VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

# DIPLOMOVÁ PRÁCE

2014 Pavel Veselý

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Simulátor logických obvodů ve webovém prohlížeči

# Web-based Logic Circuits Simulator

2014 Pavel Veselý

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání diplomové práce

Student: **Bc. Pavel Veselý**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Simulátor logických obvodů ve webovém prohlížeči
Web-based Logic Circuits Simulator

Zásady pro vypracování:

Cílem diplomové práce je navrhnout počítačový simulátor, který bude simulovat chování jednotlivých součástek i komplexních elektronických obvodů. Simulátor bude zcela editovatelný a bude možné v něm přes uživatelské prostředí nastavovat prvkům hodnoty. Dostupnost výsledného simulátoru bude možné zajistit na mnoha různých zařízení, bez nutnosti instalace, díky jeho implementaci v prostředí Javascriptu a Html.
Výsledný simulátor logických obvodů mohou využívat především uživatelé zabývající se elektronikou, ale taktéž studenti v rámci výuky daného oboru.

Práce bude obsahovat:
1. Přehled existujících řešení.
2. Návrh řešení a implementace.
3. Implementace simulátoru.
4. Knihovnu součástek.
5. Ukázková zapojení.

Seznam doporučené odborné literatury:

[1] Nicholas C. Zakas, Professional JavaScript for Web Developers, ISBN-13: 978-1118026694, Third ed., Wrox, 2012
[2] Adam Freeman, The Definitive Guide to HTML5, ISBN-13: 978-1430239604, Apress, 2011

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Platoš, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA 1

doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

# ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 23. 4. 2014 .....

**ABSTRAKT**

Diplomová práce ve své teoretické části pojednává o JavaScritpu a HTML5 jako prostředků pro vytvoření webové aplikace. V rámci obou jazyků je popsán jejich vývoj, principy a způsoby použití. Práce také popisuje číslicovou techniku a logické obvody jako podklad pro vytvoření simulátoru logických obvodů.

V praktické části jsou vysvětleny jednotlivé součásti aplikace, jako je paměť webové aplikace; vrstvy plátna; použité logické prvky a chod samotné aplikace.

**KLÍČOVÁ SLOVA**

*HTML5, Canvas, webový prohlížeč, JavaScript, Canvas, datové typy, bitové operátory, logické soustavy, logické funkce, logické prvky, sekvenční logické obvody, kombinační logické obvody, simulace.*

**ABSTRACT**

Diploma thesis in theoretical part deals with JavaScript and HTML5 as a means of creating web application. In connection with both languages is described their development, the principles and methods of use. Thesis also described digital techniques and logic circuits. This is the basis for creating logic circuits simulator.

Practical part explains the individual application components: web application memory, layers canvas; used logic elements and running the application itself.

**KEY WORDS**

*HTML5, Canvas, web browser, JavaScript, Canvas, data types, bitwise operators, logical systems, logical functions, logical elements, sequential logic circuits, combinational logic circuits, simulation.*

## SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| API | Application Programming Interface |
| ARIA | Accessible Rich Internet Application |
| DOM | Document Object Model |
| ECMA | European Computer Manufactures Associations |
| HTML | Hyper Text Markup Language |
| HTTP | Hyper Text Transfer Protocol |
| IE | Internet Explorer |
| IEC | International Electrotechnical Commission |
| IETF | Internet Engineering Task Force |
| JSON | JavaScript Object Notation |
| PHP | PHP: Hypertext Preprocessor |
| SVG | Scalable Vector Graphics |
| VML | Vector Markup Language |
| W3C | World Wide Web Consortium |
| WAI | Web Accessibility |
| WHATWG | Web Hyper Text Application Working Group |
| XHTML | Extensible Hyper Text Markup Language |
| XML | Extensible Markup Language |

**OBSAH**

# ÚVOD

Webové aplikace v současnosti již neodvratně patří k běžnému používání uživatele, stejně jako tomu je u softwarových programů.
S rozmachem moderních webových prohlížečů začali vývojáři vytvářet webové aplikace různého zaměření. Tyto aplikace nahrazují některé softwarové programy v počítači. To ulehčuje uživateli používání aplikace, kterou není nutno instalovat.

V rámci softwarových programů existují různé simulátory, které napodobují skutečné věci stavy nebo procesy. Simulátory se používají v mnoha oblastech zahrnující např. simulaci přírodních systémů s cílem získat poznatky o jejich fungování či ve výrobě s cílem zoptimalizovat výrobní procesy, efektivní plánování a využití pracovníků.
Pravděpodobně největší zastoupení simulací je především ve formě počítačových her. V těchto hrách je možné si vyzkoušet např. řízení dopravních prostředků, farmaření. Simulovat lze téměř všechno od simulací nemocí a jejich rozptylu přes simulaci růstu obyvatelstva až po simulaci elektrického proudu či napětí.

V mém blízkém okolí se nachází několik osob s elektrotechnickým profesním zaměřením. Při vytváření elektrických obvodů vycházejí z načrtnutých pracovních návrhů. Napadla mě tedy myšlenka vytvoření webové aplikace provádějící simulaci elektrických obvodů. Nedokázal jsem však přijít na to, jak tyto obvody simulovat. Proto jsem pro praktickou část práce zvolil simulaci logických obvodů, které nemají tak složité výpočty, jako obvody elektrické.

Vytvořená webová aplikace byla koncipována tak, aby byla přístupna ve všech zařízeních, které disponují moderním webovým prohlížečem.
Simulátor je vytvořen v jazyku HTML a JavaScriptu. Aplikace může být načítána ze vzdáleného serveru i z lokálního disku, a to díky tomu, že obsahuje jen programovací jazyky, které jsou načítány až v prohlížeči uživatele.
Webová aplikace simuluje chování logických prvků (základních, kombinačních, klopných i paměťových) v logickém obvodu.
Celá aplikace je naprogramovaná tak, aby i při použití webového prohlížeče byla optimalizovaná natolik, že zastíní patrný rozdíl v rychlosti a plynulosti oproti softwarovému programu.

Diplomová práce je členěna do tří teoretických kapitol a jedné kapitoly praktické.

První kapitola se soustřeďuje na jazyk HTML5. Je zde popsán jeho vývoj z jazyka HTML do HTML5 a společnosti, které se na jeho vývoji podíleli; principy specifikace a jeho součásti. Následně se kapitola zaměřuje více na rozhraní Canvas, které je součástí specifikace. Jsou zde uvedeny principy rozhraní a způsob jeho využití.
Druhá kapitola popisuje JavaScript. Je zde uvedeno krátké srovnání s jinými jazyky, historie JavaScriptu, standardy (ECMAScript a DOM). Dále je např. popsáno, z čeho se skládá program napsaný v JavaScriptu, jaké jsou prvky klientského JavaScriptu.
Třetí kapitola se zaobírá více elektrotechnikou, a to číslicovou technikou – logickými systémy a funkcemi, kombinačními a sekvenčními logickými obvody.
Poslední kapitola se zaměřuje na samotný projekt *„simulátor logických obvodů ve webové aplikaci“*. Je zde zmíněna podpora aplikace ve webových prohlížečích, používané externí zdrojové kódy, vrstvy plátna vytvořené v aplikaci, paměť aplikace a následně chod samotného simulátoru.

# 1. HTML 5

## 1.1 Vývoj specifikace HTML 5

Návrh jazyka *HTML (HyperText Markup Language)*[1] byl poprvé zveřejněn v roce 1993. Od tohoto roku se postupně objevila verze specifikace 2.3, 3.2 a 4.0. Rok 1999 poté přinesl konverzi 4.01.

Vládu nad specifikací v průběhu jejího vývoje převzalo konsorcium *„W3C" (World Wide Web Consortium).*

Po vzniku výše uvedených čtyř verzí se však zaměření webových standardů specifikace HTML přesunulo k *XML*[2] *(Extensible Markup Language)* a *XHTML*[3] *(Extensible HyperText Markup Language)*. I přes změnu tohoto zaměření však byla většina obsahu na webu stále ve formátu HTML.

Ve snaze povýšit webovou platformu na novou úroveň, založila v roce 2004 komunita lidí skupinu *„WHATWG" (Web HyperText Application Working Group*[4]*)*.

Komunita WHATWG vytvořila specifikaci HTML5 (viz kap. 1. 2) a zahájila práci na nových funkcích specificky cílených na webové aplikace.

Konsorcium W3C se do vývoje HTML5 zapojilo v roce 2006 a v roce 2008 byl zveřejněn první pracovní návrh specifikace HTML5 a následně v roce 2009 XHTML2.

Kromě organizace WHATWG a konsorcia W3C se na odpovědnosti za specifikaci HTML5 podílí také *„IETF"*[5] *(Internet Engineering Task Force)*. Součástí této organizace jsou skupiny odpovědné za internetové protokoly jako je *„HTTP"*[6] *(HyperText Transfer Protocol.)*

## 1.2 Představení specifikace HTML5

HTML5 definuje nové rozhraní WebSocket[7], které staví na protokolu WebSocket, jenž vyvíjí jedna z pracovních skupin organizace IETF.

HTML5 staví na různých principech a specifikacích komunity WHATWG. Mezi tyto principy patří:

1. *Kompatibilita:* jestliže nejsou funkce HTML5 podporované ve webovém prohlížeči, musí dojít k ladné degradaci chování. Je tedy důležitá podpora stávajícího obsahu;
2. *Přínos a upřednostnění uživatelů:* tzn., v případě pochyb, specifikace upřednostňuje uživatele před autory, programátory a těmi, kdo stojí za specifikací (W3C, WHATWG);
3. *Zjednodušení interoperability:* HTML5 je především o jednoduchosti a odstranění zbytečné složitosti. K tomu mu například dopomáhá:
   - nativní podpora ze strany prohlížečů namísto komplexního kódu v JavaScriptu;
   - nový zjednodušený DOCTYPE (typ dokumentu);

[1] HTML je hypertextový značkovací jazyk, který je určen k vytváření dokumentů, obsahujících hypertextové odkazy a pokročilejší formátování. [17]

[2] XML je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Umožňuje snadné vytváření konkrétních značkovacích jazyků pro různé účely a široké spektrum různých typů dat. [19]

[3] XHTML je značkovací jazyk pro tvorbu hypertextových dokumentů v prostředí WWW vyvinutý konsorciem W3C. [19]

[4] Tato komunita vyvíjí specifikací HTML a rozhraní *„API" (Application Programming Interface)* – aplikační programovací rozhraní - pro vývoj webových aplikací. Otevřeně spolupracuje s tvůrci prohlížečů a dalšími zainteresovanými stranami.

[5] IETF vyvíjí a podporuje internetové standardy (specifikace a pravidla pro tvorbu WWW stránek), přičemž úzce spolupracuje s konsorciem W3C. [8]

[6] HTTP je internetový protokol, který definuje pravidla komunikace mezi klientem a serverem. [20]

[7] WebSocket umožňuje vytvořit webovou aplikaci, kde klient (v rámci aplikace v prohlížeči) může navázat obousměrné spojení se serverem, a po tomto spojení si vyměňovat informace v reálném čase. [13]

[7] HTTP je protokol, pomocí kterého putují data od webového serveru k uživateli. [20]

- nová, zjednodušená deklarace znakové sady;
- mocná a přitom jednoduchá rozhraní API[8] (Application Programming Interface) v HTML5.

4. *Obecný přístup:* tento přístup je dělen do tří částí.
   - Přístupnost: v rámci podpory uživatelům s postižením pracuje HTML5 se standardem „WAI" (Web Accessibility Initiative) a „ARIA"[9] (Accessible Rich Internet Application);
   - Nezávislost na médiu: jazyk HTML5 by měl fungovat napříč zařízeními a platformami;
   - Podpora všech světových jazyků: např. nová značka `<ruby>` podporuje anotace „Ruby"[10], které se používají v typografii Východní Asie.

Součástí jazyka HTML5 jsou následující funkce (níže se práce zaměřuje na rozhraní Canvas, který je podstatný pro praktickou část práce):

- Canvas (2D a 3D);
- Výměna zpráv po kanálu;
- Výměna zpráv mezi dokumenty;
- Geolokace;
- MathML[11];
- Microdata;
- Události Server-Sent;
- SVG (ScalableVectorGraphics);
- Protokol a rozhraní API WebSocket;
- Koncept původu webu;
- Webové úložiště;
- Webová SQL databáze;
- Web Workers[12];
- XMLHttpRequest Level 2[13].

## 1.3 Canvas

HTML5 definuje element `<canvas>` jako „bitmapové kreslící plátno závislé na rozlišení, které může být za běhu využíváno pro vykreslování grafů, herní grafiky a ostatních vizuálních prvků". [Pilgrim, M.]

Za účelem vytvoření widgetů[14] pracovní plochy představil původně princip plátna Apple ve svém WebKitu[15] pro Mac OS X.

[8] API je rozhraní pro programování aplikací. Jde o sbírku procedur, funkcí či tříd určité knihovny, které může programátor, využívající knihovnu, používat. [2]
[9] Standardy WAI a ARIA definují způsob, jak vytvářet webový obsah a webovou aplikaci pro osoby se zdravotním postižením. [24]
[10] Ruby je skriptovací interpretovaný programovací jazyk. [11]
[11] MathML je spcifikace, která poskytuje potřebné základy pro zařazení matematických výrazů v internetových stránkách. [26]
[12] Web Workers nabízí standardizovanou cestu, jak mohou prohlížeče spouštět javascriptový kód na pozadí. S pomocí web workers je možno spustit několik „vláken" a ty pak mohou běžet ve stejném čase. [16]
[13] Jedná se o javascriptový objekt, zajišťuje načtení dat z adresy URL bez aktualizace stránky. [28]
[14] Widget je ovládací prvek počítače, který umožňuje komunikaci uživatele s počítačem a práci s daty (např. nabídkové menu, ikona programu, dialogová okna. [27]
[15] WebKit je Open Source knihovna určená k vykreslování webových stránek. [25]

Před příchodem plátna bylo možné v prohlížeči realizovat malování pouze skrze zásuvné moduly, jako jsou např. ty pro Internet Explorer od Adobe pro Flash, SVG (ScalableVectorGraphics) a VML (VectorMarkupLanguage)[16] anebo jiné, čistě javasrctiptové techniky.

Obecný přístup k rozhraní Canvas je pro vývojáře velmi žádoucí a proto vznikají stále nové projekty, snažící se nabídnout podporu plátna i ve starších nestandardních prohlížečích.

### *1.3.1 Princip plátna*

Když je ve stránce použit element `canvas`,vytvoří se v ní obdélníková oblast. Ve výchozím stavu je tato oblast 300 pixelů široká a 150 pixelů vysoká. Je možné však zadat přesnou požadovanou velikost, stejně tak jako jiné atributy elementu `canvas`.

Nejzákladnějším elementem `canvas`, který je možné do stránky HTML5 přidat, je `<canvas> </canvas>`.
Pokud je element `canvas` přidán do stránky, je možno s ním pomocí JavaScriptu jakkoli pracovat, tedy přidávat grafiku, čáry či texty. Je možné v něm malovat či dokonce vkládat pokročilé animace.
Rozhraní Canvas podporuje stejné dvourozměrné grafické operace, jaké je možné najít ve většině moderních operačních systémů a frameworků.

Pokud chce uživatel programově použít plátno, je nutné nejdříve získat jeho „kontext“. S kontextem je následně možné provádět různé akce a v závěru je na kontext aplikovat.
Na úpravy obsahu plátna je možno pohlížet jako na databázové transakce – transakce se zahájí, provedou se požadované akce a následně se transakce dokončí (provede se tzv. *commit*).
Souřadnice plátna začínají na hodnotách x = 0 a y = 0 v levém horním rohu, který je možné označovat jako počátek, a zvyšují se (po pixelech) horizontálně po ose X a vertikálně po ose Y.
V případě, že k používanému webu přistoupí prohlížeč, který element `canvas` anebo některou z funkcí rozhraní Canvas nepodporuje, je vhodné nabídnout alternativní zdroj (např. alternativní obrázek anebo pouhý text objasňující uživateli, o co se připravuje používáním nemoderního prohlížeče. Viz ukázka níže:

```
<canvas>
  Chcete-li používat plátno, aktualizujte váš prohlížeč.
</canvas>
```

Na element `canvas`, stejně jako u mnoha jiných elementů, je možné aplikovat kaskádové styly a přidat tak např. rámeček, vnější nebo vnitřní odsazení apod.

S výjimkou Internet Exploreru podporují nyní rozhraní Canvas všechny prohlížeče. Několik částí specifikace bylo přidáno až později a jejich podpora není příliš rozšířená (např. rozhraní Canvas Text).

### *1.3.2 Použití rozhraní Canvas*

Pro použití rozhraní Canvas je třeba učinit dva kroky:

1. Před samotným použitím elementu `canvas` je vhodné se ujistit, že ho používaný prohlížeč podporuje. V případě, že tomu tak není, je možné nabídnout alternativní text či obrázek (viz výše);
2. Přidání elementu `canvas` do stránky HTML. Viz ukázka níže:

[16] SVG aVML jsou značkovací jazyky popisující dvojrozměrnou vektorovou grafiku pomocí XML. [5]

```
<canvas height="200" width="200"></canvas>
```

Výsledné plátno se ve stránce na základě uvedeného kódu zobrazí jako prázdný obdélník 200x200 pixelů. V rámci kódu se nejprve získá objekt plátna pomocí identifikátoru elementu plátna. Následně se s použitím metody `getContext` objektu plátna, která jako parametr bere typ požadovaného plátna, určí jeho kontext. Získaný kontext je následně možné použít ke kreslení na plátno.

Dále je možné s rozhraním Canvas pracovat: tvořit diagonální čáry, pracovat s cestami (Tj. obrysy, které si uživatel přeje vykreslit. Cesty jsou mnohem komplikovanější než diagonální čáry. Mají více čar, zakřivených oblastí a také podcesty.), aplikovat transformace, nastavovat styly a pracovat s výplněmi, křivkami, vkládat obrázky a text do plátna, upravovat pozadí a velikost plátna. [12]

# 2. JAVASCRIPT

JavaScript je objektově orientovaný programovací jazyk, využívaný při tvorbě webových stránek. Na rozdíl od serverových programovacích jazyků (např. PHP) sloužících ke generování kódu samotné stránky, JavaScript běží na straně klienta, tedy v prohlížeči až po stažení do počítače uživatele.
JavaScrit se používá především pro vytváření interaktivních webových stránek. Příkladem použití mohou být nejrůznější kontroly správného vyplnění formulářů, obrázky měnící se po přejetí myši, rozbalovací menu atd. [10]

Univerzální jádro jazyka bylo vloženo do NetScape Navigatoru[17], Internet Exploreru a dalších webových prohlížečů. Pro webové programování bylo také rozšířeno přidáním objektů reprezentujících okno webového prohlížeče a jeho obsah. Tato klientská verze JavaScriptu umožňuje vložit do webových stránek proveditelný obsah – tzn., že stránka na webu již nemusí být jen statickým dokumentem HTML, ale může obsahovat dynamické programy, které komunikují s uživatelem, řídí prohlížeč a dynamicky vytvářejí obsah HTML.
Jádro jazyka JavaScript syntakticky připomíná C, C++ a Javu, a to díky příkazu `if`, smyčce `while` a operátoru `&&`.

Javascript je programovací jazyk bez typové kontroly, což znamená, že proměnné nemusí mít specifikovaný typ. Objekty v JavaScriptu se podobají spíše asociativním polím jazyka Perl, než strukturám v C[18] nebo objektům v C++ a Javě[19]. Mechanismus dědičnosti objektové orientace JavaScriptu odpovídá prvkům málo rozšířených jazyků Self[20] a NewtonScript[21] a výrazně se liší od dědičnosti v C++ a Javě. Podobně jako Perl je JavaScript interpretovaný jazyk a mnohé myšlenky přebírá z jazyka Perl. Příklady jsou jeho regulární výrazy a prvky práce s poli. [6]

## 2.1 Historie JavaScriptu

JavaScript je hovorový název pro specifikaci formálně nazývanou jako *„ECMAScript“* (viz níže). Javascript byl původně vyvinut Brendanem Eichem ve společnosti NetScape mezi lety 1995 – 1996. V té době se tento jazyk nazýval *„LiveScript“*. Lidé z marketingu však tento jazyk přejmenovali na JavaScript, což způsobilo zmatení. V té době již existoval samotný jazyk Java. Jazyk Java si však vysloužil špatnou pověst, neboť některé z webových stránek používali Javu k prezentaci nebo k přidání zbytečných vylepšení. Obecně však Java způsobovala návštěvníkům stránek potíže, protože vyžadovala načtení zásuvného modulu (pluginu), čímž zpomalovala proces surfování. Javascript byl neúprosně s Javou a s jejími problémy spojován. Později však JavaScript toto špatné propojení obou jazyků setřásl.

Brzy po svém vzniku se JavaScript zaměřil na ověřování obsahu formulářů na straně klienta a práci s obrázky na webových stránkách, aby poskytl základní, ale užitečnou interaktivitu a zpětnou vazbu návštěvníkům[22].

[17] NetScape Navigator je proprietární webový prohlížeč.
[18] Jazyk C je univerzální procedurální programovací jazyk, který pracuje je se základními datovými typy (číslo, znak). [4]
[19] C++ a Java jsou objektové orientovanými jazyky.
[20] Jazyk Self nezahrnuje pouze programovací jazyk jako takový, ale i celé nezávislé aplikační prostředí. Neexistují v něm třídy objektů, nerozlišují se členské proměnné objektů a metody. [22]
[21] NewtonScript je objektově orientovaný programovací jazyk, vyvinutý zaměstnancem Applu Walterem Smithem. [3]
[22] Když uživatel webových stránek vyplnil formulář, JavaScript se mohl použít k okamžité kontrole obsahu webového formuláře, aby se tak data nemusela odesílat webovému serveru.

Podporu jádra JavaScriptu - známou jako JScript - společně s podporou dalšího skriptovacího jazyka VBscript zahrnul Microsoft do třetí verze Internet Exploreru vydané v roce 1996. Ačkoli byly tyto jazyky stejné, nebyly stejné ve své implementaci. Proto byla vyvinuta detekující metoda – *„detekce prohlížeče“*. Díky této metodě bylo zjišťováno, který prohlížeč návštěvník webových stránek používá. Podle výsledku mu byl pak nabídnut takový JavaScript, jenž v daném prohlížeči fungoval.

## 2.2 Standardy v JavaScriptu

### *2.2.1 ECMAScript*

V polovině roku 1997 společnosti Microsoft a Netscape spolupracovaly v rámci skupiny *„European Computer Manufactures Associations“ (ECMA)* na první verzi ECMAScriptu, který prohlížeče nejen uvedených společností implementují do svých prohlížečů.
Poslední verze ECMAScriptu, přijatá jako standard pod jménem ECMA – 262, byla vydána v roce 1999. Každý důležitý prohlížeč od té doby podporuje verzi JavaScriptu přijatou ve standardu ECMA-262.

### *2.2.2 DOM*

Jiným standardem, který je používán programátory JavaScriptu, je *„objektový model dokumentu“* (DOM – Document Object Model) vyvinutý konsorciem W3C.
W3C popisuje DOM jako *„na platformě - a jazyku - nezávislé rozhraní umožňující programům a skriptům dynamicky přistupovat a aktualizovat obsah, strukturu a styly dokumentů.”*
DOM reprezentuje dokumenty HTML a XML ve formě stromové struktury a umožňuje dynamický přístup k objektům této struktury.
JavaScript silně souvisí s modelem DOM a nabízí pro práci s ním mnoho funkcí.

## 2.3 Program v JavaScriptu

Program napsaný v JavaScriptu se skládá z příkazů tvořených znaky, operátory a identifikátory umístěných za sebou v řadě. Toto uspořádání pak dává smysl interpreteru JavaScriptu, který obsahuje většina webových prohlížečů. Stejně jako v jakémkoli programovacím jazyce příkazy uspořádané za sebe způsobí, že program vykoná jednu nebo více funkcí.
JavaScript je velkou měrou komplementární jazyk. Není tedy běžné, aby byla celá aplikace napsaná pouze v JavaScriptu bez pomoci jiných jazyků, jako třeba HTML, a bez prezentace ve webovém prohlížeči.

Mnoho z toho, co JavaScript nedokáže, je výsledkem toho, že použití JavaScriptu je limitované prostředím webových prohlížečů.

1. Javascript není možné klientovy vnutit: toto je myšleno ve třech odlišných případech:
   - klient má prohlížeč, který podporuje JavaScript - sám však určuje co JavaScript může dělat;
   - klienti používají staré prohlížeče, které zcela JavaScript nepodporují;
   - klienti použití JavaScriptu v prohlížeči zakážou.
2. Javascript negarantuje bezpečnost dat: provádění kódu na straně klienta má vážné bezpečnostní důsledky. Jakmile je jednou program v počítači klienta, klient může zaměnit části programového kódu samotného, než odešle výsledek zpět serveru.
3. Javascript nemůže překročit hranice domény: vývojář v JavaScriptu si musí být vědom tzv. *politiky stejného původu* (Same-Origin Policy), jež přikazuje, že skripty spuštěné v jedné doméně nesmí mít přístup k prostředkům jiné domény, ani nemohou ovlivňovat skripty a data v jiné doméně.

4. Javascript nepracuje na straně serveru: pokud je vyvíjen kód na straně serveru, jako v případě jazyků Visual Basic.NET[23], nebo PHP[24], je celkem logicky jasné, že server implementuje určité funkce jako je komunikace s databází nebo k poskytnutí přístupu k modulům nutným pro chod webové aplikace. JavaScript ovšem nemá k proměnným na straně serveru přístup. [21]

## 2.4 Klientský JavaScript

Většina lidí, pokud se zmiňuje o JavaScriptu, má na mysli *„klientský JavaScript“*. Tento typ JavaScriptu vzniká, pokud je překladač JavaScriptu vložen do webového prohlížeče.
Klientský JavaScript kombinuje možnost zpracování skriptu interpretem JavaScriptu s DOMem, definovaným webovým prohlížečem.
Tyto dvě odlišné technologie se synergicky doplňují, takže výsledek je větší než prostý součet jejich součástí: klientský JavaScript umožňuje distribuci spustitelného obsahu přes web a je srdcem dokumentů dynamického HTML (DHTL) nové generace.

### *2.4.1 Prvky klientského JavaScriptu*

Mezi prvky klientského JavaScriptu patří:

1. Řízení obsahu a vzhledu dokumentů: objekt Document JavaScriptu dovoluje prostřednictvím metody `write()` zapistovat libovolný kód HTML do dokumentu v okamžiku, kdy je dokument nahráván do prohlížeče. Objekt Document je možné také použít ke generování dokumentů úplně od začátku. Vlastnosti objektu Document dovolují specifikovat barvy pozadí daného dokumentu, textu a v něm obsažených hypertextových odkazů;
2. Řízení prohlížeče: řízení chování prohlížeče umožňuje několik objektů JavaScriptů.
   - Objekt `Window`: tento objekt podporuje metody dialogových oken pro zobrazování jednoduchých zpráv uživateli a jednoduchý uživatelský vstup. Tento objekt rovněž definuje metodu pro vytváření a otevírání (uzavírání) úplně nových oken prohlížeče, která mohou mít danou jakoukoli velikost a jakoukoli kombinaci uživatelských ovládacích prvků.
   - Objekt `Location`: umožňuje načtení a zobrazení obsahu jakékoli adresy URL v kterémkoli okně nebo rámci prohlížeče.
   - Objekt `History`: umožňuje pohyb vpřed a zpět historií prohlížeče uživatele, čímž simuluje účinek tlačítek „vpřed“ a „zpět“ prohlížeče.
3. Interakce s formuláři HTML: dalším důležitým aspektem klientského JavaScriptu je jeho schopnost pracovat s formuláři HTML. Tuto schopnost zajišťuje objekt `Form` a objekty prvků formuláře zahrnující: `Button`, `Checkbox`, `Hidden`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text` a `Textarea`. Tyto objekty prvků umožňují číst a zapisovat hodnoty vstupních prvků ve formulářích a dokumentů.
   Mezi obvyklé využití klientského JavaScriptu ve spojení s formuláři patří kontrola dat formuláře před jeho odesláním. Pokud je JavaScript na straně klienta schopen provést veškerou kontrolu chyb uživatelského vstupu, pak není zapotřebí kontaktovat server, který by musel uživatele upozornit na triviální chyby v zadání. Klientský JavaScript může rovněž provést předběžné zpracování vstupních dat, což zredukuje množství dat, která musí být odeslána na server.

Za další rysy klientského JavaScriptu je možné považovat níže uvedené body.

- JavaScript může měnit obrázek zobrazený značkou `<img>` a vytvářet tak přechodové a animační efekty.

[23] Univerzální programovací jazyk.
[24] PHP je rekurzivní zkratka, jež značí “PHP: Hypertext Preprocessor“.

- JavaScript může provádět libovolné výpočty, má totiž datový typ s pohyblivou desetinnou čárkou, aritmetické operátory, které s ním pracují a plnou sadu standardních matematických funkcí pro pohyblivou desetinnou čárku.
- Objekt `Document` podporuje vlastnost, která specifikuje datum poslední úpravy aktuálního dokumentu. Tu lze použít k automatickému zobrazení „časového razítka“ v libovolném dokumentu.
- V klientském JavaScriptu 1.2 poskytuje objekt Screen informace o velikosti a barevné hloubce monitoru, na němž je zobrazen obsah webového prohlížeče.
- Pokud se jedná o JavaScript 1.1, pak metoda `scroll()` objektu `Window` umožňuje programům JavaScriptu pohybovat obsahem okna v rozměrech X a Y. V JavaScriptu 1.2 je tato metoda rozšířena o mnoho dalších, které umožňují přesunovaní a změnu velikosti oken prohlížeče.

Výše uvedené prvky JavaScriptu jsou omezeny na úlohy spojené s prohlížeči a dokumenty. JavaScript nemá žádné grafické schopnosti s výjimkou silných možností dynamicky generovat kód HTML zobrazovaný v prohlížeči. Další vlastností, kterou JavaScript nedisponuje je to, že klientský JavaScript neumožňuje čtení a zapisování souboru.
JavaScript také nepodporuje prácí v síti s tou výjimkou, že může prohlížeč přinutit k zobrazení určité adresy URL a že může odeslat obsah formulářů HTML přes síť serverovým skriptům a na adresy elektronické pošty.

## 2.5 Zabezpečení JavaScriptu

Dokumenty odesílané přes internet nebo elektronickou poštou nesou riziko přenesení virů nebo jiných zákeřných programů. Díky tomu, že programy klientského JavaScriptu nemohou číst lokální soubory a vykonávat síťové operace nemají tyto programy možnost vykonávat škodlivé akce. Díky složitosti prostředí webových prohlížečů se však v prvních verzích objevilo několik bezpečnostních problémů. Například v Navigatoru 2.0 bylo možné vytvořit kód JavaScriptu, který automaticky kradl e-mailové adresy všem, kdo zavítaly na stránku obsahující tento kód. Pak bylo možné automaticky odesílat zprávy elektronické pošty jménem takového návštěvníka, ale bez jeho vědomí a souhlasu.

## 2.6 Datové typy v JavaScriptu

Počítačové programy fungují tak, že pracují s *„hodnotami“*, jako je číslo nebo text. Typy hodnot, které lze reprezentovat a s nimiž lze pracovat v nějakém programovacím jazyce, se označují za *„datové typy“*.
JavaScript dovoluje pracovat se třemi základními datovými typy: čísly, řetězci textu (označovanými jen za „řetězce“) a logickými hodnotami (pravda a nepravda – `true` a `false`). Kromě těchto datových typů pracuje JavaScript také s objekty, funkcemi, poli.
JavaScript také definuje dva triviální datové typy `null` a `undefined`, přičemž každý z nich definuje jen jednu hodnotu.

Kromě funkcí a polí definuje JavaScript další specializované typy objektů, které nepředstavují nové datové typy, jenom nové *„třídy“* objektů.
Třída Date definuje objekty představující kalendářní data; Třída RegExp definuje objekty představující regulární výrazy a Třída Error definuje objekty představující chyby syntaxe za běhu, k nimž může dojít v programu JavaScriptu.

### 2.6.1 *Čísla*

Čísla tvoří nejzákladnější datový typ. JavaScript reprezentuje čísla pomocí 64bitového formátu pohyblivé desetinné čárky, jenž je definován standardem UEE754 - tzn. že dokáže reprezentovat tak velká čísla jako je $\pm 1.7976931348623157 \times 10^{308}$ a tak malá čísla, jako je $\pm 5 \times 10^{-324}$ [25].

### 2.6.2 *Řetězce*

Řetězec je sada nulového počtu nebo více znaků Unicode uzavřených v uvozovkách nebo apostrofech (“ nebo ´). Znaky uvozovek mohou být obsaženy v řetězcích ohraničených apostrofy a apostrofy se mohou nacházet v řetězcích vymezených uvozovkami – toto bývá označováno jako *„řetězcový literál“*.

### 2.6.3 *Logické hodnoty*

Logické hodnoty jsou představovány literály `true` (pravda) a `false` (nepravda).
Logická hodnota představuje „hodnotu pravdivosti“ – říká, zda je něco pravda či nikoliv. Logické hodnoty jsou zpravidla výsledkem porovnávání v programech JavaScriptu a obvykle se používají v jejich řídicích strukturách.

.

### 2.6.4 *Objekty*

*Objekty* jsou složené datové typy. Seskupují více hodnot do jediné jednotky a umožňují ukládat a číst tyto hodnoty podle názvu. Totéž lze říci také tak, že objekt je neuspořádanou kolekcí *vlastností*, z nichž každá má název a hodnotu. Pojmenovanými hodnotami objektu mohou být primitivní hodnoty, jako jsou čísla a řetězce, nebo mohou být sami dalšími objekty.
Objekty JavaScriptu mají schopnost sloužit jako asociativní pole[26] – tzn., že mohou k libovolným řetězcům přidružit libovolné hodnoty.

Objekty se vytvářejí pomocí operátoru `new`. Za tímto operátorem musí následovat funkce konstruktoru, která slouží k inicializaci daného objektu. JavaScript podporuje další vestavěné funkce konstruktoru, které inicializují nově vytvořené objekty jinými, složitějšími způsoby (např. konstruktor `Date()`, který inicializuje objekt představující datum a čas).
Další možnost vytváření a inicializování objektů představují literály objektů. Literál objektů umožňuje literálně vložit popis objektů do kódu JavaScriptu podobným způsobem, jakým jsou vkládány do kódu JavaScriptu textová data ve formě citovaných řetězců. Literál objektu se skládá z čárkami odděleného seznamu specifikací vlastností, který je uzavřen ve složených závorkách. Každá specifikace vlastnosti literálu objektů se skládá z názvu vlastnosti, za nímž následuje dvojtečka a hodnota dané vlastnosti.

Objekty JavaScritpu „dědí“ vlastnosti z *„objektu prototypu“*. Každý objekt má nějaký prototyp a všechny vlastnosti prototypového objektu se jeví jako vlastnosti každého objektu, jehož je prototypem. Tzn., že každý objekt *dědí* vlastnosti svého prototypu.
Prototyp objektů je definován funkcí konstruktoru, která byla použita k vytvoření a inicializování daného objektu. Všechny funkce JavaScriptu mají vlastnost `prototype` odkazující na určitý objekt. Tento prototypový objekt je zpočátku prázdný, ale všechny vlastnosti, které jsou v něm definovány, budou zděděny všemi objekty, vytvořenými daným konstruktorem.

---

[25] JavaScript se od programovacích jazyků, jako je C a Java liší tím, že nedělá rozdíl mezi celočíselnou hodnotou a hodnotami s pohyblivou desetinnou čárkou. Všechna čísla v JavaScriptu jsou reprezentována jako hodnoty s pohyblivou čárkou.
[26] Asociativní pole jsou silným datovým typem a jsou užitečná pro mnoho programovacích technik. Asociativní pole je také chápáno jako datová struktura umožňující dynamicky sdružovat libovolné hodnoty s libovolnými řetězci.

Prototypový objekt je přidružen ke konstruktoru, který definuje třídu objektů a inicializuje vlastnosti. Každý člen třídy tedy zdědí přesně stejnou sadu vlastností od prototypu. Z toho vychází, že objekt prototypu je ideálním místem pro uložení metod a dalších konstantních vlastností. K dědění dochází automaticky v rámci procesu vyhledávání hodnoty vlastnosti.
Vlastnosti se nekopírují z prototypových objektů do nových objektů. Jenom se jeví, jako by byli vlastnostmi takových objektů. To má dva důležité dopady:

1. Použití prototypových objektů může výrazně snížit množství paměti vyžadované každým objektem, protože objekt může mnoho svých vlastností zdědit.
2. Objekt dědí vlastnosti i v případě, kdy jsou přidány k prototypu až po vytvoření daného objektu.

Každá třída má jeden prototypový objekt s jednou sadou vlastností. Protože jednu vlastnost prototypu může zdědit mnoho objektů, musí JavaScript zavést zásadní nerovnoměrnost mezi čtením a zápisem hodnot vlastností. Pokud je čtena vlastnost `v` objektů `o` JavaScript si nejprve zjistí, zda má objekt `o` vlastnost nazvanou `v`. Nemá-li, pak se přesvědčí, zda má vlastnost nazvanou `v` prototypový objekt `o` – proto dědičnost z prototypů funguje.
Prototypové vlastnosti jsou sdíleny všemi objekty ve třídě. Jejich použití má obecný smysl pouze v případě, že chceme definovat vlastnosti, které budou stejné pro všechny objekty ve třídě.

### *2.6.5 Funkce*

*Funkce* je část vykonatelného kódu JavaScriptu, který je definovaný v programu JavaScriptu nebo předdefinován v implementaci JavaScriptu. Třebaže je funkce definována jen jednou, program JavaScriptu ji může vykonat neboli zavolat mnohokrát. Funkcím JavaScriptu lze předat argumenty neboli parametry, které specifikují hodnotu či hodnoty, s nimiž může funkce operovat, a mohou rovněž vracet hodnotu představující výsledek takového výpočtu.

Funkce se nejčastěji definují pomocí příkazu `function`. Za tímto příkazem – klíčovým slovem - následuje název funkce; volitelný čárkami oddělovaný seznam názvů parametrů uzavřených v závorkách; příkazy JavaScriptu uzavřené ve složených závorkách, které tvoří tělo funkce.
Funkce mohou být definovány tak, aby očekávali měnící se počet argumentů a že mohou a nemusí obsahovat příkaz `return` (tento příkaz způsobuje zastavení vykonání funkce a vrácení hodnoty svého výrazu volajícímu). Pokud funkce neobsahuje příkaz `return`, prostě se vykoná každý příkaz v jejím těle a volajícímu se vrátí hodnota `undifined`. Jakmile je funkce definována, lze jí volat pomocí operátoru `()`.

Důležitým prvkem JavaScriptu je to, že funkce jsou hodnoty, s nimiž lze manipulovat kódem JavaScriptu. Tím, že jsou funkce v JavaScriptu opravdovými datovými typy, dávají JavaScriptu velkou flexibilitu. Protože funkce jsou hodnotami právě tak jako čísla nebo řetězce, lze je přiřadit vlastnostem objektu, stejně jako ostatní hodnoty. Když je funkce přiřazena vlastnosti objektu, označuje se často za *„metodu“* daného objektu.

### *2.6.6 Pole*

*Pole* je, stejně jako objekt, souborem datových hodnot. Zatímco každá datová hodnota obsažena v objektu má název, každá datová hodnota v poli má číslo, neboli *„index“*. V JavaScriptu se hodnota z pole zjišťuje uzavřením indexu do hranatých závorek za názvem pole.
Pole mohou obsahovat jakýkoli typ dat JavaScriptu, včetně odkazů na jiné pole nebo na objekty či funkce. Tato pole se liší od asociativních polí. Obvyklá pole jsou indexována s celými nezápornými čísly. Asociativní pole jsou indexována řetězci.

JavaScript nepodporuje vícerozměrná pole, které tak lze aplikovat jenom ve formě polí v polích. JavaScript je také netypovým jazykem, kdy všechny prvky polí nemusí být stejného typu, jako je tomu v typových jazycích, např. v Javě.

### *2.6.7 Null*

Klíčové slovo JavaScriptu `null` je specialní hodnota, která indikuje „žádnou hodnotu“. Speciální hodnotou objektového typu je myšlena hodnota, která nepředstavuje žádný objekt, pole, číslo, řetězec ani logickou hodnotu.

### *2.6.8 Undefined*

Tato speciální hodnota `undefined` je vrácena, pokud je použita proměnná nebo vlastnost objektu, která neexistuje, nebo proměnnou, která sice byla deklarována, ale nikdy jí nebyla přiřazena hodnota. Speciální hodnota `undefined` neodpovídá hodnotě `null`. [6]

## 2.7 Bitové operátory

Bitové operátory jsou logické operátory, které pracují na úrovni jednotlivých bitů, kde jsou čísla vyjádřena ve dvojkové soustavě, tedy jedničkami a nulami. [18]

Navzdory skutečnosti, že všechna čísla jsou v JavaScriptu s pohyblivou desetinnou čárkou, bitové operátory vyžadují číselné operandy, které mají celočíselnou hodnotu. Nad těmito celočíselnými operandy operují za použití 32bitové celočíselné reprezentace namísto odpovídající reprezentace s pohyblivou čárkou.
Čtyři z těchto operátorů provádějí logickou algebru s jednotlivými bity těchto operandů a chovají se tak, jako kdyby každý bit v každém operandu byl logickou hodnotou. Další tři bitové operátory jsou určeny pro bitový posuv vlevo a vpravo.
Bitové operátory jsou používány pro manipulaci s dvojkovými čísly na nízké úrovni a nejsou při programování v JavaScriptu běžně používány.

Bitovými operátory jsou bitové AND, OR, XOR, NOT a další.

- Bitové AND (&): operátor `&` provádí logickou operaci AND na každém bitu svých celočíselných argumentů. Ve výsledku je bit nastaven pouze tehdy, je-li odpovídající bit nastaven v obou operandech.
- Bitové OR (|): operátor | provádí logickou operaci OR na každém bitu svých celočíselných argumentů. Ve výsledku je bit nastaven, pokud je odpovídající bit nastaven v jednom nebo v obou operandech.
- Bitové XOR (^): operátor ^ provádí logickou operaci exkluzivní OR na každém bitu svých celočíselných argumentů. Exkluzivní OR znamená, že je `true` buď první operand, nebo druhý operand, nikoli však oba. Ve výsledku této operace je bit nastaven, pokud je odpovídající bit nastaven v jednom (ale nikoli v obou) ze dvou operandů.
- Bitové NOT (~): operátor ~ je unární operátor, který se objevuje před svým jediným celočíselným argumentem. Operuje pomocí převrácení všech bitů v operandu.
- Posuv vlevo (<<): operátor << posouvá všechny bity ve svém prvním operandu doleva o počet míst specifikovaný ve druhém operandu, což musí být celé číslo mezi 0 a 31[27].
- Posuv vpravo se znaménkem (>>): operátor >> posouvá všechny bity ve svém prvním operandu vpravo o počet míst specifikovaných ve druhém operandu (celé číslo mezi 0 a 31).

[27] Příklad: v operaci `a << 1` se první bit `a` (bit jedničky) stane druhým bitem, druhý bit `a` (bit dvojky) se stane třetím bitem atd. Jako nový první bit se použije nula a hodnota 32. bitu se ztratí.

Bity, které jsou odsunuty vpravo, jsou ztraceny. Bity doplněné vlevo jsou stejné, jako je bit znaménka původního operandu, aby bylo zachováno znaménko výsledku. Jestliže je první operand kladný, výsledek bude mít ve vyšších bitech doplněné nuly; pokud je první operand záporný, výsledek bude mít na vyšších bitech doplněné jedničky.

- Posuv vpravo s doplněním nuly: operátor >>> je jako operátor >>, jenom bity doplněné vlevo jsou vždy nuly, bez ohledu na znaménko prvního operandu. [6]

# 3. ČÍSLICOVÁ TECHNIKA

Každé číslicové zařízení pracuje s čísly, které jsou symbolem určitého množství – kvantity. Číselná soustava používá číslice 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, není však vhodná pro počítače nebo číslicové systémy, neboť číslicové zařízení by muselo rozlišovat 10 různých např. napěťových úrovní. To může klást velké nároky na jeho kvalitu a přesnost. Jsou využívány soustavy o jiných základech. Nejčastěji používaným základem pro číslicové systémy je základ 2 s číslicemi 0 a 1.

V každodenním životě je nejužívanější soustavou soustava *desítková (dekadická)* o základu **Z = 10**[28].

V logických systémech se však nejčastěji používají níže uvedené číselné soustavy.

- *Soustava dvojková (binární, dyadická)* o základu **Z = 2**: tato soustava používá pouze dvě číslice, a to 0 a 1[29].
  V teorii reléových spojů obvykle značí 1 sepnutý kontakt a 0 kontakt rozepnutý.
  Tato soustava nalézá uplatnění mj. v elektronických spínacích obvodech.
- *Soustava osmičková (oktávová)* o základu **Z = 8**: soustava využívá osmi číslic jako násobících koeficientů, kterými jsou 0, 1, 2, 3, 4, 5, 6, 7[30].
  Dnes je použití této soustavy omezené.
- *Soustava šestnáctková (hexadecimální)* o základu **Z = 16**: soustava využívá šestnácti číslic (znaků) jako násobících koeficientů příslušné mocniny základu soustavy. Těchto šestnáct znaků je vyjádřeno alfabetickými znaky začátku abecedy A až F pro čísla 10 až 15 (viz tabulka č. 1). Šestnáctková soustava se používá v mikropočítačové technice k popisu dat na adresové a datové sběrnici. Každý symbol soustavy odpovídá čtyřem bitům dvojkové soustavy.
  Hexadecimální vyjádření dat se v mikropočítačové technice značí symbolem h nebo též H za šestnáctkovým číslem.

| Dekadická soustava Z = 10 | Dvojková soustava Z = 2 | Osmičková soustava Z = 8 | Šestnáctková soustava Z = 16 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 3 | 3 |
| 4 | 100 | 4 | 4 |
| 5 | 101 | 5 | 5 |
| 6 | 110 | 6 | 6 |
| 7 | 111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |

[28] Člověk má na rukou 10 prstů, kterými umí rozlišit 10 různých znaků.
[29] V rámci této soustavy je možné například nadefinovat, že 1 odpovídá napěťovým úrovní 9 až 10 V a 0 odpovídá napěťovým úrovním 0 až 1 V. Pro toto značení je používán výraz „pozitivní logika“.
[30] Dříve se tato soustava spolu s šestnáctkovou soustavou používala v souvislosti se staršími typy sálových počítačů s logickými obvody, procesory a mikroprocesory.

| | | | |
|---|---|---|---|
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

*Tabulku č. 1 Zobrazení čísel v nejčastěji používaných číselných soustavách*

Při práci s aplikacemi, logickými obvody či mikroprocesory dochází k přepočtu čísel mezi různými číselnými soustavami. Samotné číslo je neustále stejné (má stejnou hodnotu). Jeho vyjádření v různých soustavách je však různé. [1]

### 3.1 Logické funkce

Logika je nauka o základech myšlení v procesu vytváření úsudků a důkazů.

Logický obvod je takový obvod, u něhož může každá veličina na vstupu i výstupu ustáleném stavu nabývat s určenou přesností jen jedné ze dvou možných hodnot a který obsahuje takové prvky, jejichž vstupní a výstupní veličiny mohou nabývat jen jedné ze dvou možných hodnot. Logický obvod je realizován skupinou logických členů vzájemně spojených tak, aby realizovali požadované logické funkce. Podle druhu realizované logické funkce rozeznáváme níže uvedené logické obvody.

*Kombinační logické obvody*: systémy, jejichž odezva je v určitém časovém okamžiku podmíněna výhradně hodnotami, které panují na vstupech tohoto systému.

*Sekvenční logické obvody*: jedná se o systémy, jejichž odezva je v určitém časovém okamžiku dána nejen hodnotami proměnných na vstupech tohoto systému, ale i posloupností (sekvencí) předcházejících vstupních hodnot. Sekvenční obvod je proto opatřen pamětí, která svým stavem definuje vnitřní stav tohoto systému v závislosti na posloupnosti signálu, které přicházeli na vstup.

Při navrhování logického obvodu (kombinačního či sekvenčního) jsou rozlišovány dva základní kroky:

1. vytvoření přesného popisu chování systému a případné zjednodušení tohoto popisu;
2. vytvoření obvodové realizace logického systému, který splňuje chování podle bodu 1.

Vstupní proměnné nazývané $x_1$ až $x_n$ jsou nositeli jedné ze dvou možných logických hodnot 0 nebo 1[31]. Logický systém pak pomocí hodnot výstupních proměnných $y_1$ až $y_m$ působí zpětně na okolí. Úkolem kombinačního logického obvodu je realizovat logické funkce $f_i$ tak, že každé kombinaci vstupních proměnných přiřadí určitou hodnotu výstupní proměnné. Na vstup obvodu může přijít v znějšku celkem $2^n$ různých kombinací hodnot vstupních proměnných. Pro n vstupních proměnných můžeme získat $2^{2^n}$ různých logických funkcí $f_i$.

[31] Log. 0 je možné si představit jako stav „vypnuto“. Jako synonymum je používán symbol L, což pochází z anglického Low (nízký); log. 0 má nižší napěťovou úroveň než log. 1.
Log. 1 je možné si představit jako stav „zapnuto“. Synonymem je symbol H (High – vysoký); log. 1 má vyšší napěťovou úroveň. [14]

### *3.1.1 Logické funkce jedné proměnné*

Logickou funkci jedné nezávislé proměnné lze vyjádřit čtyřmi způsoby, které jsou znázorněny v tabulce č. 2. Vstupní proměnná je zde označena a, logické funkce jsou $f_0$, $f_1$, $f_2$, $f_3$.

| a | $f_0$ | $f_1$ | $f_2$ | $f_3$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |

*Tabulka č. 2 Logické funkce jedné nezávisle proměnné*

$f_0$– triviální identita, nulová funkce
$f_1$– identická funkce, její hodnota se shoduje s hodnotou proměnné a
$f_2$– funkce negace, hodnota funkce se shoduje s hodnotou ā
$f_3$– triviální identita, jednotková funkce.

Praktický význam mají pouze funkce identická a funkce negace. Negace logické proměnné a se označuje nad příslušnou veličinou, tj. $f_2 = \bar{a}$.

### *3.1.2 Logické funkce dvou proměnných*

Pro dvě vstupní proměnné je možné najít šestnáct různých logických funkcí.

| A | 0 | 0 | 1 | 1 | Význam jednotlivých funkcí |
|---|---|---|---|---|---|
| B | 0 | 1 | 0 | 1 | |
| $f_0$ | 0 | 0 | 0 | 0 | $f_0 = 0$, triviální identita |
| $f_1$ | 0 | 0 | 0 | 1 | $f_1 = a.b$, logický součin, AND |
| $f_2$ | 0 | 0 | 1 | 0 | $f_2 = a.\bar{b}$, přímá inhibice |
| $f_3$ | 0 | 0 | 1 | 1 | $f_3 = a$, funkce identická |
| $f_4$ | 0 | 1 | 0 | 0 | $f_4 = \bar{a}.b$, zpětná inhibice |
| $f_5$ | 0 | 1 | 0 | 1 | $f_5 = b$, funkce identická |
| $f_6$ | 0 | 1 | 1 | 0 | $f_6 = a \oplus b$, nonekvivalence, XOR, součet module 2 |
| $f_7$ | 0 | 1 | 1 | 1 | $f_7 = a + b$, logický součet, OR |
| $f_8$ | 1 | 0 | 0 | 0 | $f_8 = \overline{a + b}$, negovaný logický součet, NOR, Piercova funkce |
| $f_9$ | 1 | 0 | 0 | 1 | $f_9 = \overline{a \oplus b}$, ekvivalence, XNOR |
| $f_{10}$ | 1 | 0 | 1 | 0 | $f_{10} = \bar{b}$, negace |
| $f_{11}$ | 1 | 0 | 1 | 1 | $f_{11} = a + \bar{b}$, zpětná implikace |
| $f_{12}$ | 1 | 1 | 0 | 0 | $f_{12} = \bar{a}$, negace |
| $f_{13}$ | 1 | 1 | 0 | 1 | $f_{13} = \bar{a} + b$, přímá implikace |
| $f_{14}$ | 1 | 1 | 1 | 0 | $f_{14} = \overline{a.b}$, negovaný logický součin, NAND, Shefferova funkce |
| $f_{15}$ | 1 | 1 | 1 | 1 | $f_{15} = 1$, triviální identita |

*Tabulka č. 3 Logické funkce dvou vstupních proměnných*

V tabulce č. 3 je několik významných logických funkcí, které se používají při návrhu logických obvodů.
Mezi tři základní logické funkce, které jsou definovány v rámci Booleovy algebry[32] patří *logický součin, logický součet a negace.*

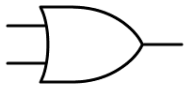
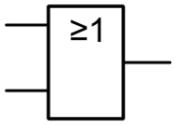
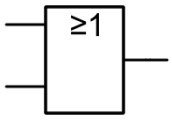
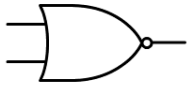
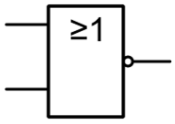
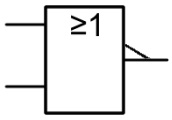
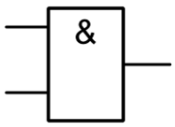
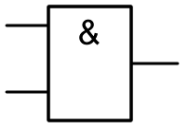
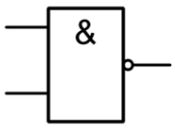
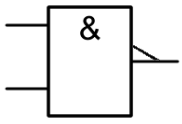
1. *Logický součin* ($f_1$) dvou proměnných a, b, označovaný jako AND. Je-li výstup logického obvodu označen y, pak je logický součin zapsán jako $y = a.b$.
2. *Logický součet* ($f_7$), označovaný jako OR. Při stejném značení ho můžeme zapsat vztahem $y = a + b$.
3. *Negace* ($f_{10}$ a $f_{12}$), označovaná jako NOT, je dána vztahy $y = \bar{b}$ nebo $y = \bar{a}$.

Výše uvedené vztahy se nazývají úplný *soubor logických funkcí*, protože jejich kombinací je možné navrhnout jakýkoli číslicový obvod.[33]

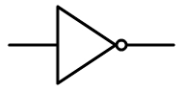
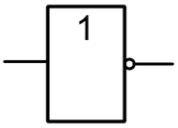
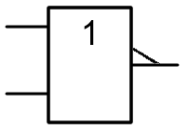
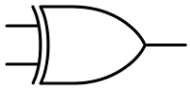
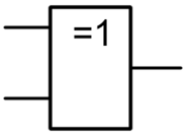
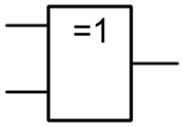
Dalšími významnými funkcemi uvedených v tabulce jsou *NAND, NOR*[34]*, XOR, XNOR*[35].

1. *NAND* negovaný logický součin ($f_{14}$): $y = \overline{a.b}$;
2. *NOR* negovaný logický součet ($f_8$): $y = \overline{a + b}$;
3. *Nonekvivalence* ($f_6$), XOR: $y = a \oplus b = \bar{a}.b + a.\bar{b}$;
4. *ekvivalence* ($f_9$), XNOR: $y = \overline{a \oplus b} = a.b + a.b$ .

Logické funkce jsou realizovány pomocí základních logických členů, jejichž spojováním je získán logický obvod. Při kreslení schématu logických obvodů jsou používány schematické značky těchto členů. Níže je uvedena tabulka nejčastěji používaných schematických značek v ČR i v zahraničí.

| $Y = A + B$ | Logický součet OR | |
|---|---|---|
|  |  |  |
| $Y = \overline{A + B}$ | Negovaný logický součet NOR | |
|  |  |  |
| $Y = A \cdot B$ | Logický součin AND | |
|  |  |  |
| $Y = \overline{A \cdot B}$ | Negovaný logický součin NAND | |
|  |  |  |

[32] Booleova algebra je matematická disciplína, která je přímo aplikovatelná při návrhu číslicových obvodů. Zahrnuje pravidla a teorémy pro operace s logickými proměnnými a funkcemi.
[33] Příklad: dvě funkce AND a NOT, nebo OR a NOT stačí pro vyjádření jakékoli logické funkce.
[34] Pomocí funkcí NAND a NOR je možno vyjádřit a realizovat každou logickou funkci.
[35] Obě funkce XOR a XNOR jsou významné pro realizaci aritmetických obvodů a číslicových komparátorů.

| $Y = \overline{A}$ | Negace NOT | |
|---|---|---|
|  |  |  |
| $Y = \overline{A} \cdot B + A \cdot \overline{B} = A \oplus B$ | Nonekvivalence XOR | |
|  |  |  |
| US | ČSN | IEC |
| Americké značení schematických značek. Je možné se s ním setkat ve většině zahraničních katalogů číslicových prvků. | Československé značení schematických značek. V minulosti často používané. | Schematické značení dle doporučení mezinárodní elektrotechnické komise *IEC (International Electrotechnical Commission)*. Je možné jej v katalozích najít jako alternativní značení. |

*Tabulka č. 4 Používané schematické značky základních logických členů - upraveno*

Značka logické funkce se skládá z bloku, v jehož horní části je umístěn symbol logické funkce. Značky ve schématech se obvykle řadí tak, aby vstupy byly umístěny vlevo a výstupy vpravo. Vstupy a výstupy schematických značek nesou také značení, zda je vstup nebo výstup aktivní v jedničce nebo je aktivní v nule.[36]

### 3.3 Kombinační logické obvody

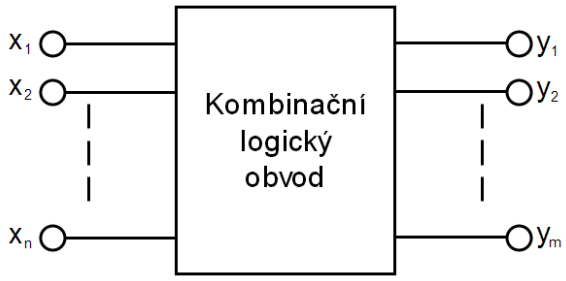
Logické obvody jsou takové obvody, u nichž mohou vstupní i výstupní proměnné v ustáleném stavu nabývat jedné ze dvou možných hodnot, logické 0 nebo logické 1.
Kombinační logický obvod je realizován spojením základních logických členů tak, aby splňoval požadovanou logickou funkci. Okamžitá hodnota výstupních proměnných kombinačního logického obvodu je daná pouze okamžitou kombinací vstupních proměnných (tzn., že jsou těmito obvody realizovány výhradně takové situace, které nejsou závislé na předchozích kombinacích vstupů.).

Tyto systém je možné popsat níže uvedenou rovnicí:

$$Y_i = f(X_j), \text{ kde } i = 1,2,...m, j = 1,2,...n.$$

Proměnné $Y_i = y_1, y_2, ... y_m$ značí výstupní proměnné kombinačního obvodu
a proměnné $X_j = x_1, x_2, ... x_n$ jsou vstupní proměnné.

[36] Tento popis logické funkce platí pro ČSN.




*Obrázek č. 1 Blokové znázornění kombinačního logického obvodu*

Kombinační obvody jsou považovány za funkční celky, které se realizují buď spojením základních logických členů, nebo pomocí integrovaných obvodů, se střední hodnotou integrace.
Mezi typické představitelé kombinačních logických obvodů patří: dekodéry; multiplexery a demultiplexery (viz níže); komparátory; obvody pro aritmetické operace (sčítačky, generátory přenosu apod.).

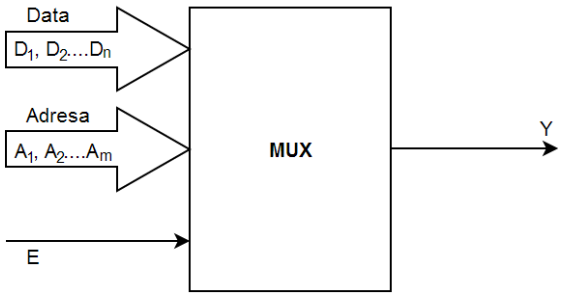
### *3.3.1 Multiplexery*

Multiplexer[37] je kombinační logický obvod, který umožňuje převádět číslicové informace z některého vybraného vstupního kanálu na výstup. Česky je možné tento obvod označit jako přepínač číslicových signálů. Obsahuje *n* vstupů $D_1$ až $D_n$ a jeden výstup Y. Volba jednoho ze vstupů je prováděna pomocí řídících a adresových signálů $A_1$ až $A_m$. Pro *n* datových vstupů je použito *m* adresových vstupů.

Mezi počtem datových *n* a adresových *m* vstupů platí závislost

$$n = 2^m \text{ nebo } 2^{m-1} < n < 2^m.$$

Pokud tedy má multiplexer přepínat např. jeden ze čtyř vstupů na jeden výstup musí být adresa vstupů určována dvěma adresovými logickými proměnnými $A_1$, $A_2$.




*Obrázek č. 2 Schematický znak multiplexeru*

Kromě vstupů dat $D_1$ až $D_n$ a adresových vstupů $A_1$ až $A_m$ je na výše zobrazeném obrázku vyznačen ještě blokovací vstup E (Enable), který je používán pro ovládání činnosti celého obvodu. Jeho aktivní úroveň uvádí výstup do neaktivní logické úrovně (např. do logické 1) bez ohledu na hodnoty signálu ostatních vstupů. Pokud by dále mělo být přepínáno více vstupů, než je k dispozici

[37] Multipexer je používán tam, kde má být z množství různých logických signálů vybrán jediný, např. na vstupech aritmeticko-logických jednotek, obvodech pro převod paralelní informace na sériovou, při multiplexním buzení sedmisegmentových displejů apod.

na použitém integrovaném obvodu, je třeba spojit více multiplexerů paralelně (sloučit jejich výstupy například součtovým hradlem) a vstupem E volit, z které skupiny vstupů má být signál vybrán.
Pro výstup multiplexeru je možné napsat logickou funkci

$$Y = \left(Adr_1 \cdot D_1 + Adr_2 \cdot D_2 + \cdots + Adr_n \cdot D_{1_n}\right) \cdot E,$$

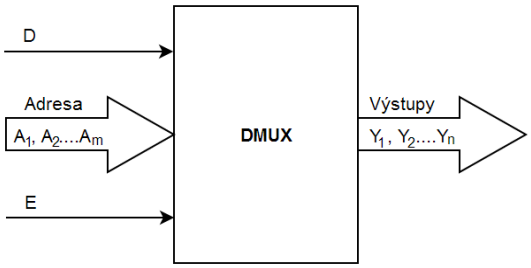
kde $Adr_j$ označuje příslušnou kombinaci adresových vstupů $A_1$ až $A_m$. Např. v případě osmivstupového multiplexeru je potřeba tří adresových vodičů $A_1$, $A_2$ a $A_3$. Kombinací adresových vstupů je pak chápáno při výběru vstupu $D_1$ kombinaci $\overline{A}_3\ \overline{A}_2\ \overline{A}_1$, pro vstup $D_2$ kombinaci $\overline{A}_3\ \overline{A}_2\ A_1$, atd.

### *3.3.2 Demultiplexery*

Demultiplexer plní přesně opačnou funkci než multiplexer. „V závislosti na logickém stavu adresových vodičů přenáší z jediného vstupního vodiče data na jeden z výstupních vodičů, zatímco na ostatních výstupních vodičích setrvává neaktivní stav.“
Demultiplexer je tedy kombinační logický obvod, který je možné popsat n Booleovými výrazy

$$Y_i = Adr_i \cdot D \cdot E,$$

kde $Adr_i$ je kombinace adresových vstupů $A_1$, $A_2$ až $A_m$. Pomocí těchto vstupů je vybíráno 1 z n výstupů. Tento vztah je vyjádřen na níže zobrazeném obrázku.




*Obrázek č. 3 Principální schéma demultiplexeru*

Integrované demultiplexery jsou vyráběny téměř výhradně jako obvod s aktivní úrovní logické 0 na vybraném výstupu (tzn., že na všech výstupech je logická 1 a pouze na výstupu, jehož adresa je určena stavem zavedeným na adresové vodiče, se přenese logický stav datového vstupu D). [1]

### *3.3.3 Dekodéry*

Dekodéry (kodéry)[38] jsou kombinační logické obvody, které v závislosti na operaci vstupních proměnných generují určitý kód, kombinaci binárních stavů.
Důležité a často používané jsou *binární dekodéry*, které převádějí binární stavy uspořádané v jednom kódu na binární stavy v jiném kódu[39].
Základním stavebním prvkem dekodéru je součinové hradlo AND, které má na výstupu logickou 1 v případě, že všechny jeho vstupy jsou logická 1, nebo hradlo NAND, které má na výstupu logickou 0 pro vstupy ve stavu logická 1.

[38] Některé publikace mohou používat oba pojmy s tím, že kodéry určité kódy generují a dekodéry určité kódy identifikují. Podstata obou procesů je však shodná a obě zařízení provádějí převod z jednoho kódu na jiný.
[39] Nejjednodušším takovým dekodérem je dekodér dvoubitového slova na kód 1 ze 4.

### 3.4 Sekvenční logické obvody

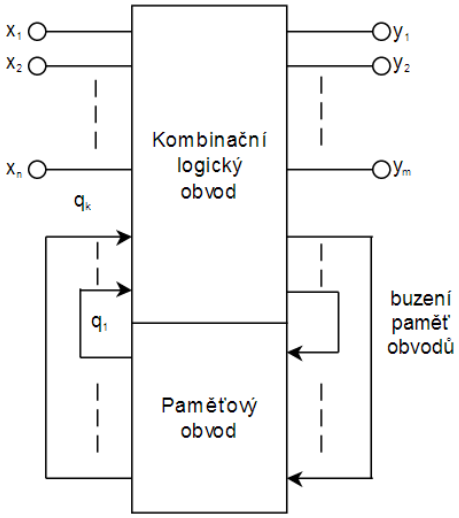
Sekvenční logické obvody se liší od kombinačních obvodů tím, že okamžitá hodnota výstupu závisí nejen na okamžitých hodnotách vstupů, ale i na jejich předcházející posloupnosti.
Tento typ logických obvodů obsahuje paměťové prvky, ve kterých uchovávají předchozí stav obvodu a reagují na něj.
Funkci sekvenčních obvodů je možné popsat dvěma rovnicemi, rovnicí pro výstupy

$$Y_i = f(\{X_j\}, \{Q_p^t\})$$

a rovnicí pro následující vnitřní stav

$$Q_p^{t+1} = f(\{X_j\}, \{Q_p^t\})^{40}$$

$Y_i$ = množina výstupů sekvenčního obvodu $Y_i = (y_{1,}y_{2,\ldots,}y_m)$;
$X_j$ = množinu vstupů $X_j = (x_{1,}\, x_{2,}\ldots, x_n)$;
$Q_p$ = množina vnitřních stavů $Q_p = (q_{1,}\, q_{2,}\ldots, q_{k)}$.
Proměnné $q_{1,}\, q_{2,}\ldots, q_k$ jsou nazývány vnitřními proměnnými a jsou tvořeny výstupy jednotlivých klopných obvodů. Množina všech výstupů klopných obvodů v určitém čase $t$ tvoří vnitřní stav. Indexem t je rozlišován současný vnitřní stav $Q_p^t$ a t + 1 následující vnitřní stav sekvenčního obvodu $Q_p^{t+1}$.




*Obrázek č. 4 Blokové schéma sekvenčního logického obvodu*

Z výše uvedeného obrázku vyplývá, že sekvenční logický obvod má kombinační část, která generuje hodnoty výstupů $y_{1,}y_{2,\ldots,}y_m$ a dále budící signály klopných obvodu. Taktéž má obvod sekvenční část, tvořenou klopnými obvody. Tyto klopné obvody, na základě budících signálů, generují vnitřní proměnné sekvenčního obvodu.

[40] Logický obvod popsaný výše uvedenými rovnicemi je nazýván *Mealyho sekvenční obvod (Mealyho automat)*. Jestliže je výstup sekvenčního obvodu určen jednoznačně jeho vnitřním stavem, pak je takový obvod nazýván *Moorův sekvenční obvod (Moorův automat)*. Moorův sekvenční obvod je popsán rovnicí pro výstupy ve tvaru:

$$Y_i = f_i(\{Q_p^t\})$$

Vstupními signály kombinační části jsou vstupní signály $x_1, x_2, ..., x_n$ a vnitřní proměnné $q_1, q_2, ..., q_k$ z výstupů paměťové části.
Sekvenční část je tvořena klopnými obvody, které jsou řízeny periodickým číslicovým signálem s periodou T, který se v praxi nazývá *hodinový signál* nebo *taktovací signál* z anglického *clock*. Hodinový signál taktuje klopné obvody sekvenční části a definuje vnitřní stavy sekvenčního obvodu v časech $t$ , $t + 1$, $t + 2$ atd.

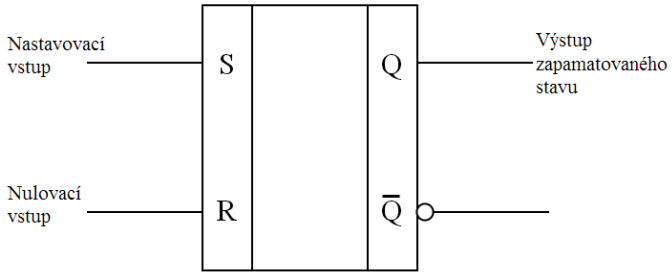
### *3.4.1 Klopné obvody*

Klopné obvody, jako paměťové členy, jsou nejjednodušší sekvenční logické obvody. Jedná se o obvody Moorova typu. Hodnota jejich výstupu je totiž totožná s vnitřním stavem.
Klopné obvody se užívají jednak jako paměťové členy v sekvenčních obvodech a jednak jako samostatné funkční bloky (např. v posuvných registrech, v čítačích, statických pamětech apod.). Stěžejním úkolem klopných obvodů je zaznamenat přítomnost přechodné informace a uchovat tento stav i tehdy, když informace zmizí. Klopné obvody jsou tedy elementární paměti. Jejich výstupní stavy se mění skokem mezi dvěma hodnotami logických úrovní.

Klopné obvody je možné členit dle počtu stabilních stavů na klopné obvody *bistabilní*, *monostabilní* a *astabilní*.

1. *Bistabilní klopné obvody:* tento typ obvodů má dva možné stabilní stavy. Jeden přechází skokem v druhý v případě, že je na vstup přiveden budící vstupní signál, příp. taktovací signál. Používají se jako registry, děliče frekvence a pro statické paměťové obvody.
   Mezi některé typy bistabilních klopných obvodů patří *jednoduchá klopný obvod RS; klopný obvod RST; klopný obvod typu D; dvojčinný klopný obvod RST; dvojčinný klopný obvod JK* a *čelem řízený klopný obvod typu D.*

   A. *Jednoduchý klopný obvod RS:* je nejjednodušším klopným obvodem. Má dva vstupy R a S, výstup Q a jeho negaci $\overline{Q}$. Vstup S (Set – nastavení) slouží jako vstup signálu, kterým se nastavuje klopný obvod do logické 1. Je-li přivedena na vstup S logická 1, přejde výstup do stavu Q = 1. Vstup R (Reset – nulování) slouží jako vstup signálu pro nulování klopného obvodu. Je-li přivedena na vstup R logická 1, přejde do stavu Q = 0 a $\overline{Q} = 1$.



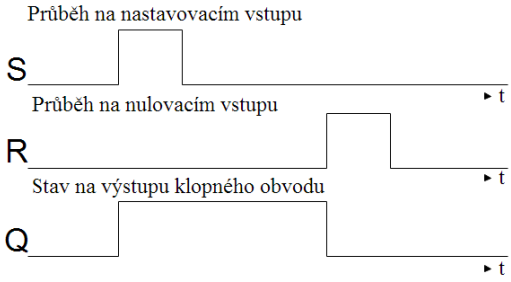

*Obrázek č. 5 Vstupy a výstupy klopného obvodu*

Klopný obvod RS má dva stabilní stavy Q = 1 a Q = 0. Současně musí platit, že výstup $\overline{Q}$ je negací výstupu Q, tedy $\overline{Q} = 0$ a $\overline{Q} = 1$.
Činnost klopného obvodu RS je popsána níže uvedenou pravdivostní tabulkou.

| S | R | $Q^{t+1}$ | $\overline{Q^{t+1}}$ |
|---|---|---|---|
| 0 | 0 | $Q^t$ | $\overline{Q^t}$ |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | (1) | (1) |

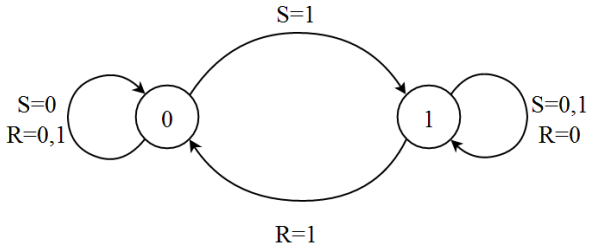
*Tabulka č. 5 Pravdivostní tabulka RS klopného obvodu*

Na obrázku č. 6 je znázorněn časový diagram signálů RS klopného obvodu. Je z něho patrná funkce RS klopného obvodu, nastavení a nulování výstupu Q klopného obvodu v závislosti na průběhu vstupů R a S.



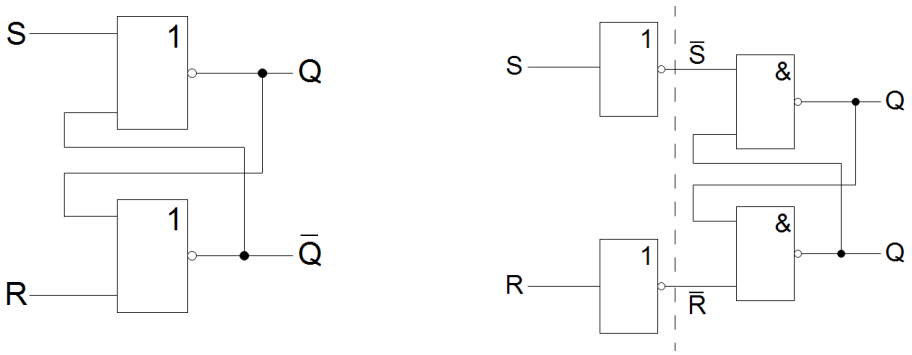

*Obrázek č. 6 Časový diagram klopného obvodu RS*

Diagram přechodů RS klopného obvodu se signály odpovídajícími pravdivostní tabulce č. 5 je zobrazen na obrázku č. 7.
Diagram graficky znázorňuje způsob přechodu mezi oběma stavy klopného obvodu a uvádí, kterými hodnotami budících vstupů jsou tyto přechody vyvolány.




*Obrázek č. 7 Diagram přechodu klopného obvodu RS*

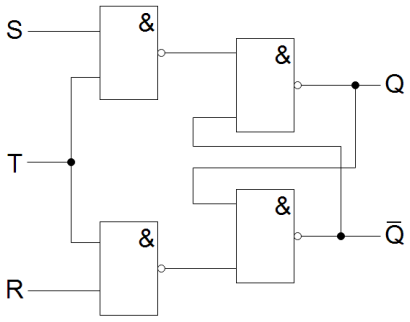
Pravdivostní tabulka, časové diagramy i diagram přechodů odpovídají zapojení RS klopného obvodu z hradel NOR (viz obrázek č. 8a). Stejnou funkci má i zapojení na obrázku č. 8b, kde je sice klopný obvod sestaven z hradel NAND, ale jsou jim předřazeny invertory. [1]




*Obrázek č. 8 Klopný obvod RS sestavený z hradel a) NOR, b) NAND*

Klopný obvod RS je velmi důležitý, protože se jedná o nejjednodušší zapojení vykazující paměťový efekt. Je také základem pro vytvoření klopných obvodů D a jiných sekvenčních obvodů. [14]

B. *Klopný obvod RST:* pokud je požadován záznam informace ze vstupů R a S do klopného obvodu v určitém časovém intervalu, je možné doplnit vlastní klopný obvod vstupními hradly, která se otvírají taktovacím impulsem T (viz příklad tohoto obvodu sestaveného z hradel NAND na obrázku č. 9).



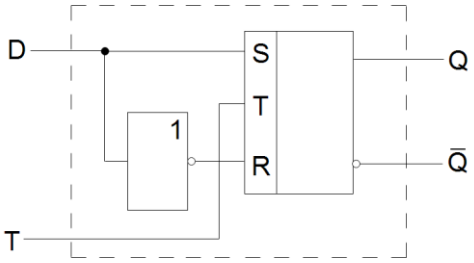

*Obrázek č. 9 Schéma taktovaného RS klopného obvodu*

Pravdivostní tabulka č. 6 udává stav, do kterého přejde výstup tehdy, když taktovací impuls má hodnotu logická 1, a ve kterém výstupy setrvají po skončení impulsu. Výstupy reagují na změny vstupních veličin po celou dobu trvání taktovacího impulsu. Stav 1, 1 na vstupech R a S je opět nedovolený. Ani tento obvod není tedy proti vzniku zakázaného stavu ošetřen.“ Z pravdivostní tabulky č. 6 vyplývá, že taktovací signál T ve stavu logická 1 vytváří vodivou cestu pro agresivní 0 obvodů NAND.

| T | S | R | $Q^{t+1}$ | $\overline{Q^{t+1}}$ |
|---|---|---|---|---|
| X | 0 | 0 | $Q^t$ | $\overline{Q^t}$ |
| 0 | X | X | $Q^t$ | $\overline{Q^t}$ |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | (1) | (1) |

*Tabulka č. 6 Pravdivostní tabulka RST klopného obvodu*

C. *Klopný obvod typu D*[41]: jsou-li spojeny vstupy S a R klopného obvodu RST z obrázku č. 9 přes invertor, je tak zabráněno vzniku zakázaného stavu na výstupu klopného obvodu.
Spojený vstup je označován jako „D“. Samotný obvod je nazýván D klopný obvod. Ten je zobrazen na obrázku č. 10.
Stav na vstupu D je přenášen na výstup po celou dobu trvání taktovacího impulsu na úrovni logická 1.



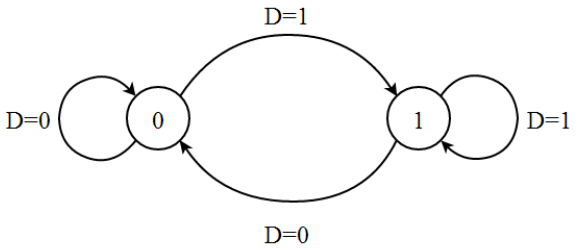

*Obrázek č. 10 Blokové schéma klopného obvodu typu D*

Pravdivostní tabulka klopného obvodu typu D je znázorněna v tabulce č. 7.
.

| T | D | $Q^{t+1}$ |
|---|---|---|
| 0 | x | $Q^t$ |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Tabulka č. 7 Pravdivostní tabulka RST klopného obvodu*

Diagram přechodů klopného obvodu D je uveden na obrázku č. 11. Obvody tohoto typu, taktované úrovní taktovacího pulsu, se nazývají klopné obvody typu „latch“ (závora) a používají se jako paměťové registry (např. dočasné a vyrovnávací paměti mezi procesorem a vstupně/výstupními jednotkami).




*Obrázek č. 11 Diagram přechodů klopného obvodu D*

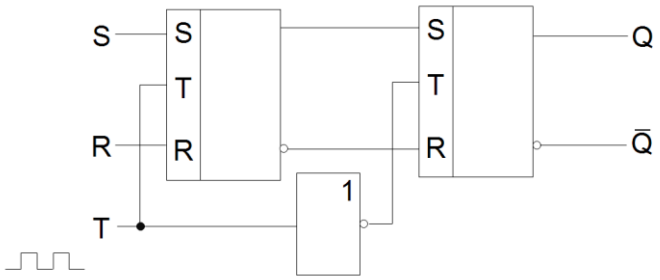
V tabulce č. 8 je znázorněna pravdivostní tabulka JK klopného obvodu a na obrázku č. 12 poté diagram přechodů.
JK klopné obvody jsou spolu s D klopnými obvody významnými obvody pro konstrukci sekvenčních logických obvodu jako stavební blok integrovaných obvodů.

[41] Tyto obvody je možné najít realizované jako obvody s označením 7475 a 7477 jako čtveřici bistabilních obvodu nebo jako osmi a vícebitové budiče sběrnic s paměťmi, které mají značení 74373, 74374, 74841 aj.

| J | K | $Q^{t+1}$ | $\overline{Q^{t+1}}$ |
|---|---|---|---|
| 0 | 0 | $Q^t$ | $\overline{Q^{t+1}}$ |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | $\overline{Q^{t+1}}$ | $Q^t$ |

*Tabulka č. 8 Diagram přechodů JK klopného obvodu*

D. *Dvojčinný klopný obvod RST (obvod s mezipamětí):* jedná se o dva stupně klopných obvodů s odděleným taktováním, které mají oddělené vstupy od výstupů.
Dvojčinný klopný obvod RST se skládá ze dvou klopných obvodů RST a z invertoru, který odděluje taktovací signál obou obvodů (viz obrázek č. 12).
Stav, který je na vstupech RS, se zapisuje do prvního klopného obvodu v době, kdy je taktovací impuls T na úrovni logická 1. Změny na vstupech mohou ovlivňovat zapsaný stav po celou dobu trvání vrcholu taktovacího impulsu.

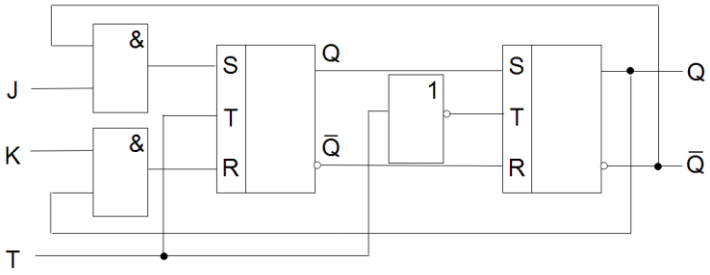



*Obrázek č. 12 Schéma dvojčinného RST klopného obvodu*

Druhý klopný obvod je negovaným taktovacím impulsem blokován a na jeho výstup se změny vstupů nepřenesou. V okamžiku, kdy přichází týl taktovacího impulsu, je zafixována poslední informace ze vstupů R, S v prvním klopném obvodu a jeho stav je přepsán do druhého klopného obvodu logickou 1 na jeho taktovacím vstupu.
Tento typ klopného obvodu není zajištěn proti možnosti vzniku nedefinovaného překlopení, pokud týl hodinového impulsu přichází ve stavu, kdy R a S jsou ve stavu logická 1. Stav výstupního klopného obvodu zachycuje stav vstupů v jediném okamžiku, a to ve chvíli, kdy probíhá týl taktovacího impulsu. V jiných fázích taktovacího průběhu nelze výstup ovlivnit. Takové klopné obvody jsou řízeny *hranou* taktovacího signálu.

E. *Dvojčinný klopný obvod JK*[42]*:* princip tohoto obvodu umožňuje definovat logickou funkci i pro případ, že oba vstupy jsou ve stavu logická 1.
Tento typ obvodu bývá spouštěn týlem taktovacího impulsu. Zapojení obvodu zobrazuje obrázek č. 13.

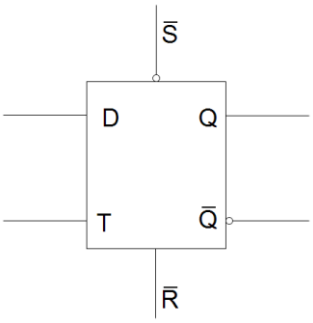
[42] Klopné obvody jsou vyráběny jako integrované obvody pod označením 7470, 7472 a 7473, které se liší především počtem J a K vstupů, příp. dodatečnými nastavovacími a nulovacími vstupy.




*Obrázek č. 13 Schéma dvojčinného JK klopného obvodu*

Obvod má opět oddělené taktovací vstupy obou stupňů klopných obvodů pomocí invertoru. Zapojením zpětné vazby z výstupu Q na vstup K a z výstupu $\overline{Q}$ na vstup J se docílí toho, že vstupní kombinaci J = 1 a K = 1 na vstupech obvodu odpovídá překlopení výstupního obvodu do opačného stavu, než byl před příchodem týlu taktovacího impulsu.
Klopný obvod JK je zabezpečen proti vzniku zakázaného stavu na výstupu. Zabezpečení v tomto obvodu pro J = K = 1 je možné snadno odvodit, pokud je známo, že na vstupu jsou součinové členy. Alespoň jeden z nich má totiž na vstupu logickou 0 zavedenou z výstupu, a proto na vstupech R, S prvního RST klopného obvodu nikdy nenastane současně kombinace obou jedniček.

F. *Čelem řízený klopný obvod typu D:* jedná se o klopný obvod, který přenese na výstup stav vstupu D pouze v okamžiku přechodu impulsu T z úrovně logická 0 na úroveň logická 1. Stav výstupu při konstantní hodnotě T rovné logické 0 nebo logické 1 není změnami na vstupu D ovlivňován.
Příkladem klopného obvodu D řízeného náběžnou hranou může být integrovaný obvod 7474, který obsahuje dvojici D klopných obvodů. Každý z dvojice D klopných obvodů obsahuje datový vstup D a taktovací vstup T. Dále obsahují dvojici *asynchronních vstupů* $\overline{S}$ a $\overline{R}$ viz obrázky č. 14a, 14b.




| vstupy | | | | výstupy | |
|---|---|---|---|---|---|
| $\overline{S}$ | $\overline{R}$ | T | D | Q | $\overline{Q}$ |
| 0 | 1 | X | X | 1 | 0 |
| 1 | 0 | X | X | 0 | 1 |
| 0 | 0 | X | X | (1) | (1) |
| 1 | 1 | ↑ | 1 | 1 | 0 |
| 1 | 1 | ↑ | 0 | 0 | 1 |

*Obrázek č. 14 Schematická značka klopného obvodu typu D 7474*
*Tabulka č. 9 Funkční tabulka klopného obvodu typu D7474*

2. *Monostabilní klopné obvody:* tyto obvody mají jeden stabilní stav a jejich výstupní signál je jeden impuls, který vznikne teprve přivedením signálů na vstup klopného obvodu. Používají se pro generování jednotlivých impulsů, jejichž délka je dána hodnotou vnějších RC prvků.

3. *Astabilní klopné obvody:* nemají žádný stabilní stav a jejich výstupní signály jsou periodické impulsy. Jejich amplitudy a frekvence závisejí na parametrech obvodu. Používají se jako zdroje taktovacích signálů v různých aplikacích (např. v sekvenčních obvodech, mikroprocesorech aj.). [1]

# 4. SIMULÁTOR LOGICKÝCH OBVODŮ – WEBOVÁ APLIKACE

Vytvořený simulátor logických obvodů je koncipován tak, aby dokázal simulovat logický obvod složený z logických prvků ve webovém prohlížeči.
Aplikace je rozdělena do dvou uživatelských modů - editačního a simulačního. V editačním módu je možné si logický obvod sestrojit pomocí dostupných logických prvků a v simulačním si pak provést simulaci sestrojeného obvodu. Uživateli se tak zobrazí chování obvodu včetně případných chyb.

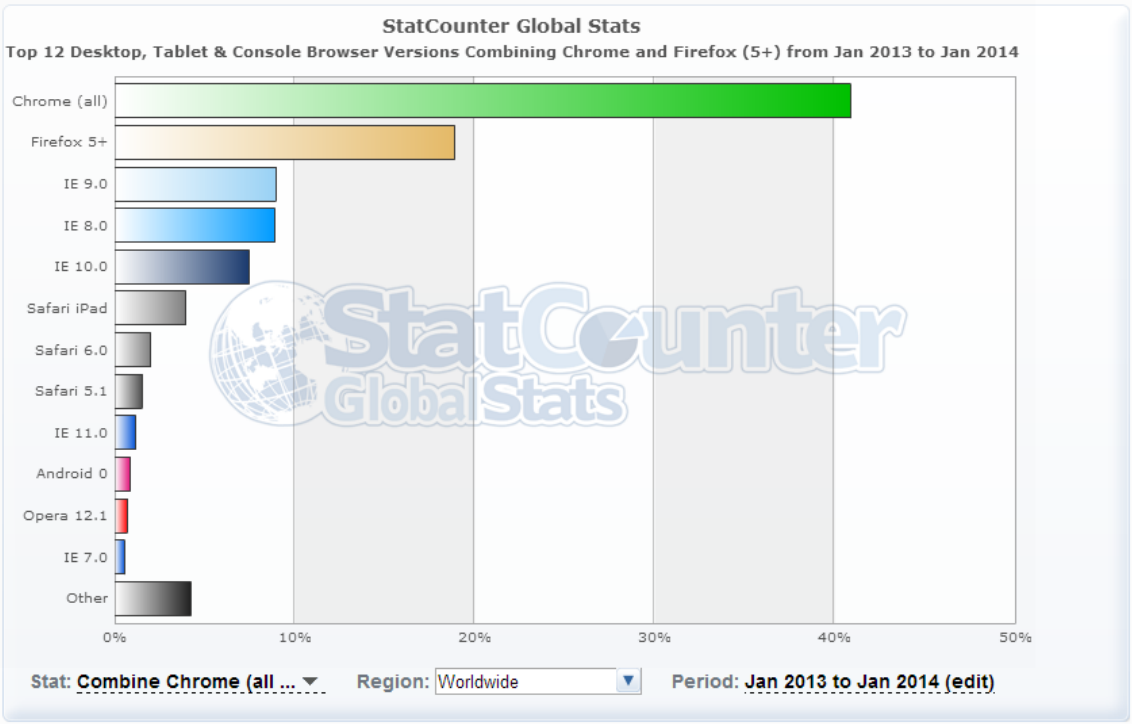
Webová aplikace byla vyvíjená v prohlížeči Google Chrome. Během vytváření aplikace byl tento prohlížeč několikrát automaticky aktualizován. To však nemělo žádný vliv na webovou aplikaci.
Projekt byl testován v prohlížečích Opera verze 20, Firefox verze 28, Google Chrome verze 33 a Internet Explorer verze 7, 8, 10.

Pro prácí s webovou aplikací je nutno použít myš. Pro zařízení s dotykovým displejem není aplikace vhodná, neboť při ovládání prstem je část logického prvku uschována pod ním. Tím se tedy hůře nastavuje pozice vybraného logického prvku. Manipulace s logickými prvky, po vložení dotykových událostí, byla velmi náročná.

## 4.1 Podpora webové aplikace v prohlížečích

Webová aplikace, jejíž vytvoření bylo cílem diplomové práce, byla naprogramována pro moderní prohlížeče s podporou HTML5, ve které je zahrnut element `canvas`. Tento element je pro aplikaci nezbytný, neboť slouží k dynamickému skriptovatelnému vykreslování grafických objektů.

Na obrázku č. 15 je zobrazen graf používaných webových prohlížečů za rok 2013, 2014. Nejčastěji používanými prohlížeči jsou: Google Chrome, Firefox, verze Internetu Exploreru (dále jen „IE“) 9.0, 8.0, 10.0.
Pro výše uvedené prohlížeče bylo nutno zajistit bezproblémové fungování webové aplikace s podporou dostupných pluginů jako jsou např. json, excanvas.
IE 8.0 a starší verze tohoto prohlížeče však bohužel nenabízí rozhraní Canvas. Tento nedostatek byl odstraněn použitím Open Source pluginu – excanvas.js, který dokáže nedostatky prohlížeče odstranit. Jelikož IE 7.0 má podobné nedostatky jako již zmiňovaný IE 8. 0, byla i tato verze zařazena do podporovaných prohlížečů.




*Obrázek č. 15 Webové prohlížeče používané v období leden 2013 – leden 2014 [23]*

Přestože IE podporuje rozhraní Canvas až od verze 9.0, starší verze podporují proprietární technologii VML (viz výše), která dokáže skoro totéž, co element `canvas`. Díky této technologii lze dosáhnout stejných metod pomocí knihovny excanvas.js[43].
ExplorerCanvas je javascriptová Open Source knihovna, která implementuje rozhraní Canvas do IE. ExplorerCanvas neumí nahradit všechny metody, které jsou podporovány rozhraním Canvas. Ve webové aplikaci jsou proto při vykreslování používány jen metody, které jsou podporovány právě knihovnou ExplorerCanvas. Starší verze IE nepodporují také JSON (viz níže).

## 4.2 Externí zdrojové kódy (pluginy)

Na internetu se nachází mnoho pomocných, předem vytvořených zdrojových kódů (tzv. pluginů). Úkolem těchto pluginů je usnadnit vývojáři práci při vytváření webové aplikace tím, že vybraný plugin do aplikace vloží (nemusí tedy část kódu v aplikaci vymýšlet).

Pluginy mohou pomáhat v různých oblastech webové aplikace. Jedná se zejména o:

- Správu jazyka HTML a CSS stylů pomocí JavaScriptu: různé webové prohlížeče mohou mít odlišný postup jak pracovat (tj. přidávat, upravovat, nebo mazat) s HTML dokumentem, a CSS styly pomocí javascriptového kódu;
- Doplnění chybných metod JavaScriptu: většina moderních prohlížečů má v současnosti zabudovaný *„JSON“ (JavaScript Object Notation)*[44]. Starší webové prohlížeče však JSON zabudovaný nemají. Je-li zapotřebí, přidává se javascriptový plugin, který tento nedostatek řeší;
- A další.

[43] Knihovna poskytuje možnost dvourozměrného kreslení v Internet Exploreru.
[44] JSON je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. Je založen na podmnožině programovacího jazyka JavaScritp, Standard ECMA-262 3rd Edition – December 1999. [9]

Pluginy jsou vývojáři vytvářeny velmi obecně. Je tomu tak proto, aby s nimi mohla pracovat velká skupina lidí. Takový přístup může být však nevýhodou samotných pluginů. Vývojář nemusí totiž použít všechny metody, které plugin nabízí a mnoho metod tak není ve webové aplikaci využito.
Jinou nevýhodou může být situace, kdy si vývojář stáhne chybný plugin. Tento plugin pak může zatěžovat webovou aplikaci více než je nezbytné (např. zacyklení aplikace).
Z tohoto důvodu bylo při vytváření aplikace omezeno používání pluginů stahovaných z internetu. Pro aplikaci byly vytvářeny pluginy vlastní. Pluginy tak byly pod kontrolou a obsažené metody plně využity.
Každý plugin byl navržen tak, aby pracoval jen se svými vlastními metodami nebo s metodami, které byly k němu přidruženy jako parametry příběhu webové aplikace. Toto přímo neplatí pro pluginy, které byly vytvořeny jako rozšířením jiného pluginu.

Mezi pluginy, které byly v projektu použity, patří: paint, modal, select a float.

1. Paint: tento plugin byl vytvořený pro práci s elementem `canvas`. Tím bylo dosaženo toho, že byly metody pro vytváření, vkládání a použití událostí na jednom místě.
   Hlavním účelem bylo zjednodušení práce s rozhraním Canvas tak, aby mohl být použit jakýkoliv atribut id z elementu `canvas`. Plugin Paint vytvoří potřebný kontext a předpřipraví element `canvas` tak, aby s ním bylo možné pracovat jen pomocí metod a jejich argumentů.
   Dalším účelem bylo zjednodušení vykreslení tvarů, pomocí pluginu, v jednoduchém asociativním poli, a tím vykreslit tvary s jejich vlastnostmi na základě daných požadavků.
   Třetím účelem tohoto pluginu je výpočet souřadnic na plátně a detekce různých typů událostí (`onclick`, `onmousemove`, `onmouseover`, `onmouseout`, `onmouseup`, `onmousedouwn` a další.) tak, aby výpočet souřadnic byl shodný v různých typech webových prohlížečů.
2. Modal: tento plugin vytvoří modální okno, ve kterém se např. může zobrazovat formulář pro nastavení vybraného logického prvku.
3. Select: účelem tohoto pluginu je zjednodušení práce s ikonami jako nástroji pro výběr. Plugin byl vytvořen pro práci s elementem `a` (odkaz), který obsahuje atribut `href` (cíl odkazu). Tento atribut slouží jako identifikátor pro inicializaci a práci v JavaScriptu.
   Plugin Select umí, dle nastavení nástrojům pro výběr, přidělovat dvě HTML třídy. Těmito třídami je třída Active (tzn. nástroj pro výběr je aktivní, je uživatelem používaný) a Disabled (tj. nástroj pro výběr je vypnut, uživatel jej nemůže použít).
4. Float: jedná se o plovoucí okno, které se zobrazí u logického prvku při jeho aktivaci a při jeho detekci se kurzor zobrazí v levém horním rohu. Plovoucí okno se zobrazuje při existenci základní informace detekovaného logického prvku (jestliže tato informace u prvku je).

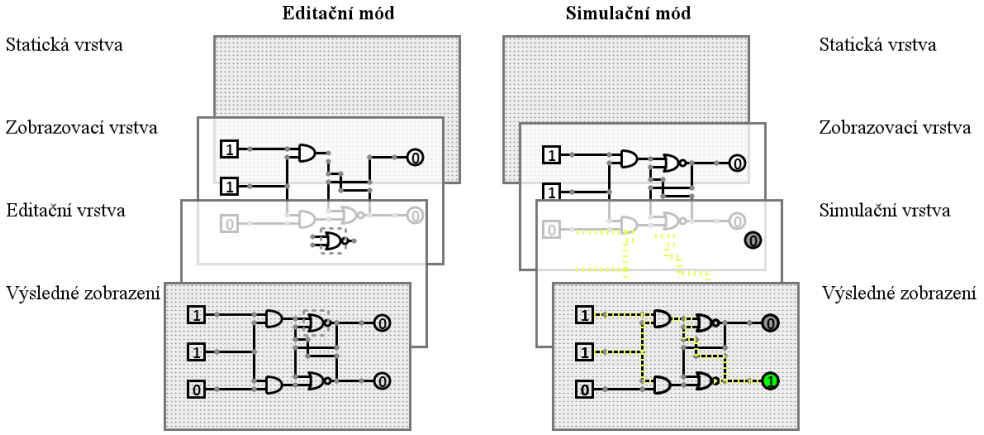
### 4.3 Plátno

Obecně jsou v plátnu postupně vykreslovány tvary, jakými jsou např. kruh, čtverec, obdélník, čára, křivka. Může však být vykreslen i obrázek či text. Vše je vykreslováno dle zadaných parametrů týkající se pozice, velikosti, šířky čáry, barvy hrany, barvy výplně apod.
Pokud jsou tyto tvary vykresleny na plátno, není možné již s nimi dále pracovat. Je-li přesto nutné změnit barvu či jinou vlastnost tvaru, textu či obrázku, je potřeba plátno vymazat a znovu tvary s novými vlastnostmi vykreslit.
Nejedná-li se o větší počet tvarů, webový prohlížeč je schopný tyto tvary znovu vykreslit za krátkou časovou jednotku. V popisované webové aplikaci však může mít logický prvek více jak deset tvarů; s použitím více logických prvků se může součet těchto tvarů pohybovat k tisícům. Prohlížeč tak bude

používat svou dostupnou paměť jen pro vykreslování a nebude mít volnou paměť pro jiné výpočty týkající se simulace.

Vytvořená webová aplikace má více pláten. Tyto plátna, chovající se jako vrstvy, jsou skládány nad sebe. Jsou transparentní, tudíž např. tvary vykreslené na statickém plátně jsou viditelné skrze ostatní plátna.
Jednotlivá plátna jsou na sebe centrovány pomocí CSS stylů.
Pro potřeby webové aplikace bylo zapotřebí čtyř vrstev plátna (viz obrázek č. 16). Každé z nich má svůj účel.

1. *Statická vrstva:* tato vrstva je vrstvou základní a je překrývána všemi ostatními vrstvami. Vrstva slouží statickému účelu, kterým je zobrazení aplikace při jejím spuštění. Nemění se po celou dobu používání aplikace či ostatních vrstev plátna. Obsahem jsou mřížky, tečky sloužící jako podklad pro přidávání prvků.
2. *Zobrazovací vrstva:* v této vrstvě jsou zobrazovány jiné tvary, než ty, které se právě upravují. Tvary jsou pouze zobrazeny, jsou nehybné. Mají pouze informovat o tom, kde se jednotlivé tvary nacházejí.
   Tvary ve vrstvě je možné překreslit. Vybere se požadovaný tvar, který se následně dostane do editační vrstvy (viz níže), kde je možné tvar překreslit. Po dokončení požadovaných úprav, se zobrazovací vrstva překreslí. Totéž se děje při přidávání nových tvarů a editaci.
3. *Editační vrstva:*třetí vrstva pracuje pouze v editačním módu. Jsou zde zobrazovány tvary při vkládání nového logického prvku a při potřebě překreslení tvaru logického prvku ze zobrazovací vrstvy.
   Jelikož se jedná jen o několik tvarů, není vykreslování tolik zatížené a pohyb těchto tvarů je plynulý.
   Po výběru pozice nového logického prvku, jsou jeho tvary vytištěny na zobrazovací vrstvu.
   Třetí vrstva se s poslední vrstvou vylučuje. Tzn., pokud je jedna vrstva aktivní a viditelná, druhá vrstva se automaticky skryje. Každá z vrstev má své vlastní události a může tak bránit druhé vrstvě ve správné funkčnosti.
4. *Simulační vrstva:* vrstva slouží pro zobrazení simulace. Dle pinů logických prvků probíhá v časovém intervalu simulace v simulačním módu. Simulace je obnovována do jejího pozastavení či ukončení.




*Obrázek č. 16 Typy vrstev plátna pro editační a simulační mód.*

## 4.4 Paměť webové aplikace

Pamětí je myšlen objekt pojmenovaný jako `mBrain`. Objekt `mBrain` neobsahuje žádné metody, pouze vlastnosti, na kterých jsou prováděny změny.
K některým těmto vlastnostem mohou být přiřazeny celé třídy, nebo jen čísla či řetězce. Třída mMain si v objektu může uchovávat spojení mezi různými třídami, mazat je či upravovat.

Práce a ukládání logických prvků v paměti aplikace:

```
mBrain = {
      circuit: [],
      activeElm:{
            cls: null,
            move: false,
            clone: false,
            select: false,
            shift: [],
            mouseDownCoor: null
      },
      .
      .
      .
}
```

Při úpravě či výběru logických prvků ve webové aplikaci se základní údaje prvků ukládají do objektu `mBrain` a podobjektu `activeElm`. V tomto podobjektu dochází k nastavení počátečních hodnot třídy logického prvku do proměnné *cls*. Tato proměnná je používaná pro rychlé adresování logického prvku tak, aby nemusela být vyhledávána při každé změně v seznamu logických prvků.

Mezi další proměnné podobjektu `active Elm` patří:

- *move* – pohyb logického prvku;
- *clone* – duplikace logického prvku se všemi jeho vlastnostmi;
- *select* – výběr logického prvku pro jeho editaci;
- *mouseDownCoor* – práce s myší;
- *shift* – ukládání pozice myši při kliknutí na plátno tak, aby při výběru prvků a jeho posunu bylo zaznamenáno, kam přesně bylo na ten prvek kliknuto. Tím se prvek bude držet kurzoru po celou dobu na stejném místě. Tato proměnná je určená pro nakreslení drátu (wire) a určení první pozice stisknutí myši.

## 4.5 Logické prvky

Třídy logických prvků jsou objektem tvořené tak, že při jeho instanci se vrátí do daného logického prvku objekt, který automaticky dědí všechny metody prototypu, včetně `GetVoltage`, `SetVoltage` a dalších.
Níže je uváděn příklad použití dědičnosti metod pro logické prvky na části třídy cAnd.

Logika použití vytvořené dědičnosti pro logický prvek spočívá ve třech krocích:

1. K proměnné *cAnd* je přiřazena funkce, která vrátí instanci cAnd třídy. `cAnd.fn.Init` je konstruktorem, který odkazuje na objekt `cAnd`.

```
cAnd = function(oSettings){
      return new cAnd.fn.Init(settings);
};
```

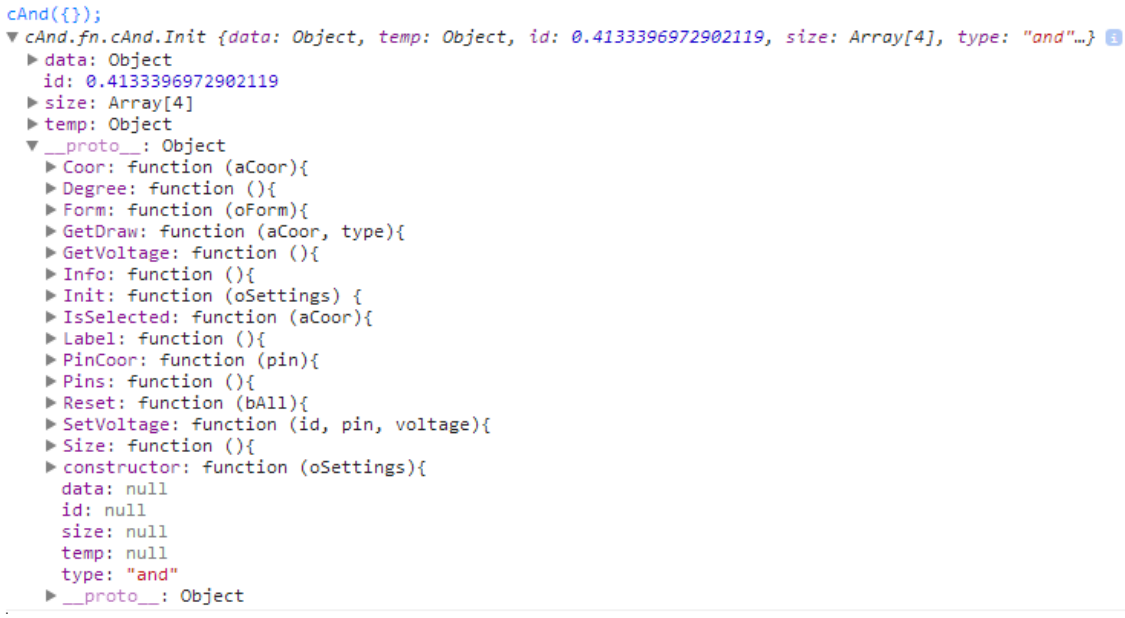
2. K proměnné je taktéž přiřazen přes prototyp (dědičnost) objekt metod. Tyto metody se dědí skrze všechny instance proměnné *cAnd*.
   Objekt musí obsahovat povinnou metodu `Init`, která nastavuje parametry. Jejím výstupem je objekt `this` (instance objektu `cAnd`), jenž ukazuje na element, ve kterém je obsažen.

```
cAnd.fn = cAnd.prototype = {
data: null,
temp: null,
constructor: cAnd,
Init: function(oSettings) {
      this.data = oSettings;
      return this;
};
```

3. Instance cAnd je jen obyčejný objekt, který má vlastnosti a metody zděděné prototypem `cAnd`.

```
cAnd.fn.Init.prototype = cAnd.fn;
```

Vzniklá třída cAnd v prohlížeči má tyto vlastnosti: `data`, `size`, `temp` a `__proto__` (jsou v něm obsažené děděné metody). K metodám obsažených v `__proto__` se přistupuje stejně jako ke klasickým metodám.




*Obrázek č. 17 Zobrazení inicializované třídy cAnd*

**Metody a proměnné tříd logických prvků**

Struktura tříd logických prvků je stejná pro všechny logické prvky. Nepotřebuje-li některý z logických prvků určitou metodu, je nastaven k vlastnosti této metody hodnota `null` namísto funkce. Metody mají často stejnou strukturu pro více logických prvků. Takové metody jsou přebírány z třídy cElement.

Mezi metody tříd logických prvků patří:

- `Init` - inicializování třídy logického prvku (přiřazení dat, přidělení identifikace a vypočtení velikosti logického prvku);
- `GetVoltage` - volání při simulaci; vrací pole obsahující všechny piny, tvořící výstupní piny logického prvku s přiřazenými logickými operátory;
- `SetVoltage` - nastavení hodnoty vstupních pinů logického prvku, které byly převzaty z výstupních pinů, s nimiž byly zapojené;
- `Pins` – vrací identifikační pole všech pinů z logického prvku;
- `PinCoor` – vrací souřadnice x a y daného pinu logických prvků;
- `Reset` – metoda resetující dočasná data potřebná pro simulaci;
- `Form` – generuje data logického prvku pro formulář nastavení logického prvku a po uložení jej aplikuje na logický prvek;
- `Info` – generuje základní data plovoucího okna prvku při simulaci, která se mohou dynamicky měnit;
- `Label` - vrací název logického prvku;
- `RecSave` – nahrává změnu logických prvků;
- `RecReset` – maže nahraná data;
- `Degree` - nastavuje rotaci logického prvku;
- `Coor` – nastavuje souřadnice logického prvku;
- `Size` - vrací velikost logického prvku;
- `IsSelected` – detekuje, zda se logický prvek nachází v místě nastavených souřadnic;
- `GetDraw` - vykresluje logické prvky pro zobrazení v: klasickém režimu s použitím grafických tvarů; editačním režimu, kdy je logický prvek vyzdvižen tak, aby ho šlo rozeznat od ostatních; simulačním režimu, kdy je na logickém prvku zřetelné, že přes něj protéká signál (jako např. logická 1).

Proměnnými logických prvků jsou:

- *data* – používá se k ukládání, načítání či duplikaci logického prvku;
- *temp* – zajišťuje ukládání dočasných dat pro chod aktivní simulace. Po ukončení simulace se operátor `temp` automaticky vyprázdní metodou `Reset`;
- *type* – určuje typ logického prvku;
- *name* – určuje standardní název logického prvku;
- *size* – udává velikost logického prvku, ze které se odvozuje označení logického prvku na plátně. Pokud je logický prvek dynamický, je potom i jeho velikost dynamická;
- *id* – zajišťuje jedinečnou identifikaci logických prvků;
- *isVoltage* – potvrzuje, zda se jedná o aktivní logický prvek.

## 4.6 Třída Element

Hlavní spojnicí všech logických prvků a jejich tříd je třída Element. Třída Element spojuje třídy logických prvků se základní třídou, která se stará o práci s tlačítky a vykreslovaní do rozhraní Canvas. Vstupem je název třídy elementu popřípadě nastavená data. Výstupem pak vygenerovaný objekt, který si nese předaná data, nebo nově inicializovaná data vhodná pro daný objekt.

Tato třída není jen spojnicí, ale také se v ní vyskytují další dva typy metod.

- metody, které jsou potřebné pro chod jiných metod ve třídě logických prvků.
- často opakované metody tříd logických prvků. Většinou jsou tyto metody stejné napříč všemi třídami logických prvků. Pokud se však jedná o logický prvek, který vyžaduje speciální úpravu vlastností, jde tuto metodu napsat nově v dané třídě logického prvku.

## 4.7 Simulace logických prvků v logickém obvodu

Při simulaci logického obvodu prochází signál vstupy logických prvků. Na základě hodnot logických prvků vyhodnotí signál jejich výstupy. Za jeden krok v simulaci mohou být tyto hodnoty změněny několikrát. Pokud však dojde k zacyklení simulace, může se změna provádět tak dlouho, dokud simulace nepřejde do dalšího kroku. Tím dochází k ukončení zacyklení jednoho kroku.
Mezi jednotlivé kroky simulace je nastaveno 50 milisekundové zpoždění. Toto zpoždění ponechává simulaci dostatek času na to, aby se v nezacykleném obvodu stihla celá provést.

### *4.7.1 Souřadnice koncových pinů jako uzlů*

Souřadnice koncových pinů logického prvku se odvíjí od umístění logického prvku. Ve většině případů je tato souřadnice udána kolem středu logického prvku. Výchozí hodnota souřadnic logického prvku je x = 0 a y = 0.
Od hodnoty této souřadnice se při vytváření logického prvku vypočítá daný pin. Jeho hodnota je dána staticky nebo dynamicky (v případě, že je možné logický člen a umístění pinů změnit). K výchozí souřadnici logického prvku se připočítává reálná souřadnice umístěna na plátně.

### *4.7.2 Startovní logické prvky*

Webová aplikace disponuje dvěma druhy logických prvků, a to *aktivními* a *neaktivními*. Neaktivními logickými prvky jsou myšleny takové prvky, které jsou v klidovém režimu a samy od sebe nespouští simulaci. Tento typ logického prvku může být zařazen do simulace, pokud je pomocí uzlu propojen s aktivním logickým prvkem.
Aktivní logické prvky jsou vyžadovány při inicializování simulačního obvodu. Odvíjí se od nich budoucí simulace logického obvodu. Tento typ logických prvků nemusí mít všechny své piny aktivní. Dle vstupu či samoúčelně může probíhat změna pinů.

### *4.7.3 Tok signálu v logickém obvodu*

Tok signálu v logickém obvodu pobíhá ve třídě cCalculate , která vypočítává průběh simulace pomocí třech metod: `TableToKnots`, `GetVoltage` a `SetVoltage`.

a. `TableToKnots`: metoda prochází všechny vytvořené logické prvky a pomocí metody `PinCoor` získává jejich souřadnice na plátně. Tyto souřadnice jsou chápány jako uzly a jsou podle těchto souřadnic indexovány do objektu `oKnots` pro rychlé použití v simulaci.
b. `GetVoltage`: z vytvořených logických prvků pomocí jsou metody `isVoltage` postupně vybírány aktivní logické prvky a vkládány do pole objektu `oFlow.queue`. Pokud uživatel

použije některý z logických prvků, je přiřazena k frontě hodnota označující to, že je prvek používán.

c. `SetVoltage`: metoda přiřazuje hodnoty k pinům logických prvků. Celý tento proces začíná již předvytvořenou základní frontou v objektu `oFlow`. Z fronty je vybrán první logický prvek a následně hodnoty z jeho výstupních pinů. Ke každému logickému prvku ve frontě je přiřazen informace o tom, že prvek již čeká ve frontě.

   Pokud je při metodě `GetVoltage` v logickém prvku vrácena odezva v podobě hodnoty `null`, znamená to, že je logický prvek již koncový. Pokračuje se s dalším logickým prvkem ve frontě. V opačném případě vrátí metoda pole pinů z logického prvku obsahující hodnoty výstupu. Tyto piny jsou postupně procházeny cyklem a pomocí metody `PinCoor` jsou získávány souřadnice těchto pinů pro určení pozice uzlu. Tento uzel je použit jako index v objektu `oKnots`, který vrací všechny logické prvky v poli se stejnou souřadnicí pinu jako je uzel. Pole cyklicky prochází piny a ke každému přiřazuje pomocí metody `SetVoltage`, ve třídě logických prvků, hodnotu z výstupu logického prvku.

   Logické prvky jsou dále vloženy na konec fronty, v případě, že tam ještě nejsou. U každého přidaného logického prvku je zobrazeno, že se jedná o prvek čekající ve frontě. Poté, co logický prvek předá hodnoty všem logickým prvkům napojených na jeho výstupní piny, je zrušen jeho čekající stav. Pokračuje se následujícím logickým prvkem ve frontě, dokud cyklus nedojde na konec fronty.

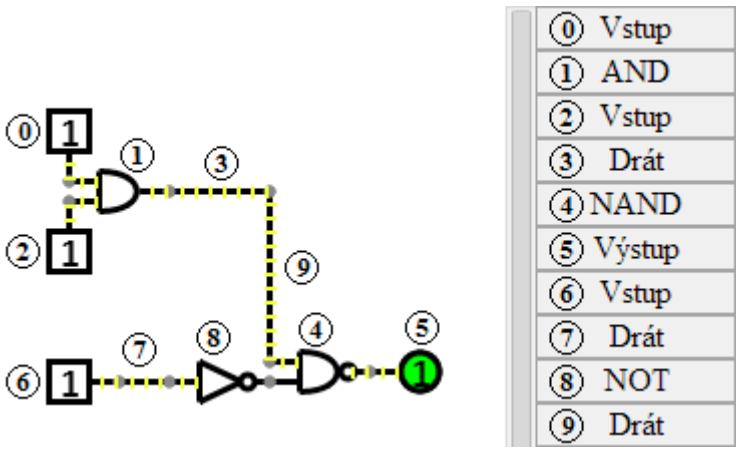
   Tato metoda simulace zajišťuje nižší dopad zacyklení na chod simulace. V každém kroku prochází signál každým logickým prvkem propojeným s aktivním logickým prvkem. Pokud je určitá část logického obvodu zacyklena, tak se může cyklit nekonečně krát až na konci fronty. Jestliže se tak děje, je toto zacyklení zrušeno dalším krokem simulace.

### *4.7.4 Příklad průběhu signálu v simulaci*

Logické prvky na poli mají, při vytváření logického obvodu, různé pozice ve frontě. Toto je dáno tím, že logický obvod není vytvářen od počátku obvodu až ke konci, i když může uživatel takto postupovat.

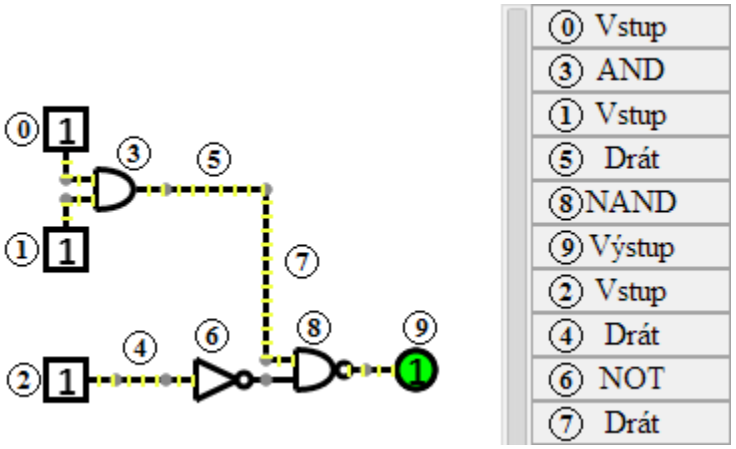
Vytváření logického obvodu a průběžná práce s ním zahrnuje přidávání a odebírání logických prvků. Nově přidaný prvek je zařazen vždy na konec fronty.

Následující obrázek zobrazuje vytvořený logický obvod s jeho uspořádanými logickými prvky v poli.




*Obrázek č. 18 Uspořádání logických prvků v poli*

Základem výše zobrazené simulace je vytvoření fronty pomocí aktivních logických prvků – indexy vstupů 0, 2, a 6.
Celá fronta je vložena do metody `SetVoltage` a v rámci simulace spuštěna prvním logickým prvkem v pořadí – index vstupu 0. Tento logický prvek předá své výstupní hodnoty následujícímu logickému prvku AND.
Dalším logickým prvkem ve frontě je index vstupu 2. Jelikož je jeho následujícím logickým prvkem AND, který již ve frontě čeká, nepřidává se již další logický prvek do fronty, ale pokračuje se dalším logickým prvkem – index vstupu 6. Následujícím logickým prvkem je drát – index 7. Do jeho vstupních pinů se vloží výstupní piny z indexu vstupu 6. Logický prvek – drát - je přidán na konec fronty.
Ve frontě opět čeká logický prvek AND. Ten je propojen s drátem – indexem 3, kterému jsou předány výstupní hodnoty. Jelikož index 3 není ve frontě, je přidán na její konec.
Tímto způsobem je cyklicky procházen logický obvod do chvíle, kdy signál projde všechny logické prvky ve frontě.




*Obrázek č. 19 Zápis logických prvků do fronty*

### *4.7.5 Prvek drát (propojení)*

Prvek drát je jednoduchou úsečkou, kterou je možné vykreslit pouze horizontálně a vertikálně. V rámci simulace je tento prvek chápán spíše jako uzel.
Výstupem drátu jsou všechny piny logických prvků napojených na oba konce úsečky.

### *4.7.6 Základní logické členy*

Pro základní logické prvky jsou vytvořeny třídy cAnd, cOr, cNot, cXor, které rozeznávají, zda se jedná o základní logický člen (např. AND), nebo jeho negaci (např. NAND). U těchto prvků se předpokládá stejná binární hodnota, jak na vstupních hodnotách, tak na hodnotách výstupních.

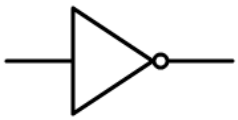
Základní logické prvky pracují s bitovými operátory, které pracují na úrovni jednotlivých bitů. Čísla jsou zde vyjádřena ve dvojkové soustavě (1 a 0).
Rovnice pro získání výstupní binární hodnoty je skládána do řetězce, složeného z binárních hodnot se správným počtem bitů.
Při negování výsledku, použitím bitového operátoru NOT ~ , se vyhodnocená hodnota změní na záporné binární číslo. V rovnici je proto nutné posunout zápornou binární hodnotu pomocí operátoru >>> .
Rovnice je vyhodnocena pomocí metody `Eval()`[45].

[45] Metoda `Eval()` vyhodnocuje kód JavaScriptu reprezentovaný jako řetězec.

*NOT*

Jedná se o prvek, který neguje vstupní hodnoty do výstupní hodnoty pomocí bitového operátoru ~ .

Řetězec, který reprezentuje rovnici pro logický prvek NOT, může být zapsán v těchto rovnicích: „(0)>>>0“, „~(1)>>>0“, ale také „~(0101)>>>0“.



| A | B | ~A | ~B |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

| A | B | ~A | ~B |
|---|---|---|---|
| 0000 | 0000 | 1111 | 0000 |
| 0011 | 0101 | 1100 | 1010 |
| 1100 | 1010 | 0011 | 0101 |
| 1111 | 1111 | 0000 | 0000 |

*Obrázek č. 20 Schematický znak logického prvku „NOT“*

*Tabulka č. 10 a) Stavová tabulka logického prvku NOT pro jednobitové binární hodnoty; b) Stavová tabulka logického prvku NOT pro čtyřbitové binární hodnoty*

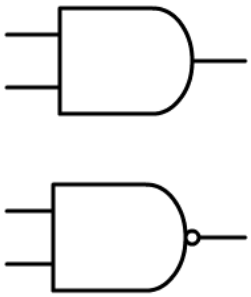
A – datový vstup;
B – datový vstup;
~A – použití logického prvku NOT na vstup A;
~B – použití logického prvku NOT na vstup B.

*AND A NAND*

Logická funkce AND je logickým součinem, který může být definován i pro více vstupních proměnných. Výsledek logického součinu několika proměnných je roven pouze v případě, že všechny vstupní proměnné jsou současně rovny jedné. [7]

Vstupní data těchto logických prvků se mohou pohybovat od dvou vstupních pinů. Data logického prvku AND jsou cyklicky sbírána. Po sběru dat jsou jejich vstupní hodnoty vkládány do řetězce.

Mezi hodnoty je vkládán bitový operátor & , reprezentující logický prvek AND. Výsledný řetězec – rovnice může vypadat takto: „1&0” nebo „110&010&011”. V případě negace logického prvku AND, tedy NAND vypadá výsledná rovnice takto: „~(110&010&011)>>>0”.



| A | B | A&B | ~A&B |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

| A | B | A&B | ~A&B |
|---|---|---|---|
| 0000 | 0000 | 0000 | 1111 |
| 0011 | 0101 | 0001 | 1110 |
| 1100 | 1010 | 1000 | 0111 |
| 1111 | 1111 | 1111 | 0000 |

*Obrázek č. 21 Schematický znak logického prvku a) AND; b) NAND*

*Tabulka č. 11 a) Stavová tabulka logického prvku AND a NAND pro jednobitové binární hodnoty;b) Stavová tabulka logického prvku AND a NAND pro čtyřbitové binární hodnoty*

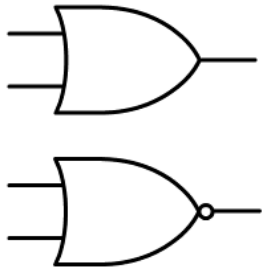
A – datový vstup;
B – datový vstup;
A&B – použití logického prvku AND;
~A&B – použití logického prvku NAND.

*OR A NOR*

Logická funkce OR je logickým součtem, který může být také definován i pro více vstupních proměnných. Výsledek logického součtu několika proměnných je roven jedné vždy, když alespoň jedna vstupní proměnná je rovna jedné. [7]

U těchto logických prvků se vstupní data mohou také pohybovat od dvou vstupních pinů. Data logického prvku OR jsou cyklicky sbírána. Po sběru dat jsou jejich vstupní hodnoty vkládány do řetězce.

Mezi hodnoty je vkládán bitový operátor | , reprezentující logický prvek OR. Výsledný řetězec – rovnice – může vypadat takto: „1|0|0“ nebo také „110|010|011“. V případě negace logického prvku OR - NOR, vypadá výsledná rovnice takto: „~(1|0|0)>>>0“.



| A | B | A\|B | ~A\|B |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |

| A | B | A\|B | ~A\|B |
|---|---|---|---|
| 0000 | 0000 | 0000 | 1111 |
| 0011 | 0101 | 0111 | 1000 |
| 1100 | 1010 | 1110 | 0001 |
| 1111 | 1111 | 1111 | 0000 |

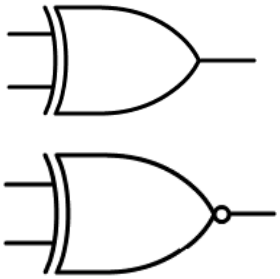
*Obrázek č. 22 Schematický znak logického prvku a) OR; b) NOR*
*Tabulka č. 12 a) Stavová tabulka logického prvku OR a NOR pro jednobitové binární hodnoty; b) Stavová tabulka logického prvku OR a NOR pro čtyřbitové binární hodnoty*

A – datový vstup;
B – datový vstup;
A|B – použití logického prvku OR;
~A|B – použití logického prvku NOR.

*XOR a XNOR*

Jedná se o výlučný logický součin, který může být také definován pro více vstupních proměnných. Výstup je roven jedné vždy, pokud se hodnoty vstupu liší.

U těchto logických prvků platí také již výše uvedené skutečnosti: je možné nastavit jejich vstupy od dvou možných pinů; hodnoty vstupních pinů jsou cyklicky sbírány. Po sběru dat jsou jejich vstupní hodnoty vkládány do řetězce. Mezi hodnoty je vkládán binární operátor ^ , reprezentující XOR. Výsledná řetězec – rovnice může vypadat takto: „1^0^0“ nebo také „110^010^011“. V případě negace logického prvku NOR – XNOR vypadá výsledná rovnice takto: „~(1^0^0)>>>0“.



| A | B | A^B | ~A^B |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

| A | B | A^B | ~A^B |
|---|---|---|---|
| 0000 | 0000 | 0000 | 1111 |
| 0011 | 0101 | 0110 | 1001 |
| 1100 | 1010 | 0110 | 1001 |
| 1111 | 1111 | 0000 | 1111 |

*Obrázek č. 23 Schematický znak logického prvku a) XOR; b) NXOR*
*Tabulka č. 13 a) Stavová tabulka logického prvku XOR a NXOR pro jednobitové binární hodnoty; b) Stavová tabulka logického prvku XOR a NXOR pro čtyřbitové binární hodnoty*

A – datový vstup;
B – datový vstup;
A|B – použití logického prvku XOR;
~A|B – použití logického prvku XNOR.

### *4.7.7 Klopné obvody*

Sekvenční logický obvod je složen z kombinační a paměťové části. Hodnotu výstupu proměnné *Q* je možné určit sledováním nejen vstupních proměnných, ale také proměnných uložených v paměti. Jedná se o proměnné, které jsou uchovávány v dynamické paměti třídy Temp.

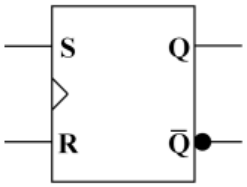
U sekvenčních logických obvodů je zaveden řídící synchronizační signál v podobě vstupu. Při příchodu řídícího signálu se vstupní proměnné změní dle stavových tabulek.
Výstup negovaného Q je pro sekvenční logické obvody v této aplikaci nastaven negací výstupu Q.
U žádného z klopných obvodů nesmí být vstupní piny v hodnotě `null`. Pokud tomu tak je, vychází se dle paměti logického prvku.

*RS klopný obvod*

Vstupem RS klopného obvodu jsou piny R (reset – nulování; S (set – nastavení); C (clock – hodiny). V paměti jsou dále uloženy proměnné *active* (uložení předchozího stavu pinu C) a *save* (uložení výstupního stavu Q).
Z těchto vstupních pinů a paměti jsou vybrána data a vyhodnocena pomocí podmínky:

```
if((R!==null && C!==null && S!==null) && !this.temp.active && C===true && R!==S){
        if(S) Q=true;
        if(R) Q=false;
}else{
        Q=this.temp.save;
}
```

Z první části podmínky je patrné, že nový stav bude uložen pouze tehdy, pokud je předchozí stav logické 1 převeden na `false` a nový stav logické 1 převeden na `true`.
Druhou část podmínky určuje, že nová změna nebude brána z paměti logického prvku, pokud budou vstupy R a S od sebe odlišné.
Pokud se RS klopný obvod dostane do zakázaného stavu, není provedena žádná změna a výstupy jsou načteny z paměti.




| R | S | $Q_{n+1}$ | $\overline{q_{n+1}}$ |
|---|---|---|---|
| 0 | 0 | $Q_n$ | $\overline{Q_n}$ |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | Zakázaný stav | |

*Obrázek č. 24 Schematický znak logického prvku RS*
*Tabulka č. 14 Stavová tabulka logického prvku RS*

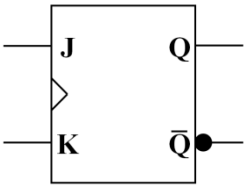
R – datový vstup;
S – datový vstup;
$Q_{n+1}$ – datový výstup následující stav;
$\overline{Q_{n+1}}$ – datový výstup negovaný následující stav;
$Q_n$ – aktuální stav.
$\overline{Q_n}$ – negovaný aktuální stav.

*JK klopný obvod*

JK klopný obvod se liší od RS klopného obvodu v tom, že umožňuje definovat logickou hodnotu i v případě, že vstupy J a K mají logickou 1.
Stejně tak jako je tomu u klopného obvodu RS, má i JK klopný obvod vstupní pin C a proměnné *active* a *save* uložené v paměti.

```
if((J!==null && C!==null && K!==null) && !this.temp.active && C===true && J!==K){
        if(J) Q=true;
        if(K) Q=false;
} else if(!this.temp.active && C===true && J===true && K===true){
        Q=!this.temp.save;
} else {
        Q=this.temp.save;
}
```

První část podmínky se neliší od podmínky RS klopného obvodu. U JK klopného obvodu však nenastává zakázaný stav.
Druhá část podmínky je tedy založena na tom, že J i K budou logické 1. Tato skutečnost způsobí negaci hodnoty výstupního pinu Q uložené v paměti. Jestliže budou mít vstupy J i K logickou 1, hodnota výstupního pinu Q se bude rovnat paměti logického prvku (*this.temp.save*).




| J | K | $Q_{n+1}$ | $\overline{q_{n+1}}$ |
|---|---|---|---|
| 0 | 0 | $Q_n$ | $\overline{Q_n}$ |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | $\overline{Q_n}$ | $Q_n$ |

*Obrázek č. 25 Schematický znak logického prvku JK*
*Tabulka č. 15 Stavová tabulka logického prvku JK*

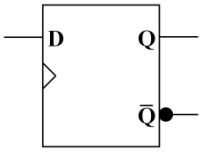
J – datový vstup;
K – datový vstup;
$Q_{n+1}$ – datový výstup následující stav;
$\overline{Q_{n+1}}$ – datový výstup negovaný následující stav;
$Q_n$ – aktuální stav.
$\overline{Q_n}$ – negovaný aktuální stav.

*D klopný obvod*

Výstupy D klopného obvodu kopírují stav vstupu po dobu, kterou je vstupní signál C ve stavu logické1. Jestliže se změní hodnota vstupu C v logickou 0, zůstává na výstupu Q zachován poslední stav, který byl zaznamenán při hodnotě, kdy byl vstup C = 1.

```
if((D!==null && C!==null) && !this.temp.active && C===true){
        Q = D;
} else {
        Q = this.temp.save;
}
```

Pokud nastane změna řídícího signálu C v logickou 1, pak je na výstupní pin Q vypsána hodnota pinu D. V případě, že časový vstupní pin C bude logická 0, potom Q se bude rovnat hodnotě z paměti.




| D | $Q_{n+1}$ | $\overline{q_{n+1}}$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

*Obrázek č. 26 Schematický znak logického prvku D*
*Tabulka č. 16 Stavová tabulka logického prvku D*

D – datový vstup;
$Q_{n+1}$ – datový výstup následující stav;
$\overline{Q_{n+1}}$ – datový výstup negovaný následující stav;

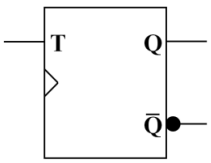
*T klopný obvod*

Klopný obvod T je přepínačem paměti. S každou změnou řídícího signálu a vstupního pinu T, který je roven logické 1, se provede změna výstupního pinu Q na jeho negaci.

```
if(!this.temp.active && C===true && T===true){
        Q = !this.temp.save;
} else {
        Q = this.temp.save;
}
```

V případě, že bude vstupní pin T ve stavu logická 1 a dojde k časové změně – vstupní pin C bude také ve stavu logická 1, proběhne záměna výstupního pinu Q ve stavu logická 1 za logickou 0

nebo naopak. Jestliže proběhne změna časového vstupního pinu C ve stavu logická 1 na logickou 0, hodnota Q se nezmění.




| D | $Q_{n+1}$ | $\overline{q_{n+1}}$ |
|---|---|---|
| 0 | $\overline{Q_n}$ | $Q_n$ |
| 1 | $Q_n$ | $\overline{Q_n}$ |

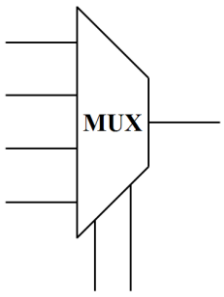
*Obrázek č. 27 Schematický znak logického prvku T*
*Tabulka č. 17 Stavová tabulka logického prvku T*

D – datový vstup
Q – datový výstup;
$Q_{n+1}$ – datový výstup následující stav;
$\overline{Q_{n+1}}$ – datový výstup negovaný následující stav;
$\mathrm{Q_n}$ – aktuální stav.
$\overline{\mathrm{Q_n}}$ – negovaný aktuální stav.

### *4.7.8 Kombinační logický obvod*

*Multiplexer*

Multiplexer je kombinační logický obvod, který plní funkci elektronického přepínače několika vstupních logických signálů na jeden výstup, přičemž přepínání je řízeno adresovým signálem.
Multiplexer má obecně $k$ datových vstupů, příslušený počet adresových vstupů (pro počet adresových vstupů $n$ platí k=2^n) a jeden datový výstup. [15]
Data pro výstup jsou získávána tak, že se postupně vypočtou logické hodnoty z adresového vstupu složené do dvojkové hodnoty a tato hodnota je použita pro výpočet datových vstupů. Vypočtená hodnota vstupu bude použita do datového výstupu.




| $S_0$ | A | B | $y_0$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

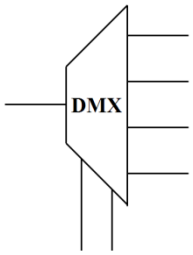
*Obrázek č. 28 Schematický znak multiplexeru*
*Tabulka č. 18 Stavová tabulka multiplexeru*

$S_0$ - adresový vstup;
A, B – datové vstupy;
$y_0$ - datový výstup.

*Demultiplexer*

Demultiplexer přepíná jeden vstupní signál na několik výstupů, přičemž příslušný výstup se vybírá adresovým signálem. Je-li tedy přenášen multiplexovaný signál z n kanálů po jednom vedení, můžeme ho pomocí demultiplexoru transformovat zpět do n kanálů.

Data pro výstup jsou získávány podobně jako u multiplexoru. Logické hodnoty z adresového vstupu S se složí do dvojkové hodnoty, která se následně převede na desítkovou hodnotu. Dle výsledného čísla je vybrán výstupní pin a do něj je vložena hodnota ze vstupního pinu.



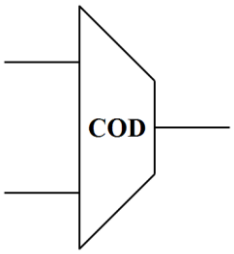

| $S_0$ | A | $y_0$ | $y_1$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |

*Obrázek č. 29 Schematický znak demultiplexeru*
*Tabulka č. 19 Stavová tabulka demultiplexeru*

*Coder*

Kodér převádí vstupní kód 1 z n na binární kód, ve kterém je v daném okamžiku aktivní pouze jeden vstup logického prvku.

Kodér zajišťuje, aby byl aktivní pouze jeden výstup logického prvku.



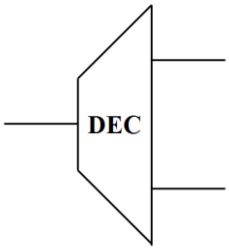

| $U_7$ | $U_6$ | $U_5$ | $U_4$ | $U_3$ | $U_2$ | $U_1$ | $U_0$ | $y_2$ | $y_1$ | $y_0$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

*Obrázek č. 30 Schematický znak coderu*
*Tabulka č. 20 Stavová tabulka coderu*

*Decoder*

Dekodér převádí binárně kódované n-bitové číslo na $2^n$ výstupu kódovaných jako 1 z $2^n$. Binární dekodér má tedy n vstupů a $2^n$ výstupů, z nichž je v daném okamžiku vždy aktivní pouze jeden. Číslo aktivního výstupu odpovídá hodnotě vstupního binárního čísla.

Ze vstupů je sestaveno binární číslo, které je převedeno na číslo v desítkové soustavě a vložené do příslušného výstupu na pozici získaného desítkového čísla.




| $U_2$ | $U_1$ | $U_0$ | $y_7$ | $y_6$ | $y_5$ | $y_4$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Obrázek č. 31 Schematický znak decoderu*
*Tabulka č. 21 Stavová tabulka decoderu*

# ZÁVĚR

Diplomová práce slučovala dvě oblasti – informatiku a elektrotechniku za účelem vytvoření webového simulátoru logických obvodů, který byl cílem práce.
První část práce tvořila teoretický souhrn o jazyce HTML5, JavaScriptu, číslicových technikách a logických obvodech. Získání těchto informací bylo nutné pro vytvoření webové aplikace - simulátoru logických obvodů.
Druhá – praktická část diplomové práce, prostřednictvím obrázků a grafů popisovala jednotlivé části – prvky webové aplikace.

Vytváření webové aplikace probíhalo bez větších problémů. Menší komplikací bylo sjednocení JavaScriptu pro různé typy prohlížečů a jejich verze. Čím byla starší verze prohlížeče, tím měla více chybějících metod, které jsou ve vyšších verzích implementovány automaticky.
Pro tyto prohlížeče byl implementován plugin obsahující JavaScriptový kód, který vytvářel stejný výsledek.

Webová aplikace je pilotní verzí. Pokud by měl být projekt vydán, je nutná jeho úprava spíše v grafické stránce projektu. Tato oblast je nedostatečná.
Celá aplikace by měla být taktéž více uživatelsky přívětivá, tzn. usnadnění uživatelům práci při vkládání nových logických prvků. Pro usnadnění by bylo vhodné: automatické přitahování uzlů logických prvků, jako by se jednalo o dva magnety; vykreslování drátu přímo z uzlu, aniž by byl vkládán z panelu a hromadný výběr logických prvků.

# LITERATURA


[1] ANTOŠOVÁ, M., DAVÍDEK, V. *Číslicová technika* 1.vyd. České Budějovice: KOPP, 2003. 286 s. ISBN 80-7232-206-0

[2] API. *Itbiz* [online]. Dostupné z www <http://www.itbiz.cz/slovnik/informacni-technologie-it/api>.

[3] Apple Newton aneb první PDA z roku 1993. *Softlmage.* [online]. Dostupné z www <http://magazin.softimage.cz/apple-newton-aneb-prvni-pda-z-roku-1993/>.

[4] BIELIKOVÁ, M. Programovacie jazyky. *Homepage of Mária Bieliková* [online]. © 2001. Dostupné z www <http://www2.fiit.stuba.sk/~bielik/courses/flp-slov/jazyky/programovacie_jazyky.html#Self>.

[5] DANIEL. *Svg. Blog. Geospy. Org. Official Blog of Headquarters Geospy .org* [online], [cit. 2013-12-09]. Dostupné z www <http://blog.geospy.org/post/69467083133/svg-autor-daniel-hq-geospy-svg-scalable>.

[6] FLANAGAN, D. *JavaScript: Kompletní průvodce.* 2. akt. vyd. Praha: Computer Press, 2002. 825 s. ISBN 80-7226-626-8.

[7] FOLTÝNEK, P. Logické prvky a systémy. *Server reálných aplikací „352LAB. vsb. cz"* [online]. © 2005. Dostupné z www <http://352lab.vsb.cz/MinServer/PraceStud/LPaS/_2002_2003/_Foltynek/okruh1. html>.

[8] IETF, IPTC, Sémantický web. *Centrum podpory goneo.cz* [online]. Dostupné z www <http://podpora.goneo.cz/975363-IETF-IPTC-S%C3%A9mantick%C3%BD-web?r=1>.

[9] Introduction JSON. *JSON* [online]. Dostupné z www <http://www.json.org/>.

[10] JavaScript. *Adaptic, s. r. o.* [online]. © 2005 – 2014. Dostupné z www <http://www.adaptic.cz/znalosti/slovnicek/javascript/>.

[11] LARES, J. Ruby I. *Linuxsoft* [online], [cit. 2011-12-03]. © 2003 – 2014. Dostupné z www <http://www.linuxsoft.cz/article.php?id_article=1883>.

[12] LUBBERS, P., ALBERS, B., SALIM, F. *HTML5: Programujeme moderní webové aplikace.* 1. vyd. Brno: Computer Press, a. s., 2011. 304 s. ISBN 978-80-251-3539-6.

[13] MALÝ, M. Web Sockets. *Zdroják* [online], [cit. 2009-12-14]. Dostupné z www <http://www.zdrojak.cz/clanky/web-sockets/>.

[14] MATOUŠEK, D. *Číslicová technika: základy konstruktérské praxe.* 1.vyd. Praha: BEN, 2001. 208 s. ISBN 80-7300-025-3.

[15] Multiplexor. *Mikrokontroléry PIC. Web o číslicové technice a mikrokontrolérech PIC* [online]. © 2012. Dostupné z www <http://mikrokontrolery-pic.cz/zaciname/cislicova-technika/kombinacni-logicke-obvody/multiplexor/>.

[16] PILGRIM, M. Říkejme tomu (plocha na) kreslení. *HTML5* [online]. Dostupné z www <http://kniha.html5.cz/canvas.html>.

[17] PÍSEK, S. *HTML. Začínáme programovat.* 3. akt. vyd. Praha: Grada Publishing, 2010. 190 s. ISBN 978-80-247-3117-9.

[18] POLLOCK, J. *JavaScript. Příručka programátora.* České vyd. Praha: SoftPress, 2003. 560 s. ISBN 80-86497-44-5.

[19] Princip značkovacích jazyků a základní struktura (X)HTML dokumentu. *Informatika @ grafika* [online]. Dostupné z www <http://www.gjszlin.cz/ivt/esf/dreamweaver/princip-znackovacich-jazyku-a-zakladni-struktura-html-dokumentu.php>.

[20] SEZEMSKÝ, M. Co je to HTTP a HTML. *Linux* [online], [cit. 1999-03-20]. © 2007 – 2014. Dostupné z www <http://www.linux.cz/noviny/1999-0304/clanek10.html>.

[21] SUEHRING, S. *JavaScript*: *Krok za krokem*. 1. vyd. Brno: Computer Press, a. s., 2008. 335 s. ISBN 978-80-251-2241-9.

[22] ŠVEC, M. Programovací jazyk a aplikační prostředí SELF. *Vysoké učení technické. Fakulta informačních technologie* [online], [cit. 2014-02-07]. Dostupné z www: <http://www.fit.vutbr.cz/study/courses/TJD/public/0304TJD-Svec.pdf>.

[23] Top 12 Desktop, Tablet & Consele Browser Versions Combining Chrome and Firefox (5+) form Jan 13 to Jan 14. *StatCounter Global Stats* [online]. © 1999 – 2014. Dostupné z www <http://gs.statcounter.com/#browser_version_partially_combined-ww-monthly-201301-201401-bar>.

[24] WAI-AIRA Overview. *W3C. Web Accessibility initative* [online]. © 1994 – 2014. Dostupné z www <http://www.w3.org/WAI/intro/aria>.

[25] WebKit Projext Goals. *The WebKit Open Source Project* [online]. Dostupné z www <http://www.webkit.org/projects/goals.html>.

[26] What is MathML? *W3C. Web Accessibility initative* [online]. Dostupné z www <http://www.w3.org/Math/>.

[27] Widget. *Co je to?* [online]. © 2010 – 2013. Dostupné z www <http://cojeto.superia.cz/software/widget.php>.

[28] XMLHttpRequest. *Mozilla Developer Network* [online]. © 2005 – 2014. Dostupné z www <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>.

# SEZNAM TABULEK

# SEZNAM OBRÁZKŮ