TECHNICKÁ UNIVERZITA v LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 - Informační technologie

Studijní obor: 1802R007 - Informační technologie

Editor konečných stavových automatů

Finite state machine editor

Bakalářská práce

Autor: Rudolf Byšický

Vedoucí BP: Ing. Martin Rozkovec, PhD.

V Liberci 17.5.2012

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2013/2014

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Rudolf Byšický**
Osobní číslo:	**M09000119**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Editor konečných stavových automatů**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Seznamte se s konečnými stavovými automaty a jejich reprezentací v jazyce VHDL.
2. Seznamte se s jazykem C#, XAML a platformou .NET.
3. Vytvořte grafický editor stavových automatů s možností dynamické editace počtu stavů, vstupní a výstupní abecedy a přechodové a výstupní funkce.
4. Automaty ukládejte ve vlastním formátu a v jazyce VHDL. Provažte funkci editoru s nástroji z balíku Xilinx ISE/

Rozsah grafických prací:	**Dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 30 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Pinker J., Poupa M.: Číslicové systémy a jazyk VHDL, BEN 2006, ISBN 80-7300-198-5**

[2] **Petzold Charles: Mistrovství ve Windows Presentation Foundation, BEN 2008, ISBN 978-80-251-2141-2**

[3] **Xilinx Inc.: Command Line Tools User Guide (ug628), 2012, online, http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_2/devref.pd**

[4] **Kocur, Pavel: Úvod do teorie konečných automatů a formálních jazyků, ZČU v Plzni 2001, ISBN 80-7082-813-7**

Vedoucí bakalářské práce:	**Ing. Martin Rozkovec, Ph.D.** Ústav informačních technologií a elektroniky
Datum zadání bakalářské práce:	**12. září 2013**
Termín odevzdání bakalářské práce:	**16. května 2014**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
vedoucí ústavu

V Liberci dne 12. září 2013

Prohlášení

Byla jsem seznámena s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracovala samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

V Liberci dne 17.5.2013

….....

Podpis

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé bakalářské práce Ing. Martinu Rozkovci PhD. za rady čas a především trpělivost, které mé práci věnoval.

A dále bych chtěl také poděkovat svým rodičům a celé rodině a také své přítelkyni za podporu při studiu.

Abstrakt

Bakalářská práce se zabývá konečnými stavovými automaty a jejich reprezentací v jazyce VHDL a pomocí klopných obvodů. Výsledkem práce je grafický editor stavových automatů, který může převést grafický návrh na VHDL soubor. Dále je možno z tohoto souboru vygenerovat bitstream soubor a ten nahrát do FPGA obvodu.

Klíčová slova:

VHDL, stavové automaty, FPGA, klopné obvody

Abstract

This work deals with finite-state machines and their representation in VHDL language and using flip-flop circuit. Result of this work is graphic editor of state machines, which can convert graphic design to VHDL file. It is also possible to generate a bitstream file and load it into an FPGA.

Key words:

VHDL, state machine, FPGA, flip-flop circuit

Obsah

Seznam zkratek

.NET - dotnet

CLB - Configurete Logic Block

CPLD - Complex Programmable Logic Device

ESD - Elektostatic discharge

HDL - Hardware Description Language

ISE - Integrated Software Environment

LUT - Look Up Table

PROM - Programmable Read Only Memory

VGA - Video Graphics Array

VHDL - Hardware Description Language

VHSIC - Very High Speed Integrated Circuits

WPF - Windows Presentation Foundation

XML - Extensible Markup Language

Úvod

Pomocí konečných stavových automatů lze popsat mnoho problémů a příkladů z reálného života. Příkladem může být obyčejný turniket. Stavový automat lze popsat pomocí jazyku pro popis číslicových systémů VHDL (z anglického *Very high speed integrated circuits Hardware Description Language*), jenž dnes velice používaným nástrojem a dále z něj vygenerovat bitstream a ten následně nahrát do FPGA (z anglického *Field programmable gate array*) obvodu.

Bakalářská práce se věnuje návrhu grafického rozhraní, které umožňuje jednoduché vytváření návrhů stavových automatů a jejich převedení do VHDL a následně i do bitsteamu, které je možné nahrát do FPGA obvodu. První kapitola se věnuje stavovým automatům a jejich základním dělením. Ve druhé kapitole je obecný popis FPGA a programu ISE(*Integrated Software Environment*). Třetí kapitola obsahuje seznámení s jazykem VHDL a jeho základní syntaxí. Čtvrtá kapitola je věnována kombinačním a sekvenčním obvodům, pomocí kterých je možné stavové automaty také popsat. Páta kapitola je věnována navrženému programu Autonima a popisuje jeho základní funkčnost.

1. Konečné stavové automaty

Konečný stavový automat, zkráceně pouze stavový automat, je teoretický matematický model používaný v informatice pro studium vyčíslitelnosti a obecně formálních jazyků. Je koncipován jako abstraktní stroj, který může být v jednom z konečného počtu stavů. Automat nemůže být ve více stavech zároveň. Přesun mezi jednotlivými stavy se nazývá přechod. Ten nastává aktivací událostí nebo určitých podmínek.

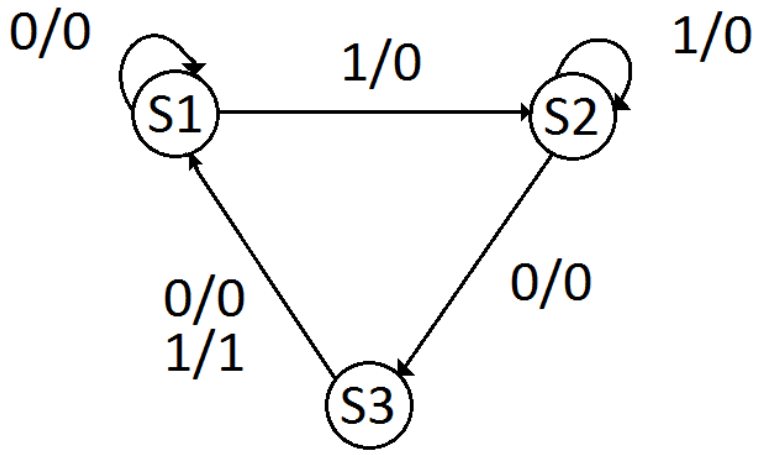
Matematicky je stavový automat definován jako uspořádaná pětice(S, Σ, σ, s, a), kde:

S je konečná neprázdná množina stavů
Σ je konečná neprázdná množina vstupních symbolů
σ je přechodová funkce, popisující pravidla přechodů mezi stavy
s je počáteční stav, s $\subseteq S$
A je množina přijímajících stavů, $a \subseteq S$

Stavový automat je velice jednoduchý výpočetní model, jehož množina stavů je konečná. Nemá žádnou další paměť, kromě informací o aktuálním stavu. v informatice se rozlišují dva hlavní typy automatu a to Mealyho automat a Mooreův automat.

1.1 Mealyho automat

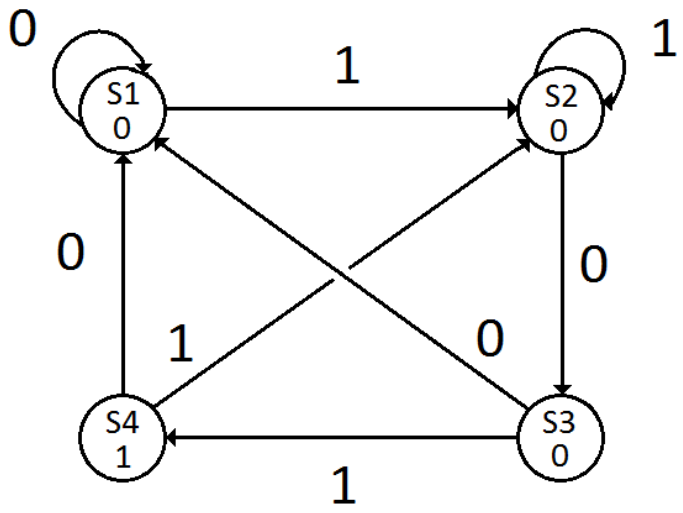
V informatice se pod pojmem Mealyho automat rozumí konečný stavový automat, jehož výstupní funkce je závislá jak na aktuálním stavu, tak na vstupu. To znamená, že ke každý přechod má přiřazenou nejen vstupní hodnotu, kterou je spuštěn, ale i výstupní hodnotu, která se generuje při aktivaci přechodu.



Obrázek 1.1 Příklad Mealyho automat zachytávající sekvenci 101

1.2 Mooreův automat

Automat typu Moore je konečný stavový automat, jehož výstup je závislý pouze na aktuálním stavu, ve kterém se nachází. Změna na vstupu se u něj projeví na výstupu až v následujícím stavu.



Obrázek 1.2 Příklad Mooreův automat zachytávající sekvenci 101

2. FPGA

FPGA (Field Programmable Gate Array) neboli programovatelná hradlová pole, jsou speciální číslicové integrované obvody, které jsou navrženy tak, aby je mohl konfigurovat zákazník. Obsahují různě složité programovatelné bloky propojené konfigurovatelnou maticí spojů.

V dnešní době obvody FPGA nacházejí uplatnění díky svému snadnému návrhu, flexibilitě, programovatelnosti a snižující se spotřebě energie. Typicky se používají tam, kde se nevyplatí návrh integrovaného obvodu a současně konvenční řešení systému s procesorem není vhodné. Moderní programovatelná hradlová pole dnes umožňují i implementaci komplikovaných procesorů nebo celých systémů na čipu.

Mimo obvodů FPGA jsou dále rozšířené například obvody CPLD (*Complex Programmable Logic Device*). Ty se hodí především na jednodušší aplikace.

Obrázek 2.1 FPGA typu Spartan

2.1 Bitstream

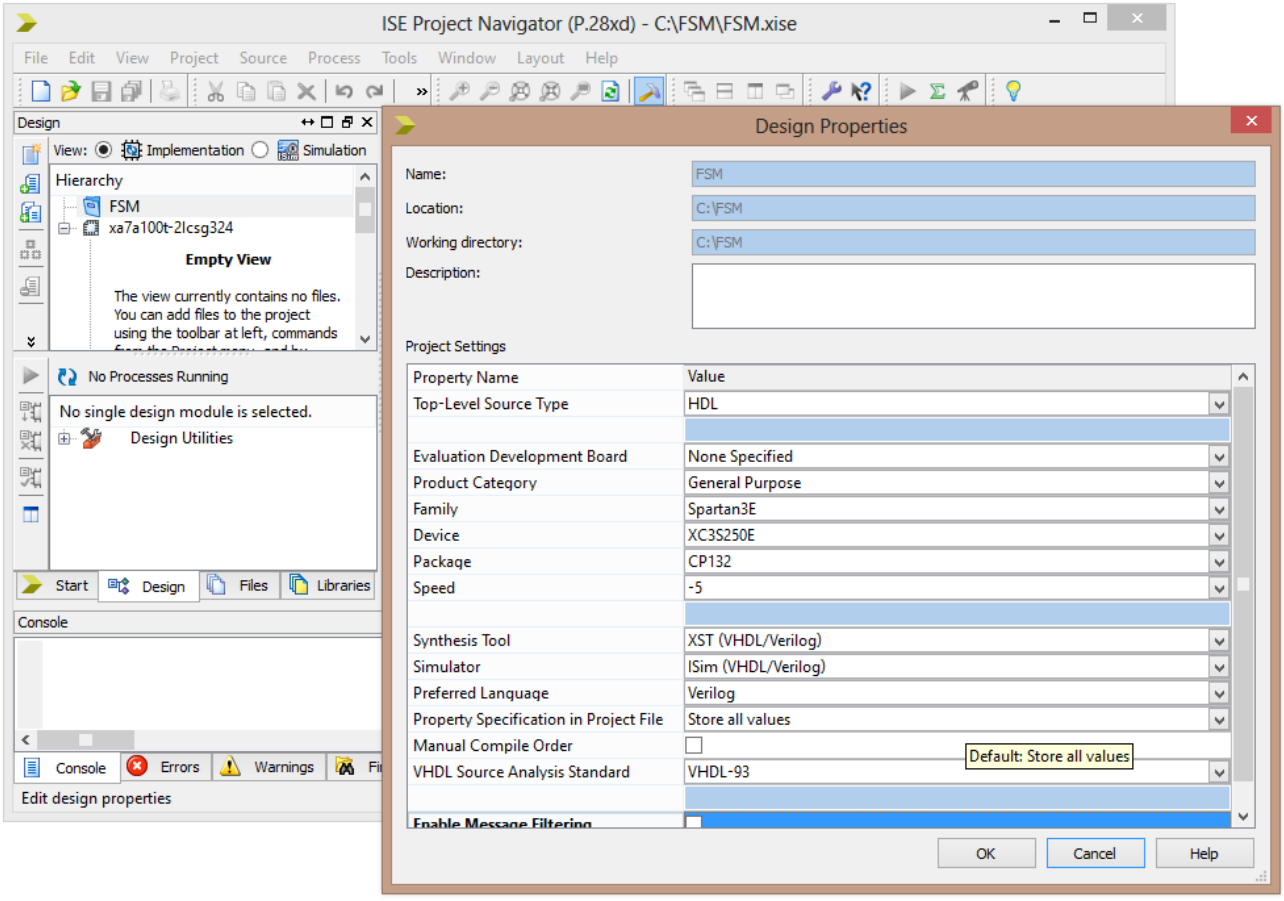
Bitstream nebo bitový tok je časová řada nebo posloupnost bitů. Bytestream je série bajtů, typicky po 8 bitech, a může být považováno za speciální případ bitového proudu.

Termín bitstream je často používán pro popis konfiguračních údajů, které budou načteny do programovatelné hradlové pole. Toto použití vzniklo na základě společných metod pro konfiguraci FPGA ze sériového bitového proudu, typicky ze sériové PROM(*Programmable Read Only Memory*) nebo čipu flash paměti, i když většina FPGA podporuje také byte-paralelní načítání. Podrobný popis formátu bitového proudu pro konkrétní FPGA čip obvykle najdeme na stránkách dodavatele daného FPGA [4].

2.2 ISE

ISE (Integrated Software Environment) je softwarový nástroj, vyrobený firmou Xilinx pro syntézu a implementaci HDL (*Hardware Description Language)* vzorů, což umožňuje vývojářům syntetizovat své návrhy, provádět analýzy načasování, zkoumat RTL diagramy, simulovat reakci designu na různé podněty a nakonfigurovat cílové zařízení programátorem.

Web Edition je bezplatná verze Xilinx ISE, kterou lze stáhnout zdarma. Poskytuje syntézu a programování pro omezený počet Xilinx obvodů. Zejména jsou nedostupná zařízení s velkým počtem vstupních a výstupních pinů. Rodina Spartan je tímto vydáním plně podporována , stejně jako všechna CPLD, což znamená, že vývojáři a vzdělávací instituce jsou ušetřeny od režijních nákladů na vývoj softwaru.

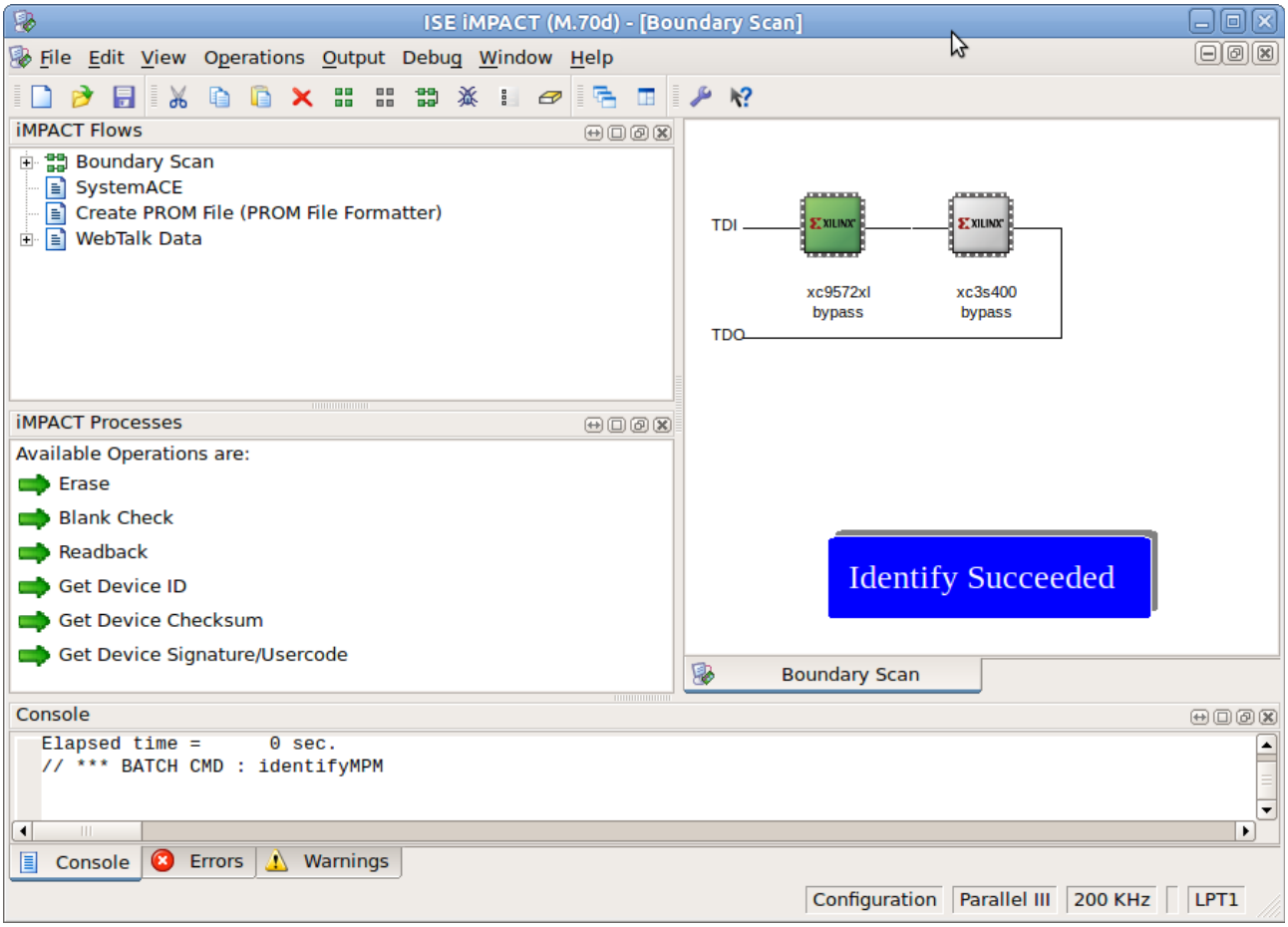


Obrázek 2.2 Ukázka vývojového prostředí ISE

2.3 ISE iMPACT

Program ISE iMPACT slouží k nahrání vygenerovaného bitstreamu do FPGA obvodu. Je součástí všech verzí programu ISE i jeho bezplatné verze Web Edition.

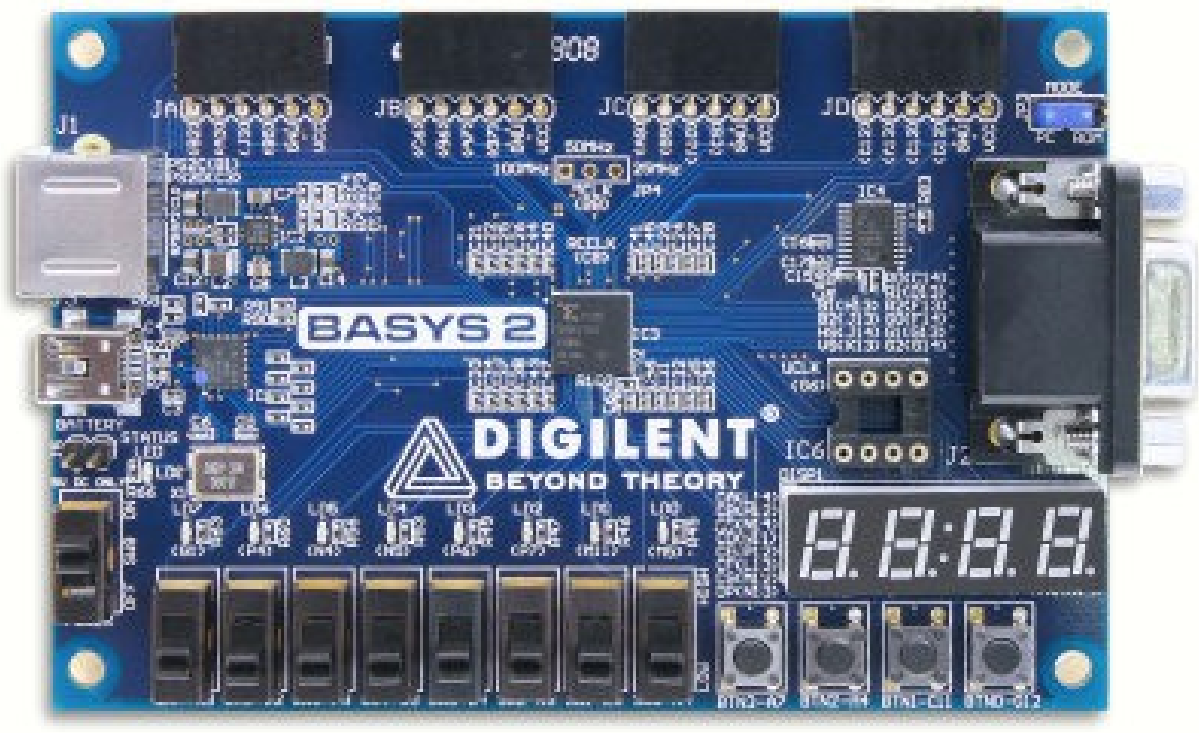
Obsluha programu je velice jednoduchá a postup nahrávání bitstreamu je detailně popsán v nápovědě programu.



Obrázek 2.3 Ukázka programu iMPACT

2.4 Basys 2

Basys 2 je vývojová deska, kterou může kdokoliv použít k získání zkušeností s tvorbou reálných číslicových obvodů. Tato deska obsahuje hradlové pole typu Spartan-3E a Atmel AT90USB2 USB řadič. Basys 2 poskytuje kompletní, předpřipravený hardware, vhodný pro testování a zkoušení základních i složitějších logických obvodů.



Obrázek 2.4 Ukázka vývojové desky Basys 2

Čtyři standardní rozšiřující konektory umožňují doplnit vybavení desky o rozšiřující moduly, například o pokusné, uživatelem navržené desky plošných spojů, nebo výrobcem dodávané moduly s názvem Pmod. Pmod jsou levné, analogové a digitální vstupní a výstupní moduly, které nabízejí analogově-digitální a digitálně-analogové převody, senzorové vstupy a mnoho dalších funkcí.

Signály na 6-pinových konektorech jsou chráněny proti ESD (*Elektostatic discharge*) poškození a zkratu. To zajišťuje dlouhodobou životnost v jakémkoliv prostředí. Kromě 6-pinových konektorů deska dále obsahuje osm posuvných spínačů, čtyři tlačítka, čtyři sedmisegmetovky, VGA(Video Graphics Array) výstup a PS/2 konektor.

Deska Basys 2 pracuje se všemi verzemi Xilinx ISE, včetně volného WebPacku. Na propojení s počítačem je potřeba pouze kabel typu USB Mini-A. Ten slouží zároveň i k napájení desky.

2.5 CLB

Konfigurovatelné logické bloky CLB (*Configurete Logic Block*), tvoří hlavní logický prostředek pro implementaci synchronních, stejně jako kombinačních obvodů. Každý CLB blok obsahuje čtyři *slice* bloky a každý *slice* obsahuje dva LUTy (Look Up Table). LUT lze také použít

jako paměť 16x1(RAM16) nebo jako 16-bitový posuvný registr(SRL16), multiplexor, a také umožňuje provádět a zjednodušovat aritmetické funkce.

Většina logiky v návrhu se automaticky mapuje na řetěz zdrojů v CLB. Podrobnosti o CLB zdrojích jsou užitečné při odhadování počtu opakovaných zdrojů, potřebných pro podání žádosti, nebo při optimalizaci designu architektury.

CLB jsou uspořádány v pravidelném poli v řádcích a sloupcích, jak je znázorněno na obrázku 2.4.

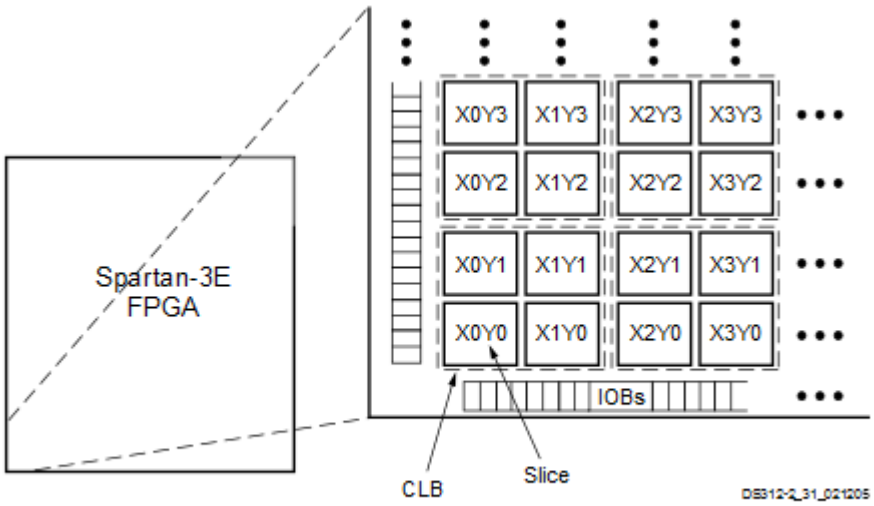


Figure 5-1: **CLB Locations**

Obrázek 2.4 Vnitřní struktura FPGA

Pro lepší představu zmíním, že například FPGA obvod Spartan3E-250, obsahuje 34 řádků CLB bloků a 26 sloupců, celkově tedy 612. Těchto 612 CLB bloků obsahuje 2448 *slice* a 4896 LUT bloků.

Každé CLB je totožné v rámci rodiny daného FPGA. Výkon se mírně liší mezi rodinami, kvůli drobným odchylkám při zpracování.

3. Jazyk VHDL

VHDL (*VHSIC Hardware Description Language*) je jazyk pro popis hardwaru, speciálně navržený pro popis a simulaci rozsáhlých číslicových obvodů a systémů. VHDL může být použito jako univerzální paralelní programovací jazyk. v dnešní době je to jeden z nejpoužívanějších jazyků pro popis číslicových systému implementovaných na hradlových polích. Umožňuje návrh jak kombinačních, tak i sekvenčních struktur a hlavní výhodou je jeho univerzálnost. Implementace navržené struktury je závislá až na syntéze VHDL kódu. Díky tomu lze pomocí tohoto jazyka provádět návrhy pro hradlová pole většiny výrobců (Altera, Lattice, Xilinx, apod.) a poté použít vhodný syntézní nástroj, jenž je obvykle ve volné verzi dostupný na internetových stránkách výrobců.

Návrh struktury je vhodné členit na jednotlivé bloky, které jsou spojeny ve finálním modulu pomocí signálů, které většinou odpovídají reálnému elektronickému signálu. Tento postup je v podstatě adekvátní s propojováním skutečných obvodů v reálu.

3 .1 Historie jazyka VHDL

Kromě jazyka VHDL existují i jiné jazyky pro popis číslicových systémů. Jedním z nich je Verilog. Standard jazyka Verilog byl poprvé publikován v roce 1995 pod označením *IEEE Standard Hardware Description Language Based on the Verilog. v* roce 2001 byla pak publikována jeho první revize pod označením *IEEE Std 1364-2001.* Jazyk Verilog má podobnou syntaxi jako jazyk C a má podobné vyjadřovací schopnosti jako má jazyk VHDL. Společně s VHDL je dnes Verilog používán k návrhu číslicových systémů (např. procesorů, čipsetů, programovatelných logických obvodů, aj.). v dnešní době většina návrhových a simulačních systémů podporuje oba jazyky. Proto není vyjímkou že návrh číslicového systému se může skládat z částí napsaných v jazyce Verilog i VHDL. Verilog je v dnešní době rozšířen především ve Spojených státech amerických, zatímco VHDL využívají hlavně firmy a univerzity v Evropě.

V dnešní době se rozvíjejí jazyky určené pro popis číslicových systémů, které jsou založeny na jazyce C nebo C++. Mezi ně patří například SystemC, který je založen na objektovém jazyku C++ a Handel-C , založený na C. Tyto jazyky umožňují přímou syntézu do cílové technologie, nebo syntézu založenou na překladu do RTL popisu systému v jazycích VHDL nebo Verilog a následnou syntézu do cílové technologie.

3.2 Základní struktura jazyka VHDL

Model v jazyku VHDL má dvě základní části: deklaraci entity a tělo (popis) architektury. Deklarace entity popisuje vstupy a výstupy konstrukce, popis architektury definuje její funkci. Mezery, tabulátory a znaky nového řádku slouží jako oddělovače, nemají další význam. Můžeme je tedy dle libosti použít k rozčlenění a zpřehlednění kódu.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Multiplexor is
   port(
      a, b, c, d, e, f, g, h : in std_logic;
      X: in std_logic_vector(2 downto 0);
      Y: out std_logic);
end entity Multiplexor;


architecture Behavioral of Multiplexor is
   begin
      process(a, b, c, d, e, f, g, h, X) is
         begin
            case X is
               when "000" => Y <= a;
               when "001" => Y <= b;
               when "010" => Y <= c;
               when "011" => Y <= d;
               when "100" => Y <= e;
               when "101" => Y <= f;
               when "110" => Y <= g;
               when others => Y <= h;
            end case;
         end process;
end architecture Behavioral;
```

Příklad 3.1 Popis multiplexoru v kódu VHDL

3.2.1 Deklarace entity

Číslicové systémy se obvykle navrhují jako hierarchická kolekce modulů. Každý z nich má definovanou množinu portů, jenž představují vstupní a výstupní rozhraní. Porty jsou deklarovány v závorce klíčovým slovem *PORT*. Jsou to datové objekty a definují odpovídající signály. Deklarace se skládá ze jména identifikátoru příslušného signálu, a poté z určení směru jejího přenosu a typu dat, představující daný signál. Porty mohou být v jedno z následujících módů:

- IN – vstup (data lze z portu pouze číst)

- OUT – výstup (výstupní signál nemůže být použit jako vstup uvnitř entity)
- BUFFER – výstup se zpětnou vazbou (z něj signál může být použit uvnitř entity)
- INOUT – obousměrný tok
- LINKAGE – neznámý směr (návaznost na jiné než VHDL modely, např. Verilog)

Identifikátor entity VHDL musí splňovat striktní pravidla, například že identifikátor musí být unikátní, nesmí obsahovat mezeru, musí začínat písmenem nebo že nelze použít 2 podtržítka za sebou.

```
entity NewProjekt is
     port(
          X : in STD_LOGIC;
          Y3 : out STD_LOGIC;
          Y2 : out STD_LOGIC;
          Y1 : out STD_LOGIC;
          Y0 : out STD_LOGIC;
          CLK : in STD_LOGIC;
          RST : in STD_LOGIC
     );
end entity NewProjekt;
```

Příklad 3.2 Ukázka entity

3.2.2 Tělo architektury

Po deklarování entity se všemi jejími náležitostmi je třeba deklarovat architekturu. Ta popisuje chování systému. Architektura může být popsána různými styly. Pojem stylu není v jazyku VHDL přímo definován. Nejčastější styly jsou behaviorální, strukturální a styl popisující tok dat.

Behaviorální styl je takový, jenž je složen z jednoho či více procesů. Jednotlivé procesy jsou popsány až na úroveň algoritmu. Oproti tomu strukturální styl obsahuje pouze instance komponent. v praxi se často používají oba přístupy i v rámci jedné architektury.

3.2.3 Proces

Proces je základním stavebním prvkem při popisu sekvenční struktury. Jde o samostatné části kódu a může jich být libovolný počet. Proces se stává aktivním pouze v okamžiku, kdy dojde ke změně citlivých proměnných, které jsou v procesu definovány.

Příkazy uvedené uvnitř procesu se vykonávají sekvenčně, podobně jako v běžných programovacích jazycích. Navenek se však proces chová jako paralelní příkaz.

Syntaxe:

jméno_procesu: **PROCESS**(citlivostní proměnné)

--deklarace proměnných a signálů

BEGIN

--tělo procesu

END;

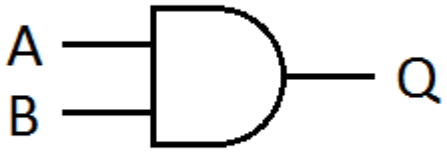
4. Kombinační a sekvenční obvody

Kombinační a sekvenční obvody jsou elektronické obvody, pomocí kterých jsme schopní sestavit libovolný číslicový obvod. Pomocí Karnaughových map a Booleovy algebry lze kombinační část obvodu zjednodušit, aby bylo použito co nejméně hradel.

4.1 Kombinační obvody

Kombinační logické obvody jsou obvody, jejichž výstupní funkce je závislá pouze na okamžitých vstupních kombinacích a nezávisí na předchozích hodnotách. Logické obvody nemají žádnou paměť, jedné kombinaci vstupních proměnných odpovídá právě jedna výstupní kombinace.

Závislost vstupních a výstupních funkčních hodnot popisujeme pomocí logických výrazů nebo pomocí pravdivostní tabulky. Ta se vzhledem ke své snadné čitelnosti používá velmi často. Pravdivostní tabulka obsahuje ve sloupcích vlevo vstupní logické proměnné a v pravé části výstupní funkci, případně více funkcí. Počet řádků v tabulce odpovídá počtu všech možných kombinací vstupních proměnných.



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Obrázek 4.1 Hradlo AND a jeho pravdivostní tabulka

4.2 Sekvenční obvody

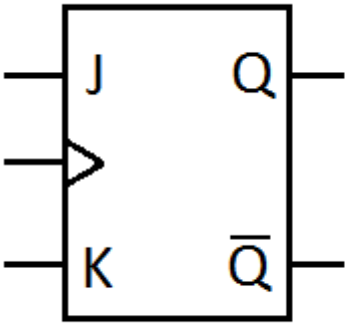
Sekvenční obvody se liší od kombinačních tím, že jejich výstupní funkce je závislá nejen na současných hodnotách vstupních signálů, ale ještě i na jeho vnitřní proměnné, vnitřním stavu. Tyto proměnné jsou uchovávány v paměťových členech. Jejich existence způsobuje, že stejné hodnoty vstupních proměnných na vstupu, nevyvolají vždy stejnou odezvu na výstupu z obvodu.

Paměťová část obvodu je tvořena kombinačním obvodem, ve kterém je zavedena zpětná vazba neboli zpětnovazební smyčka. Toto zapojení se nazývá bistabilní klopný obvod. Jeho úkolem

je převzít informaci ze vstupu a uchovat ji, i když vstupní informace již zmizí.

Sekvenční obvody se dělí na asynchronní a synchronní. u asynchronních obvodů působí změna na vstupu okamžitě výstup. Zpoždění výstupu je dáno jen průchodem jednotlivými logickými členy. Takovýto obvod může reagovat na podnět vstupu velmi rychle, ale v rozsáhlých logických obvodech může však různě velkým hodnotám zpoždění. To může vést ke vzniku *hazardních stavů.*

Synchronní obvody jsou synchronizovány samostatnými signály, jenž se nazývají synchronizační, nebo hodinové a určují jednotlivé takty. Obvod nemění stav na výstupu ihned po změně na vstupu, ale až po změně synchronizačního signálu. Mezi synchronní obvody patří například čítače a některé klopné obvody.



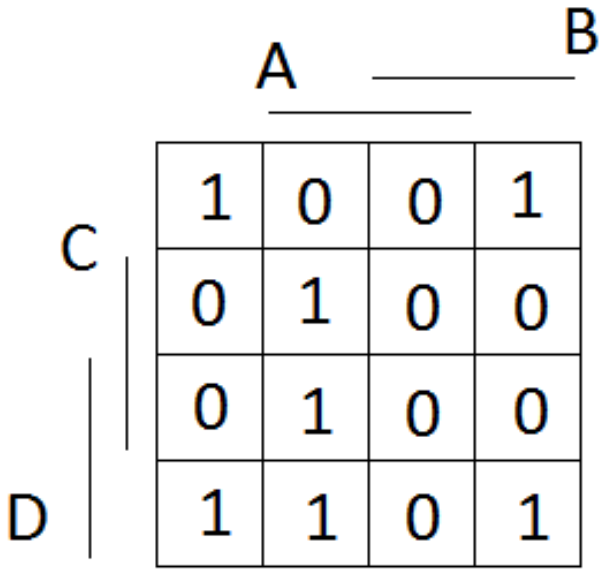
J	K	Q_{n+1}	$\overline{Q}_{n+1}$
0	0	Q_n	$\overline{Q}_n$
0	1	0	1
1	0	1	0
1	1	$\overline{Q}_n$	Q_n

Obrázek 4.2 Synchronní obvod JK

4.3 Karnaughova mapa

Karnaughova mapa se používá k zjednodušení Booleovy algebry, která modeluje vlastnosti množinových a logických operací. Mapa redukuje počet rozsáhlých výpočtů, čímž právě minimalizací vstupní funkce dosáhneme toho, že při její realizaci budeme potřebovat méně logických prvků jako například hradla AND, OR, NAND, aj.

Mapa je tabulka, jenž obsahuje tolik políček, kolik je kombinací vstupních proměnných (2^n ,kde *n* je počet vstupních proměnných). Jednotlivá políčka v tabulce odpovídají jedné z možných kombinací a zapisujeme do něj odpovídající funkční hodnotu. Hlavní vlastností Karnaughovy mapy je, že sousední políčka se od sebe liší hodnotou jediného vstupního argumentu.

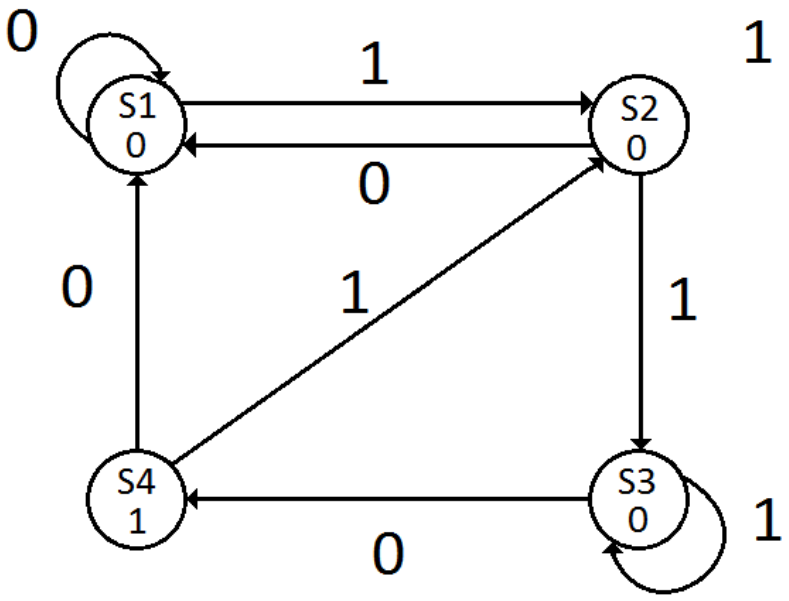


Obrázek 4.3 Ukázka Karnaughovy mapy

4.4 Stavový automat pomocí klopných obvodů

Jak již bylo řečeno výše, stavové automaty jde realizovat i pomocí kombinačních a klopných obvodů. v této podkapitole bude vysvětlen postup jak automat pomocí těchto obvodů navrhnout. Konkrétně se bude jednat o konečný stavový automat typu Moore, který zachytává posloupnost bitů 110.

V prvním kroku je třeba správně daný stavový automat nakreslit, viz obrázek 4.4. Jedná se o automat typu Moore, takže konkrétní výstup závisí pouze na aktuálním stavu.



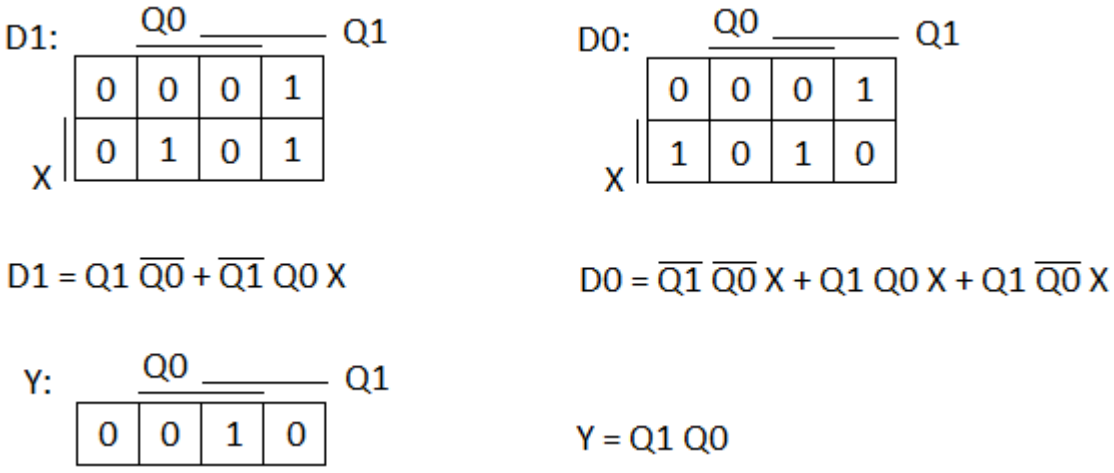
Obrázek 4.4 Automat typu Moore zachytávající sekvenci 110

Po grafickém návrhu je třeba sestavit přechodovou tabulku. Tu sestavíme právě pomocí návrhu, ze kterého vyčteme většinu hodnot. Hodnoty *Q1 a Q0* binárně zakódovávají daný stav. Toto kódování posléze použijeme, abychom doplnily sloupce *D1 a D0*. Tyto hodnoty se získávají tak, že binárně zakódujeme stav, který se nachází ve sloupci *St+1*. Sestavená přechodová tabulka je zobrazena na obrázku 4.5.

St	Q1	Q0	X	St+1	D1	D0	Y
S0	0	0	0	S0	0	0	0
			1	S1	0	1	
S1	0	1	0	S0	0	0	0
			1	S2	1	0	
S2	1	0	0	S3	1	1	0
			1	S2	1	0	
S3	1	1	0	S0	0	0	1
			1	S1	0	1	

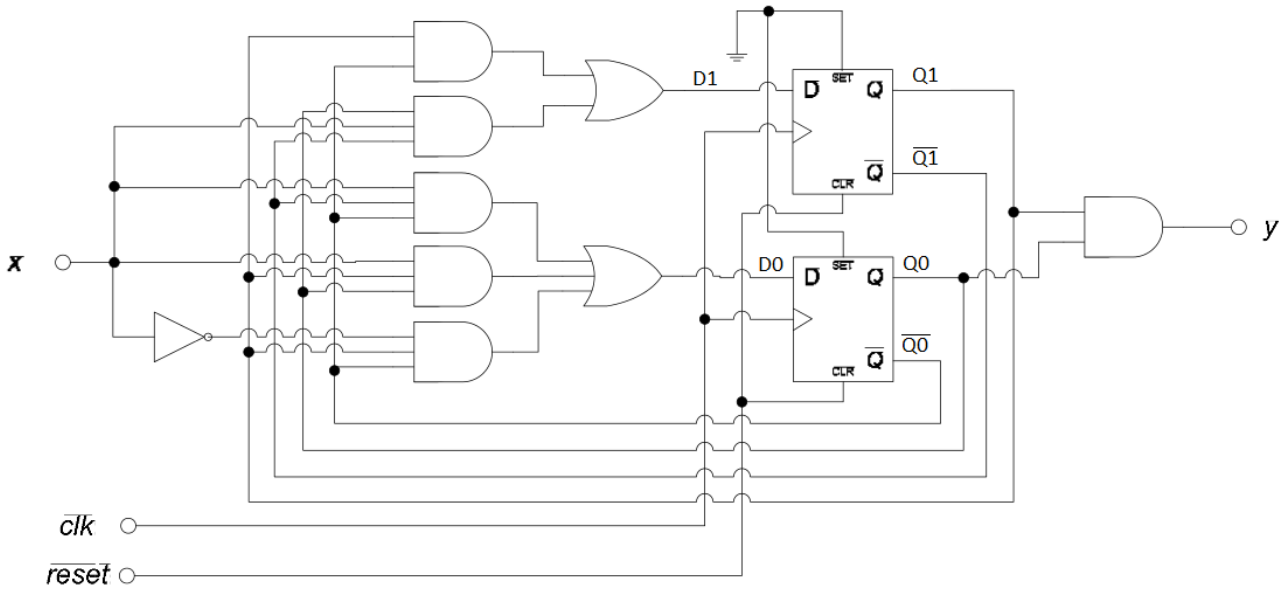
Obrázek 4.5 Přechodová tabulka

Ze sestavené přechodové tabulky lze sestavit Karnaughovy mapy pro jednotlivé D klopné obvody. Výsledek minimalizace těchto map bude udávat, co do jednotlivých odvodů máme přivést na vstup, abychom dostali požadovaný výstup. Karnaughovy mapy pro zadaný příklad je možné vidět na obrázku 4.6.



Obrázek 4.6 Karnaughovy mapy pro daný příklad

Po získání hodnot z Karnaughových map je možné obvod pospojovat pomocí hradel a kopných obvodů tak, jak je možné vidět na obrázku 4.6.



Obrázek 4.6 Mooreův automat zachytávající bitovou sekvenci 110

Tento postup navrhování a sestavování stavového automatu je ovšem velmi časově náročná. Navíc i velmi náchylný na chyby u tvorby tabulek. Zvláště u složitějších a komplexnějších automatů. Proto byl navrhnut program Autonima.

5. Program Autonima

Program Autonima slouží jako grafický editor stavových automatů. Dále je možno tento grafický návrh zpracovat do VHDL kódu, případně poté dále do bitstreamu, který je možné nahrát do FPGA obvodu. Pro vygenerování bitstreamu je nutné mít nainstalované syntézní a implementační nástroje od společnosti Xilinx, jelikož program na tvorbu bitstreamu si volá určité programy právě z balíku ISE. Program Autonima obsahuje možnost zobrazení vytvořeného automatu pomocí přechodové tabulky. Pomocí této tabulky lze automat sestavit pomocí Booleovy algebry, klopných a kombinačních obvodů.

Pro realizaci programu jsem použil jazyk C#. Tento jazyk je vysokoúrovňový, objektově orientovaný, vyvinutý firmou Microsoft pro platformu .NET. C# je založen na jazycích C++ a Java.

Mezi hlavní vlastnosti jazyka C#, patří například, že v něm neexistuje vícenásobná dědičnost, takže každá třída může být potomkem pouze jedné třídy. Třída však může implementovat libovolný počet rozhraní. v C# neexistují žádné globální metody a proměnné. Vše musí být deklarováno uvnitř tříd. Jazyk neobsahuje a ani nepotřebuje dopřednou deklaraci, takže není důležité pořadí deklarace metod. Je i daleko typově bezpečnější než C++.

Program Autonima je psán pro operační systém Windows. Proto jsem se rozhodl použít jako grafický frame work pro psaní aplikace WPF (Windows Presentation Foundation). WPF je nástupcem Windows Forms, ale plně je nenahrazují. Jde spíše o další možný způsob psaní aplikací pro Windows. Oproti Windows Forms nabízí WPF nové možnosti hlavně v grafické oblasti zpracování aplikace.

Síla WPF je právě v grafice, ve které nabízí nové možnosti:

- vektorová grafika – umožňuje bezztrátovou změnu velikosti prvků
- efekty – zrcadlení, rozostření, průhlednost, stíny nebo záře
- multimédia – práce s videem a audiem
- animace – rotace, přechod barev, 3D animace

Program Autonima je jediný samostatný EXE soubor, ovšem k plné funkčnosti potřebuje soubor devices.xml, který obsahuje seznam zařízení FPGA, a dále soubor settings.txt, který

obsahuje cestu k programu ISE. Tento soubor se generuje sám. Dotaz na cestu se zobrazí po prvním spuštění Autonimy.

5.1 Založení nového projektu

Pokud program Autonima spouštíme poprvé, nejspíše se zobrazí výzva pro zadaní cesty k programu ISE. Cesta je předvyplněna standardním umístěním. Pokud je ISE umístěn jinde je možno nastavení změnit. Program byl navržen pro ISE ve verzi 14.2. Má-li uživatel nainstalovanou jinou verzi, je nutné cestu ručně změnit.

Nový projekt založíme vybráním záložky *Soubor z hlavní nabídky a* v ní zvolíme položku *Nový.* Tímto se otevře nové dialogové okno. Okno obsahuje základní informace k vytvoření nového projektu. Pokud chceme pracovat pouze s grafickou částí a generovat VHDL kód, tak nás zajímají pouze položky *Typ automatu*, *X*, *Y* a *Jméno projektu*.

Položka *Typ automatu* určuje o jaký typ automatu se bude jednat. Uživatel volí mezi variantami Mealyho a Mooreův automatu. *X* označuje vstupní abecedu, nebo-li kolik vstupních proměnných nám vstupuje do automatu. *Y* nám označuje velikost výstupní abecedu.

Položky *Product*, *Family*, *Device*, *Package* a *Speed* obsahují informace o FPGA obvodu, pro který bude generován bitstream. Tyto informace jsou uloženy v souboru devices.xml. Zařízení v něm jsou popsány jednoduchou XML (*Extensible Markup Language*) strukturou a je možné soubor dále editovat a rozšiřovat o další zařízení.

```
<family>Spartan3E
        <device>XC3S100E
                <package>VQ100</package>
                <package>CP132</package>
                <package>TQ144</package>
                <speed>-4</speed>
                <speed>-5</speed>
        </device>
        <device>XC3S250E
                <package>VQ100</package>
                <package>CP132</package>
                <package>TQ144</package>
                <package>FT256</package>
                <package>PQ208</package>
                <speed>-4</speed>
                <speed>-5</speed>
        </device>
        <device>XC3S500E
```

Obrázek 5.1 Ukázka ze souboru devices.xml

Jméno projektu označuje, pod jakým jménem bude náš automat reprezentován. Zadaným jménem se ihned po potvrzení všech hodnot, vytvoří složka Projekty. v ní jsou uloženy všechny doposud vytvořené automaty.

5.2 Práce s editorem

Po založení nového projektu se v horní části programu zobrazí panel nástrojů obsahující dvě položky. První je položka na přidávání stavů, druhá na jejich mazání. Výběr probíhá kliknutím levého tlačítka myši, zrušení výběru pravým tlačítkem.

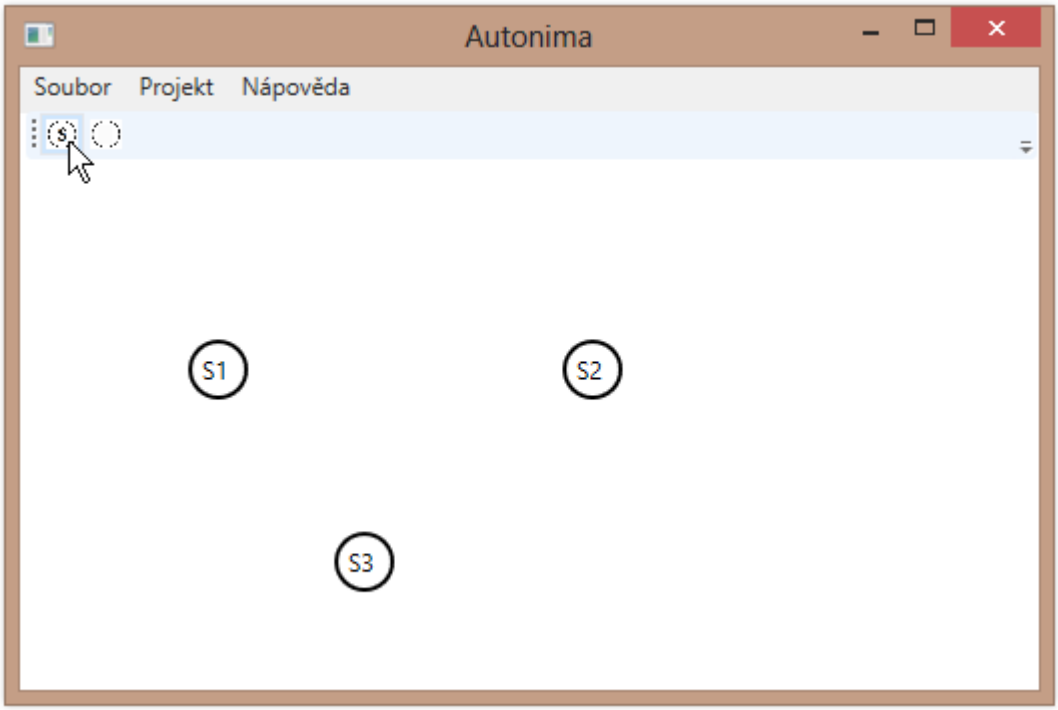
5.2.1 Editace stavů

Pro vytvoření stavu je nutno vybrat první položku z panelu nástrojů. Poté stačí kliknout libovolně na volnou plochu v programu a stav se vytvoří. Jméno stavu se automaticky inkrementuje. Stavy nelze umístit blízko sebe. Pokud nejde vytvořit nový stav, je nejspíše problém právě v to, že blízko je jiný stav. Toto omezení zabraňuje, aby se jednotlivé stavy mezi sebou překreslovaly.

Jednotlivé stavy lze přetahovat. Stačí na ně kliknout levým tlačítkem, držet jej, přetáhnout na požadované místo a pustit tlačítko myši. I zde platí, že pokud se přetažení neprovede, je nejspíše

poblíž jiný stav.

Pro mazání stavů je třeba vybrat druhou položku v panelu nástrojů a opět kliknout na libovolný stav, jenž chceme smazat.



Obrázek 5.2 Ukázka editace stavů

5.2.2 Editace přechodů

Dvojklikem na libovolný stav se nám otevře dialogové okno daného stavu pro editaci přechodů, které z něj mají vést. Toto okno se liší podle toho, zda bylo při zakládání projektu vybrán Mealyho či Mooreův automat.

Pokud byl vybrán Mooreův automat, tak má oproti Mealymu navíc v pravé části rolovací seznam, který reprezentuje hodnotu daného stavu. Hodnotu není třeba nijak potvrzovat, nastaví se automaticky po vybrání ze seznamu. Hodnoty v seznamu jsou generovány v závislosti na tom, kolika bitová logika byla zvolena pro výstup.

Samotné okno se skládá ze 2 hlavních částí, příkazového řádku a stromové zobrazovací struktury. Pokud se stane, že zapomeneme, pro který stav vytváříme přechody, je název stavu

v liště. v příkazovém řádku je vždy při otevření okna zadán vzorový příkaz na přidávání přechodu. Syntaxe příkazu se liší podle typu automatu:

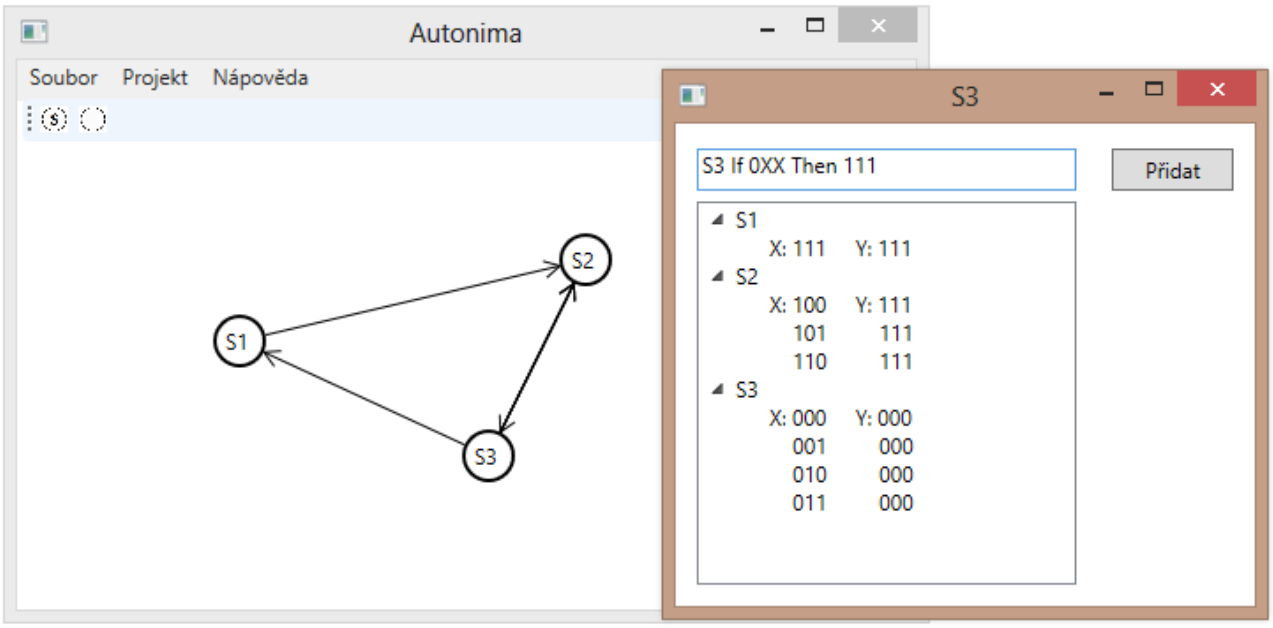
Pro Mealyho automat – *KAM* **If** *vstupní podmínka* **Then** *výstupní podmínka*

Pro Mooreův automat - *KAM* **If** *vstupní podmínka*

Parametr *KAM* reprezentuje stav do kterého má vést přechod, ze stavu, v němž se právě nacházíme. *Vstupní podmínka*, stejně jako *výstupní podmínka* představuje vstupní a výstupní logiku. *Vstupní podmínku* lze vymaskovat a tím urychlit editaci přechodů, zvláště u vícebitové logiky. Libovolný bit lze nahradit písmenem *X*. Tím je programu řečeno, že nezáleží na hodnotě daného bitu. Zde je možné si to prohlédnout na příkladu:

S3 If XXX – při jakékoliv kombinaci vstupu se přidá přechod do stavu S3
S3 If 1X1 – při kombinaci 101 a 111 se přidá přechod do stavu S3
S3 If 0XX – při kombinacích 000, 001, 010 a 011 se přidá přechod do stavu S3

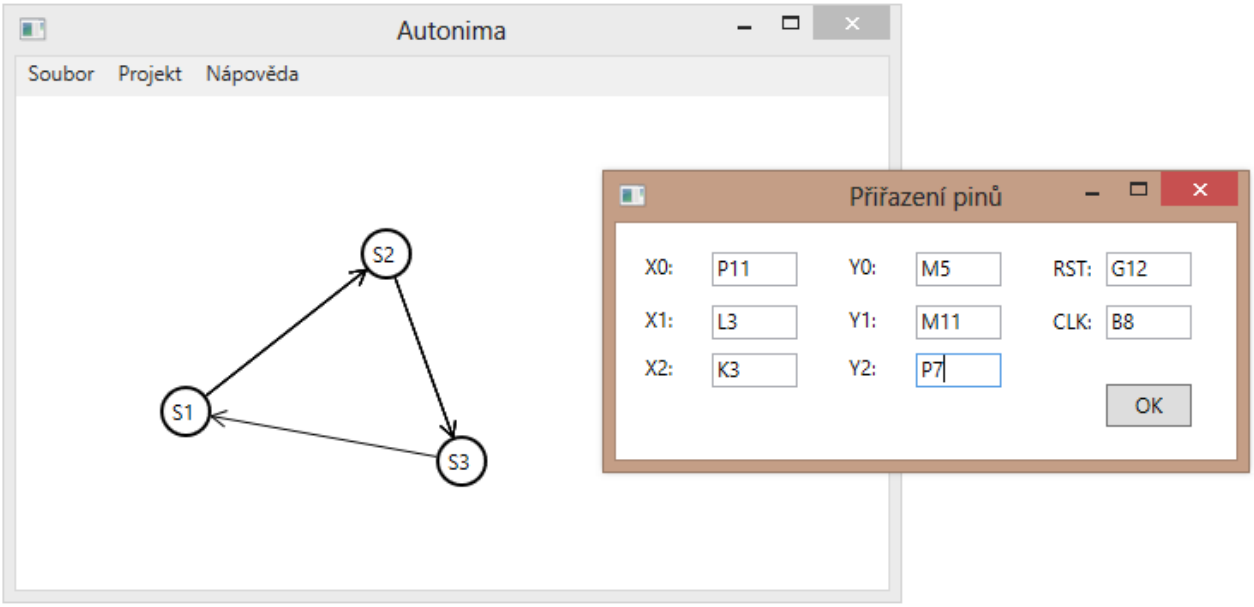
Ve stromové zobrazovací struktuře je přehledně zobrazeno, kam jsou již přechody vytvořeny. Automaticky, při vytvoření stavu, jsou vytvořeny všechny možné přechody a odkazují na stav, ze kterého vycházejí. Tím je zajištěno, že budou popsány všechny přechody a nebude žádný vynechám, což by mohlo působit značné potíže.



Obrázek 5.3 Ukázka editace přechodů

5.3 Vygenerování BIT souboru

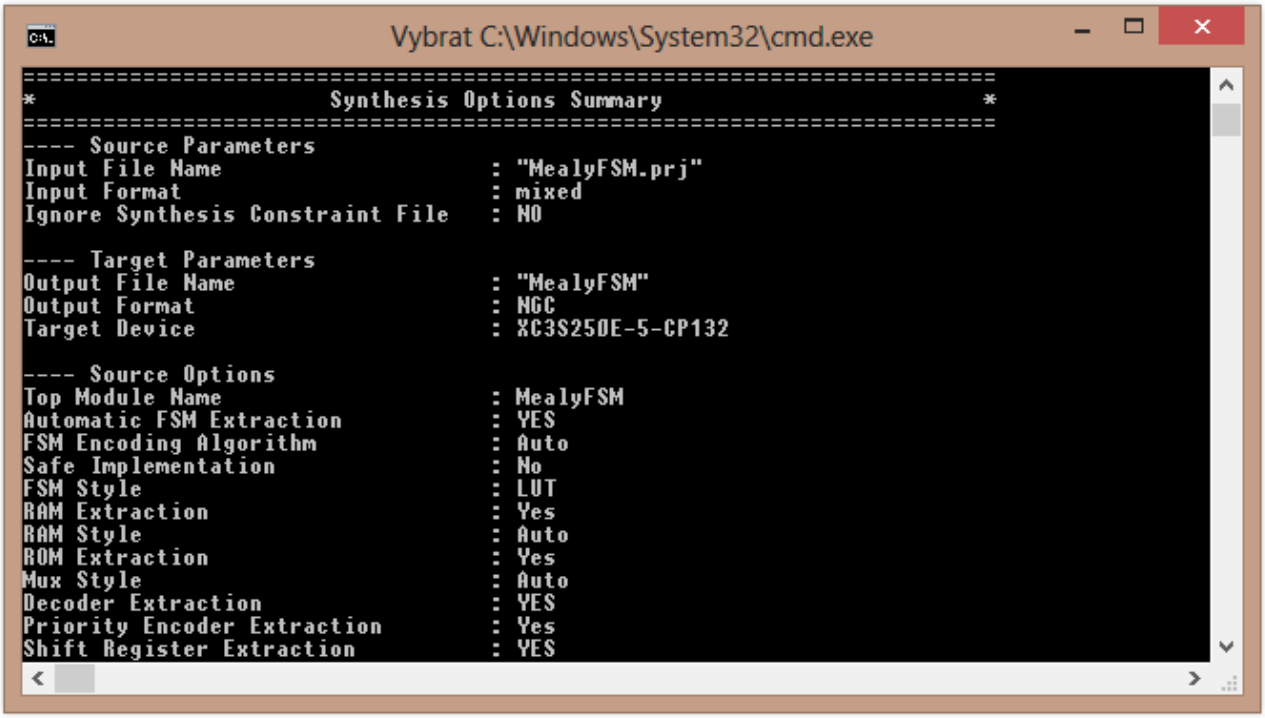
Po navržení konečného stavového automatu . Ten je možné spustit v záložce *Projekt* z **hlavní nabídky**. Tím se otevře okno s možností přiřazovat jednotlivé piny. Označení pinů se mezi jednotlivými druhy zařízení liší. Zpravidla je možné označení pinů zjistit přímo z desky. Na obrázku 5.4 je vidět přiřazování pinů vůči FPGA typu Spartan3E-250. Konkrétně, jsou vstupní proměnné a reset(RST) přiřazeny na spínače a výstupy na diody.



Obrázek 5.4 Ukázka přiřazování pinů

Po přiřazení pinů a jejich potvrzení se vytvořený stavový automat uloží do VHDL souboru. Poté se postupně ve složce projektu vytvoří soubory typu *.prj*, *.xst*, *.ucf* a *run.bat*. Okamžitě poté se spustí dávkový soubor *run.bat*, který obsahuje jednotlivé příkazy na vygenerování požadovaného bitstreamu. v případě, že při generování bitstreamu nastane chyba, je možné ji dohledat v souboru *.bld*, do kterého se zaznamenává průběh vytváření bitstreamu.

V případě že byli piny zadány chybně, je možné upravit, pomocí libovolného textového editoru soubor *.ucf,* který obsahuje informace o jednotlivých pinech. Po upravení a uložení souboru stačí pouze spustit soubor *run.bat*. Není tedy vůbec potřeba zpětně načítat vygenerovaný stavový automat.



Obrázek 5.5 Ukázka generování bitstreamu

5.3.1 Soubor run.bat

Soubor *run.bat* je dávkovým souborem, jenž se automaticky vytvoří při požadavku na vytvoření BIT souboru. Obsahuje sadu příkazů sloužících k vygenerování bitstreamu, daného automatu, viz. Obrázek 5.5

```
call C:/Xilinx/14.2/ISE_DS/settings64.bat
xst -ifn Mealy.xst
ngdbuild -dd _ngo -uc Mealy.ucf -p XC3S500E-FG320-5 Mealy.ngc Mealy.ngd
map -p XC3S500E-FG320-5 -cm area -ir off -pr off -c 100 -o Mealy.ncd Mealy.ngd Mealy.pcf
par -w -ol high -t 1 Mealy.ncd Mealy.ncd Mealy.pcf
bitgen Mealy.ncd
```

Obrázek 5.3 Příklad seznam příkazů v souboru run.bat

Tato sada příkazů postupně provede *NGDBuild*, *Mapping, Place and Routing* a *BitstreamGeneration*.

NGDBuild slouží k vytvoření souboru NGD (Native Generic Database), který obsahuje

logický popis návrhu z hlediska logických prvků, jako jsou například hradla AND, LUTy, klopné obvody nebo paměti typu RAM. NGD soubor obsahuje jak logický popis návrhu, tak poskytuje informace o původní hierarchii vyjádřeném ve vstupním netlistu.

Mapping mapuje logickou konstrukci na FPGA obvodech. Vstupem mapování je NGD soubor, který obsahuje logický popis návrhu z hlediska hierarchických složek, použitých při vývoji designu. Příkaz MAP poté mapuje logiku komponent (logické buňky, vstupy a výstupy buněk a další komponenty) v cílovém FPGA. Výstup příkazu MAP je NCD soubor, který je fyzickým vyobrazením mapovaných komponent v FPGA.

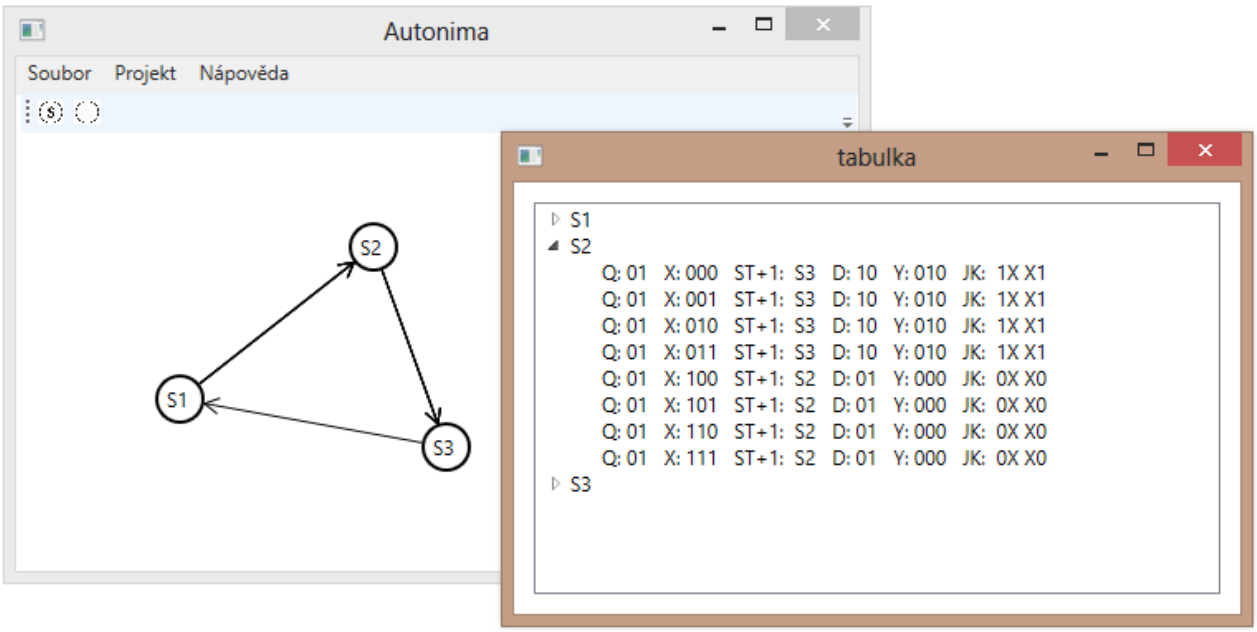
Place and Routing se spouští příkazem PAR. Vstupem je soubor NCD. *Place and Routing* umísťuje a propojuje návrh s výstupy. Snaží se přitom jednotlivé části kódu umístit tak, aby byli co nejefektivněji využity. Výstupem je nový, upravený NCD soubor.

BitstreamGeneration generuje BIT soubor, jenž obsahuje bitstream, pro dané FPGA. Jako vstup mu slouží NCD soubor. Po vytvoření BIT souboru je možné tento bitstream nahrát do FPGA pomocí programu ISE iMPACT.

5.4 Tabulka přechodů

V tabulce přechodů jsou detailně zobrazeny přechody všech stavů. Tato tabulka je pro přehlednost ve stromové zobrazovací struktuře. Každý stav obsahuje celkem 6 prvků. Základní a nejdůležitější jsou prvky *X*, *ST+1* a *Y*. *X* a *Y* jsou vstupní a výstupní funkce. *ST+1* značí následující stav, pro danou výstupní funkci

Prvky *Q*, *D* a *JK* slouží k převedení automatu na klopné obvody. *Q* je binárně zakódovaný daný stav. Kódování se bere vhledem k pozici v tabulce. Je-li stav na prvním řádku bude zakódován jako 000. Na pátém řádku bude mít hodnotu 100. *D* je binárním ekvivalentem *ST+1*. Slouží k tomu, chceme-li automat sestavit pomocí D klopných obvodů. *JK* zas využijeme, chceme-li výsledný automat sestavit JK klopných obvodů.



Obrázek 5.5 Ukázka přechodové tabulky

5.6 Instalace

Program Autonima nemá vlastní instalaci. Jedná se o jediný *.exe* soubor. k plné funčnosti je ale potřeba další soubory, jako například *devices.xml*, obsahující informace o FPGA obvodech. Dále je třeba mít i nainstalovaný program ISE nebo jeho bezplatnou verzi Web Edition.

Bez těchto přiložených souborů umožňuje program Autonima pouze grafické navrhování stavových automatů a jejich reprezentaci ve VHDL.

Program Autonima je psán pro operační systém Windows, obsahující platformu .NET, o verzi alespoň 4.0. Verze .NET 4.0 je součástí Windows 8. Pro systém Windows 7 je třeba tuto verzi nainstalovat, stejně tak jako pro Windows Vista a Windows XP. Aby bylo možné .NET nainstalovat pro poslední dva zmíněné operační systémy, je třeba mít nainstalovaný příslušný Servis Pack. Pro Windows Vista je potřeba minimálně Servis Pack 1 a pro Windows XP je to Servis Pack 3.

5.7 Vnitřní struktura programu Autonima

Každý jednotlivý stav je reprezentován jako třída *Stav*. Tato třída obsahuje privátní proměnné *x, y, hodnota, jmeno*. Proměnné *x a y* jsou datového typu integer a reprezentují souřadnice daného stavu. Proměnné *hodnota a jmeno* jsou datové typu string. *Hodnota* představuje výstupní funkci. Ta

se využívá pouze, jednáli se o stavový automat typu Moore. *Jmeno* obsahuje informaci o jménu, která je následně vykreslena doprostřed stavu.

Předchod je reprezentován třídou *Prechod*. Ta obsahuje privátní proměnné typu string. *OdStavu* uchovává hodnotu ze kterého stavu přechod vede a *DoStavu* naopak, do kterého daný přechod vede. Dále třída obsahuje proměnné *vektorX* a *vektorY*. Proměnná *vektorX* představuje vstupní podmínku, při které má přechod nastat. *VektorY* reprezentuje výstupní funkci na daném přechodu. *VektorY* využijeme pouze, pokud máme stavový automat typu Mealy, který má pro každou kombinaci vstupu a stavu jinou výstupní funkci.

Jak třída *Stav*, tak třída *Prechod* využívají zapouzdření svých proměnných. v objektově orientovaném programování se používá, aby nikdo, krom dané třídy, neměl přímí přístup k proměnným. Přístup k nim je možné pouze přes metody, jenž se označují jako *setter a getter*. k proměnné *x*, ve třídě *Stav* je tedy přistupováno pomocí metody *getX()* nebo *setX()*, jak je možné vidět na obrázku 5.6.

```
public void setX(int x)
{
    this.x = x;
}

public int getX()
{
    return x;
}
```

Obrázek 5.6 Setter a getter pro x

Vykreslování přechodů probíhá způsobem, že se vezme středová souřadnice obou stavů, pro která má být daná přechodová funkce vykreslena. z těchto dvou bodů se sestaví obecná rovnice přímky. Dále pro každý stav je sestavena obecná rovnice kružnice. Pomocí rovnice kružnice a přímky nalezneme průsečíky. Vždy vyjdou dva průsečíky dané rovnice. Vezme se ten průsečík, který je souřadnicově blíže k druhému stavu. z tohoto bodu je poté veden samotný přechod.

```
private void rovnicePrimky(int x1, int y1, int x2, int y2, out int a, out int b, out int c)
{
    int u1, u2;

    u1 = x2 - x1;
    u2 = y2 - y1;

    a = u2;
    b = u1 * -1;
    c = -1 * (a * x1 + b * y1);
}
```

Obrázek 5.7 Metoda na počítání rovnice přímky

Závěr

Hlavním cílem této práce bylo vytvořit snadno použitelný grafický editor stavových automatů s možností převést grafický návrh do VHDL kódu. Dále mít i možnost ukládání do formátu pro možné zpětné načtený návrhu. Program Autonima byl opakovaně a úspěšně testován na přípravku Basys 2 s FPGA obvodem Spartan3E-250. Na testování byla použita i různá zadání úloh z předmětu Číslicová technika.

Program do budoucna lze rozšířit a tím zvýšit jeho použitelnost, například o Karnaughovy mapy a převedení stavových automatů do klopných obvodů D a JK. Zobrazení pomocí těchto obvodů by sloužilo především k možnosti sestavit si stavový automat pomocí hradel a klopných obvodů a i pro kontrolu a výuku studentů. Do FPGA by se primárně měl nahrávat stavový automat popsaný pomocí VHDL, protože zabere méně LUTů, oproti například automatu složenému z klopných obvodů D a logických hradel typu NAND.

Další možností o rozšířením by mohlo být, že vytvořený stavový automat by šlo převést do blokového schéma. Spolu s přidáním nových prvků v grafické části, jako například s logickými hradly (AND, OR, XOR, atd.) a jejich propojením s blokovým znázorněním stavového automatu.

Jednou z dalších možností o rozšíření programu Autonima, by mohla být možnost zpětné načítání VHDL souborů, jak vytvořených pomocí programu Autonima, tak i z programu ISE.

Zdrojové kódy jsou přiloženy na CD v adresáři *source*. Spustitelná verze programu najdeme ve složce *Autonima*. Ta obsahuje i zabalenou verzi programu a to jak ve formátu *.RAR* ,tak *.ZIP*.

Literatura

[1] Pinker J., Poupa M. : Číslicové systémy a jazyk VHDL, Praha, 2006

[2] Maxfield C. : The design warrior's guide to FPGAs, 2004

[3] Parnell K. , Mehta N. : Programmable logic design quick start handbook, 2004

[4] http://www.xilinx.com/support/documentation/application_notes/xapp452.pdf