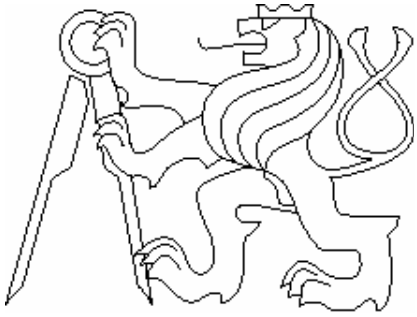
České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

Nástroj pro generování zabezpečovacího zařízení pro železnici

Bc. Pavel Vít

Vedoucí práce: Ing. Jaroslav Borecký

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný

Obor: Výpočetní technika

květen 2010

Poděkování

Rád bych chtěl poděkovat své rodině, přátelům, kteří mě podporovali a nerušili od práce, panu Ing. Jaroslavovi Boreckému za vedení práce a pečlivé konzultace. Také bych chtěl poděkovat kamarádům ze školy, kteří mě zásobovali dobrými radami během mé práce.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto díla ve smyslu §60 Zákona č.121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 14. 5. 2010 ……

Abstract

This work is about railway station generator. The programme generates VHDL file by user definitions of connections between five basic blocks needed to design any safety device of the railway station. Generated file can be imported to project of hardware design with prepared blocks. These blocks are used from previous diploma thesis and remade for better automatic generation.

We tried to create a simple editor to make a new railway station, which could be easy to use for simulation on FPGA or practical used.

This software was made in programming language Java because of multiplatform usability. The programme is suitable to create any configuration of railway station safety device on FPGA.

Abstrakt

Tato práce se zabývá tvorbou generátoru staničního zabezpečovacího zařízení železničního nádraží. Jde o program, který vytváří VHDL soubor, ve kterém jsou propojeny bloky staničního zabezpečovacího zařízení podle uživatelského návrhu. Vygenerovaný soubor stačí vložit do projektu s připravenými bloky. Ty byly použity z dřívější diplomové práce a přepracovány do podoby, která je vhodnější pro automatické generování.

Snahou bylo vytvoření jednoduchého editoru, pomocí kterého se snadno vytvoří nové nádraží, které lze následně nahrát do přípravku FPGA, kde je možné provádět další simulace, nebo jej použít pro praxi.

Program byl napsán v programovacím jazyku Java z důvodu přenositelnosti na jiné operační systémy. Software je vhodný pro tvorbu staničního zabezpečovacího zařízení založeného na pěti základních blocích napsaných pro FPGA.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Integrované obvody typu FPGA (Field Programmable Gate Array) jsou programovatelná hradlová pole, která obsahují pravidelnou strukturu bloků, které lze propojovat dle vlastního návrhu. Výhodou oproti zákaznickým integrovaným obvodům je možnost překonfigurování obvodu u zákazníka nebo přímo v přípravku.

Historie FPGA obvodů se řadí přibližně do 70. let 20. století, kdy se začaly vyrábět první programovatelné paměti typu PROM. Dalším vývojovým krokem byl vznik obvodů FPLA. Ty obsahovaly dvě programovatelné matice pro tvorbu součtových a součinových termů. Kvůli dosažení vyšší rychlosti byly vytvořeny obvody PLA, které obsahovaly pouze jednu programovatelnou matici pro součinové termy a pevnou matici pro součtové termy. Postupem času vznikaly mnohem složitější a komplexní obvody CPLD, které obsahovaly více PLD obvodů v jednom jádře. Přechod k FPGA už byl velice blízko a první čip byl vyroben firmou Xilinx v roce 1985 s názvem XC2064.

Hlavním cílem této práce je vytvoření editoru staničního zabezpečovacího systému pro železnice, který bude generovat VHDL soubor na základě popisu XML a bude použitelný v projektu s připravenými pěti základními bloky. Tyto stavební bloky byly připraveny pro pozdější zabezpečení bloků samých a byly sloučeny vstupy a výstupy do skupin podle jejich propojení mezi ostatními bloky. Konečná verze editoru bude generovat VHDL soubor pro libovolné nádraží a dovolí tak testování na modelech železnice a následně se počítá s nasazení výsledného produktu do provozu.

2 Vymezení cílů a požadavků

Práce je postavena na základech diplomové práce z roku 2008 studenta Martina Zatřepálka. V původní práci bylo vytvořeno zabezpečovací zařízení pro železniční stanici založené na FPGA. Autor přepracoval a sloučil původní reléové bloky do bloků napsaných v jazyce VHDL. Jako ukázkovou práci vytvořil jednoduché nádraží, které bylo ovládáno přes sériovou linku a výsledný stav se zobrazoval na LCD displeji. K realizaci byl použit přípravek Spartan 3E Starter kit.

Požadavky na projekt:

- Upravit stávající bloky staničního zabezpečovacího zařízení pro jednodušší zabezpečení jejich funkčnosti a online testování.
- Upravit vstupy a výstupy bloků pro přehledné generování libovolného staničního zabezpečovacího zařízení.
- Vytvořit skupiny vodičů podle jejich propojení mezi bloky. Rozdělit vstupní a výstupní vodiče do těchto skupin.
- Vytvořit definici XML souboru použitelného k popisu nádraží s možností budoucího rozšíření a doplnění.
- Vytvořit generátor VHDL kódu z připraveného XML souboru, zajistit kontrolu a správné propojení bloků.
- Zjednodušit tvorbu zdrojového XML souboru pomocí grafické aplikace.
- Připravit několik ukázkových nádraží vygenerovaných z XML do VHDL.
- Otestovat vytvořené bloky a funkčnost nádraží.

3 Základní bloky

Účelem této práce je zjednodušení budoucího přechodu železničních stanic od reléových obvodů k programovatelným obvodům FPGA. Jejich výhoda je v dnešní době veliká. Celé zapojení staničního zabezpečovacího zařízení pro nádraží lze realizovat pomocí obvodů na jednom čipu FPGA. Oproti stávajícímu zařízení, které zabírá většinou celou místnost a někdy to i nestačí, je to velký posun vpřed. Energetická spotřeba těchto obrovských zařízení je řádově odlišná od spotřeby čipů FPGA. Určitě také nesmíme opomenout, že reléové bloky jsou složeny z pohyblivých mechanických částí, které podléhají opotřebení a tím se zvyšuje poruchovost systému.

Pro přepis chování reléových bloků do podoby vhodné pro FPGA lze využít jazyk VHDL. VHDL je zkratka anglických slov VHSIC Hardware Description Language, jedná se o jazyk popisující hardware. Zkratka VHSIC značí Very High Speed Integrated Circuits, což znamená velmi rychlé integrované obvody. Jazyk může být použit pro tři různé metody popisu systému. Pro nás je důležitá metoda pro popis chování systému. Aby byl systém jednoduše srozumitelný, bývá často rozdělen do mnoha částí (bloků). V našem případě mohou bloky kolejiště korespondovat s bloky v popisu hardware.

3.1 Analýza stávajících bloků

Bloky byly vytvořeny po inspiraci v staničním reléovém zabezpečovacím zařízení. Jeho dokumentace byla velmi špatná a bloky musely být vytvořeny sloučením menších bloků původního zařízení. To dovolilo zjednodušení bloků, snížení počtu vstupů a výstupů a vynechání některých zpožďovacích členů. Využitím FPGA došlo k veliké úspoře místa oproti reléovým blokům.

Bloky jsou popsány v jazyku VHDL a jsou určeny k vytvoření jednoduchého staničního zabezpečovacího zařízení na jednom FPGA. Původně byly z bloků vytvořeny schematické značky. Propojení bloků bylo realizováno pomocí těchto značek. Jednotlivé sběrnice a vodiče musely být umístěny na správné vstupy a výstupy. To znamená, že uživatel musel být dostatečně seznámen s významem jednotlivých signálů a musel vědět, kam který signál napojit.

V našem řešení rozdělení bloků koresponduje s rozložením kolejových částí železničního nádraží. Toto rozdělení odpovídá jednoduchému nádraží. Základní kolejové bloky bylo nutno přepracovat do podoby vhodné pro zabezpečování a pro automatické generování vlastního návrhu nádraží.

Rozdělení bloků:

- **VJZD**
  - Blok je určen k vjezdu vlaků do nádraží a je napojen na vjezdové návěstidlo a jeho předzvěst, kde zobrazuje povolení k vjezdu a maximální rychlost jízdy.
  - Blok umožňuje stavění cesty. Lze z něho nebo do něho postavit vlakovou cestu. Cestu vytvořenou z tohoto bloku lze pouze tímto blokem zrušit.

- **M**
  - Blok kontroluje obsazenost úseku koleje, který mu přísluší.
  - Dohlíží na správnost směru jízdy vlaku.

- **SD**
  - Blok znázorňuje výhybku, kontroluje její nastavení směru jízdy a dorazů kolejiště. Generuje příkazy pro přestavění výhybky.
  - Dohlíží nad odvratem výhybky, pokud za ní následuje jiná výhybka.
  - Kontroluje směr jízdy.

- **HQ**
  - Blok ovládá hlavní odjezdové návěstidlo.
  - Tento blok je určen ke stavění cesty. Také lze z něj nebo do něj postavit a zrušit vlakovou cestu stejně jako u bloku VJZD.

- **K**
  - Blok kontroluje obsazenost úseku koleje.

- Dohlíží na směr jízdy vlaku a vlak zde může ukončit jízdu.
- Vlak zde může zastavit a změnit směr jízdy.

3.2 Upravení bloků

Hlavním důvodem úpravy funkčních bloků byla budoucí potřeba je zabezpečit před poruchami. Kontrolovat správnost funkce celého bloku by bylo velmi složité, až možná nereálné. Funkční bloky se skládají ze tří částí.

Logické výrazy určují zobrazování rychlosti na hlavním návěstidle a na předzvěsti. Skládají se jen z podmíněných výrazů, které jsou reprezentovány logickými hradly.

Stavový automat je hlavní částí bloku, dohlíží na správnou činnost bloku a zamezuje vzniku zakázaných stavů. Dohlíží zároveň nad správným vybavováním vlaku (správném směru jízdy). Vlak nesmí zastavit a otočit směr jízdy na blocích, které pro to nejsou určeny. V opačném případě by mohlo dojít k porušení bezpečnosti provozu. Vlak musí být neprodleně zastaven a vše musí být uvedeno do bezpečného stavu. Tyto stavové automaty plní funkci původních reléových bloků. Stavové automaty předávají informace ostatním stavovým automatům v okolních blocích.

Poslední častí bloku je čítač, který hlídá čas v případě rušení cest. Čítač se skládá z procesu předděličky, která je nastavena na hodnotu 50 000 000 pro hodinovou frekvenci 50 MHz kvůli krystalu, který je na desce s čipem Spartan 3E. Předdělička generuje signály o délce jedné sekundy. Ty vstupují do procesu *citac,* který signalizuje na výstupu entity ukončení čítaného času podle nastaveného vstupu ze stavového automatu.

Některé bloky neobsahují všechny tři části, ale pouze podmnožinu z nich. Například bloky M a K se skládají pouze ze stavového automatu, v tomto směru nebyla práce při rozdělování bloku složitá, šlo spíše jen o kontrolu a přípravu pro další upravení bloku. Naopak bloky VJZD a HQ obsahují všechny části a blok SD dvě (stavový automat a logické výrazy).

3.2.1 Bloky VJZD a HQ

Tyto bloky jsou uvedeny společně, protože práce při jejich rozdělování byla podobná. Bloky byly rozděleny do čtyř souborů.

Základními stavebními bloky jsou entity *automat*, *logika* a *čítač*.

Každá entita je uložena ve vlastním VHDL souboru, kde je definována samostatně. Tím je zjednodušeno nejen zabezpečení celého obvodu, ale i hledání případných chyb a poruch, které by mohly nastat.

Původně byly tyto stavební bloky v jedné entitě a nebyl problém je propojit signálem. Rozdělené stavební bloky musí navíc obsahovat vstupní a výstupní signály, aby mohly být propojeny tak, jak byly v původní entitě. Automat sděluje čítači, počet sekund kolik musí odměřit. Nazpět mu čítač vrací signálem informaci o tom, zda byl čas odměřen. Tyto entity musely být propojeny dvouvodičovou sběrnicí od automatu k čítači a jedním vodičem od čítače k automatu. Proto k automatu a čítači přibyly vstupní a výstupní signály.

Logické výrazy pro ovládání hlavního návěstidla a předzvěsti také potřebují získávat data od stavového automatu. Zde stačí jen jednosměrný jednobitový signál od automatu k logice.

Nejvyšší modul nese ve jménu souboru název bloku a příponu TOP. Soubor definuje vstupní a výstupní signály celého nejvyššího modulu. Ty musely být zachovány jako u původních bloků. Tento nejvyšší modul musí ve výsledku vykonávat funkci a chovat se jako původní blok. Proto byly do tohoto souboru importovány všechny tři předchozí komponenty (čítač, stavový automat a logické výrazy). Ty musely být následně propojeny již zmíněnými signály tak, aby mezi sebou komunikovaly všechny tři části bloku.

3.2.2 Blok SD

Výhybkový blok obsahuje nejsložitější stavový automat a nejsložitější logický obvod oproti ostatním blokům. To vyplývá i z toho, že tento blok je jediný, do kterého vedou tři koleje. Musí tedy dohlížet na příchozí signály od bloků z více stran a porovnávat je se stavem dorazu výhybky. Z tohoto bloku nelze postavit ani zrušit cestu, takže blok neobsahuje čítač.

Logický výraz u tohoto bloku je složitější a závisí na signálu z kolejiště. Podle toho, jestli je výhybka rovně nebo šikmo, tak blok posílá signály o povolené rychlosti

k nejbližším návěstidlům. Logický výraz není propojen se stavovým automatem žádným signálem. Blok je tedy složen ze tří souborů, kde hlavní modul s příponou TOP posílá signály do příslušných stavebních bloků. Některé vstupní signály jsou posílány do obou částí zároveň.

3.2.3 Bloky M a K

Tyto bloky jsou složeny pouze z logického automatu. Do funkce těchto členů nebylo zasahováno. Bylo jen otestováno, zda jsou jejich vlastnosti takové, jak uvádí autor a zda se v sestaveném staničním zabezpečovacím zařízení chovají správně. Bloky K a M jsou složeny jen z jednoho souboru. Vkládat logický automat do nějakého TOP modulu by bylo zbytečné. Entity M_automat a K_automat jsou tedy jak entitami automatu, tak i TOP modulem těchto bloků.

3.3 Skupiny vodičů

Pro zjednodušení práce při propojování bloků mezi sebou a pro přehlednost jsem použil strukturu balíčků a záznamů - record ve VHDL. Shluky vodičů velice zjednodušili práci při ručním psaní kódu VHDL a také hlavně při automatickém generování top modulu pro celé nádraží.

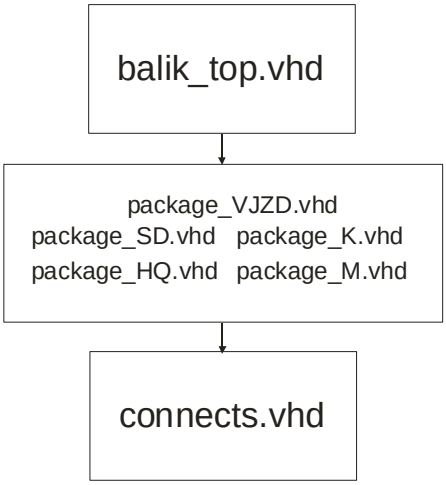
Záznamy record s vodiči byly rozděleny do tří skupin balíčků. Balíček s propojením mezi bloky *(connects.vhd)*, jednotlivé balíčky bloků *(package_blok.vhd)* a nejvyšší balíček se všemi bloky *(balicek_TOP.vhd)*.

Soubor *connects.vhd* obsahuje balíček *connects* se všemi shluky vodičů mezi bloky. Každý record, který spojuje dva bloky je pojmenován jejich názvy oddělenými podtržítkem. Na příklad „*SD_VJZD*“. Dále existují shluky vodičů, které vedou do kolejiště nebo do ovládacího pultu. Jejich název je tvořen jménem bloku a doplněním slova „*rail*“ nebo „*remote*“ a určením, zda jde o vstup nebo o výstup. Například „*HQ_remote_out*“.

Nadřazené balíčky souboru *connects.vhd* jsou balíčky jednotlivých bloků *package_blok.vhd*. Těchto balíčků je pět, pro každý blok jeden. Obsahují vždy pouze dva záznamy record a to vstupy a výstupy. Signál vstupu je značen předponou „*from*“, za kterou následuje jméno bloku, ze kterého signál vychází. Na příklad „*from_M*“.

Výstupní signály jsou řešeny obdobným způsobem ovšem obsahují předponu „*to*“. Těmto signálům je přiřazen vždy jeden záznam z balíčku nižší vrstvy, tj. balíček *connects*.

Nejvyšším balíčkem je *package_TOP* v souboru *balik_top.vhd*, který obsahuje záznamy record s názvy všech bloků. Záznam obsahuje jen celé shluky vstupů a výstupů a lze k nim přistupovat jako k celku. Tato struktura balíčku a záznamů record je stromového charakteru, kde listy jsou shluky vodičů mezi jednotlivými bloky ze souboru *connects.vhd* a kořenem je nejvyšší balíček *package_TOP*. Struktura souborů je zobrazena na obrázku č. 1.



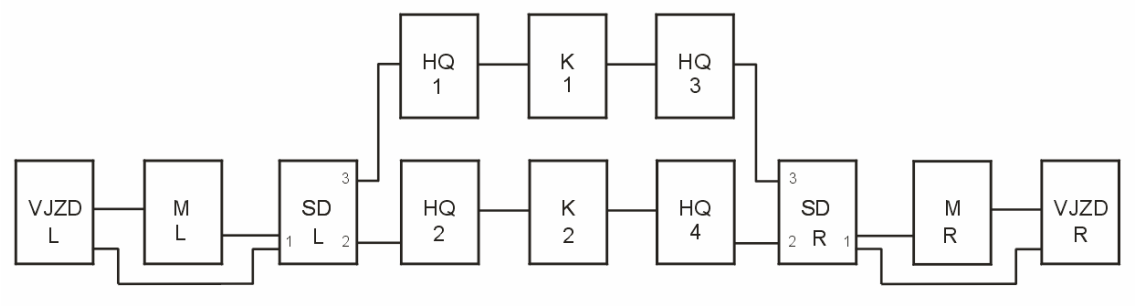
Obrázek č. 1: Struktura balíčků

3.4 Propojování bloků

Všechny upravené bloky měly zprvu stejné vstupy a výstupy jako bloky původní. To sice muselo být zachováno i nadále, ale signály se pro zjednodušení skryly pod skupiny vodičů. Skupiny pro vstupy byly u jednotlivých bloků vytvořeny podle toho, jak byl blok zapojen v základním jednoduchém nádraží se dvěma kolejemi. Zapojení nádraží je zobrazeno na obrázku č. 2. Na tyto vstupy se připojují i jiné bloky, které mohou být k danému bloku připojené v jiné variantě staničního zabezpečovacího zařízení. Tento způsob zapojení vstupů je jednodušší, protože se v kódu bloků nemusí řešit problém více vstupů na jeden signál. V případě chybného zapojení by funkce bloku nebyla vůbec jasná.

Skupiny vodičů pro výstupy byly vytvořeny podle způsobu zapojení bloků. Každý blok obsahuje větší množství výstupů, než by bylo bezprostředně nutné. Některé

výstupy se totiž použijí pro jednu variantu propojení bloků, jiné pro jinou variantu. Výstupní skupiny vodičů jsou vždy značeny tak, aby bylo jasné, ke kterému bloku se mají připojit. Tyto signály musí být stejného typu, jako jsou na vstupu připojovaného bloku. Skupiny výstupů často obsahují stejné nebo jen lehce rozšířené signály, ale vždy tak, aby se správně propojily s daným blokem.



Obrázek č. 2: Propojení bloků základního nádraží

Při využití skupin vodičů se propojení bloků velice zjednodušilo. Před vlastním použitím skupin vodičů ve VHDL musíme vložit do hlavičky hlavního souboru text *use work.package_TOP.all*, aby překladač věděl, že má tento balíček načíst. V něm nalezne seznam všech bloků a propojení. Pak už stačí přidat pouze jeden signál pro každý blok a všechny další vodiče jsou skryty pod tímto signálem.

Přistupovat k podskupinám a signálům na nižší úrovni můžeme pomocí teček od hlavního signálu tak, jak je vytvořena struktura balíčků. Tímto způsobem může napojit bloky jiným nestandardním způsobem, nebo nastavit na některá místa trvalé logické jedničky nebo logické nuly tak, aby nádraží mělo požadované vlastnosti. Pro běžné využití tento zásah nebude nutný.

Ukázka vytvoření signálu pro blok M:

```
signal M2_sig:       M;
```

Ukázka připojení bloku M:

```
M1: M_automat PORT MAP(
          from_VJZD => VJZD1_sig.vystupy.to_M,
          from_SD => SD1_sig.vystupy.to_M,
          from_rail => M1_sig.vstupy.from_rail,
```

```
        to_VJZD => M1_sig.vystupy.to_VJZD,
        to_SD => M1_sig.vystupy.to_SD,
        to_HQ => open,
        to_remote => M1_sig.vystupy.to_remote,
        CLK => CLK,
        RESET => RESET
    );
```

Ukázka připojení vstupu a výstupu bloku M:

```
M2_sig.vstupy.from_rail.K_Obsazeno <= M2in;
M2out_remote <= M2_sig.vystupy.to_remote;
```

3.4.1 Propojení bloku VJZD

Blok VJZD je určen pro vjezd vlaků do nádraží. Podle základního nádraží byl tento blok napojen pouze na blok M a SD. Z toho důvodu jsou jeho vstupy jen *from_M* a *from_SD*. Později bylo zjištěno, že mezi bloky M a SD může být přidán blok HQ, který by mohl blokovat vjezd vlaku do výhybky.

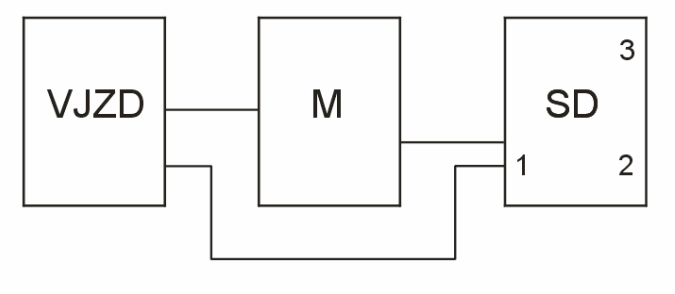
Připojení bloku M a SD:

Blok M má výstup *to_VJZD*, který se přímo připojí k bloku VJZD na vstup *from_M*. Podobným způsobem se jednoduše připojí i blok SD, který má opět výstup *to_VJZD*. Všechny výstupy bloku VJZD se připojí na signál tohoto bloku, jen některé z nich již nebudou dále využity. Propojení je ukázáno na obrázku č. 3.

Zápis ve VHDL:

```
        from_M => M2_sig.vystupy.to_VJZD,
        from_SD => SD3_sig.vystupy.to_VJZD,
```

Blokové schéma:



Obrázek č. 3: Propojení bloku VJZD s M a SD

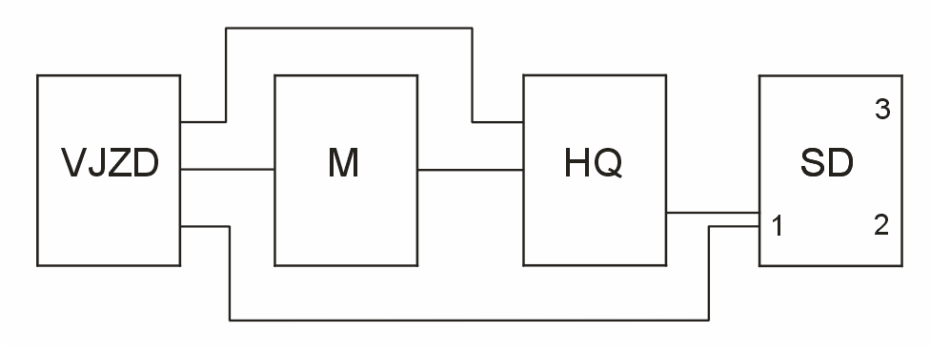
Připojení bloku M, HQ a SD:

Blok M se připojí stejně jako v předchozí variantě. Blok HQ má opět výstup *to_VJZD*, který se připojí na vstup *from_SD* bloku VJZD. Všechny výstupy bloku VJZD se opět připojí. V zápisu VHDL je ukázáno, že signál mezi VJZD a SD je připojen a vede jen jedním směrem od bloku VJZD k bloku SD. Blokové zapojení je na obrázku č. 4.

Zápis ve VHDL:

```
from_M => M2_sig.vystupy.to_VJZD,
from_SD => HQ3_sig.vystupy.to_VJZD,
to_SD => VJZD1_sig.vystupy.to_SD
```

Blokové schéma:



Obrázek č. 4: Propojení bloku VJZD s M, HQ a SD

3.4.2 Propojení bloku M

Blok M je určen pro příjezd vlaků z bloku VJZD a hlídá obsazenost úseku. Jeho vstupy podle základního nádraží jsou *from_VJZD* a *from_SD*. Existují opět dvě varianty, jak lze blok zapojit.

Připojení bloku VJZD a SD:

Blok VJZD má výstup *to_M*, který se přímo připojí k bloku M na vstup *from_VJZD*. Podobným způsobem se jednoduše připojí i blok SD, který má opět výstup *to_M*. Všechny výstupy bloku M se připojí na signál tohoto bloku, jen některé z nich již nebudou dále využity.

Zápis ve VHDL:

```
from_VJZD => VJZD1_sig.vystupy.to_M,
from_SD => SD3_sig.vystupy.to_M,
```

Blokové schéma:

Stejné jako u obrázku č. 3 u zapojení bloku VJZD.

Připojení bloku VJZD a HQ:

Blok VJZD se připojí stejně jako v předchozí variantě. Blok HQ má opět výstup *to_M*, který se připojí na vstup *from_SD* bloku M. Výstupní signál pro blok SD může být připojen nebo stačí k výstupu přiřadit text *open* a překladač bude vědět, že signál nemá být zapojen.

Zápis ve VHDL:

```
from_VJZD => VJZD1_sig.vystupy.to_M,
from_SD => HQ3_sig.vystupy.to_M,
to_SD => M2_sig.vystupy.to_SD
```

Blokové schéma:

Stejné jako u obrázku č. 4 u zapojení bloku VJZD.

3.4.3 Propojení bloku SD

Blok SD značí výhybku a v základním nádraží byl připojen na portu 1 k blokům VJZD a M. Na obou portech 2 a 3 ke dvěma blokům HQ, kde vzdálenější blok HQ (nextHQ) informoval blok SD o nastavení návěstidla a jeho předzvěsti.

Ve složitějších nádražích byly zapojeny výhybky za sebou a to z libovolného do libovolného portu. Také zapojení dvou bloků HQ bylo realizováno i do portu 1. Vjezd z bloků VJZD a M byl povolen pouze z portu 1. Do ostatních portů tyto bloky připojit nejde.

Připojení bloku HQ a nextHQ:

Bloky HQ mají výstupy *to_SD* a *to_nextSD*, které se přímo připojí k bloku SD na vstup *from_HQ* a *from_nextHQ*. U všech těchto vstupních signálů je číslo portu, odkud signál vstupuje. Například *from_nextHQ2*. Bloky HQ mohou být zapojeny tímto způsobem do portu 2 a 3. Výstup z bloku SD vede jen do bližšího bloku HQ. Schéma zapojení je na obrázku č. 5.

V případě jako je tento, kdy na vstup bloku SD není připojena další výhybka se musí nastavit na vstup ťapání bloku SD na logickou jedničku.

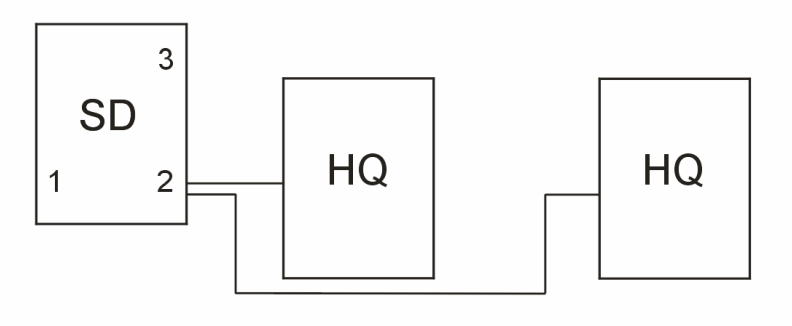
Zápis ve VHDL:

```
from_HQ2 => HQ5_sig.vystupy.to_SD,
from_nextHQ2 => HQ7_sig.vystupy.to_nextSD,
to_HQ2 => SD4_sig.vystupy.to_HQ2,
```

Nastavení ťapání:

```
SD8_sig.vstupy.from_tapani.B_tapani2 <= '1';
```

Blokové schéma:



Obrázek č. 5: Propojení bloku SD s HQ a nextHQ

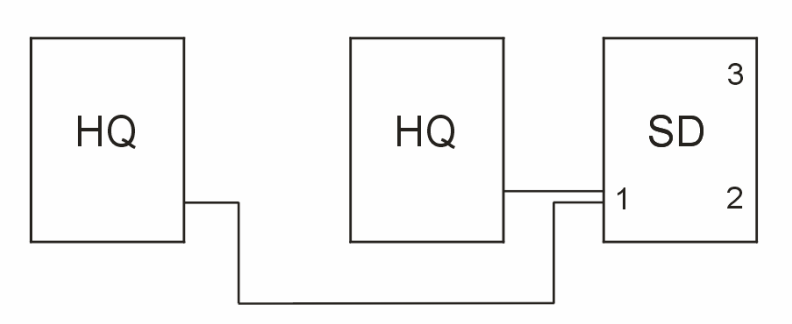
Připojení bloku HQ a nextHQ na port 1:

Blok HQ má pro tento případ výstup *to_SD1* a *to_nextSD1*, který se přímo připojí k bloku SD na port 1 na vstup *from_M* a *from_VJZD*. Bližší blok HQ se připojuje místo bloku M a vzdálenější místo bloku VJZD. Tomu odpovídá i propojení signálů. Z bloku SD vede opět jen do bližšího bloku HQ signál *to_HQ1*. Ani v tomto případě, jak zobrazuje obrázek č. 6, není připojena na port 1 další výhybka, proto se musí nastavit ťapání na logickou jedničku tak, jak bylo ukázáno v předchozím případě.

Zápis ve VHDL:

```
from_VJZD => HQ15_sig.vystupy.to_nextSD1,
from_M => HQ13_sig.vystupy.to_SD1,
to_HQ1 => SD12_sig.vystupy.to_HQ1,
```

Blokové schéma:



Obrázek č. 6: Propojení bloku SD s HQ a nextHQ na portu 1

Připojení bloku M, HQ a VJZD na port 1:

Připojení bloků VJZD a M, případně ještě bloku HQ bylo již demonstrováno na obrázcích č. 3 a č. 4. Tyto bloky lze připojit pouze do portu 1. Pro bloky VJZD a M jsou připraveny vstupy na bloku SD *from_VJZD* a *from_M*. Jejich připojení je jednoduché. Připojení bloku HQ je realizováno místo bloku M. Blok se v tomto případě do bloku SD vůbec nezapojí. Signál z bloku VJZD zůstává zapojen stejně. Výstupy bloku SD jsou připojeny všechny. Bloky M nebo HQ si vyberou správný z nich a připojí ho. Ani v tomto případě nesmíme zapomenout nastavit ťapání na logickou jedničku.

Zápis ve VHDL:

```
from_VJZD => VJZD1_sig.vystupy.to_SD,
from_M => HQ3_sig.vystupy.to_SD1,
```

Blokové schéma:

Na obrázcích č. 3 a č. 4.

Připojení bloku SD:

Blok SD má pro tento případ výstup *to_SD2HQ* a *to_SD2nHQ*, které slouží k připojení bloku SD na daný port. Tyto výstupy jsou na každém ze tří portů a jsou rozlišeny číslem portu. Vstupy bloku SD jsou *from_HQ* a *from_nextHQ*, do kterých se připojují výstupy z jiných bloků SD. Porty bloků lze libovolně kombinovat. To znamená, že na každý port lze připojit libovolný port jiného bloku SD.

Při propojení portů 1 a 2 (nebo 3) dochází k propojení výhybek bez odvratu. Na obrázku č. 7 je to propojení levého bloku s pravým horním. To znamená, že signál *B_odvraceno* musí být nastaven na logickou jedničku, signál *B_odvrat* na logickou nulu a signál ťapání také na logickou nulu. V dalším příkladu si ukážeme propojení portů 2 nebo 3 navzájem.

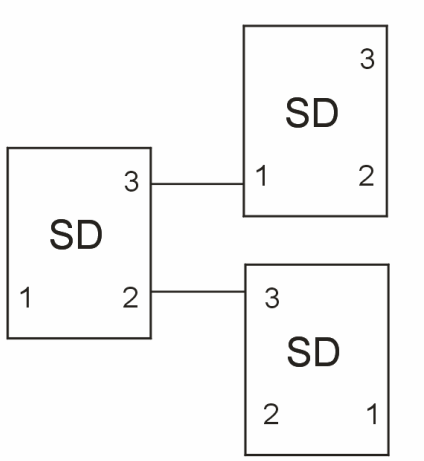
Zápis ve VHDL:

```
from_HQ2 => SD12_sig.vystupy.to_SD3HQ,
from_nextHQ2 => SD12_sig.vystupy.to_SD3nHQ,
from_HQ3 => SD8_sig.vystupy.to_SD1HQ,
from_nextHQ3 => SD8_sig.vystupy.to_SD1nHQ,
to_SD2HQ => SD4_sig.vystupy.to_SD2HQ,
to_SD2nHQ => SD4_sig.vystupy.to_SD2nHQ,
to_SD3HQ => SD4_sig.vystupy.to_SD3HQ,
to_SD3nHQ => SD4_sig.vystupy.to_SD3nHQ,
```

Nastavení odvratu a ťapání:

```
SD4_sig.vstupy.from_odvrat3.B_odvrat <= '0';
SD4_sig.vstupy.from_odvrat3.B_odvraceno <= '1';
SD4_sig.vstupy.from_tapani.B_tapani3 <= '0';
```

Blokové schéma:



Obrázek č. 7: Propojení bloku SD s SD

Připojení bloku SD s odvratem:

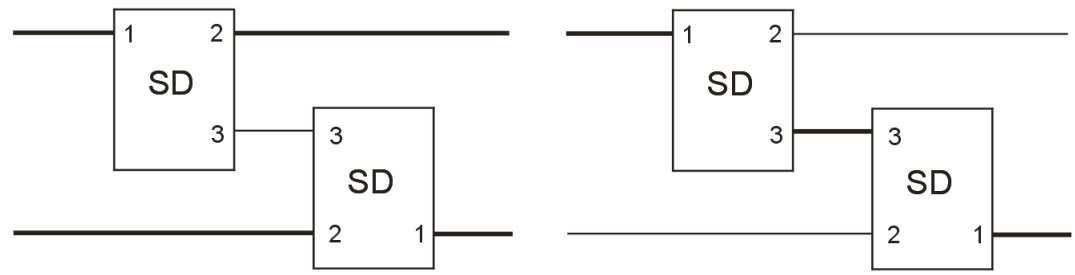
Připojení bloku s odvratem nastává při napojení portů 2 nebo 3 také na porty 2 nebo 3 tak, jak je propojen levý blok s pravým dolním blokem na obrázku č.7. Bloky se musí navzájem propojit skupinou signálů odvratu. Tím se zajistí, že se nemohou navzájem ohrožovat dvě koleje spojené výhybkami. Obě výhybky musí být buď ve směru šikmo,

aby vytvořili cestu, nebo rovně, aby z jedné koleje nemohl vjet vlak do druhé s uzavřenou výhybkou. Obě varianty jsou na obrázku č. 8. Levý obrázek ukazuje dvě cesty a vpravo na obrázku č. 8 je cesta šikmo. Připojení signálů odvratu slouží k doplnění klasického propojení dvou bloků SD.

Zápis ve VHDL:

```
SD8_sig.vstupy.from_odvrat3 <=
                    SD12_sig.vystupy.to_odvrat2;
```

Grafické znázornění:



Obrázek č. 8: Propojení bloku SD s blokem SD a odvratem

3.4.4 Propojení bloku HQ

Blok znázorňuje návěstidlo. Připojuje se vždy z jedné strany k výhybce a z druhé typicky k bloku K a další výhybce (nextSD). Výjimečně může být zapojen do vjezdového návěstidla a bloku M. Z původního jednoduchého nádraží z obrázku č. 2 má blok vstupy *from_K* a *from_SD.*

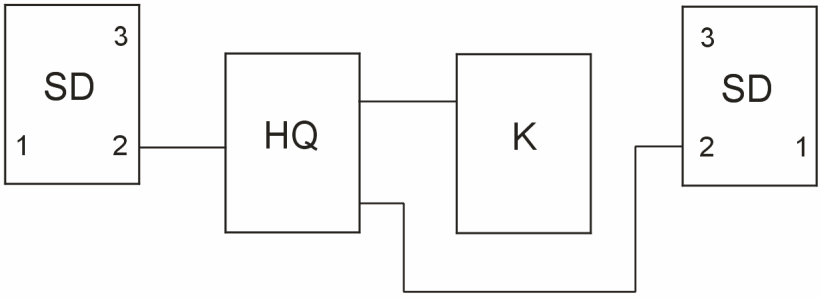
Připojení bloku K a nextSD:

Blok K se připojí na vstup *from_K* bloku HQ. Připojení bloku SD je realizováno přes vstup *from_SD*. Výstup ke vzdálenějšímu bloku je *to_nextSD*. Tímto signálem je vzdálenější blok informován o povolené rychlosti. Ukázka na obrázku č. 9.

Zápis ve VHDL:

```
from_K => K5_sig.vystupy.to_HQ_L,
from_SD => SD3_sig.vystupy.to_HQ3,
to_nextSD => HQ4_sig.vystupy.to_nextSD¨
```

Blokové schéma:



Obrázek č. 9: Propojení bloku HQ s SD a nextSD

Připojení bloku M:

Blok M se připojí na vstup *from_K* bloku HQ. Připojení bloku SD zůstává stejné. Blok VJZD se ke vstupu tohoto bloku nepřipojuje. Výstup k bloku VJZD je signál *to_VJZD*.

Zápis ve VHDL:

```
from_K => M2_sig.vystupy.to_HQ,
from_SD => SD4_sig.vystupy.to_HQ1,
to_VJZD => HQ3_sig.vystupy.to_VJZD,
```

Blokové schéma:

Jako na obrázku č. 4.

Volné zakončení:

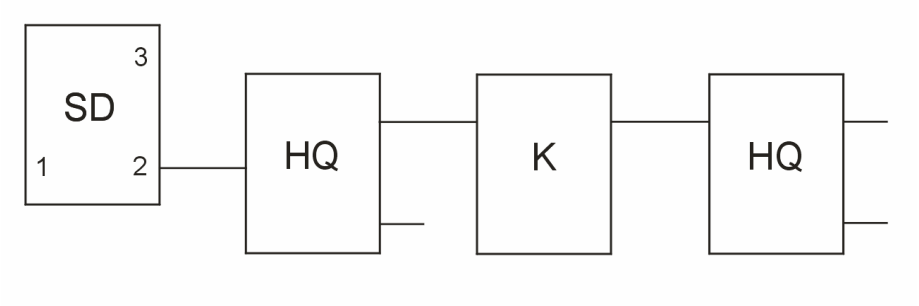
Blok HQ může být umístěn na konci koleje. Za ním se už nenachází žádný blok. Typické zapojení bloků v tomto případě je znázorněno na obrázku č. 10. Blok HQ nejvíce vpravo měl být připojen k bloku SD. Na jeho vstupy se musí nastavit logické nuly, vstup nesmí být v neurčité

hodnotě. Výstupy jsou připojeny k signálu, který dále nevede. Výstupní signály nemusí být připojeny. U bloku HQ vlevo je pouze výstup nepřipojený.

Zápis ve VHDL:

```
HQ7_sig.vstupy.from_SD.B_Status <= "0000";
HQ7_sig.vstupy.from_SD.B_Zrus <= "00";
HQ7_sig.vstupy.from_SD.B_p_akt <= "000";
HQ7_sig.vstupy.from_SD.B_p_pr <= "000";
```

Blokové schéma:



Obrázek č. 10: Volné zakončení bloku HQ

3.4.5 Propojení bloku K

Blok K značí staniční kolej. Je určen pro přejezd vlaků mezi bloky HQ. Hlídá obsazenost staniční koleje. Vlak se na tomto bloku může zastavit a změnit směr jízdy. Jeho vstupy jsou na všech nádražích stejné jako na základním. Blok K je osově symetrický. Na obě strany má stejné vstupy a výstupy.

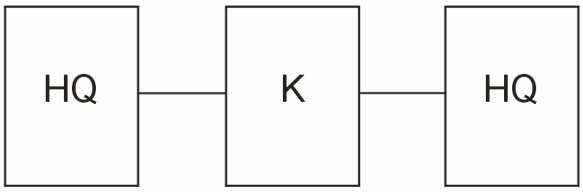
Připojení bloku HQ:

Bloky HQ se připojují na vstupy *from_HQ_L* a *from_HQ_R*. Připojení bloku je stejné z obou stran tak, jak je ukázáno na obrázku č. 11. Bloky HQ mají výstup *to_K*, který se přímo připojí na vstup bloku K. Je nutné zapojit levý blok HQ na levý vstup a levý výstup bloku K, signály se nesmí překřížit.

Zápis ve VHDL:

```
from_HQ_L => HQ5_sig.vystupy.to_K,
from_HQ_R => HQ7_sig.vystupy.to_K,
to_HQ_L => K6_sig.vystupy.to_HQ_L,
to_HQ_R => K6_sig.vystupy.to_HQ_R,
```

Blokové schéma:



Obrázek č. 11: Propojení bloku K s HQ L a HQ R

3.5 Testování bloků

Vytvořené bloky měly mít stejnou funkci jako bloky původní. Pro kontrolu správného zapojení a celé úpravy bylo třeba otestovat bloky na stejnou funkčnost. Základní otestování bylo provedeno tak, že bloky byly propojeny do základního nádraží, ze kterého bylo od počátku vycházeno. Vytvořený projekt obsahoval propojení již pomocí skupin vodičů přímo ve VHDL kódu oproti původnímu schématickému propojení.

Při editaci nedošlo k zásahu do hlavních částí bloků. Bloky byly jen rozděleny, nikoliv zásadně přestavěny. Tím jsme se mohli omezit pouze na hledání chyb v propojení částí bloků nebo špatném napojení skupin vodičů na vstupy nebo výstupy.

Z upravených bloků bylo sestaveno základní nádraží. Abychom mohli testovat zapojení na přípravku, musely být do projektu přidány bloky jak pro vnější, tak i pro vnitřní komunikaci. Stav nádraží se uchovával v registrech. Ty byly dvojího typu, jedny pro vstup a jedny pro výstup z funkčních bloků. Vstupní registry byly připojeny přes komponentu *UART* a *radic_vstupu* a nastavovaly se pomocí sériové linky z počítače. Vstupní data z UARTu a výstupní registry se zobrazovaly na LCD displeji. Ten ukazoval na dvou řádcích stavy bloků tak, že jeden znak znamenal jeden funkční blok nádraží a bylo patrné, co se s konkrétním blokem děje. Zda je obsazen, nebo je přes něj vytvářena nebo rušena cesta.

Testování bylo podmíněno tím, že jak syntéza, tak i mapování projektu na design a generování programovacího souboru projektu, proběhne v pořádku. Tyto všechny podmínky byly splněny téměř na první pokus a mohli jsme přejít k testování celého nádraží. Funkce nádraží byla testovaná na obsazenost všech bloků vlakem, na korektní odhalení špatného vybavování vlaku a na stavbu a rušení cest.

3.5.1 Testování nádraží

Nádraží bylo testováno na obsazování jednotlivých bloků a na správné vybavování vlaků. To znamená, že všechny vstupy z kolejiště byly postupně nastavovány na obsazeno. Tento stav se musel projevit na LCD displeji přípravku. Obsazování částí kolejiště bylo prováděno postupně. Vlak tedy byl správně vybavován a bloky musely nabývat pouze přípustných stavů. Další způsob testovaní probíhal úmyslně způsobeným špatným vybavováním vlaků.

Nastavování obsazenosti bloků probíhalo přes sériový port z počítače. Přípravku se posílají tří bajtové příkazy. Na příkaz „XXX“ odpovídá také „XXX“. To značí, že by měla komunikace probíhat správně. Pokud chceme komunikovat s registry, první znak musí být písmeno „A“, na které přípravek začne reagovat a začne nastavovat vstupní registry. Druhý znak značí adresu registru a třetí pak dekadickou hodnotu čísla, kterou chceme nastavit do registru.

Každý blok kolejiště má v testovaném projektu své registry vstupu a výstupu. Například blok VJZD má dva registry na vstup z ovládání, dva na vstup z kolejiště. V tabulce č. 1 jsou znázorněny všechny bloky a jejich adresy pro vstupy z kolejiště. Bloky HQ mají sice vstup z kolejiště, ale ten nenastavuje obsazení bloku. HQ značí návěstidlo a to nemůže být obsazeno. U ostatních bloků jsou pro nás při testování obsazení důležité právě signály na obsazení bloku. Názvy bloků v tabulce č. 1 odpovídají nákresu v obrázku č. 2.

Blok	Adresa	Blok	Adresa
VJZD L	A	SD L	F
VJZD R	C	SD R	G
M L	D	K 1	H
M R	E	K 2	I

Tabulka č. 1: Adresy vstupních registrů z kolejiště

Nastavování registrů probíhalo příkazy typu AH1. Ten značí obsazení bloku K 1. U všech bloků musí být nultý bit nastaven na jedničku, aby byl blok obsazen. To znamená, že posílaná hodnota musí být liché číslo. U bloků VJZD musí být ještě navíc nastaven bit první, který značí, že signalizace červené je v pořádku. Pro tyto bloky se posílal signál AA3 pro obsazení a AA2 pro uvolnění bloku. Ostatní bloky se obsazovaly číslem 1 a uvolňovaly číslem 0. Signál z kolejiště u bloku výhybky je tříbitový, protože do bloku vstupuje ještě informace o dorazu výhybky. To určuje směr jízdy vlaku. Aby byla výhybka v šikmém směru, musí se nastavit na hodnotu 6. Pro obsazení šikmé výhybky se musí poslat do registru hodnota 7 a pro uvolnění zpět hodnota 6.

Během testování se objevilo několik drobných chyb, které byly způsobeny kopírováním jednotlivých bloků. Chyby se projevily tak, že se blok neobsadil vlakem vůbec, nebo byl obsazen jiný blok než jsme očekávali. Všechny tyto chyby byly rychle odstraněny a nádraží funguje v tomto směru stejně, jako původní návrh.

3.5.2 Stavění cest

Správně musela fungovat i komunikace mezi bloky. Potřebovali jsme si ověřit, že propojení bloků ve VHDL bylo správné a že napojení skupin signálů funguje. Tato kontrola probíhala stavěním cest mezi bloky VJZD a HQ. Pouze mezi těmito bloky lze vytvořit vlakovou cestu. Po vytvoření cesty byly bloky opět obsazovány a byl simulován průjezd vlaku. Vlak musel být správně vybavován. V případě, že vlak nejel ve správném směru nebo že někde nebyla správně nastavena poloha vlaku, musely bloky hlásit chybu a přepnout se do chybového stavu.

Stavění cesty probíhá postupným zadáváním počátku a cíle. Všechny bloky na vlakové cestě ověří, zda nejsou obsazeny a mezi počátkem a cílem se vytvoří cesta.

Vytvořená vlaková cesta je znázorněna na LCD displeji přípravku. Po této cestě může projet vlak. K zadávání cesty se používají vstupy bloků, které vedou z ovládání. Tyto vstupy jsou mapovány na jiné adresy vstupních registrů než byly adresy vstupů z kolejiště.

Blok	Adresa	Blok	Adresa
VJZD L	@	HQ1	J
VJZD R	B	HQ2	L
SD L	F	HQ3	N
SD R	G	HQ4	P

Tabulka č. 2: Adresy vstupních registrů z ovládání

Registry ovládání jsou dvoubitové. Nultý bit se musí nastavit při určování cíle cesty. Počátek cesty pak určuje první bit. Nastavení začátku cesty se provádělo zadáním hodnoty 2 do registru, tedy nastavením prvního bitu na jedničku. Pro určení cíle se zadala hodnota 1. Cesty, které vedou přes výhybky, se vytvoří jen když jsou všechny výhybky ve správné poloze. Proto jsou v tabulce č. 2 uvedeny i adresy pro bloky SD. Výhybka se do šikmého směru nastaví hodnotou 6 a do rovného 0. Výhybka je po inicializaci v rovném směru.

Ukázka stavění cesty mezi bloky VJZD L a HQ 3:

```
A@2 – určení startu
AN1 – určení cíle
A@0 – počátek hledání cesty
AN0 – dokončení cesty
AF6 – nastavení výhybky do správného směru
```

Ukázka průjezdu vlaku mezi bloky VJZD L a HQ 3:

```
AA3 – obsazení bloku VJZD L
AD1 – obsazení bloku M L
AF7 – obsazení výhybky ve směru šikmo
AH1 – obsazení bloku K 1
AA2 – uvolnění bloku VJZD L
AD0 – uvolnění bloku M L
AF6 – uvolnění bloku SD L se zachováním šikmého směru
AH0 – uvolnění bloku K 1
```

3.5.3 Rušení cest

Rušení cest probíhá postupným zadáváním hodnoty 3 a 0 do registru vstupu z ovládání. Hodnotou 3 se nastaví do logické jedničky jak start, tak cíl. Tím se pozná, že uživatel chce cestu zrušit. Cestu lze zrušit pouze blokem, kterým byla vytvořena. To znamená blokem, kde byl nastaven start cesty. Pokud cesta ještě nebyla vytvořena, zruší se okamžitě. V ostatních případech zrušení cesty trvá určitou dobu v závislosti na obsazenosti rozhodujícího úseku před návěstidlem. Pokud je rozhodující úsek volný, cesta se zruší po 5 s. Jestliže je tento úsek obsazen, rušení cesty potrvá 180 s. Během rušení je na počátečním návěstidle rozsvíceno znamení zakazující vjezd.

4 XML soubor

Před samostatnou tvorbou VHDL generátoru bylo nutno zvolit vhodný typ souboru pro ukládání informací o konfiguraci bloků nádraží. Zprvu bylo několik variant, jaký má mít soubor formát. Hlavním kritériem, jak vybrat správný formát, byla jeho čitelnost a srozumitelnost pro běžného uživatele. Z toho vyplynulo, že výsledný soubor musí být čitelný v obyčejném textovém editoru.

Jednou z možností bylo vytvořit si vlastní textový formát. V tomto souboru by mohl jeden nebo více řádků popisovat jeden blok nádraží. To by znamenalo, že by počet řádků na blok a tedy i formát souboru byl pevně dán programem. Tím by se velice omezila rozšiřitelnost formátu. Při programovém čtení takového souboru by musel být speciálně vytvořen parser, který by uměl tento typ souboru přečíst a rozložit. Další obtížnost by nastala při vytváření takového souboru. Uživatel by musel znát přesnou koncepci souboru a jakákoliv sebemenší chyba by znamenala nezdar při parsování. Tím by se značně prodloužilo vytváření souboru. V případě zásahu do souboru nějakým jiným softwarem nebo pokusem o automatickou úpravu, by musel autor pečlivě studovat strukturu formátu. Použitý formát by měl být spíše otevřeným dokumentem připraveným pro další úpravy.

Nejvhodnější běžně používaný typ souboru se jeví formát XML. Ten nabízí jednoduchou čitelnost pro běžného uživatele. Podmínkou je samozřejmě smysluplné pojmenování tagů. Mnoho volně dostupných textových editorů zvládá barevné zvýrazňování XML kódu, což ještě zvýší orientaci uživatele v textu. Některé programovací nástroje a editory nabízí našeptávání kódových slov XML většinou při stisku kombinace CTRL + mezera. K tomu je nutné, aby pro soubor XML byly vytvořeny definice obsahu, takzvaný XSD soubor. Tvorba nádraží se tak může stát poměrně jednoduchou záležitostí. V neposední řadě je tento formát podporován alespoň v programovacím jazyce Java, který jej umí jednoduše rozdělit, aniž by programátor musel psát vlastní parser.

XML znamená extensible markup language, což je v překladu rozšířitelný značkovací soubor. Značkovací soubor je termín pro soubor, který obsahuje jak zdrojový text, tak pokyny pro jeho zpracování. Jazyk XML je typickým deskriptivním jazykem, který slouží po popis obsažených informací. Formát XML byl standardizován konsorciem W3C (Wolrd Wide Web Consorcium). Toto mezinárodní konsorcium se snaží vyvíjet webové standardy a rozvíjet světový web pomocí protokolů a směrnic.

Konsorcium vzniklo v roce 1994 a standardizovalo jazyk HTML. Formát XML byl vytvořen pro jednoduché přenášení dokumentů mezi různými platformami.

XML soubor by měl být navrhován tak, aby mohl obsáhnou možné potřeby dalších softwarů, které by jej mohly zpracovávat. Měly by v něm být uloženy všechny základní informace o nádraží. Soubor XML musí být jednoduchý a jasný, okamžitě čitelný pro méně informovaného člověka. Po ujasnění těchto cílů můžeme přistoupit k tvorbě souboru.

4.1 Struktura XML

K vytvoření nového nádraží je potřeba znát informace o počtu bloků, typu bloků a jejich propojení. Tento zdrojový soubor bude sloužit pro generátor VHDL kódu. Struktura souboru je rozdělena do několika částí. Úvodní tag celého souboru nese název *schema*. V něm jsou zabaleny hlavní informace o názvů schématu a komentáři o nádraží a projektu. Několik dalších tagů obsahuje informace o šířce a výšce kreslící plochy, poměru zvětšení a posunutí kreslící plochy vůči obrazovce. To jsou právě ty zmiňované doplňkové informace, které by mohly být vhodné pro budoucí rozšíření VHDL generátoru o grafický editor, který by dovolil jednoduchým kliknutím přidávat bloky a navzájem je propojovat.

Nejdůležitějším tagem v této skupině je tag *blocks*, který obsahuje všechny bloky s jejich vlastnostmi a propojením. Bloky jsou definovány tagem s jejich jménem (např. <*VJZD*>). V tomto tagu jsou zabaleny informace o jméně prvku, komentáři, poloze objektu na kreslící ploše a jeho vertikálním a horizontálním převrácení. Nejdůležitější informací o objektu je jeho identifikační číslo, které musí být v daném XML souboru unikátní a slouží k informaci o propojování. Identifikační číslo musí být v oboru přirozených čísel. To znamená jen celá čísla bez nuly. Číslo nula je rezervováno pro propojení u bloku HQ, který není dále připojen k dalšímu bloku a ukončuje kolej.

V dalším tagu jsou uchovány informace o propojení bloků. Každý blok může být propojen pouze s některými bloky. Podle toho, jak může být blok propojen jsou ve skupině <*connectionBLOK*> vytvořeny příslušné tagy, které určují spojení. U těchto tagů je nutné uvést identifikační číslo bloku, na který je tento blok připojen. Pokud blok

není dále připojen, do kolonky se vyplní hodnota nula. V souboru XML nesmí zůstat nevyplněné tagy bez hodnoty.

Všechny tagy v tomto XML jsou párové. To znamená, že každý otevírací tag musí k sobě mít ještě tag uzavírací, který začíná znakem „/“. Po vložení všech bloků tedy musíme ukončit prostor pro bloky tagem *</blocks>* a ještě uzavřít celé schéma tagem *</schema>*.

4.2 Definice XML

Pro kontrolu správného obsahu souboru XML byl vytvořen soubor XSD. Tento soubor obsahuje definice a přesnou strukturu souboru XML. Určuje typy všech proměnných. Určuje elementy, které se mohou opakovat, které musí být jen jednou v celém dokumentu, a definuje stromovou strukturu a hierarchii jednotlivých elementů. Pro některé textové editory může sloužit jako nápověda při tvorbě XML souboru.

Vytvořený XML soubor splňuje všechny definice uvedené v souboru XSD. To zaručí bezproblémovou funkci VHDL generátoru. Soubor XML je závislý na pořadí vkládání tagů. V našem případě záměna pořadí některých částí kódu vůbec neovlivní jeho výsledné zpracování. Nicméně kontrola souboru neprojde a validátor bude hlásit chyby. Validátor většinou informuje uživatele, kde nastala chyba a radí mu, co opravit. Pokud by chyby přetrvávaly, musíme soubor XML vytvořit znovu a lépe, přesně podle definic v XSD souboru. Zkontrolovat vytvořený XML soubor podle definice XSD lze na webové stránce[5].

Ukázka kódu XML s jedním blokem VJZD:

```
     <?xml version="1.0" encoding="utf-8"?>
     <schema xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi: noNamespaceSchemaLocation="xmlDefinitions/pavel.xsd">
          <scale>1.0</scale>
          <left>0</left>
          <top>0</top>
          <width>0</width>
          <height>0</height>
          <name>Schema</name>
          <comment>Komentar</comment>

          <blocks>
               <VJZD>
```

```
                    <left>0</left>
                    <top>0</top>

                    <flip>
                          <horizontal>false</horizontal>
                          <vertical>false</vertical>
                    </flip>

                    <id>1</id>
                    <name>VJZD1</name>
                    <comment></comment>

                    <connectionVJZD>
                          <conM>2</conM>
                          <conSD>3</conSD>
                    </connectionVJZD>
               </VJZD>
          </blocks>
</schema>
```

5 Tvorba generátoru

Přepracované bloky ve formátu VHDL je potřeba propojovat automatickým nástrojem. Ačkoliv propojování bylo značně zjednodušeno seskupením vodičů, využití souboru XML je ještě jednodušší. Prvotním cílem při tvorbě generátoru bylo automatické vytvoření vzorového nádraží dle námi připraveného XML souboru. Generátor byl vytvořen jako konzolová aplikace nezávislá na operačním systému. Byl naprogramován v programovacím jazyce Java 1.6.

5.1 Analýza problému

Požadavkem na funkčnost a použitelnost generátoru VHDL kódu byla spustitelnost na více operačních systémech. V úvahu přicházely programovací jazyky C/C++ a Java. Jazyk C/C++ má výhodu v rychlosti prováděného kódu. Rychlost v tomto případě nebyla hlavní prioritou. Předpokládá se, že se generátor VHDL použije pouze jednou k vytvoření hlavního souboru nádraží a poté zůstane delší dobu nevyužit. Přípustná délka výpočtu by mohla být i v jednotkách minut. Program v jazyce C/C++ je potřeba pro každý operační systém překládat zvlášť. Na rozdíl od přeloženého kódu v Javě, který lze spustit na všech počítačích s JVM (Java Virtual Machine). To v dnešní době podporuje každý běžně rozšířený operační systém.

Volba tedy padla na programovací jazyk Java, který byl také hojně probírán v mnoha školních předmětech a zkušenosti s ním byly větší než s jazykem C/C++. Navíc nabízel jednodušší přenositelnost mezi různými OS. Další výhodou bylo povědomí autora o tom, že pro Javu existují balíčky od konsorcia W3C, které dovolí jednodušší procházení XML souboru. Výhodou programovacího jazyka Java byla i značná zkušenost autora s prostředím NetBeans 6.7.1, které je volně dostupné na internetu a je vyvíjeno společností Sun jako Open-Source.

Prvotní myšlenkou bylo vytvoření grafického rozhraní, ve kterém se bude editovat celé nádraží a bude nezávislé na formátu XML. Tato myšlenka byla brzy zamítnuta, protože tvorba grafického rozhraní vyžaduje externí grafické knihovny. To může občas způsobovat problémy při spouštění programu. Grafické objekty nemusí být na správném místě a program by nebylo možné využívat v plném rozsahu. Z tohoto důvodu bylo zvoleno vytvoření pouze konzolové aplikace, která ke svému fungovaní

bude potřebovat pouze vstupní XML soubor, který lze vytvořit samostatně v textovém editoru. Důraz tedy byl kladen hlavně na spolehlivost softwaru. Při vývoji takovýchto aplikací je důležitější funkčnost než barevné grafické rozhraní. To nejlépe splňuje konzolová aplikace. Grafická aplikace by tedy mohla být vytvořena jen jako rozšíření k tomuto VHDL generátoru. Například jako bakalářská práce, která by zobrazovala pozice bloků tak, jak jsou uloženy v XML a zobrazovala propojení mezi bloky.

5.2 Struktura programu

Program psaný v jazyce Java obsahuje mnoho tříd rozdělených do tří balíčků z důvodu přehlednosti a orientace. Balíčky jsou rozděleny podle funkčnosti jednotlivých tříd, které jsou do nich vloženy. Balíčky mají názvy *data*, *IO*, *dip*. Balíčky *data* a *IO* obsahují větší množství tříd kvůli přehlednosti a jednoduché editaci programu.

5.2.1 Balíček *data*

Datovým základem je balíček *data* obsahující třídy pro reprezentaci bloků kolejiště a schématu. Základní třídou je třída *Block*, která je pro všechny ostatní třídy rodičem, kterého rozšiřují. Třída *Block* obsahuje privátní proměnné pro určení pozice na kreslící ploše, převrácení objektu ve vertikální a horizontální ose, identifikační číslo, název a komentář. Kromě metod pro nastavení nebo přečtení hodnoty těchto proměnných obsahuje ještě třída metodu na zjištění, zda jsou všechny proměnné inicializovány. Pokud proměnná nebyla nastavena z vnějšího vstupu, má proměnná přiřazenou hodnotu -1. Tím lze jednoduše zjistit, zda uživatel při tvorbě XML vyplnil všechny potřebné údaje. Všechny další metody jsou přepsány v třídách, které třídu *Block* rozšiřují tak, aby odpovídaly vlastnostem bloku. Jsou to metody na kontrolu správného zapojení objektu a metody pro výpis textů do VHDL souboru.

Názvy tříd rozšiřující třídu *Block* korespondují s názvy bloků v kolejišti. Máme tu dalších pět podobných tříd s názvy *VJZD, M, SD, HQ, K*. Všechny tyto třídy navíc obsahují privátní proměnné, které ukazují identifikačním číslem na blok, ke kterému jsou připojeny. Všechny proměnné obsahující identifikační číslo jsou typu integer, protože tato čísla nemohou nabývat jiných hodnot. To je určeno jak v definici XML, tak při načítání souboru by došlo k chybě a program by byl ukončen s výpisem výjimky,

ke které došlo. Například třída *VJZD* obsahuje proměnné s názvy *conSD*, *conM* a *conHQ*. Samozřejmě i zde existují metody na nastavení a přečtení celočíselné hodnoty z proměnných.

Ke všem proměnným pro propojení existují metody na kontrolu správného propojení. Například metody *isCorrectConHQL()*. Metoda zjistí, jaké identifikační číslo má připojený blok a porovná, zda tento blok je také připojen k původnímu bloku. Jednodušeji řečeno, jestli je spojení oboustranné a zda nevznikla někde chyba. Různé bloky mají jiný počet těchto metod a proměnných, protože mají jiné počty skupin vodičů a propojení. Proto byla již v třídě *Block* připravena metoda s názvem *isCorrect()*, která zjišťuje, zda je daný objekt správně zapojen. Protože je každý blok jiný, je vždy tato metoda přepsána podle potřeb příslušné třídy a vrací hodnotu typu boolean, zda jsou úplně všechny propojení bloku správné. To zjistí tak, že zavolá metody na kontrolu spojení a zjistí, jestli je blok zapojen správným způsobem. To znamená, že propojení musí být jednoznačná a nemohou se dohromady míchat různé způsoby zapojení bloku.

Každá tato třída obsahuje metody pro výpis do VHDL souboru. Tyto metody obsahují předdefinovaný kus VHDL souboru a do správných míst doplní své identifikační číslo nebo číslo bloku, ke kterému jsou připojeny a vypíšou se do generovaného souboru. Samostatné vypisování těchto textů vypadá zprvu jednoduše. Musíme ale vždy dávat pozor na to, jak je blok připojen. Vždy existuje více možností, jak bloky do sebe propojit. Pravidla byla zmíněna v kapitole 3.4 a podle těchto pravidel se generování VHDL kódu musí řídit. Vstupy bloků jsou vždy stejné a jde o to, jakým způsobem se mají ostatní bloky připojit. V případě složitějších bloků, jako je na příklad blok SD, se v této metodě musí zaručit správné doplnění vodičů na odvraty a nastavení hodnot ťapání. Připojení k bloku SD je v XML souboru určeno jeho identifikačním číslem. To ale nevypovídá nic o tom, na který port bloku SD se připojujeme. Proto při generování kódu musí docházet k dohledání správného portu a vypsání tohoto propojení.

5.2.2 Balíček *IO*

V balíčku jsou třídy, které slouží programu pro datový vstup a výstup. Hlavním vstupem je soubor XML. Tento soubor je načítán pomocí třídy *XMLreader.java*, ta obsahuje knihovny vytvořené konsorciem W3C, které definovalo formát XML

a vytvořilo užitečné metody pro parsování XML kódu. Pomocí těchto metod lze jednoduše načíst vstupní soubor a přistupovat jedním příkazem k elementům XML.

Třída *Writer.java*, která se také nachází v balíčku *IO*, slouží k otevření souboru a uložení obsahu proměnné *stringBuffer*. Pro zápis textových dat do této proměnné slouží veřejné metody *printSB(String text)* a *printlnSB(String text)*, jejichž názvy jsou převzaty z balíčku *System.out* tak, aby programátor jasně pochopil, jakou mají funkci. Koncovka SB značí, že se text ukládá do proměnné *stringBuffer*. Třída *WriterVHDL.java* rozšiřuje třídu *Writer.java* a přepisuje některé metody. Slouží k vytvoření VHDL výstupu. Přesnější postup bude popsán v kapitole 5.4.

Balíček *IO* obsahuje ještě třídu *VHDLtexts.java*, která slouží pro ukládání textů vhodných pro tvorbu VHDL souboru. Jsou zde uloženy komponenty bloků, hlavička souboru a další potřebné texty, které by přímo v kódu zbytečně překážely. Tyto texty jsou uloženy v proměnných typu String.

5.2.3 Balíček *dip*

Tento balíček obsahuje pouze třídu *main*, která slouží ke spouštění programu. Program potřebuje pro funkci zadané vstupní argumenty. Abychom věděli, co zpracovávat, musíme zadat alespoň jméno vstupního souboru. Pokud soubor není ve stejném adresáři, je nutné zadat i relativní cestu k němu. Jestliže uživatel nezadá žádný vstupní soubor, nemůže se program vykonat a bude ukončen.

Jako druhý argument můžeme zadat název výstupního souboru. Pokud název nezadáme, bude automaticky výstup uložen v souboru *nadrazi.vhd*. Jestliže už tento soubor existuje, bude přepsán.

Program spustí načítání souboru XML. Pokud při kontrole vyplněných informací narazí třída *XMLreader.java* na chybu, program se ukončí. Po správném načtení vstupu se samozřejmě zavolají všechny kontrolní metody. Zkontroluje se, zda jsou všechna propojení mezi bloky správná a oboustranná. Program také prověří všechna zadaná identifikační čísla na to, zda jsou kladná. Pokud jsou splněny všechny tyto podmínky, program vytvoří výstupní soubor. Pokud ne, program skončí chybovou hláškou, která oznámí, kde nastala chyba a co je nutné opravit.

5.3 Načítání XML

Načítání souboru je prováděno třídou *XMLreader*, ta je umístěna v balíčku *IO*, jak již bylo zmíněno výše. Do třídy jsou importovány balíčky od konsorcia W3C, které slouží k otevření souboru a nabízí metody pro parsování XML souboru. S využitím těchto metod nejprve přistoupíme k hlavičce XML souboru. Načítání souboru začne zavoláním konstruktoru v třídě *main* příkazem `XMLreader(args[0]).` Nejprve se vytvoří veřejné proměnné pro ukládání bloků nádraží a celého schématu. Poté načteme hodnoty, které by mohly být použity pro případný grafický editor nebo pro zobrazování polohy bloků a spojení bloků. Tyto hodnoty jsou uloženy v proměnné *projekt*, která je typu *Schema*. To je třída, která shromažďuje všechny tyto proměnné včetně názvu schéma a komentáře k němu.

Jakmile dojde k parsování tagu *blocks*, program vybírá a zpracovává jednotlivé bloky. Každý blok vytvoří instanci třídy stejného názvu. Do této instance vkládá program veškeré načtené hodnoty z XML souboru. Po načtení celého obsahu bloku je zavolána privátní metoda *isSet()*. Metoda vrací informaci o tom, zda byl blok správně vyplněn, zda uživatel zadal správné kombinace propojení podle toho, jak lze daný blok zapojit. Pokud došlo k chybnému vyplnění, skončí program chybou a hlásí, o který typ bloku se jedná včetně jeho identifikačního čísla. Jestliže tato kontrola proběhne v pořádku, uloží se instance dané třídy do dvou seznamů typu *ArrayList*. Jeden slouží pro ukládání všech bloků a jeho instance jsou typu *Block* a druhý slouží pro ukládání instancí pouze jednoho stejného typu. Po načtení celého XML jsou tyto seznamy naplněny všemi bloky nádraží a lze jednoduše zjistit, jestli se určitý typ bloku v nádraží vyskytuje. Z toho důvodu jsou zde seznamy jednotlivých bloků.

5.4 Generování výstupu

Tvorba VHDL výstupu se spustí zavoláním konstruktoru třídy *WriterVHDL.java*, který dědí metody na zápis do souboru od třídy *Writer.java*. Do konstruktoru vstupuje parametr typu *String*, který určuje název výstupního souboru. Zavolání konstruktoru předpokládá, že veškeré podmínky nutné k funkci jsou splněné a soubor XML je správně načten.

Volání konstruktoru:

```
new WriterVHDL(args[1]);
```

Třída *WriterVHDL.java* obsahuje přepsanou metodu *zpracujData()*, která slouží pro generování textu. Tato metoda volá jiné metody, které přidávají text do *stringBufferu*, který se na konci běhu programu vypíše do souboru. Metoda *zpracujData()* přistupuje do třídy *VHDLtexts.java*, odkud nejprve čerpá text hlavičky VHDL souboru. Nadefinuje název entity, připojí vstupní signály pro hodiny a reset. Poté zjistí od všech bloků jaké vstupní signály mají být připojeny k entitě nádraží. Tuto akci provede zavoláním metody *vypisVstupy()* na každém bloku nádraží. Podobným způsobem se zjistí výstupy jednotlivých bloků a vypíší se. Metoda pro získání výstupů se nazývá *vypisVystupy()*.

Po vypsání všech portů entity následuje zápis komponent. Ty čerpá třída *WritelVHDL.java* z třídy *VHDLtexts.java*. Komponenty, které nejsou v nádraží použity je zbytečné vypisovat, proto se nejprve zjistí, jestli daný blok vůbec v nádraží existuje. To program zjistí podle toho, zda daný seznam typu *ArrayList* v třídě *XMLreader.java* je prázdný nebo ne. Podle toho se vypíše příslušná komponenta.

V poslední řadě se vypíší propojení mezi bloky. Třída zavolá metodu *vypisConnection()* pro každý blok nádraží a ten vypíše správné napojení svých vstupů a výstupů tak, jak bylo načteno z XML souboru a je uloženo v privátních proměnných každého bloku. Metoda *vypisConnection()* je jednou z nejdůležitější částí programu a musí správně generovat propojení bloků v závislosti na tom, kam jsou bloky připojeny. Musí také řešit připojení vstupů a výstupů bloku z celé entity nádraží.

5.5 Spouštění a chybové hlášky

Generátor VHDL se spouští v příkazové řádce příkazem *java –jar dip1.java soubor.xml*. Ke spuštění programu je nutné mít nainstalovanou Javu 1.6. Jako vstupní parametr musí být zadán zdrojový soubor XML. Pokud nezadáte název souboru, program bude ukončen s hláškou „Nezadán zdrojový XML soubor!“. Druhý parametr je nepovinný a určuje název výstupního souboru.

Soubor XML musí obsahovat přesně danou strukturu tak, jak je uvedeno v definici XSD. Pokud nějaká položka bude chybět nebo nebude správně uzavřena,

skončí program chybou při parsování souboru. Stejně skončí chybou i pokud bude místo číselné hodnoty zadán textový řetězec, který nebude očekáván.

V případě, že nějaký blok nebude vůbec obsahovat určitou povinnou část, bude uživatele program informovat, v jakém bloku a u jakého identifikačního čísla se chyba vyskytuje. To se může stát, když uživatel zapomene třeba vyplnit jedno propojení v bloku. Výstupní soubor se při chybové hlášce nevytvoří.

Chybová hláška:	**Řešení:**
Nezadán zdrojový XML soubor!	Musíte jako parametr při spouštění zadat jméno XML souboru.
Opakuje se ID:	Bloky musí mít různá identifikační čísla.
ID nesmí byt záporné nebo rovno 0, chyba na bloku: VJZD1	Opravte identifikační číslo u bloku s vypsaným názvem.
Chybné propojeni v ID:	Blok s vypsaným identifikačním číslem je špatně zapojen.
Chyba nevyplněno VJZD ID: 1	Nejsou vyplněny všechny položky, zkontrolujte uvedený blok.
The end-tag for element type	Špatná struktura souboru XML, zkontrolujte vstupní soubor.
The element type …	

Tabulka č. 3: Chybové hlášky VHDL generátoru

Jestliže se ve vstupním souboru ani v propojení bloků nevyskytly žádné chyby, které by mohl VHDL generátor odhalit, vypíše informaci o tom, jaký soubor otevírá, jméno dokumentu a kam VHDL ukládá. Pak program začne vytvářet výstupní VHDL soubor. Jakmile je program hotov s generováním, vypíše informaci o dokončení tvorby souboru. Při reálném testování doba, po jakou program vytvářel výstup nikdy nepřekročila jednu sekundu ani u nejsložitějšího nádraží, které bylo vytvářeno a obsahovalo přibližně 30 bloků nádraží. Můžeme tvrdit, že velká nádraží, která budou obsahovat až stovky bloků, bude generátor zpracovávat déle. Odhadem by se podle současné doby výpočtu a použitých algoritmů dalo říci, že to nebude trvat déle než několik sekund, maximálně desítek sekund.

Výpis při správném XML souboru:

```
C:\> java –jar dip1.java soubor.xml

Oteviram soubor: nadrazi.xml
Jmeno dokumentu: Schema
Propojeni bloku v XML je v poradku
Vytvarim soubor: nadrazi.vhd
Hotovo!
```

5.6 Testování výstupu

Kontrola vygenerovaného VHDL souboru probíhala nejprve jednoduchým porovnáváním s ručně vytvořeným základním nádražím. Hledání chyb se mohlo omezit jen na propojení skupin vodičů, protože správná funkce upravených bloku byla prokázána již dříve. Během testování bylo odhaleno několik chyb. Převážná většina z nich byla syntaktických. Generátor například vkládal v entitě nádraží středník za každý signál, ačkoliv u posledního signálu být neměl. Chyby občas nastávaly u bloku SD, který byl pro určení podmínek propojení signálů nejsložitější. Některé signály pro nastavení odvratů a ťapání nebyly správně nastavovány. Podmínky připojování signálů musely být znovu zkontrolovány a v programu upraveny do správné podoby. Všechny tyto chyby byly úspěšně odhaleny a mohli jsme přejít k testování na přípravku.

Vstupy a výstupy z celé entity musely být napojeny na ostatní komponenty, které sloužily k přenosu signálů do vstupních a výstupních registrů. Poté nastalo opětovné připojování komponent pro ovládání LCD displeje, komunikaci přes UART a řadičů vstupu a výstupu. Tyto komponenty byly testovány již dříve a tak šlo hlavně o to, zda jsou bloky správně propojeny mezi sebou.

Syntéza projektu i mapování na design proběhly v pořádku a mohli jsme zkonstatovat, že výstupní signály byly napojeny na vstupní minimálně stejného typu. Jinak by nedošlo ani k syntéze a program by hlásil nekompatibilitu signálů. Takto vytvořené nádraží jsme mohli nahrát do přípravku. Ten na LCD displeji zobrazoval stavy bloků, které jsme potřebovaly otestovat na správné obsazování a uvolňování, na vytváření a rušení cest.

Všechny bloky byly otestovány na obsazení a uvolnění tak, jako když jsme testovali správnou funkčnost bloků. Všechny tyto testy proběhly v pořádku a mohli jsme vytvářet vlakové cesty. Tvorba cest probíhala bezchybně. Byly vytvořeny všechny kombinace vlakových cest a vždy byly vytvořeny tak, jak jsme očekávali. Postavené

cesty ještě bylo potřeba otestovat na správné vybavování a tedy na složitější komunikaci mezi bloky. Průjezdy všemi cestami prokázaly správnou funkci bloků a jejich propojení. Jestliže jsme schválně porušili směr jízdy vlaku, bloky ihned hlásily chybné vybavování. Rušení cest také fungovalo správně. Vygenerovaný soubor VHDL byl otestován na veškeré funkce nádraží. Tím jsme ověřili, že výsledný VHDL generátor funguje správně a lze jej použít pro stavbu libovolných nádraží.

6 Grafická část editoru

Samotné využití generátoru VHDL kódu by bylo trochu složitější. Ručně vytvořit XML souboru z připravených ukázkových souborů lze poměrně jednoduše, ale tímto způsobem nemá uživatel přesný přehled o tom, který blok má jaké vstupy a výstupy. Musí tedy pečlivě prostudovat funkci bloků a možnosti jejich propojení. Ke zjednodušení tvorby XML byl vytvořen grafický generátor tohoto kódu. V něm lze vkládat začátek XML souboru, bloky železnice a konec souboru do textového pole. Toto pole se může uložit do souboru a následně použít pro generátor VHDL. Výhodou tohoto generátoru je to, že XML kód bude vytvořen správně a VHDL generátor by neměl hlásit chyby při parsovaní XML.

6.1 Analýza problému

Požadavkem na XML generátor byla opět spustitelnost na více operačních systémech, aby mohl být program dodáván společně v kombinaci s VHDL generátorem. Volba programovacího jazyka padla opět na jazyk Java, aby se zachoval stejný styl programu. Jediným rozdílem bylo potřeba vytvoření grafického menu. Grafické rozhraní v Javě nabízí knihovna SWING, ta poskytuje dostatečné množství grafických komponent pro tvorbu programu. Jedinou její nevýhodou je nutnost distribuce knihovny s programem. Jako vývojové prostředí byl vybrán software NetBeans 6.7.1 od společnosti Sun, který podporuje tvorbu GUI pomocí využití myši.

6.2 Grafické rozhraní

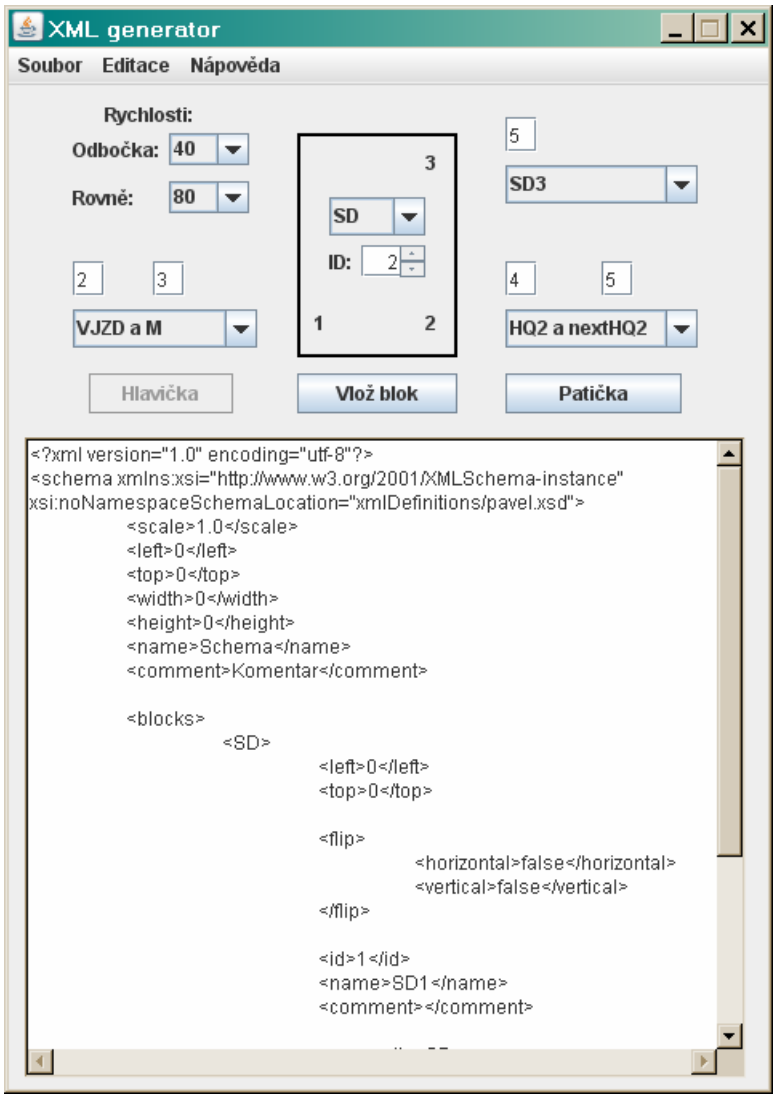
Hlavní okno simulátoru je pevně horizontálně rozděleno do dvou částí tak, jak je zobrazeno na obrázku č. 12. Ve spodní části je zobrazen textový editor, ve kterém se vypisují texty XML generátoru podle zvolených parametrů. Případně se do této části dají vpisovat vlastní poznámky, data a upravovat XML kód ručně. Obsah editoru lze přes schránku kopírovat nebo vkládat a tím doplňovat vytvořený text.

Horní část programu slouží ke konfiguraci. Uprostřed černého obdélníku je objekt typu *ComboBox*, který slouží k výběru bloku, který chceme vložit. Pod ním je

JSpinner, který je při vložení bloku automaticky inkrementován a slouží k nastavení identifikačního čísla vkládaného bloku. Podle zvoleného bloku se zobrazí další *ComboBoxy*, ve kterých si uživatel vybere typ propojení k ostatním blokům a do přilehlých textových polí vloží identifikační čísla jednotlivých připojených bloků. U bloku SD se zobrazí ještě navíc *ComboBoxy* s předdefinovanými hodnotami pro výběr rychlosti do odbočky a pro směr rovně.

V prostoru mezi schematickou značkou bloku a textovým polem jsou tři tlačítka. První tlačítko zleva s nápisem „Hlavička“ slouží k vepsání textové hlavičky XML souboru, která je nutná podle XSD definice a pro správnou funkci VHDL generátoru. Hlavičku lze vložit jen jednou a to na začátku textového pole. Tlačítko „Vlož blok“ vypíše do textového pole XML reprezentaci zvoleného bloku, jeho typ propojení a zadá identifikační čísla napojených bloků do textu podle toho, jak je uživatel vyplnil do textových polí poblíž *ComboBoxu* propojení. Poslední a zároveň třetí tlačítko zleva s nápisem „Patička“ funguje jako závěrečné tlačítko na vložení ukončení XML souboru. Bez tohoto zakončení není soubor korektní podle definice a VHDL generátor bude hlásit chybu a nevytvoří výstup. Toto tlačítko lze použít jen při dokončení kódu a po vložení patičky nepůjde již vložit další blok.

Rozměry okna programu jsou 480x680 pixelů a jeho velikost nelze měnit. Pevné okno bylo vytvořeno z důvodu jednoduššího rozmisťování objektů na designu. Knihovna Swing nabízí mnoho různých způsobů, takzvaných layoutů, jak rozmisťovat ovládací prvky. Problémem většiny z nich bylo nepředvídatelné chování při skrytí objektu. Toho je hojně využíváno při změně typu bloku. Vertikální rozměr byl zvolen tak, aby se vešel na běžný typ notebooku, který má vertikální rozlišení 800 pixelů.



Obrázek č. 12: Grafické rozhraní XML generátoru

6.3 Položky menu

Na liště menu programu jsou umístěny tři základní prvky. První zleva „Soubor“ typicky obsahuje položku „Ulož jako“ pro uložení obsahu textového pole. Pro rychlý přístup slouží klávesová zkratka CTRL + S. Po vyvolání této nabídky se zobrazí dialogové okno pro výběr cesty a názvu souboru, kam se má textový obsah uložit. Druhá položka slouží k ukončení programu a je mu přiřazena obvyklá klávesová zkratka ALT + F4.

V nabídce „Editace“ je položka pro volbu zpět. Ta si pamatuje jeden zpětný krok, který lze vyvolat kombinací CTRL + Z a položka pro nastavení programu do původních hodnot, která vymaže obsah textového pole a obnoví možnost vložení

hlavičky, bloku a patičky. Tato tlačítka mohou být zablokována podle toho, v jakém stavu program je.

V menu nápověda je položka, po jejímž vyvolání se zkráceně zobrazí postup, jak použít generátor XML a jak výsledný soubor přeložit v generátoru VHDL. V položce „O programu“ je popsáno, za jakým účelem a kdy byl tento program vytvořen.

6.4 Uživatelská příručka

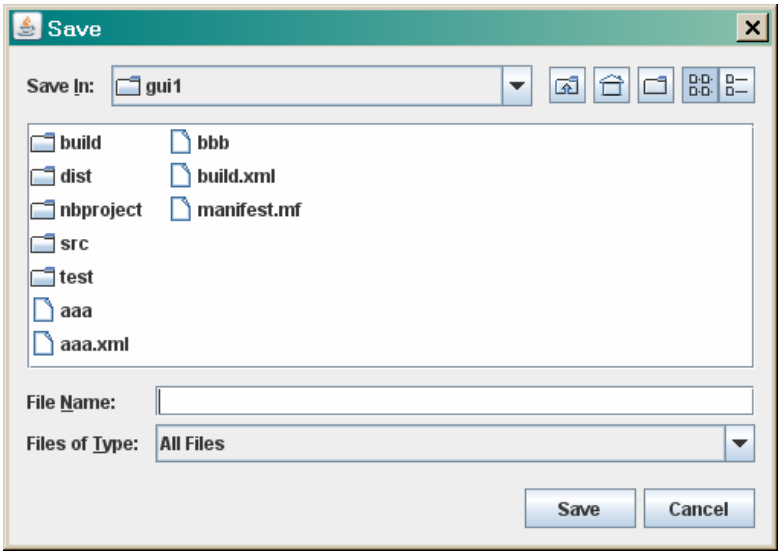
Program je napsán v programovacím jazyce Java 1.6 a je použita grafická knihovna Swing. Program se spouští nejlépe z příkazového řádku příkazem „*java –jar gui1.jar*“ nebo dvojklikem myši, pokud je v systému nastaveno chování pro soubory s příponou jar. Adresář, kde se nachází aplikace, musí obsahovat podadresář s názvem *lib*, ve kterém se nachází grafická knihovna *swing-layout-1.0.3.jar*. Program požaduje minimální rozlišení monitoru 1024 x 768 pixelů. Jeho vertikální velikost je pevně nastavena na 680 pixelů a při použití nižšího rozlišení by mohlo dojít k nesprávné funkci a některé prvky by mohly zůstat skryty.

XML generátor lze použít dvěma způsoby. Uživatel může jednak nakonfigurovat celé nádraží pouze v tomto programu a využít všechny jeho funkce nebo psát XML kód ručně a použít program jen ke generování některých bloků, případně pro doplnění stávajícího nádraží o nové bloky. Uživatel se také může tímto programem jen inspirovat, aby věděl, jak lze bloky navzájem propojit. Program tedy může sloužit jen jako nápověda při ručním psaní.

Tvorbu celého nádraží musíme začít stiskem tlačítka „Hlavička“, které vloží úvodní text XML souboru. Poté vybereme v černém obdélníku název bloku, který chceme vložit a nastavíme identifikační číslo tohoto bloku. Podle toho, jaký blok jsme zvolili, se zobrazily možnosti propojení s okolními bloky. V nabídce vybereme možnost, která odpovídá navrhovanému nádraží. Do kolonek vyplníme identifikační čísla připojených bloků. Pokud máme správně vyplněn celý blok, můžeme stisknout tlačítko „Vlož blok“, tím se vypíše XML podoba bloku do textové oblasti. Vložený text můžeme kdykoliv ručně editovat a změnit například čísla bloků nebo pozměnit názvy a komentáře. Tento postup opakujeme, dokud nevytvoříme celé vlakové nádraží.

Nakonec musíme XML soubor ukončit. To docílíme stisknutím tlačítka „Patička“. Ještě než soubor uložíme, můžeme soubor zkontrolovat a doplnit. Poté

vybereme v menu položku „Uložit jako“ nebo použijeme zkratku CTRL + S a vybereme nebo vepíšeme název souboru, pod kterým se má XML soubor uložit. Dialogové okno ukládání je na obrázku č. 13. Soubor s XML definicí nádraží přeložíme pomocí VHDL generátoru a výstupní VHDL soubor pak importujeme do projektu s definicemi bloků nádraží.



Obrázek č. 13: Dialogové okno pro ukládání

6.5 Testování

Program byl testován tvorbou XML souborů podle tvaru nádraží. Vygenerované XML bylo vloženo do internetového validátoru, který překontroloval tvar XML podle XSD souboru. Poté byl vložen do VHDL generátoru, který by v případě jakékoliv chyby nevytvořil výstup a vypsal by chybovou hlášku. Struktura XML souboru je lehce čitelná a tím pádem i lehce kontrolovatelná. Vytvořené soubory XML měly správnou strukturu a propojovaly bloky jen tak, jak je to dovoleno.

Všechny přeložené XML soubory do VHDL byly vloženy do nového projektu v programu Xilinx ISE 10.1. Tím byla zkontrolovaná i správná syntaxe souboru. Následně byla provedena syntéza celého projektu, která byla dokončena bez chyb. Pro kontrolu bylo vytvořeno několik vzorových nádraží.

Samostatné uživatelské rozhraní bylo dále testováno na funkčnost. Tlačítka, která mohla být stisknuta pouze jednou, se správně okamžitě zablokovala. Při stisku kombinace zpět se tlačítko správně uvolnilo. Po vložení patičky již nebylo možné do textu vkládat další bloky. Editace uživatelem zůstala povolena. Veškeré ovládací

prvky fungovaly korektně podle návrhu. Grafické rozhraní bylo spuštěno na operačním systému Windows XP SP3, na OS Linux OpenSuse a Gentoo. V operačním systému Linux se chybně zobrazovaly některé textové popisky. Chyba byla pravděpodobně způsobena požitím jiných typů písem. Prostor pro popisky byl tedy patřičně rozšířen, aby k problému nedocházelo.

6.6 Ukázková nádraží

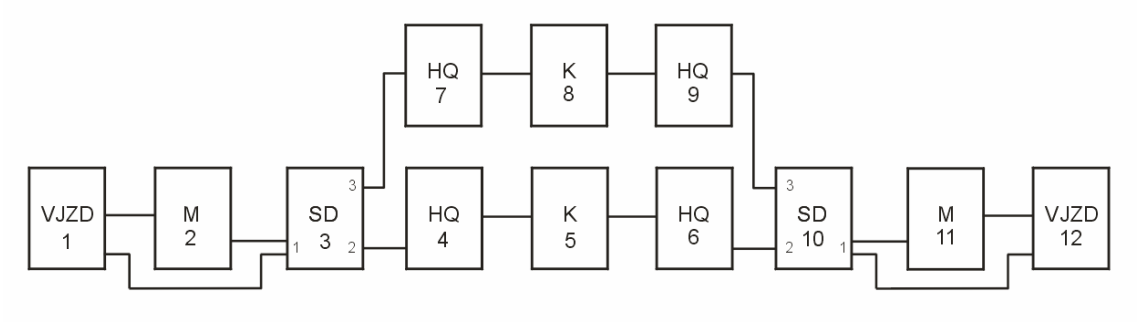
Pro demonstraci funkčnosti nástroje bylo vytvořeno několik vzorových nádraží. Zprvu z důvodu testování funkčnosti VHDL generátoru byla nádraží psaná ručně v textovém editoru. Později pro kontrolu byly k tvorbě nádraží využity oba generátory. Grafický editor značně urychlil práci při tvorbě nádraží. Stačilo pouze jednoduchým klikáním vkládat bloky nádraží a projekt byl hotov.

Nejdříve se vytvořila stejná nádraží, jako byla v původní diplomové práci. Jen s výjimkou, že nádraží byla generována automaticky a k propojení bloků sloužily skupiny vodičů. Byly použity přepracované bloky, které měly správné vstupy a výstupy.

Do nového projektu jsme vložili všechny soubory bloků, soubory balíčků a propojení. Všechny tyto soubor jsou na přiloženém CD v adresáři *„Bloky“*. Nakonec jsme vložili vygenerovaný soubor s rozmístěním bloků nádraží a projekt byl hotov.

6.6.1 Jednoduché nádraží

Prvním vytvořeným nádražím bylo základní nádraží se dvěma kolejemi. Nákres propojení bloků je na obrázku č. 14. Identifikační čísla bloků byla volena automaticky generátorem. Vytvořený soubor byl zkontrolován ve webovém validátoru. Definice v souboru XSD ukázaly, že vygenerovaný tvar XML je správný. Soubor byl použit generátorem VHDL a byl vytvořen soubor, který byl přidán do projektu se základními bloky kolejiště. Celý projekt byl zkontrolován na správnou syntaxi a syntetizován. Všechny operace proběhly v pořádku bez chyb. Vygenerované soubory XML a VHDL a celý projekt pro Xilinx ISE 10.1 se nacházejí na přiloženém CD v adresáři *„nadrazi1“*.

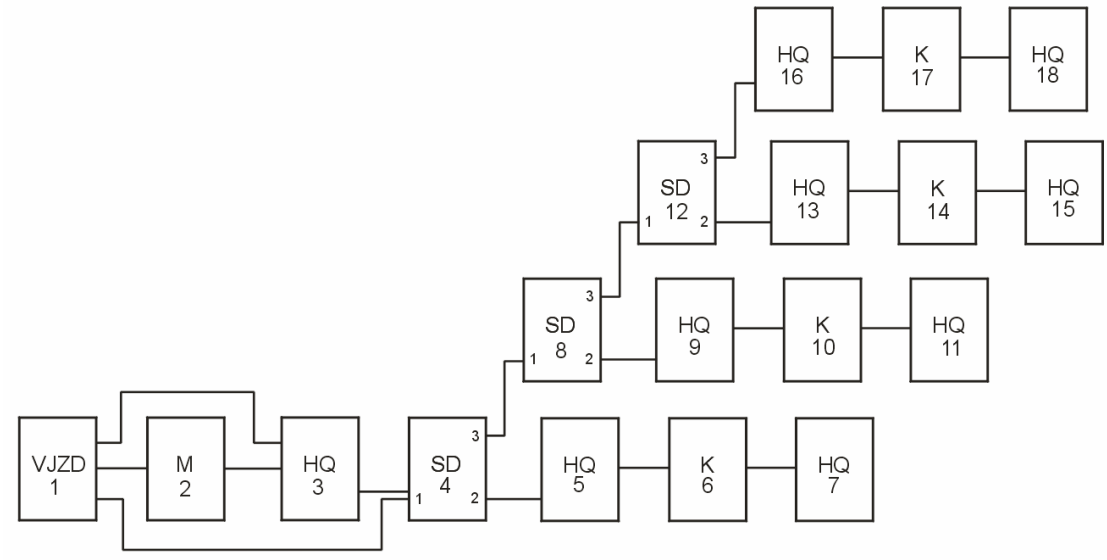


Obrázek č. 14: Jednoduché nádraží

6.6.2 Rozvětvené náraží

Abychom mohli ukázat plný rozsah obou generátorů, bylo vytvořeno složitější nádraží, které obsahovalo čtyři staniční koleje a jednu vjezdovou. Podobné nádraží bylo realizováno v původní diplomové práci. Jeho propojení na úrovni schematických symbolů muselo trvat poměrně dlouhou dobu. Vytvoření takového nádraží pomocí XML trvá jen několik minut a překlad do formátu VHDL je prakticky okamžitý.

Struktura XML byla vytvořena podle obrázku č. 15. Opět byla zkontrolována ve validátoru a výsledný VHDL soubor byl vložen do dalšího projektu v programu ISE. Všechny kontroly signálů prokázaly, že staniční zabezpečovací zařízení je propojeno správně. Všechny soubory spojené s tímto nádražím jsou opět na disku CD v adresáři *„nadrazi2“*.

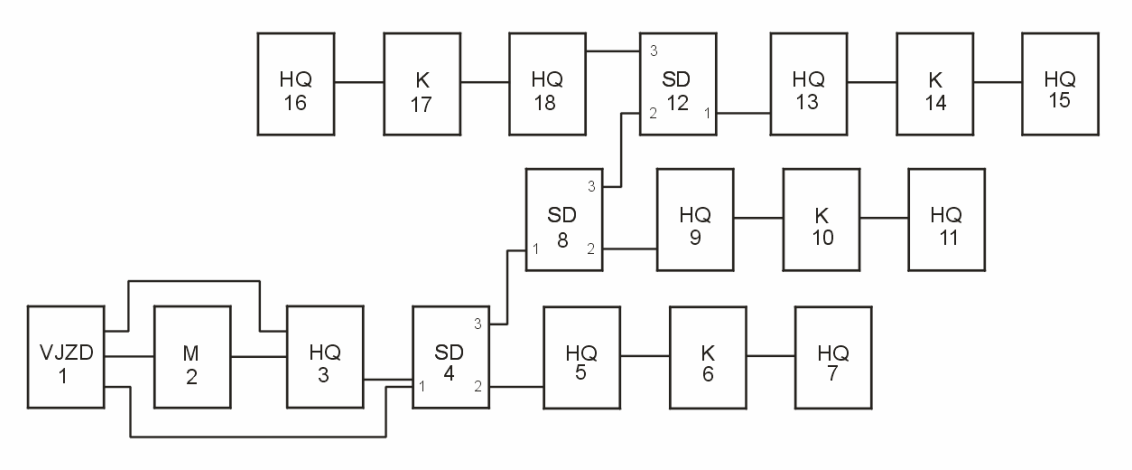


Obrázek č. 15: Rozvětvené nádraží

6.6.3 Nádraží s odvratovou kolejí

Vytvoření odvratové koleje vyžaduje složitější překlad do VHDL. Bylo nutné vyzkoušet, zda i tato část překladače pracuje správně. Nádraží obsahuje čtyři staniční koleje, jednu vjezdovou a tři bloky výhybek, kde výhybky 8 a 12 jsou propojeny odvratovým signálem. Bloky jsou propojeny podle obrázku č. 16. To znamená, že mohou být obě jen ve stavu rovně nebo ve stavu šikmo.

Kontrola nádraží byla provedena na všech úrovních tak, jak bylo u předchozích dvou. Na všechny podmínky nádraží vyhovělo. Soubory tohoto nádraží jsou na disku CD v adresáři *„nadrazi3“*.



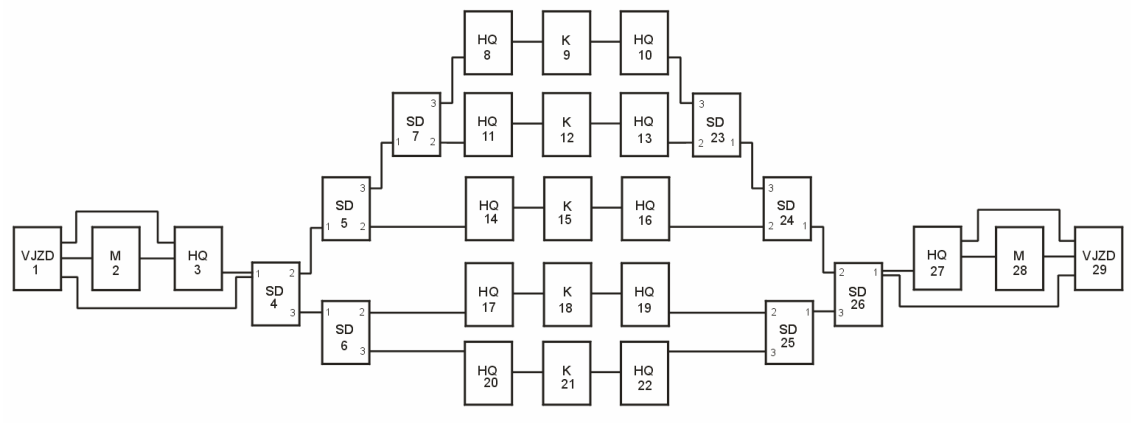
Obrázek č. 16: Kolejiště s odvratem

6.6.4 Nádraží v Rosicích nad Labem

K vytvoření tohoto vzorového nádraží byl autor inspirován při přestupování během cesty do Prahy. Vlakové nádraží v Rosicích nad Labem má pět staničních kolejí. Vlaky mohou vjíždět a vyjíždět oběma směry. V polovině staniční koleje se vlaky vždy rozdělují, jedna část směřuje na Chrudim a druhá na Hradec Králové. To by odpovídalo tomu, že k této koleji připadá blok K. Ostatní rozložení výhybek a návěstidel přibližně odpovídá skutečnosti.

Není jisté, zda nádraží v Rosicích nad Labem je opravdu složeno z podobných bloků v podobném pořadí jako je na obrázku č. 17. Je možné, že se nádraží v místech nepřístupných obyčejným cestujícím ještě někde dále dělí nebo obsahuje odvratovou

kolej. Berme toto nádraží spíše jako další vzorový příklad, který byl inspirován reálným nádražím. Soubory tohoto nádraží jsou na disku CD v adresáři *„nadrazi4“*.



Obrázek č. 17: Nádraží v Rosicích nad Labem

7 Závěr

Tato práce se zabývá tvorbou generátoru VHDL kódu popisujícího staniční zabezpečovací zařízení. Generátor byl navržen podle zadání a otestován při vytváření několika vzorových staničních zabezpečovacích zařízení pro různá nádraží. V průběhu tvorby výstupního souboru generátor kontroluje správné zapojení bloků a v případě nesrovnalostí lokalizuje chyby ve vstupním souboru. Generátor může být používán pro tvorbu libovolného staničního zabezpečovacího zařízení.

Při práci bylo upraveno pět základních stavebních bloků staničního zabezpečovacího zařízení. Bloky jsou připraveny pro jednodušší zabezpečení jejich funkčnosti. Rozdělení vstupních a výstupních signálů bylo provedeno podle způsobu zapojení bloků. Tím se zjednodušilo jejich ruční i automatické propojování. Všechny možné kombinace propojení bloků mezi sebou jsou v této práci zmapovány a detailně popsány včetně jejich zápisu ve VHDL.

Během práce byla vytvořena definice struktury XML souboru, který je využíván VHDL generátorem. V souboru XML je určeno, které bloky staniční zabezpečovací zařízení obsahuje a jak jsou mezi sebou propojeny. XML soubor je vytvořen přehledně a je čitelný pro běžného uživatele. Podle vzorových XML souborů lze sestavit libovolné zapojení staničního zabezpečovacího zařízení.

Pro ještě jednodušší vytváření staničního zabezpečovacího zařízení byla naprogramována grafická aplikace, jejímž výstupem je XML popis nádraží, který slouží jako vstupní soubor pro VHDL generátor. Pomocí tohoto nástroje byly sestaveny vzorové projekty.

V budoucnu by bylo vhodné zabezpečit funkčnost rozdělených částí bloků. Případně by bylo možné vytvořit grafický editor celého staničního zabezpečovacího zařízení. Pro tuto variantu je struktura XML souboru i VHDL generátoru připravena. Hlavní cíle této práce byly splněny.

8 Seznam použitých zkratek a symbolů

FPGA	Field Programmable Gate Array
VHDL	Jazyk určený k popisu hardware
XML	Extensible Markup Language
W3C	Wolrd Wide Web Consorcium
XSD	XML Schema Definition
LCD	Liquid crystal display
UART	Universal asynchronous receiver/transmitter
VJZD	Vjezdový blok
M	Blok bezvýhybkového úseku
HQ	Blok odjezdového návěstidla
SD	Výhybkový blok
K	Blok staniční koleje
OS	Operační systém
JVM	Java Virtual Machine

9 Seznam literatury

[1] M.ZATŘEPÁLEK: Zabezpečovací zařízení pro železniční stanici založené na FPGA, Diplomová práce, ČVUT, 2008

[2] VHDL - Wikipedie, the free encyclopedia
http://en.wikipedia.org/wiki/VHDL

[3] W3C XML Schema
http://www.w3.org/XML/Schema

[4] World Wide Web Consorcium - Wikipedie, the free encyclopedia
http://en.wikipedia.org/wiki/World_Wide_Web_Consortium

[5] XML Schema Validator
http://www.xmlme.com/Validator.aspx

10 Obsah přiloženého CD

- **Text diplomové práce:**
  - **dp.doc** – zdrojový soubor diplomové práce
  - **dp.pdf** – soubor ve formátu PDF

- **Zdrojové soubory:**
  - **Bloky** – složka s přepracovanými bloky kolejiště
    - **HQ_automat.vhd** – soubor se stavovým automatem
    - **HQ_citac.vhd** – soubor s čítačem
    - **HQ_logika.vhd**– soubor s logickým obvodem
    - **HQ_top.vhd** – top modul celého bloku
    - **M_automat.vhd** – top modul celého bloku
    - **SD_automat.vhd** – soubor se stavovým automatem
    - **SD_logika.vhd**– soubor s logickým obvodem
    - **SD_top.vhd** – top modul celého bloku
    - **VJZD_citac.vhd** – soubor s čítačem
    - **VJZD _logika.vhd**– soubor s logickým obvodem
    - **VJZD _top.vhd** – top modul celého bloku
    - **K_automat.vhd** – top modul celého bloku
    - **rychlosti_gen.vhd** – generátor rychlostí pro blok SD
    - **balik_top.vhd –** balíček se skupinami vodičů
    - **connects.vhd –** balíček s propojením mezi bloky
    - **package_HQ.vhd** – balíček propojení bloku HQ
    - **package_M.vhd** – balíček propojení bloku M
    - **package_K.vhd** – balíček propojení bloku K
    - **package_SD.vhd** – balíček propojení bloku SD
    - **package_VJZD.vhd** – balíček propojení bloku VJZD

  - **nadrazi1** – složka s projektem nádraží v Xilinx ISE 10.1
    - **nadrazi.xml**
    - **nadrazi.zip**

- **nadrazi2** – složka s projektem nádraží v Xilinx ISE 10.1
  - **nadrazi2.xml**
  - **nadrazi2.zip**
- **nadrazi3** – složka s projektem nádraží v Xilinx ISE 10.1
  - **nadrazi3.xml**
  - **nadrazi3.zip**
- **nadrazi4** – složka s projektem nádraží v Xilinx ISE 10.1
  - **nadrazi4.xml**
  - **nadrazi4.zip**

- **genVHDL** – složka s generátorem VHDL
  - **dip1.jar** – spustitelný soubor generátoru
  - **dip1.zip** – zkomprimovaný projekt z NetBeans

- **genXML** – složka s generátorem XML
  - **gui1.jar** – spustitelný soubor generátoru
  - **gui1.zip** – zkomprimovaný projekt z NetBeans
  - **lib** – složka potřebná ke spuštění simulátoru
    - **swing-layout-1.0.3.jar** – soubor potřebný k funkci grafického rozhraní

- **XML** – složka se vzorem a definicí XML
  - **nadrazi.xml** – ukázkové nádraží v XML
  - **xmlDefinitions** – složka s definicí XML
    - **pavel.xsd –** definice XML souboru