape. Výpočet je realizovaný na základe vzdialenosti od troch kontrolných bodov, ktoré sú umiestnené na fixných pozíciach na mape bludiska. Vypočítaná hodnota je reprezentovaná v podobe súradníc X a Y. Obrázok 4.2 ilustruje princíp výpočtu pozície. Vzdialenosti od troch kontrolných bodov (DIST_A, DIST_B, DIST_C) tvoria vstupy pre výpočet, ktorý je realizovaný pomocou Pythagorovej vety. Výsledok tvoria hodnoty APP_X a APP_Y.



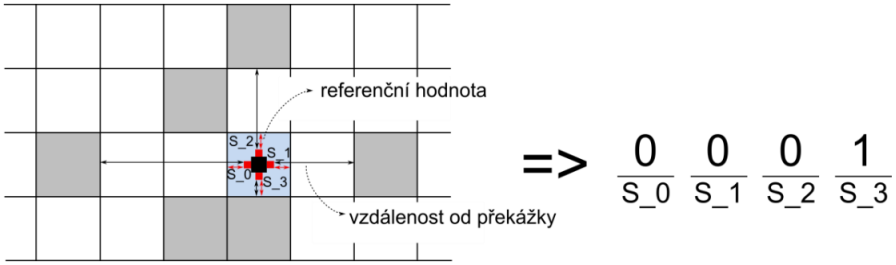
Obr. 4.2: Pozícia robota na mape.[8]

4.2 Vyhodnotenie prekážok

Blok pre vyhodnotenie prekážok (*Barrier Detection Unit - BDU*) slúži k vyhodnoteniu prekážok v mape. Pre vyhodnotenie prekážok sa využívajú 4 senzory (S_0, S_1, S_2, S_3). Každý senzor je umiestnený práve na jednej strane robota. Umiestnenie senzorov demonštruje obrázok 4.3. Výpočet vektora prekážok spočíva v porovnaní hodnoty získanej zo senzora s referenčnou hodnotou. V prípade, ak hodnota získaná zo senzora je menšia, než referenčná hodnota, na danom políčku sa nachádza prekážka. V prípade, ak získaná hodnota je väčšia, než referenčná hodnota, dané políčko je prázdne a robot sa môže na neho presunúť. Princíp výpočtu je znázornený na obrázku 4.4 .



Obr. 4.3: Senzory robota.[8]



Obr. 4.4: Detekcia prekážok.[8]

4.3 Hľadanie cesty v bludisku

Blok pre hľadanie cesty v bludisku (*Path Finding Unit - PFU*) realizuje algoritmus pre hľadanie cesty v bludisku. Výpočet prebieha na základe informácií o mape, ktorá sa vytvára priebežne. Výstupom tohto bloku je zoznam pozícií, ktoré musí robot prejsť v nasledujúcom kroku. Pre vyhľadanie cesty sa využíva algoritmus „ľavou rukou po stene“. Toto pravidlo zaručí, že pokiaľ sa počas prechádzania bludiska držíme ľavou rukou steny, tak v danom bludisku nezablúdime. Buď nájdeme cestu do cieľa alebo sa vrátime späť na počiatočnú pozíciu.

4.4 Ovládanie pohybu

Blok pre ovládanie pohybu (*Engine Control Unit* - ECU) nastavuje rýchlosť robota v požadovanom smere pohybu. Vzdialenosť, ktorú daný robot prejde určuje čas, v rámci ktorého sa robot hýbe. Vstupom pre tento blok je zoznam políčok, ktoré musí robot prejsť. Na základe zoznamu sa priebežne nastavuje rýchlosť pohybu v danom smere. Po nastavení rýchlosti nastáva pohyb robota. Jeho zastavenie je realizované pomocou nastavenia rýchlosti na hodnotu nula.

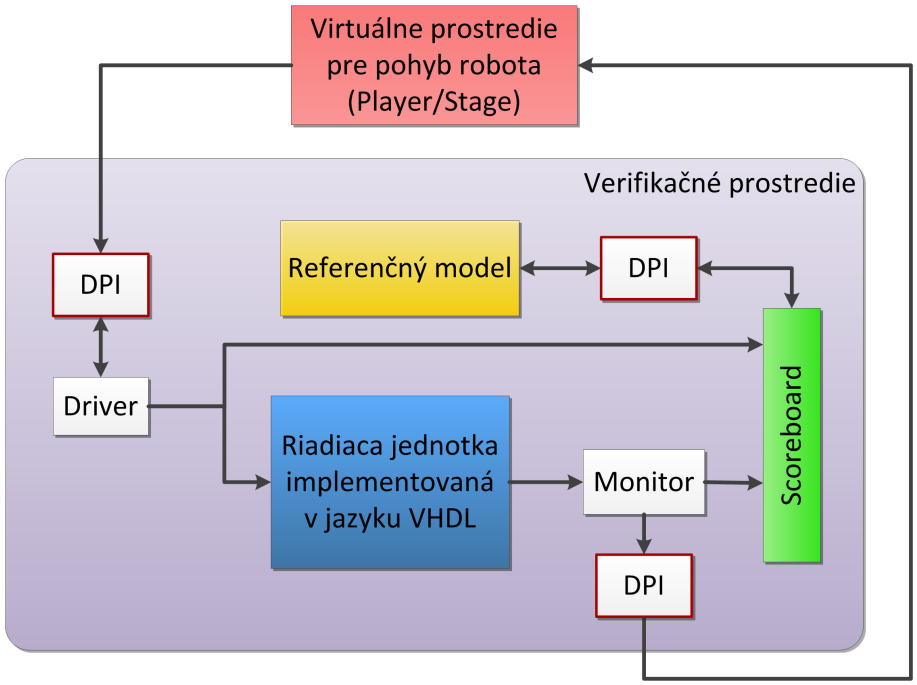
Kapitola 5

Návrh

V rámci kapitoly sú uvedené podrobné informácie týkajúce sa návrhu verifikačného prostredia. Súčasťou kapitoly sú uvedené požiadavky pre úpravu riadiacej jednotky, aby bolo možné verifikáciu vykonať.

5.1 Návrh verifikačného prostredia

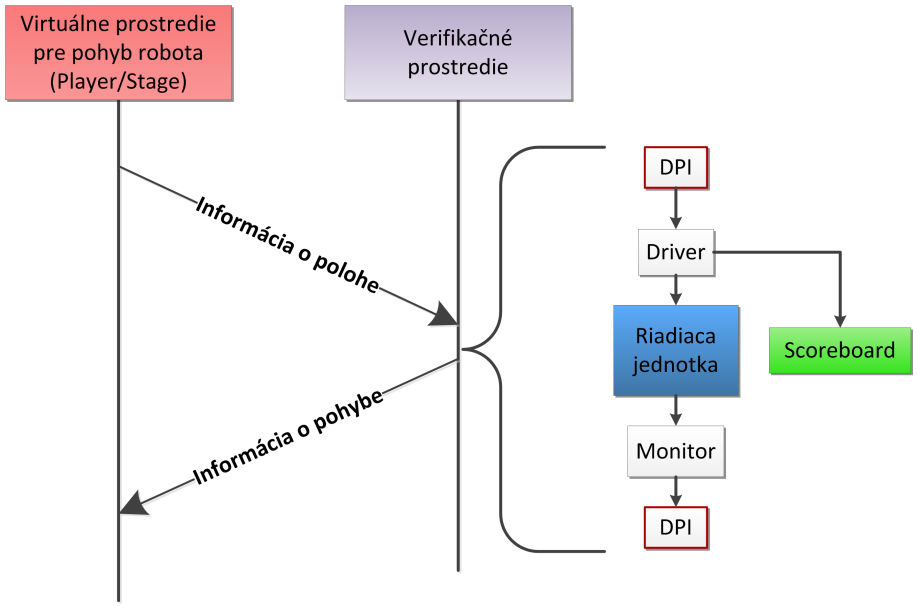
Ako bolo spomenuté v úvode, cieľom tejto práce je vytvoriť verifikačné prostredie pre overovanie korektnosti riadiacej jednotky robota. Koncept verifikačného prostredia je ilustrovaný na nasledujúcom obrázku(5.1):



Obr. 5.1: Návrh verifikačného prostredia.

Verifikačné prostredie je navrhnuté pomocou metodiky UVM v jazyku SystemVerilog. Verifikačné prostredie komunikuje s virtuálnym prostredím pre pohyb riadiacej jednotky (Player/Stage). Nejedná sa teda o reálneho fyzického robota, ale virtuálne prostredie pre

experimentálne účely ho plne nahradzuje. Virtuálne prostredie umožňuje zobrazenie pohybu robota v reálnom čase v rámci zvoleného bludiska. Samotná komunikácia medzi verifikačným prostredím a virtuálnym prostredím prebieha prostredníctvom sieťových socketov [10] s využitím transportného protokolu UDP. Princíp komunikácie je zobrazený na obrázku 5.2. Virtuálne prostredie zasiela verifikačnému prostrediu údaje o aktuálnej polohe (vzdialenosti od fixných bodov A, B, C) robota a údaje o aktuálnej vzdialenosti od prekážok (hodnoty senzorov S0, S1, S2, S3). Verifikačné prostredie zasiela virtuálnemu prostrediu údaje o práve vykonávanom pohybe (hodnoty pohybu v smere X a Y).



Obr. 5.2: Schéma komunikácie medzi verifikačným prostredím a virtuálnym prostredím.

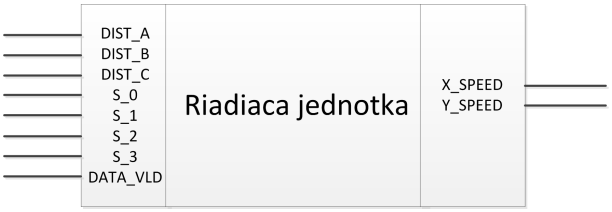
Na základe týchto údajov, virtuálne prostredie vykresľuje pohyb robota bludiskom v reálnom čase. Komunikácia prebieha v nekonečnej slučke, pokiaľ robot nedosiahne cieľovú pozíciu v bludisku. Z pohľadu verifikačného prostredia sú pre komunikáciu využívané DPI rozhrania. Na prijímacej strane verifikačné prostredie volá funkcie, ktoré vytvoria sieťový socket využívajúci transportný protokol UDP a príjme správu od virtuálneho prostredia. Na odosielacej strane verifikačné prostredie volá funkcie, ktoré opäť vytvoria sieťový socket, ďalej vytvoria odosielanú správu a s využitím transportného protokolu UDP ju následne odošlú do virtuálneho prostredia. Formát zasielaných údajov je zobrazený na obrázku 5.3. Hlavička tvorí prvú položku prenášanej správy. Na základe hodnoty hlavičky je možné rozlíšiť rôzne typy prenášaných správ. Hodnota hlavičky je v prípade oboch typov správ (údaje o polohe a údaje o pohybe) nastavená na hodnotu „0“. V prípade potrebného rozšírenia komunikačného rozhrania je možné nadefinovať nové typy prenášaných správ a identifikovať ich podľa hodnoty hlavičky.

Jednotlivé správy sú prijímané komponentom verifikačného prostredia, ktorý sa nazýva „**Driver**“. Tento komponent transformuje prijatú správu na signálovú reprezentáciu a odosiela ju na vstup verifikovanej jednotky a na vstup komponenty „Scoreboard“. Vstupné a výstupné rozhranie riadiacej jednotky je schematicky znázornené na obrázku 5.4. Popis týchto rozhraní je uvedený na obrázkoch 5.5 a 5.6.

Prijímaná správa								
Položka	Hlavička	DIST_A	DIST_B	DIST_C	S_0	S_1	S_2	S_3
Dátová šírka	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

Odosielaná správa			
Položka	Hlavička	SPEED_X	SPEED_Y
Dátová šírka	1 Byte	1 Byte	1 Byte

Obr. 5.3: Formát prijímaných a odosielaných údajov z pohľadu verifikačného prostredia.



Obr. 5.4: Riadiaca jednotka s rozhraniami.

Údaje z výstupného rozhrania verifikovanej jednotky sú privádzané na vstup komponenty „**Monitor**“. V tejto komponente sa z prijatých signálov vytvára výstupná transakcia, ktorá sa odosiela na vstup komponenty „Scoreboard“ a zároveň prostredníctvom DPI rozhrania sa vytvára sieťový socket využívajúci transportný protokol UDP a výstupná správa, ktorá sa odosiela do virtuálneho prostredia (Player/Stage).

Názov rozhrania	Dátová šírka rozhrania	Popis rozhrania
DIST_A	DATA_WIDTH	Vzdialenosť od bodu A.
DIST_B	DATA_WIDTH	Vzdialenosť od bodu B.
DIST_C	DATA_WIDTH	Vzdialenosť od bodu C.
S_0	DATA_WIDTH	Hodnota senzoru 0.
S_1	DATA_WIDTH	Hodnota senzoru 1.
S_2	DATA_WIDTH	Hodnota senzoru 2.
S_3	DATA_WIDTH	Hodnota senzoru 3.
DATA_VLD	logic	Príznak platnosti dát.

Obr. 5.5: Vstupné rozhranie riadiacej jednotky.

Názov rozhrania	Dátová šírka rozhrania	Popis rozhrania
X_SPEED	DATA_WIDTH	Rýchlosť v smere X
Y_SPEED	DATA_WIDTH	Rýchlosť v smere Y.

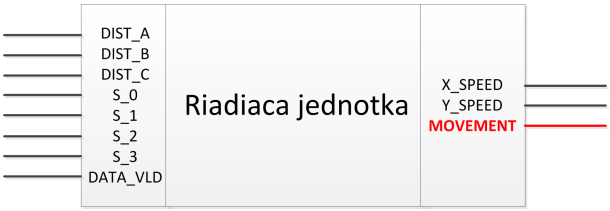
Obr. 5.6: Výstupné rozhranie riadiacej jednotky.

Komponenta „**Scoreboard**“ vykonáva funkciu porovnávania hodnôt medzi očakávaným pohybom robota a medzi skutočným pohybom robota. Po prijatí vstupnej transakcie od jednotky „Driver“ odošle prijaté hodnoty prostredníctvom DPI rozhrania do referenčného modelu, ktorý implementuje rovnaký algoritmus hľadania cesty v bludisku ako verifikovaný

obvod. Presný popis referenčného modelu sa nachádza v kapitole 5.3. Prostredníctvom DPI rozhrania príjme z referenčného modelu vypočítanú hodnotu pohybu a porovná ju so skutočnou hodnotou pohybu, ktorú prijal z kopomenty „Monitor“. Výsledok porovnania sa následne zaznamená pre potreby vyhodnotenia. Komponenta „Scoreboard“ taktiež na začiatku verifikácie nastavuje cieľovú pozíciu v bludisku, ktorú prostredníctvom DPI rozhrania oznámi referenčnému modelu. Referenčný model na základe tejto informácie kontroluje, či robot už dosiahol cieľovú pozíciu. V prípade, ak robot dosiahol na základe informácie z referenčného modelu cieľovú pozíciu v bludisku, verifikácia sa úspešne ukončí a informuje o výsledných porovnaniach formou štatistiky.

5.2 Úprava riadiacej jednotky

Pre overenie správnej funkcionality riadiacej jednotky robota je potrebné kontrolovať hodnoty na jej výstupnom rozhraní. Ide o hodnoty nastavenia rýchlosti v smere X (signál X_SPEED) a v smere Y (signál Y_SPEED). Ako bolo uvedené v kapitole 4, tieto hodnoty sa nastavujú v jednotke ECU. Nakoľko však doba výpočtu nasledujúcej pozície robota nie je konštantná a teda čas, v ktorom môžeme očakávať nastavenie hodnôt nie je možné jednoznačne určiť, je potrebné informovať verifikačné prostredie, kedy je možné vykonať porovnanie. Z tohto dôvodu bola vznesená požiadavka na implementáciu signálu, ktorý by na základe svojej hodnoty informoval, či sa robot pohybuje. Po analýze zdrojového kódu riadiacej jednotky robota bolo zistené, že takýto signál v rámci implementácie bloku ECU existuje a stačilo ho iba vyviesť na výstupné rozhranie riadiacej jednotky. Tento signál má označenie „MOVEMENT“ a je súčasťou výstupného rozhrania. Aktualizovaná schéma riadiacej jednotky s rozhraniami je znázornená na obrázku 5.7.



Obr. 5.7: Riadiaca jednotka s rozhraniami po úprave.

5.3 Návrh referenčného modelu

Referenčný model (tiež označovaný ako „golden model“ alebo „prediktor“) implementuje funkciu hľadania cesty v bludisku. Tento model slúži na overenie korektnosti riadiacej jednotky robota. Pre hľadanie cesty v bludisku využíva algoritmus „ľavou rukou po stene“. Spôsob jeho fungovania, teda spracovávania vstupných údajov, ich vyhodnocovanie a výpočet pozície pre pohyb, je implementované tak, aby bolo možné overiť všetky kroky, ktoré vykonáva riadiaca jednotka robota. Postup operácií, ktoré musí referenčný model vykonať je nasledovný:

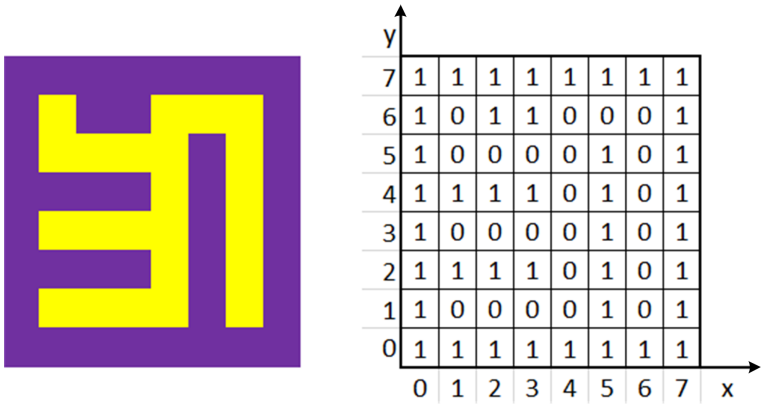
1. Načítanie cieľovej pozície.
2. Načítanie vstupných údajov.
3. Výpočet aktuálnej pozície.

4. Výpočet vektora prekážok.

5. Výpočet cieľa pohybu.

5.3.1 Načítanie údajov

Na začiatku si referenčný model načíta cieľové súradnice. Súradnice sú reprezentované dvojicou <X , Y>, pričom hodnota súradnice vyjadruje políčko bludiska. Mapa bludiska môže mať rôznu veľkosť, v základnej verzii pracuje referenčný model s veľkosťou 8 x 8 políčok. Samotná mapa je reprezentovaná pomocou dvoch bitov - „1“ a „0“. Políčko, ktoré reprezentuje stenu je označené bitom s hodnotou „1“ a políčko, ktoré reprezentuje cestu je označené bitom s hodnotou „0“. Príklad mapy s jej binárnou reprezentáciou je uvedený na obrázku 5.8 .



Obr. 5.8: Mapa a jej bitová reprezentácia

Načítanie údajov je realizované pomocou odoslania príslušných parametrov z verifikačného prostredia. Ide o sedem parametrov - DIST_A, DIST_B, DIST_C, S_0, S_1, S_2, S_3.

5.3.2 Výpočet pozície

Aktuálna pozícia sa vypočítava pomocou vyhodnotenia vzdialenosti od troch kontrolných bodov - A, B a C, ktoré sú rozmiestnené na fixných pozíciach. Na základe vzdialenosti od týchto bodov (DIST_A, DIST_B, DIST_C) vieme pomocou Pythagorovej vety vypočítať presnú pozíciu robota (APP_X, APP_Y). Na základe tejto vety boli odvodené nasledujúce 2 rovnice:

$$APP_X^2 + APP_Y^2 = DIST_B^2 \tag{5.1}$$

$$(X_{MAX} - APP_X)^2 + APP_Y^2 = DIST_C^2 \tag{5.2}$$

Po úprave vyššie uvedených rovníc je možné vyjadriť hodnoty APP_X a APP_Y, ktoré reprezentujú súradnice X a Y. Nakoľko sa robot pohybuje po jednotlivých políčkach, je potrebné získané hodnoty súradníc previesť na hodnotu súradnice, ktorá je vyjadrená v políčkach mapy. Toto vyjadrenie zabezpečíme vydelením získanej hodnoty súradnice hodnotou konštanty BOX_SIZE. Táto konštanta reprezentuje veľkosť políčka bludiska.

$$APP_X = \frac{X_{MAX}^2 - DIST_C^2 + DIST_B^2}{2.X_{MAX}} \tag{5.3}$$

$$X = \frac{APP_X}{BOX_{SIZE}} \tag{5.4}$$

$$APP_Y = \frac{Y_{MAX}^2 - DIST_A^2 + DIST_B^2}{2.Y_{MAX}} \tag{5.5}$$

$$Y = \frac{APP_Y}{BOX_{SIZE}} \tag{5.6}$$

5.3.3 Výpočet vektora prekážok

Pre vyhodnotenie prekážok sa využívajú 4 senzory (S_0, S_1, S_2, S_3), pričom každý senzor je umiestnený práve na jednej strane robota. Výpočet vektora prekážok spočíva v porovnaní vstupných hodnôt z jednotlivých senzorov s referenčnou hodnotou DIST_MAX. V prípade, ak je získaná hodnota senzora menšia, než referenčná hodnota, tak potom sa v danom smere nenachádza žiadna prekážka a políčko je voľné. V opačnom prípade sa v danom smere nachádza prekážka a pohyb týmto smerom nie je možný.

5.3.4 Výpočet cesty v bludisku

Výpočet cesty v bludisku je realizovaný pomocou algoritmu „ľavou rukou po stene“. Tento algoritmus vychádza z pravidla, ktoré tvrdí, že pokiaľ sa budeme počas hľadania východu z bludiska držať vždy ľavou rukou steny, tak nezablúdime. Buď nájdeme východ z bludiska alebo sa vrátime späť na počiatočnú pozíciu. Referenčný model v každom kroku vyhodnotí svoju aktuálnu pozíciu, porovná ju s cieľovou pozíciou a v prípade, ak sa nenachádza v cieli, určí nasledujúcu súradnicu pohybu tak, aby sa vždy držal ľavou rukou steny bludiska.

Návrh algoritmu je nasledujúci: robot sa pri každom pohybe snaží prejsť na políčko, ktoré sa nachádza na ľavej strane od jeho pozície. Tento princíp je realizovaný tak, že samotný pohyb je možné realizovať v štyroch smeroch, ktoré robot môže absolvovať v nasledujúcom poradí cyklicky: východ → sever → západ → juh → východ → ... Táto cyklická závislosť pohybu zabezpečí, že robot sa vždy bude „držať ľavou rukou pri stene“. Ak je teda políčko naľavo voľné, presunie sa naň. Pokiaľ sa však na danom políčku nachádza prekážka, otočí sa smerom doprava a opäť sa pokúša prejsť na políčko, ktoré sa nachádza na ľavej strane od jeho pozície. Robot sa otáča do pravej strany dovtedy, kým nie je políčko naľavo od jeho pozície voľné. Tento proces sa cyklicky opakuje, pokým robot neprejde na cieľovú súradnicu políčka. Princíp výpočtu cesty je demonštrovaný pseudokódom v Algoritme 1.

Algoritmus 1: Pseudokód pohybu robota

```
1: smer = VÝCHOD;
2: while nie som v cieli do
3:     smer = smer - 1 ; // otočenie doľava
4:     while ciel pohybu == STENA do
5:         smer = smer +1 ; // otočenie doprava
6:     end while
7:     vykonaj pohyb(smer) ; // robot sa presunie na novú pozíciu vo
       vypočítanom smere
8: end while
```

Samotná realizácia pohybu spočíva v nastavovaní rýchlosti pohybu v požadovanom smere v rámci svetových strán:

- V prípade pohybu smerom na sever sa hodnota X_SPEED nastaví na hodnotu „1“,
- V prípade pohybu smerom na juh sa hodnota X_SPEED nastaví na hodnotu „-1“,
- V prípade pohybu smerom na východ sa hodnota Y_SPEED nastaví na hodnotu „1“,
- V prípade pohybu smerom na západ sa hodnota Y_SPEED nastaví na hodnotu „-1“.

V prípade, ak sú premenné X_SPEED a Y_SPEED nastavené na hodnotu „0“, robot sa nehýbe. Situácia, kedy by boli obidve premenené nastavené súčasne na inú hodnotu než „0“ nemôže nastať, nakoľko robot sa môže hýbať iba vo vertikálnom alebo horizontálnom smere.

Kapitola 6

Implementácia a analýza výsledkov

V rámci kapitoly sú uvedené implementačné podrobnosti verifikačného prostredia, referenčného modelu a komunikačného rámca. Taktiež sú tu uvedené popisy implementovaných tried, návrhy testov, ich analýza a vyhodnotenie.

6.1 Implementácia simulácie virtuálneho prostredia pre pohyb robota

V čase implementácie verifikačného prostredia nebolo možné využiť virtuálne prostredie pre pohyb robota (Player/Stage) z dôvodu nefunkčnosti komunikačného rozhrania medzi týmto virtuálnym prostredím a jeho okolím. Nakoľko implementácia komunikačného rozhrania nie je predmetom tejto práce a pre verifikáciu riadiacej jednotky je funkčnosť virtuálneho prostredia nutná, bolo rozhodnuté, že virtuálne prostredie bude nahradené jeho zjednodušenou podobou, ktorá bola implementovaná v programovacom jazyku C. Ide o aplikáciu, ktorá beží na serveri a pre svoju činnosť využíva sieťové sockety. Táto aplikácia poskytuje možnosť zasielať hodnoty na vstup verifikačného prostredia. Po spustení aplikácie sa vytvorí sieťový socket a následný prenos je realizovaný pomocou transportného protokolu UDP. Na požadovanú cieľovú adresu a požadovaný cieľový port sa v pevných časových intervaloch odosielajú vstupné informácie pre verifikačné prostredie vo formáte, aký bol prezentovaný v kapitole 5.1. Jednotlivé hodnoty, ktoré aplikácia odosiela sú pre jednoduchosť a efektívnosť implementácie súčasťou zdrojového kódu. Pri ich zmene alebo doplnení týchto údajov je potrebné vykonať opätovný preklad aplikácie. Aplikácia sa spúšťa z terminálu (konzoly) daného počítača a ako vstupné parametre požaduje cieľovú adresu a cieľový port počítača na ktorý má odoslať údaje. Parametre aplikácie je možné upraviť v rámci hlavičkového súboru. Verifikačné prostredie musí byť spustené pred spustením tejto aplikácie.

6.2 Implementácia referenčného modelu

Referenčný model bol implementovaný v programovacom jazyku C. Nakoľko program je využívaný vo forme zdieľanej knižnice v operačnom systéme Linux (*shared object* - *.so*), tak vstupným rozhraním nie je štandardná funkcia jazyka C: „`main()`“. Program je používateľovi dostupný ako skupina individuálnych funkcií, ktoré je možné nezávisle na sebe volať z prostredia operačného systému. Z tohto dôvodu bolo potrebné vyriešiť udržiavanie kontextu v rámci celého programu počas doby jeho využívania. Štandardným spôsobom, ako tento problém vyriešiť je využitie globálnych statických premenných, ktoré si udržujú svoju

hodnotu aj po skončení vykonávanej funkcie. Pri opätovnom volaní požadovanej funkcie je možné nadviazať na predchádzajúcu postupnosť vďaka informáciám z týchto premenných. V rámci implementácie sú využívané dve globálne statické premenné: „`X_goal`“ a „`Y_goal`“. Tieto premenné si počas celej doby uchovávajú súradnice cieľa pohybu robota v mape. Všetky definície funkcií, vymenované typy (*enum*) a konštanty sú uvedené v hlavičkom súbore „`golden_model.h`“. Postupnosť volaní implementovaných funkcií je nasledujúca:

1. Volanie funkcie `set_goal(...)` - nastaví cieľové súradnice. Táto funkcia musí byť zavolaná na začiatku práce s programom.

2. Volanie funkcie `get_speed(...)` - táto funkcia obsahuje hlavnú časť výpočtu celého programu. Ako svoj vstup požaduje osem parametrov, ktoré sú poskytované virtuálnym prostredím pre pohyb robota (vzdialenosti od bodov A, B, C, hodnoty senzorov S0 až S3 a požadovaná rýchlosť v smere X alebo Y). V rámci tejto funkcie je implementovaný samotný algoritmus „ľavou rukou po stene“ a kontrola, či robot dosiahol cieľovú súradnicu. Návratová hodnota funkcie predstavuje vypočítanú hodnotu rýchlosti v smere pohybu pre požadovanú súradnicu. V prípade, ak robot dosiahol cieľovú pozíciu, funkcia vráti hodnotu „`GOAL_REACHED`“. V rámci svojej činnosti volá nasledujúce funkcie:

   (a) `my_position_x(...)`, `my_position_y(...)` - na základe vzdialeností od bodov A, B, C vypočíta svoju pozíciu v bludisku,

   (b) `check_direction(...)` - na základe hodnoty jednotlivých senzorov kontroluje, či sa v požadovanom smere pohybu nachádza stena,

   (c) `normalize_direction(...)` - zabezpečuje cyklické otáčanie v smere svetových strán (východ, juh, západ, sever, východ, ...),

   (d) `move (...)` - vykonáva samotný pohyb. Návratovou hodnotou je nastavená rýchlosť v požadovanom smere pohybu.

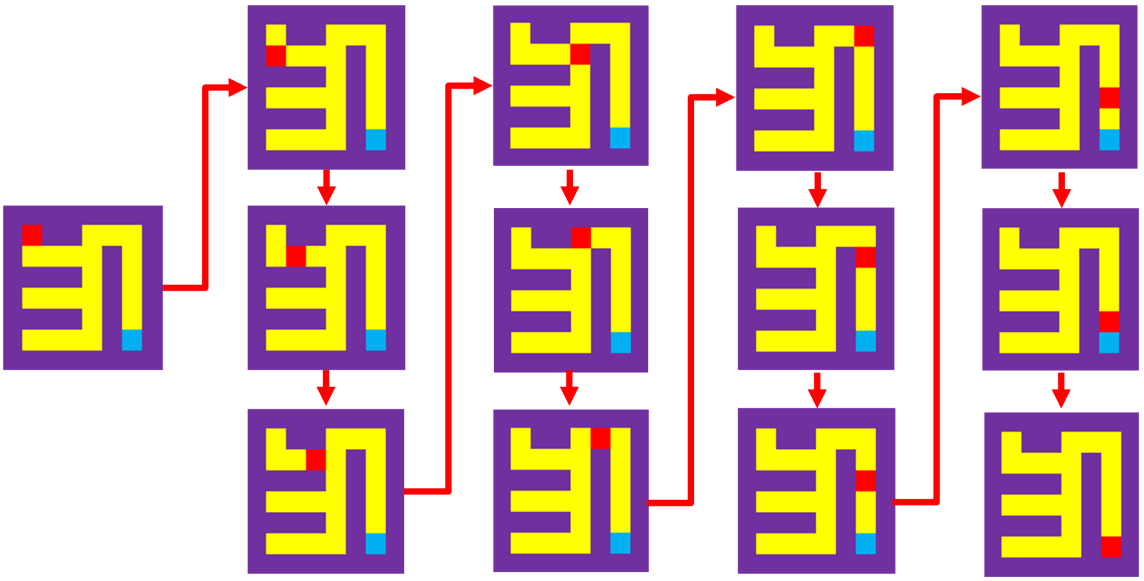
6.3 Overenie funkčnosti referenčného modelu

Funkčnosť implementácie referenčného modelu bola overená prostredníctvom jednoduchého testovacieho prostredia implementovaného v jazyku C. V každom kroku výpočtu, od počiatočnej pozície až do cieľovej pozície, sa sledovala aktuálna pozícia robota a smer, ktorým sa robot pohol. Tieto hodnoty sa následne porovnávali s očakávanými hodnotami. Počiatočné a cieľové pozície boli zvolené tak, aby počet potrebných krokov, ktoré musí robot vykonať v rámci priechodu každého bludiska bol minimálne 10. V rámci overovania bolo využitých 6 bludísk (5 bludísk o rozmere 8x8 políčok a 1 bludisko o rozmere 16x16 políčok). V rámci každého bludiska boli vykonané 2 experimenty s rozdielnymi počiatočnými a cieľovými pozíciami. Konkrétne údaje o vykonaných experimentoch sú uvedené v tabuľke 6.1.

Rozmer bludiska	Priemerný počet krokov	Počet vykonaných experimentov
8x8	22	10
16x16	39	2
Počet experimentov spolu		**12**

Tabuľka 6.1: Experimentálne overenie referenčného modelu.

Funkčnosť implementovaného referenčného modelu je demonštrovaná pomocou diagramu (obrázok 6.1), ktorý znázorňuje pohyb robota po bludisku o veľkosti 8x8 políčok z počiatočnej pozície do cieľovej pozície.



Obr. 6.1: Pohyb robota po bludisku.

6.4 Implementácia verifikačného prostredia

Verifikačné prostredie bolo implementované v jazyku SystemVerilog s využitím metodiky UVM. Toto verifikačné prostredie využíva pre svoju činnosť simulačné prostredie ModelSim verzie 10.0 od spoločnosti Mentor Graphics a knižnicu UVM verzie 1.1d. Jednotlivé zložky verifikačného prostredia boli implementované pomocou komponent, ktoré obsahuje knižnica UVM. Nasledujúca sekcia obsahuje podrobný popis implementovaných tried.

6.4.1 Popis implementovaných tried

1. **rcInputTransaction** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_sequence_item`“. Táto trieda definuje presný formát vstupných transakcií. V rámci triedy používa metódu „`do_copy()`“, pomocou ktorej sa vytvárajú nové transakcie a metódu „`print()`“, ktorá slúži pre výpis obsahu transakcie.

2. **rcOutputTransaction** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_sequence_item`“. Táto trieda definuje presný formát výstupných transakcií. V rámci triedy používa metódu „`do_copy()`“, ktorá slúži na vytváranie nových transakcií metódu „`print()`“, ktorá slúži pre výpis obsahu transakcie a metódu „`do_compare()`“, ktorá sa využíva na porovnanie hodnôt výstupných transakcií (nastavenie rýchlosti X_SPEED a Y_SPEED) .

3. **TransactionSequence** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_sequence`“. V rámci tejto triedy sa nastavujú konkrétne hodnoty vstupnej transakcie. Pre nastavenie hodnôt sa využívajú DPI volania funkcií implementovaných

v jazyku C. V rámci sekcie „`TransactionSequence::body`“ sa volá funkcia „`receive_data()`“, ktorá vytvorí sieťový socket a s využitím transportného protokolu UDP príjme vstupné dáta z virtuálneho prostredia pre pohyb robota (Player/Stage). Nakoľko pre príjem dát sa využívaná funkcia „`recvfrom()`“, ktorá je blokujúceho typu, verifikácia bude čakať, kým vstupné dáta neprijme. Po ich prijatí pomocou funkcie „`get_data()`“ naplní jednotlivé položky vstupnej transakcie.

4. **TransactionSequencer** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_sequencer`“. Táto trieda koordinuje zasielanie vstupných transakcií medzi komponentami „Sequence“ a „Driver“. V prípade využívania paralelného zasielania transakcií, bude táto trieda rozhodovať o poradí v akom budú zasielané.

5. **rcDriver** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_driver`“. Táto trieda prijíma vstupné transakcie a transformuje ich do signálovej reprezentácie. Hodnoty signálov následne odosiela na vstupné rozhranie riadiacej jednotky robota. Odosielanie dát je synchronizované na nástupnú hranu hodinového signálu „`CLK`“. rcDriver zároveň využíva tzv. „`analysis port`“, pomocou ktorého vstupnú transakciu odosiela do komponentu „Scoreboard“ pre ďalšie spracovanie.

6. **rcMonitor** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_monitor`“. Táto trieda prijíma signály z výstupného rozhrania riadiacej jednotky robota a transformuje ich do formy výstupnej transkacie. Pomocou rozhrania „`analysis port`“ odosiela transakciu do komponentu „Scoreboard“. Odosielanie transakcie je synchronizované na nástupnú hranu hodinového signálu „`CLK`“.

7. **rcScoreboard** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_scoreboard`“. Trieda prijíma vstupné transakcie z komponenty „Driver“ a výstupné transakcie z komponenty „Monitor“. Po prijatí vstupnej transakcie zasiela prijaté údaje pomocou DPI rozhrania do referenčného modelu. Následne z referenčného modelu príjme vypočítané výstupné hodnoty a porovná ich s hodnotami prijatými z komponentu „Monitor“. Výsledky jednotlivých porovnaní sú zaznamenávané. Zároveň kontroluje, či robot riadený riadiacou jednotkou dosiahol cieľové súradnice. V prípade ich dosiahnutia ukončí verifikáciu a vypíše štatistiku vykonaných porovnaní. Ak robot nedosiahol cieľové súradnice ani po predpokladanom (nastavenom) počte krokov, ukončí verifikáciu ako neúspešnú a vypíše štatistiku vykonaných porovnaní.

8. **rcAgent** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_component`“. V rámci triedy sú definované komponenty „`TransactionSequencer`“, „`rcDriver`“ a „`rcMonitor`“. Zároveň sa vytvára prepojenie medzi týmito komponentami.

9. **rcEnv** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_env`“. V rámci triedy sú definované komponenty „`rcAgent`“ a „`rcScoreboard`“ a následne je medzi nimi vytvorené prepojenie.

10. **rcTest** - pre svoju činnosť rozširuje (dedí) štandardnú triedu „`uvm_test`“. Trieda definuje a nastavuje verifikačné prostredie „`rcEnv`“ a v rámci sekcie „`rcTest::main_phase`“ spúšťa zasielanie vstupných transakcií.

Súčasťou verifikačného prostredia sú nasledujúce súbory, ktoré konfigurujú verifikačné prostredie pomocou parametrov a prepájajú ho s riadiacou jednotkou robota:

1. **top_level.sv** - komponent najvyššej úrovne. Definuje hodinový signál „`CLK`“ a resetovací signál „`RST`“. Prepája vstupné a výstupné rozhranie riadiacej jednotky robota s verifikačným prostredím prostredníctvom tzv. „virtuálnych rozhraní“. Spúšťa samotnú verifikáciu.

2. **interface.sv** - súbor, v rámci ktorého je definované vstupné a výstupné rozhranie verifikačného prostredia. V rámci súboru je nastavená synchronizácia vstupných a výstupných signálov na nábehovú hranu hodinového signálu „`CLK`“.

3. **testing_parameters.sv** - súbor, ktorý obsahuje konfiguračné parametre pre verifikačné prostredie.

4. **simulation.fdo** - skript, ktorý slúži pre preklad a spustenie verifikácie.

Všetky tieto súbory sú priložené na CD nosiči v rámci prílohy A.

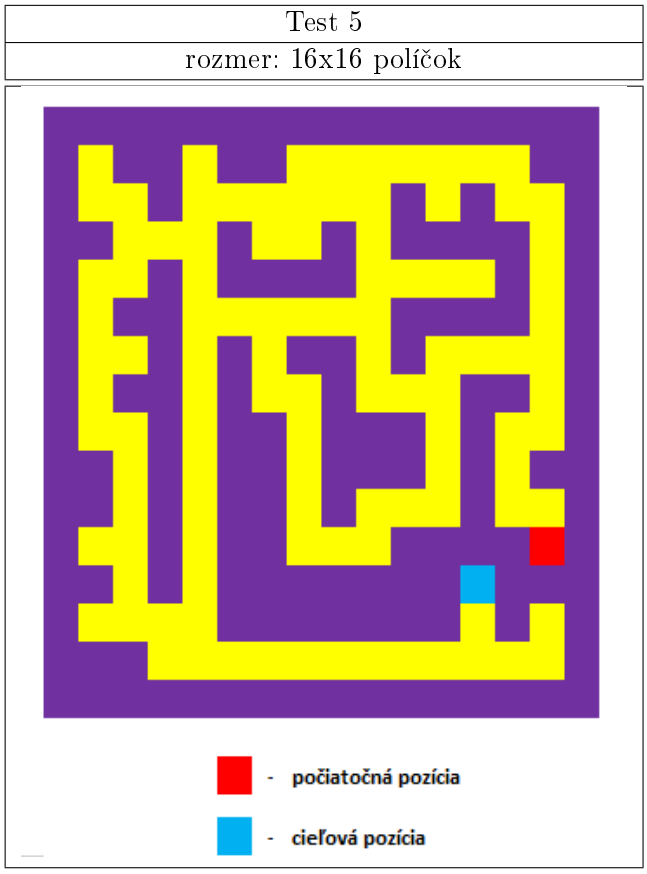
6.5 Návrh testov

Pre overenie funkčnosti a korektnosti riadiacej jednotky robota bolo navrhnutých päť testov. Každý test pozostával z jedného typu bludiska. Cieľom testu bolo overenie, či robot v presne stanovenom počte krokov dorazí z počiatočnej pozície do cieľovej pozície. Jednotlivé testovacie bludiská sú znázornené na obrázkoch 6.2 a 6.3.

Test 1	Test 2	Test 3	Test 4
rozmer : 8x8 políčok	rozmer: 8x8 políčok	rozmer: 8x8 políčok	rozmer: 8x8 políčok
			

Obr. 6.2: Testovacie bludiská - rozmer 8x8.

Pred verifikáciou je potrebné nastaviť parametre verifikačného prostredia. Ich hodnoty sú uvedené v tabuľke 6.2.



Obr. 6.3: Testovacie bludisko - rozmer 16x16.

Názov parametra	Popis parametra	Hodnota parametra
DATA_WIDTH	Dátová šírka rozhraní.	16 bitov
CLK_PERIOD	Perióda hodinového signálu.	30 ns
RESET_TIME	Doba, počas ktorej je aktívny signál reset.	100 ns
WAIT_TIME	Čas čakania medzi vstupnými transakciami.	2 300 ns
GOAL_X	Cieľová súradnica v rovine X.	podľa konkrétneho testu
GOAL_Y	Cieľová súradnica v rovine Y.	podľa konkrétneho testu
TRANS_COUNT	Počet vstupných transakcií.	podľa konkrétneho testu
TRANS_COMPARE_COUNT	Počet porovnaní v komponente Scoreboard.	10

Tabuľka 6.2: Parametre verifikačného prostredia.

Pre jednotlivé testy je potrebné nastaviť špecifické parametre, ktoré sú závislé od konkrétneho bludiska. Hodnoty parametrov pre jednotlivé testy sú uvedené v tabuľke 6.3.

Názov	Test 1	Test 2	Test 3	Test 4	Test 5
GOAL_X	6	6	3	6	12
GOAL_Y	1	1	1	6	3
TRANS_COUNT	13	33	27	29	48

Tabuľka 6.3: Hodnoty parametrov pre jednotlivé testy.

6.6 Priebeh verifikácie

Pri realizácií verifikácie je veľmi dôležité mať možnosť sledovať presnú postupnosť prechodu transakcií jednotlivými komponentami verifikačného prostredia. Z dôvodu rýchleho odhalenia a jednoduchšej analýzy chýb sú v rámci tejto sekcie uvedené kontrolné výpisy obsahu transakcií. Všetky výpisy sa zobrazujú v rámci okna „Transcript“ programu ModelSim.

Na začiatku simulácie sa vypíšu nastavené parametre pre aktuálne spúšťaný test. Tento výpis je zobrazený na obrázku 6.4.

```
# UVM_INFO @ 0: reporter [RNTST] Running test rcTest...
# ----------------------------------------------------------
#              TESTING        PARAMETERS
#
# CLOCK PERIOD:                    30 ns
# RESET TIME:                     100 ns
# INITIAL WAIT TIME:           125000 ns
# WAIT TIME:                   151000 ns
#
# GOAL COORDINATES:             ( 6, 1 )
#
# TRANSACTION COUNT:                13
# TRANSACTION COMPARE COUNT:      5000
# ----------------------------------------------------------
```

Obr. 6.4: Výpis parametrov testu.

Výpis obsahujúci vstupné transakcie je možné vidieť na obrázku 6.5. Tento výpis zobrazuje vstupnú transakciu, ktorej hodnoty boli získané z virtuálneho prostredia pre pohyb robota prostredníctvom DPI rozhrania. Výpis je realizovaný v rámci komponenty „Driver“, ktorá danú transakciu obdržala od komponenty „Sequencer“. Tento výpis predstavuje prvé miesto kontroly správnej funkčnosti verifikačného prostredia. Pokiaľ by komunikácia medzi verifikačným prostredím a virtuálnym prostredím pre pohyb robota nefungovala správne, v rámci tohto výpisu by sme spozorovali nesprávne nastavenie hodnôt vstupnej transakcie.

```
# ----------------------------------------------------------
# DRIVER: INPUT TRANSACTION:
# Time =              276115 ns
# dist_a: 28,dist_b: 48,dist_c: 62,   s_0: 12   s_1: 20   s_2:  4   s_3:  4
# ----------------------------------------------------------
```

Obr. 6.5: Výpis vstupnej transakcie - Driver.

V rámci komponenty „Monitor“ je zobrazený výpis obsahujúci výstupnú transakciu, ktorú komponenta obdržala z výstupného rozhrania riadiacej jednotky. Tento výpis predstavuje druhé miesto kontroly správnej funkčnosti verifikačného prostredia. V prípade, ak by riadiaca jednotka robota vypočítala hodnoty iné ako „-1“, „0“ alebo „1“, tak v rámci tohto výpisu by bolo možné vidieť chybu. Výpis je zobrazený na obrázku 6.6.

```
# ----------------------------------------------------------
# MONITOR: OUTPUT TRANSACTION:
# Time =              742995 ns
# X_SPEED:  0, Y_SPEED:  1
# ----------------------------------------------------------
```

Obr. 6.6: Výpis výstupnej transakcie - Monitor.

Komponenta „Monitor“ v rámci svojej funkcionality navyše kontroluje, či robot nevykonáva pohyb v čase, kedy je signál „MOVEMENT“ nastavený na hodnotu „0“. Ako bolo spomenuté v kapitole 5.2, signál „MOVEMENT“ informuje o práve vykonávanom pohybe. Pokiaľ je

signál nastavený na hodnotu „0“, tak sa robot nesmie hýbať. Táto kontrola je veľmi dôležitá, najmä z dôvodu využitia verifikačného prostredia v budúcnosti pre overovanie miery zabezpečenia riadiacej jednotky robota voči chybám, ktoré budú do nej injektované. V prípade, ak teda nastane pohyb v rámci doby, kedy je signál „MOVEMENT“ nastavený na hodnotu „0“, tak verifikačné prostredie vypíše informáciu o takto vzniknutej situácii (viď obrázok 6.7).

```
# UVM_ERROR @ 1879245: uvm_test_top.rc_env.rc_agent.rcMonitor [MONITOR] Robot movement not allowed while signal MOVEMENT is set to "0"
#
# UVM_ERROR @ 1879275: uvm_test_top.rc_env.rc_agent.rcMonitor [MONITOR] Robot movement not allowed while signal MOVEMENT is set to "0"
#
# UVM_ERROR @ 1879305: uvm_test_top.rc_env.rc_agent.rcMonitor [MONITOR] Robot movement not allowed while signal MOVEMENT is set to "0"
```

Obr. 6.7: Výpis chyby - Monitor.

Najväčší počet výpisov je realizovaný v komponente „Scoreboard“. Ako prvý je možné vidieť výpis prijatej vstupnej transakcie z jednotky „Driver“ (viď obrázok 6.8). Hodnoty transakcie budú využité ako vstup pre referenčný model. Tento výpis predstavuje tretie miesto kontroly.

```
# ------------------------------------------------------
# SCOREBOARD: INPUT TRANSACTION FROM DRIVER::
# Time =              609675 ns
# dist_a: 41,dist_b: 57,dist_c: 52,   s_0: 28   s_1:  4   s_2: 12   s_3: 36
# ------------------------------------------------------
```

Obr. 6.8: Výpis vstupnej transakcie - Scoreboard.

Výpis, ktorý je uvedený na obrázku 6.9 zobrazuje vypočítané hodnoty, ktoré verifikačné prostredie prijalo z referenčného modelu prostredníctvom DPI rozhrania. Výpis predstavuje štvrté miesto kontroly. V prípade, ak by referenčný model nesprávne vypočítal rýchlosť pohybu, v rámci tohto výpisu je možné odhaliť chybu.

```
# ------------------------------------------------------
# SCOREBOARD: VALUES RECEIVED FROM GOLDEN MODEL::
# Time =              609675 ns
# X_SPEED:  0, Y_SPEED:  1
# ------------------------------------------------------
```

Obr. 6.9: Výpis výstupnej transakcie - Referenčný model.

V rámci činnosti porovnávania skutočnej hodnoty výstupnej transakcie (teda pohybu robota) a očakávanie hodnoty výstupnej transakcie (vypočítaná hodnota z referenčného modelu) je možné zobraziť hodnoty týchto transakcií. Výpis je zobrazený na obrázku 6.10. Tento výpis reprezentuje piate miesto kontroly.

```
# ------------------------------------------------------
# :SCOREBOARD EXPECTED OUTPUT::
# Time =              761655 ns
# X_SPEED:  0, Y_SPEED:  1
# ------------------------------------------------------
# ------------------------------------------------------
# :SCOREBOARD   REAL  OUTPUT::
# Time =              761655 ns
# X_SPEED:  0, Y_SPEED:  1
# ------------------------------------------------------
```

Obr. 6.10: Výpis porovnania transakcií - Scoreboard.

Posledný výpis, ktorý sa zobrazuje v rámci verifikačného prostredia je oznámenie o vykonanej verifikácii. V prípade, ak robot dosiahol cieľové súradnice, verifikačné prostredie oznámi túto skutočnosť prostredníctvom výpisu, ktorý sa nachádza na obrázku 6.11. V prípade, ak robot nedosiahol cieľové súradnice, verifikácia je ukončená ako neúspešná a informuje o tom prostredníctvom výpisu, ktorý je uvedený na obrázku 6.12.

```
#
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!        GOAL REACHED        !!!!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#
#
# ------------------------------------------------------------------------
# -----------------          SCOREBOARD         --------------------------
# ------------------------------------------------------------------------
#
#  MATCHES:       60000
#
#  MISMATCHES:           0
#
#  VERIFICATION ENDED CORRECTLY, ROBOT REACHED GOAL COORDINATES
```

Obr. 6.11: Úspešné ukončenie verifikácie.

```
# ------------------------------------------------------------------------
# -----------------          SCOREBOARD         --------------------------
# ------------------------------------------------------------------------
#
#  MATCHES:       55000
#
#  MISMATCHES:        5000
#
#  SCOREBOARD MISMATCHES OCCURED IN VERIFICATION.
#
#
#  VERIFICATION FAILED, ROBOT DIDN'T REACH GOAL COORDINATES
#
```

Obr. 6.12: Neúspešné ukončenie verifikácie.

6.7 Odhalenie a analýza chybových stavov

Stav je možné označiť ako chybový vtedy, pokiaľ komponenta verifikačného prostredia produkuje hodnoty, ktoré nezodpovedajú špecifikácii. Ak hodnoty na výstupnom rozhraní verifikovanej jednotky nezodpovedajú očakávaným hodnotám, je možné tento stav taktiež označiť ako chybový. V nasledujúcej časti sú popísané problémy a chybové stavy, ktoré boli odhalené počas verifikácie.

6.7.1 Test 1

Počas vykonávania testu 1 boli odhalené nasledujúce chybové stavy:

Chyba č. 1: Na základe špecifikácie bolo navrhnuté oneskorenie zasielania vstupných transakcií na rozhranie riadiacej jednotky. Z dôvodu správnej funkčnosti riadiacej jednotky je potrebné, aby vstupné hodnoty boli odosielané na vstup riadiacej jednotky v čase, keď robot vykonáva pohyb. Nakoľko robot riadený riadiacou jednotkou vykonával pohyb po dobu 10 hodinových taktov, bolo veľmi náročné vypočítať presný čas, kedy vstupné hodnoty odoslať. Navyše doba výpočtu prvej pozície je dlhšia, než doba výpočtu nasledujúcich pozícií.

Riešenie: V rámci verifikačného prostredia bol pridaný parameter „INITIAL_WAIT_TIME“, ktorý určuje prvotnú dobu čakania. Po uplynutí tejto doby odošle verifikačné prostredie na vstup riadiacej jednotky vstupné hodnoty. Hodnota parametra bola na základe experimentov nastavená na hodnotu „3 700 ns“. Po zavedení tohto parametra bola verifikácia úspešne ukončená (viď obrázok 6.13).

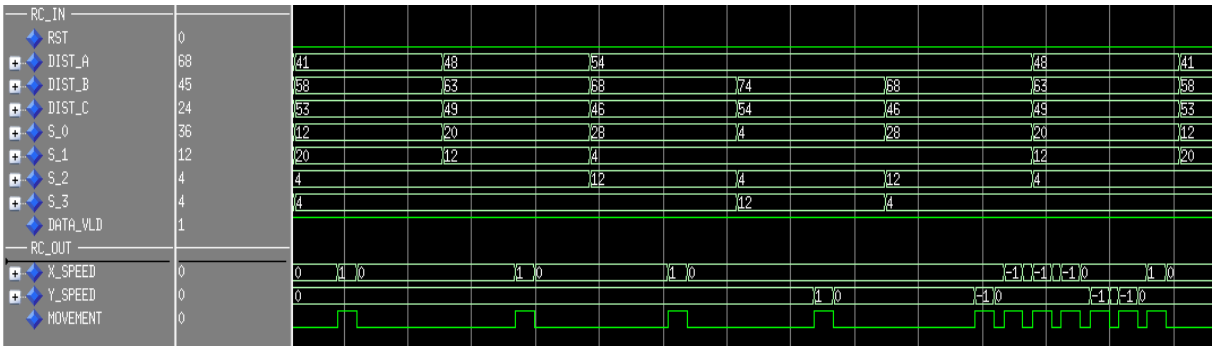
```
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!!        GOAL REACHED        !!!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#
#
# ------------------------------------------------------------------------
# ------------------          SCOREBOARD        --------------------
# ------------------------------------------------------------------------
#
#  MATCHES:        120
#
#  MISMATCHES:          0
#
#  VERIFICATION ENDED CORRECTLY, ROBOT REACHED GOAL COORDINATES
```

Obr. 6.13: Test 1 - úspešné ukončenie verifikácie.

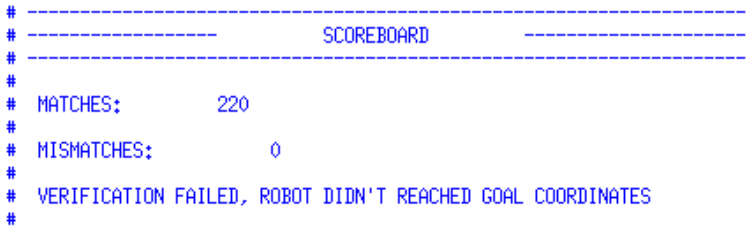
6.7.2 Test 2

Počas vykonávania testu 2 boli odhalené nasledujúce chybové stavy:

Chyba č. 2: Počas hľadania cesty v bludisku špecifikovanom v teste 2 nastala situácia, kedy sa robot musel vrátiť po trase, ktorú už raz absolvoval, nakoľko sa dostal do slepej uličky. Keďže robot túto trasu už raz prešiel, výpočet potrebný pre určenie nasledujúcej súradnice pohybu nebolo potrebné realizovať. Z tohto dôvodu vznikla situácia, kedy hodnota nastavenia rýchlosti v príslušnom smere bola z riadiacej jednotky odoslaná už po piatich hodinových taktoch. Verifikačné prostredie teda nemohlo vykonať porovnanie vo vopred stanovenom čase a verifikáciu ukončilo ako neúspešnú. Túto situáciu ilustrujú obrázky 6.14 a 6.15.



Obr. 6.14: Test 2 - priebeh signálov v ModelSime.



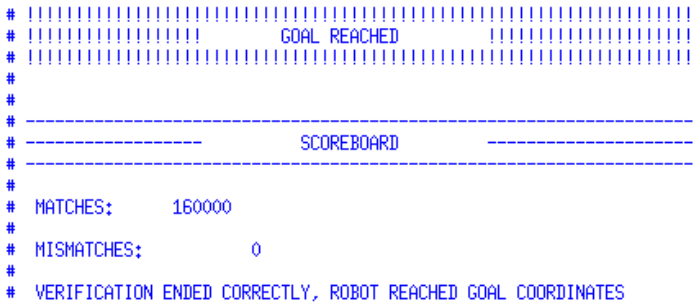
Obr. 6.15: Test 2 - neúspešné ukončenie verifikácie.

Riešenie: Pri analýze zdrojového kódu riadiacej jednotky bolo zistené, že doba počas ktorej roboy vykonáva pohyb je nastavovaná pomocou konštanty „MAX_COUNT“, ktorá sa nastavuje v rámci bloku ECU (Engine Control Unit). Pri vývoji samotnej riadiacej jednotky bola táto konštanta nastavená na hodnotu 10 hodinových taktov z dôvodu rýchleho testovania. V reálnom nasadení riadiacej jednotky v FPGA sa však táto konštanta nastavuje na hodnotu niekoľko miliónkrát vyššiu. Z dôvodu verifikácie bola teda táto konštanta upravená na hodnotu „5000“ hodinových taktov. Na základe tejto zmeny bolo potrebné upraviť parametre verifikačného prostredia. Upravené hodnoty parametrov sú uvedené v tabuľke 6.4.

Názov parametra	Pôvodná hodnota parametra	Nová hodnota parametra
WAIT_TIME	2 300 ns	151 000 ns
INITIAL_WAIT_TIME	3 700 ns	125 000 ns
TRANS_COMPARE_COUNT	10	5 000

Tabuľka 6.4: Úprava hodnôt verifikačného prostredia.

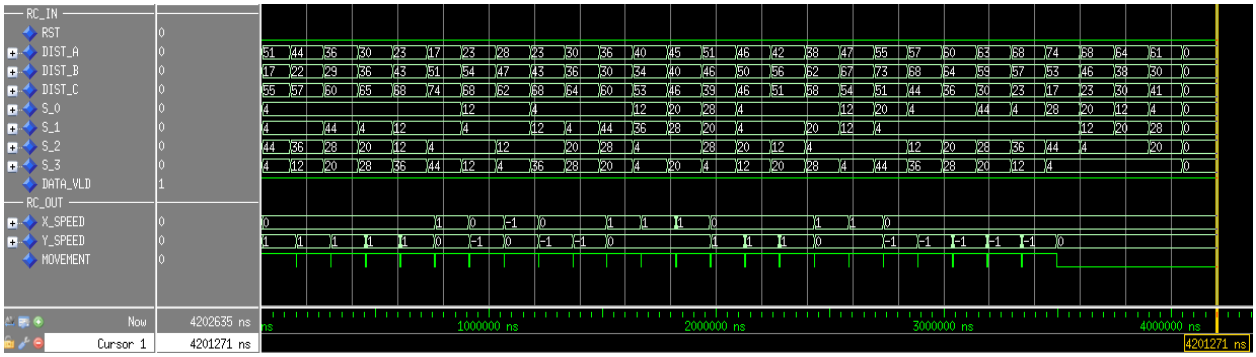
Po vykonaní vyššie uvedených úprav bola verifikácia úspešne ukončená. Túto situáciu ilustruje obrázok 6.16.



Obr. 6.16: Test 2 - úspešné ukončenie verifikácie.

6.7.3 Test 3

Chyba č. 3: Počas hľadania cesty v bludisku, ktoré je špecifikované v teste 3, bola odhalená chyba. Pri priechode bludiskom sa robot prestal pohybovať po tom, keď sa presunul na súradnicu <6,1>. Robot sa ďalej nepohyboval napriek tomu, že obdržal korektné údaje v korektnom čase. Na základe prijatých hodnôt mal vykonať pohyb v smere na západ. Túto chybu ilustrujú obrázky 6.17 a 6.18.



Obr. 6.17: Test 3 - priebeh signálov v ModelSime.

```
# ----------------------------------------------------------------
# ------------------          SCOREBOARD         ------------------
# ----------------------------------------------------------------
#
#  MATCHES:      115000
#
#  MISMATCHES:           0
#
#  VERIFICATION FAILED, ROBOT DIDN'T REACHED GOAL COORDINATES
#
```

Obr. 6.18: Test 3 - neúspešné ukončenie verifikácie.

Riešenie: Pri analýze zdrojového kódu riadiacej jednotky bolo zistené, že v rámci bloku PFU (Path Finding Unit), ktorý realizuje algoritmus hľadania cesty v bludisku sa nachádzajú konštanty „X_TARGET“ a „Y_TARGET“. Tieto konštanty mali napevno priradené hodnoty „6“ a „1“. Na základe hodnôt týchto dvoch konštánt, riadiaca jednotka nesprávne ukončila svoju činnosť pri hľadaní cesty v bludisku. Po úprave príslušných konštánt na požadované hodnoty („X_TARGET = 3“ a „Y_TARGET = 1“) bola verifikácia úspešne ukončená (viď obrázok 6.19). Existencia takýchto signálov v implementácii riadiacej jednotky robota je nezmyselná, a preto sa v budúcnosti táto situácia vyrieši tak, že konštanty sa odstránia a nahradia sa vstupnými signálmi v rozhraní riadiacej jednotky robota.

```
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!        GOAL REACHED        !!!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#
#
# ----------------------------------------------------------------
# ------------------          SCOREBOARD         ------------------
# ----------------------------------------------------------------
#
#  MATCHES:      130000
#
#  MISMATCHES:           0
#
#  VERIFICATION ENDED CORRECTLY, ROBOT REACHED GOAL COORDINATES
```

Obr. 6.19: Test 3 - úspešné ukončenie verifikácie.

6.7.4 Test 4

V rámci vykonávania testu 4 neboli odhalené žiadne chyby. Riadiaca jednotka pri hľadaní cesty zo štartovacej pozície do cieľovej pozície vykonala očakávaný počet krokov (28).

```
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!        GOAL REACHED        !!!!!!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#
#
# ------------------------------------------------------------------------
# ------------------          SCOREBOARD         ---------------------
# ------------------------------------------------------------------------
#
#  MATCHES:      140000
#
#  MISMATCHES:           0
#
#  VERIFICATION ENDED CORRECTLY, ROBOT REACHED GOAL COORDINATES
```

Obr. 6.20: Test 4 - úspešné ukončenie verifikácie.

6.7.5 Test 5

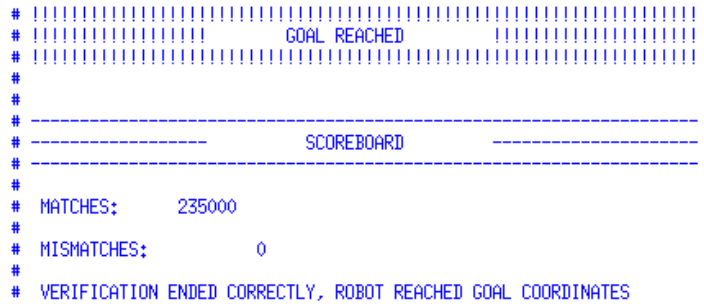
Test 5 využíva pre overenie korektnosti riadiacej jednotky bludisko o rozmeroch 16 x 16 políčok. Z tohto dôvodu bola potrebná malá modifikácia referenčného modelu. V rámci hlavičkového súboru „`golden_model.h`“ bolo potrebné zmeniť hodnoty konštánt „`x_dimension`“ a „`y_dimension`“ na hodnotu „16“ a vykonať preklad. Nakoľko hodnoty, ktoré sa budú pre výpočet pozície využívať budú mať hodnotu väčšiu ako „127“, je potrebné navýšiť veľkosť prenášaných správ zo simulácie virtuálneho prostredia pre pohyb robota (Player/Stage) na dvojnásobnú hodnotu. Jednotlivé položky prenášaných správ nebudú mať veľkosť „1 Bajt“ ale „2 Bajty“.

Nakoľko riadiaca jednotka robota bola nastavená tak, že vedela pracovať iba s bludiskom o rozmeroch 8 x 8 políčok, bolo potrebné vykonať modifikáciu jej zdrojového kódu. Po analýze zdrojového kódu bolo zistené, že je potrebné vykonať modifikáciu konštánt, ktoré sa nachádzajú v zdrojovom kóde bloku PEU (Position Evaluation Unit). Modifikácia konštánt je uvedená v tabuľke 6.5.

Názov konštanty	Pôvodná hodnota konštanty	Nová hodnota konštanty
X_MAX	4 096	16 384
EXP_X_MAX	6	7
Y_MAX	4 096	16 384
EXP_Y_MAX	6	7

Tabuľka 6.5: Modifikácia konštánt PEU v riadiacej jednotke.

Po aplikovaní vyššie uvedených modifikácií bola vykonaná úspešná verifikácia (viď obrázok 6.21). Robot riadený riadiacou jednotkou vykonal pri hľadaní cesty v bludisku predpokladaný počet krokov (47). V rámci testu nebola odhalená žiadna chyba. Na základe tohto testu bola demonštrovaná schopnosť riadiacej jednotky korektne pracovať aj s bludiskom o rozmeroch 16 x 16 políčok, ako z pohľadu verifikačného prostredia, tak aj z pohľadu riadiacej jednotky.



Obr. 6.21: Test 5 - úspešné ukončenie verifikácie.

6.8 Analýza pokrytia kódu

Pokrytie kódu tvorí veľmi dôležitú metriku, ktorá sa kontroluje v rámci verifikácie. Poskytuje nám informáciu o tom, aká veľká časť zdrojového kódu verifikovanej jednotky bola v rámci konkrétneho testu počas verifikácie overená. V tabuľke 6.6 sú uvedené hodnoty pokrytia zdrojového kódu riadiacej jednotky robota, ktoré boli získané pre jednotlivé testy. V tabuľke je uvedený aj počet transakcií, ktoré boli v rámci testu realizované. V rámci overovania pokrytia kódu sa kontrolovali nasledujúce parametre:

- Príkazy (angl. *Statements*) - príkazy (riadky zdrojového kódu), ktoré boli vykonané. Do pokrytia sa nepočítajú komentáre a riadiace štruktúry (ako napr. `library, generic, port, module, begin, end,` atď.).
- Vetvy (angl. *Branches*) - vykonanie každej vetvy podmieňovacieho príkazu.
- Podmienky (angl. *Conditions*) - overenie platnosti splnenia podmienky v podmieňovacom príkaze.
- Výrazy (angl. *Expressions*) - overenie nastavenia hodnôt signálov, ktoré môžu ovplyvniť výpočet.

Súčasťou hodnotenia pokrytia kódu sú správy s podrobnými informáciami o dosiahnutom pokrytí konkrétnych blokov riadiacej jednotky. Z dôvodu ich veľkosti sú priložené na CD nosiči v rámci prílohy A.

Pokrytie kódu	Test 1	Test 2	Test 3	Test 4	Test 5
Príkazy (angl. *Statements*)	93,0 %	93,6 %	93,6 %	93,6 %	93,6 %
Vetvenie (angl. *Branches*)	94,4 %	95,2 %	95,2 %	95,2 %	95,1 %
Podmienky (angl. *Conditions*)	84,8 %	85,8 %	84,8 %	84,8 %	84,8 %
Výrazy (angl. *Expressions*)	74,6 %	74,6 %	74,6 %	74,6 %	74,6 %
Počet transakcií	13	33	27	29	48

Tabuľka 6.6: Pokrytie kódu riadiacej jednotky v rámci testov.

Pri porovnaní hodnôt dosiahnutého pokrytia kódu medzi jednotlivými testami, ktoré boli vykonané je vidieť, že pokrytie v rámci testu 1 je menšie, než u iných testov. Pri analýze vykonaného testu je vidieť, že robot pri hľadaní cesty v bludisku nikdy neodbočil smerom na západ. Po analýze podrobnej správy o vykonanom pokrytí bolo odhalené, že v porovnaní

s ostatnými testami nastalo nižšie pokrytie zdrojového kódu v rámci bloku „ECU“ (Engine Control Unit), teda bloku, ktorý riadi pohyb robota.

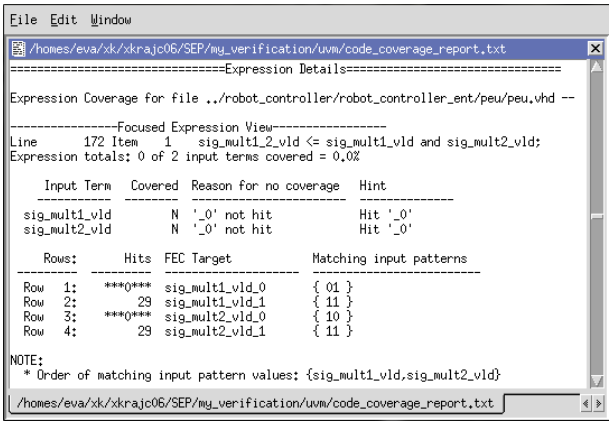
Pri verifikácii sa vždy kladie dôraz na to, aby pokrytie kódu dosahovalo čo najvyššie hodnoty, ideálne 100 %. Verifikačné prostredie zasiela na vstupné rozhranie riadiacej jednotky robota sedem vstupných hodnôt (vzdialenosť od bodov A, B, C a hodnoty senzorov S0, S1, S2, S3). Iba na základe týchto hodnôt môže ovplyvňovať fungovanie riadiacej jednotky. Z vykonaných experimentov je zrejmé, že kombináciou vstupných hodnôt nie je možné z pohľadu verifikačného prostredia dosiahnuť pokrytie vyššie, než je uvedené v tabuľke 6.6. Na základe tohto bola vykonaná analýza podrobných správ o dosiahnutom pokrytí s cieľom odhaliť tie časti zdrojového kódu, ktoré neboli pokryté. Výsledky analýzy sú nasledujúce:

- Najväčšiu časť nepokrytého kódu tvoria vetvy riadiacej štruktúry výberových príkazov (`CASE <výraz> IS WHEN ... => <príkaz>`) v rámci ktorých sa kontroluje hodnota výrazov. Pri návrhu obvodu musia byť v rámci tohto riadiaceho príkazu kontrolované všetky možné hodnoty kontrolovaného výrazu, z dôvodu ošetrenia možných chýb. Kontrola na hodnoty výrazu, ktoré nie sú z pohľadu funkcionality obvodu rozhodujúce, sa v jazyku VHDL spravidla rieši pomocou vetvy „`WHEN others => <príkaz>`“. Nakoľko riadiaca jednotka pracovala korektne, vykonanie týchto vetiev nikdy nenastalo a teda v rámci analýzy pokrytia kódu neboli tieto časti nikdy pokryté. Prípadné odstránenie týchto častí kódu by mohlo spôsobiť problémy z pohľadu správnej funkčnosti riadiacej jednotky.
- Ďalšiu časť nepokrytého kódu tvoria vetvy riadiacej štruktúry podmieňovacích príkazov, v ktorých sa kontroluje hodnota výrazu voči konštante. Nepokryté časti zdrojových kódov konkrétnych súborov sú uvedené v tabuľke 6.7.

Súbor	Číslo riadku	Zdrojový kód
move.vhd	163	`if COUNT = MAX_COUNT then COUNT:=(OTHERS => '0');`
wb_sif_write_master.vhd	89	`if (COUNT_INT = 5) then COUNT_INT := 0;`
wb_mu_read_master.vhd	210	`if (COUNT_INT = 15) then COUNT_INT := 0;`
wb_pfu_read_master.vhd	135	`if (n = const_in_num) then`
	207	`if (COUNT_INT = 15) then COUNT_INT := 0;`

Tabuľka 6.7: Nepokryté časti zdrojového kódu.

- Nepokrytie výrazov predstavuje poslednú časť nepokrytého kódu riadiacej jednotky. V rámci zdrojového kódu sa kontrolujú kombinácie vstupných hodnôt, ktoré ovplyvňujú hodnoty výrazu. Nakoľko počet nepokrytých výrazov v rámci analýzy kódu bol väčší, na obrázku 6.22 je uvedený príklad nepokrytia konkrétneho výrazu v rámci zdrojového kódu bloku „PEU“ (Position Evaluation Unit - jednotka pre vyhodnotenie polohy robota). Signál „`sig_mult1_2vld`“ nastavuje svoju hodnotu na základe hodnoty výrazu „`sig_mult1_vld AND sig_mult2_vld`“. Nakoľko v rámci výpočtu nikdy nenastala situácia, kedy by hodnota signálu „`sig_mult_1_vld`“ alebo signálu „`sig_mult_2_vld`“ nadobúdala hodnotu „0“, výraz „`sig_mult1_2vld <= sig_mult1_vld AND sig_mult2_vld`“ je označený, ako nepokrytý.



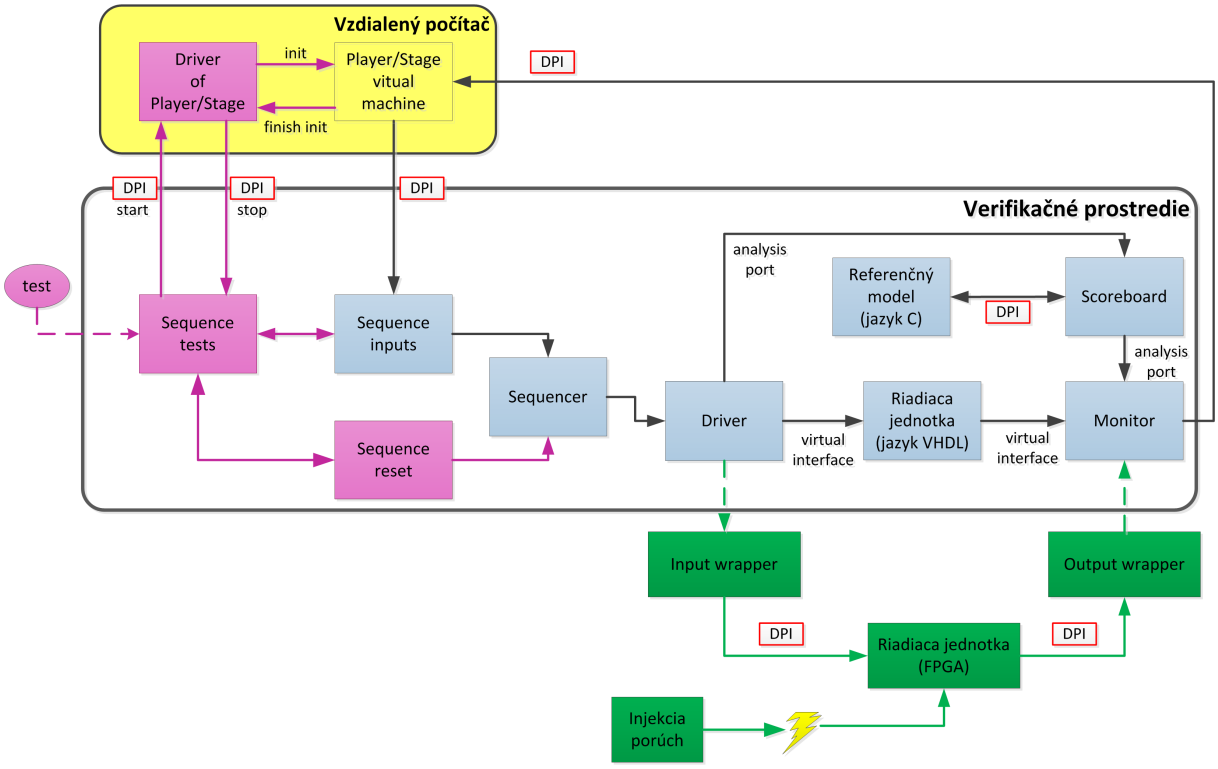
Obr. 6.22: Nepokrytie výrazu.

Nepokryté časti zdrojového kódu riadiacej jednotky je potrebné podrobiť detailnej analýze a vykonať úpravy tých častí kódu, ktoré nedosahujú plné pokrytie. V rámci budúcej práce s riadiacou jednotkou, ktorej súčasťou bude overovanie zabezpečenia jednotky voči poruchám, ktoré budú do nej injektované je dôležité, aby pokrytie kódu bolo čo najvyššie. V prípade, ak injektovaná porucha ovplyvní funkčnosť riadiacej jednotky v mieste, ktoré nie je pokryté, bude veľmi náročné odhaliť, aký vplyv injektovaná porucha na fungovanie riadiacej jednotky ako celku v skutočnosti mala.

Kapitola 7

Budúca práca

Ako bolo uvedené v kapitole 2, v rámci výskumnej skupiny Diagnostika na ÚPSY Fakulty informačných technológií VUT, prebieha výskum zameraný na overovanie odolnosti systémov proti poruchám. V procese overovania bude využívaná softvérová simulácia, v rámci ktorej sa do overovaného systému budú umelo injektovať rôzne typy porúch. Tvorba verifikačného prostredia za účelom funkčnej verifikácie riadiacej jednotky robotického systém je jednou z dielčich činností, ktoré sú v rámci výskumnej skupiny realizované. Verifikačné prostredie vytvorené v tejto práci bude ako súčasť budúcej práce využité pre automatizovanú verifikáciu riadiacej jednotky, do ktorej budú injektované poruchy za účelom overenia miery odolnosti tejto jednotky proti injektovaným poruchám.



Obr. 7.1: Schéma automatizovaného verifikačného prostredia.

Schéma uvedená na obrázku 7.1 predstavuje koncept využitia verifikačného prostredia ako súčasť automatizovaného systému verifikácie riadiacej jednotky robotického systému. Jednotlivé bloky, ktoré sú v schéme označené bledomodrou farbou predstavujú komponenty, ktoré boli implementované v rámci tejto práce. Popis činnosti verifikačného prostredia je nasledovný: Virtuálne prostredie pre pohyb robota (Player/Stage) prostredníctvom DPI rozhrania odosiela informácie, na základe ktorých riadiaca jednotka vykonáva pohyb. Komponenta „Sequence Inputs“ z prijatých hodnôt vytvára vstupné transakcie, ktoré sú prostredníctvom komponenty „Sequencer“ odosielané do komponenty „Driver“. Transakcie, ktoré komponenta „Driver“ príjme, sú prostredníctvom „analysis portu“ odosielané do komponenty „Scoreboard“. Zároveň sa vstupné transakcie prevádzajú na signálovú reprezentáciu a prostredníctvom virtúalneho rozhrania (angl. *virtual interface*) sú odosielané na vstup riadiacej jednotky. Výstupné signály z riadiacej jednotky sa transformujú do podoby výstupnej transakcie, ktorá sa odosiela prostredníctvom „analysis portu“ do komponenty „Scoreboard“ a zároveň sa odosiela prostredníctvom DPI rozhrania do virtuálneho prostredia pre pohyb robota (Player/Stage). Komponenta „Scoreboard“ prostredníctvom DPI rozhrania odošle vstupnú transakciu do referenčného modelu a následne z neho príjme vypočítané hodnoty vo forme výstupnej transkacie. Transakciu, ktorú obdržal od komponenty „Monitor“ a transakciu, ktorú prijal z referenčného modelu porovná a vyhodnotí. Tento proces sa opakuje, kým robot nedosiahne cieľové súradnice alebo počas verifikácie nevznikne chyba.

Pre implementáciu overovania odolnosti riadiacej jednotky robota voči poruchám je potrebné implementovať bloky, ktoré sú v schéme označené zelenou farbou. Blok „Input wrapper“ slúži na transformáciu vstupnej transakcie do formy dátovej jednotky, ktorá bude prostredníctvom DPI rozhrania odosielaná na vstup riadiacej jednotky. Riadiaca jednotka bude umiestnená v hardvérovom obvode FPGA. Do tejto jednotky budú injektované poruchy a prostredníctvom DPI rozhrania budú výstupné hodnoty zasielané z riadiacej jednotky do komponenty „Output wrapper“. Táto komponenta transformuje prijaté dátové jednotky do formy výstupných transakcií. Výstupné transakcie následne odošle do komponenty „Monitor“, ktorá prijaté transakcie odošle prostredníctvom DPI rozhrania do virtuálneho prostredia pre pohyb robota a zároveň ich odošle do komponenty „Scoreboard“. V rámci jednotky „Scoreboard“ bude realizované porovnávanie s očakávanými hodnotami z riadiacej jednotky, ktorá nebude obsahovať funkčné chyby (odladená verzia implementácie riadiacej jednotky v jazyku VHDL z verifikačnej fázy), prípadne s hodnotami z referenčného modelu implementovanom v jazyku C. Na základe výsledkov tohto porovnania bude možné určiť, či injektovaná porucha spôsobila chybu v riadiacej jednotke alebo nie.

Za účelom vytvorenia plne automatizovaného prostredia pre verifikáciu riadiacej jednotky je potrebné implementovať bloky, ktoré sú v rámci schémy označené ružovou farbou. Celý proces bude riadený pomocou komponenty „Sequence tests“. V rámci tejto komponenty budú implementované tzv. „hierarchické sekvencie“, ktoré budú slúžiť na riadenie jednotlivých podúrovňových sekvencií. Vstupom pre túto jednotku budú testovacie scenáre. Na základe týchto testovacích scenárov sa vytvorí konfiguračný súbor určený pre inicializáciu virtuálneho prostredia pre pohyb robota (Player/Stage). Prostredníctvom DPI rozhrania sa tento konfiguračný súbor odošle do komponenty „Driver of Player/Stage“, ktorý inicializuje virtuálne prostredie. Po úspešnej inicializácii túto informáciu oznámi naspäť komponente „Sequence tests“. Na základe tejto informácie odošle požiadavku na reset verifikačného prostredia a reset riadiacej jednotky robota prostredníctvom „Sequence reset“. Po resetovaní sa spustí vstupná sekvencia „Sequence Inputs“, ktorá prijíma hodnoty z virtuálneho prostredia. Proces verifikácie následne prebieha tak, ako je uvedený v rámci implementačnej časti tejto práce. Test sa ukončí potom, čo robot riadený riadiacou jednotkou dosiahne cieľové

súradnice, prípadne v rámci verifikácie vznikne chyba, ktorá neumožní robotovi dosiahnuť cieľové súradnice. Po ukončení testu sa s využitím vyššie uvedených krokov automaticky spustí ďalší test.

Týmto princípom bude realizovaná automatizácia verifikačného prostredia.

Kapitola 8

Záver

Cieľom práce bol návrh a implementácia verifikačného prostredia pomocou metodiky UVM pre riadiacu jednotku robotického systému. V úvodnej časti práce bol predstavený zámer a motivácia k tvorbe verifikačného prostredia v rámci výskumnej činnosti, ktorá je realizovaná na Ústave počítačových systémov Fakulty informačných technológií VUT v Brne.

Predmetom verifikácie bola riadiaca jednotka robotického systému, ktorá hľadá cestu v bludisku. Na základe teoretických informácií o funkčnej verifikácií, jazyka SystemVerilog a metodike UVM bolo v rámci práce navrhnuté a implementované verifikačné prostredie pre riadiacu jednotku robota. Súčasťou verifikačného prostredia je referenčný model, ktorý vykonáva rovnakú funkciu ako riadiaca jednotka. Bol navrhnutý na základe špecifikácie, z ktorej vychádzala aj implementácia riadiacej jednotky. Referenčný model slúži na overovanie funkčnosti riadiacej jednotky. Súčasťou implementácie je simulácia virtuálneho prostredia pre pohyb robota (Player/Stage), ktorá slúži na zasielanie vstupných informácií do verifikačného prostredia. V rámci práce je popísaný postup verifikácie riadiacej jednotky spolu s demonštráciou možností odhalenia chýb. Za účelom overenia funkčnosti riadiacej jednotky bolo vytvorených päť testov. Na základe výsledkov testovania boli odhalené chyby v riadiacej jednotke, ktoré boli následne opravené. Po vykonaných úpravách zdrojového kódu riadiacej jednotky bola verifikácia vo všetkých testovacích scenároch úspešne ukončená, robot v bludisku dosiahol cieľové súradnice. Po úspešnom overení funkčnosti riadiacej jednotky bola vykonaná analýza pokrytia jej zdrojového kódu. Vo výsledku analýzy sú uvedené nepokryté časti zdrojového kódu a možné dôvody ich nepokrytia. V závere textu je prezentovaný koncept budúcej práce. Verifikačné prostredie, ktoré bolo vytvorené v rámci tejto práce tvorí základnú súčasť celého konceptu, ktorého cieľom je v budúcnosti vytvoriť automatizovaný systém pre overovanie odolnosti riadiacej jednotky robotického systému proti poruchám.

Princíp využitia funkčnej verifikácie v oblasti odolnosti systémov proti poruchám je možné považovať za inovatívny, nakoľko podobný prístup nebol zatiaľ nikde realizovaný. Je dôležité poznamenať, že tento prístup bol navrhnutý generický tak, aby bolo možné verifikovať akýkoľvek číslicový obvod a pripraviť ho na následné overovanie odolnosti proti poruchám. V tejto práci bol uvedený princíp demonštrovaný na príklade riadiacej jednotky.

Výsledkom tejto práce je verifikačné prostredie, ktoré slúži na overenie korektnosti riadiacej jednotky a množina vstupných hodnôt s informáciou o dosiahnutom pokrytí zdrojového kódu. Nakoľko dosiahnuté pokrytie kódu nebolo v žiadnom z vykonaných testov 100 %-né, je dôležité, aby pred budúcou realizáciou overovania odolnosti riadiacej jednotky robota proti poruchám bola vykonaná úprava jej zdrojového kódu tak, aby sa eliminovali tie časti, ktoré sa v rámci vykonaných testov nepokryli. V prípade, ak injektovaná porucha ovplyvní činnosť riadiacej jednotky v mieste, ktoré nie je pokryté, bude veľmi náročné

odhaliť aký vplyv injektovaná porucha na fungovanie riadiacej jednotky ako celku v skutočnosti mala. V rámci úpravy zdrojového kódu riadiacej jednotky odporúčam, aby bola zavedená možnosť parametrizovania nasledujúcich hodnôt: nastavenie cieľových súradníc, nastavenie veľkosti bludiska. V aktuálnej implementácii riadiacej jednotky sú tieto hodnoty nastavené napevno v rámci zdrojového kódu a v prípade ich zmeny je potrebné vykonať opätovný preklad zdrojových súborov riadiacej jednotky robota.

Literatúra

[1] UVM Cookbook. 2012, <https://verificationacademy.com/cookbook/uvm>.

[2] IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. *IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009)*, Feb 2013.

[3] Cohen, B.: *Component Design by Example: A Step-by-step Process Using VHDL with UART as Vehicle*. VhdlCohen Publ., 2001, ISBN 9780970539403.

[4] Geffroy, J.-C.; Motet, G.: *Design of Dependable Computing Systems*. Hingham, MA, USA: Kluwer Academic Publishers, 2002, ISBN 1-4020-0437-0.

[5] Glasser, M.: *Open Verification Methodology Cookbook*. Springer, 2009, ISBN 9781441909695.

[6] Hlavička, J.: *Číslicové systémy odolné proti poruchám*. Praha: ČVUT, 1992, ISBN 8001008525.

[7] Koren, I.; Krishna, C. M.: *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, ISBN 0120885255, 9780080492681.

[8] Podivinsky, J.: *VHDL návrh řídicí jednotky robota určeného pro samočinný pohyb v bludišti*. Diplomová práce.

[9] Podivínský, J.; Čekan, O.; Šimková, M.; a j.: The Evaluation Platform for Testing Fault-Tolerance Methodologies in Electro-mechanical Applications. In *17th Euromicro Conference on Digital Systems Design*, IEEE Computer Society, 2014, ISBN 978-1-4799-5793-4, s. 312–319.
URL <http://www.fit.vutbr.cz/research/view_pub.php?id=10665>

[10] Stevens, W. R.: *UNIX Network Programming: Networking APIs: Sockets and XTI*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1997, ISBN 013490012X.

[11] Sutherland, S.; Mills, D.: HDVL + = (HDL & HVL) SystemVerilog 3.1 The Hardware Description AND Verification Language.

Dodatok A

Obsah CD

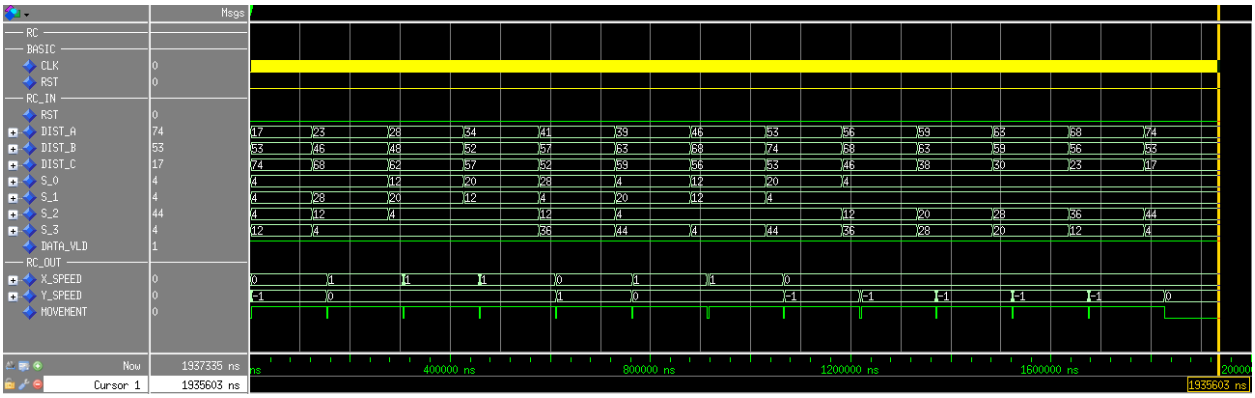
Na priloženom CD sa nachádzajú nasledujúce súbory:

- Diplomová práca vo formáte pdf v adresári `/DP/pdf`.
- Zdrojové kódy diplomovej práce vo formáte LaTeX v adresári `/DP/latex`.
- Zdrojové kódy diplomovej práce vo formáte LyX v adresári `/DP/lyx`.
- Zdrojové kódy verifikačného prostredia v adresári `/verification_enviroment`.
- Testovacie scenáre reprezentované vo forme vstupných hodnôt v adresári `/test_inputs`.
- Analýza pokrytia zdrojového kódu riadiacej jednotky robotického systému v adresári `/test_coverage`.

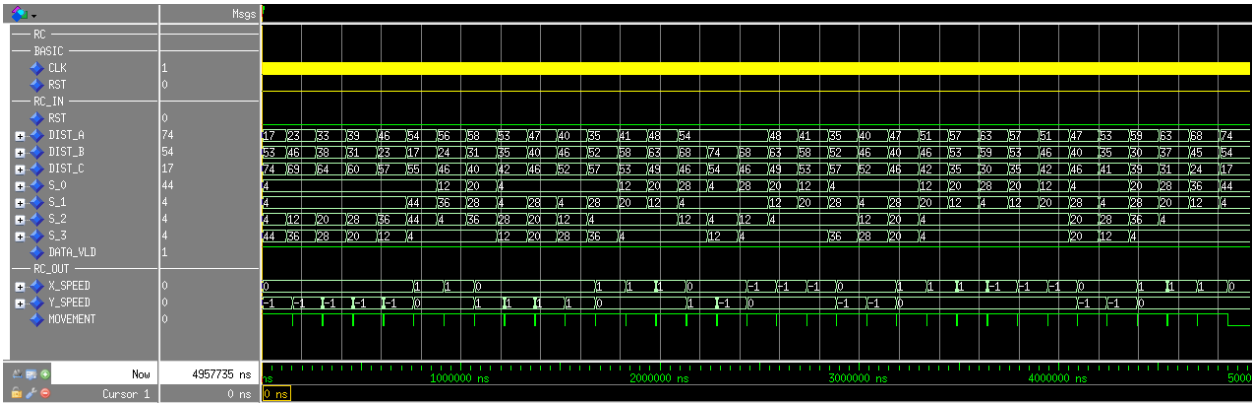
Dodatok B

Verifikačné prostredie v ModelSime

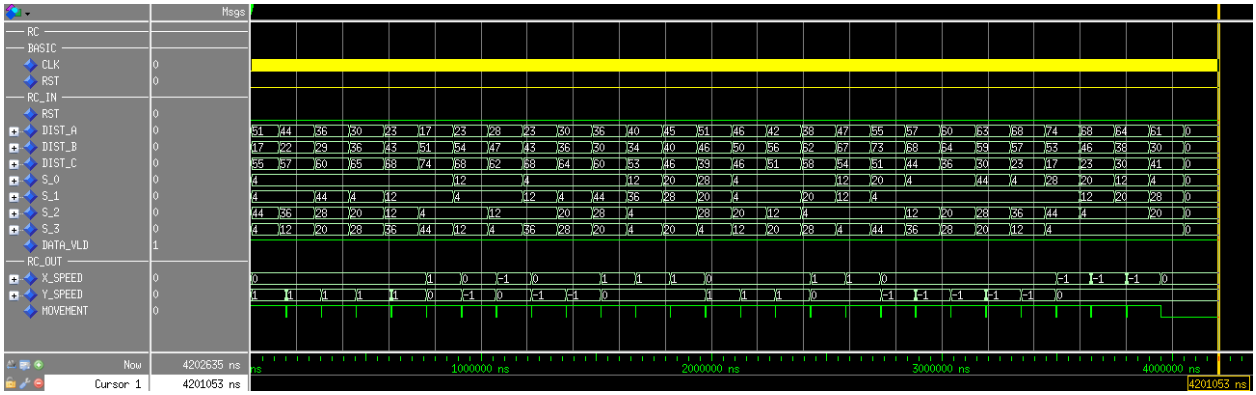
Nástroj ModelSim poskytuje prostredie pre verifikáciu v jazyku SystemVerilog. Na nižšie uvedených obrázkoch sa nachádzajú priebehy signálov pre jednotlivé testy, ktoré boli vykonané za účelom verifikácie riadiacej jednotky robota.



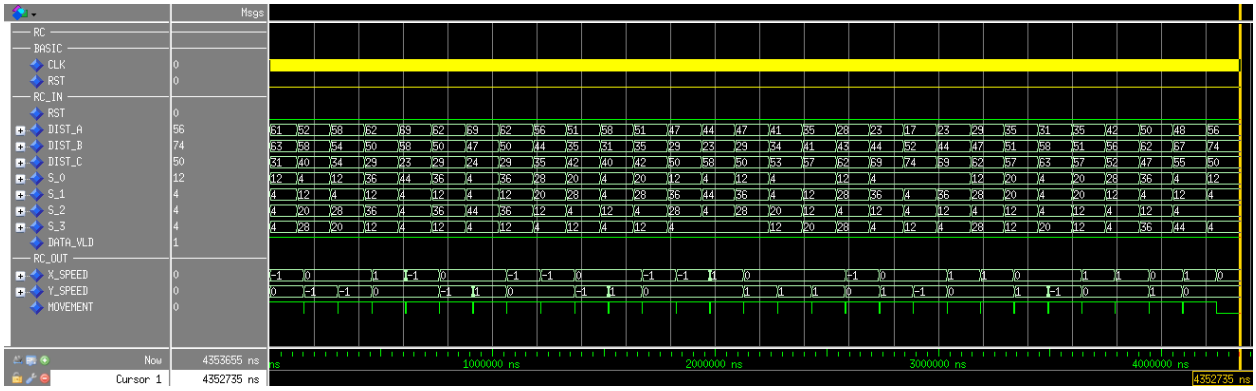
Obr. B.1: Verifikácia riadiacej jednotky - Test 1.



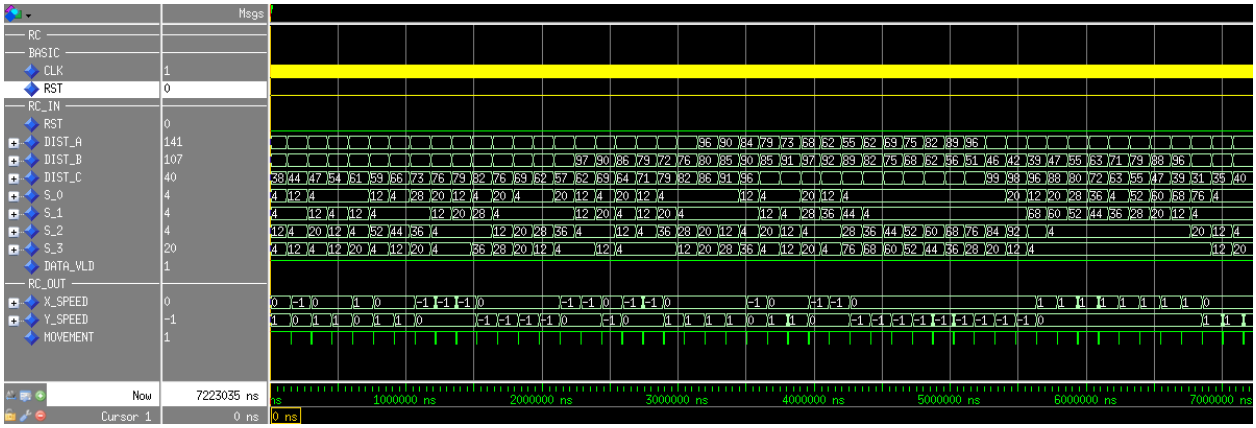
Obr. B.2: Verifikácia riadiacej jednotky - Test 2.



Obr. B.3: Verifikácia riadiacej jednotky - Test 3.



Obr. B.4: Verifikácia riadiacej jednotky - Test 4.



Obr. B.5: Verifikácia riadiacej jednotky - Test 5.