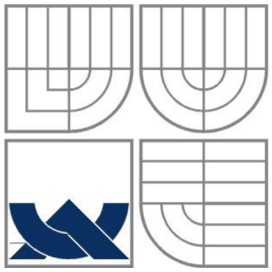
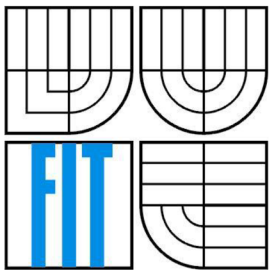


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VHDL NÁVRH ŘÍDICÍ JEDNOTKY ROBOTA URČENÉHO PRO SAMOČINNÝ POHYB V BLUDIŠTI

VHDL DESIGN OF ROBOT CONTROLLER FOR AUTONOMOUS ROBOT MOVEMENT IN MAZE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAKUB PODIVÍNSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARCELA ŠIMKOVÁ
SUPERVISOR

BRNO 2013

Abstrakt

V této práci je popsán návrh a implementace řídicí jednotky robota určeného pro samočinný pohyb v bludišti. Jedná se o exemplární systém, který je určen pro testování a ověřování metodik pro zajištění odolnosti proti poruchám. Součástí práce je uvedení do problematiky spolehlivosti číslicových systémů, především systémů založených na technologii programovatelných hradlových polí (FPGA). Práce se také zabývá představením technik pro zajištění odolnosti číslicových systémů proti poruchám, pozornost je věnována možnostem FPGA v této oblasti včetně představení možností využití částečné dynamické rekonfigurace.

Abstract

This master thesis describes design and implementation of a robot controller for autonomous movement in a maze. Robot represents an exemplary system, which is designed for testing and validation of fault-tolerance methodologies. A part of this work contains introduction to reliability of digital systems, especially those which are based on Field Programmable Gate Array (FPGA). Moreover, this introduces techniques that ensure robustness against faults in digital systems; attention is devoted to the usage of FPGA technology in this area and a technique called partial dynamic reconfiguration.

Klíčová slova

Řídicí jednotka, robot, číslicový systém, FPGA, spolehlivost, odolnost proti poruchám, dynamická rekonfigurace.

Keywords

Controller, robot, digital system, FPGA, reliability, fault tolerance, dynamic reconfiguration.

Citace

Podivínský Jakub: VHDL návrh řídicí jednotky robota určeného pro samočinný pohyb v bludišti, diplomová práce, Brno, FIT VUT v Brně, 2013

VHDL návrh řídicí jednotky robota určeného pro samočinný pohyb v bludišti

Prohlášení

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

.....
Jakub Podivínský
22. května 2013

Poděkování

Na tomto místě bych chtěl poděkovat vedoucí mé práce Ing. Marcele Šimkové za cenné rady a připomínky při tvorbě diplomové práce. Rád bych poděkovat také Ing. Janu Kaštilovi za odborné konzultace.

Obsah

Úvod

Spolehlivost je důležitým parametrem u všech číslicových systémů, se kterými se setkáváme v našem každodenním životě. Na různé číslicové systémy jsou kladeny různé nároky na spolehlivost. Vysoká spolehlivost je nutná především u systémů, jejichž selhání by mohlo způsobit velké materiální škody a v horším případě dokonce ohrozit zdraví a život mnoha lidí. Typickým příkladem jsou průmyslové systémy, systémy řízení moderních automobilů nebo letadel. U systémů pracujících ve vesmíru je také vyžadována vysoká míra spolehlivosti, protože pracují v extrémních podmínkách a jejich selhání by vyžadovalo velmi vysoké náklady na opravu.

Jedním ze současných trendů je neustálé zvyšování komplexnosti číslicových systémů, což vede na jejich vyšší integraci. Avšak negativním jevem je, že s vysokou integrací zároveň klesá spolehlivost číslicových systémů a je nutné hledat cesty, jak ji opětovně zvýšit. Existují dva přístupy: předcházení poruchám a odolnost proti poruchám, neboli tolerance poruch. První přístup se snaží potlačit možnost vzniku poruchy, ale naráží na fyzikální překážky a vede k vysokým nákladům. Druhý přístup, tolerance poruch, bere možnost výskytu poruch v úvahu a respektuje je při návrhu systému. Systém je navrhován tak, aby se vliv poruch v systému neprojevil, případně jen minimálně. Tento přístup je v současné době intenzivně rozvíjen a vznikají nové metodiky pro toleranci poruch v číslicových systémech.

Cílem této práce je zajistit zvyšování spolehlivosti číslicových systémů založených na technologii programovatelných hradlových polí (*FPGA – Field Programmable Gate Array*). Jádrem práce je návrh a implementace řídící jednotky robota, který bude hledat cestu v bludišti. Řadič robota představuje exemplární systém, který bude demonstrovat použití různých technik pro zajištění odolnosti proti poruchám, zejména technik vyvinutých na Ústavu počítačových systémů (ÚPSY) Fakulty informačních technologií VUT v Brně. Řídící jednotka robota zároveň reprezentuje komplexní systém, který zachytává různé aspekty návrhu číslicových systémů (např. sekvenční a kombinační obvody, sběrnici, různé typy pamětí, atd.), které je nutné zabezpečit různými technikami. Hlavním přínosem této práce je návrh nové metodiky, jak zabezpečit takto složitý systém proti poruchám. Publikace, které se zabývají podobnou úlohou, obvykle demonstrují techniky zabezpečení proti poruchám na malých číslicových obvodech a zabývají se vlivem chyb na jednoduchý číslicový systém. Vytvořená řídicí jednotka řídí simulátor robota a umožňuje zkoumat vliv poruch jak na výstup číslicového systému, tak na simulovanou mechanickou část, kterou tento systém řídí.

Kapitola 1 se zabývá uvedením do problematiky poruch v číslicových systémech. Část 1. kapitoly se věnuje výskytu poruch v FPGA. 2. kapitola je věnována představení dostupných technik pro zajištění odolnosti systémů proti poruchám, především pak systémů založených na FPGA. 3. kapitola se věnuje návrhu řadiče robota na vyšší úrovni abstrakce včetně podrobného popisu jednotlivých bloků. Ve 4. kapitole je podrobně rozebrána implementace celé řídicí jednotky, jsou zde popsány použité techniky a komponenty. Ověření funkčnosti implementované řídicí jednotky se věnuje 5. kapitola, kde je také stručná zmínka o použité vývojové desce a je zde stručně představeno simulační prostředí. Návrhu zajištění odolnosti jednotlivých bloků řídicí jednotky proti poruchám se věnuje 6. kapitola. Shrnutí práce a představení budoucí práce se věnuje kapitola 7.

1 Poruchy v číslicových systémech

Tato kapitola se věnuje úvodu do problematiky zajištění spolehlivosti číslicových systémů a výskytu poruch v číslicových systémech [1], především v systémech založených na FPGA [2]. Část kapitoly je věnována úvodu do technologie FPGA obvodů a vychází z [3] [4].

1.1 Definice spolehlivosti

Abychom se mohli věnovat zajištění spolehlivosti, je nutné nejprve definovat, co je to spolehlivost a jak ji měříme. Spolehlivost sama o sobě není kvantifikovatelná a norma ČSN IEC 50(191) ji definuje jako:

> *"obecnou vlastnost objektu spočívající ve schopnosti plnit požadované funkce při zachování hodnot stanovených provozních ukazatelů v daných mezích a v čase podle stanovených technických podmínek".*

Uvedená definice hovoří o objektu, což je obecný pojem, za který můžeme dosadit celý číslicový systém, jeho libovolně velkou část případně samostatnou součástku. Spolehlivost zřejmě nelze vyjádřit jako jednu vlastnost, která by zahrnovala několik různých hledisek, proto se pro její vyjádření používají atributy spolehlivosti [5], které tvoří ohodnocení dílčích vlastností a souhrnně jimi lze vyjádřit spolehlivost (*dependability*). Jsou to následující:

- **Dostupnost** (*availabillity*)
  - pohotovost k provedení správné služby
- **Spolehlivost** (*reliability*)
  - kontinuita poskytování správné služby
- **Zabezpečenost** (*safety*)
  - absence katastrofických následku pro uživatele a prostředí
- **Důvěrnost** (*security*)
  - absence nepovolených odhalení informace
- **Integrita** (*integrity*)
  - absence nevhodných změn stavu systému
- **Udržovatelnost** (*maintainability*)
  - schopnost procházet opravy a modifikace

V souvislosti se spolehlivostí hovoříme o poruše a chybě:

> *Porucha* je jev, který vede k narušení schopnosti objektu vykonávat požadovanou funkci. Zpravidla za vznikem poruchy stojí nějaká vnější příčina.

> *Chyba* je obvykle důsledkem poruchy, ale ne každá porucha se musí projevit jako chyba. Chyba se projevuje produkováním nesprávných výstupů objektu.

Na výslednou spolehlivost má značný vliv i rozdělení objektů na *neobnovované* a *obnovované*, jinak nazývané také neopravované a opravované [1]. Rozdíl spočívá v možnosti obnovy objektu do bezporuchového stavu. Obnovované objekty je možné opravit a tím zajistit přechod do bezporuchového stavu. Neobnovované objety jsou takové, které nelze opravit, nejsou přístupné (např. kosmické sondy), nebo nejsou opravovány z organizačních důvodů (např. oprava se finančně nevyplatí). Tato práce se zabývá číslicovými systémy založenými na FPGA, které umožňují tzv. rekonfiguraci, čímž rozumíme možnost programovat obvod i za běhu aplikace, a tím i opravu, jedná se tedy o obnovované systémy.

1.2 Metody zajištění spolehlivosti

Zajištění spolehlivosti lze dosáhnout zlepšením všech dílčích ukazatelů spolehlivosti, což je většinou nemožné realizovat, případně jen v omezené míře. Nejlepší vliv na zvýšení spolehlivosti má snížení intenzity poruch, což vede na zlepšení všech důležitých ukazatelů spolehlivosti. Takový zásah je však téměř nerealizovatelný, protože metody snížení intenzity poruch jsou velmi složité a především příliš nákladné. Tento přístup bývá označován jako *předcházení poruchám*.

Odlišným přístupem vedoucím k zajištění spolehlivosti je *odolnost proti poruchám*. Tento přístup bere v úvahu možnost výskytu poruchy a snaží se zamezit, aby se vliv poruchy projevil na chování systému. Při tomto přístupu rozlišujeme poruchu součástky systému a systému jako celku, která se označuje jako selhání systému. Porucha systému je změna v chování, která vede k neschopnosti systému plnit požadovanou funkci.

1.2.1 Předcházení poruchám

Předcházení poruchám je podle [1] teoreticky poměrně podrobně rozpracováno, ale ve většině případů jeho využívání brání fyzikální překážky a vysoké náklady. Existují i metody, které jsou dostupné a zároveň mají pozitivní vliv na spolehlivost systému a lze je aplikovat při návrhu, výrobě i provozu systému. Tolerovat lze vždy pouze omezený počet poruch, proto je dobré při návrhu číslicového systému využít všech dostupných metod předcházení poruchám.

Předcházet poruchám lze při návrhu, výrobě i provozu systému. Při návrhu je vhodné volit spolehlivé součástky a spolehlivou technologii, obojí s ohledem na podmínky, ve kterých bude systém provozován. Vstupní kontrola je klíčová při výrobě, kontrolují se součástky, polotovary a materiály od subdodavatelů. Při výrobě hraje významnou roli také dodržování technologické kázně, průběžné kontroly při výrobě a také testy výsledného výrobku. Při provozu systému je důležité dodržování provozních podmínek. Za provozu přicházejí do styku se systémem i lidé, kteří mohou nevhodným zásahem přispět ke vzniku poruch. V takových případech je vhodné klást důraz na jasnou a srozumitelnou komunikaci systému s člověkem a také na kvalitní dokumentaci, uživatelskou příručku a zaškolení.

1.2.2 Odolnost proti poruchám

Systém odolný proti poruchám je takový, který správně vykonává svou funkci i v případě výskytu poruchy jeho technického vybavení nebo chyb v programech. Termín "správně vykonávat svou funkci" lze chápat různě, obvykle se funkce považuje za správnou při splnění následujících podmínek uvedených v [1]:

1) zpracování dat nebylo zastaveno ani změněno v důsledku poruchy,
2) výsledek je správný,
3) výsledek byl získán v předepsané době.

Systém, který nesplňuje všechny tři uvedené podmínky, ale pouze některé z nich, bývá označován jako *částečně odolný proti poruchám*. Vždy ale musí být splněna první podmínka, tedy požadavek na zachování funkceschopnosti systému.

Podle [6] rozlišujeme dva základní typy technik pro zajištění odolnosti proti poruchám. Prvním typem je *pasivní redundance,* která využívá maskování poruch, které se v systému vyskytnou. V tomto případě obvykle nejsou využívány žádné techniky pro detekci a opravu poruch. Nejčastějším příkladem je technika označovaná jako *TMR (Triple Modular Redundancy)*, která spočívá ve ztrojení části systému a výstupem této části je výsledek s největším zastoupením. Podrobněji je technikám pro zajištění odolnosti proti poruchám věnována 2. kapitola.

Druhým přístupem je *aktivní redundance*, která navíc využívá možnost detekce a opravy poruch. Při tomto přístupu je prováděno testování systému za běhu, které informuje o případném výskytu poruchy v systému, která je následně lokalizována a izolována, aby neovlivňovala funkci ostatních částí systému. Nakonec je zjištěná porucha opravena a napadená část je schopna opět správně plnit svou původní funkci.

1.3 Obvody FPGA

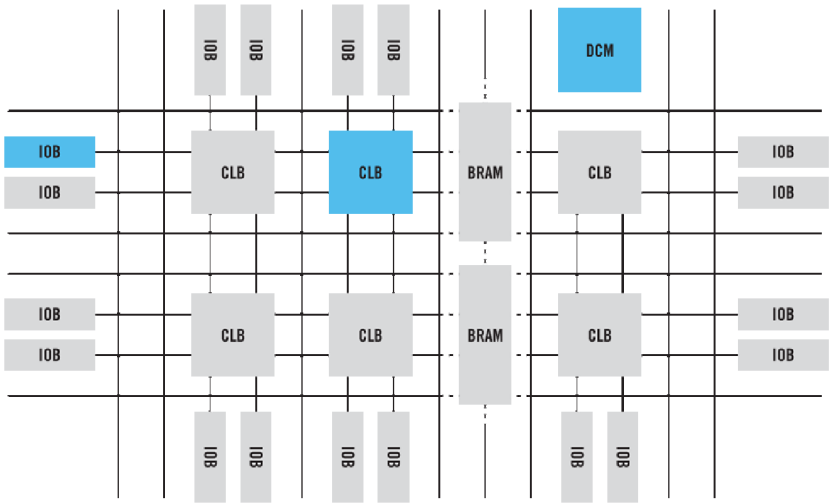
Programovatelná hradlová pole (*FPGA – Field Programmable Gate Array*) jsou obvody, které je možné programovat a tím dosáhnout požadované funkcionality. Obvody FPGA je možné programovat jednak před jejich použitím, ale i rekonfigurovat, což znamená programovat obvod za běhu aplikace. Využít lze i částečnou dynamickou rekonfiguraci, kdy probíhá programování pouze části obvodu, zatímco zbytek obvodu pracuje. Právě programovatelností se liší FPGA od zákaznických integrovaných obvodů (*ASIC – Application Specific Integrated Circuit*), které získají svou funkcionalitu již ve výrobě.

Obvody FPGA jsou stále populárnější a nacházejí uplatnění v řadě aplikací, především díky uváděné programovatelnosti, snadnému návrhu, flexibilitě, snižující se spotřebě a také klesajícím cenám. Použití nalézají především tam, kde je třeba vyrábět malé série a nevyplatí se návrh ASIC a konvenční řešení s mikroprocesorem je nevhodné. S výhodou lze FPGA použít pro prototypování složitějších zákaznických obvodů, což umožní testovaní systému již během návrhu. Programovatelnost lze využít i pro změnu chování obvodu u zákazníka, což umožňuje opravovat chyby v návrhu nebo přidávat nové funkce do již používaného zařízení.

1.3.1 Struktura obvodu FPGA

Strukturu FPGA zachycuje obrázek 1.1. Obvod se skládá z matice konfigurovatelných logických bloků (*CLB – Configurable Logic Block*), které jsou propojeny pomocí programovatelné propojovací sítě. Mimo CLB obsahují moderní obvody i řadu dalších prvků, jako je například bloková RAM paměť (BRAM), rychlé násobičky, procesorová jádra a bloky specializovaných procesorů pro zpracování digitálních signálů (*DSP – Digital Signal Processor*). Pro komunikaci s okolím slouží vstupně výstupní bloky (*IOB – Input/Output Block*).

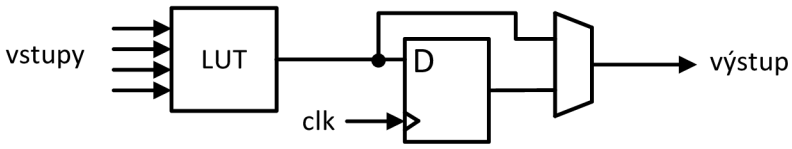
Konfigurace jednotlivých bloků a propojovací sítě obvodu je uložena v paměti SRAM (*Static Random Access Memory*) ve formě tzv. *bitsteramu* (bitová posloupnost obsahující konfiguraci obvodu FPGA), což usnadňuje programování a rekonfiguraci. Na druhou stranu je ale nutné obvod rekonfigurovat při každém spuštění, jelikož uložená konfigurace je volatilní, což vychází z principu SRAM. Existují i FPGA obvody s nevolatilní konfigurací, které mají nižší spotřebu, ale přináší problémy při rekonfiguraci. V současné době jsou nejpoužívanější FPGA obvody s konfigurací uloženou v SRAM.



Obrázek 1.1: Struktura obvodu FPGA [4].

CLB bloky

Strukturu konfigurovatelného bloku, který umožňuje implementaci libovolné logické funkce, znázorňuje obrázek 1.2. Jedná se o hlavní prostředek pro implementaci sekvenčních a kombinačních logických obvodů. Každý CLB blok je složen z SRAM paměťové buňky (*LUT – Look-Up Table*), která realizuje logické funkce pomocí vyhledávací tabulky. CLB bloky obsahují také klopné obvody typu D a multiplexory.



Obrázek 1.2: Struktura CLB bloku.

1.3.2 Poruchy v FPGA

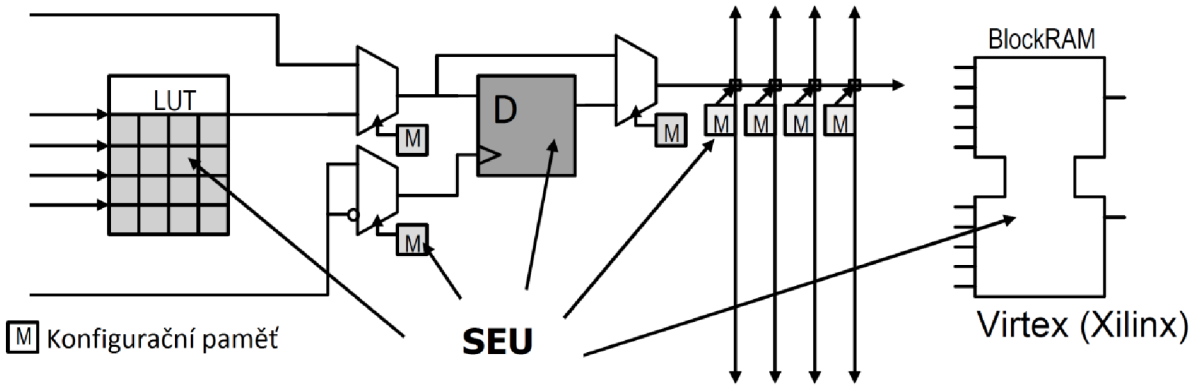
V obvodech FPGA se setkáváme s poruchami, které typicky vznikají vlivem vysoce energetických částic, které se vyskytují nejen ve vesmíru, ale i ve vrchních částech atmosféry Země. Vlivem slunečního záření vznikají různé druhy částic, rozlišujeme částice s nábojem, kterými mohou být protony, elektrony nebo ionty a částice vzniklé elektromagnetickou radiací, které nazýváme fotony. Koncentrace těchto částic roste s nadmořskou výškou. Při návrhu systémů, u kterých je vyžadována vysoká spolehlivost, je musíme brát v úvahu i v nižších nadmořských výškách.

Při zásahu paměťové buňky částicí může dojít ke snížení napětí paměťové buňky, což vede ke změně uložené hodnoty. Tento efekt se nazývá *Single Even Upset* (SEU) a je to nejčastější porucha

postihující FPGA obvody. Veškeré konfigurační informace o logice FPGA obvodu jsou uloženy v SRAM paměti a výskyt SEU v této paměti může mít různé projevy ve fungování obvodu. K opravě SEU v FPGA lze s výhodou použít rekonfiguraci, která zajistí původní funkcionalitu obvodu pomocí obnovení původní konfigurace.

Obrázek 1.3 znázorňuje možné projevy SEU v FPGA. Výskyt chyby v LUT tabulce obvykle vede ke změně logické funkce, kterou LUT tabulka implementuje. SEU se může také vyskytnout v klopném obvodu typu D v CLB, což vede k poruše v sekvenční logice. Porucha v části konfigurační paměti, která nese informaci o konfiguraci propojovací sítě, může vést k chybnému propojení jednotlivých bloků. Bloková paměť BRAM je také citlivá na výskyt poruchy, která zde může způsobit změnu uložených dat.

Druhým typem poruchy, o kterém se hovoří v souvislosti s FPGA obvody, je *Single Event Transient* (SET), která nastává v případě zásahu kombinační logiky a projevuje se jako krátký zákmit.



Obrázek 1.3: Ukázka možných projevů SEU v obvodech FPGA [2].

2 Techniky zajištění odolnosti systémů založených na FPGA proti poruchám

Při zajištění odolnosti proti poruchám u systémů založených na FPGA lze využít jeden ze dvou základních přístupů. Prvním přístupem je vytvoření nové architektury FPGA, která bude složena z komponent odolných proti poruchám. Lze vycházet z již existujících FPGA obvodů a nahradit jednotlivé komponenty jejich zabezpečenými ekvivalenty nebo vytvořit zcela novou architekturu obvodu FPGA. Při tvorbě FPGA odolného proti poruchám se využívají speciální materiály a nejnovější technologie, příkladem takového FPGA je Stratix V od firmy Altera. Jedná se o poměrně drahý a časově náročný přístup, který je využíván především pro armádní nebo kosmické účely. V komerčních aplikacích není tento přístup příliš využíván, protože je zde často vyžadována krátká doba od vývoje do prodeje (*time to market*) a nízké náklady.

Druhým přístupem je využití existujících FPGA obvodů a zabezpečení proti poruchám na úrovni návrhu. Jedná se o přístup, který je v současné době používaný v komerčních aplikacích a stále se vyvíjí nové metody pro návrh takto zabezpečených systémů. Tento způsob zabezpečení je založen na redundanci, přičemž rozlišujeme hardwarovou, časovou a informační redundanci. Nejčastěji využívána je redundance hardwarová, existují i metodiky, které využívají kombinaci více typů redundancí.

Redundance je sama o sobě schopná zajistit odolnost proti poruchám pomocí maskování pouze při určitém počtu poruch, je tedy třeba detekovat moduly napadené chybou a zajistit jejich opravu. K opravě komponent napadených poruchou lze s výhodou použít dynamickou rekonfiguraci.

Tato kapitola se věnuje jednotlivým typům redundance a jejich možnostem, výhodám a nevýhodám. Následuje úvod do metod pro detekci a lokalizaci poruch, v poslední části této kapitoly je představena možnost využití dynamické rekonfigurace.

2.1 Informační redundance

Informační redundance slouží k ochraně dat před poruchami, nejčastější formou informační redundance je kódování. Používají se nejrůznější kódy, které jsou schopny detekovat a případně i opravit chybu v datech. Data zakódovaná pomocí těchto kódů jsou obvykle rozšířena o kontrolní bity, proto označení informační redundance. Do kategorie informační redundance lze zařadit také zálohování dat, příkladem je záloha pevných disků pomocí techniky RAID (*Redundant Array of Independent Disk*) zapojení.

K rozlišení, zda je kód schopen detekce nebo opravy chyby, používáme tzv. Hammingovu vzdálenost, což je počet bitů, ve kterých se liší zakódovaná slova. Například kód, jehož Hammingova vzdálenost je rovna 2, je schopen detekovat jednu chybu. Kód s Hammingovou vzdáleností 3 je schopen detekovat dvě chyby, nebo jednu chybu opravit.

2.1.1 Parita

Parita je velmi jednoduchý způsob detekce chyb v binárním slově. Jedná se o jeden kontrolní bit, který je přidán k užitečným datům. Rozlišujeme mezi sudou a lichou paritou. Sudá parita znamená, že

ve slově je sudý počet logických jedniček, u liché parity se jedná o lichý počet jedniček. Parita umožňuje detekci lichého počtu chyb a neumožňuje opravu dat, protože neobsahuje informaci o místě chyby.

2.1.2 Kontrolní součet

Kontrolní součet je primárně určen pro detekci chyb v datech přenášených prostřednictvím komunikačních kanálů. Před odesláním je nad daty proveden výpočet kontrolního součtu a výsledná hodnota je přenášena společně s užitečnými daty. Po přijetí dat je opět proveden výpočet kontrolního součtu a hodnota je porovnána s přijatým kontrolním součtem. V případě, že se tyto dvě hodnoty liší, je detekována chyba.

2.1.3 Hammingův kód

Tento kód je pojmenován po svém objeviteli Richardu Hammingovi. Jedná se o kód (*n, k*) s Hammingovou vzdáleností 3, kde *n* je celková šířka slova a *k* je šířka užitečných informací. Tento kód je zadán kontrolní maticí, v níž jsou vystřídány všechny možné nenulové (*n-k*)-bitové sloupce a žádný z nich se neopakuje [1].

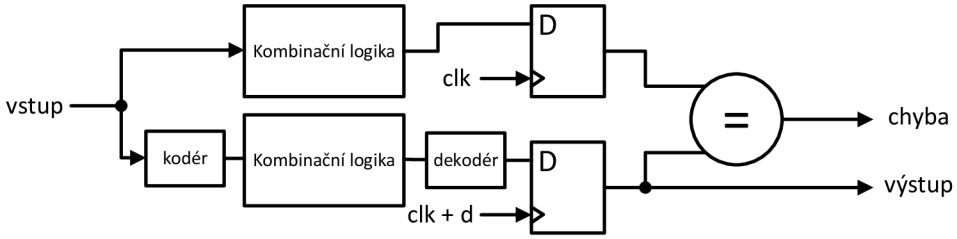
Příkladem může být Hammingův kód (7,4), který ve slově o šířce 7 bitů nese 4 bity užitečných informací. Zbylé 3 bity jsou kontrolní a jsou generovány pomocí rovnic. Ke kontrole slouží kontrolní rovnice, jejichž výsledkem je syndrom chyby. Syndrom chyby je informace o tom, zda jsou data bez chyby, s opravitelnou chybou nebo obsahují vícenásobné neopravitelné chyby. V případě opravitelné chyby je výsledkem kontrolních rovnic i informace o pozici chyby.

Rozšířením každého slova v Hammingovu kódu o sudou paritu získáme tzv. *rozšířený Hammingův kód*. Hammingova vzdálenost je rovna 4, což značí, že tento kód umožňuje opravu jednobitových chyb a zároveň detekci 2-bitových chyb. Tento kód eliminuje nebezpečí nesprávné opravy při výskytu dvojnásobné chyby u obyčejného Hammingova kódu, což je způsobeno shodným syndromem některých jednobitových a dvoubitových chyb.

2.2 Časová redundance

Zabezpečení pomocí časové redundance spočívá v opakování výpočtu stejnou komponentou v různých časových intervalech a porovnání výsledků jednotlivých výpočtů, čímž lze zabezpečit především kombinační logiku proti permanentním poruchám [7]. Opakování výpočtu přináší problém v podobě snížení rychlosti obvodu, jelikož se výpočet musí několikrát opakovat. Snížení rychlosti může představovat problém v aplikacích, kde je vyžadován vysoký výkon. Naopak výhodou tohoto způsobu zabezpečení je pouze mírný nárůst použitého HW, který naroste pouze o klopné obvody pro uložení dočasných výsledků a o obvod pro porovnání jednotlivých mezivýsledků.

Opakováním výpočtu se stejnými operandy lze odhalit především dočasnou poruchu v datech, pro detekci permanentní poruchy je vhodné operandy před opakováním výpočtu nějakým způsobem upravit. Vhodným způsobem úpravy je bitový posun operandů, záměna horní a dolní poloviny operandů nebo jejich bitový doplněk. Obrázek 2.1 znázorňuje dvojité opakování výpočtu včetně úpravy operandů před druhým výpočtem. Úpravu operandů zajišťuje komponenta *kodér*, přičemž komponenta *dekodér* zajišťuje úpravu výsledku do podoby vhodné pro porovnávání a detekci permanentní poruchy.



Obrázek 2.1: Detekce poruch pomocí časové redundance.

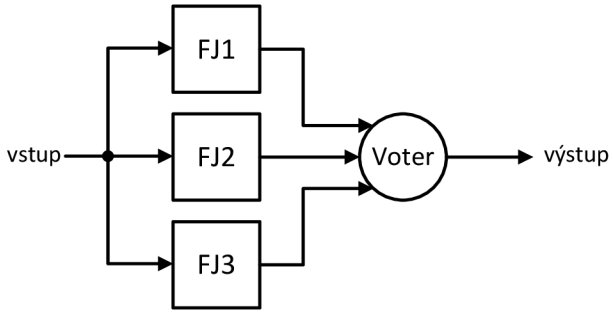
2.3 Hardwarová redundance

Hardwarová redundance je nejčastější způsob pro zabezpečení systému proti poruchám. Jedná se o poměrně jednoduchý způsob, jednotlivé komponenty jsou několikrát duplikovány a doplněny o speciální obvod, který hlídá, zda všechny jednotky poskytují stejný výstup. Hardwarová redundance je vhodnější pro zabezpečování sekvenční logiky [7]. Nevýhodou hardwarové redundance je značný nárůst použitého hardware, ale v případě implementace na FPGA lze takto využít nevyužité volné místo. Příkladem hardwarové redundance jsou systémy NMR a duplexní systémy [5].

2.3.1 NMR a TMR systémy

Triple Modular Redundancy (TMR) spočívá ve vytvoření tří stejných funkčních jednotek, které všechny současně zpracovávají stejná vstupní data. Výstupy jednotlivých jednotek jsou porovnávány pomocí rozhodovací jednotky nazývané majoritní *voter*, jehož výstupem je výsledek, který produkuje většina jednotek, v případě TMR se jedná o 2 jednotky. TMR systém je tedy schopen maskovat pouze jednu poruchu a lokalizovat jednotku, která poskytla chybný výstup, což znamená, že je napadena poruchou. Ukázku zabezpečení pomocí TMR znázorňuje obrázek 2.2.

Systémy označované jako NMR jsou zobecněnou verzí TMR, kdy zabezpečení nezajišťuje ztrojení komponent, ale obecně N duplikací stejné komponenty. Čím větší je počet duplikovaných komponent, tím větší počet poruch je schopen NMR systém maskovat, ale tím větší jsou spotřebované zdroje.

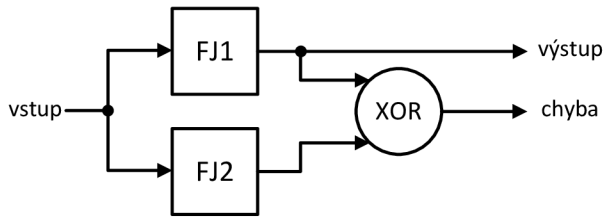


Obrázek 2.2: Příklad zabezpečení pomocí TMR.

2.3.2 Duplexní systémy

Jedná se o metodu zabezpečení proti poruchám pomocí zdvojení zabezpečované komponenty. Výstupy z obou komponent jsou porovnávány pomocí logické operace XOR, která zajišťuje detekci chyby pomocí nonekvivalence. Zabezpečení pomocí zdvojení ukazuje obrázek 2.3, jsou zde zobrazeny dvě funkční jednotky (FU), jejichž výstupy jsou porovnávány a v případě, že se nerovnají,

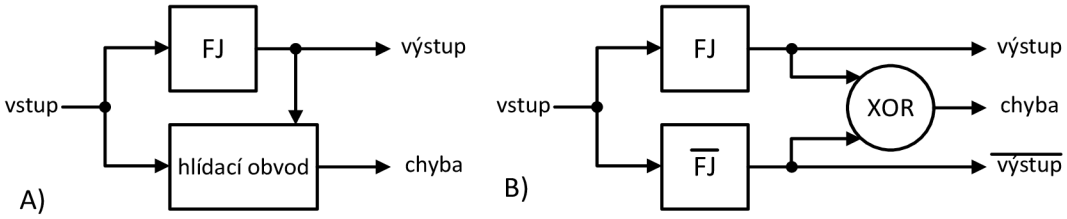
je hlášen výskyt chyby. Duplexní systémy poskytují pouze informaci o tom, zda se v komponentě vyskytla porucha a nejsou schopny tuto poruchu maskovat ani lokalizovat.



Obrázek 2.3: Ukázka duplexního systému.

2.4 Detekce a lokalizace poruch

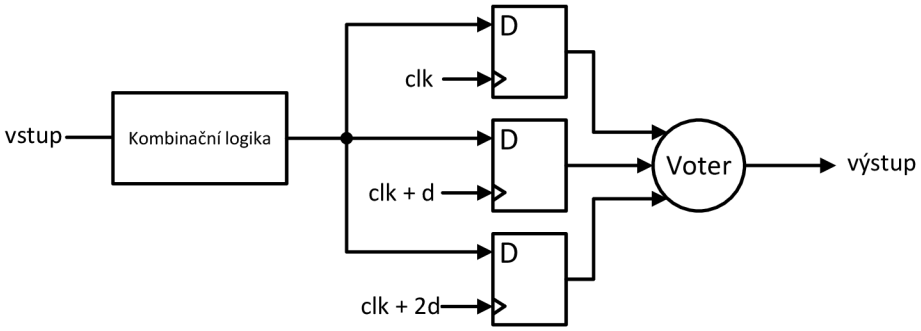
Uvedené metody zabezpečení pomocí redundance jsou schopny především detekovat a maskovat poruchy. Pro možnost opravení komponenty napadené poruchou je nutné poruchu lokalizovat a k tomu slouží techniky označované jako CED (*Concurent Error Detection*), některé z nich ukazuje obrázek 2.4. Tyto techniky zahrnují detekci a lokalizaci chyb pomocí hlídacího obvodu nazývaného *on-line checker*, kontrolou parity, kontrolou inverzní funkcí, pomocí časové redundance a pomocí tzv. dvou-drátové logiky. CED techniky jsou využívány v nejrůznějších metodikách pro zabezpečení proti poruchám, některé z nich jsou uvedeny v následujících odstavcích.



Obrázek 2.4: Vybrané CED techniky: A) hlídací obvod (on-line checker), B) kontrola inverzní funkcí.

2.4.1 Kombinace časové a hardwarové redundance

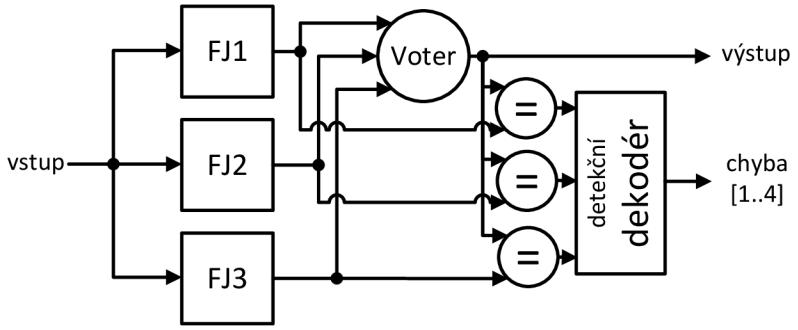
Tato metodika, představena v [2], spočívá v kombinaci duplexního systému a CED techniky založené na časové redundanci a slouží k zabezpečení kombinační logiky. Výhodou je především snížení počtu použitých vstupních a výstupních pinů FPGA obvodu. Při použití velkých kombinačních bloků vede tento způsob k použití menšího prostoru na FPGA. Časovou redundanci ukazuje obrázek 2.5.



Obrázek 2.5: Časová redundance zajišťující detekci poruch v kombinační logice.

2.4.2 TMR s lokalizací poruchy

Výše uvedené zabezpečení pomocí TMR bylo schopné poruchy pouze maskovat. Obrázek 2.6 zobrazuje schéma TMR systému, který je obohacen o obvod pro detekci jednotky napadené poruchou a rozlišení, zda je tato porucha dočasná nebo permanentní. Takto upravený TMR systém umožňuje jednoznačně určit jednotku, kterou je třeba opravit a následně provést opravu například pomocí dynamické rekonfigurace.



Obrázek 2.6: Schéma TMR systému s detekcí a lokalizací poruchy.

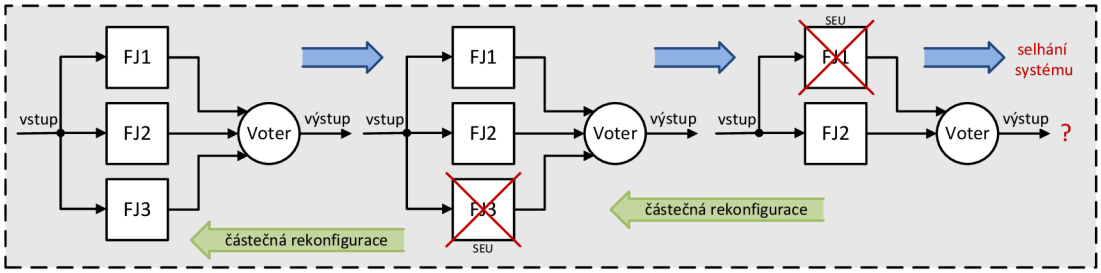
2.5 Částečná dynamická rekonfigurace

Částečná dynamická rekonfigurace (*PDR – Partial Dynamic Reconfiguration*) umožňuje změnu funkcionality části FPGA obvodu bez přerušení provádění operací ostatních částí systému. PDR lze využít pro zvýšení bezpečnosti, snížení použitého místa na FPGA pomocí změny funkcionality v čase dle potřeby a také pro zajištění opravy části FPGA napadené poruchou. Statická rekonfigurace vyžaduje přerušení provádění všech činností, což u dynamické rekonfigurace není vyžadováno a dochází ke změně za běhu systému.

Část, kterou je možné měnit prostřednictvím částečné dynamické rekonfigurace na FPGA je označována jako PRR (*Partial Reconfiguration Region*). Komponenta, která je implementována v jednom PRR se nazývá PRM (*Partially Reconfigurable Module*).

Jak již bylo zmíněno, rekonfiguraci lze použít pro opravu modulů napadených poruchou a za tímto účelem byl vyvinut řadič částečné dynamické rekonfigurace [8], který bude dále představen. Zmíněny budou také metody *bitstream scrubbing* a *readback*, které slouží k opravě SEU v konfiguračním bitové posloupnosti.

Obrázek 2.7 ukazuje příklad použití částečné dynamické rekonfigurace v systému zabezpečeném pomocí TMR. Systém obsahuje tři shodné moduly, které nejprve všechny správně plní svou funkci a poskytují stejné výsledky. Po výskytu poruchy v jednom modulu plní svou funkci již jen dva moduly. Výskyt další poruchy by vedl k selhání celého systému, čemuž lze zabránit obnovením poškozeného modulu pomocí rekonfigurace a uvedením celého systému do původního bezporuchového stavu.



Obrázek 2.7: Příklad použití částečné dynamické rekonfigurace u systému TMR.

2.5.1 Metody čtení a opravy bitové posloupnosti

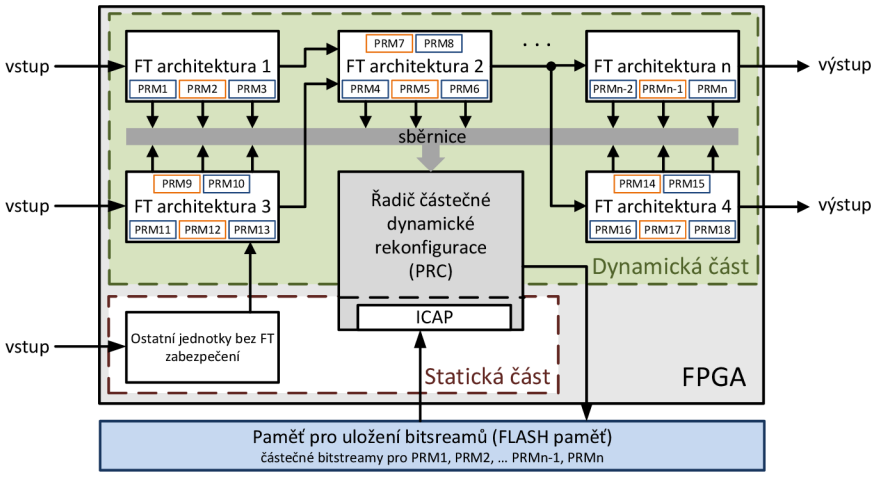
Bitstream scrubbing je metoda, při které dochází k plánovanému přepsání konfigurační paměti v době, kdy obvod není používán, přesněji před použitím obvodu nebo v pravidelně stanovených intervalech. Použitím této metody nedochází k detekci poruch, ale k jejich preventivnímu odstranění.

Bitstream readback je založen na pravidelném čtení bloků konfigurační paměti a kontrole, zda není napadena poruchou. Kontrola může být provedena porovnáním přečtené konfigurační bitové posloupnosti s původním bitstreamem uloženým v externí paměti. Pokud za běhu nedochází ke změně konfigurační bitové posloupnosti, lze si předpočítat a uložit hodnoty CRC a ty následně porovnávat.

Uvedené metody nejsou schopny detekovat chyby, které nebyly způsobeny poruchou v paměti a pro praktické použití je nutné je kombinovat s dalšími technikami, jako je například TMR.

2.5.2 Řadič částečné dynamické rekonfigurace

Řadič částečné dynamické rekonfigurace je prezentován v [8]. Jedná se o řadič, který je schopen zajistit rekonfiguraci požadované jednotky v systému odolném proti poruchám. Uspořádání takového systému zobrazuje obrázek 2.8. Je zobrazeno rozdělení FPGA na statickou část, kde jsou implementovány jednotky bez zabezpečení proti poruchám, a dynamickou část, kde jsou implementovány jednotlivé zabezpečené jednotky, které jsou složeny z jednotlivých modulů PRM. Nejmenším blokem, který může být rekonfigurován je právě modul PRM, jehož identifikace je vstupem pro řadič, který takto identifikovaný modul rekonfiguruje a po opravě zajistí jeho synchronizaci s ostatními jednotkami. Konfigurační bitové posloupnosti, které řadič používá pro rekonfiguraci, jsou uloženy v externí paměti.



Obrázek 2.8: Struktura systému odolného proti poruchám využívající PDR.

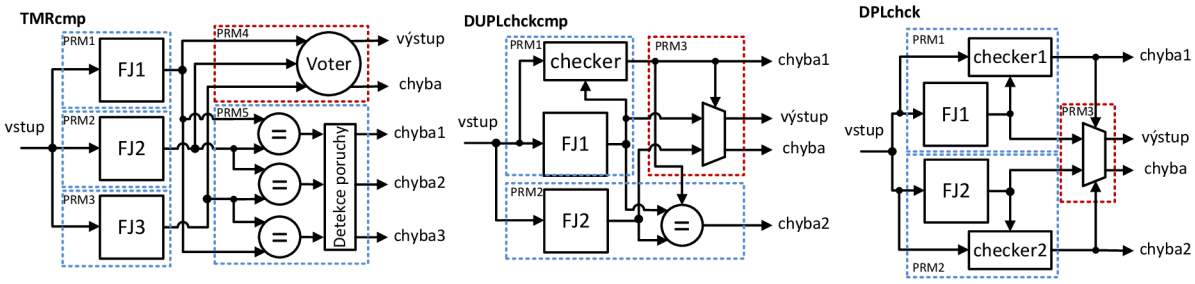
Řízení rekonfigurace probíhá podle indexu jednotky, která je napadena poruchou. Řadič vyčte z externí paměti konfigurační bitovou posloupnost a přes interní rekonfigurační rozhraní (ICAP) změní konfiguraci indexované jednotky. Řadič rekonfigurace je schopen detekovat permanentní poruchu, tuto detekci provádí na základě první provedené rekonfigurace, pokud i po provedení rekonfigurace jednotka stále produkuje nekorektní výsledky, porucha je prohlášena za trvalou.

Po rekonfiguraci jednotky je třeba jednotku synchronizovat s ostatními jednotkami. Pokud by byla jednotka uvedena do provozu bezprostředně po rekonfiguraci, pravděpodobně by byla opět označena jako jednotka napadená poruchou. Řadič rekonfigurace zajišťuje synchronizaci tak, že po rekonfiguraci čeká určitou dobu, po kterou nereaguje na hlášení o poruchách na svém vstupu a neprovádí rekonfiguraci. Autoři uvádí, že tato doba by měla být minimálně tak velká, jako je nejdelší doba mezi lokálními resety spravovaných jednotek.

Uváděný řadič částečné dynamické rekonfigurace lze přizpůsobit počtu připojených jednotek, je možné nastavit dobu čekání a šířku adresové sběrnice externí paměti. Uvedené nastavení je možné díky generické implementaci řadiče.

2.5.3 Metodiky založené na částečné dynamické rekonfiguraci

Autoři v [8] uvádí možnosti rozšíření TMR architektury a duplexní architektury o on-line hlídací obvody, které zajišťují detekci a především lokalizaci chyby. Základní idea je taková, že každá jednotka je implementována v jednom PRM modulu včetně hlídacího obvodu, který ji kontroluje. Tento způsob umožňuje detekci, ve kterém modulu PRM nastala chyba a tento modul následně opravit pomocí řadiče částečné dynamické rekonfigurace, aniž by došlo k přerušení běhu ostatních modulů.



Obrázek 2.9: FT architektury založené na PRM a částečné dynamické rekonfiguraci.

Obrázek 2.9 zobrazuje tři navržené architektury pro zabezpečení proti poruchám založené na modulech PRM a částečné dynamické rekonfiguraci. První architektura (TMRcmp) je již představené TMR, kde každá jednotka je implementována v samostatném modulu PRM, další PRM obsahuje rozhodovací obvod a poslední PRM obsahuje detekci a lokalizaci poruchy.

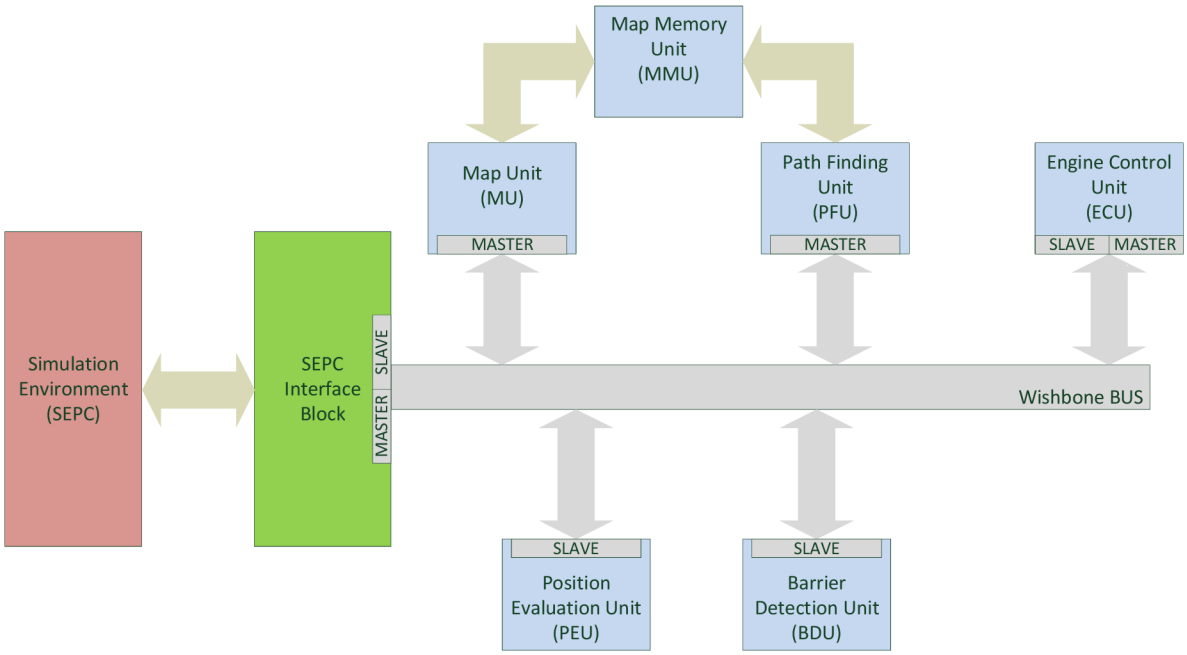
Druhá a třetí zobrazená architektura jsou duplexní systémy, které již byly obecně popsány v předchozím textu. V prvním případě (DUPLchckcmp) se jedná o duplexní architekturu rozšířenou o checker a komparátor, zatímco v posledním případě (DUPLchck) je uvedena duplexní architektura, kde obě jednotky jsou vybaveny vlastním hlídacím obvodem.

3 Návrh řídicí jednotky

Jádrem této práce je návrh a implementace řídicí jednotky robota určeného pro pohyb v bludišti. Navržená řídicí jednotka je exemplární systém, který musí obsahovat různé aspekty návrhu číslicových systémů (např. sekvenční a kombinační obvody, sběrnici, různé druhy pamětí, atd.), na kterých bude možné demonstrovat a testovat různé metodiky pro zabezpečení systému proti poruchám. Tato kapitola se věnuje návrhu řídicí jednotky na vyšší úrovni abstrakce a popisu hlavních vlastností a rysů jednotlivých bloků.

3.1 Blokové schéma řídicí jednotky

Obrázek 3.1 zobrazuje blokové schéma řídicí jednotky. V Příloze A je uvedeno podrobnější blokové schéma včetně zobrazení datových toků mezi jednotlivými funkčními bloky. Celá řídicí jednotka bude připojena k počítači, na kterém běží simulačnímu prostředí (*SEPC – Simulation Enviroment PC*) přes rozhraní SEPC (*SEPC Interface Block*), což je rozhraní celé řídicí jednotky, jehož prostřednictvím jsou získávána data a předávány pokyny k pohybu robota. Jednotlivým funkčním blokům řídicí jednotky jsou věnovány následující podkapitoly.

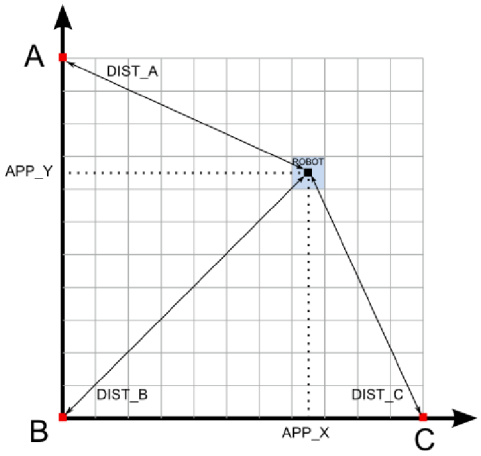


Obrázek 3.1: Blokové schéma řídicí jednotky robota.

3.1.1 Blok pro vyhodnocení polohy

Blok pro vyhodnocení polohy (*PEU – Position Evaluation Unit*) získává vzdálenost od kontrolních bodů, které jsou rozmístěny na určitých fixních pozicích v mapě. Z nich vypočítá pozici robota v mapě, kterou poskytuje dalším jednotkám v podobě souřadnic x a y. Situaci zobrazuje obrázek 3.2, kde lze vidět pozici robota a tři kontrolní body A, B a C. Zobrazené vzdálenosti od kontrolních bodů (DIST_A, DIST_B, DIST_C) jsou vstupní hodnoty pro výpočet pozice robota (APP_X, APP_Y).

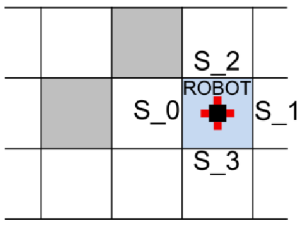
Pozici robota v mapě lze vypočítat pomocí Pythagorovy věty, z čehož lze odvodit, že náplní práce tohoto bloku jsou především aritmetické výpočty, jako je sčítání, násobení a dělení. Zejména iterační výpočty, například dělení, mají často komplikované chování, které je velmi zajímavé z hlediska zabezpečení odolnosti proti poruchám.



Obrázek 3.2: Pozice robota v mapě.

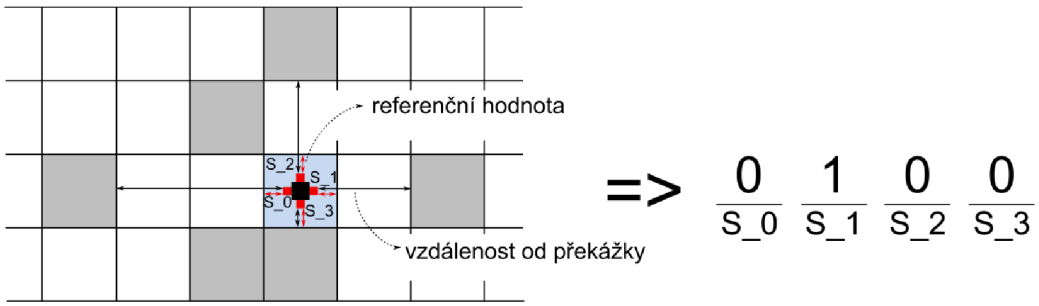
3.1.2 Blok pro vyhodnocení překážek

Pro vyhodnocení překážek v bloku BDU (*Barrier Detection Unit*) jsou k dispozici čtyři senzory, každý je umístěn na jedné straně robota (můžeme si jej představit jako čtyřboký hranol) a poskytuje informaci o vzdálenosti nejbližší překážky. Umístění senzorů znázorňuje obrázek 3.3. Výstupem vyhodnocení překážek je vektor o délce čtyř bitů, který reprezentuje čtyř-okolí pozice robota a informuje o výskytu překážek v tomto okolí.



Obrázek 3.3: Osazení senzorů pro detekci překážek.

Činnost této jednotky, kterou demonstruje obrázek 3.4, spočívá v porovnání hodnot získaných ze senzorů (vzdálenost od překážky) s referenční hodnotou určující, kdy se překážka nachází na sousední souřadnici a kdy se vyskytuje překážka mimo zmíněné čtyř-okolí. Jedná se o čistě kombinační obvod, který jde poměrně dobře zabezpečit proti poruchám. Oblast zabezpečení kombinačních obvodů proti poruchám je totiž poměrně dobře prozkoumaná.



Obrázek 3.4: Ukázka detekce překážek.

3.1.3 Blok aktualizace mapy

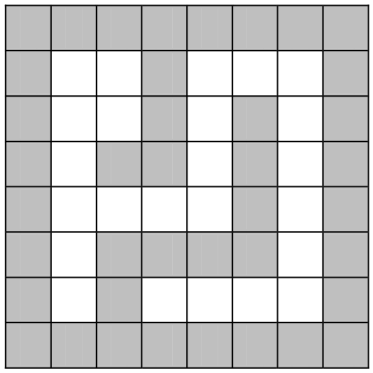
Aktualizace mapy v bloku MU (*Map Unit*) bude probíhat na základě informací o pozici robota získané z bloku PEU a na základě informací o výskytu překážek v čtyř-okolí poskytnutých blokem BDU. Blok aktualizace mapy na základě získaných informací zapíše informaci do paměti pro uložení mapy. Činnost tohoto bloku je nezávislá na řízení ostatních bloků, jakmile dojde k zápisu hodnot do paměti, jednotka získá další data od BDU a PEU a celý algoritmus se bude opakovat.

Činnost této jednotky spočívá v transformaci vstupních informací na adresy pěti paměťových míst a hodnoty, které budou na paměťová místa uloženy. Stejně jako u bloku BDU se jedná o kombinační obvod.

3.1.4 Paměť pro uložení mapy

Paměť MMU (*Map Memory Unit*) je určena pro ukládání informací o aktuální mapě, je uložena v blokové paměti BRAM dostupné na FPGA. Každá souřadnice v mapě bude reprezentována jedním bitem, který informuje o výskytu překážky na této pozici (viz obrázek 3.5). Na rozdíl od ostatních bloků nebude paměť připojena ke sběrnici, ale přímo k blokům, které potřebují s pamětí komunikovat, tedy blok pro aktualizaci mapy a také blok pro hledání cesty v bludišti.

Bloková paměť BRAM je náchylná k výskytu poruch a vyžaduje speciální metodiky zabezpečení odolnosti proti poruchám. Z hlediska odolnosti proti poruchám si zvýšenou pozornost zaslouží také propojení paměti s ostatními částmi systému.



1	1	1	1	1	1	1	1
1	0	0	1	0	0	0	1
1	0	0	1	0	1	0	1
1	0	1	1	0	1	0	1
1	0	0	0	0	1	0	1
1	0	1	1	1	1	0	1
1	0	1	0	0	0	0	1
1	1	1	1	1	1	1	1

Obrázek 3.5: Ukázka mapy a příslušné bitové reprezentace v paměti.

3.1.5 Blok pro hledání cesty

Nejdůležitějším blokem, který řídí činnost ostatních bloků v robotickém systému je blok PFU (*Path Finding Unit*) realizující algoritmus pro hledání cesty v bludišti. Jedná se o blok, který na základě informace o aktuální a požadované cílové pozici vyhledá cestu v mapě. K dispozici má přímý přístup do paměti, do které jsou průběžně ukládány informace o překážkách.

Výpočet bude probíhat na základě informací o mapě, které se budou vytvářet průběžně. Výstupem tohoto bloku bude seznam pozic, které musí robot projít v následujícím kroku. Současně s pohybem robota bude probíhat aktualizace mapy příslušným blokem MU, což umožní pokračovat ve výpočtu cesty k cílové pozici a produkovat další seznam souřadnic, které robot musí projít. Pro vyhledávání cesty v bludišti existuje množství algoritmů, v této kapitole jsou zmíněny algoritmy založené na prohledávání stavového prostoru a jednoduchý algoritmus „rukou po stěně“:

1) **Prohledávání stavového prostoru**

Prohledávání stavového prostoru [9] zahrnuje metody pro řešení nejrůznějších úloh, které spadají do oblasti umělé inteligence. Stavový prostor je množina všech možných stavů řešené úlohy, nejčastěji je reprezentován ve formě orientovaného grafu tvořeného uzly a hranami. V případě bludiště uzly grafu reprezentují jednotlivé křižovatky a hrany reprezentují chodby mezi křižovatkami. Prohledávání stavového prostoru pak spočívá v postupném generování tohoto grafu a jeho procházení. Metody prohledávání stavového prostoru můžeme rozdělit do dvou kategorií a to na neinformované a informované metody.

a) **Neinformované metody**

Nemají k dispozici žádné informace o stavovém prostoru a nemají ani žádné prostředky jak jednotlivé stavy ohodnotit. Jejich práce spočívá v systematickém procházení všech uzlů stavového prostoru. Jako příklad takové situace si lze představit hledání určitého místa na mapě, přičemž nevíme, v jakém směru od výchozího místa toto cílové místo leží. Existuje celá řada metod, které se liší způsobem prohledávání stavového prostoru, například metoda prohledávání do šířky, do hloubky nebo metoda zpětného navrácení.

b) **Informované metody**

Informované metody mají k dispozici informace o stavovém prostoru, které využívají při jeho prohledávání a také mají k dispozici prostředky pro hodnocení jednotlivých uzlů. Příkladem může být hledání cílového místa na mapě, kdy máme k dispozici informaci, ve kterém směru od výchozího místa cíl leží a tím se nám zmenší prohledávaná oblast. Opět existuje celá řada metod, příkladem je známá metoda A* využívající heuristickou funkci pro výběr uzlu, kterým se při prohledávání má pokračovat. Příkladem může být také metoda Greedy search.

2) **Hledání cesty „rukou po stěně“**

Velmi dávné pravidlo [10], jak nezabloudit v bludišti, je držet se rukou (stále stejnou) stěny při procházení bludiště. Toto pravidlo zaručí, že v bludišti nezabloudíme, buď najdeme cestu do cíle, nebo se vrátíme zpět do počáteční pozice. Bohužel tento postup není použitelný ve všech bludištích, selhat může v bludištích, kde je šířka chodby větší než dvě políčka. Pokud by se v bludišti objevilo širší místo a cílová pozice by nebyla umístěna těsně u stěny, nastala by situace, kdy bychom cílovou pozici nikdy neobjevili.

Jedná se o poměrně složitý blok realizující sekvenční algoritmus, který je zajímavý z hlediska zabezpečení proti poruchám. Při realizaci algoritmu hledání cesty bude využívána poměrně rozsáhlá komunikace, jak přes sběrnici, tak s pamětí, což má také svá specifika při zabezpečení odolnosti proti poruchám. Zajímavá je také úvaha o realizaci algoritmu hledání cesty v procesoru implementovaném na FPGA, například PicoBlaze, který je poskytován firmou Xilinx, což by s sebou neslo další možnosti zabezpečení proti poruchám.

U této jednotky je výhodné klást důraz na kvalitní rozhraní, které umožní snadné nahrazení implementace algoritmu jinou implementací, například již uvedenou realizací s procesorem PicoBlaze nebo MicroBlaze. Nabízí se tak možnost porovnávání možností zabezpečení různých implementací.

3.1.6 Blok pro ovládání robota

Pohyb robota je ovládán nastavením rychlosti v požadovaném směru pohybu prostřednictvím bloku ECU (*Engine Control Unit*). Vzdálenost, kterou robot urazí, určuje čas, po který je robot v pohybu.

Po nastavení rychlosti dojde k uvedení robota do pohybu, k zastavení dojde nastavením rychlosti na nulu. Vstupem tohoto bloku je seznam polí, které musí robot projít. Na základě tohoto seznamu je průběžně nastavována rychlost a směr pohybu robota.

Odolnost tohoto bloku proti poruchám je velmi důležitá, protože se jedná o blok, který robota přímo řídí a při napadení poruchou by mohlo dojít k nesprávnému řízení a tím i ke kolizi robota, což by v případě reálného robota v reálném světě mohlo způsobit značné škody.

3.1.7 Sběrnice

Centrálním prvkem celé řídicí jednotky je sběrnice, prostřednictvím které budou komunikovat jednotlivé bloky. Použití sběrnice pro komunikaci usnadní budoucí možné rozšíření o další jednotky pro podporu hledání cesty v bludišti.

Obrázek 3.1 ukazuje, mimo jiné, jaký vztah mají jednotlivé bloky ke sběrnici, tedy zda mají funkci *master*, *slave* nebo *master i slave*. Bloky pro vyhodnocování pozice (PEU) a překážek (BDU) jsou vybaveny pouze rozhraním slave, jelikož jejich úkolem bude poskytovat informace na základě požadavků od jednotek pro aktualizace mapy (MU) a hledání cesty v mapě (PFU). Aktuální data ze senzorů získají od bloku rozhraní SEPC, které je vybaveno jak rozhraním master, tak slave. Dalším blokem, který je vybaven rozhraním master i slave, je blok pro ovládání robota (ECU). Tento blok je vybaven rozhraním master za účelem ovládání robota prostřednictvím rozhraní SEPC, pro získání seznamu kroků od jednotky pro hledání cesty (PFU) je vybaven rozhraním slave. Bloky pro aktualizaci mapy a hledání cesty jsou vybaveny pouze rozhraním master, prostřednictvím kterého si žádají o data. V případě bloku pro aktualizaci mapy (MU) jde o pozici a vektor překážek, blok pro hledání cesty (PFU) vyžaduje ke své činnosti pouze informaci o poloze a prostřednictvím rozhraní master předává seznam hodnot, který musí robot projít, bloku pro ovládání robota (ECU).

Z pohledu zabezpečení odolnosti proti poruchám je sběrnice velmi kritické místo, jelikož leží v centru celého systému a při jejím selhání dojde k narušení korektnosti veškeré komunikace probíhající přes sběrnici. Sběrnice využívá velké množství propojovací sítě v FPGA, přičemž použité zdroje propojovací sítě určuje program PAR (*Place And Route*) a lze je těžko ovlivnit. Využití sběrnice v řídicí jednotce robota je tedy výhodné nejen z pohledu budoucí rozšiřitelnosti, ale také možnosti testování vhodné metodiky pro zabezpečení odolnosti sběrnice proti poruchám.

4 Implementace řídicí jednotky

V předcházející kapitole byl představen blokový návrh řídicí jednotky robota, bylo uvedeno rozdělení celého systému na jednotlivé základní bloky, ze kterých je systém složen. Tato kapitola je věnována detailnímu rozboru jednotlivých bloků z pohledu implementace, jsou zde podrobněji probrány použité algoritmy, případně jejich alternativy, a použité techniky.

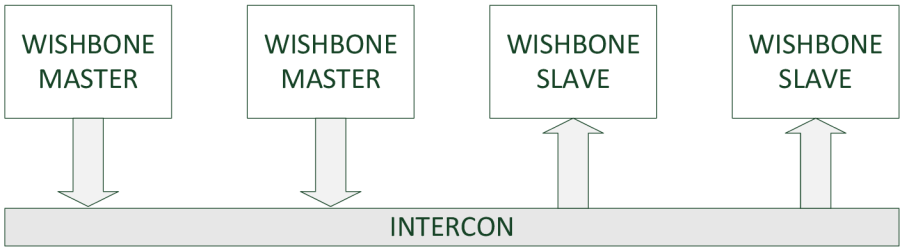
4.1 Sběrnice Wishbone

Sběrnice je klíčovým blokem celé řídicí jednotky, proto byl kladen důraz na pečlivý výběr použité sběrnice. Nabízelo se použití sběrnice určené pro SoC (*System on Chip*), výběr byl zúžen pouze na sběrnici AXI [11] od firmy Xillinx a open-source sběrnici komunity OpenCores: Wishbone [12]. Nakonec byla zvolena sběrnice Wishbone, která zaujala svou jednoduchou volně dostupnou implementací a především volně dostupnou přehlednou dokumentací.

4.1.1 Architektura sběrnice Wishbone

Specifikace sběrnice Wishbone umožňuje několik architektur (*Point to Point, Shared Bus, Crossbar Switch* a *Data Flow Interconnection*), ze kterých byla zvolena sdílená sběrnice (*Shared Bus Interconnections*), kterou znázorňuje obrázek 4.1. Jedná se o architekturu, v které všechna připojená zařízení sdílejí jednu sběrnici, což umožňuje v jednom časovém okamžiku komunikaci pouze jednoho master zařízení, ostatní master zařízení musí čekat, až na ně přijde řada.

Určováním, které zařízení bude moci v daný čas využít sběrnici, se zabývá arbitr sběrnice. Ve zvolené implementaci Wishbone sběrnice je každému master zařízení přidělena priorita. Priorita udává počet sběrnicových cyklů, které budou danému zařízení přiděleny v rámci jedné periody. Perioda je dána součtem všech sběrnicových cyklů, které jsou požadovány jednotlivými zařízeními.

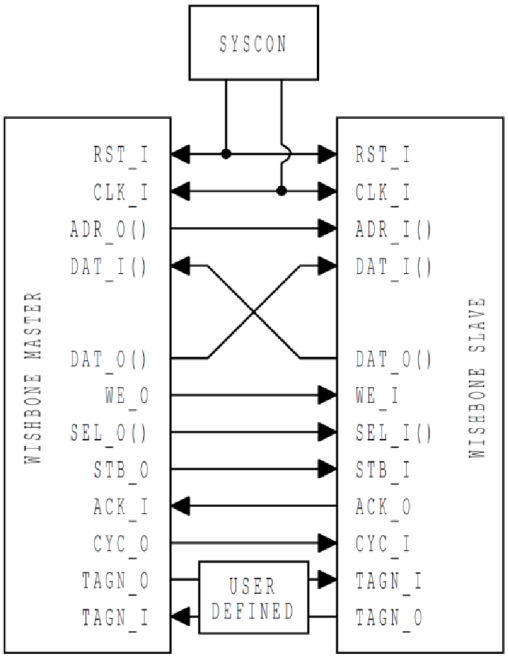


Obrázek 4.1: Schéma architektury sdílené sběrnice Wishbone.

4.1.2 Signály sběrnice a sběrnicové cykly

Obrázek 4.2 a Příloha B ukazuje signály, které jsou přenášeny přes sběrnici, mimo adresy a přenášených dat jsou zde i řídicí signály. Všechny signály můžeme rozdělit do tří kategorií, první jsou signály společné pro master i slave rozhraní, druhou kategorií jsou signály pouze master rozhraní a třetí kategorií jsou signály pouze slave rozhraní.

Příloha A také popisuje časování jednotlivých signálů během přenosových cyklů. Sběrnice Wishbone umožňuje čtecí i zápisové sběrnicové cykly. Přenosové cykly mohou být jak jednoduché, tak blokové, které umožňují přenést více dat v rámci jednoho cyklu.

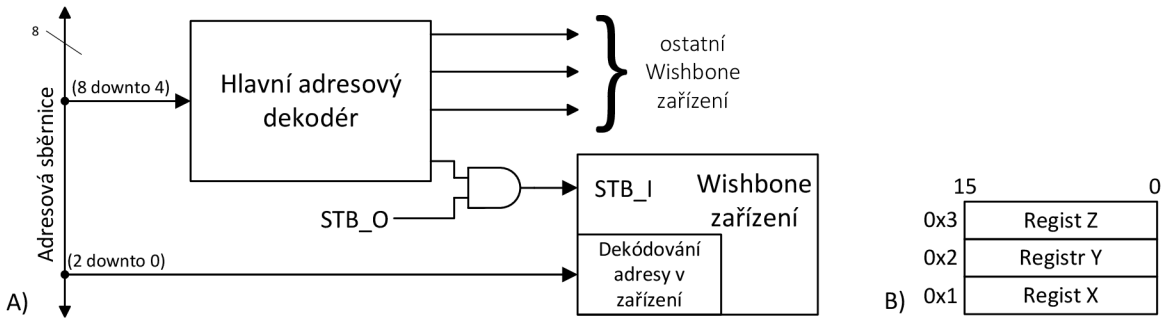


Obrázek 4.2: Přehled signálů přenášených přes sběrnici.

4.1.3 Částečné a úplné dekódování adresy

Wishbone sběrnice dle specifikace umožňuje jak úplné, tak částečné dekódování adresy vystavené na sběrnici. Úplným dekódováním adresy se rozumí, že každé slave zařízení dostane na svůj vstup celou adresu, na základě které zjistí, s kterým registrem tohoto zařízení má zájem pracovat jiné master zařízení na sběrnici. V případě, že zařízení obsahuje například 3 registry, postačí 2bity pro jejich adresaci. Na adresový vstup je ale přivedena celá šířka adresy, v našem případě 8bitů, které musí zařízení dekódovat.

Částečné dekódování adresy, které je použito v implementaci řídicí jednotky robota, využívá hlavní adresový dekodér. Ten na základě části adresy rozhodne, které slave zařízení je adresou vybráno a nastaví příslušný signál pro výběr slave zařízení. K slave zařízení je připojena druhá část adresy, která slouží k adresování jednotlivých registrů zařízení. Výhodou tohoto přístupu je, že na vstup slave zařízení je přivedeno pouze tolik adresových bitů, které vystačí pro adresování všech jeho registrů, což vede k menším nárokům na adresový dekodér jednotlivých slave zařízení. Částečné dekódování adresy přehledně ukazuje obrázek 4.3.



Obrázek 4.3: Částečné dekódování adresy: A) ukázka zapojení, B) adresový prostor zařízení.

4.1.4 Generátor sběrnice

Sběrnice Wishbone není konkrétní implementace sběrnice (*IP core*), kterou by bylo možné ihned použít ve vlastním systému, ale jedná se pouze o standard, podle kterého může být sběrnice implementována. Neznamená to však, že nejsou k dispozici konkrétní implementace podle této specifikace. Jen na internetových stránkách komunity OpenCores je k dispozici několik takových implementací, ze kterých jsem jednu vybral pro použití ve své práci. Jedná se o skript Wishbone Builder [13], který odpovídá požadavkům na implementaci v jazyce VHDL.

Wishbone Builder je skript napsaný v jazyce Perl, který na základě definovaných požadavků vygeneruje implementaci sběrnice ve VHDL. Požadavky na parametry sběrnice a rozhraní jednotlivých Wishbone zařízení jsou definovány v souboru `wishbone.defines`, jež ukazuje obrázek 4.4. V tomto souboru jsou nejprve definovány požadavky na sběrnici, tedy na blok `interconnection` propojující jednotlivá zařízení. Zde se volí šířka datové a adresové části sběrnice, typ sběrnice (v našem případě sdílená sběrnice) a další parametry, které jsou podrobně uvedeny v dokumentaci [13]. Dále jsou zde definovány požadavky na jednotlivá rozhraní slave a master zařízení, uvádí se především požadavky na přítomnost různých signálů na sběrnici. U master zařízení je dále definována priorita, jejíž význam již byl představen. Slave zařízením v souboru `wishbone.defines` přiřazujeme adresu a rozsah adres vnitřních registrů včetně šířky adresového vstupu. Opět podrobnější informace lze nalézt v přehledné dokumentaci [13].

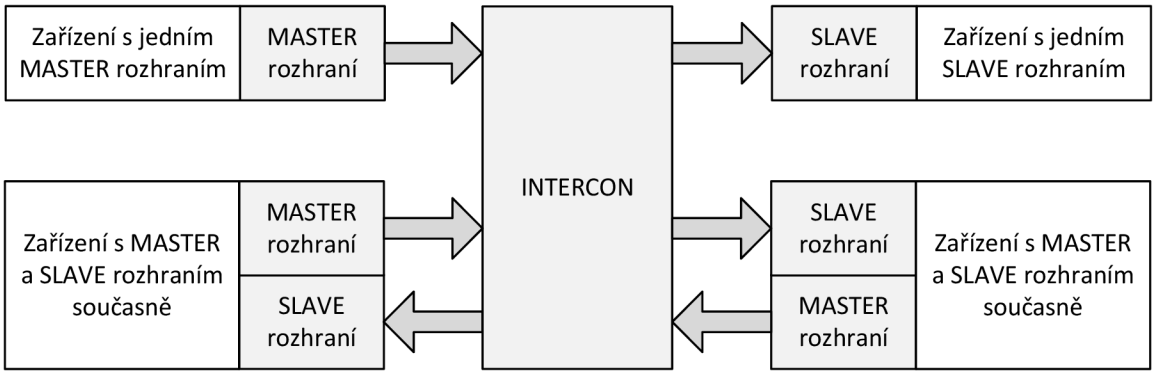
```
filename=wb
intercon=intercon
syscon=syscon
target=generic
hdl=vhdl
dat_size=16
adr_size=8
mux_type=andor
interconnect=sharedbus

master sif
  type=rw
  lock_o=0
  err_i=0
  rty_i=0
  priority=7
end master sif

slave peu
  type=rw
  adr_i_hi=3
  adr_i_lo=0
  lock_i=0
  err_o=0
  rty_o=0
  baseadr=0x10
  size=0x10
end slave peu
```

Obrázek 4.4: Ukázka souboru `wishbone.defines` – parametry sběrnice, master a slave zařízení.

4.1.5 Implementace master a slave rozhraní

Dle specifikace Wishbone sběrnice může být jedno rozhraní pouze typu master nebo slave. Tuto situaci znázorňuje obrázek 4.5, konkrétně dvě horní zařízení. Zařízení s rozhraním typu master i slave současně lze implementovat pomocí použití více rozhraní sběrnice Wishbone u jednoho zařízení [14]. Tuto situaci představuje obrázek 4.5 na dvou zařízeních v dolní části.

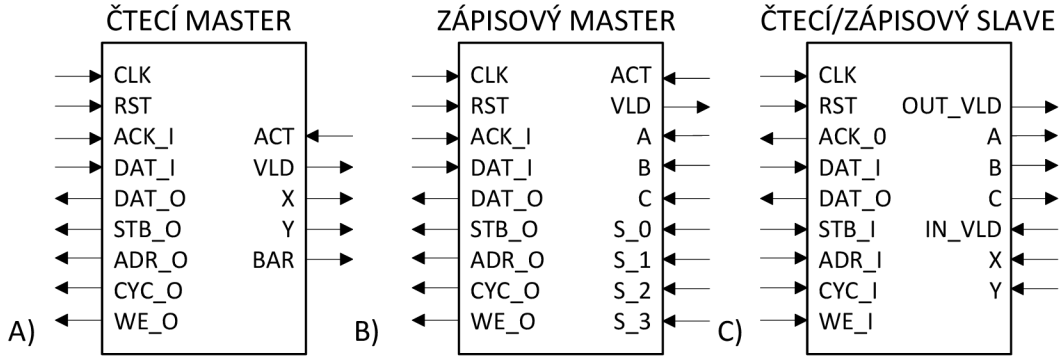
Pro každé zařízení, které musí komunikovat přes sběrnici, byl vytvořen blok rozhraní, který tuto komunikaci obstará. Jedná se o tři typy rozhraní, prvním je čtecí master rozhraní, druhým je zápisové master rozhraní a třetím je čtecí i zápisové slave rozhraní.



Obrázek 4.5: Ukázka zařízení s různými sběrnicovými rozhraními.

Master rozhraní je implementována jako blok, který má na jedné straně signály pro připojení ke sběrnici a na druhé straně vstupní signál ACT, jehož prostřednictvím dojde k zahájení požadované operace (čtení nebo zápis). Ukončení operace indikuje výstupní signál VLD. V případě čtecího rozhraní (obrázek 4.6 A) jsou k dispozici výstupní vektory, které po ukončení operace nesou požadované hodnoty. Naopak zápisové rozhraní (obrázek 4.6 B) obsahuje vstupy pro vektory s informacemi, které je nutné v průběhu operace přenést do slave zařízení.

Každý blok master rozhraní je implementován pomocí konečného automatu, který obstarává nastavování příslušných signálů tak, aby byla vykonána požadovaná operace (viz Příloha B). Adresy zařízení a registrů, se kterými je třeba komunikovat, jsou uloženy ve zdrojovém kódu jako konstanty a zmíněný konečný automat je ve správný časový okamžik vystavuje na sběrnici.



Obrázek 4.6: Ukázka implementovaných rozhraní: A) čtecí master rozhraní, B) zápisové master rozhraní a C) čtecí i zápisové slave rozhraní

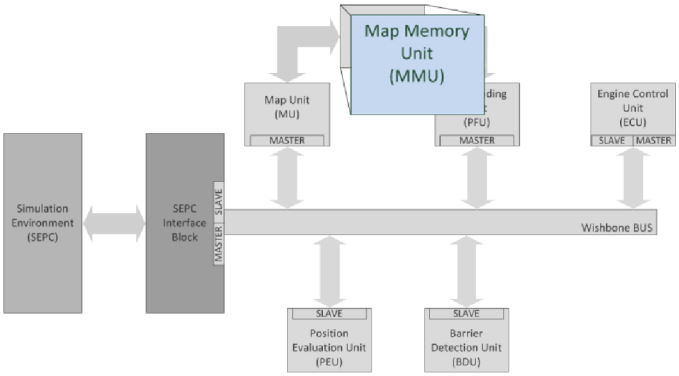
Slave rozhraní (obrázek 4.6 C) musí zajistit, že na každý požadavek master zařízení bude vykonána příslušná reakce, tedy zápis získaných dat do příslušného registru nebo odeslání dat z požadovaného registru. Blok slave rozhraní je na jedné straně osazen, stejně jako blok master rozhraní, signály pro připojení ke sběrnici. Na druhé straně je osazen vstupními a výstupními vektory. Výstupní vektory po provedení operace obsahují data získaná prostřednictvím sběrnice. Na vstupních vektorech jsou očekávány hodnoty, které budou při čtecím cyklu odeslány přes sběrnici. Aby nenastala situace, kdy jiný blok využívající toto rozhraní přečte část výstupních vektorů aktualizovaných při probíhajícím sběrnicovém cyklu a část výstupních vektorů z minulého cyklu, je blok slave rozhraní vybaven signálem OUT_VLD, který je nastaven pouze tehdy, když jsou všechna výstupní data současně platná. Po zahájení sběrnicového cyklu je signál OUT_VLD nastaven do nízké úrovně a po získání všech dat je opět nastaven do vysoké úrovně. Na podobném principu je založen i význam signálu IN_VLD, který informuje blok rozhraní o platnosti všech vstupních dat.

V případě, že je signál IN_VLD nastaven do nízké úrovně a zároveň master zařízení požaduje čtení, jsou na sběrnici vystaveny čekací stavy, dokud nejsou všechna data platná a připravená pro odeslání.

Stejně jako blok rozhraní master zařízení je i blok rozhraní slave zařízení implementován pomocí konečného automatu, který čeká na požadavky od sběrnice, na které odpovídá nastavováním příslušných signálů. Jelikož je využíváno částečné dekódování adresy, na vstup bloku slave rozhraní je přivedena pouze část adresy, která slouží pro adresování jednotlivých registrů. Adresy jednotlivých registrů jsou opět uloženy ve formě konstant a adresový dekodér zajistí vystavení správných dat na sběrnici.

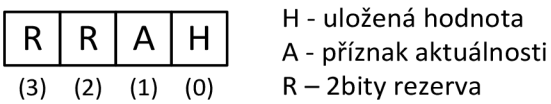
4.2 Paměť pro uložení mapy

Jak bylo uvedeno v kapitole o návrhu, informace o mapě budou ukládány do paměti (obrázek 4.7), která bude realizována pomocí blokové paměti Block RAM na FPGA.



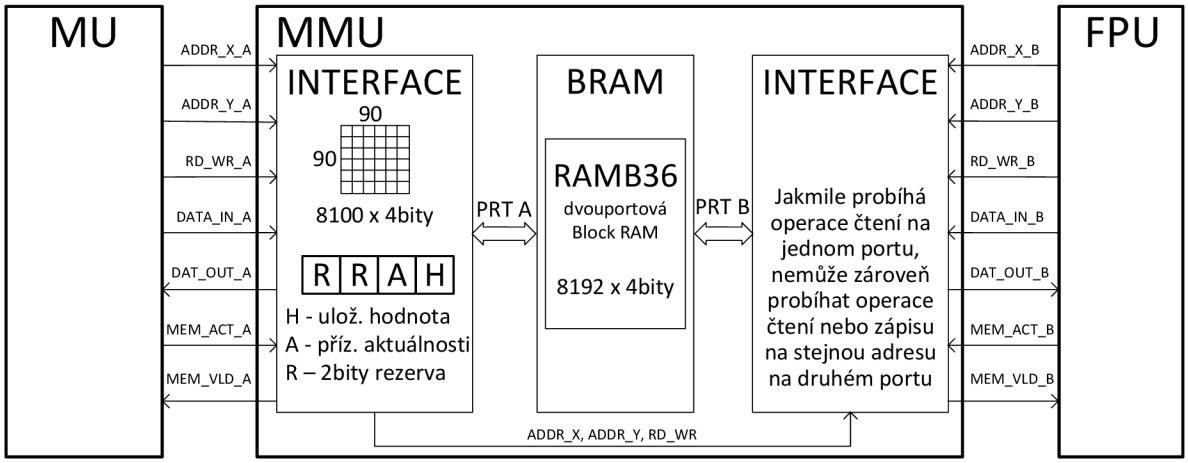
Obrázek 4.7: Paměť pro uložení mapy.

V návrhu se předpokládalo, že každé políčko mapy bude v paměti uloženo na jednom bitu. V průběhu implementace se ukázalo, že bude nutné reprezentovat jedno políčko mapy pomocí více bitů uložených v paměti. Nutné bylo rozšíření o jeden bit, který plní roli příznaku aktuálnosti. Nulová hodnota tohoto bitu informuje o tom, že dané políčko mapy ještě nebylo prozkoumáno. Současně s rozšířením reprezentace políčka v mapě o příznak aktuálnosti byly přidány další dva bity, které jsou zatím nevyužity a tvoří rezervu, například pro potřeby algoritmu hledání cesty v bludišti. Celkovou reprezentaci jednoho políčka mapy v paměti ukazuje obrázek 4.8.



Obrázek 4.8: Ukázka bitové reprezentace jednoho políčka mapy v paměti.

Celý blok pro uložení mapy je složen z několika dílčích bloků. Mimo samotné dvouportové blokové paměti Block RAM jsou zde bloky tvořící rozhraní paměti pro zajištění jednoduché komunikace s ostatními bloky. Dále blok pro synchronizaci jednotlivých portů, který musí zabezpečit, že při zápisu do paměťového místa prostřednictvím jednoho portu nebude probíhat na stejné adrese žádná operace na druhém paměťovém portu. Celou situaci zachycuje obrázek 4.9, na kterém je zobrazeno i rozhraní obou paměťových portů celého bloku MMU a propojení s bloky pro aktualizaci mapy (MU) a hledání cesty v bludišti (PFU). Jednotlivé uvedené dílčí bloky, které jsou v obrázku zobrazeny, budou dále rozebrány podrobněji.



Obrázek 4.9: Vnitřní zapojení bloku pro uložení mapy včetně ukázky propojení s ostatními bloky.

4.2.1 Bloková paměť BlockRAM

Pro realizaci rozsáhlejších pamětí je v FPGA obvodu k dispozici speciální paměťová struktura: bloková paměť Block RAM, jejíž použití v řídicí jednotce robota vyplynulo již z návrhu uvedeného v kapitole 3. Pro realizaci paměti s využitím Block RAM se nabízí několik možností:

- behaviorální popis [15],
- vygenerování IP Core [16],
- knihovní primitivum RAM36 [17].

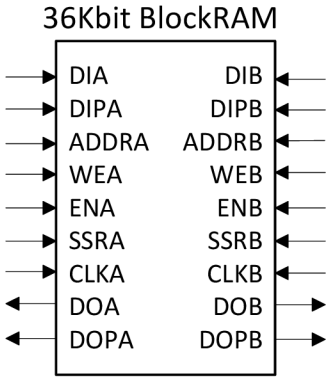
První možností realizace paměti je pomocí behaviorálního popisu, kdy je popsáno chování paměti pomocí jazyka pro popis hardware, v našem případě VHDL. Při syntéze je rozpoznáno, že se jedná o paměť a syntetizér ji realizuje pomocí Block RAM. Výhodou tohoto přístupu je snadná přenositelnost, nevýhodou ovšem je nutnost poměrně přesného behaviorálního popisu paměti, tak aby syntetizér rozpoznal, že se jedná o paměť. Obvykle je nutné použít přesné šablony a i drobné odchylky mohou způsobit použití jiného typu paměti, než jsme předpokládali. Další nevýhodou je, že při behaviorálním popisu paměti nedokáží syntetizéry plně využít možností cílové blokové paměti.

Druhou možností je použít generátor bloků realizujících požadovanou funkci (IP Core), například program *Coregen* od firmy Xilinx. Tento program na základě zadaných požadavků vygeneruje blok, který implementuje paměť s požadovanými parametry. Vygenerovaný blok je vždy závislý na konkrétním typu použitého FPGA, protože používá Block RAM, která je specifická pro dané FPGA. Výhodou tohoto přístupu je, že vygenerovaný blok má vždy stejné vstupní a výstupní rozhraní, tudíž při přechodu na jiné FPGA postačí vygenerovat nový blok pro požadované FPGA. Nevýhodou a zároveň důvodem, proč nebyl použit tento přístup, byla nemožnost nastavit parametry paměti, které by přesně odpovídaly požadavkům.

Využít blokovou paměť lze i pomocí příslušného bloku z knihovních primitiv dostupných pro FPGA Xilinx. V případě blokové paměti pro FPGA Virtex 5 se jedná o knihovní primitivum RAMB36, které ukazuje obrázek 4.10.

Tabulka 4.1 obsahuje vysvětlení významu jednotlivých vstupních a výstupních signálů. Primitivum RAMB36 nabízí 36Kbit dvouportové blokové paměti a lze poměrně snadno nastavit požadované parametry při jeho instanciaci, jedná se o:

- šířku datového vektoru,
- šířku vektoru pro uložení parity,
- použití výstupního registru,
- propojení více bloků pamětí.



Obrázek 4.10: Bloková paměť Block RAM – primitivum RAMB36.

Název	Význam
Dix	Vstupní datový port.
DIPx	Vstupní port pro paritu (nevyužito).
ADDRx	Adresa paměťového místa.
WEx	Čtení/zápis (log. 0 znamená čtení).
ENx	Povolovací vstup, zahájí paměťovou operaci.
SSRx	Reset.
CLKx	Hodinový signál.
DOx	Výstupní datový port.
DOPx	Výstupní port pro paritu (nevyužito).

Tabulka 4.1: Význam signálů knihovního primitiva RAMB36.

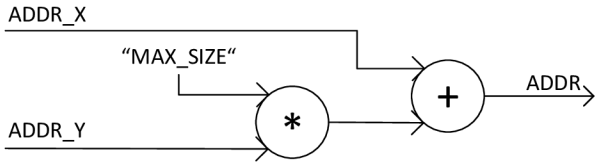
Nevýhodou využití knihovního primitiva je, stejně jako v předchozím případě, nutnost použít odlišná primitiva pro různá FPGA. Výhodou ovšem je možnost plně ovlivnit parametry paměti a využít tak všechny nabízené možnosti. Z tohoto důvodu jsem zvolil realizaci paměti pomocí knihovního primitiva RAMB36. Nutnost použít různá primitiva pro různá FPGA jsem vyřešil pomocí zapouzdření blokové paměti do bloku s neměnným rozhraním. Tento blok v sobě obsahuje pouze instanci primitiva RAM36, jehož rozhraní je transformováno na rozhraní zapouzdřujícího bloku. Při změně FPGA postačí pouze změnit knihovní primitivum a transformovat jeho rozhraní na rozhraní zapouzdřujícího bloku, čímž se vyhneme případným zásahům do dalších bloků paměťového systému řídící jednotky robota.

V implementaci budou do paměti ukládány vektory o šířce čtyř bitů, což bylo také nastaveno při instanciaci primitiva RAMB36. Od šířky datového vektoru je odvozena šířka adresového vektoru, která je v našem případě 13bitů, což umožňuje adresovat 8192 paměťových míst. Jak ukazuje obrázek 4.9, bylo zvoleno omezení rozměrů mapy na 90 x 90 políček, což vyžaduje 8100 paměťových míst.

Zvolená konfigurace je tedy dostačující. Současná implementace nevyužívá možnost uložení paritních bitů společně s daty, proto jsou paritní vstupy a výstupy primitiva RAMB36 nevyužity (viz tabulka 4.1).

4.2.2 Rozhraní pro přístup k paměti

Mimo samotné paměti obsahuje jednotka pro uložení mapy také rozhraní pro přístup k paměti. Tohoto rozhraní je využito z důvodu snadnějšího přístupu ostatních bloků řídící jednotky k paměti. Konkrétně se jedná o blok pro aktualizaci mapy a blok pro hledání cesty v bludišti. Tyto bloky pracují se souřadnicemi x a y jednotlivých políček v mapě a je jednodušší, když mohou adresovat jednotlivá políčka v paměti také ve formě souřadnic. Funkce rozhraní, kterou ukazuje obrázek 4.11, tedy spočívá v transformaci adresy ve formě souřadnic x a y na adresu paměťového místa, tak jak je chápána z pohledu paměti.



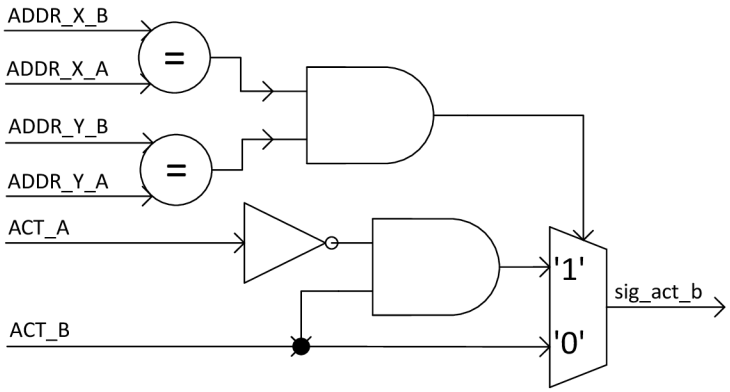
Obrázek 4.11: Transformace adresy políčka v mapě na adresu v paměti.

Rozhraní pro přístup k paměti plní také funkci paměťového kontroléru, který zajišťuje časování signálů přivedených k paměti tak, aby odpovídali specifikaci [18]. Hlídá korektní průběh paměťové operace. Komunikace ostatních bloků s blokem pro uložení mapy tak spočívá pouze v nastavení adresy, požadované operace a signálu ACT, čímž dojde k zahájení paměťové operace. Korektní ukončení paměťové operace je indikováno vysokou úrovní signálu VLD.

4.2.3 Synchronizace portů

Ze specifikace [18] plyne, že probíhá-li zápisová operace na jednom portu paměti, nemůže probíhat žádná operace s paměťovým místem na shodné adrese na druhém portu dvouportové paměti. V řídící jednotce robota by tato situace mohla nastat, proto je nutné ji předem ošetřit a zajistit tak synchronizaci. Z návrhu řídicí jednotky robota plyne, že přednost musí mít zápis do paměti ze strany jednotky pro aktualizaci mapy, tedy ze strany portu A. Jednotka pro řízení robota čte hodnoty uložené v paměti a proto by pro ni nemělo význam číst dříve, než proběhne zápis.

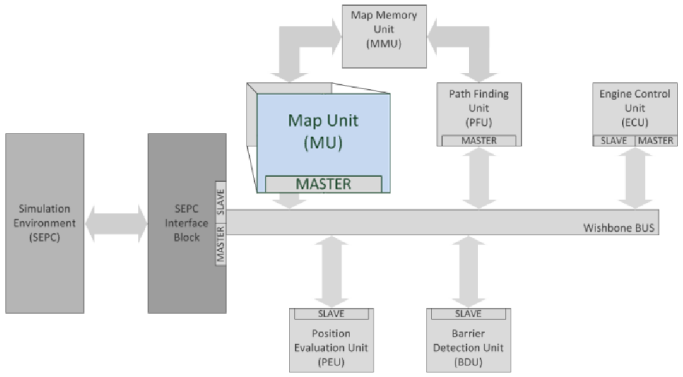
Zvolený způsob synchronizace zachycuje obrázek 4.12. Jedná se o velmi jednoduchou synchronizaci, kdy je přerušen vstupní aktivační signál portu B (ACT_B), který je řízen na základě porovnání adres na jednotlivých portech. V případě, že jsou adresy různé, aktivační signál není přerušen. Naopak, při shodnosti adres je vstupní signál ACT_B pozdržen, dokud je aktivní aktivační signál portu A.



Obrázek 4.12: Synchronizace portů dvouportové paměti.

4.3 Aktualizace mapy

Aktualizaci mapy zajišťuje blok pro aktualizaci mapy (obrázek 4.13), činnost toho bloku spočívá v aktualizaci 5ti paměťových míst na základě informace o aktuální pozici a aktuálním vektoru překážek. Součástí aktualizace mapy je také získávání aktuálních dat ze sběrnice a řízení bloku pro aktualizaci mapy.



Obrázek 4.13: Blok pro aktualizaci mapy.

4.3.1 Blok pro aktualizaci mapy

Hlavní náplní práce tohoto bloku je již uvedená aktualizace paměti na základě informací o aktuální poloze a vektoru překážek. Princip aktualizace mapy znázorňuje obrázek 4.14. Jedná se o sestavení pěti dvojic, sestávajících z adresy paměťového místa a informace o výskytu překážky. Sestavené dvojice jsou postupně procházeny a pro každou dvojici adresa a hodnota se provede aktualizace mapy. Po zpracování všech dvojic blok oznámí ukončení operace nastavením signálu VLD a čeká na požadavek na další operaci.

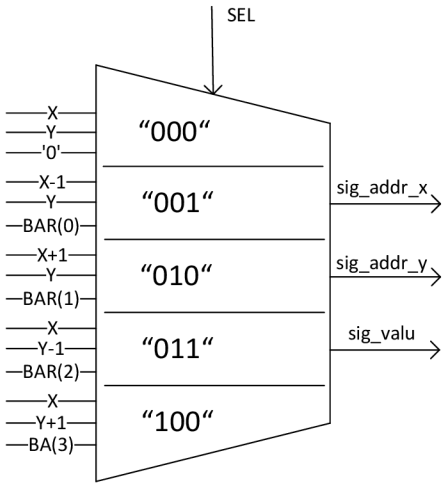
```
for each in
```

addr	X,Y	X-1,Y	X+1,Y	X,Y-1	X,Y+1
value	0	BAR (0)	BAR (1)	BAR (2)	BAR (3)

```
{
    update(addr, value)
}
```

Obrázek 4.14: Pseudokód ukazující činnost MMU při aktualizaci mapy.

Praktická realizace uvedeného principu je založená na využití multiplexoru (obrázek 4.15), jehož výběrový vstup SEL je řízen pomocí čítače. Na vstupu multiplexoru jsou hodnoty, které tvoří výše uvedené dvojice, tedy adresa a příslušná hodnota z vektoru překážek. Na základě hodnoty čítače je vybrána příslušná dvojice, podle které se provede aktualizace paměťového místa. Po potvrzení provedení aktualizace paměťového místa čítač inkrementuje svou hodnotu a dojde ke zpracování další dvojice. Po zpracování všech dvojic řízení jednotky aktivuje signál VLD a čeká na pokyn k další operaci prostřednictvím signálu ACT.

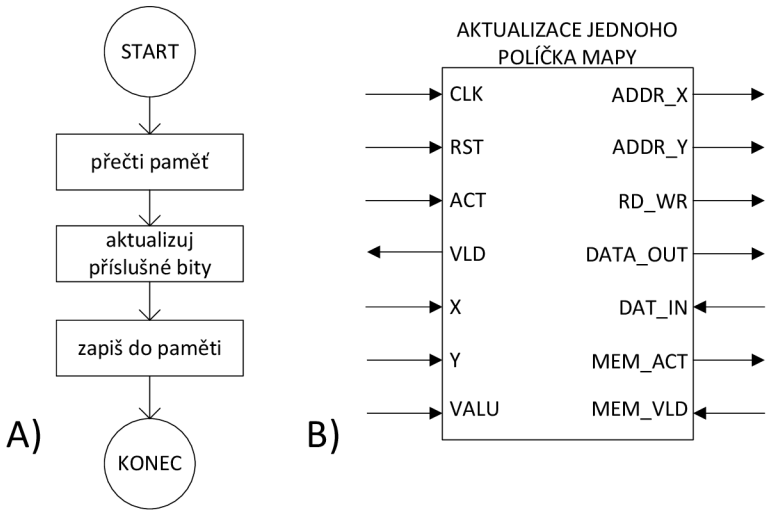


Obrázek 4.15: Multiplexor realizující procházení pěti paměťových míst.

4.3.2 Aktualizace jednoho paměťového místa

Jedno políčko mapy je v paměti reprezentováno čtyřmi bity (obrázek 4.8), u dvou bitů se předpokládá využití pro potřeby algoritmu hledání cesty v bludišti. Aktualizace políčka mapy nespočívá pouze v zapsání nové hodnoty na příslušnou pozici v paměti, ale nejprve je přečten stávající vektor uložený v paměti. V tomto vektoru se nastaví bit informující o aktuálnosti zapsané hodnoty do logické jedničky a především se nastaví bit informující o výskytu překážky. Postup aktualizace jednoho políčka mapy v paměti znázorňuje obrázek 4.16 A.

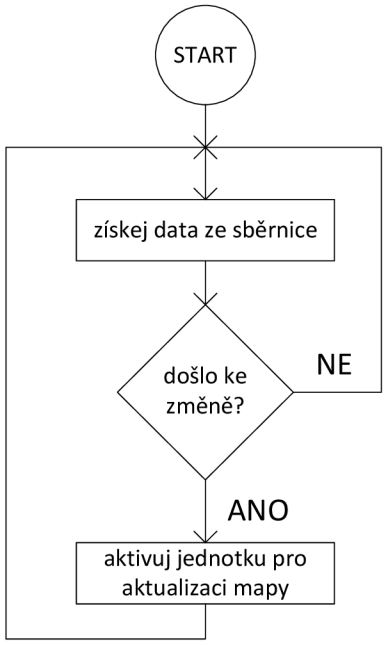
Aktualizaci jednoho paměťového místa obstarává blok, který na vstupu získá adresu ve formě souřadnic políčka v mapě a hodnotu informující o výskytu překážky. Po zahájení činnosti nastavením signálu ACT obstará konečný automat vykonání všech operací, tedy přečtení uloženého vektoru, jeho aktualizaci a následně zápis aktualizovaného vektoru. Tento konečný automat nastavuje příslušné výstupní signály, které jsou připojeny k bloku pro uložení mapy. Vstupy a výstupy bloku pro aktualizaci jednoho paměťového místa ukazuje obrázek 4.16 B.



Obrázek 4.16: Aktualizace jednoho paměťového místa – A) průběh činností, B) vstupy a výstupy.

4.3.3 Řízení aktualizace mapy

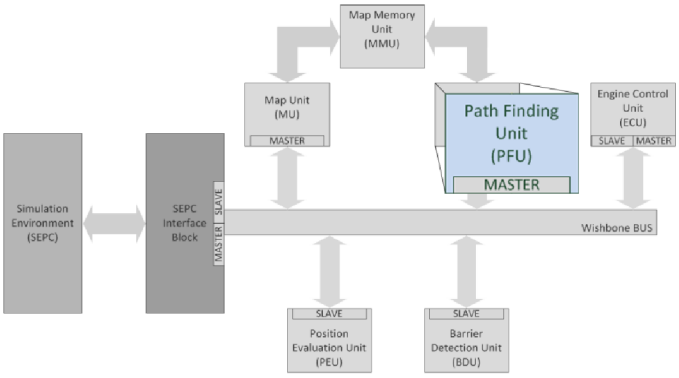
Řízení aktualizace obstarává blok obsahující konečný automat, který řídí signály rozhraní sběrnice na jedné straně a signály rozhraní jednotky pro aktualizaci mapy na straně druhé. Nejprve je nutné získat aktuální data od jednotek pro výpočet aktuální pozice a pro výpočet vektoru překážek prostřednictvím sběrnice. Tato data jsou podrobena testu, zda došlo k jejich změně oproti poslednímu čtení ze sběrnice. Jednalo by se o zbytečnou práci, kdyby se prováděla aktualizace mapy se stejnými hodnotami, jelikož tato aktualizace již byla jednou provedena. V případě, že jsou data stejná jako při posledním čtení, tak se provede nové čtení. Toto se opakuje, dokud nejsou získána pozměněná data, což poukazuje na změnu polohy robota v mapě. Aktuální data jsou předána jednotce pro aktualizaci mapy, konečný automat nastavením aktivačního signálu zahájí její činnost a čeká na ukončení operace. Poté dojde k opakování celého procesu, tedy k čtení hodnot ze sběrnice.



Obrázek 4.17: Vývojový diagram řízení aktualizace mapy.

4.4 Hledání cesty v bludišti

Blok pro hledání cesty v bludišti (obrázek 4.18) je jedním z nejdůležitějších bloků celé řídicí jednotky. Jeho úkolem je vygenerovat posloupnost pozic, které musí robot projít, aby se dostal do určeného cíle. Jednotlivé pozice jsou posílány jednotce pro ovládání robota, která je ukládá do paměti a postupně zpracovává. Tento blok má k dispozici rozhraní sběrnice, pomocí kterého může získat aktuální pozici v mapě a také disponuje přímým přístupem k paměti, ze které může číst informace o překážkách a případně i číst a zapisovat dva rezervní bity, které byly vyhrazeny pro účely algoritmu pro hledání cesty v bludišti.



Obrázek 4.18: Blok pro hledání cesty v bludišti.

4.4.1 Implementace algoritmu „rukou po stěně“

Pro implementaci v řídicí jednotce byl vybrán algoritmus „rukou po stěně“, jelikož je z algoritmů uvedených v kapitole 3.1.5 nejjednodušší a pro počáteční experimenty s řídicí jednotkou robota je plně dostačující. V budoucnu je možné tento algoritmus nahradit některou z metod prohledávání stavového prostoru, ať už implementovaných přímo ve VHDL nebo na procesoru PicoBlaze v FPGA.

Princip algoritmu spočívá v tom, že robot na každé křižovatce odbočí stejným směrem, což má stejný efekt, jako kdyby neustále držel jednu ruku na stěně. V implementaci bylo zvoleno, že robot odbočí vždy vlevo, ale změna na odbočování vpravo je poměrně snadná. Obrázek 4.19 ilustruje činnost algoritmu pomocí pseudokódu. Robot má pro každý směr pohybu definovánu číselnou konstantu (0-3), směr pohybu je uložen v proměnné `smer`, která je na počátku vynulována. Před zahájením pohybu je tato proměnná dekrementována, což simuluje odbočení vlevo. Na základě informací v paměti se zjistí, zda je pozice v daném směru volná. Pokud ano, provede se pohyb pomocí zaslání odpovídajících souřadnic do bloku pro ovládání pohybu robota. Pokud pozice v daném směru volná není, je proměnná inkrementována, což simuluje točení robota vpravo. Je-li pozice v daném směru volná, provede se pohyb. Pokud pozice stále volná není, robot se dále otáčí vpravo, dokud nenarazí na volnou pozici. V nejhorším případě se bude vracet tam, odkud přišel. Tento případ nastane, pokud robot zabloudil ve slepé uličce.

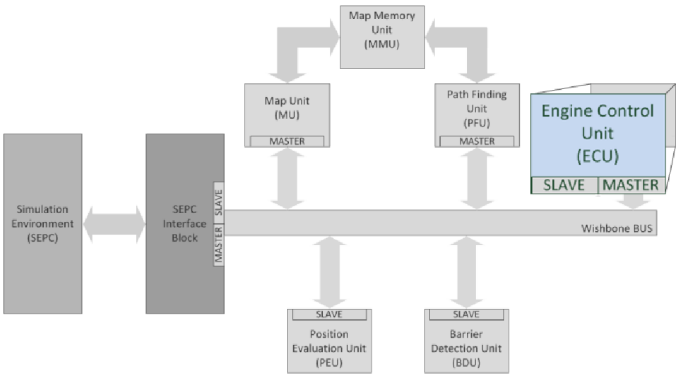
```
smer = 0;
while ( nejsem v cili )
{
      smer = smer + 1;
      while ( policko(smer) == stena )
      {
            smer = smer - 1;
      }
      proved_pohyb(smer);
}
```

Obrázek 4.19: Pseudokód algoritmu pro hledání cesty v bludišti „rukou po stěně“.

Reálná implementace je složitější, protože je realizována pomocí konečného automatu, který musí zajistit provedení všech požadovaných operací. Před zahájením uvedeného algoritmu je nutné získat aktuální pozici robota prostřednictvím sběrnice, tuto pozici poslat jednotce pro ovládání pohybu robota, která jí potřebuje pro správné vyhodnocení směru a až poté se začne provádět uvedený algoritmus. Provedení pohybu spočívá v odeslání cílové souřadnice do bloku pro ovládání pohybu. V případě, že je seznam pozic plný, je třeba počkat, dokud se neuvolní a poté může algoritmus pokračovat. Zároveň je nutné změnit informaci o aktuální pozici na novou pozici. Test, zda je daná pozice volná, spočívá v přečtení odpovídající hodnoty z paměti. Kontrola přečtené hodnoty říká, zda je pozice volná, případně zda již bylo požadované políčko prozkoumáno. Ověření, zda robot došel do cílové pozice je z uvedených operací nejjednodušší a spočívá pouze v porovnání aktuální a cílové pozice.

4.5 Ovládání pohybu robota

Pohyb robota ovládá jednotka ECU (obrázek 4.20). Získává hodnoty od jednotky pro hledání cesty v bludišti, ukládá si je do seznamu a na základě nich řídí nastavování rychlostí pohybu robota v horizontálním a vertikálním směru. Dráha, kterou robot urazí, je dána dobou, po kterou je nastavena rychlost. Součástí této jednotky je seznam pozic, který je implementovaný pomocí paměti FIFO a také blok pro samotné ovládání rychlosti a doby pohybu robota.



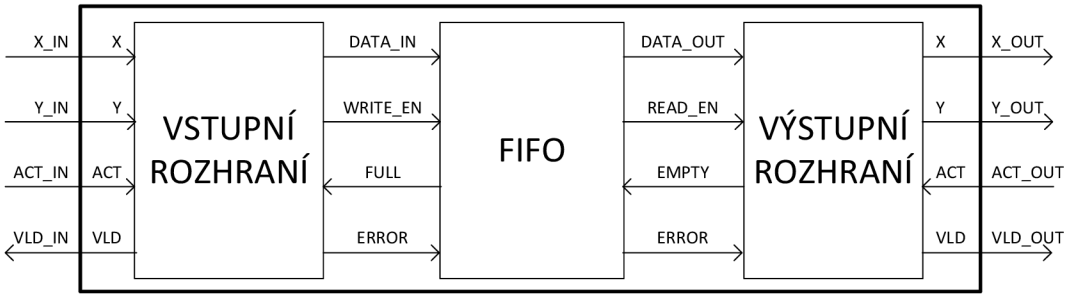
Obrázek 4.20: Blok pro ovládání pohybu robota.

4.5.1 Implementace seznamu

Seznam je důležitou součástí této jednotky, slouží pro uchovávání hodnot, které je třeba projít, aby se robot dostal do cíle. Seznam postupně plní jednotka pro hledání cesty v bludišti, která nejprve pošle

aktuální pozici a poté posílá vypočítané pozice, které je třeba projít. Tento seznam je implementován pomocí paměti FIFO (First In – First Out), která přesně odpovídá požadavkům kladeným na tento seznam. Je třeba ukládat pozice, které jsou následně čteny ve stejném pořadí, v jakém byly uloženy a nejsou potřeba žádné speciální průchody seznamem, které by paměť typu FIFO neumožňovala.

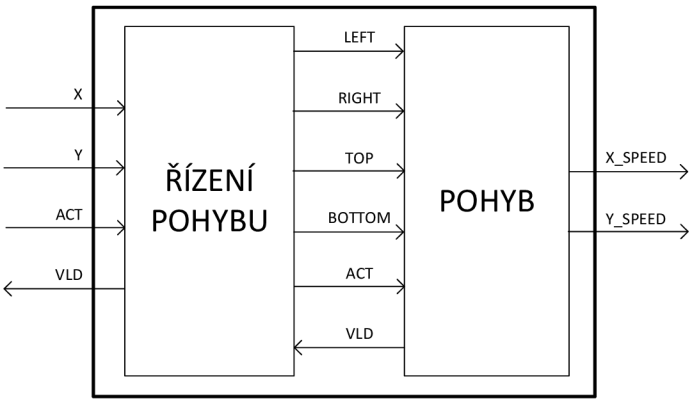
Paměť typu FIFO je možné implementovat pomocí blokové paměti RAM dostupné na FPGA nebo paměti sestavené z LUT. Byla zvolena implementace pomocí LUT, aby bylo možné testovat metodiky zaměřené na zabezpečení této paměti. Každé paměťové místo paměti FIFO musí být schopné pojmout informaci o pozici robota ve formě souřadnic. Šířka ukládaného vektoru je tedy dvojnásobná oproti šířce vektoru jedné souřadnice. Hloubka paměti FIFO byla zvolena 15 položek, což vidím jako dostačující. V případě, že by bylo nutné tuto hloubku rozšířit, tak postačí pouze změnit konstantu. Paměť je zapouzdřena v bloku, který obsahuje vstupní a výstupní přístupové rozhraní (obrázek 4.21). Pomocí vstupního rozhraní se komunikuje s jednotkou pro hledání cesty v bludišti. Zde jsou přiváděny souřadnice pozice, která se má uložit do paměti, nastavením aktivačního signálu se zahájí proces uložení, o jehož konci informuje potvrzovací signál. Výstupní rozhraní slouží k čtení položek (souřadnice x a y), je vybaveno výstupním vektorem, na kterém se objeví uložená data. Proces čtení je opět zahájen nastavením aktivačního signálu a o přítomnosti platných dat informuje potvrzovací signál. Vnitřní činnost vstupního a výstupního rozhraní spočívá v řízení signálů paměti tak, aby všechny operace proběhly korektně.



Obrázek 4.21: Vnitřní struktura bloku pro ukládání seznamu pozic.

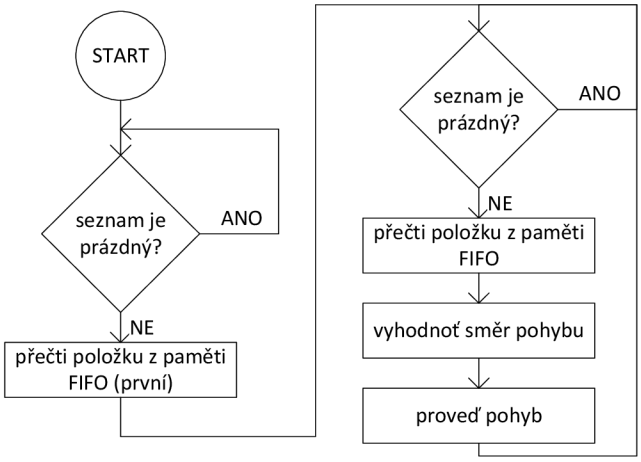
4.5.2 Implementace ovládání rychlosti

Blok pro ovládání rychlosti přímo řídí pohyb robota a sestává ze dvou částí, které ukazuje obrázek 4.22. První částí je blok, který na svém vstupu očekává informaci o směru pohybu. Na základě této informace provede nastavení rychlosti pohybu v příslušném směru. Zároveň zahájí odpočítávání časového intervalu, čímž zajistí pohyb robota o jedno políčko v mapě v nastaveném směru. Odpočítávání časového intervalu je implementováno pomocí čítače, který s každou periodou hodinového signálu inkrementuje výstupní hodnotu. Jakmile je na výstupu čítače hodnota odpovídající nastavené konstantě, je čítání ukončeno a oznámeno ukončení pohybu nastavením potvrzovacího signálu.



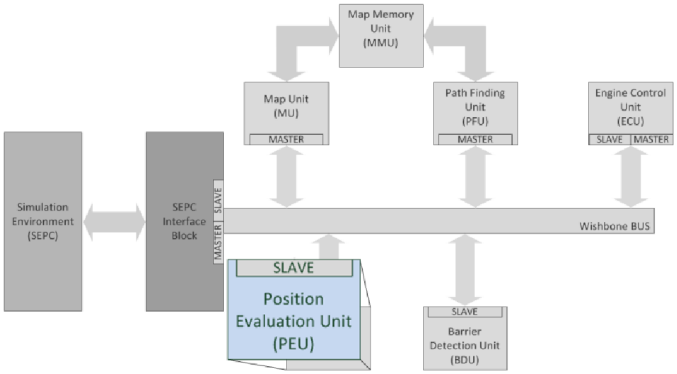
Obrázek 4.22: Vnitřní struktura bloku pro ovládání rychlosti.

Druhou částí bloku pro ovládání rychlosti robota je řídicí blok, který obstarává čtení uložených hodnot z paměti FIFO, jejich zpracování, nastavení informace o požadovaném směru pohybu a zahajuje pohyb nastavením aktivačního signálu. Průběh činností ukazuje obrázek 4.23. Jednotka pro hledání cesty v bludišti na začátku své činnosti provede uložení aktuální pozice do paměti FIFO, která je následně přečtena jako první a slouží k vyhodnocení směru pohybu. Po přečtení další hodnoty ze seznamu jsou tyto dvě hodnoty porovnány a je vyhodnocen směr, kterým se robot musí vydat, aby se přesunul na příslušnou pozici. Vyhodnocování směru pohybu v dalších iteracích již probíhá na základě aktuálně přečtené hodnoty ze seznamu a předchozí přečtené hodnoty, což je po provedení pohybu aktuální pozice robota. Představené řízení ovládání rychlosti robota je implementováno pomocí konečného automatu, který provádí nastavování příslušných signálů tak, aby byly provedeny požadované činnosti.



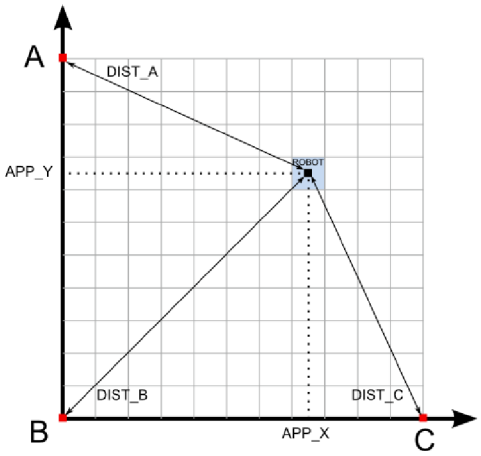
Obrázek 4.23: Vývojový diagram průběhu ovládání rychlosti robota.

4.6 Výpočet aktuální pozice



Obrázek 4.24: Blok pro výpočet pozice robota v mapě bludiště.

Výpočet pozice robota v mapě provádí blok PEU (obrázek 4.24).V kapitole návrhu bylo uvedeno, že výpočet pozice robota v mapě bude probíhat na základě informací o vzdálenosti od tří kontrolních bodů v mapě a k výpočtu bude použita Pythagorova věta. V této kapitole je podrobně rozebrán postup výpočtu pozice robota ve formě souřadnic, k ilustraci výpočtu slouží obrázek 4.25, který již byl uveden v kapitole věnující se návrhu řídicí jednotky robota.



Obrázek 4.25: Pozice robota v mapě.

Pro výpočet souřadnice x lze použít soustavu dvou rovnic (1) a (2), obě jsou odvozeny z obrázku pomocí Pythagorovy věty.

$$APP\_X^2 + APP\_Y^2 = DIST\_B^2 \quad (1)$$

$$(X\_MAX - APP\_X)^2 + APP\_Y^2 = DIST\_C^2 \quad (2)$$

Z uvedené soustavy dvou rovnic pomocí několik úprav můžeme vyjádřit APP_X (3), což je souřadnice robota na ose x. Z pohledu řídicí jednotky se robot v mapě pohybuje po jednotlivých políčcích. Proto je nutné vypočítanou souřadnici převést na hodnotu souřadnice vyjádřené v políčcích mapy, což zajistí vydělení souřadnice na ose x velikostí políčka BOX_SIZE (4).

$$APP\_X = \frac{X\_MAX^2 - DIST\_C^2 + DIST\_B^2}{2 \cdot X\_MAX} \quad (3)$$

$$X = \frac{APP\_X}{BOX\_SIZE} \quad (4)$$

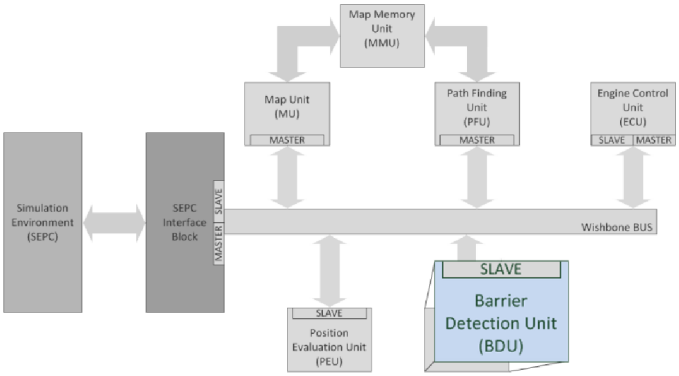
Postup výpočtu souřadnice y je shodný s uvedeným postupem výpočtu souřadnice x, proto jsou zde uvedeny pouze výsledné vztahy výpočtu APP_Y (5) a Y (6).

$$APP\_Y = \frac{Y\_MAX^2 - DIST\_A^2 + DIST\_B^2}{2 \cdot Y\_MAX} \tag{5}$$

$$Y = \frac{APP\_Y}{BOX\_SIZE} \tag{6}$$

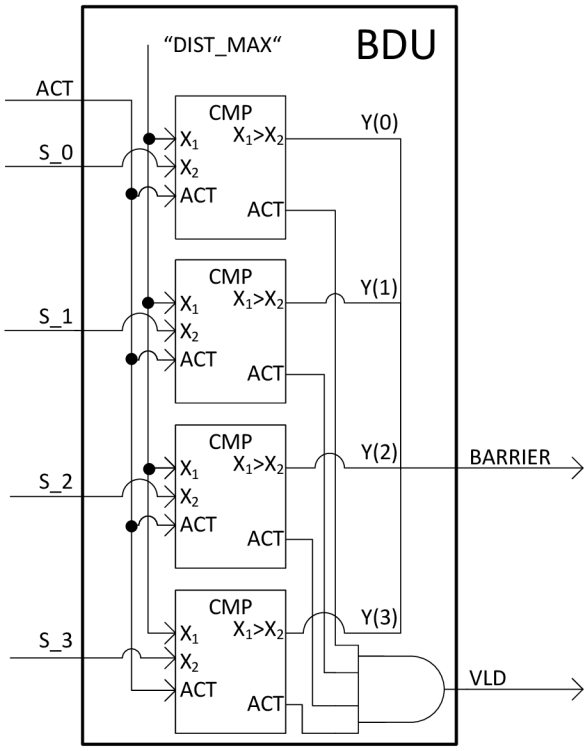
Činnost bloku pro výpočet pozice robota spočívá pouze v transformaci vstupních hodnot, tedy informací o vzdálenosti od kontrolních bodů na výstupní hodnoty ve formě souřadnic robota v mapě. Náplní práce tohoto bloku jsou tedy aritmetické výpočty odpovídající rovnicím (3) a (4) pro souřadnici x a (5) a (6) pro souřadnici y. Implementované výpočty plně korespondují s těmito vztahy. V aktuální implementaci řídicí jednotky robota jsou použity násobičky a sčítačky dostupné na FPGA. Dělení je realizováno pomocí bitových posunů, což vede ke zrychlení výpočtu, ale zároveň také k omezení rozměrů mapy a jednoho políčka mapy na mocniny dvou. V budoucnu se počítá s vlastní implementací všech aritmetických operátorů, včetně dělení.

4.7 Výpočet vektoru překážek



Obrázek 4.26: Blok pro výpočet vektoru překážek.

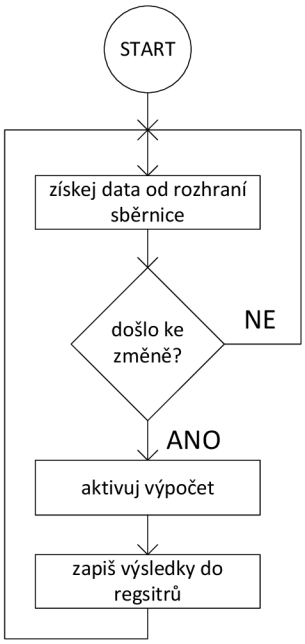
Výpočet vektoru překážek obstarává blok BDU (obrázek 4.26) s velmi jednoduchou implementací, kterou ukazuje obrázek 4.27. Jeho činnost spočívá pouze v porovnání vstupních hodnot informujících o vzdálenosti překážek od senzorů robota (umístěných na všech jeho čtyřech stěnách) s referenční hodnotou (DIST_MAX). V případě, že je vzdálenost od překážky menší, než je nastavená referenční hodnota, tak se v daném směru vyskytuje překážka a dojde k nastavení bitu ve vektoru překážek. Naopak vzdálenost překážky větší než referenční hodnota znamená, že v daném směru se překážka nevyskytuje.



Obrázek 4.27: Implementace výpočtu vektoru překážek.

4.8 Řízení výpočtu pozice a vektoru překážek

Představené bloky pro výpočet pozice robota v mapě a vektoru překážek transformují vstupní hodnoty na výstupní na základě nastavení aktivačního signálu a o ukončení výpočtu informují nastavením potvrzovacího signálu. Tyto bloky je třeba doplnit o obvody, které budou řídit jejich činnost. Obrázek 4.28 znázorňuje postup činností při řízení výpočtu pozice v mapě a vektoru překážek.

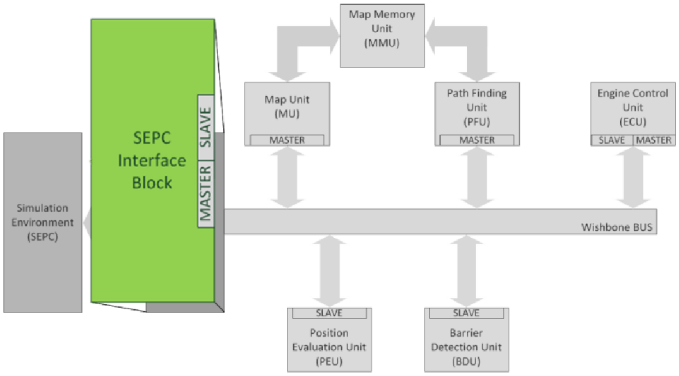


Obrázek 4.28: Vývojový diagram řízení výpočtu pozice v mapě a vektoru překážek.

Blok řídicí výpočet má k dispozici přístup k rozhraní sběrnice, na jehož výstupu jsou hodnoty, které byly zaslány prostřednictvím sběrnice. Tyto hodnoty jsou přečteny a porovnány s předchozími hodnotami, čímž dojde k detekci změny vstupních hodnot. Pokud nebyla detekována změna, opakuje se čtení dat. V okamžiku, kdy dojde k detekci změny vstupních dat, jsou hodnoty předány bloku pro výpočet pozice (vektoru překážek) a je zahájen výpočet. Po ukončení výpočtu jsou výsledky zapsány do příslušných registrů rozhraní sběrnice tak, aby mohli být v případě požadavku odeslány přes sběrnici. Popsaný proces řízení je opět implementován pomocí konečného automatu, který řídí potřebné signály a zajišťuje vykonání požadovaných činností.

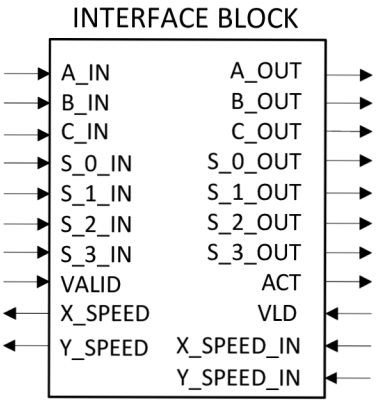
4.9 Rozhraní řídicí jednotky

Hlavním úkolem bloku rozhraní (obrázek 4.29) řídicí jednotky je distribuce hodnot ze senzorů a kontrolních bodů (získaných ze simulačního prostředí) prostřednictvím sběrnice k dalším blokům. Hodnoty od kontrolních bodů jsou distribuovány k bloku pro výpočet pozice robota v mapě a hodnoty od senzorů překážek jsou distribuovány k bloku pro výpočet vektoru překážek.



Obrázek 4.29: Blok rozhraní řídicí jednotky.

Dalším úkolem rozhraní je předávání hodnot rychlosti pohybu k robotovi. Z pohledu vstupů a výstupů je blok rozhraní na jedné straně vybaven signály pro komunikaci s rozhraním sběrnice a na straně druhé signály pro komunikaci s robotem (obrázek 4.30). Celý blok je opět implementován jako konečný automat, který vhodným nastavováním jednotlivých signálů zajistí distribuci hodnot. Ve vhodný okamžik provede přečtení vstupních hodnot. Ty předá bloku rozhraní sběrnice, který po nastavení aktivačního signálu provede rozeslání hodnot do příslušných registrů uvedených bloků. Rozhraní sběrnice nastavením potvrzovacího signálu oznámí úspěšné odeslání všech hodnot a celý cyklus se opakuje. Vhodný okamžik pro čtení vstupních hodnot je takový, kdy robot stojí a nepohybuje se. V době, kdy nastane tato situace, je robot přibližně ve středu políčka mapy. Nemůže tak nastat situace, že hodnoty dorazí ve chvíli, kdy se robot nachází na přelomu dvou políček. Toto by totiž mohlo vést k nepřesnostem ve vyhodnocení pozice a ve výpočtu vektoru překážek.



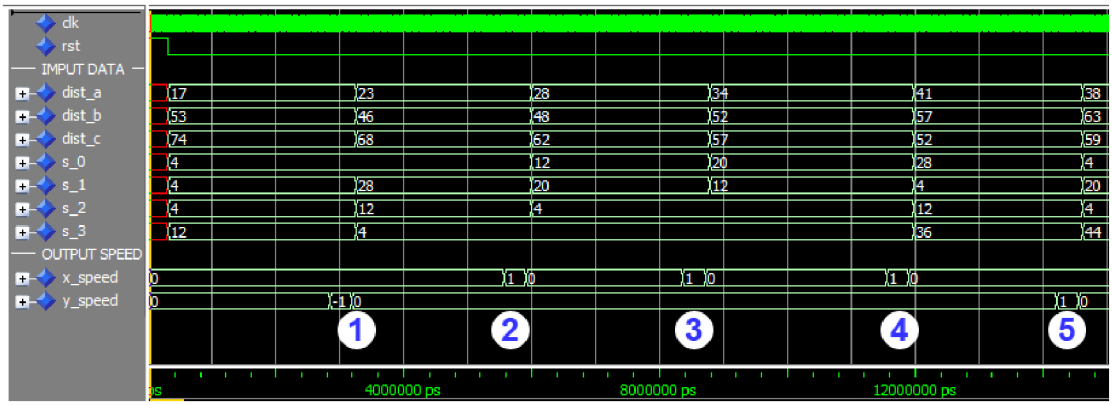
Obrázek 4.30: Blok rozhraní řídicí jednotky robota – zobrazení vstupních a výstupních signálů.

5 Demonstrace funkčnosti

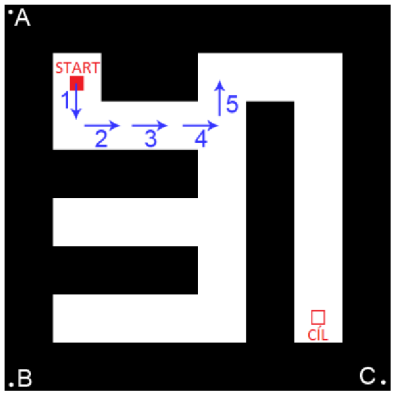
Představená řídicí jednotka byla implementována v jazyce VHDL a její funkčnost byla průběžně ověřována prostřednictvím simulačního programu ModelSim [19]. Výsledná implementace byla také ověřena na reálném FPGA, pro tyto účely byla použita vývojová deska ML506 [20] od firmy Xilinx. Simulace robota, kterého řídicí jednotka implementovaná na FPGA řídí, je realizována pomocí simulačního prostředí Player/Stage [21].

5.1 Ověření funkčnosti pomocí simulace

V průběhu vývoje byla funkčnost řídicí jednotky robota ověřována pomocí programu ModelSim pro simulaci číslicových systémů (v našem případě popsaných ve VHDL). Ověřovány byly nejprve jednotlivé komponenty systému a následně i celý systém. Grafické zobrazení simulace zachycuje obrázek 5.1, jsou zde vidět vstupní signály, které nesou informace o překážkách a pozici robota v mapě, a výstupní signály informující o rychlosti pohybu robota ve směru osy x a y. Vstupní signály jsou generovány v testovacím obvodu. Jsou voleny tak, aby odpovídaly reálnému jednoduchému bludišti, které zobrazuje obrázek 5.2. Na tomto obrázku je možné vidět počáteční a cílovou pozici robota. Zobrazené průběhy signálů odpovídají projití pouze očíslované části bludiště, časové průběhy odpovídající projití celé trasy jsou zobrazeny v Příloze C.



Obrázek 5.1: Časové průběhy vstupních a výstupních signálů řídicí jednotky.

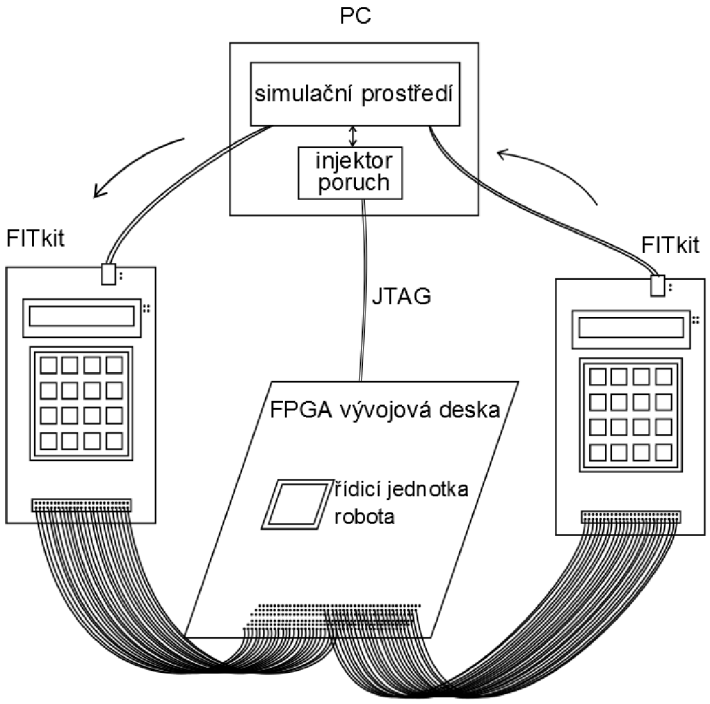


Obrázek 5.2: Testovací bludiště.

5.2 Implementace na reálném FPGA

Implementovaná řídicí jednotka byla ověřena také pomocí reálného FPGA. Jak již bylo zmíněno, jedná se o FPGA dostupné na vývojové desce ML506. V této kapitole je stručně představena uvedená vývojová deska a FPGA. Následuje také stručný popis simulačního prostředí Player/Stage.

Obrázek 5.3 ukazuje celkové schéma propojení řídicí jednotky robota na FPGA a počítače se simulačním prostředím. Komunikace mezi vývojovou deskou, na které je umístěna řídicí jednotka robota, a počítačem se simulačním prostředím probíhá prostřednictvím dvou FITkitů [22]. Jeden FITkit slouží jako prostředník při přenosu dat z řídicí jednotky robota do simulačního prostředí a druhý FITkit zajišťuje komunikaci v opačném směru. FITkity jsou s prostředím propojeny pomocí USB sběrnice, s vývojovou FPGA deskou jsou FITkity propojeny pomocí sady signálních vodičů. K vývojové desce je navíc připojen injektor poruch, pomocí kterého budeme v budoucnu testovat reakce systému na uměle injektované poruchy. Injektor je realizován na PC a je připojen pomocí JTAG rozhraní vývojové desky.



Obrázek 5.3: Schéma propojení jednotlivých částí testovací platformy.

5.2.1 Vývojová deska ML506

Vývojová deska ML506 je osazena obvodem Virtex5, který nese označení XC5VSX50T. Dále je deska osazena běžně používanými periferiemi, jako jsou tlačítka, LCD displej, USB rozhraní, JTAG rozhraní a další používané periferie. Kompletní přehled je uveden v [20].

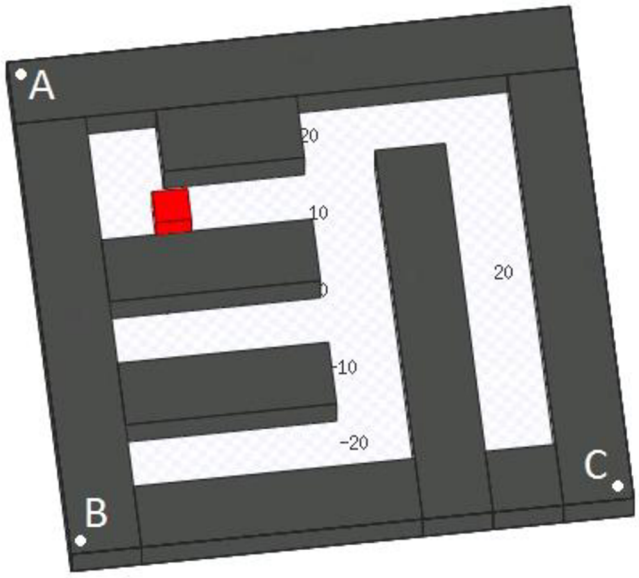
Pro účely řídicí jednotky jsou důležité především parametry dostupného FPGA a přítomnost dostatečného množství vstupně/výstupních pinů pro propojení s robotem, přesněji se simulačním prostředím. Virtex5 je moderní FPGA obvod, který je schopný pracovat na frekvenci až 550MHz a obsahuje 8 160 *slices*, které jsou složeny ze 4 LUT a 4 klopných obvodů. Dále je zde dostupných 132 bloků paměti BRAM, což je celkem 4 752Kb dostupné paměti. Maximální počet vstupně/výstupních uživatelských pinu je 480. Podrobnější popis FPGA Virtex5 je uveden v [23] .

5.2.2 Simulační systém Player/Stage

Simulační prostředí umožňuje vygenerovat virtuální bludiště, ve kterém se pohybuje robot. Tento robot je ovládán řídicí jednotkou, což nám umožňuje sledovat jeho reakce na poruchy. Za tímto účelem byl vybrán Player Project [21]. Jedná se o projekt, který má sloužit pro výzkum v oblasti robotů a senzorových sítí. Tento simulační systém byl přizpůsoben našim požadavkům a byl propojen s vývojovou deskou, na které je implementována řídicí jednotka robota. Připravené simulační prostředí jsem měl k dispozici po celou dobu vytváření řídicí jednotky robota.

5.2.3 Ukázky robota v bludišti

Obrázek 5.4 ukazuje bludiště v simulačním prostředí. Toto bludiště odpovídá testovacímu bludišti, které zobrazuje obrázek 5.2. Je zde vidět také robot, který se nachází na startovací pozici a jeho úkolem je dojít do cíle. Průchod robota bludištěm je možné vidět na videu, které je součástí přiloženého CD (Příloha D).



Obrázek 5.4: Robot v bludišti v simulačním prostředí.

6 Návrh zabezpečení proti poruchám

V předchozích kapitolách byl představen návrh a implementace řídicí jednotky robota určeného pro samočinný pohyb v bludišti. Navržená a implementovaná řídicí jednotka, dle původního záměru, zahrnuje různé aspekty návrhu číslicových systémů. Jedná se o:

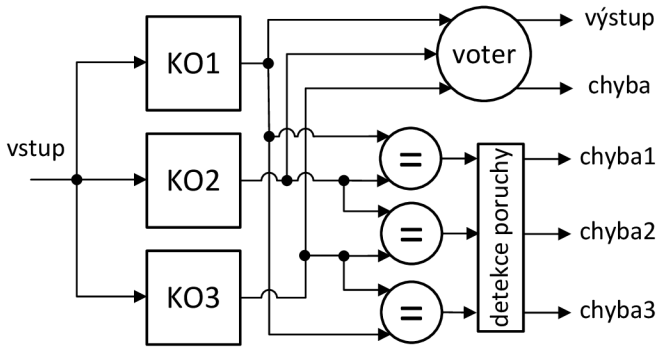
- kombinační obvody,
- sekvenční obvody řízené konečným automatem,
- sekvenční obvody bez explicitního řízení konečným automatem,
- sběrnici,
- paměti (bloková paměť a LUT paměť).

Řídicí jednotka bude sloužit jako komplexní obvod pro ověřování metodik pro zajištění odolnosti systémů založených na FPGA proti poruchám. V současné implementaci řídicí jednotky nejsou zatím použity žádné techniky zajišťující odolnost proti poruchám. Řídicí jednotka je tak náchylná k vzniku chyb, protože při výskytu poruchy hrozí selhání celého systému. Proto jedním z hlavních cílů této práce je navrhnout i možnosti zajištění odolnosti proti poruchám u hlavních bloků celé řídicí jednotky.

6.1 Kombinační obvody

Kombinační obvody jsou jedním z typů číslicových systémů, které jsou obsaženy v implementované řídicí jednotce. Příkladem je blok pro detekci překážek (BDU). Dále jsou v řídicí jednotce kombinační obvody obsaženy jako nejrůznější dílčí podbloky, například v bloku pro řízení mechanických částí robota (ECU) slouží kombinační obvod k výpočtu směru pohybu.

Pro zabezpečení těchto bloků bylo navrženo použití tří-modulové redundance (TMR) v kombinaci s využitím řadiče částečné dynamické rekonfigurace pro obnovu bloků napadených poruchou. Obrázek 6.1 ukazuje zabezpečení pomocí techniky TMR, jsou zde zobrazeny tři instance kombinačního obvodu (KOx), které provádí shodné operace se shodnými daty a výsledky jsou porovnávány pomocí rozhodovacího obvodu (voter). Zároveň je v činnosti obvod pro detekci poruchy, který v případě výskytu poruchy určí, která instance sekvenčního obvodu byla poruchou napadena. Takto označený obvod je poté možné obnovit pomocí představeného řadiče částečné dynamické rekonfigurace, jehož vstupem je identifikace obvodu napadeného poruchou.



Obrázek 6.1: Zabezpečení kombinačních obvodů proti poruchám pomocí TMR.

6.2 Sekvenční obvody

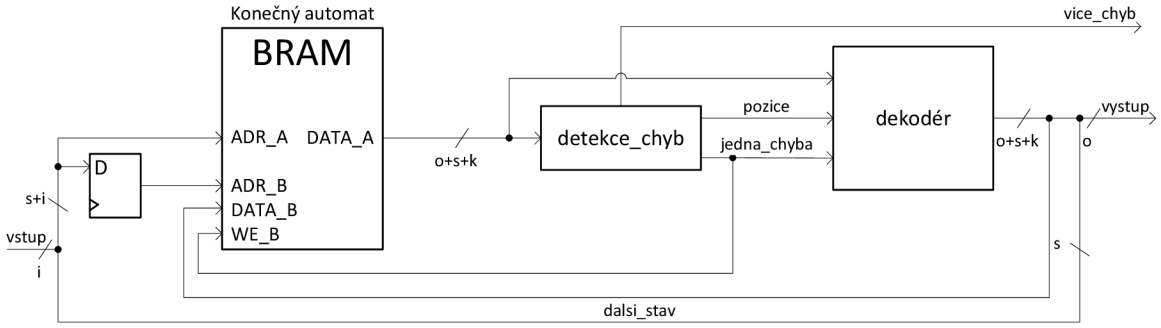
Sekvenční obvody můžeme rozdělit na dva druhy. Prvním jsou sekvenční obvody řízené explicitně definovaným konečným automatem a druhým jsou sekvenční obvody bez explicitně definovaného konečného automatu. Ty mohou být řízeny například pomocí čítače. V navržené řídicí jednotce se vyskytují zástupci obou uvedených kategorií, proto je dále uveden návrh na zabezpečení sekvenčních obvodů z obou kategorií.

6.2.1 Sekvenční obvody řízené konečným automatem

Konečné automaty se v řídicí jednotce robota vyskytují ve velkém množství. Každý blok obsahuje rozhraní sběrnice, které je řízeno pomocí konečného automatu. Stejně tak většina funkčních bloků je řízena konečným automatem, který zajišťuje správné provádění všech aktivit. V centru celé řídicí jednotky stojí konečný automat, který zajišťuje hledání cesty v bludišti. V této kapitole jsou zmíněny obecné principy, pomocí kterých je možné jednotlivé konečné automaty zabezpečit.

V [24] jsou představeny různé metodiky pro zabezpečení konečných automatů proti poruchám. Jádrem tohoto článku je metodika pro návrh konečných automatů odolných proti poruchám s využitím blokové paměti a tato technika bude použita v řídicí jednotce robota.

Běžně jsou konečné automaty navrhovány tak, že obsahují registry, které uchovávají informaci o stavu a kombinační obvod pro výpočet budoucího stavu. Z tohoto návrhu vyplývá možnost zabezpečení pomocí TMR, kdy jsou použity tři instance stejného konečného automatu, a výslednou hodnotu vybere rozhodovací obvod. Je zde také možnost použít duplexní architekturu a stavy automatů reprezentovat pomocí kódu umožňujícího detekci a opravu chyb, například Hamingova kódu. Metodika představená v [24] implementuje konečný automat pomocí paměti, přesněji blokové paměti dostupné na FPGA. Kombinace vstupní adresy a aktuálního stavu slouží jako adresa, pomocí které je identifikováno paměťové místo, kde je uložena informace o budoucím stavu a výstupní hodnota. Pro zabezpečení obsahu paměti proti poruchám je použit samoopravný Hammingův kód. Každé políčko paměti je zakódováno pomocí tohoto kódu, čímž je zajištěna detekce a korekce jednoduchých chyb. V případě výskytu více chyb je nutné provést obnovení obsahu paměti pomocí částečné dynamické rekonfigurace.



Obrázek 6.2: Implementace konečného automatu v paměti zabezpečeného Hammingovým kódem.

Obrázek 6.2 znázorňuje představenou implementaci konečného automatu v paměti. Příští stav (vektor o šířce `s`) a výstupní hodnota (šířka `o`) jsou uloženy v dvouportové paměti společně s kontrolními bity Hamingova kódu (šířka `k`). Na základě adresy složené ze vstupního vektoru (šířka `i`) a aktuálního stavu (šířka `s`) je prostřednictvím portu A přečtena příslušná paměťová položka. Blok

`detekce_chyb` provede kontrolu přečtené hodnoty. Detekce jednoduché chyby je indikována nastavením signálu `jedna_chyba`, tento signál zároveň slouží k indikaci zápisu na port B. Signál `pozice` informuje o pozici chyby, kterou dekodér opraví a opravená hodnota je přivedena na vstup paměti kde dojde k jejímu uložení. Opravená hodnota je zároveň výstupním vektorem. Signál `vice_chyb` nese informaci o výskytu vícenásobné chyby. Paměť je zabezpečená proti poruchám díky použití Hammingova kódu, ale pro zabezpečení pomocných obvodů musí být použita jiná technika, například TMR.

6.2.2 Řízení bez explicitně definovaného konečného automatu

V implementované řídicí jednotce se vyskytují i sekvenční obvody, které nejsou řízeny konečným automatem. Jedná se o aritmetické výpočty, které probíhají v jednotce pro výpočet pozice robota v mapě (PEU). Řízení konečným automatem také chybí v jednotce pro aktualizaci mapy, kde je aktualizace jednotlivých pozic prováděna v cyklu, který je řízen čítačem.

Stejně jako u kombinačních obvodů bude pro zabezpečení sekvenčních obvodů bez explicitně definovaného konečného automatu použita technika TMR. Použití je shodné, jako v případě kombinačních obvodů. Nabízí se také možnost zabezpečení pomocí duplexní architektury, která je uvedena v kapitole 2.5.3. Zde by ale mohly nastat problémy s vygenerováním odpovídajícího hlídacího obvodu, z tohoto důvodu bude použita technika TMR.

6.3 Paměti

Paměť se v řídicí jednotce vyskytuje ve dvou blocích a v každém z nich je použita jiná realizace paměti. Prvním blokem, který obsahuje paměť, je blok pro uložení mapy (MMU), který ukládá mapu do blokové paměti BRAM. Druhým blokem, kde je použita paměť, je blok pro ovládání robota (ECU), přesněji jeho část uchovávající seznam pozic, které je nutné projít. Zde je paměť realizována z LUT a je používaná jako paměť typu FIFO.

V této kapitole navrhnu zabezpečení uvedených bloků, přesněji uvedených typů pamětí. Dále se také zaměřím na možnost opravy uložených hodnot při detekci poruchy pomocí techniky zvané *memory scrubbing*.

6.3.1 Možnosti zabezpečení

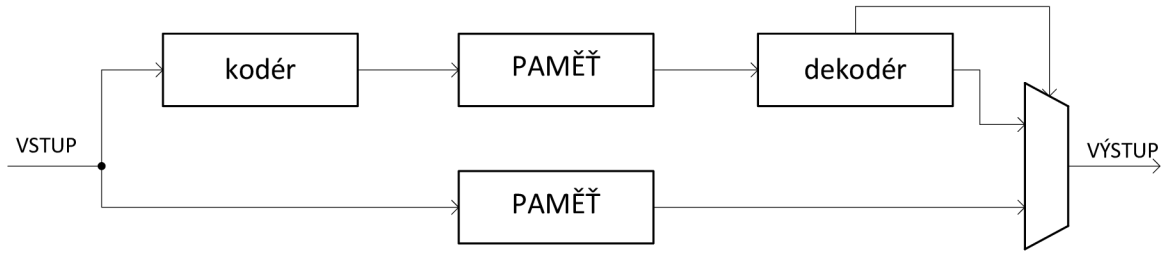
Možnosti zabezpečení pamětí jsou diskutovány v publikaci [25]. Autoři této publikace uvádí tři základní možnosti zabezpečení odolnosti proti poruchám u pamětí. Těmito možnostmi jsou:

- TMR (ztrojení),
- duplikace v kombinaci s kódy pro detekci chyb,
- kódy pro detekci a opravu chyb.

Ztrojení zabezpečovaného bloku (TMR) je klasická metoda zabezpečení proti poruchám. Dle výsledků experimentů uvedených v [25] je tato metoda nejspolehlivější, nicméně její nevýhodou je poměrně velký nárůst spotřebovaných zdrojů.

Další možností pro zabezpečení paměti proti poruchám je duplikace paměti v kombinaci s kódy pro detekci chyb (obrázek 6.3). Jedna instance paměti obsahuje uložená data v kódu, který umožňuje detekci chyb. Příkladem může být jednoduchá parita. Druhá instance paměti uložená data

nijak speciálně nekóduje. V případě detekce chyby v první instanci je jako výstup použita hodnota z druhé instance. Tento přístup má menší nárůst použitých zdrojů než TMR. Nárůst je zde způsoben duplikací paměti, jedna paměť musí být větší z důvodu redundantních bitů detekčního kódu a také je potřebná logika pro kódování a dekódování dat uložených v první instanci.



Obrázek 6.3: Zabezpečení paměti pomocí duplikace a kódu pro detekci chyb.

Použití kódů pro detekci a opravu chyb spočívá v zakódování dat ukládaných do paměti pomocí speciálního kódu (obrázek 6.4), který je schopný detekovat a opravit chybu. Příkladem takového kódu může být Hamingův kód. Výhodou je menší nárůst spotřebovaných zdrojů oproti předešlým způsobům. Jemný nárůst je způsoben pouze rozšířením o obvod kodéru a dekodéru a podle počtu redundantních bitů použitého kódu.

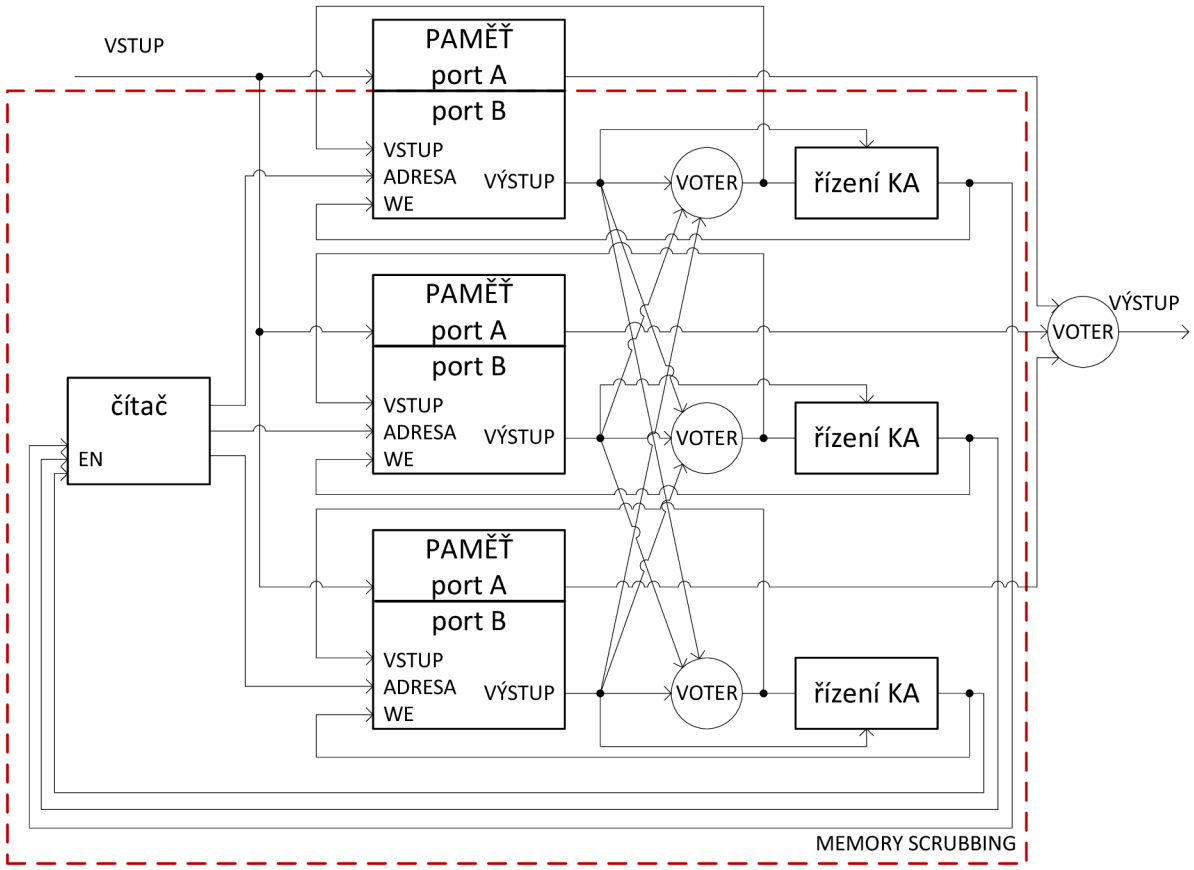
Obrázek 6.4: Zabezpečení paměti pomocí kódu pro detekci a opravu chyb.

6.3.2 Oprava uložených hodnot

Uvedené techniky pro zabezpečení paměti jsou schopny maskovat omezené množství poruch. V případě, že je toto množství překročeno, dojde k produkování chybných výsledků, stejně jak bylo uvedeno v kapitole 2.5. V případě pamětí bohužel není možné použít k obnově bezporuchového stavu rekonfiguraci, ale je nutné použít techniku zvanou *memory scrubbing*. Tato technika spočívá v čtení jednotlivých položek v paměti, kontrole jejich správnosti a v případě detekce chyby k uložení správné hodnoty do paměti. Rozlišujeme deterministickou a nedeterministickou verzi opravy paměti.

Obrázek 6.5 zobrazuje deterministickou verzi, která zajišťuje pravidelné čtení a obnovu paměti. Na obrázku je použit *memory scrubbing* v kombinaci s TMR, ale je možná kombinace se všemi uvedenými metodami pro zabezpečení paměti. Jsou zde zobrazeny tři instance dvouportové paměti. Jedná se o blokovou paměť, kde jeden port slouží k čtení a zápisu hodnot a druhý port je využit pro pravidelné čtení hodnot, které zajišťuje čítač. Přečtené hodnoty ze všech pamětí jsou porovnávány a v případě, že je detekována nesprávná hodnota v některé z nich, dojde k zapsání správné hodnoty, kterou produkuje *voter* na základě informací ze zbylých dvou nepoškozených pamětí.

Nedeterministická varianta spočívá v absenci řídicího prvku, tedy čítače. Hodnoty jsou čteny a případně opravovány pouze v případě, že je požadavek na čtení z paměti. Tento přístup je výhodný díky menším nárokům na použité zdroje. Nicméně je zde větší pravděpodobnost, že dojde k nevratnému poškození dat v paměti v případě výskytu většího množství poruch.



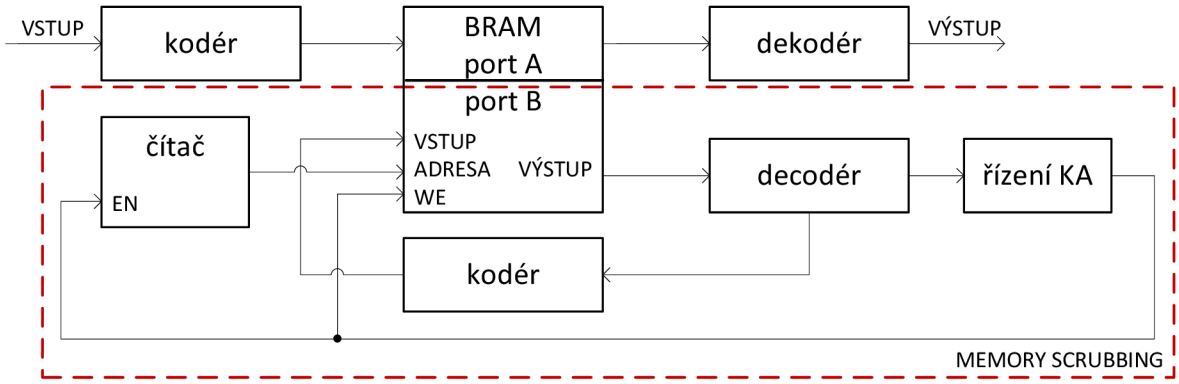
Obrázek 6.5: Deterministický *memory scrubbing*.

6.3.3 Bloková paměť BRAM

Bloková paměť je použita v jednotce pro uložení mapy. Je zde použita paměť o velikosti 8192 položek po čtyřech bitech, což je velikost jednoho bloku paměti BRAM. V případě, že by tato paměť byla zabezpečena pomocí TMR, bude potřeba použít třikrát více zdrojů, což znamená tři bloky. V případě použití duplikace s kódem pro detekci chyb bude nárůst použitých zdrojů více než dvojnásobný, bude způsoben především zdvojením paměti, ale také přidáním dalších obvodů pro kódování a dekódování dat. Co se týče nárůstu použitých zdrojů, vychází nejlépe použití kódu pro detekci a opravu chyb, který bude použit pro zabezpečení blokové paměti v jednotce pro uložení mapy.

Mimo použití opravného Hamingova kódu bude také vhodné zajistit opravu hodnot uložených v paměti pomocí *memory scrubbingu*. Zde nastává problém, jelikož oba porty dvouportové blokové paměti jsou již využity a to blokem pro aktualizaci mapy a blokem pro hledání cesty v bludišti. Jako řešení se nabízí vytvořit další obvod, který zajistí přístup obou jmenovaných bloků do paměti prostřednictvím jednoho portu. Bohužel to povede k menší rychlosti řídicí jednotky. Ale vzhledem k rychlosti pohybu a nutnosti čekání všech bloků řídicí jednotky na přesun robota je způsobené zmenšení rychlosti zanedbatelné. Tímto dojde k uvolnění jednoho portu blokové paměti, což umožní použít *memory scrubbing*. Dále se nabízí otázka, zda použít deterministickou či nedeterministickou variantu. Vzhledem k nevýhodě nedeterministické varianty se přikláním spíše k použití deterministické varianty. Bude tedy docházet k pravidelnému čtení a opravě hodnot uložených v paměti. Navrhované zabezpečení blokové paměti zachycuje obrázek 6.6, kde lze vidět jednak kodér a dekodér pro zápis a čtení dat, ale také pro potřeby obnovování správného obsahu

paměti. Dále je zde použit čítač pro postupné generování adres a konečný automat, který řídí *memory scrubbing* na základě informací z dekodéru, tedy informací o správnosti či nesprávnosti přečtených dat.



Obrázek 6.6: Zabezpečení paměti pro uložení mapy (BRAM) využívající Hammingův opravný kód a *memory scrubbing*.

6.3.4 LUT paměti

Paměť vytvořená z LUT tabulek je použita v jednotce pro ovládání mechanických částí robota (ECU). Zde je použita jako paměť typu FIFO, jedná se o poměrně malou paměť o velikosti 15 paměťových položek, kdy každá položka má šířku 32 bitů. Tato paměť tedy nezabírá příliš velké množství zdrojů, proto padlo rozhodnutí zajistit její odolnost proti poruchám pomocí techniky TMR v kombinaci s technikou *memory scrubbing* pro zajištění obnovy instance paměti, která je napadena poruchou. Memory scrubbing musí být použit v deterministické verzi, jelikož v nedeterministické verzi dochází ke kontrole pouze čtených dat. Oprava těchto dat je zbytečná, protože se jedná o paměť FIFO, tedy jednou přečtená data už vícekrát čtena nebudou.

Tato paměť je implementována pouze s jedním čtecím a zápisovým portem. Bude ji nutné rozšířit o druhý čtecí a zápisový port, který umožní použití *memory scrubbingu*. Po tomto rozšíření implementovaných pamětí bude princip zabezpečení shodný s principem, který představil výše uvedený obrázek 6.5.

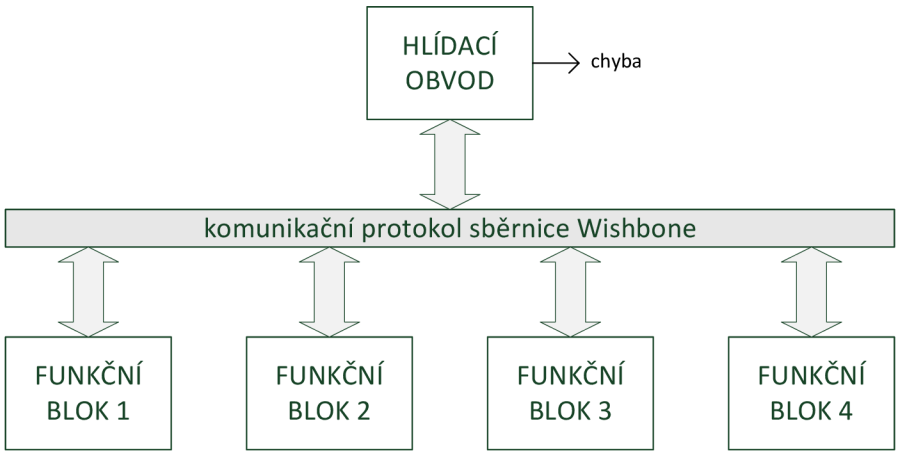
6.4 Sběrnice

Porucha sběrnice může vést k selhání celé řídicí jednotky, protože sběrnice je centrálním prvkem celého systému a slouží pro komunikaci jednotlivých funkčních bloků mezi sebou a mezi blokem rozhraní. Pro zabezpečení sběrnice se nabízí použití již několikrát zmiňované techniky TMR, tato technika ale přináší značný nárůst použitých zdrojů na FPGA a bude použita pro zabezpečení jiných částí řídicí jednotky. Proto padlo rozhodnutí zajistit odolnost sběrnice proti poruchám pomocí hlídacího obvodu.

Problematikou hlídacích obvodů se zabývá [26]. Je zde prezentována metodika pro zabezpečení sběrnice pomocí hlídacích obvodů a představen nástroj pro automatické generování hlídacího obvodu na základě popisu komunikačního protokolu pomocí speciálně vyvinutého jazyka. Vygenerovaný hlídací obvod je konečný automat, který kontroluje průběhy signálů na sběrnici. Použitý jazyk umožňuje kontrolovat nejen chyby protokolu, ale je také schopen popsat i některé poruchy na úrovni funkce. Jednou z možností je například kontrola rozsahu přenášených hodnot.

Hlídací obvod kontroluje, zda komunikace na sběrnici odpovídá danému komunikačnímu protokolu a jeho výstupem je signál informující okolí o výskytu poruchy. Tento signál může sloužit k zahájení částečné rekonfigurace systému, což povede k obnovení správné funkčnosti komunikační sběrnice.

Obrázek 6.7 ukazuje koncept propojení komunikační sběrnice, funkčních bloků a hlídacího obvodu. Je zde vidět zmíněný signál informující o abnormálním chování na sběrnici. Komunikační protokol bude popsán pomocí jazyka zmíněného v [26], což umožní vygenerovat příslušný hlídací obvod v jazyce VHDL, který bude zapojen do systému. Metodika byla ověřována na komunikačním rozhraní Local Link, ale nic nebrání jejímu použití k hlídání sběrnice Wishbone.



Obrázek 6.7: Kontrola komunikačního protokolu sběrnice hlídacím obvodem.

7 Závěr

V této práci byl čtenář seznámen s problematikou výskytu poruch v číslicových systémech, byla uvedena definice spolehlivosti a výčet základních spolehlivostních ukazatelů. Jsou představeny dva základní přístupy pro zajištění spolehlivosti, předcházení poruchám a odolnosti proti poruchám, přičemž hlavním tématem této práce je odolnost číslicových systémů proti poruchám. Své místo si v práci našla kapitola o obvodech FPGA, která se věnuje rozboru struktury obvodů FPGA a především poruchám, které mohou v FPGA nastat. Samostatná kapitola je věnována technikám zajištění odolnosti proti poruchám, která shrnuje základní pojmy z této oblasti a hlavní důraz klade na zajištění odolnosti proti poruchám u systémů založených na obvodech FPGA. Důležitou vlastností FPGA obvodů je možnost provedení částečné dynamické rekonfigurace, což přináší značné výhody při obnovování částí systémů napadených poruchou.

Jádrem této práce je návrh řídicí jednotky robota, který bude hledat cestu v bludišti. Jedná se o komplexní systém, který bude sloužit pro testování různých metodik zajištění odolnosti proti poruchám. Ve čtvrté kapitole jsem představil celkový koncept řídicí jednotky, včetně podrobného popisu jednotlivých bloků celého systému. Pátá kapitola se věnuje popisu implementace navržené řídicí jednotky, jsou zde rozebrány různé alternativy implementace dílčích částí a jsou zde uvedeny důvody pro výběr konkrétních alternativ. Zvolené principy a metody implementace jsou podrobně popsány a vysvětleny.

Práce pokračuje ověřením funkčnosti implementované řídicí jednotky, funkčnost je ověřena nejprve simulací v programu ModelSim. Následuje představení zapojení kompletní testovací platformy a popis jednotlivých částí, vývojové desky ML506, simulačního prostředí robota a propojení pomocí FITkitů. Implementovaná řídicí jednotka je syntetizovatelná do FPGA a na přiloženém CD je video ukazující pohyb robota v bludišti.

Poslední kapitola práce se věnuje výběru a představení metodik pro zajištění odolnosti proti poruchám. Kapitola je členěna do podkapitol dle typu obvodů, jednotlivé podkapitoly se věnují kombinačním a sekvenčním obvodům, sběrnicím a pamětem. Nejprve jsou uvedeny jednotlivé dílčí části odpovídající danému typu obvodu a dále je navržena metodika pro jejich zabezpečení. Jedná se pouze o konceptuální návrh zabezpečení, který je nutné před realizací rozšířit o implementační detaily.

7.1 Budoucí práce

Byl představen teoretický úvod do problematiky spolehlivosti číslicových systémů, především pak zajištění jejich odolnosti proti poruchám. Představen byl také základní koncept řídicí jednotky robota, který byl v dalších kapitolách detailně rozpracován. Poslední kapitola se věnuje výběru metodik pro zajištění odolnosti proti poruchám. V rámci další práce bych se nejprve rád věnoval rozšíření implementace, zejména implementaci pokročilejšího algoritmu pro hledání cesty v bludišti. Především bych rád provedl detailní návrh zabezpečení jednotlivých bloků proti poruchám a tato zabezpečení bych rád zakomponoval do implementace řídicí jednotky. K ověření kvality zabezpečení řídicí jednotky bude sloužit externí injektor poruch, který umožňuje zanesení poruchy do konkrétních částí řídicí jednotky, čímž umožní sledovat vliv těchto poruch jak na výstup řídicí jednotky, tak na samotného robota. Budeme moci vizuálně sledovat vliv poruch na pohyb robota.

Výsledkem celé práce bude volně dostupný framework pro ověřování a testování metodik pro zabezpečení odolnosti proti poruchám. Tomuto frameworku budou v blízké době vytvořeny webové stránky, jejichž cíl bude seznámení s testovací platformou a její propagace. Samozřejmostí bude možnost použití jakéhokoliv systému zapouzdřeného pomocí vytvořeného frameworku. Ideální situace nastane, když budou ověřovány různé metodiky na jednom číslicovém systému, což umožní přímé porovnávání kvalit jednotlivých metodik.

Literatura

[1] HLAVIČKA, J., et al. *Číslicové systémy odolné proti poruchám*. Praha: Vydavatelství ČVUT, 1992, 330 s. ISBN 80-01-00852-5.

[2] KASTENSMIDT, F. L., L. CARRO a R. REIS. *Fault-tolerance techniques for SRAM-based FPGAs*. Dordrecht: Springer, 2006, 183 s. ISBN 0-387-31068-1.

[3] ŠŤASTNÝ, J. *FPGA prakticky: realizace číslicových systémů pro programovatelná hradlová pole*. Praha: BEN - technická literatura, 2010, 199 s. ISBN 978-80-7300-261-9.

[4] XILINX. *FPGA* [online]. [cit. 2012-12-01]. Dostupné z: http://www.xilinx.com/fpga/index.htm

[5] KOREN, I. a C. M. KRISHNA. *Fault-tolerant systems*. San Francisco: Morgan Kaufmann Publishers, 2007, xix, 378 s. ISBN 978-0-12-088525-1.

[6] GEFFROY, J. C. a G. MOTTET. *Design of dependable computing systems*. Dordrecht: Kluwer Academic Publishers, 2002, xix, 672 s. ISBN 1-4020-0437-0.

[7] KASTENMIDT, F. L., G. NEUBERGER , L. CARRO a R. REIS. Designing and testing fault-tolerant techniques for SRAM-based FPGAs. In: *CF '04: Proceedings of the 1st conference on Computing frontiers*. New York, NY, USA: ACM, 2004, s. 419-32.

[8] STRAKA M., J. KAŠTIL, Z. KOTÁSEK a L. MIČULKA. Fault Tolerant System Design and SEU Injection Based Testing. *In: Microprocessors and Microsystems*. Amsterdam, NL: 2012, č. 01, s. 16. ISSN 0141-9331.

[9] ZBOŘIL, F. V. a F. ZBOŘIL. *Základy umělé inteligence: studijní opora*. Brno: Fakulta informačních technologií, 2006.

[10] VEJMOLA, S. *Chvála bludišť*. Praha: Státní pedagogické nakladatelství, 1991. ISBN 80-04-25235-4.

[11] XILINX. *AXI Reference Guide* [online]. 2011 [cit. 2013-03-02]. Dostupné z: http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

[12] OPENCORES. *Wishbone B4: WISHBONE System-on-Chip (SoC) Interconnection Architecture Portable IP Cores* [online]. 2010 [cit. 2013-02-25]. Dostupné z: http://cdn.opencores.org/downloads/wbspec_b4.pdf

[13] OPENCORES. *Wishbone builder: User's manual* [online]. [cit. 2013-02-27]. Dostupné z: http://opencores.org/websvn,filedetails?repname=wb_builder&path=%2Fwb_builder%2Fweb_uploads%2Fusers_manual.pdf

[14] HERVEILLE, R. *Combining WISHBONE interface signals: Application note* [online]. 2001 [cit. 2013-02-27]. Dostupné z: http://cdn.opencores.org/downloads/appnote_01.pdf

[15] KOLOUCH, J. Možnosti realizace pamětí RAM v obvodech. In: *Elektrorevue* [online]. 2003 [cit. 2013-03-25]. Dostupné z: http://www.elektrorevue.cz/clanky/03027/index.html

[16] XILINX. *IP Processor Block RAM* [online]. 2011 [cit. 2013-03-26]. Dostupné z: http://www.xilinx.com/support/documentation/ip_documentation/bram_block.pdf

[17] XILINX. *Virtex-5 Libraries Guide for HDL Designs* [online]. 2008 [cit. 2013-03-27]. Dostupné z: http://www.xilinx.com/itp/xilinx10/books/docs/virtex5_hdl/virtex5_hdl.pdf

[18] XILINX. *Virtex-5 FPGA User Guide* [online]. 2012 [cit. 2013-03-25]. Dostupné z: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[19] *ModelSim* [online]. [cit. 2012-01-04]. Dostupné z: http://model.com/

[20] XILINX. *XtremeDSP Development Platform — Virtex-5 FPGA ML506 Edition* [online]. [cit. 2013-01-04]. Dostupné z: http://www.xilinx.com/products/boards-and-kits/HW-V5-ML506-UNI-G.htm

[21] GERKEY, B., R. T. VAUGHAN a A. HOWARD. The player/stage project: Tools for multi-robot and distributed sensor systems. In: *Proceedings of the 11th International Conference on Advanced Robotics*. ser. ICAR '03. Coimbra (Portugal): 2003, s. 317–23.

[22] VAŠÍČEK, Z. *FITkit* [online]. [cit. 2013-04-25]. Dostupné z: http://merlin.fit.vutbr.cz/FITkit/

[23] XILINX. *Virtex-5 Family Overview*. 2009 [cit. 2013-03-25]. Dostupné z: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf

[24] FRIGERIO, L. a F. SALICE. RAM-based fault tolerant state machines for FPGAs. In: *Proceedings of the 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*. Washington, DC, USA: IEEE Computer Society, 2007, s. 312-20. ISBN 0-7695-2885-6.

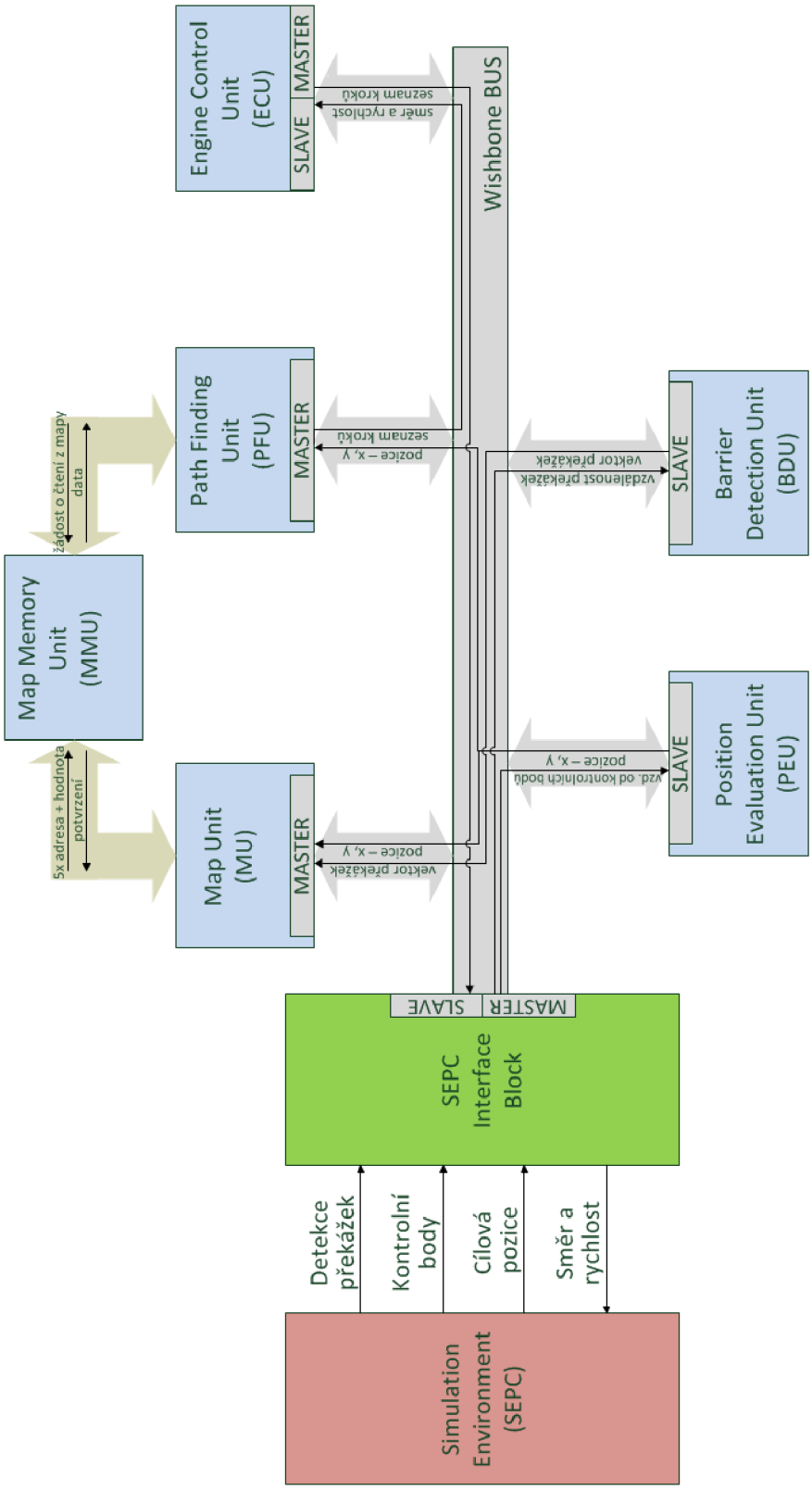
[25] ROLLINS, N., M. FULLER a M. J. WIRTHLIN. A Comparison of fault-tolerant memories in SRAM-based FPGAs. In: *Aerospace Conference, 2010 IEEE*. 2010, s. 1-12.

[26] STRAKA, M. Generátor hlídacích obvodů pro komunikační protokoly Xilinx. In: *Počítačové architektury a diagnostika 2007*. Plzeň (CZ): University of West Bohemia in Pilsen, 2007, s. 129-36. ISBN 978-80-7043-605-9.

Seznam příloh

Příloha A

Blokové schéma řídicí jednotky



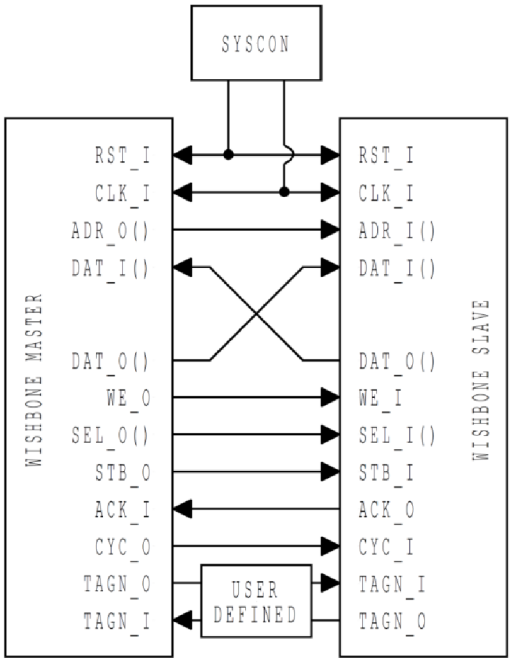
Příloha B

Sběrnice Wishbone

Tato příloha se věnuje popisu signálů přenášených přes sběrnici a také popisu časových průběhů jednotlivých přenášených signálů během sběrnicového cyklu.

B.1 Signály sběrnice

Obrázek B.1 ukazuje signály, které jsou přenášeny přes sběrnici, mimo adresy a přenášených dat jsou zde i řídicí signály. Všechny signály můžeme rozdělit do tří kategorií, první jsou signály společné pro master i slave rozhraní, druhou kategorií jsou signály specifické pro master rozhraní a třetí kategorií jsou signály specifické pro slave rozhraní.



Obrázek B.1: Přehled signálů přenášených přes sběrnici.

B.1.1 Signály master i slave zařízení

V této kapitole jsou představeny signály společné pro master i slave zařízení.

RST_I

Signál, který resetuje zařízení připojená ke sběrnici do výchozího stavu. V implementované řídicí jednotce je tento signál společný i pro reset zařízení, která nejsou přímo připojena ke sběrnici.

CLK_I

Hodinový signál pro všechna zařízení, která jsou připojena ke sběrnici. Všechny ostatní Wishbone signály jsou synchronní s náběžnou hranou hodinového signálu.

DAT_O()

Datový výstup jednotlivých zařízení. V případě implementované řídicí jednotky se jedná o vektor o šířce 16bitů.

DAT_I()

Datový vstup jednotlivých zařízení, parametry jsou stejné jako v případě datového výstupu.

TAGN_O a TAGN_I

Jedná se o doplňkové příznaky, které mohou být přenášeny po sběrnici. Význam těmto příznakům přiřazuje uživatel. V implementaci řídicí jednotky robota nejsou tyto příznaky využity.

B.1.2 Signály master zařízení

V této kapitole jsou představeny signály, které jsou dostupné pouze na rozhraní master zařízení kompatibilních se sběrnicí Wishbone.

ADDR_O()

Adresový výstup master zařízení. V implementované řídicí jednotce je adresa ve formě vektoru o velikosti 8 bitů, z toho 4 bity slouží pro adresaci zařízení a 4 bity pro adresaci registru v zařízení.

WE_O

Výstupní signál indikující, zda je požadován čtecí nebo zápisový sběrnicový cyklus. Během čtecího cyklu je signál v logické nule, zatímco během zápisového cyklu je nastaven do logické jedničky.

SEL_O()

Výstupní signál, který určuje, které bity z vektorů datových signálů jsou během probíhajícího sběrnicového cyklu platné. V implementaci řídicí jednotky není tento signál využíván a vždy jsou platné všechny bity.

STB_O

Výstupní signál indikující platný přenosový cyklus na sběrnici, v průběhu kterého jsou platné další signály, jako například SEL_O(). Na každé nastavení tohoto signálu musí odpovědět slave zařízení potvrzením signálem ACK_I.

ACK_I

Vstupní signál indikující potvrzení sběrnicového cyklu ze strany slave zařízení v případě, že je normálně ukončen. Specifikace umožňuje použít i jiné typy ukončení, jako je chyba nebo žádost o opakování, ale tyto nejsou v implementaci řídicí jednotky použity. Každý sběrnicový cyklus je tedy normálně ukončen.

CYC_O

Výstupní signál indikující, že právě probíhá sběrnicový cyklus. Tento signál je nastaven v průběhu všech přenosových cyklů, například během blokového přenosu, kdy probíhá více přenosových cyklů. V implementaci řídicí jednotky je zároveň použit jako žádost o přidělení sběrnice.

B.1.3 Signály slave zařízení

Přehled signálů, které jsou dostupné pouze na rozhraní slave zařízení kompatibilních se sběrnicí Wishbone je představen v této kapitole.

ADDR_I()

Adresový vstup slave zařízení. V implementaci je tento vstup ve formě vektoru o šířce 4 bitů, které slouží k adresaci registrů v zařízení.

WE_I

Vstupní signál indikující, zda se jedná o zápisový nebo čtecí sběrnicový cyklus.

SEL_I()

Vstupní signál, který má stejný význam, jako výstupní signál master zařízení SEL_O().

STB_I

Vstupní signál indikující, že dané slave zařízení je vybráno adresovým dekodérem a požaduje se po něm přenos. Slave zařízení smí odpovídat na jiné signály pouze, když je STB_I nastaveno.

ACK_O

Výstupní signál, pomocí kterého slave zařízení informuje o normálním ukončení přenosového cyklu.

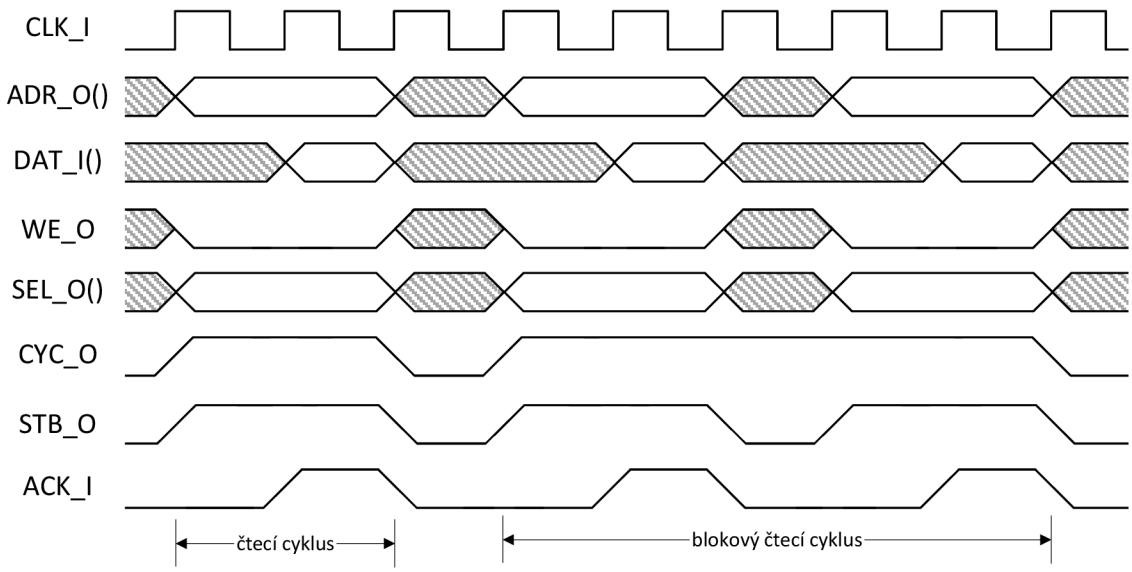
CYC_I

Vstupní signál informující o probíhajícím sběrnicovém cyklu, jeho význam je stejný jako u signálu CYC_O u master zařízení.

B.2 Sběrnicové cykly

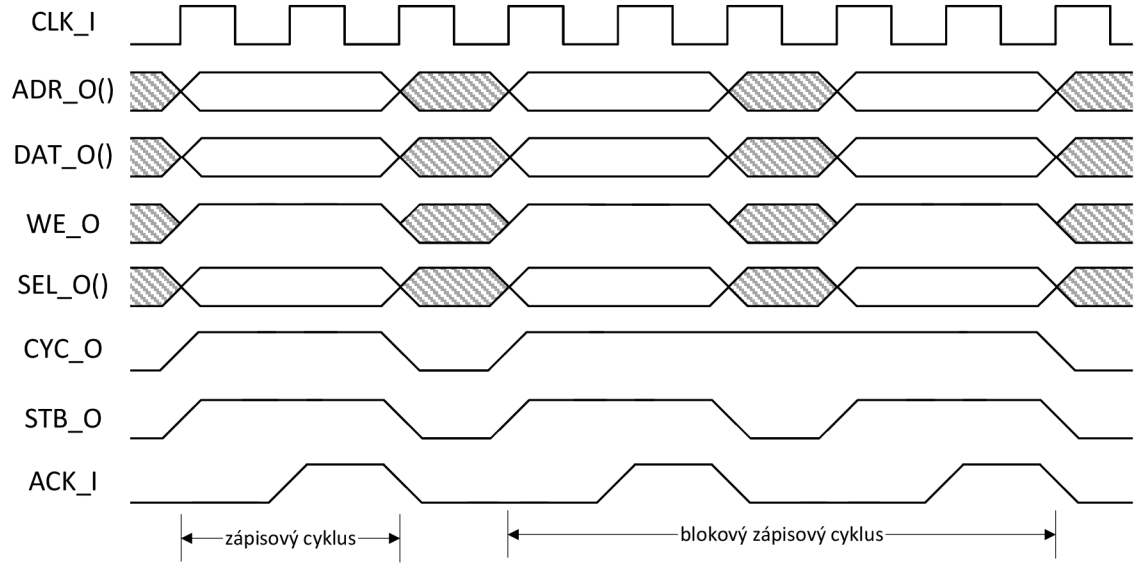
K přenosu dat po sběrnici dochází během *přenosového cyklu*, který může zahájit pouze master zařízení. Každý *sběrnicový cyklus* je zahájen nastavením signálů CYC_O a STB_O, signál CYC_O je nastaven po celou dobu sběrnicového cyklu, zatímco STB_O je nastaven pouze v průběhu přenosového cyklu. U jednoduchého sběrnicového cyklu (např. čtecí cyklus na obrázku B.3) jsou nastaveny oba signály současně, rozdíl je viditelný u blokového sběrnicového cyklu (např. blokový čtecí cyklus na obrázku B.3). Současně s nastavením signálu STB_O musí být na sběrnici vystavena platná adresa ADR_O(). Slave zařízení odpovídá nastavením signálu ACK_O.

Při čtecím cyklu (obrázek B.3) je signál WE_O nastaven do logické nuly. Aktivní signál ACK_I informuje master zařízení o tom, že na datovém vstupu DAT_I() jsou platná data.



Obrázek B.3: Ukázka jednoduchého a blokového čtecího sběrnicového cyklu.

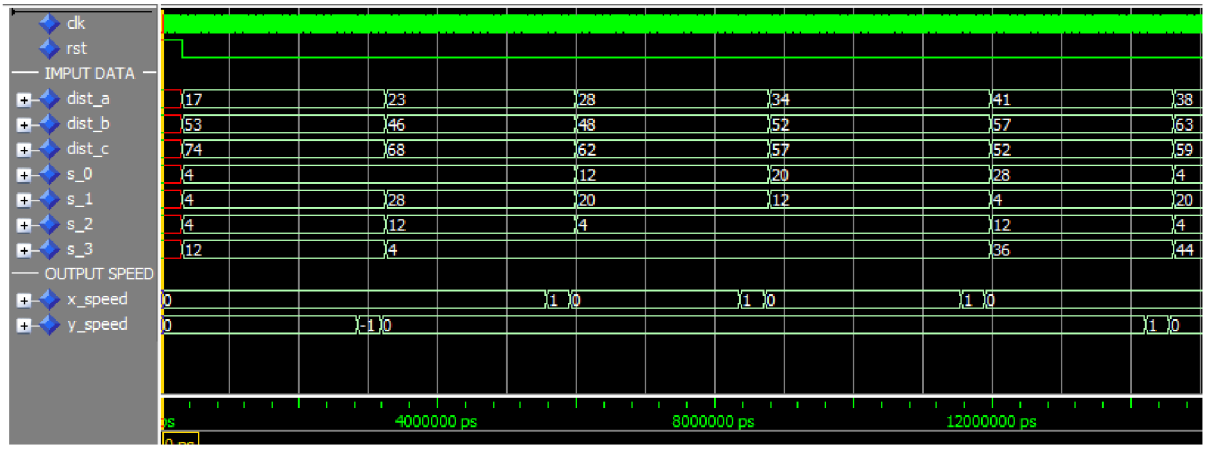
Zápisový cyklus znázorňuje obrázek B.4. Při zápisovém cyklu je signál WE_O nastaven do logické jedničky, čímž je indikováno, že se jedná právě o zápisový cyklus. Master zařízení současně vystaví na datový výstup platná data.

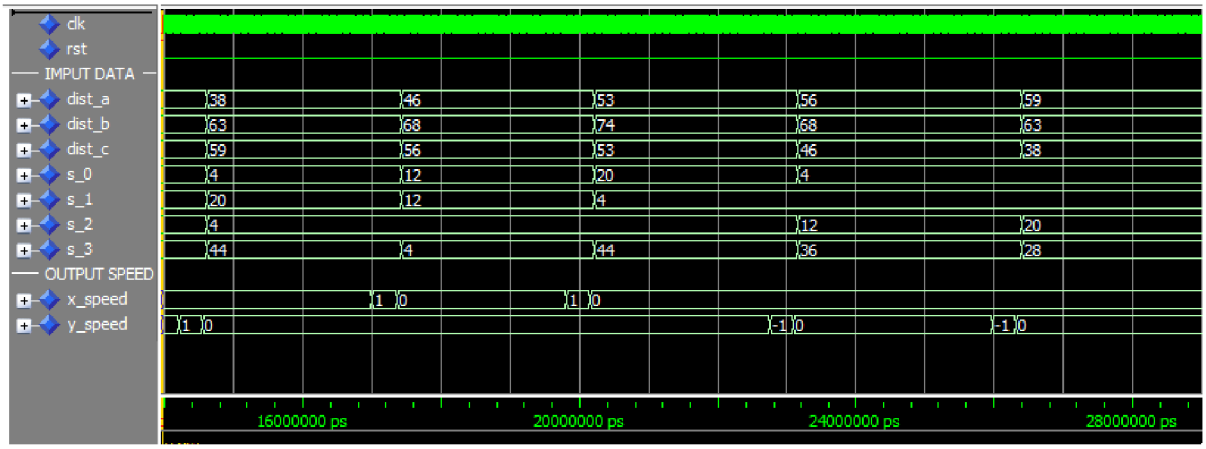


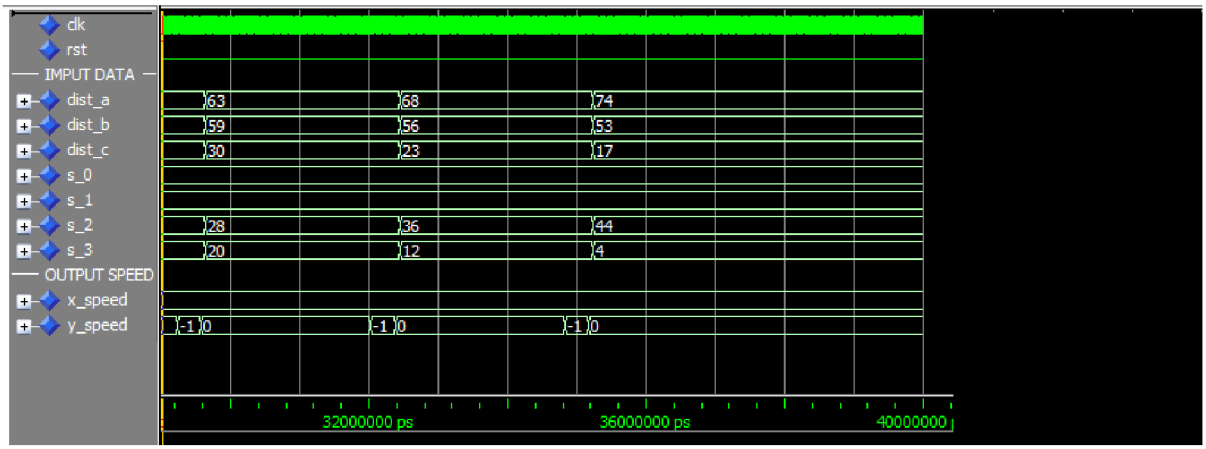
Obrázek B.4: Ukázka jednoduchého a blokového zápisového sběrnicového cyklu.

Příloha C

Časové průběhy vstupních a výstupních signálů při simulaci řídicí jednotky







Příloha D

Adresářová struktura přiloženého CD

- **implementace** – obsahuje zdrojové kódy řídicí jednotky
  - **bdu** –blok pro detekci překážek
  - **bdu_fsm** – řízení bloku pro detekci překážek
  - **ecu** – řízení mechanických částí robota
  - **mem_top_level** – paměťový subsystém pro účely simulace
  - **mmu** –paměť pro uložení mapy
  - **mu** – aktualizace mapy
  - **mu_fsm** – řízení aktualizace mapy
  - **peu** – blok pro výpočet pozice v mapě
  - **peu_fsm** – řízení bloku pro výpočet pozice
  - **robot_controller** – sestavená řídicí jednotka
    - robot_controller.vhd
    - sim_demo.fdo
    - simulation.fdo
    - testbench.vhd
  - **sif_fsm** – řízení rozhraní řídicí jednotky
  - **wishbone_bus** – zdrojové kódy sběrnice a rozhraní pro připojení ke sběrnici
- **zprava** – obsahuje tuto technickou zprávu v elektronické podobě
  - xpodiv01.docx
  - xpodiv01.pdf
- **video** – demonstrační video ukazující pohyb robota v bludišti
  - xpodiv01_video.mp4
  - xpodiv01_video.wmv