VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

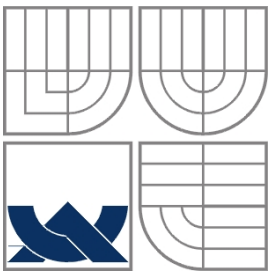
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

JEDNODUCHÝ SIMULÁTOR ČÍSLICOVÝCH OBVODŮ

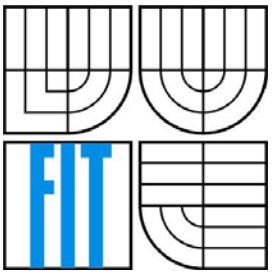
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ALEŠ KOLMAN
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

JEDNOUCHÝ SIMULÁTOR ČÍSLICOVÝCH OBVODŮ

NÁZEV BAKALÁŘSKÉ PRÁCE ANGLICKY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ALEŠ KOLMAN

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Lukáš Sekanina, Ph.D.

BRNO 2009

Abstrakt

Tato práce je zmařena na simulování číslicových obvodů, především menších kombinačních a sekvenčních obvodů. Je orientována zejména na dosažení co nejvyšší rychlosti simulace, z tohoto důvodu byla naimplementována v jazyce C. Vstupním formátem pro tento projekt byl zvolen zápis číslicového obvodu v EDIFu. Výstup nebyl specifikován.

Abstract

This work is oriented on the simulation of digital circuits, especially small combinational and sequential circuits. This project is focused particularly on achieving the highest possible speed of simulation, for this reason was chosen programming language C. As Input format for this project was selected EDIF format of digital circuits. Output has not been specified.

Klíčová slova

číslicový obvod, simulátor, EDIF

Keywords

Digital circuits, simulator, EDIF

Citace

Aleš Kolman: Jednoduchý simulátor číslicových obvodů, bakalářská práce, Brno, FIT VUT v Brně, 2009

Jednoduchý simulátor číslicových obvodů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Ing. Lukáše Sekaniny Ph. D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Aleš Kolman
20. května 2009

Poděkování

Za odborné vedení, připomínky a návrhy bych chtěl poděkovat doc. Ing. Lukáši Sekaninovi, Ph. D., který trpělivě dohlížel na průběh zpracování mé bakalářské práce.

Obsah

1 Úvod

V dnešní době se na trhu vyskytuje veliké množství simulátorů číslicových obvodů např. modelsim, multisim, dsch simulátor apod. Proč jsem se tedy rozhodl obohatit, už tak dost obsáhlý trh o další simulátor? Hlavním podnětem byla snaha vytvořit simulátor, který by se dal využít především při evolučním návrhu číslicových obvodu ve fázi testování.

Cílem projektu bylo vytvořit velice rychlý simulátor specificky směrován na jednodušší, menší, diskrétních systémy, tedy takový, který by zbytečně neobsahoval obecné, nepotřebné, funkce a hlavně takový, kde bych si mohl kdykoliv beztrestně a jednoduše zasahovat do zdrojového kódu (měnit simulační algoritmy, přidávat podporované komponenty atd.).

Bakalářská práce je strukturována následovně: Druhá kapitola vysvětluje základní pojmy z oblasti číslicových obvodů. Poté, co se dozvíme, co to jsou číslicové obvody, čím sou charakteristické, čím výhodné a jak se dělí, přejdeme k otázce jejich simulace, kterou se zabývá třetí kapitola. Třetí kapitola popisuje nejen simulaci číslicových obvodů, ale snaží se (stejně jako druhá kapitola) nejprve definovat základní pojmy týkající se obecné simulace. Poté zmiňuje specifika simulace číslicových obvodů a uvádí elektronické formáty používajících se pro popis číslicových obvodů. Následuje čtvrtá kapitola, která popisuje proces návrhu, implementace a ověření jednoduchého simulátoru číslicového obvodu. Na konec práce je zařazeno zhodnocení dosažených výsledků a výpis nápadů možných inovací či rozšíření pro konečný program.

2 Číslicové obvody

Číslicové obvody, diskrétní obvody, jsou podmnožinou elektrických obvodů. Hlavním rysem takovýchto obvodů je, že reprezentují informaci diskrétně, to znamená pouze dvěma hodnotami. V počítačové terminologii se také často používá označení „0“, logická nula, a „1“, logická jednička.

Přínosných vlastností číslicových obvodů je celá řada. Mezi nejevidentnější bych zařadil skutečnost, že v číslicových systémech může být representována informace v teoreticky neomezeném rozsahu s neomezenou přesností, dále zde přecházejí do pozadní hlavní problémy analogových obvodů, jako problém šumu, rušení, teplotních výkyvů atd., také návrh těchto obvodů je značně jednodušší, než návrh analogových obvodů, kde je nutné znát matematické modely složitých součástek jako je tranzistor, kondensátor apod. Na závěr bych uvedl pragmatickou vlastnost, která vedla k jejich masivnímu využití, kterou je ekonomická výhodnost. V mnoha případech dokáží číslicové obvody vyřešit stejně složité problémy jako obvody analogové za jednoznačně nižší cenu.

Logika číslicových obvodů je odvozena z Booleovy algebry a právě Booleově algebře je věnována následující podkapitola.

Číslicové obvody můžeme modelovat různými způsoby: pravdivostní tabulkou, Karnaughovou mapou či výrazem Booleovy algebry.

2.1 Booleova algebra

Hned ze začátku této kapitoly bych rád podotkl, že inspiraci pro tuto podkapitolu jsem čerpal především z [3]. Booleova algebra je algebraická struktura, nazvaná podle irského matematika George Boolea. Mimo jiné se používá pro popis logických obvodů. Logické výrazy, které popisují logické obvody, obsahují hodnoty pravda a nepravda, což můžeme chápat jako hodnotu logická „1“ a hodnotu logická „0“.

Booleova algebra je uspořádaná šestice:

$$(B,0,1,\vee,\wedge,-)$$

- B symbolizuje neprázdnou množinu
- 0 symbolizuje tzv. nejmenší prvek množiny B
- 1 symbolizuje tzv. největší prvek množiny B
- $\vee$ symbolizuje binární operaci na množině B (logický součet)
- $\wedge$ symbolizuje binární operaci na množině B (logický součin)
- - symbolizuje unární operace na množině B (negace)

Aby tato šestice byla Booleovou algebrou, musí splňovat všechny základní axiomy (viz následující podkapitola).

2.1.1 Základní Axiomy

Pro všechna *a*, *b*, *c* z množiny B platí:

1. Komutativita

$$a \wedge b = b \wedge a$$
$$a \vee b = b \vee a$$

2. Distributivita

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$
$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

3. Neutralita 0, 1

$$a \vee 0 = a$$
$$a \wedge 1 = a$$

4. Komplementarita

$$a \vee -a = 1$$
$$a \wedge -a = 0$$

5. Nedegenerovanost

$$0 \neq 1$$

2.1.2 Odvozené vlastnosti

Pro všechna *a*, *b*, *c* z množiny B platí:

6. Asociativita

$$(a \vee b) \vee c = a \vee (b \vee c)$$
$$(a \wedge b) \wedge c = a \wedge (b \wedge c)$$

7. Absorbce

$$a \vee (a \wedge b) = a$$
$$a \wedge (a \vee b) = a$$

8. Absorbce negace

$$a \vee (-a \wedge b) = a \vee b$$
$$a \wedge (-a \vee b) = a \wedge b$$

9. Agresivita 0, 1

$$a \vee 1 = 1$$
$$a \wedge 0 = 0$$

10. Idempotence

$$a \vee a = a$$
$$a \wedge a = a$$

11. de Morganovy zákony

$$-a \wedge -b = -(a \vee b)$$
$$-a \vee -b = -(a \wedge b)$$

12. Dvojitá negace

$$-(-a) = a$$

2.2 Rozdělení číslicových obvodů

U číslicových obvodů se můžeme setkat s děleními obvodů na obvody kombinační, sekvenční, synchronní, asynchronní, úrovňové a hranové. Následující podkapitoly se zabývají vysvětlením těchto pojmů.

2.2.1 Kombinační obvody

Kombinační logické obvody jsou specifické tím, že stavy na výstupech závisí pouze na okamžitých kombinacích vstupních proměnných a nezávisí na jejich předchozích hodnotách, s výjimkou krátkého přechodového děje. Jedné kombinaci vstupních proměnných odpovídá jediná výstupní kombinace funkčních hodnot. Kombinační logické obvody nemají žádnou paměť předchozích stavů.

Pro popis kombinačního obvodu se často používají výrazy Booleovy algebry. Obvod s n vstupy a m výstupy, které realizují funkce f_1 .. f_m , popíšeme pomocí m výrazů Booleovy algebry.

2.2.2 Sekvenční obvody

Sekvenční logické obvody jsou složeny ze dvou částí, části kombinační a části paměťové, ta může být realizována buďto jednoduchou zpětnou vazbou nebo klopným obvodem, nejčastěji se pro tyto účely využívá klopný obvod typu D. Abychom mohli určit hodnotu výstupní proměnné, je potřeba u sekvenčních obvodů sledovat kromě vstupních proměnných ještě jejich vnitřní proměnné, vnitřní stavy. Tyto proměnné, stavy jsou uchovány v paměťových členech. Existence vnitřních proměnných způsobuje, že stejné hodnoty vstupních proměnných přivedené na vstup obvodu, nevyvolávají vždy stejnou odezvu na výstupu obvodu, čili u sekvenčních obvodů záleží na pořadí v jakém vstupní kombinace přichází.

Sekvenční obvody nejčastěji modelujeme pomocí Moorova nebo Mealyho automatu. Mealyho automat se vyznačuje tím, že hodnota výstupní proměnné je závislá jak na hodnotách vstupních proměnných, tak na vnitřních proměnných. Oproti tomu Kokrův automat je hodnota výstupu závislá pouze na stavu vnitřních proměnných.

Sekvenční obvody můžeme dále rozdělit na sekvenční obvody synchronní a sekvenční obvody asynchronní. U asynchronních sekvenčních obvodů se změna vstupní proměnné promítne ihned do stavu sekvenčního obvodu. U synchronních sekvenčních obvodů je zaveden řídicí synchronizační, respektive hodinový signál. Změna vstupní proměnné se promítne do stavu sekvenčního obvodu, až při příchodu hodinového signálu. Co když ale dojde hned k několika změnám vstupních hodnot během jednoho hodinového pulsu, jak se s tím obvod vypořádá? Existují dvě reakce na tuto situaci, jednak sekvenční obvod může sledovat hodnoty vstupních proměnných a tím i jejich změny po celou dobu trvání hodinového signálu a průběžně na ně reagovat, pak takovému sekvenčnímu obvodu říkáme „Úrovňový“, nebo sekvenční obvod může reagovat na hodnoty vstupních proměnných jen při příchodu hrany hodinového signálu (náběžná nebo sestupná hrana), poté tento obvod spadá do kategorie „hranových“ synchronních sekvenčních obvodů.

2.2.3 Klopné obvody

Klopné obvody se mohou nacházet v několika stavech, ze kterých mohou být vstupem přepnuty do stavu jiného. Podle počtu stavů a způsobu přepínaní se dělí na následující druhy:

1. Astabilní klopný obvod (AKO) - nemá žádný stabilní stav a neustále se přepíná mezi dvěma nestabilními stavy. Tento typ obvodu lze použít například jako generátor impulsů.

2. Monostabilní klopný obvod (MKO) - má jeden stabilní stav, ze kterého je možné jej přepnout do stavu nestabilního. Obvod se sám po určité době přepne zpět do stabilního stavu. Tento typ obvodu je možné použít například pro realizaci zpoždění.
3. Bistabilní klopný obvod (BKO) - se může nacházet v dvou stabilních stavech, přičemž je možné jej mezi těmito stavy libovolně přepínat. Tento typ obvodu lze použít například jako paměť, neboť až do přepnutí zůstává v předchozím stabilním stavu. Tato skupina je pro číslicové obvody nejvýznamnější, proto uvedu i hlavní představitele, mezi které patří klopné obvody typu RS, JK, D a T. Komponenta RS je stavebním kamenem pro všechny ostatní vyjmenované komponenty, proto klopnému obvodu RS bude ještě věnována vlastní podkapitola.

2.3 Základní komponenty číslicových obvodů

Stejně jako jsou všechny elektrické obvody složeny z elektrických prvků, součástek, jsou i číslicové obvody tvořeny součástkami, s tím rozdílem, že u číslicových obvodů je škála součástek značně specifičtější. Mezi základní součástky číslicových obvodů patří logické členy, hradla. Základní logická hradla se vyznačují skutečností, že je možné kterékoliv z nich považovat za určitou logickou funkci v Booleove algebře $y = f(a_1, a_2, ...a_n)$

Mezi základní logické členy diskrétních systémů patří hradla: NOT, AND, OR, XOR, NAND, NOR, XNOR. Všechny tyto komponenty číslicových obvodů opisují svůj vzor z Booleovy algebry. Logické členy NAND, NOR a XNOR jsou pouze negací členů základních, rozhodl jsem se tedy nevěnovat těmto třem členům vlastní podkapitoly. Jak plyne z Booleovy algebry pomocí logických členů AND, OR a NOT je možné realizovat libovolný logický obvod a tedy i zbylá logická hradla. Dále platí, že za pomoci pouze hradel NOT a AND, nebo NOT a OR, lze realizovat jakákoliv hradla, respektive jakýkoliv číslicový obvod, Tato poslední skutečnost poukazuje na komplementárnost hradel, respektive logických funkci AND a OR.

Pro převod výrazu, který obsahuje pouze hradla OR (resp. NAND) na výraz, který obsahuje pouze hradla AND (resp. OR) se používá následující postup:

- Nahrazení hradla OR hradlem AND - Vycházíme z Booleovského výrazu pro logickou funkci OR, na který použijeme vlastnost dvojité negace, poté výraz upravíme pomocí de Morganových pravidel do finálního stavu.

$$a \vee b = -(-(a \vee b)) = -(-a \wedge -b)$$

- Nahrazení hradla AND hradlem OR - Vyjdeme z Booleovského výrazu pro logickou funkci AND, na který použijeme vlastnost dvojité negace, poté výraz upravíme pomocí de Morganových pravidel do finálního stavu.

$$a \wedge b = -(-(a \wedge b)) = -(-a \vee -b)$$

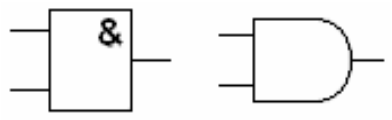
Dále následuje popis vybraných komponent, mezi vybrané komponenty jsem zařadil kromě základních logických hradel ještě hradlo XOR a asynchronní klopný obvod typu RS, realizovaný pomocí hradel NAND.

2.3.1 Logický součin AND

Výstup nabývá hodnoty „1“ pouze když mají všechny vstupy hodnotu „1“, jinak nabývá hodnoty„0“, jinými slovy hodnota „0“ na kterémkoliv ze vstupů dělá hodnotu „0“ na výstupu.

Ekvivalent logické funkce $y = a_1 \wedge a_2 \wedge ... \wedge a_n$

Schématická značka:



Obrázek 2.1Schématická značka hradla AND

a) IEC značení b) "Americké" značení

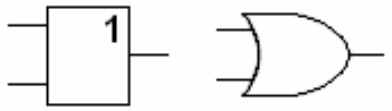
Připojením hradla NOT na výstup hradla AND získáme logickou funkci NAND, která má tedy analogickou definici jako logická funkce AND, s jediným rozdílem, že hodnota výstupu je pro všechny kombinace invertována.

2.3.2 Logický součet OR

Výstup nabývá hodnoty „0“ pouze když mají všechny vstupy hodnotu „0“, jinak nabývá hodnoty„1“, jinými slovy hodnota „1“ na kterémkoliv ze vstupů dělá hodnotu „1“ na výstupu.

Ekvivalent logické funkce $y = a_1 \vee a_2 \vee ... \vee a_n$

Schématická značka:



Obrázek 2.2 Schématická značka hradla OR

a) IEC značení b) "Americké" značení

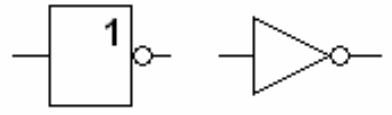
Připojením hradla NOT na výstup hradla OR získáme logickou funkci NOR, která má tedy analogickou definici jako logická funkce OR, s jediným rozdílem, že hodnota výstupu je pro všechny kombinace invertována.

2.3.3 Logická negace NOT

Když vstup má hodnotu „0“, tak výstup nabývá hodnoty„1“, a naopak, jinými slovy výstup nabývá opačné hodnoty než jaká je na vstupu

Ekvivalent logické funkce $y = -a$

Schématická značka:



Obrázek 2.3 Schématická značka hradla NOT

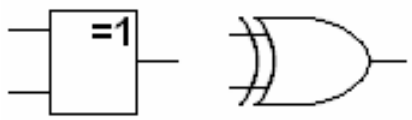
a) IEC značení b) "Americké" značení

2.3.4 Exkluzivní logický součet XOR

Výstup nabývá hodnoty „1“ právě tehdy, když je na vstupech lichý počet hodnot „1“, jinými slovy sudý počet hodnot „1“ na vstupech dává na výstup hodnotu „0“.

Ekvivalent logické funkce $y = a_1 \otimes a_2 \otimes ... \otimes a_n$

Schématická značka:



Obrázek 2.4 Schématická značka hradla XOR

a) IEC značení b) "Americké" značení

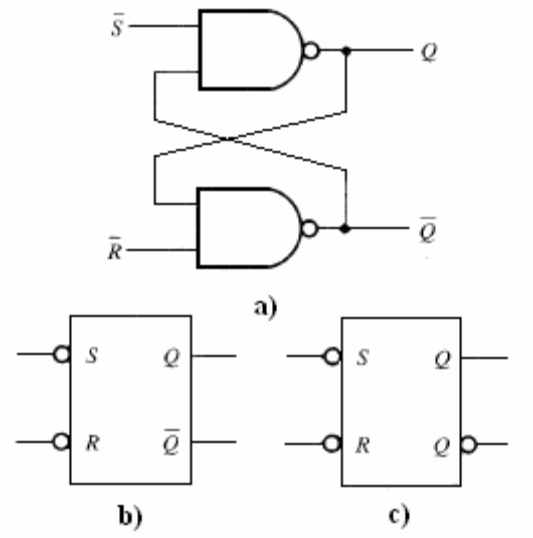
Připojením hradla NOT na výstup hradla XOR získáme logickou funkci XNOR, která má tedy analogickou definici jako logická funkce XOR s jediným rozdílem, že hodnota výstupu je pro všechny kombinace invertována.

2.3.5 Asynchronní klopný obvod typu RS

Funkce obvodu RS je následující, pokud přivedeme logickou 1 a na vstup R logickou 0, do klopného obvodu se uloží logická 1. A obráceně: pokud na vstup S přivedeme logickou 0 a na vstup R logickou 1, dojde k uložení logické 0. Když na oba vstupy přivedeme logickou 0, uložená hodnota se nezmění (obvod si pamatuje svůj minulý stav). Poslední možná kombinace, což jsou dvě logické 1, je tzv. *zakázaným stavem*.

Jelikož Hradlo NAND je nejjednodušší pro realizaci (v technologii CMOS je tvořeno pouhými čtyřmi tranzistory), tak zde uvedu realizaci RS klopného obvodu právě prostřednictvím hradel NAND.

Zapojení a schématická značka.



Obrázek 2.5 RS-KO: a) schéma zapojeni

b) a c) možné schématické značky

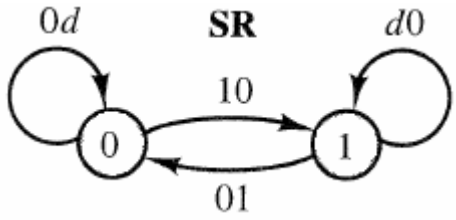
Příklad popisu klopného obvodu pravdivostní tabulkou:

R	S	Příští stav	Význam
0	0	Q = Q	Zachovej stav
0	1	Q = 1	Nastav logickou „1“
1	0	Q = 0	Nastav logickou „0“
1	1	Q = X	**Zakázaný stav**

Příklad popisu klopného obvodu charakteristickou rovnicí:

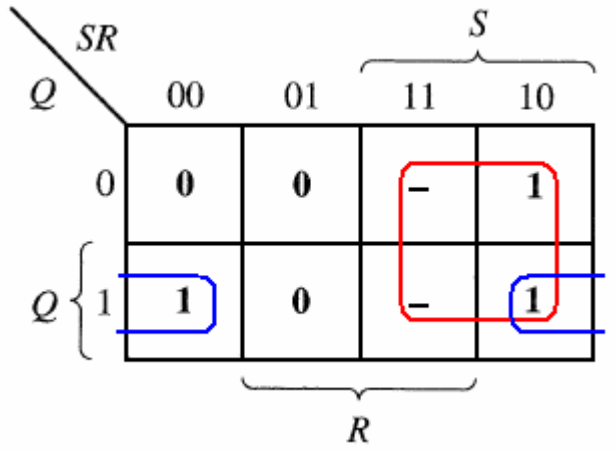
$$Q_{i+1} = S + \overline{R} \cdot Q$$

Příklad popisu klopného obvodu Moorovým automatem:



Obrázek 2.6 Moorův automat pro RS KO

Příklad popisu klopného obvodu Karnaughovou mapou:



Obrázek 2.7 Karnaghova mapa pro RS KO

3 Simulátory a simulace

3.1 Pohled na simulaci a simulátor obecně

Ještě před omezením pouze na simulaci číslicových obvodů jsem se rozhodl zde připomenout šíři pojmu simulace a uvést zde i obecné vysvětlení slova simulace, dle přednášek kurzu Modelovaní a simulace na Vysokém Učení Technickém v Brně (odkaz na literaturu, ze které tento předmět čerpá je uveden v [6]).

Modelování je proces vytváření modelů systémů na základě našich znalostí. Tento proces je obecně velmi náročný a často vyžaduje znalosti z více oborů. Kvalita vytvořeného modelu zásadním způsobem ovlivní výsledky získané experimentováním s modelem.

Simulace je metoda získávání nových znalostí o systému experimentováním s jeho modelem. Pro účely simulace musí být model popsán odpovídajícím způsobem, ale ne každý model je pro simulaci vhodný. Pro získání potřebných informací obvykle potřebujeme opakovat simulační experimenty vícekrát s různými parametry. Proces experimentování v reálném světě je vždy zatížen chybami měření a dalšími faktory, které mohou způsobit problémy při interpretaci výsledků. Navíc jsou experimenty s reálnými systémy někdy neekonomické, nebezpečné, nevhodné nebo vůbec neproveditelné. Proto používáme metod počítačové simulace, která tyto nevýhody nemá. Oblasti použití simulace:

- Biologie a lékařství: např. modely působení léků v organismu, modelování růstu bakterií
- Fyzika: např. model jaderného reaktoru, model šíření zvuku v místnosti
- Chemie: např. modely chemických reakcí, výpočty vlastností látek
- Astronomie: např. model srážky galaxií, simulace pohybu planet kolem Slunce
- Meteorologie: např. modely pro předpověď počasí
- Geologie: např. model zemětřesení
- Technika obecně: např. simulované crash testy automobilů, model mikroprocesoru, modely atomů, simulace elektrických obvodů, simulace číslicových obvodů

3.2 Simulace a simulátor číslicových obvodů

Specializace základních pojmu do problematiky číslicových obvodů by vypadla následovně:

- Modelování je vytváření, vytvoření schéma zapojení a jeho následné zjednodušení pomocí metod minimalizace.
- Simulace je numerické řešení matematických modelů, potažmo schématu číslicových systémů.
- Simulátor je výpočetní systém provádějící numerický výpočet chování modelu.

Hlavní dělení simulace číslicových obvodů je podle rozsahu simulovaných hodnot a podle přístupu k simulaci zpoždění.

3.2.1 Rozdělení simulátorů dle počtu simulovaných stavů

Na začátku této podkapitoly je asi nejvhodnější prostor pro zmínku o existenci rezoluční (rozhodovací) funkci, která se používá k řeší konfliktů mezi více budiči jednoho signálu. Bývá zvykem takovouto funkci dodat do specifikace číslicového obvodu nejčastěji formou tabulky.

Rozdělení dle počtu stavů pak vypadá následovně:

- dvoustavové (0,1) - nevystihují všechny události, které v simulovaném obvodu mohou nastat
- třístavové (0,1,X) - X: je symbol pro stav neznámé hodnoty, například pokud vstup přichází z jednoho konce hodnota „0“ a z druhého konce hodnota „1“
- čtyřstavové (0,1,X,Z) – Z je symbol pro stav vysoké impedance

3.2.2 Rozdělení dle zohlednění zpoždění

Ještě před samotným rozdělením, zde uvedu základní pojmy spojené se zpožděním:

Zpoždění hradla -základní komponenty číslicových obvodů, hradla, stejně jako všechny reálné elektrické součástky, mají určité zpoždění, vymezující dobu, po kterou probíhá vyhodnocení výstupu komponenty.

Hazard - krátká neočekávaná změna výstupního signálu, způsobená šířením signálu ze vstupu na výstup různými, ale konvergujícími cestami, které mají různé časové ohodnocení, způsobené zpožděním hradel.

Statický hazard - výstup má být trvale v 0 nebo 1, místo toho se objeví krátká změna do opačné úrovně. Tento hazard je způsoben dvěma komplementárními signály, které jsou z důvodů různých zpoždění stejné, ačkoliv stejné být nemají. Změna 0 → 1 → 0 = statický hazard v nule, Změna 1 → 0 → 1 = statický hazard v jedničce Statické hazardy můžeme odhalit v časovém diagramu nebo v mapě.

Dynamický hazard - nastane, jestliže dva signály, které mají být stejné, na chvíli stejné nejsou. Tato situace nastane, když se proměnná šíří ze vstupu na výstup různými cestami s různým zpožděním. Dynamický hazard se projeví při změně výstupu (tj. 0 → 1 → 0 → 1 nebo 1 → 0 → 1 → 0). Dynamický hazard nelze zjistit z mapy, ale lze ho zjistit z několika map.

Náběžná hrana signálu – změna signálu z hodnoty „0“ na hodnotu „1“.

Sestupná hrana signálu – změna signálu z hodnoty „1“ na hodnotu „0“.

Předstih - doba před aktivní hodinovou hranu, kdy už musí být vstup stabilní.

Přesah - doba po aktivní hodinové hraně, kdy musí být vstup ještě stále stabilní.

Dopravní zpoždění - Zachovává všechny pulzy, pouze je zpozdí - posune. Vhodné pro modelování vodičů, důležitou vlastností tohoto druhu simulace je, že se neztrácí žádné pulzy.

Setrvačné zpoždění - modeluje reálné součástky (hradla). Namísto kolmých náběžných a sestupných hran pulsů šikmé, pozvolné hrany. Filtruje úzké pulsy (puls "nestihne" dosáhnout log. 1).

3.2.2.1 Synchronní simulace

Synchronní simulace neuvažuje zpoždění, pro se zde dá použít metoda pevného časového kroku, to znamená že jediná změna výstupu může nastat při změně vstupu, není zde žádný časový interval, ve kterém se hodnota dostává postupně od vstupu k výstupu.

Řízení synchronní simulace lze oddělit výpočet kombinační části a paměťové části. Obecný algoritmus má dvoufázový průchod, zaprvé výpočet a zapamatování si nových hodnot a zadruhé přenos nových hodnot do skutečných výstupů

3.2.2.2 Asynchronní simulace

Asynchronní simulace zpoždění uvažuje. Musí se zde použít metoda proměnného časového kroku, čas, kdy dochází ke změně některého z výstupů je ovlivněn nejen krokem změny vstupů, ale i zpožděním s jakým se promítá vliv změněného vstupu na výstup.

Asynchronní simulace představuje podrobnější přístup k simulaci systému, umožňuje zjistit statické a dynamické hazardy, umožňují ověřit dodržení předstihů a přesahů, a umožňují také ověřit správnost frekvence hodinových pulsů

Druhy zpoždění uváděných pro asynchronní simulaci v [5] jsou následující: jednotkové - stejné pro všechny obvody, násobné – násobky jednotkového nebo libovolné.

3.3 EDIF

Před popisem samotného EDIFu je nutné definovat pojem *netlist*: Slovo netlist lze použít v různých kontextech, já zde o něm budu ale uvažovat pouze ve spojitosti s popisem elektrických obvodů, konkrétně diskrétních obvodů. V této souvislosti se netlist užívá pro popis číslicových obvodů, konkrétně poskytuje informace o jednotlivých komponentách obvodu a jednotlivých spojích mezi komponentami, jinými slovy definuje komponenty a jejich vzájemné propojení. Toto může být mnohdy nedostačující popis, proto se číslicové obvody popisují i jinými způsoby, například jazykem VHDL či Verilog.

EDIF (akronym z anglického „Electronic Design Interchange Format) je potom standardem stanovujícím formát výše definovaného netlistu. První verze byla vytvořena v roce 1985, první standardizovaná verze, standard IEEE, byla verze 2 0 0. Verze 2 0 0 byla vydána v roce 1988 a konkrétně jí popisuje standard ANSI/EIA-548-1988. Na tuto verzi dále navázaly verze 3 0 0 (rok vydání 1993) a 4 0 0 (rok vydání 1996).

Popis stěžejní syntaxe EDIFu a příklad netlistu v tomto formátu je uveden v příloze č. 3 a příloze číslo 4.

4 Návrh simulátoru

4.1 Specifikace zadání

Byl zadán požadavek na simulaci jednoduchých kombinačních a sekvenčních obvodů. Jednoduchost byla specifikována omezením na počet primárních vstupů a výstupů. Dodatečná specifikace se týkala požadavku na formát vstupních obvodů. Tento simulátor si tato omezení definoval následovně:

Obvod může obsahovat maximálně 64 primárních vstupů, 64 primárních výstupů a 128 komponent. Vstupní obvod pro simulaci musí být zapsán ve vstupním souboru ve formátu EDIF. Pokud chceme dodefinovat zpoždění pro jednotlivé komponenty vstupního, je nutné dopsat je explicitně do speciálního souboru. Pokud nechceme obvod simulovat na všechny možné kombinace vstupů generované automaticky musíme dodat speciální soubor s vlastními předpřipravenými hodnotami vstupních vektorů. Pokud chceme simulaci ukončit , před projitím všech vstupních vektorů (ať už generovaných automaticky nebo explicitně) je nutné dodatečně zadat čas skončení simulace jako argument při spuštění programu. Další informace o implicitních předvolbách simulátoru už nejsou natolik stěžejní abych je zde opakoval, všechna implicitní nastavení simulátoru a možnosti jejich změny jsou popsané v příloze č. 1.

4.2 Vlastní návrh a implementace

Rozhodl jsem se tento simulátor navrhnout pro simulaci tříhodnotového druhu, (0, 1, X), komponent co možná nejbližších realitě, z tohoto důvodu simulátor uvažuje i zpoždění těchto komponent.

Dalším velmi důležitým rozhodnutím pro návrh simulátoru byla volba formátu pro zápis vstupního simulovaného obvodu. Tato volba byla specifikována v zadání. Od této volby se odvíjeli návrh a implementace překladače, což nakonec stálo stejné úsilí jako návrh a implementace vlastního simulátoru. Překladač je navržen tak, aby jakkoliv nekorektní schéma nepustil k samotnému simulátoru a ukončil běh programu se stručnou chybovou hláškou vypisovanou na standardní chybový výstup.

Výstupem simulace je jednoslovná informace o úspěchu, či neúspěchu simulace podávaná na standardní výstup a především soubor s výsledkem simulace. Formát výstupního souboru oproti formátu vstupního souboru nebyl specifikován, porto je zde zvolena vlastní forma zápisu. Stručný popis formy zápisu: hodnoty všech primárních vstupů a výstup pro daný simulační čas jsou zaznamenány na jeden řádek začínající hodnotou tohoto času. Čili záznamy pro různé časy jsou mezi sebou odděleny konci řádků a záznamy pro jeden daný čas jsou mezi sebou odděleny tabulátory, respektive mezerami. Tato forma výstupu je volena cíleně, jelikož je velice vhodná pro další zpracování pomocí některého z dávkovacích programů, například gnuplot, který z dat dokáže snadno a rychle vytvořit přehledný graf (obecné bohatě okomentované šablony pro tvorbu grafu přes program gnuplotu jsou dodávány s programem v sekci examples).

Protože simulátor musí podporovat spoustu nastavení definujících průběh simulace a několik pro další doplnění definice simulovaného obvodu, bylo nutné vytvořit rozumný návrh pro získávání těchto voleb. Tento návrh, je zde vyřešen kombinaci získávání hodnot ze vstupních parametrů příkazové řádky a získávání hodnot z implicitních vstupních souborů. Mezi implicitní soubory patří: *delays.txt* a *input_variations.txt*. Jak již překlad názvů těchto souborů napovídá, soubor *delays.txt* obsahuje informace o zpoždění komponent použitých v simulovaném obvodu. Pouhé nahlédnutí do tohoto souboru stačí k pochopení jednoduchých pravidel pro vytvoření vlastních záznamů. Druhý standardní soubor *input_variations.txt* plní funkci explicitního generátoru hodnot pro primární vstupy simulovaného obvodu. Pojem explicitní generátor je chápan jako soubor s předem vygenerovanými hodnotami, které jsou zapsány dle jednoduché syntaxe (definice formátu zápisu hodnot je uvedena v hlavním komentáři původního souboru *input_variations.txt*. Druhou metodou zisku voleb o které byla řeč, je získávání z argumentů příkazové řádky. Tato metoda zde nebude rozebírána, jelikož celou problematikou parametrů příkazové řádky se zabývá příloha č. 1.

Následuje konkrétní popis simulační rutiny v pseudokódu:

```
SimulujObvod(
  vlastní struktura: uživatelskáNastaveníZPříkazovéŘádky,
  ukazatel na soubor: výstupníSoubor,
  vlastní typ pro chyby: příznakChyby
)
Začátek rutiny
Vyhodnoť uživatelská nastavení
Nuluj počet průchodů
Nuluj příznak konce
Nuluj příznak výpisu
Dokud je aktuální čas menší než celkový čas a současně kalendář není prázdný a
současně počet průchodů je menší než maximální povolený
  Inkrementuj počet průchodů
  Pokud je naplánována v aktuálním čase inkrementace
    Pokud je nastaven příznak konce
```

```
      Vyskoč z cyklu
    Inkrementuj vstupy
    Pokud není zadána volba příkazového řádku nastavující čas simulace a současně
    pokud bylo dosaženo konce souboru se vstupními hodnotami
        Nastav příznak konce
  Pokud je naplánována změna CLK vstupu
    Proveď změnu CLK vstupu
  Pokud uživatel zvolil příznak chudého výstupu
    Pokuj je naplánována událost výpis hodnot
      Nastav příznak výpisu
  Jinak (pokud není zvolen příznak chudého výpisu)
    Nastav příznak výpisu
  Dokud je naplánována událost aktivace výstupu komponenty
    Vyjmi z kalendáře první výstup k aktivaci
    Přidej do seznamu všechny spoje, které ovlivnil aktivovaný výstup
  Konec vnořeného cyklu
  Obslož všechny ovlivněné spoje
  Obslož všechny ovlivněná hradla
  Pokud obsluha hradel ovlivnila další spoje
    Pokračuj na začátek hlavního cyklu
  Pokud čas následujícího záznamu kalendáře je rozdílný od času aktuálního
    Pokud je nastaven výpis
      Zaznamenej hodnoty do výstupního souboru
Konec hlavního cyklu
Návrat z rutiny
```

4.2.1 Návrh datových struktur

Byla navržena řada datových struktur, je možné je rozdělit z hlediska účelu použití na:

1. Datové struktury pro uložení simulovaného obvodu: v hierarchii nejvýše postaveným zástupcem této skupiny je struktura pro uložení všech prvků ze kterých je simulovaný obvodu složen, mezi její základní členy patří seznam komponent, seznam vstupních spojů, seznam výstupních spojů, seznam vnitřních spojů a záznam o časovém spoji (spoj, který není ovlivňován generátorem vstupních vektorů). Dalším zástupcem pro tuto skupinu je struktura zaštiťující informace týkající se jedné komponenty, mezi členy této struktury patří informace o druhu komponenty (and, or, jk…), informace o vstupech a výstupech komponenty, informace o zpoždění komponenty a informace o stavu komponenty. Posledním důležitým zástupcem je datová struktura pro uložení jednoho spoje tvořená záznamy informujícími o aktuální hladině spoje (0, 1, X) a o vstupech a výstupech spoje. Mezi metody navržené pro práci s tímto druhem datových struktur patří pouze metody realizující inicializaci a uvolňování záznamu.
2. Struktury pro vlastní implementaci hlavního algoritmu překladače: za prvé sem patří struktury nesoucí dodatečné informace ke všem strukturám z předešlé kategorie, týkajících se především označení ve schématu. Dále sem patří struktura používaná k uložení voleb pro jednotlivé bloky, mezi její záznamy patří typ EDIF bloku, zanoření bloku, stav bloku a záznamy definující podporu pro různé konstrukce bloku. Mezi metody pro touto skupinou dat patří nejen metody pro inicializaci a uvolňování, ale i metody řešící korektnost bloku, korektnost pořadí bloků a korektnost schématických značek.
3. Struktura pro uchování uživatelských nastavení průběhu simulace: Do této skupiny patří struktura pro uchovávání uživatelských dat, zadaných přes příkazovou řádku. Záznamy této struktury v podstatě přesně kopírují podporované parametry příkazového řádku plus ukládají dodatečné informace k parametrům, které si to žádají. Tato skupina nevyžaduje implementaci speciálních metod, postačily metody pro inicializaci a uvolňování záznamu.

4. Struktury nutné pro vlastní implementaci simulačního algoritmu: hierarchicky nejvýše postaveným zástupcem této skupiny je struktura nesoucí informaci o seznamu událostí, které se mají v budoucnu vykonat, této struktuře se v simulační terminologii přezdívá *kalendář*. Další v hierarchii je struktura zapouzdřující informace jedné události. Obsahem této struktury jsou záznamy pro čas, na který je událost plánována, druh plánované události a seznam dat, nad kterými se mají dané události provést (prvek seznamu dat se skládá z reference na měněný prvek obvodu a informace o typu události, respektive typu akce, která se má s prvkem obvodu provést). V této skupině si nevystačíme s pouhými základními metodami pro inicializaci a uvolňování záznamů, proto zde bylo nutné napsat doplňující metody, mezi hlavní patří: funkce pro vkládání nových záznamů s jejich zařazením na správné místo, výběr prvního záznamu s nejmenším časem aktivace, výběr požadovaného konkrétního záznamu, toto je nutné v případě, že potřebujeme již naplánovanou událost z kalendáře zrušit.

4.2.2 Implementace

Program je konzolová aplikace napsaná v čistém jazyce C. Program neobsahuje universální binární soubor, je nutného jeho přeložení před prvním spuštěním na každé stanici, k přeložení poslouží skript Makefile, standardně dodáván se zdrojovými soubory tohoto programu. Simulátor je spouštěn příkazem ./simul v operačním systému Linux a příkazem simul v operačním sytému Microsoft Windows. Snažil jsem se program vytvořit tak, aby při spuštění bez argumentů vyhovoval co nejvyšší škále uživatelů, přesto mi ale bylo jasné, že pro důkladnější testování, je možnost volby nutnost. Proto program podporuje dosti uživatelských voleb, zadaných při spuštění simulátoru jako argumenty příkazového řádku. Pro rychlé zorientování v pravidlech správného používání argumentů je vhodné si před prvním spuštěním programu přečíst přílohu č. 5. Podrobnější popis podporovaných argumentech naleznete v příloze č. 1.

Vlastní program je tvořen ze dvou částí: první část, překladač, se stará o naplnění paměťových struktur (použitým datovým strukturám je věnován poslední odstavec této podkapitoly), dle zadaného vstupního schématu. Druhá část, simulátor, se stará o vlastní simulování obvodu.

Překladač zpracovává pouze základní příkazy EDIF, ostatní klíčová slova vyhodnotí jako nepodporované bloky (více o EDIFu a jeho o podporovaných a nepodporovaných příkazech a jejich omezeních, respektive přizpůsobeních, naleznete v příloze č. 4). Překladač však kontroluje správnou souslednost všech příkazů, kontroluje správnost zanořování, respektive vynořování do, respektive z jednotlivých bloků, kontroluje správné definice a deklarace komponent, portů a spojů. Ve všech blocích, podporovaných i nepodporovaných, probíhá kontrola výskytu klíčových slov na nesprávných pozicích.

Druhá část programu, simulátor očekává již správně naplněné datové struktury nesoucí informace o simulovaném obvodě. Takto naplněné datové struktury vyhodnotí simulačním algoritmem. Podrobnost výpisu závisí na uživatelských nastaveních a má mírný vliv na rychlost simulace. Výpis je prováděn do souboru ve formátu popsaném v předešlé sekci.

Simulovaný obvod je po přeložení uchováván v hlavní dynamické struktuře `TSimulovanyObvod`, popsané v předešlé sekci jako struktura pro uložení všech komponent a spojů simulovaného obvodu.

Program je navržený pro snadné přidávání vlastních komponent pomocí přidání vlastních modulů. Pro napsání vlastních modulů stačí prozkoumat, jak jsou koncipovány moduly stávající, (*basic_operations.c*, *rs.c*, *jk.c*, *d.c*, *t.c*). Po pochopení základních zásad psaní modulu dle příkladů, stačí napsat vlastní modul a v hlavním simulačním modulu rozšířit podporovanou instrukční sadu o nově připsané funkce. Detailněji je problematika rozšíření instrukční sady popsána v příloze 2.

Metriky kódu:

- Počet zdrojových souborů: 17
- Počet hlavičkových souborů: 20
- Počet spustitelných souborů: 1
- Počet modulů: 17
- Počet řádků celkem: 9539
- Procentní zastoupení kódu: 0,52
- Procentní zastoupení komentářů 0,26
- Procentní zastoupení prázdných řádků: 0,21

4.3 Experimentální ověření simulátoru

Pro ověření simulátoru jsem si připravil několik demonstračních příkladů (nachází se v příloze č. 6). V této sekci uvedu pouze čtyři z těchto příkladů: příklad s kombinačním obvodem, příklad s asynchronním sekvenčním obvodem, příklad se synchronním sekvenčním obvodem a přiklad realizace klopného obvodu.

Testování proběhlo na notebooku:

Procesor:	Intel Core Solo T1350, 1.86 GHz
RAM:	2048MB, sdílená
Operační systém:	Linux, distribuce Debian.

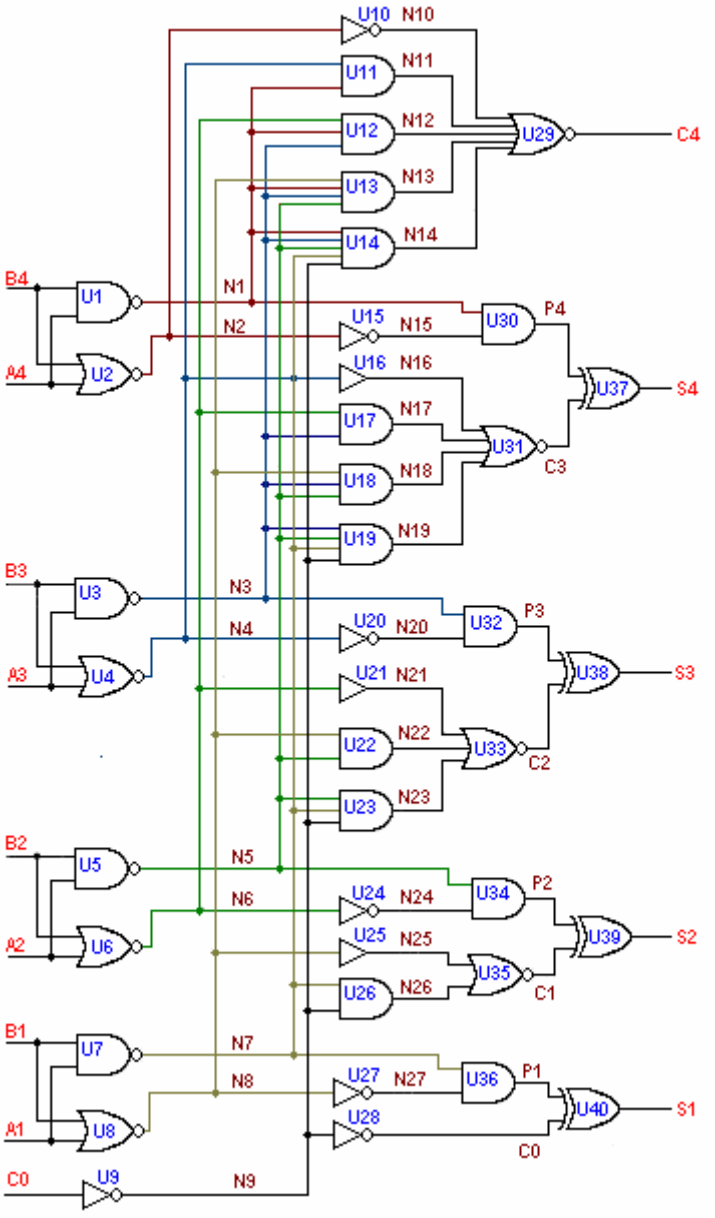
4.3.1 Příklad s kombinačním obvodem

Jako příklad na kombinační obvod jsem si vybral čtyřbitovou sčítačku MSI 7483 s urychleným výpočtem carry bitu (vysvětlení např. viz. [4]), carry není generováno postupně, ale paralelně, příklad převzatý z [4].

Rovnice příkladu:

$$c_i = c_{i-1} \cdot a_i + c_{i-1} \cdot b_i + a_i \cdot b_i$$

Schéma příkladu:



Obrázek 4.1 Schéma - 4 bitová sčítačka s paralelním carry

Umístění schéma přepsaného do vstupního formátu na CD:

```
CD:\prakticka_implementace\examples\01-add_4b\01.edif
```

Spuštěná dávka:

```
time ./simul --inc=1 -i 01.edif -eig e01-input_variations.txt -o 01.dat
```

Výstup dávky

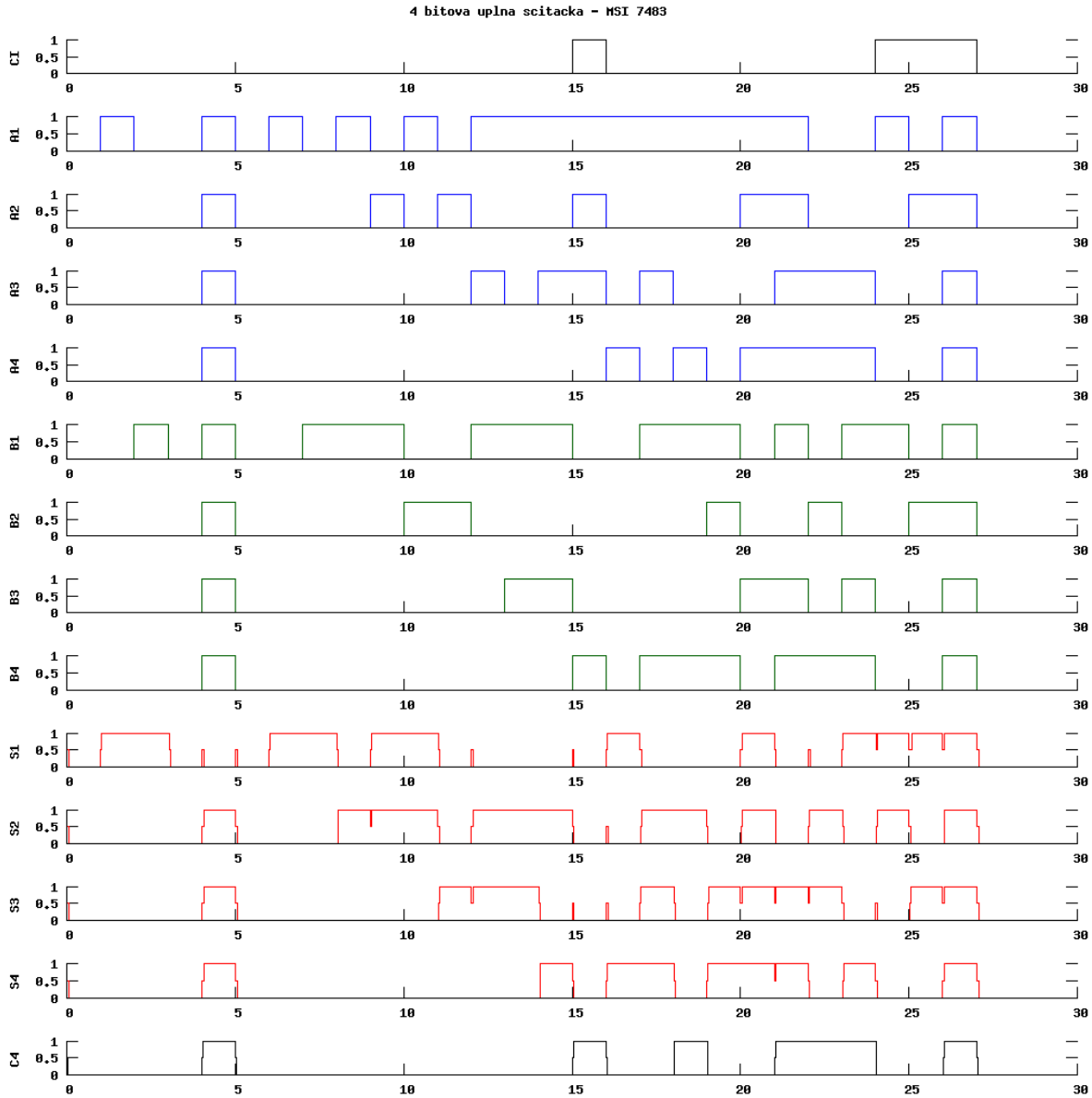
```
SUCCESS


real    0m0.005s
user    0m0.004s
sys     0m0.000s
```

Ukázka několika řádku výstupu:

```
time [ms]       A1  A2  A3  A4  B1  B2  B3  B4  CI      S1  S2  S3  S4  C4
0.000000        0   0   0   0   0   0   0   0   0       0.5 0.5 0.5 0.5 0.5
0.020000        0   0   0   0   0   0   0   0   0       0.5 0.5 0.5 0.5 0.5
```

0.040000	0	0	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	0
0.060000	0	0	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	0
0.080000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1.000000	1	0	0	0	0	0	0	0	0	0.5	0	0	0	0
1.020000	1	0	0	0	0	0	0	0	0	0.5	0	0	0	0
1.040000	1	0	0	0	0	0	0	0	0	1	0	0	0	0
2.000000	0	0	0	0	1	0	0	0	0	1	0	0	0	0

Grafické zobrazení výstupu:



Obrázek 4.2 Průběh simulace k příkladu 3

Pro tento příklad jsem se rozhodl provést zátěžový test s nastavením doby simulace na extrémní hodnotu. Z pochopitelných důvodů zde nebudu uvádět grafický výstup a ukázku několika řádků výstupu. Spouštěná dávka a výstup z této dávky potom vypadá následovně:

Spuštěná dávka:

```
time ./simul -i 06.edif -eig 06-input_variations.txt --inc=0.2 --time=3600000
```

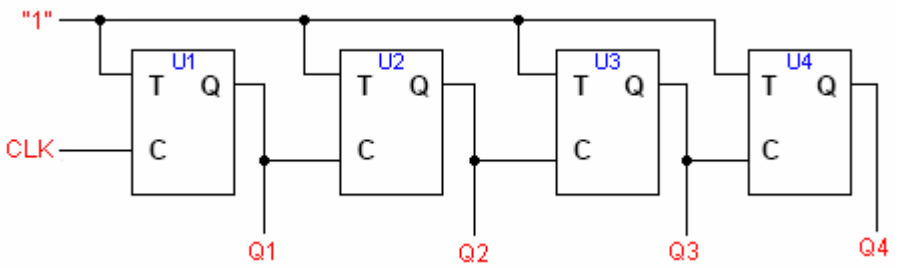
Výstup dávky

```
Pocet zaznamenanych radku: 58400021
SUCCESS

real    3m58.825s
user    3m7.764s
sys     0m5.824s
```

4.3.2 Příklad s asynchronním sekvenčním obvodem

Jako příklad na asynchronní sekvenční obvod jsem si vybral čtyřbitový čítač, příklad je z [5].

Schéma příkladu:



Obrázek 4.3 Schéma - asynchronní 4 bitový čítač

Umístění schéma přepsaného do vstupního formátu na CD:

```
CD:\prakticka_implementace\examples\sekv03-asynchronni_citac\input-sekv03.edif
```

Spuštěná dávka:

```
time ./simul --inc=1 --time=100 -i 02.edif -eig 02-input_variations.txt -o
02.dat
```

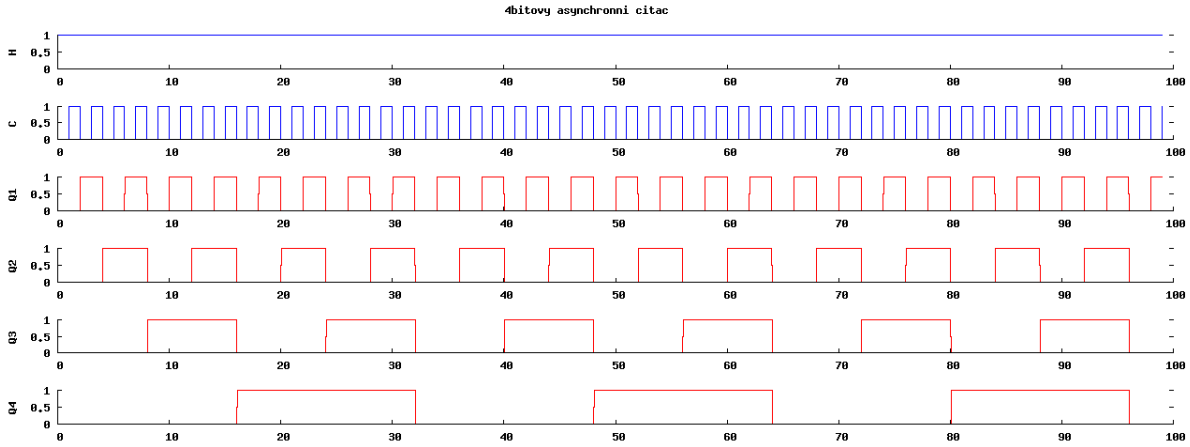
Výstup dávky

```
SUCCESS

real    0m0.006s
user    0m0.000s
sys     0m0.008s
```

Ukázka několika řádku výstupu:

```
time [ms]       H   CLK       Q1   Q2   Q3   Q4
0.000000        1   0         0    0    0    0
1.000000        1   1         0    0    0    0
2.000000        1   0         0.5  0    0    0
2.020000        1   0         1    0    0    0
3.000000        1   1         1    0    0    0
4.000000        1   0         0.5  0    0    0
4.020000        1   0         0    0.5  0    0
4.040000        1   0         0    1    0    0
5.000000        1   1         0    1    0    0
```

Grafické zobrazení výstupu:

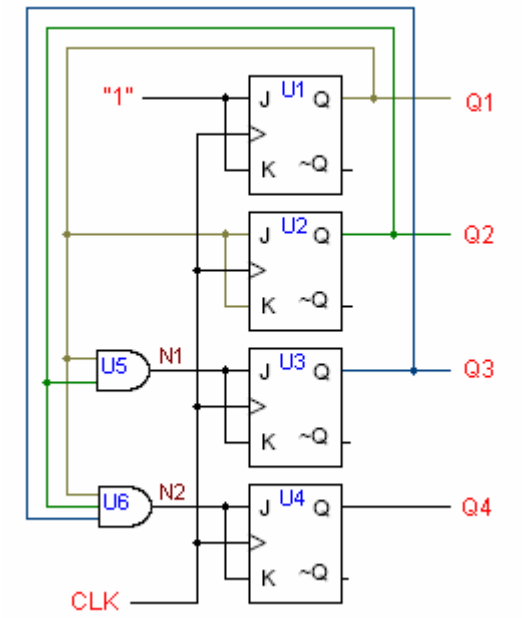


Obrázek 4.4 Průběh simulace k příkladu 1

4.3.3 Příklad se synchronním sekvenčním obvodem

Jako příklad na synchronní sekvenční obvod jsem si vybral úmyslně také čtyřbitový čítač, v rámci srovnání.

Schéma příkladu:



Obrázek 4.5 Schéma - synchronní 4 bitový čítač

Umístění schéma přepsaného do vstupního formátu na CD:

```
CD:\prakticka_implementace\examples\03-count_syn\03.edif
```

Spuštěná dávka:

```
time ./simul --inc=1 --time=100 -i 03.edif -eig 03-input_variations.txt -o 03.dat
```

Výstup dávky

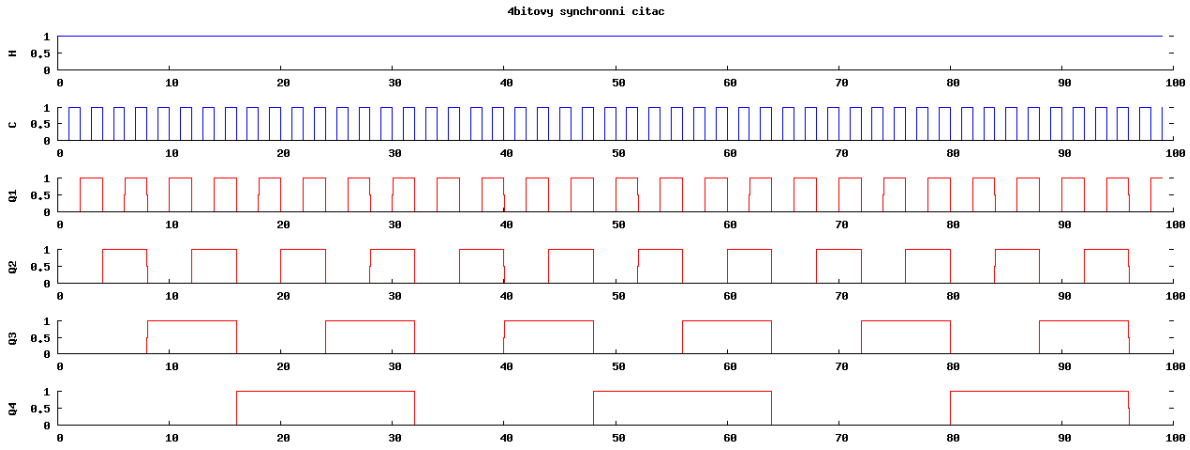
```
SUCCESS
```

```
real    0m0.010s
user    0m0.004s
sys     0m0.000s
```

Ukázka několika řádku výstupu:

```
time [ms]       H   CLK       Q1   Q2   Q3   Q4
0.000000        1   0         0    0    0    0
1.000000        1   1         0    0    0    0
2.000000        1   0         0.5  0    0    0
2.020000        1   0         1    0    0    0
3.000000        1   1         1    0    0    0
4.000000        1   0         0.5  0.5  0    0
4.020000        1   0         0    1    0    0
4.040000        1   0         0    1    0    0
5.000000        1   1         0    1    0    0
```

Grafické zobrazení výstupu:

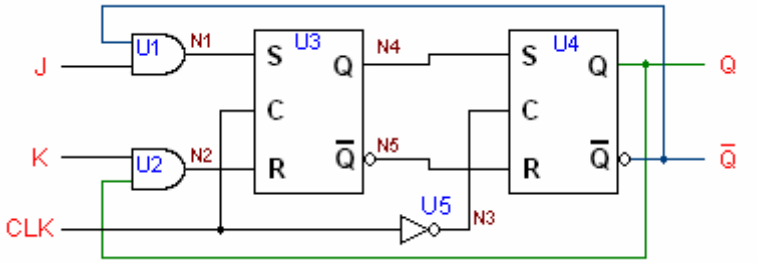


Obrázek 4.6 Průběh simulace k příkladu 2

4.3.4 Příklad s klopným obvodem

Jako příklad na klopný obvod jsem si vybral dvoufázovou implementaci dvoufázového klopného obvodu JK, pomocí dvou RS klopných obvodů.

Schéma příkladu:



Obrázek 4.7 Schéma – JK KO dvoufázový

Umístění schéma přepsaného do vstupního formátu na CD:

`CD:\prakticka_implementace\examples\06-jk_2phase\06.edif`

Spuštěná dávka pod Linuxem:

```
time ./sim09 --inc=0.25 -i 06.edif -eig 06-input_variations.txt -o 06.dat
```

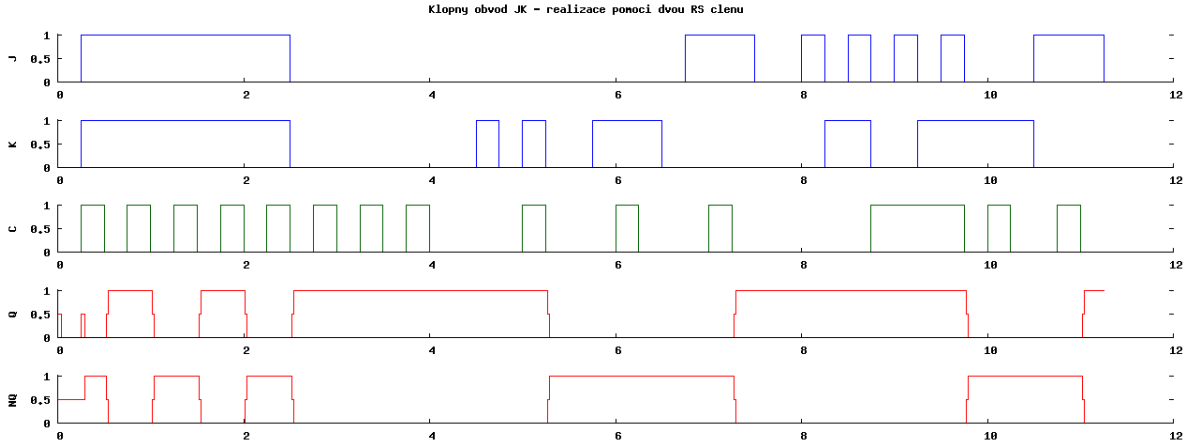
Výstup dávky

```
SUCCESS


real    0m0.029s
user    0m0.004s
sys     0m0.004s
```

Ukázka několika řádku výstupu:

time [ms]	J	K	C	Q	NQ
0.000000	0	0	0	0.5	0.5
0.020000	0	0	0	0.5	0.5
0.040000	0	0	0	0	0.5
0.250000	1	1	1	0.5	0.5
0.270000	1	1	1	0.5	0.5
0.290000	1	1	1	0	1
0.310000	1	1	1	0	1
0.330000	1	1	1	0	1
0.500000	1	1	0	0	1

Grafické zobrazení výstupu:



Obrázek 4.8 Průběh simulace k příkladu 4

5 Závěr

Se svou prací jsem v konečné fázi spokojený, podařilo se mi splnit základní úkol a to implementovat simulátor, dokonce se mi podařilo i držet se základní motivace pro tvorbu tohoto simulátoru, tedy klást důraz především na rychlost a jednoduchost simulátoru.

Jako další pokračování práce by bylo vhodné použít formát výstupu kompatibilní s některým široce používaným simulačním prostředím, například modelsimem, dále bych doladil překladač EDIFu, aby se stal více univerzálnějším, v třetím kroku bych přistoupil k podpoře více vstupních formátů, například VHDL, ve čtvrtém kroku bych se zabýval prozkoumáním možnosti dalšího vylepšení simulačního algoritmu a na závěr bych znovu prozkoumal možnosti zrychlení překladu. Po absolvování všech těchto kroků bych se možná ještě zamyslel nad tvorbou dalšího kooperujícího programu, který by zprostředkoval jednak tvorbu vstupních souborů skrze grafické návrhové prostředí a jednak grafické zobrazení všech podporovaných výstupních formátů.

Nakonec chci ještě poznamenat, že tento projekt mě zaujal, umožnil mi pracovat na rozsáhlejším projektu, který je na tisíce řádků a pomohl mi uvědomit si, jak může být těžké projekt takového rozsahu uhlídat a jak moc velkou cenu programátor zaplatí, když všechna úskalí důkladně nepromyslí.

Literatura

[1] Sekanina L.: *Evolvable Components. Natural Computing Series,* Springer-Verlag, Berlin, 2004

[2] Wakerly J. F.: *Digital Design*: *Principles and Practices*, Prentice-Hall, 2000

[3] Hordějčuk V.:*Matematika, Booleova algebra*, [online]. c2009. Dostupné na URL: <http://voho.cz/booleova-algebra/>

[4] Fučík O.: *podklady k přednáškám předmětu INC přednášeném na VUT FIT Brno*, [online]. Brno c2003-2006. Dostupné na URL: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/INC-IT/lectures>

[5] Anonymní zdroj, [online]. Dostupné na URL: <http://lob.felk.cvut.cz/x36lob/downloads/s_lob10.pdf>

[6] Peringr P.: *Modelova´nı´ a simulace - Studijnı´ opora*, [online].Brno c2008. Dostupné na URL: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IMS-IT/texts/opora-ims.pdf>

Seznam příloh

Příloha 1. CD:\prakticka_implementace\doc\Readme.txt
Příloha 2. CD:\prakticka_implementace\doc\OwnModules.txt
Příloha 3. CD:\prakticka_implementace\examples\06-jk_2phase\06.edif

Následující přílohy jsou pouze v elektronické podobě:
Příloha 4. CD:\prakticka_implementace\doc\Minimum_jak_napsat_vstupni_edif.txt
Příloha 5. CD:\prakticka_implementace\doc\QuicStar.txt
Příloha 6. CD:\prakticka_implementace\src\“zdrojové soubory“
Příloha 7. CD:\prakticka_implementace\src\examples\

Příloha 1

```
 .oooooo..o            o8o                                                  oooo
d8P'    `Y8            `"'                                                  `888
Y88bo.                 oooo          ooo. .oo.  .oo.           oooo  oooo    888
 `"Y8888o.             `888           `888P"Y88bP"Y88b          `888  `888    888
     `"Y88b             888            888   888   888           888   888    888
oo     .d8P             888            888   888   888           888   888    888
8""88888P'             o888o          o888o o888o o888o          `V88V"V8P'   o888o
```

Tento program získává veškeré informace o uživatelských potřebách simulace přes argumenty příkazového řádku.

Podporuje následující parametry:

-i „cesta k vstupnimu souboru“

Volitelný parametr, pokud není zadán, je brán defaultní soubor: /examples/00-default/00.edif

Pokud je parametr zadán, musí být bezprostředně následován cestou k vstupnímu souboru.

priklad: ./simul -i examples/01-add_4b/01.edif

-o „cesta k vystupnimu souboru“

Volitelný parametr, pokud není zadán bude výstup posílán do implicitního souboru vystup.dat

Pokud je parametr zadán, musí být bezprostředně následován cestou k výstupnímu soboru

Pozor: simulátor neřeší složkování, proto musí cílová složka předem existovat!

priklad: ./simul -o examples/01-add_4b/01.dat

-eig „cesta k soubrou s predpripravenymi vstupnímy vektory“

Volitelný parametr, pokud není zadán bude simulátor postupně generovt všechny možné kombinace vstupů

Pokud je parametr zadán, sám, bez zadání cesty k souboru, bude brán jako soubor s externě definovanými vstupními vektory soubor input_variations.txt

Parametr ale může být zadán s bezprostředně následující cestou k souboru s externími kombinacemi vstupů, poté jsou vstupy čerpány z tohoto souboru.

priklad: ./simul -o examples/01-add_4b/input_variations.txt

--dont-show-hazards

Volitelny parametr, pokud je zadan vypina se detekce hazardu, coz zrychluje beh vlastni simulace

--dont-show-x

Volitelný parametr, vyuziti si najde hlavne u obvodu ve kterych pouzijeme hradla s nulovym spozdenim, tento parametr urychli simulaci

--inc=(value [ms])

Parametr pro urceni casoveho kroku, se kterym se maji generovat vstupy, pokud nebude zadan simulator spocita krok co nejvice vyhovujici, tak aby se stihli zmeni na vstupu projevit na vystupu ale soucasne aby zmeny na vystupu nevisely zbytecne dlouho

--time=(value [ms])

Parametr pro urceni celkove delky simulace, pokud nebude zadan simulator spocita sam tuto hodnotu, a to tak aby to co nejlepe vyhovovalo danemu schematu. To znamena aby se stihly vygenerovat vsechny kombinace vstupu a soucasne aby se vsechny tyto kombinace stihli v obvodu projevit

Naselduji tri volitelne parametry pro ty kterym nestaci vypisy hodnot pri zmene, ale chteji si vypisi nejak sesynchornizovat, ci chteji vypisi pouze po jedne vterine od toho a toho casu
Bez techto parametru budou ve vystupnim souboru zaznamenany vsechny udalosti v nichz se menil nejaky vstup ci vystup

--report-step=(value [ms])

Volitelny parametr pro frekvenci vypisu pozadovanych hodnot, pokud neni zvolen nastavi se na stejnou frekvenci jako krok se kterym se generuji vstupni hodnoty

--report-delay=(value [ms])

Volitelny parametr pro zpozdeni s jakym ma zacit vypisovani pozadovanych hodnot, nejlepe vyhovuje kdyz mu nastavime hodnotu jako ma nejvetsi zpozdeni hradla v obvodu pote a soucasne parametr poor-output, pote vystupni soubor obsahuje pouze platne hodnoty s pravidelnym intervalem

--poor-output

Volitelny parametr, jeho zadanim vypneme veskere standartni vypisy a budou se provadet pouze vypis urcene parametry report, pokud neni zadan ani jeden parametr report, teto parametr je ignorovan.

Příloha 2

Zásahy v implicitních zdrojových kódech nutných pro rozšíření instrukční sady

Pro přidáni vlastního modulu je nutný zásah v následujících implicitních zdrojových souborech: *hlavni_rozhrani.h, simulator.c, main.c a hlavni_rozhrani.h*

Slovní popis úprav v souborech:

- *hlavni_rozhrani.h*: zde je nutné přidat na konec výčtového seznamu `E_SimOperace` konstantu pro přidávanou komponentu s indexem o jedna větším, než má konstanta předcházející.
- *main.c*: zde je nutné přidat na konec textového řetězce `g_PREFIXY` název, respektive zkratku odvozenou z názvu komponenty.

  Omezeni plynoucí pro tuto akci:

  1. je nutné použít kombinaci pouze dvou písmen. Tato kombinace musí být v řetězci jedinečná
  2. je nutné novou zkratku přidat na konec řetězce, z důvodu kompatibility mezi indexem nové zkratky, respektive nového podřetězce, v řetězci všech zkratek a mezi konstantou ve výčtovém typu `E_SimOperace`
- *simulator.h*: zde je nutné inkrementovat define konstantu `SIM_POCET_PODPOROVANYCH_OPEARACI` o počet přidávaných operací (ve většině případech o jedna).
- *simulator.c*: zde je nutné doplnit pole funkcí `OperaceHradel` o nově vytvořenou vývozní funkci modulu, respektive hlavni funkci komponenty

Praktická ukázka úpravy v souborech:

úpravy pro soubory:

hlavni_rozhrani.h:

```
typedef enum eSimOperace {
  SIM_AND  = 0,
  SIM_OR   = 1,
  SIM_XOR  = 2,
  SIM_NAND = 3,
  SIM_NOR  = 4,
  SIM_NEG  = 5,
  SIM_RS   = 6,
  SIM_D    = 7,
  SIM_JK   = 8,
  SIM_T    = 9,
  SIM_BF   = 10,
  SIM_NK   = 11
} E_SimOperace;
```

main.c:

```
const char g_PREFIXY[50] = "|AN|OR|XO|NA|NO|NG|RS|DD|JK|TT|BF|NK|";
```

simulator.h:

```
#define SIM_POCET_PODPOROVANYCH_OPERACI 12 //původně 11
```

simulator.c:

```
void InicializujSimulatorMemory(TSimulovanyObvod *simObvod, TSimulatorMemory
*simMem) {
  …
  simMem->OperaceHradel[SIM_AND]  = OperaceAND;
  simMem->OperaceHradel[SIM_OR]   = OperaceOR;
  simMem->OperaceHradel[SIM_XOR]  = OperaceXOR;
  simMem->OperaceHradel[SIM_NAND] = OperaceNAND;
  simMem->OperaceHradel[SIM_NOR]  = OperaceNOR;
  simMem->OperaceHradel[SIM_NEG]  = OperaceNEG;
  simMem->OperaceHradel[SIM_RS]   = OperaceRS;
  simMem->OperaceHradel[SIM_D]    = OperaceD;
  simMem->OperaceHradel[SIM_JK]   = OperaceJK;
  simMem->OperaceHradel[SIM_T]    = OperaceT;
  simMem->OperaceHradel[SIM_BF]   = OperaceBuffer;
  simMem->OperaceHradel[SIM_NK]   = NazevFunkceNoveKomponentyZNovehoModulu;
```

Příloha 3

Ukázka jednoduchého netlistu v EDIFu

Ukázka se vztahuje k příkladu uvedeném v sekci 4.3.4:

```
(edif moxon_edif (edifVersion 2 0 0) (edifLevel 0)
 (keywordMap (keywordLevel 0))

 (status
  (written (timeStamp 2009 17 05 13 3 27)
   (program "vim" (Version "6.0"))
   (dataOrigin "moxon design") (author "tom moxon")
  )
 )

 (external generic_gates (edifLevel 0) (technology (numberDefinition))
  (cell an02d1 (cellType GENERIC)
   (view Netlist_representation (viewType NETLIST)
    (interface
     (port A1 (direction INPUT))
     (port A2 (direction INPUT))
     (port Z  (direction OUTPUT))
    )
   )
  )
  (cell rs03d2 (cellType GENERIC)
   (view Netlist_representation (viewType NETLIST)
    (interface
     (port S (direction INPUT))
     (port R (direction INPUT))
     (port C (direction INPUT))
     (port Q  (direction OUTPUT))
     (port NQ (direction OUTPUT))
    )
   )
  )
  (cell ng01d1 (cellType GENERIC)
   (view Netlist_representation (viewType NETLIST)
    (interface
     (port A1 (direction INPUT))
     (port Z  (direction OUTPUT))
    )
   )
  )
 )
```

```
(library DESIGNS (edifLevel 0) (technology (numberDefinition))
 (cell jk-dvoufazovy (cellType GENERIC)
  (view Netlist_representation (viewType NETLIST)
   (interface
    (port J  (direction INPUT))
    (port K  (direction INPUT))
    (port C  (direction INPUT))
    (port Q  (direction OUTPUT))
    (port NQ (direction OUTPUT))
   )

   (contents
    (instance U1
     (viewRef Netlist_representation
      (cellRef an02d1 (libraryRef generic_gates))
     )
    )
    (instance U2
     (viewRef Netlist_representation
      (cellRef an02d1 (libraryRef generic_gates))
     )
    )
    (instance U3
     (viewRef Netlist_representation
      (cellRef rs03d2 (libraryRef generic_gates))
     )
    )
    (instance U4
     (viewRef Netlist_representation
      (cellRef rs03d2 (libraryRef generic_gates))
     )
    )
    (instance U5
     (viewRef Netlist_representation
      (cellRef ng01d1 (libraryRef generic_gates))
     )
    )

    (net J
      (joined
       (portRef J)
       (portRef A2 (instanceRef U1))
      )
    )
    (net K
      (joined
```

```
    (portRef K)
    (portRef A1 (instanceRef U2))
   )
)
(net C
   (joined
    (portRef C)
    (portRef C (instanceRef U3))
    (portRef A1 (instanceRef U5))
   )
)

(net NET1
   (joined
    (portRef Z (instanceRef U1))
    (portRef S (instanceRef U3))
   )
)
(net NET2
   (joined
    (portRef Z (instanceRef U2))
    (portRef R (instanceRef U3))
   )
)
(net NET3
   (joined
    (portRef Q (instanceRef U3))
    (portRef S (instanceRef U4))
   )
)
(net NET4
   (joined
    (portRef NQ (instanceRef U3))
    (portRef R (instanceRef U4))
   )
)
(net NET5
   (joined
    (portRef Z (instanceRef U5))
    (portRef C (instanceRef U4))
   )
)
```

```
      (net Q
        (joined
         (portRef Q)
         (portRef Q (instanceRef U4))
         (portRef A2 (instanceRef U2))
        )
      )
      (net NQ
        (joined
         (portRef NQ)
         (portRef NQ (instanceRef U4))
         (portRef A1 (instanceRef U1))
        )
      )
     )
    )
   )
  )
  (design moxon_edif (cellRef full_adder (libraryRef DESIGNS)))
)
```