Obsah

Použité obrázky

1.Úvod

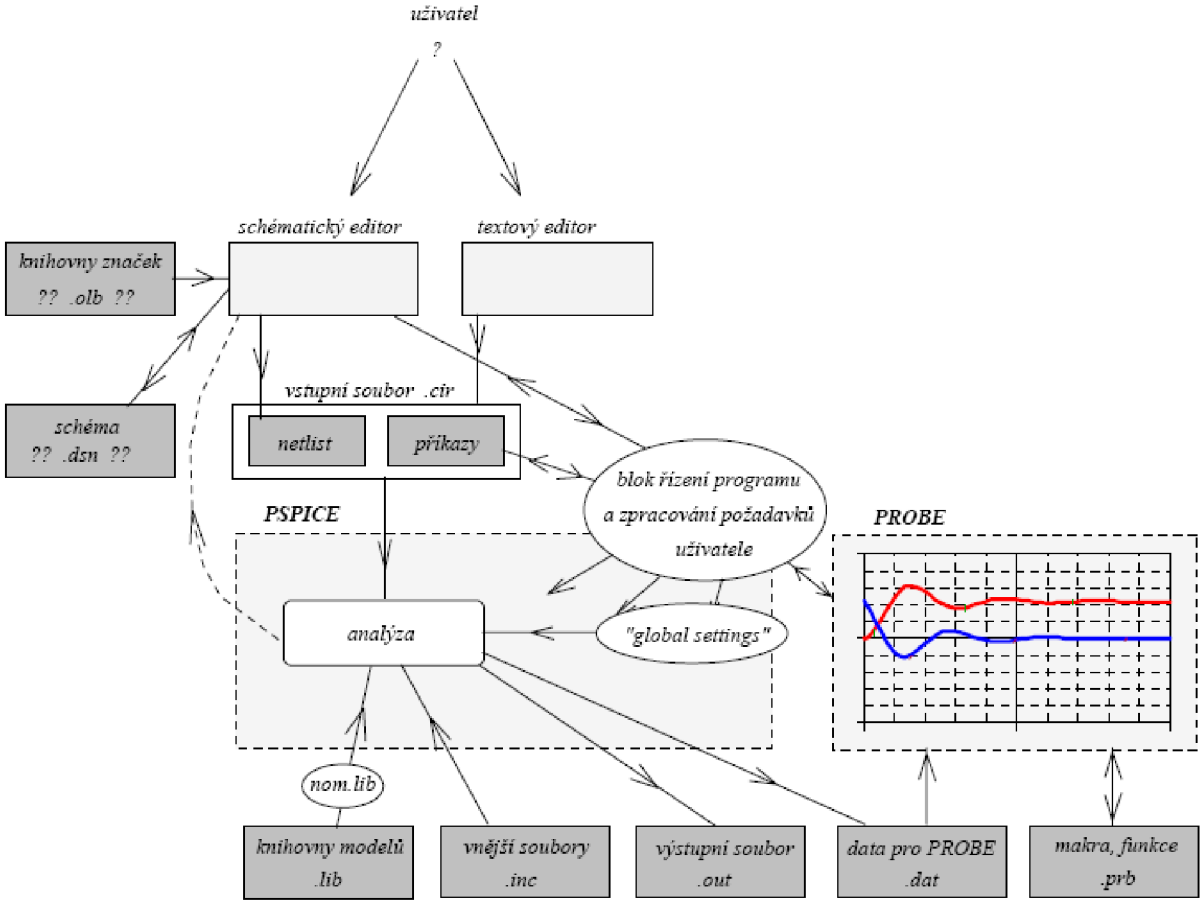
Cílem práce je vytvořit uživatelské rozhraní pro konverzi obvodového zapojení zapsaného ve formátu SPICE netlist do odpovídajícího grafického znázornění. Před úplným začátkem jsem se musel seznámit s vlastním programem na tvorbu netlistu a k čemu vlastně slouží. Samotný netlist je možné vytvořit v libovolném textovém editoru jako je např. PSPad, Poznámkový blok ve Windows. Původ netlistu je v programu Orcad PSpice, kde slouží pro zápis elektrického zapojení obvodu, který se simuluje. Netlist je možné vytvořit i pomocí programu Orcad Capture. Program Orcad PSpice Capture slouží k nákresu schéma obvodu. Program Orcad Capture má mnoho možností a jednou z nich je generování netlistu, který lze dále upravovat dle vlastní potřeby v textovém režimu. Netlist je možné generovat ze schématu. Všechny aplikace se zabývají tvorbou netlistu ze schématu, ale opačným postupem se nikdo nezabývá.

2. Řešená problematika

Uživatelské rozhraní pro konverzi obvodového zapojení zapsaného ve formátu SPICE netlist do odpovídajícího grafického znázornění je velice zajímavé a rovněž náročné na realizaci, protože kdyby šlo o rutinní problematiku, tak už by společnost Orcad nebo některá jiná společnost zabývající se touto problematikou vytvořila kompilátor a dodávala jej v kompletním balíku na zpracování a simulaci elektrických obvodů.

2.1 Historie netlistu

Netlist je jednoduchý textový zápis libovolného elektrického obvodu. Nejprve se podíváme na to, jakým způsobem vznikal jazyk SPICE a zároveň netlist. Textový zápis netlist vznikl jako součást simulačního prostředí SPICE. Výraz SPICE vznikl ze složení prvních písmen názvu simulačního prostředí „Simulation Program Integrated Circuit Emphasis“. Celý systém SPICE vytvořil student Larry Nagel na University of California, Berkeley, USA v době, kdy celá výpočetní technika byla v zárodku a psal se rok 1971. Program již tehdy umožňoval analýzu dějů v obvodech, kde se specifikoval převážně na bipolární a unipolární tranzistory. Systém v té době byl velmi věrohodný, protože se opíral o model bipolárního tranzistoru Gummel-Poon a tranzistoru typu MOSFET a JFET Shichman-Hodgesův. Systém se opíral o makromodely součástek, díky kterým bylo možné celý systém neustále doplňovat a rozšiřovat. Program SPICE byl volně šiřitelný a brzy se stal standardním simulačním prostředím v elektrotechnické praxi. Celý systém nenechal dlouho čekat na své zdokonalení, které přišlo již v roce 1975 a mělo značení SPICE2. Do roku 1983 vznikl ještě standard s označením SPICE2G.6, který je zachován až dodnes. Rok 1983 nebyl jen rok zavedením nového standardu, ale i zpřístupněním zdrojových kódů, které byly zapsány v programovacím jazyce Fortran. V tomto období rovněž dochází k rozšíření Unixových pracovních stanic a program zapsaný v jazyce Fortran přestává být vyhovující. Univerzita v Berkeley vydává rozhodnutí o přepsání celého programu do jazyka C. Tento počin dal za vznik verzi SPICE3 se standardem SPICE 3F.2. S dalším rozšiřováním výpočetní techniky a vzniku architektury PC došlo k nutnosti spouštět simulační prostředí SPICE3 i zde. Z tohoto důvodu se celý systém opět přepracoval a dal za vznik standardu PSPICE. Standardu se chopila společnost Orcad a vytvořila kompletní balík pro zpracování elektronických obvodů. Lepší znázornění systému PSpice je zobrazeno na obrázku Obr.1.

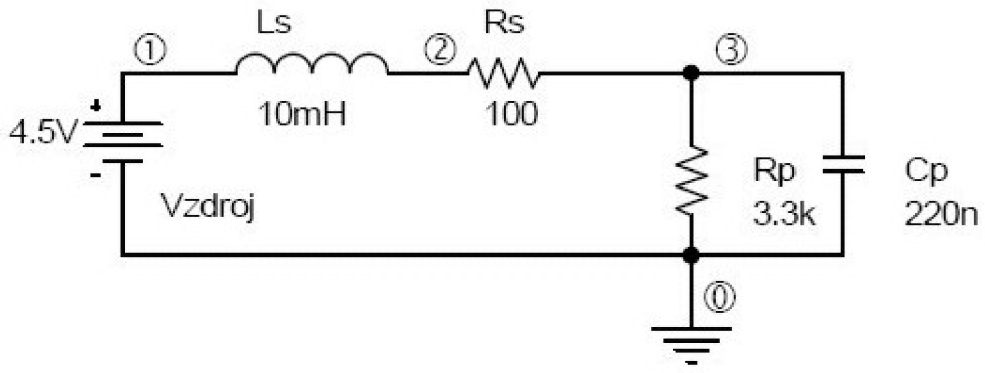


Obr. 1: Zjednodušená bloková struktura PSPICE [1]

Pro netlist byl zaveden soubor se speciální přílohou .cir, který je zkratkou výrazu circuit file. Soubor .cir je obyčejný textový záznam, a proto je možné jej použít v libovolném systému, který ho dokáže zpracovat. Nemusíme se tedy omezovat jen na systémy od společnosti Orcad.

2.2. Popis netlistu

Netlist je systematický textový zápis bez formátovacích značek. Protože je netlist prostý text, je možné jej vytvářet v libovolném textovém editoru jako je např. ve MS Windows PSPad, Poznámkový blok nebo systémech na bázi Linuxu, kde je např. KWrite. Pro tvorbu se však většinou používá program Orcad PSpice, kde je jednoduché textové rozhraní, ze kterého se vychází při vlastní simulaci obvodu. Spíše než na popis prostředí se zaměřím na popis jazyka Spice, ve kterém je napsán každý netlist. Vlastní netlist lze rozdělit do tří základních bloků. První část je první řádek, kam se zapisuje název simulovaného obvodu nebo libovolný text. Druhá část definuje vlastní zapojení obvodu, kde se udává jednotlivé rozmístění součástek a jejich parametry. V závěrečné části se uvádějí jednotlivé příkazy pro řízení simulace a každý správně zakončený netlist má na závěr zapsáno „.END“. Pro lepší představu zápisu uvádím malý příklad na RLC obvod.



Obr. 2: Schéma zapojení RLC obvodu [1]

RLC obvod	Název obvodu
Vzdroj 1 0 4.5V	
Ls 1 2 10mH	
Rs 2 3 100	Popis obvodu
Rp 3 0 3.3k	
Cp 3 0 220nF	
.OP	
.TRAN 1u 1m SKIPBP	Příkazy pro simulaci
.PROBE V(Cp) V(Ls)	
.end	Zakončení

Příklad zápisu netlistu zapojení k Obr. 2

Pro grafické vyjádření elektrického obvodu je důležitá hlavně část „Popis obvodu“. Všechny součástky jsou definovány názvem, místem zapojení(uzly) a příslušnými vlastnostmi. Jazyk Spice nerozlišuje malá a velká písmena. Nerozlišuje to v názvu ani ve velikostech příslušných veličin. Jednotlivé parametry prvků se od sebe oddělují mezerou, která může být i vícenásobná. Nikdy se však nesmí objevit v názvu prvku, ani ve velikosti veličiny. Jestliže se objeví v názvu prvku, uzlu nebo veličině mezera, kompilační jádro systému chápe text za mezerou jako další parametr objektu. Může tak dojít k chybnému definovaní zapojení nebo jen nastavení dalšího parametru, který systém nezná. Pokud se v zápisu vyskytne prázdný řádek, je ignorován při překladu. Každý řádek, kromě prvního a prázdného, musí začínat takto:

Klíčovým znakem určeným pro identifikaci obvodového prvku, např. R pro rezistor.

„*“ Obsah řádku je poznámka

„+“ Pokračovaní textu z předchozího řádku(pro lepší čitelnost v případě dlouhých zápisů)

„.“ Tečkou začínající dohodnuté příkazy např. .END

Jestliže umístíme kdekoliv v textu „;“ je vše za ním chápáno jako poznámka.

2.2.1 Název prvku

Názvy všech prvků začínají klíčovými znaky (první znak v názvu prvku) jednotlivých analogových součástek. Všechny názvy musí být jedinečné a nelze např. nazvat dva zdroje stejně, i když mají shodnou velikost napětí.

Výpis nejběžnějších analogových součástek:

R .. rezistor
C .. kapacitor
L .. induktor
K .. přiřazení magnetické vazby mezi induktory nebo mag. jádra induktoru (induktorům)
V .. nezávislý zdroj napětí ..
I .. nezávislý zdroj proudu
E .. zdroj napětí řízený napětím
F.. zdroj proudu řízený proudem
G.. zdroj proudu řízený napětím
H.. zdroj napětí řízený proudem
B .. univerzální nelineární zdroj
D .. dioda
J .. tranzistor JFET
M .. tranzistor MOSFET
Q .. bipolární tranzistor
Z .. IGBT tranzistor
S .. spínač řízený napětím
W .. spínač řízený proudem
T .. vedení
X .. podobvod (subobvod, funkční blok, makro)
N .. Číslicový vstup
O .. Číslicový výstup

2.2.2 Uzly

Všechny prvky jsou zapojeny do uzlů. Všechny obvody mají nulový uzel, který většinou znázorňuje záporný pól napájení, neboli zem. Názvy jednotlivých uzlů se mohou

skládat z číslic nebo slov. Žádný název uzlu nesmí obsahovat znak mezera. Název uzlu není velikostně omezen, ale ve většině případů se používá číselné označení. Číselné označení začíná uzlem „0“ a musí být vždy pro označení země použita nula, protože při simulaci by docházelo ke konfliktům. Do každého uzlu musí být zapojeny nejméně dva prvky. Pokud tomu tak není, dojde ke konfliktu při simulaci a systém zahlásí chybu.

2.2.3 Veličiny

Všechny zadávané veličiny se mohou definovat několika způsoby. První varianta je definování pomocí přirozených čísel v klasickém tvaru, tj. pomocí desetinné tečky. Všude v hodnotě musí být desetinná tečka, nesmí být čárka. Další varianta je pomocí exponenciálního tvaru, kde se v zápisu uvede malé nebo velké „e“ a zápis vypadá takto: 34E-12, 23e6. Oba zápisy překladač přeloží.
Hodnoty lze zadávat i pomocí inženýrské notace:

Název	**Symbol**
Giga	G,g
Mega	Meg,meg
Kilo	K,k
mili	M,m
mikro	U,u
nano	N,n
piko	P,p

Velikost písmen a text uvedený za symbolem velikosti jsou nepodstatné. Veličina může být zadávána i pomocí parametru, který se symbolizuje pomocí složených závorek a názvu parametru. Zápis parametru se definuje takto: {R1}. V názvu může být libovolné uspořádání číslic a písmen, jen se opět nesmí vyskytovat znak mezera. Každý parametr lze použít u několika prvků a počet jednotlivých parametrů není omezen. K vlastnímu definování parametrů se vyjádřím v následující podkapitole.

2.2.4 Příkazy

Jazyk Spice má příkazy, které začínají tečkou a slouží pro definování simulace a následné zpracování dat ze simulace do grafu, tabulky nebo výstupního souboru. Uvedeny jsou zde i příkazy pro větvení a vkládání souborů do netlistu.

Popis jednotlivých příkazů:

.AC – střídavá analýza obvodu pro frekvenční analýzu malého signálu. Přiřazení frekvence všem střídavým nezávislým zdrojům.

.DC – stejnosměrná analýza obvodu pro výpočty stejnosměrných charakteristik a přiřazovaní posloupnosti napětí nebo proudu stejnosměrným nezávislým zdrojům.

.DISTIBUTION – uživatelem definované rozdělení

.FOUR – Fourierův rozvoj časových signálů z přechodové analýzy (musí se používat současně s přechodovou analýzou - TRAN). Do výstupního souboru se vypíší amplitudy a fáze prvních osmi harmonických.

.FUNC – systém jazyka PSpice umožňuje vlastní vytváření funkcí pro výpočty potřebné pro simulaci.

.INC – příkaz slouží pro včlenění dalšího souboru do stávajícího netlistu(inc "C:\schemata\text.txt"). Vnořené soubory nemusí být jen .txt, ale mohou se použít opět libovolné textové soubory bez formátovacích značek. Při použití příkazu .inc vniká stejný efekt, jako kdyby na místě .inc byl obsah včleněného souboru. Soubory se mohou vnořovat až do čtvrté úrovně. Vnořený soubor nesmí obsahovat hlavičku.

.LIB – tento příkaz slouží k načtení požadované knihovny, pro definici jednotlivých prvků, protože všechny prvky jsou nadefinovány v knihovnách. Příkaz .lib lze použít samostatně, nebo lze za něj definovat cestu i název požadované knihovny (.lib "C:\schemata\pokus.lib"). Pokud je příkaz .lib zapsán samostatně, dojde k načtení všech nainstalovaných knihoven.

.MC – analýza metodou Monte Carlo. Odhad chování obvodu s uvážením tolerancí parametrů modelů prvků. Výpis tabulky výstupní proměnné setříděné podle odchylky od nominální hodnoty. Používá se spolu s .DC nebo .AC nebo .TRAN. Možno nahradit analýzou .WCASE; v jednom vstupním souboru lze použít pouze jednu z obou analýz.

.MODEL – tímto příkazem lze definovat libovolný prvek. Při definování prvků se udává konkrétní typ prvku v knihovně. Pokud je potřeba definovat vlastní typ, nebo prvek není v knihovnách, stačí místo názvu prvku uvést název modelu a definovat si vlastní model. Definice modelu ledky může být zapsána takto:
.model ledka D (N=1.7 RS=.7 CJO=23.9P

+ IS=85.3p BV=6 IBV=10U VJ=0.75 TT=4.32U).

.NODESET – odhad pracovního bodu. Počáteční odhad klidového pracovního bodu. Nastavení obvodu o více stabilních stavech do daného pracovního bodu.

.NOISE – šumová analýza. Výpočet šumu obvodu na výstupu a ekvivalentního šumu na vstupu. Šumová analýza se musí používat současně se střídavou analýzou (viz .AC).
Výpis pomocí proměnných ONOISE a INOISE, resp. DB(ONOISE) a DB(INOISE) v příkazech .PLOT a .PRINT.

.OP – příkaz slouží pro výpočet klidového pracovního bodu obvodu. Výpis uzlových napětí, proudů zdrojů, příkonu obvodu, linearizovaných parametrů nelineárních zdrojů a prvků do výstupního souboru. Linearizované parametry obvodu se ukládají pro analýzy .TF, .SENS, .AC a .NOISE (viz též .TRAN/OP)

.OPTION – předvolby a parametry simulátoru, např. pro řízení tvaru výpisů, nastavení nominálních parametrů obvodu, nastavení parametrů konvergence a přesnosti výpočtu.

.PARAM – zápis příkazu slouží definování všech parametrů v obvodu. Za .param se zapíše název parametru, rovnítko a velikost veličiny, čárka pak definuje další parametr. Celý zápis může vypadat takto: .param R1=10k Cfiltrovaci=10u;.

.PLOT – výpis výsledků do výstupního souboru ve tvaru grafů tvořených mozaikou znaků.

.PROBE – používá se chceme-li dosáhnout přesných výsledků ve vysokém rozlišení. Vytvoří se soubor PROBE.DAT s daty pro grafický post-procesor PROBE. Příkaz .PROBE/CSDF nevytvoří soubor PROBE.DAT, nýbrž soubor PROBE.TXT, který obsahuje stejné položky dat, avšak v textovém formátu.

.PRINT – výpis tabulek

.SENS – citlivostní analýza slouží pro určení citlivosti obvodových veličin na změny parametrů linearizací obvodu v pracovním bodu výpočtem parciálních derivací zvolených proměnných podle všech parametrů všech prvků. Výpis tabulek do výstupního souboru.

.STEP – Krokování analýzy. Při krokování se mění jedna proměnná pro všechny analýzy

obvodu.

.SUBCKT – jazyk Spice umožňuje definování podobvodu pomocí příkazu .subckt. Podobvod je relativně samostatný obvod, který komunikuje s okolím pomocí konkrétních vývodů. Podobvod může být nadefinovaný v knihovně nebo v aktuálním netlistu. Podobvod může být sestavený ze součástek, jejichž parametry můžeme měnit při volání. Každý podobvod začíná názvem, seznamem vývodů a případnými parametry. Všechny podobvody jsou zakončeny příkazem .ENDS. Podobvody se po správné definici volají jako kterýkoliv jiný prvek.

.TF – přenosová funkce slouží k výpočtu přenosu vstupního a výstupního odporu pro malé signály linearizací obvodu v pracovním bodu. Výpis do výstupního souboru.

.TRAN – přechodová analýza, výpočet časové odezvy obvodu na vstupní signál. Přiřazení času nezávislým zdrojům s definovaným časovým průběhem signálu.

.TEMP – umožňuje nastavit v jakých teplotách bude systém simulovat zadaný obvod a určí se chování obvodu v zadaných teplotách.

.WCASE – počítá maximální odchylku proměnné od nominální hodnoty danou tolerancí parametrů v příkazech .MODEL. Příkaz .WCASE lze nahradit analýzou .MC nebo .SENS. V jednom vstupním souboru nelze použít současně analýzy .MC a .WCASE.

.WIDTH – šířka výpisu do výstupního souboru.

2.3 Programování

Programovat konverzi jsem mohl ve dvou odlišných jazycích. Mohl jsem programovat v Delphi nebo v C#. Jazyk Delphi už používám několik let a s jazykem C# jsem začínal v roce 2006. Pokud bych program psal v Delphi, tak budu používat Delphi 5, které už mám velice dobře zažité a pracuji v něm rád. V jazyce C# se však programuje mnohem efektivněji a prostředí Microsoft Visual C# 2005 Express Edition, které je naprosto odlišné od Delphi 5, ale má mnoho nových možností a v edici Express je volně ke stažení přímo u společnosti Microsoft.

2.3.1 Programovací jazyk C#

Programovací jazyk C# vychází s platformy .NET oficiálně představený společností Microsoft v roce 2000 jako klíčový produkt. Microsoft .NET znamená novou generaci

systému vývoje aplikací pro operační systémy Windows založené na řízeném běhovém prostředí nesoucí jméno .NET framework. Hlavní důvody vzniku platformy byly:

- nekompatibilita jednotlivých programovacích jazyků a s tím související obtížná spolupráce mezi programy/knihovnami napsanými v odlišných jazycích (např. C++ a Visual Basic)
- vysoká chybovost aplikací (chyby v práci s pamětí, neplatné konverze datových typů)
- problémy s verzemi knihoven (obtížná práce s provozem více verzí knihoven)
- zastaralý a nepřehledný způsob vývoje dosavadních webových aplikací

Všechny uvedené problémy řeší platforma .NET za použití řízeného běhového prostředí assemblies, které je základním stavebním prvkem aplikací a technologií ASP .NET pro vývoj webových aplikací.

Řízené běhové prostředí, ve kterém spočívá hlavní a nejdůležitější rozdíl od ostatních vývojových jazyků C++, Visual Basic nebo Delphi, znamená, že vlastní kompilace zdrojového kódu aplikace se kompiluje až při spuštění aplikace. Kompilace tímto způsobem umožňuje lepší přenositelnost mezi jednotlivými platformami, což je od ostatních jazyků obrovský pokrok.

Klíčová vlastnost mezikódu se nazývá MSIL, tedy Microsoft Intermediate Language. Tento jazyk relativních adres je spouštěn klíčovou součástí .NET frameworku pojmenovanou CLR (Common Language Runtime neboli společné běhové prostředí) a společnost Microsoft jej dala ke standardizaci organizaci ECMA.

V prostředí CLR je věc usnadňující práci s operační pamětí – Garbage Collector. Jde o sadu složitých algoritmů pro uvolňování nepotřebných programových objektů z paměti. Díky této novince se programátoři nemusí starat o přiřazování nebo uvolňování operační paměti a odpadá riziko špatné práce s operační pamětí, která ve většině případů končí pádem aplikace. Důležitá vlastnost je CLS – Common Language Specification a s tím související Common Type systém. Výsledkem použití CLS a CTS je rovnocennost programovacích jazyků. Jinak řečeno – pro vývoj .NET aplikací je možné použít jeden z několika programovacích jazyků vyšší úrovně. Může se jednat například o tyto:

- C#, nový jazyk vyvinutý pro .NET
- Visual Basic .NET, nová generace jazyka Visual Basic
- J#, jazyk se syntaxí Java
- managed C++

Systém jazyka umožňuje výrobcům třetích stran vyvíjet další jazyky, které musí mít

kompilátor se schopností vytvářet kódy v mezijazyku MSIL.

Platforma Microsoft .NET umožňuje vyvíjet širokou škálu programů. Začít se může u klasických konzolových aplikací, které pro vstup a výstup používají příkazový řádek. Nejvíce rozšířené aplikace jsou s využitím knihoven Windows.Forms, intenzivně využívající Microsoft Win32 API. Výsledkem jejich použití jsou známé formulářové aplikace pro Windows. Možné je také vytvářet aplikaci běžící jako proces na pozadí systému – služby Windows.

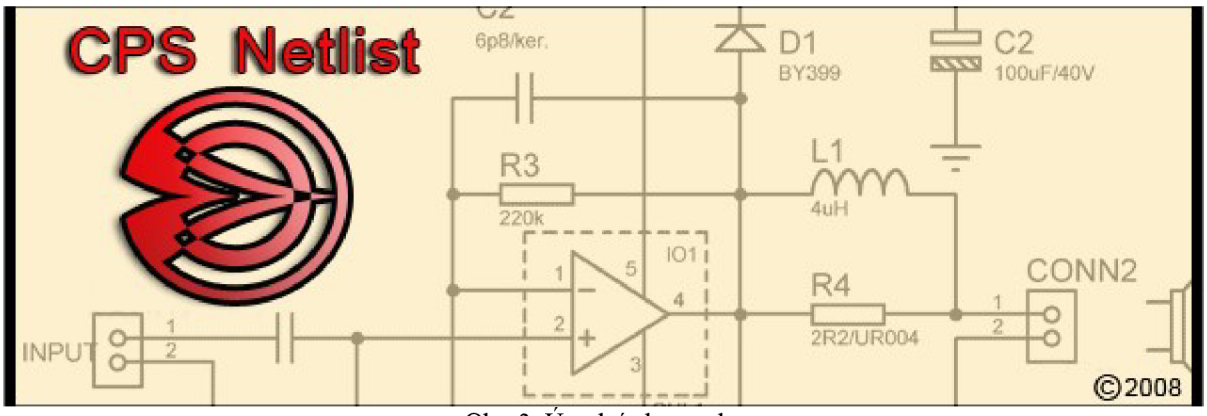
Velkým odvětvím jsou webové aplikace nahrazující zastaralé ASP 2.0, které se nově označují ASP .NET. S ASP .NET souvisí i takzvané Webové služby, které se začínají ve velkém rozvíjet a mnoho firem je používá pro své webové prezentace a informační podnikové systémy pro své zaměstnance. V této oblasti platí, že kvalitní informační systém ušetří až 50% nákladů ve firmě. Všechny webové služby vytvořené na platformě ASP .NET jsou velmi spolehlivé a rychlé. Informační systémy se často propojují s webovými prezentacemi firem a zajišťují tak aktuálnost zobrazených informací.

Systémy od společnosti Microsoft jsou dodávány jako velké instalační balíky a tím umožňují programátorům vycházet z již připravených funkcí, knihoven a celé řady objektů, které usnadní práci. Díky těmto aspektům se programuje mnohem efektivněji a rychleji než v jiných jazycích. Tím se šetří konečné náklady na vývoj požadované aplikace.

2.3.2. Program

Prostředí pro převod netlistu do odpovídajícího grafického znázornění jsem nazval CPS Netlist. Systém se původně měl jmenovat Převod PSpice, ale tento název byl moc dlouhý a nevhodný. Proto jsem raději zauvažoval o slovech kompilace, PSpice, Netlist. Jelikož všechny slova jsou z angličtiny, použil jsem anglický překlad slova kompilace na compilation. Při složení slov Compilation PSpice Netlist vznikne nádherná zkratka CPS Netlist.

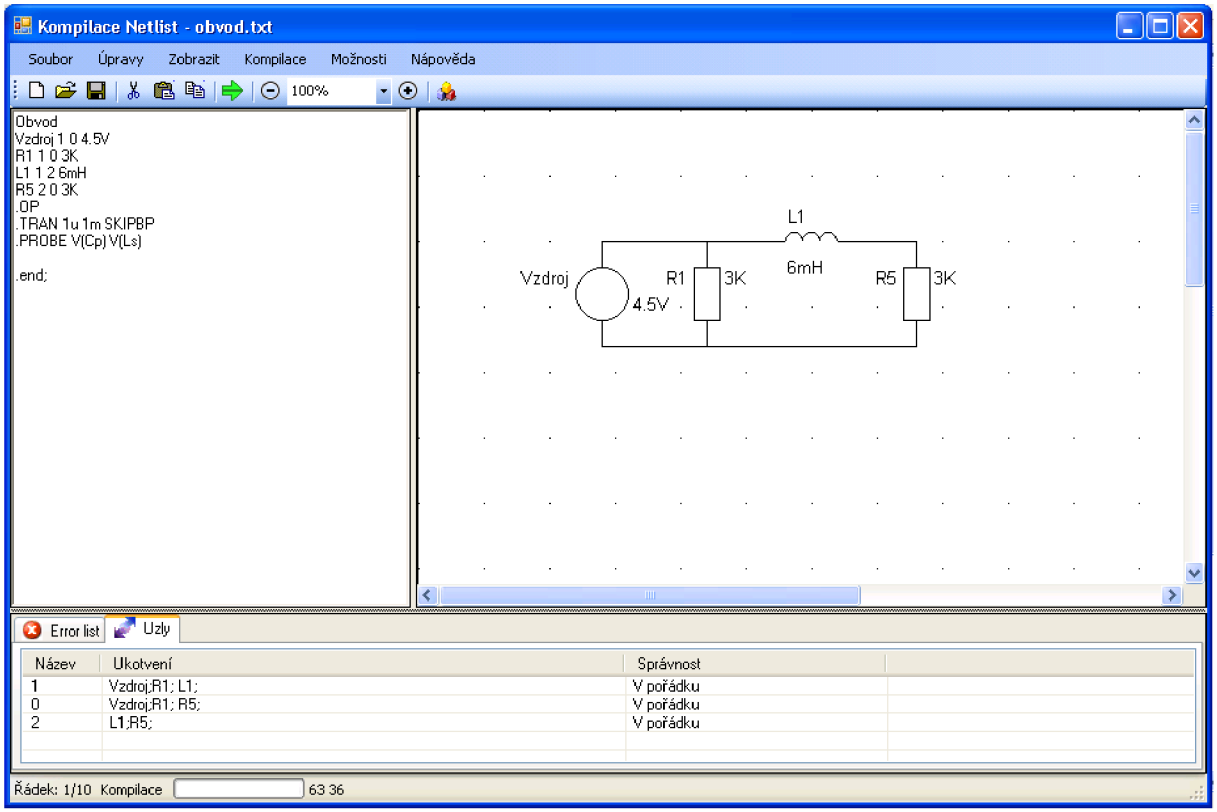
Logo programu vzniklo z výchozí myšlenky pro rozmísťování součástek na plátně. Jednotlivé kruhy symbolizují smyčky zapojení, které se transformují na požadovaný směr generovaného schématu. Úvodní obrazovku programu můžete vidět na Obr.3.



Obr. 3: Úvodní obrazovka

2.3.3 Vyvinuté prostředí

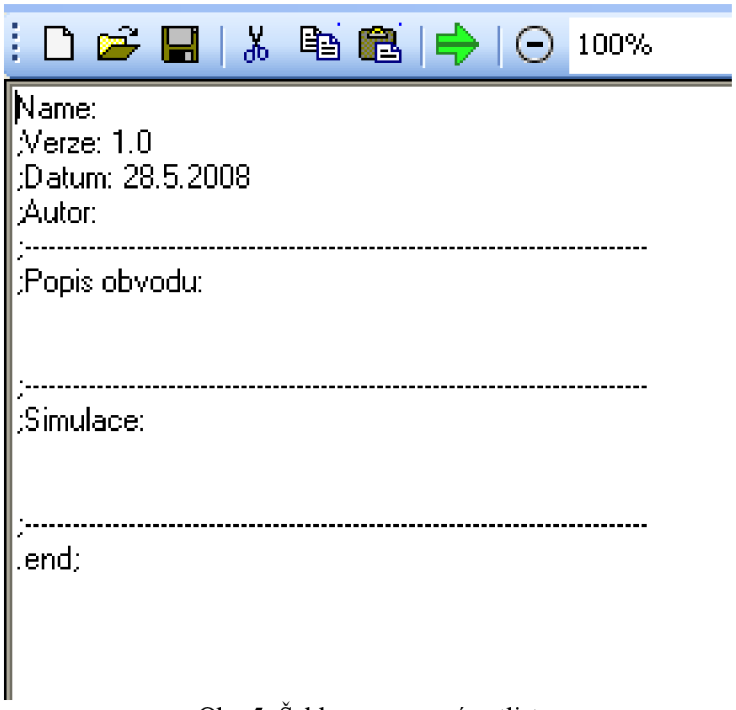
Program pro převod netlistu do grafické reprezentace jsem se pokusil vytvořit velice jednoduše a docílit tak intuitivního ovládání celého programu. Pro maximální přehlednost pracovní plochy vznikly tři základní části. Tyto tři části jsem pojmenoval jako Netlist, Plátno, Infopanel. Na obrázku Obr. 4, zobrazeném níže, se můžete přesvědčit o jejich rozmístění. Při vývoji prostředí jsem se hodně inspiroval samotným Visual Studiem v oblasti rozmístění některých prvků (Infopanel, Platno), klávesovými zkratkami pro kompilaci (F5), klávesovou zkratkou pro kontrolu netlistu (F6), kontextovou nápovědou pro psaní netlistu. Prostředí je rozděleno pomocí dvou komponent SplitContainer, díky kterým si uživatel může nastavit velikosti jednotlivých ploch. Základní ovladatelnost celého prostředí je zajištěno rolovacím menu MenuStrip a panelem nástrojů s použitím komponenty ToolStrip s jednotlivými tlačítky. Všechny parametry chování aplikace je možné nastavit v možnostech, nebo v roletě Zobrazit.



Obr. 4: Celkový pohled na systém

2.3.3.1 Zadávání netlistu

Zadávání netlistu v levé části pracovní plochy je obsluhováno komponentou RichTextBox. Její umístění do levé části mi přišlo nejpřirozenější. Komponenta je přímo předurčená pro zpracování jakýchkoliv textů. Umí zpracovávat obyčejný text i formát souborů rtf, doc. Pro zpracování netlistu jsem musel nastavit otevírání, ukládání textu v modu RichTextBoxStreamType.PlainText. Pokud bych tak neučinil, program by ukládal texty s formátovacími značkami a netlist by byl pro další použití znehodnocen. Při vytvoření nového netlistu se automaticky nachystá šablona pro úhlednější zápis netlistu. Pokud by s touto možností zadávající nesouhlasil, může ji zakázat v nastavení. Šablona pro zápis netlistu vypadá takto:

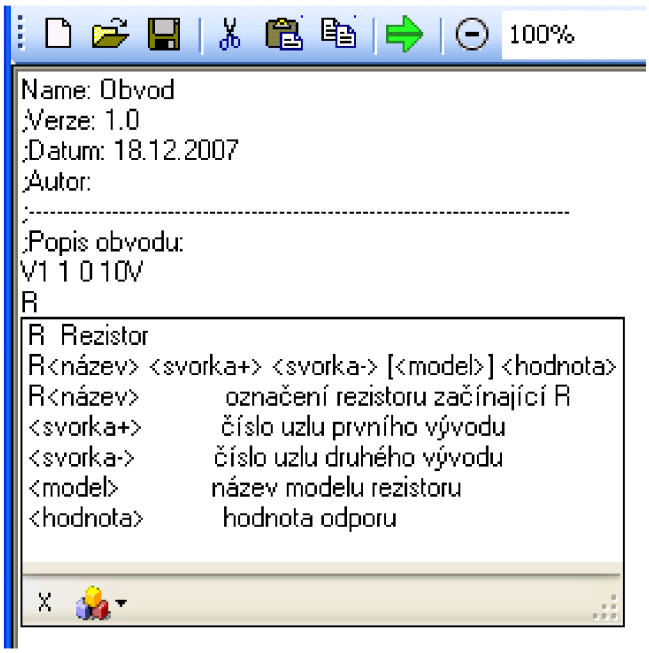


Obr. 5: Šablona pro nový netlist

Prostředí pro zadávání netlistu jsem doplnil o kontextovou nápovědu (Obr. 4) pro uživatele méně znalé zápisu prvků v jazyce PSpice. Kontextová nápověda se automaticky vyvolá po zápisu prvního symbolu prvku, nebo při stisku kombinace kláves ctrl + mezerník se vyvolá obecná nápověda se základními popisky všech prvků. V nastavení je možné definovat chování kontextové nápovědy. Zda se má nápověda zobrazovat vždy jen při stisku kláves ctr + mezerník, nebo se nemá zobrazovat vůbec. Pro kontextovou nápovědu jsem použil komponenty Panel, ListBox, ToolStrip. Tyto komponenty jsou umístěny na panel1, kterému se nastavuje visible true nebo false. Vyvolání nápovědy je obsluhováno v údalosti richTextBox1

onKeyDown, kde se provádí analýza stistků kláves. Po nalezení příslušného symbolu dojde k výpisu do listBox1 a zobrazení celého panel1.

Samozřejmostí pro zadávání netlistu jsou klasické funkce pro práci s textem jako jsou Vyjmout, Vložit, Kopírovat, Krok zpět a Obnovit. Otevírání souborů je doplněno o historii otevřených souborů pro rychlejší práci.

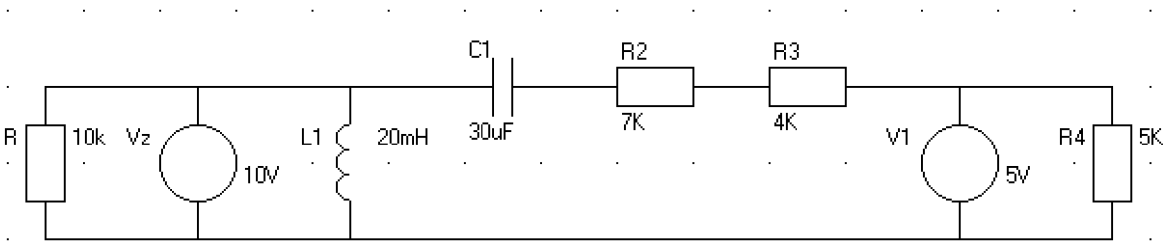


Obr. 6: Kontextová nápověda

2.3.3.2 Kreslící plátno

Kreslící plátno je jedna z nejdůležitějších částí celého systému. Plátno je určeno k vykreslování generovaného schématu. Pro kreslení schématu jsem použil komponentu Panel. Celé vykreslování obsluhuje událost OnPaint(), která přistupuje k vlastnímu plátnu komponenty pomocí proměnné e typu PaintEventArgs a pak dále pomocí objektu Graphitcs. V této události se obsluhuje celé vykreslování. Po vykreslení v události OnPaint() se musí zavolat metoda panel1.Invalidate(), přes kterou se kreslený objekt zobrazí. Vykreslení se musí zavolat při každé změně na plátně. Při použití lupy se musí všechny objekty na plátně vykreslit znovu v novém poměru. Lupa má základní velikost 1 a při zvětšování dochází k přičítání hodnoty 0,1. Hodnota jedna by uživateli nic neprozradila, proto se všechny hodnoty násobí 100. Hodnota lupy je nadefinovaná v rozmezí od 0,1 do 5. Tyto hodnoty jsou nejoptimálnější pro vykreslování. Při spuštění se vykreslí rastr, který je možné nadefinovat v nastavení programu. Plátnu je možné nastavit barvu, rozteč bodů, tloušťku a barvu vykreslovaných součástek. Všechny tyto parametry se definují v nastavení aplikace.

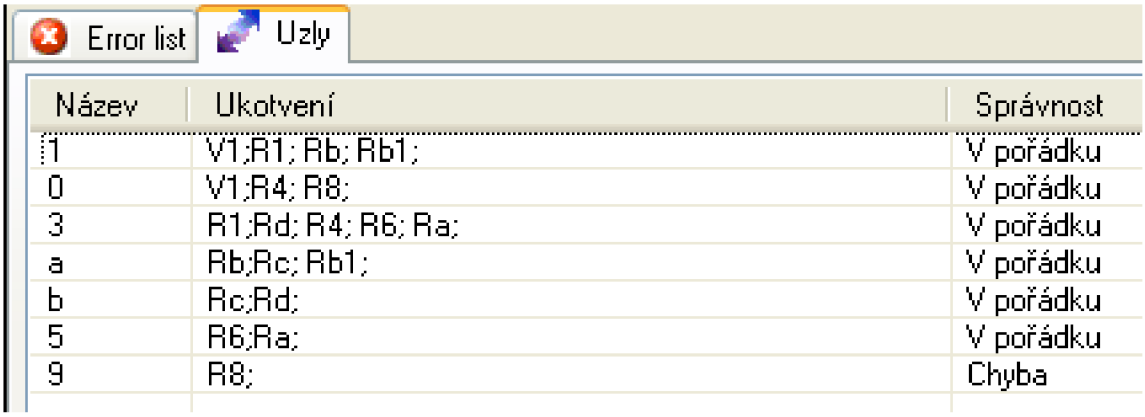
Vykreslení na plátno se provede vždy při kompletní kompilaci netlistu. Protože se plátno překresluje velmi často, je maximální snaha o optimalizaci vykreslování. Optimalizace spočívá hlavně ve vykreslování pouze viditelné časti vygenerovaného schématu. Ostatní optimalizace spočívají v maximálně možném snížení výpočtů při rozmísťování jednotlivých části schematických značek.



Obr. 7: Vygenerované schéma

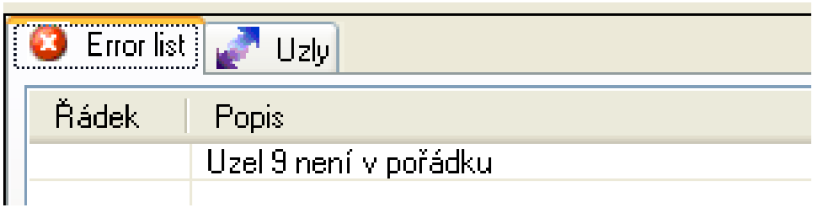
2.3.2.3 Infopanel

Panel slouží k informacím o kompilaci netlistu a následného rozmísťování na plátně. Infopanel má dvě záložky Error list a Uzly. Pro záložky se používá komponent TabControl, na nichž je možné vytvářet záložky. Na záložkách jsou umístěny komponenty ListView, které umožňují zobrazovat řádky a definovat jednotlivé sloupečky, do kterých se zapisují potřebné informace. ListView má mnoho možností, které lze využít pro mnoho funkcí. Celý Infopanel slouží k zobrazování informací o rozboru a kompilaci netlistu. Na záložce Uzly jsou vypsány všechny nalezené uzly se součástkami, které jsou do daných uzlů připojené. V posledním sloupečku těchto informací je řečeno, zda je uzel v pořádku.



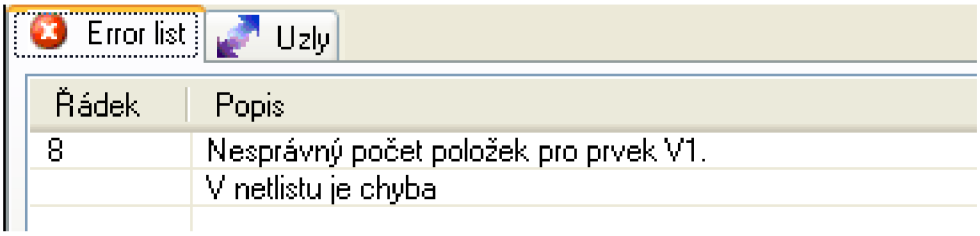
Obr. 8: Výpis správnosti uzlu

Pokud se zjistí, že uzel není vpořádku, dojde k vyvolání výjimky a zobrazí se v Error listu hlášení, na kterém je zapsáno, ve kterém uzlu je chyba.



Obr. 9: Chybové hlášení při chybě v uzlu.

List zobrazí kromě špatných uzlů i nedefinované hodnoty veličin daných prvků.

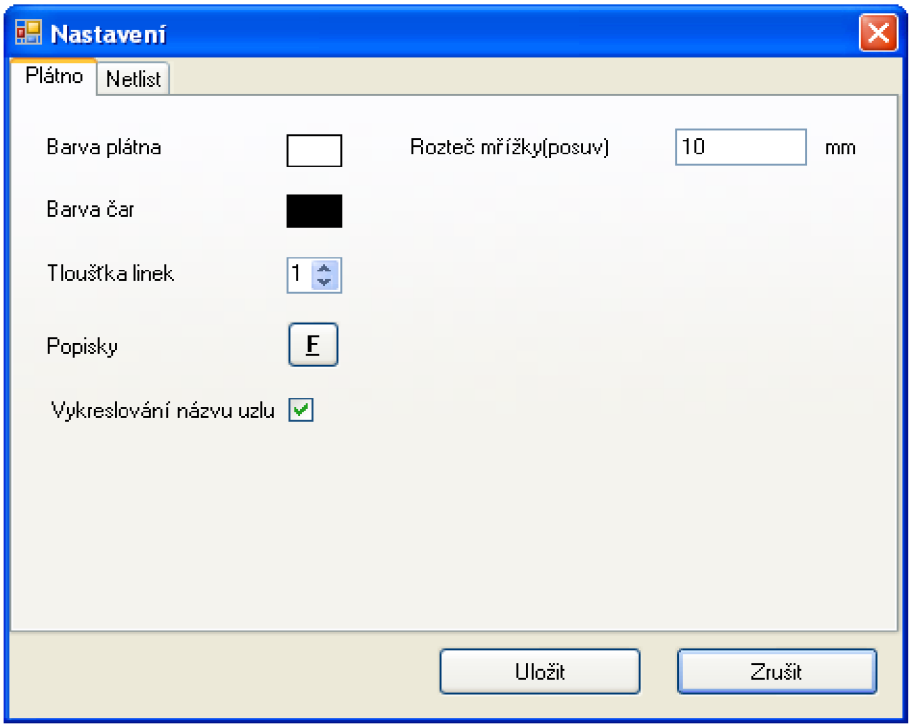


Obr. 10: Chybějící vlastnost prvku

V Error listu se rovněž zobrazí i chybové hlášení v případě zaznámenání chyby v rozmísťovacím cyklu. Systém vypíše chybové hlášení a rozmísťovací algoritmus se zastaví. Error list slouží ke kompletnímu vypisu chyb nebo nalezených výjimek.

2.3.2.4 Nastavení

V každé aplikaci je možnost nadefinovat si nastavení některých hodnot podle potřeb uživatele. Tato možnost je i zde a najdeme ji v roletě zobrazit v položce Nastavení. Pro jednotlivé druhy nastavení jsem použil záložkovou komponentu TabControl, pro níž jsem nadefinoval dvě záložky. Na první záložce s názvem Plátno se definují jednotlivé položky pro vykreslování schématu(Obr. 6). Ve druhé záložce s názvem Netlist je definováno použití kontextového menu a použití šablony netlistu při jeho vytváření. Všechny parametry nastavení se zachovávají v Properties Settings, které je přímo určeno pro uložení těchto proměnných. Zde se definuje zda ukládaná proměnná bude typu bool, int, float, color, font atd. Tyto informace se ukládájí do pomocného souboru xml.



Obr. 11: Nastavení programu

2.3.3 Rozbor netlistu

Netlist je zapsán v komponentě richTextBox1, ze které se čerpají jednotlivé informace řádek po řádku. Pro tento rozbor jsem musel vytvořit třídu soucastky, ve které se evidují potřebné informace o jednotlivých prvcích. Všechny evidované prvky si pamatují v třídě List, která představuje takzvaný lineární seznam. Třída List zatěžuje mnohem méně pamět než klasický Array nebo ArrayList, kde je nutné hned po vytvoření alokovat potřebné množství paměti. O alokaci paměti se stará sama třída List. Propojenost jednotlivých prvků v listu obstarává třída sama a k jednotlivým objektům se přistupuje jako u jednorozměrného pole. Deklarace Listu pro mé potřeby vypadá takto: List<soucastky> netlist = new List<soucastky>; Plnění probíhá pomocí metody netlist.add(prvek), list je vybaven mnoha dalšími funkcemi jako je např. Exist, Remove, Find, Count a mnoho dalších. Pokud potřebuji přistoupit např. k položce Název u prvku číslo 6, provedu jednoduchý zápis netlist[6].Nazev = “R1”;. Textový netlist převádím podle popisu níže do Listu složený s objektů soucastky s názvem netlist. Objekt soucastky obsahuje všechny důležité informace o každém prvku ve schématu. Při vytváření jednotlivých proměnných jsem se snažil všechny pojmenovávat tak, aby jejich názvy byly vypovídající a nemusely být doplněny žádnou další dokumentací.

Objektu typu soucastky obsahuje následující proměné:

```
string _Nazev = null;
PointF _NazevPozice = new PointF();
RectangleF _PlochaPrvku = new RectangleF();
int _RotacePrvku = 0;
string _Velicina = null;
PointF _VelicinaPozice = new PointF();
List<string> _NazevUzly = new List<string>();
List<PointF> _Uzly = new List<PointF>();
string _Ostatni = null;
int _Priorita = 0;
int _Umisteni = 0;
int _VeSmycce = 0;
Object _Tag = new Object();
SolidBrush _Vypln = new SolidBrush(Properties.Settings.Default.bkcolor);
Pen _Pero = new Pen(Properties.Settings.Default.linecolor,
Properties.Settings.Default.tline);
SolidBrush _VyplnTextu = new SolidBrush(Properties.Settings.Default.linecolor);
```

Zdrojový kod 1: Objekt soucastka

Celý rozbor je řešen v cyklu for a místo klasického systému podmínek jsem použil systém switch, který je mnohem efektivnější a rychlejší při rozhodování. Pomocí cyklu for se načtou jednotlivé řádky s richTextBox1 do proměnné typu string a postupně pomocí metod pro práci se stringy se rozkládají do List<string> s názvem pole. Po rozložení celého řádku dojde k identifikaci druhu prvku a zavolání metody ze třídy součástky. Pro přiřazování prvků do Listu netlist typu soucastky. Pro předání údajů do Listu netlist jsou metody Pridej, ve kterých se přesně určí jednotlivé položky součástky. Metody jsou přesně přiřazeny pro jednotlivé prvky, aby se lépe definovaly syntaxtické chyby v textovém netlistu. Pro přiřazení se používají tyto metody:

```
public void PridejDvou(List<string> pole)
public void PridejTri(List<string> pole)
public void PridejCtyr(List<string> pole)
public void PridejX(List<string> pole)
```

Zdrojový kod 2: Metody na rozbor netlist

Metody jsou rozčleněny podle počtu vývodů. V dané metodě dojde k přepisu položek z Listu<string> pole do daných položek proměnné třídy. Po předání proměnných dojde k nastavení základních hodnot pro vykreslení prvku na plátně. Základní hodnoty pro vykreslení jsou PlochaPrvku, Rotace a Priorita. Pokud cyklus najde v textu “;” ,vše za ním považuje za

poznámku a nikam se text nepřiřazuje. Při nalezení symbolu “+”,”*” dojde k ignorování celého řádku.
Jestliže cyklus nalezne texty začinající symbolem “.” , je jeho zpracování trochu jiné. Pro rozbor netlistu jsou důležité výrazy .subckt, .ends, inc. Ostatní se ignorují.

Příkazy:
.subckt – dojde k inkrementaci proměnné, která určuje zanoření netlistu a při zanoření různém od nuly dochází k přeskakování dalších řádků, protože pro vlastní rozbor textového netlistu jsou nepotřebné a nikdy se tyto informace nevypisují.

.ends – zde dojde k dekrementaci proměnné pro zanořovaní a opětovnému načítaní řádku pro rozbor.

.inc – systém při nalezení tohoto příkazu zkontroluje existenci přislušného souboru. Nalezený soubor se přímo vloží za příkaz .inc a za cestu vloží “;#”. Tímto symbolem se při opětovné kompilaci pozná, že soubor byl přidán do základního netlistu. Tímto způsobem zle přidávat soubory do nekonečna. Pokud se soubor nenachází na uvedeném místě, systém zobrazí hlášení o nenalezení souboru a zastaví kompilaci.

Systém vyhodnocuje pouze prvky, které začínají písmeny, která zná. Po nalezení písmene, které nezná, vyhodí systém chybové hlášení o nalezení neznámého prvku a do Error listu zapíše, na kterém řádku je symbol nalezen. Chybové hlášení se rovněž zobrazí při špatně definovaném prvku. Špatně definovaný prvek může být opomenutí veličiny nebo nezapsání některého vývodu.
Závěrečný cyklus rozboru kontroluje správnost jednotlivých uzlů a vytváří výpis do Infopanelu s uzly. Kontroluje, zda má každý uzel minimálně dva zapojené prvky. Pokud tomu tak není, dojde k zápisu do Error listu a k zastavení kompilace.
Pokud celý rozbor proběhne v pořádku, systém přejde k vlastnímu rozmísťování jednotlivých prvků.

2.3.4 Teorie rozmístění prvků

Algoritmus rozmístění jednotlivých prvků je nejnáročnější část celé aplikace. Vytvořil jsem několik teorií o rozmísťování prvků.

Původní teorie vycházet z nulového uzlu, který by se považoval za základ celého schématu, byla zavrhnuta. Podle dané teorie by součástky romísťovala nesmyslně vedle sebe a docházelo by k velkému zabrání plochy pro vykreslení. Většina prvků by byla umístěna

vertikálně a celé schéma by bylo nepraktické a nepřehledné.

Druhá teorie, považovat rovněž nulový uzel jako výchozí, ke kterému by byl připojen zdroj, byla rovněž zavrhnuta, protože zdrojů může být několik a rozmístění schématu by nebylo dokonalejší než v první úvaze.

Třetí teorie o rozmísťování součástek pomocí smyček již byla daleko úspěšnější než ty předešlé. Tento systém by již umožňoval rozmísťovat součástky systematicky vedle sebe a vytvářet rozumně vypadající schéma, které by nezabíralo velké množství prostoru. Tato teorie ale částečně ztroskotala na tom, jak poznat správné natočení jednotlivých součástek.
Teorie o smyčkách byla doplněna o prioritu součástek. Kde každý typ součástky dostane prioritu. Nejnižší prioritu budou mít prvky se dvěma vývody. Do této skupiny patří R, L , C, V, I atd. Druhou skupinou budou prvky se třemi vývody, které budou mít prioritu vyšší. Obecně se dá říci, že čím více vývodů tím vyšší priorita.

Výsledná teorie tedy spočívá ve vytvoření bloků schématu, ve kterých se prvně umístí prvky s nejvyšší prioritou. K těmto prvkům se ve směru vývodů připojí příšlusné prvky. Na takto připojené prvky se budou napojovat další prvky, které budou zapojováný do nejkratších smyček. Takto vytvořené bloky se spojí a vznikne schéma. Tímto postupem dojde k relativně inteligentnímu rozmístění prvků na schématu.
Systém rozmístění je vytvořen podle výsledné teororie, která se nejvíce podobá systému kreslení schématu člověkem.

Pro rozmísťování prvků na plátně jsem se zabýval teorií, která se označuje evoluční algoritmus. Evoluční algoritmus se opírá o vývoj živých organismů na Zemi. Celá teorie funguje na principu řízené náhody. Jednotlivé prvky v netlistu jsou očíslovány od jedné do celkového počtu prvků. Dále se musí zjistit, zda celkový počet prvků v netlistu je druhá mocnina nějáké hodnoty. Pokud tomu tak není, musí být do netlistu doplněny prvky Null, které nejsou nikam zapojeny, ale pouze doplňují matici. Po tomto kroku se začnou generovat genomy neboli matice druhé mocniny z příslušné hodoty velikosti. Generují se hodnoty od jedné do počtu prvků v netlistu. Těchto matic se vygeneruje nejlépe druhá mocnina počtů prvků v matici. Vygenerované matice projdou kontrolou zda prvky jsou umístěny na správných místech. Pokud se najde matice, která má všechny prvky rozmístěné správně, cyklus je ukončen. Tato situace však nastává velmi zřídka a proto se při procházení matic uděluje hodnocení matice, kterým se říká míra správnosti umístěných jednotlivých prvků. Po udělení hodnocení se matice seřadí od nejlepšího hodnocení k nejhoršímu. Pro další zpracování se použije pouze lepší polovina vygenerovaných matic a ty se vzájemně pokříží. Po tomto pokřížení dochází k opětovnému hodnocení, setřídění a použití poloviny matic. Tak to jde neustále dokola, až zbyde jedna matice, která by měla být správná. Tomuto systému jsem se věnoval velmi dlouhou dobu. Výsledkem však bylo velmi nedokonalé rozmístění

většiny prvků a v mnohých případech i nekonečný cyklus generování nových a nových matic. Proto jsem tento systém zavrhl a vrátil se k původní teorii se smyčkami.

2.3.5 Rozmísťování prvků

Naprogramovaný systém je vytvořen pouze pro schémata s nejnižší prioritou. Do této priority patří součástky se dvěma vývody. Rozmísťovací cyklus začne od prvního zapsaného prvku a k tomu najde prvky, které jsou v první smyčce. Po rozmístění prvků v první smyčce je volána funkce pro hledání dalších smyček na všechny prvky, které se aktuálně rozmístily. Pokud najde prvek, který obsahuje další smyčku, dojde k doplnění Listu s výchozími prvky. Jako výchozí prvek se vždy považuje nultý prvek v Listu s výchozími prvky. Po rozmístění prvků v tomto uzlu dojde k jeho smazání. Při dalším průchodu cyklu Listu s výchozími prvky se naváže na takto nalezené prvky a pokračuje se ve hledání prvků až do kompletního uzavření poslední smyčky.

Rozmísťování prvků na plátně určuje hlavně rotace prvku a hodnoty v položce PlochaPrvku, která je typu RectangleF. Typ RectangleF všechny údaje eviduje v desetinných číslech typu Float, a proto je mnohem lépe použitelný než typ Rectangle, u kterého jsou všechny hodnoty evidovány v číselném typu Int. Údaje v proměnných typu Float jsou lepší z hlediska výpočtu rozmísťování jednotlivých prvků na plátně. Ve velikosti RectangleF se skrývá poloha celého čtverce v proměnných X a Y a jeho velikost v proměnných width a height. Všechny čtyři hodnoty jsou při vytvoření prvku naplněny hodnotami. Pro názornou ukázku jsem použil definování rezistoru.

```
prv.PridejDvou(pole);
prv.PlochaPrvku = new RectangleF(0,0,80,80);
prv.Uzly.Add(new PointF(0, 40));
prv.Uzly.Add(new PointF(80, 40));
prv.Priorita = 1;
```

Zdrojový kod 3: Definice rezistoru

Pro prvotní umístění prvku slouží souřadnice X = 0 a Y = 0. Od těchto souřadnic se pak vypočítávají ostatní polohy uzlů v prvku. Čtvercová velikost prvku 80x80 se mi osvědčila jako nejvhodnější při rozmísťování prvků na plátně. Do této velikosti se i dobře rozpočítávají jednotlivé časti schematické značky. Rotace prvku je v základním nastavení brána jako 0°, což je reprezentováno prvkem v horizontální poloze. Hodnoty rotace nabývájí hodnot 0, 90, 180, 270. Rotace se provádí v rámci definované plochy. Nedochází tak k posunu součástky na jiné místo. Při rotaci prvku musí dojít k přesunu souřadnic popisujících umístění uzlu. Pro tento úkol jsem vytvořil metodu Rotace(int Otoceni), která zajistí kompletní přesun všech souřadnic popisujících výstupy z prvku.

Vykreslování spojnic mezi jednotlivými prvky se definuje v rozmísťovacím cyklu, kde dochází k ukládání souřadnic jednotlivých bodů do Listu Linky. V události panel.OnPaint() dojde k vykreslení spojnic mezi jednotlivými body.

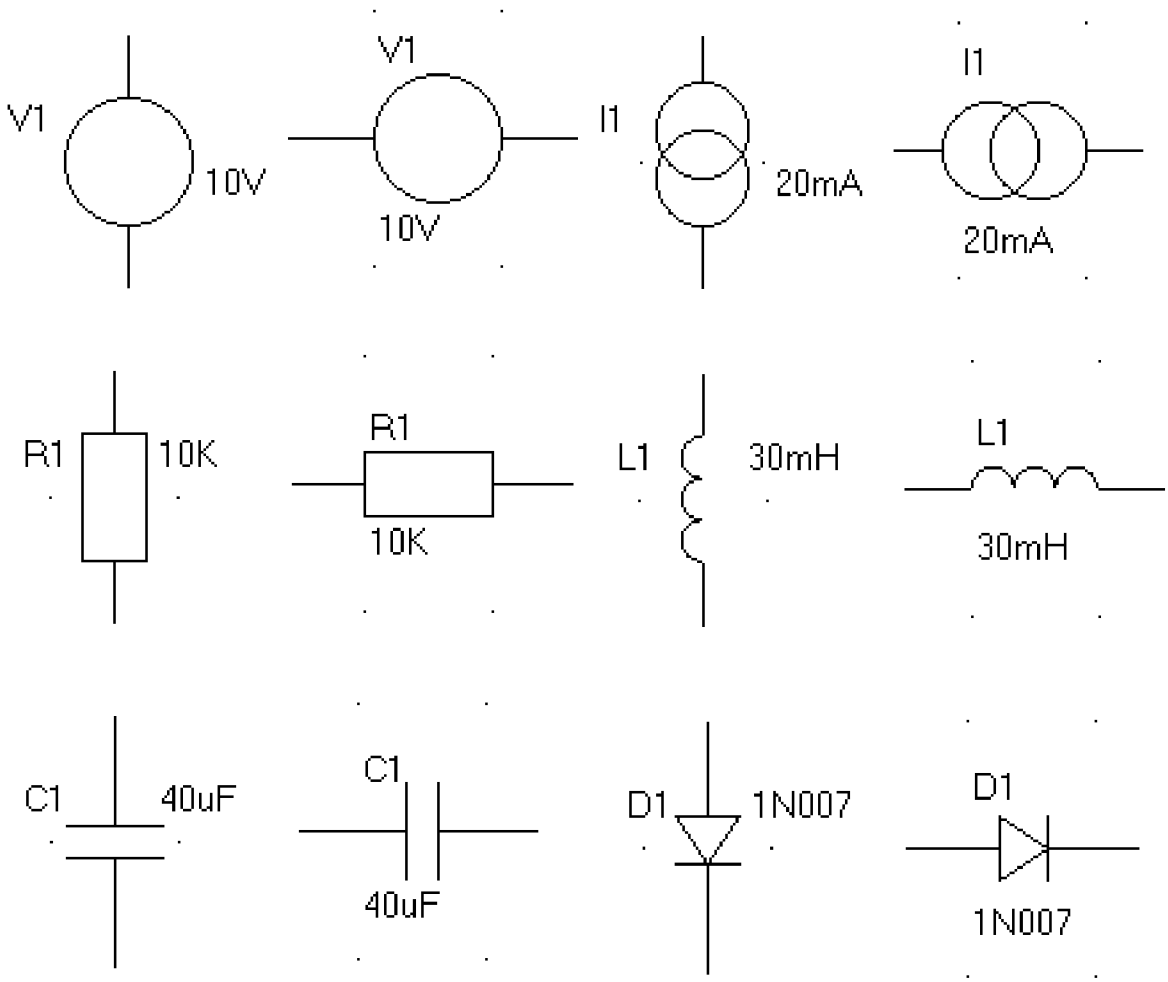
2.3.6 Ukázky součástek pro vykreslení

Každá součástka je přesně definovaná jak se má vykreslit. Do každé součástky je předána hodnota zvětšení "lupa" a odkaz na kreslící plátno "e". Ukázka definice vykreslení součástky typu rezistor. Z ukázky jsem vyřadil vykreslení pro všechny polohy.

```
public void KresliR(float lupa, PaintEventArgs e)
    {
      Font _Popisky = new Font(Properties.Settings.Default.fontpopisku.Name,
Properties.Settings.Default.fontpopisku.SizeInPoints * lupa);
// Nadefinování fontu popisku prvku

      switch (_RotacePrvku)  // Podle rotace prvku se vybere vykreslení
      {
        case 0: // Vykreslení v rotaci 0°;
          {
            e.Graphics.DrawLine(_Pero, _PlochaPrvku.X * lupa, (_PlochaPrvku.Y + 40) *
lupa, (_PlochaPrvku.X + 80) * lupa, (_PlochaPrvku.Y + 40) * lupa);
// Vykreslení linky, která znázorňuje vývody.

            e.Graphics.FillRectangle(_Vypln, (_PlochaPrvku.X + 20) * lupa,
(_PlochaPrvku.Y + 30) * lupa, 40 * lupa, 20 * lupa);
// Vykreslení obdélníku s výplní plátna, aby bylo čisté pozadí a nezasahoval do rezistoru rastr
// z plátna.
            e.Graphics.DrawRectangle(_Pero, (_PlochaPrvku.X + 20) * lupa,
(_PlochaPrvku.Y + 30) * lupa, 40 * lupa, 20 * lupa);
// Vykreslení oramování okolo obdélníku

            e.Graphics.DrawString(_Nazev, _Popisky, _VyplnTextu, new
PointF((_PlochaPrvku.X + 20) * lupa, (_PlochaPrvku.Y + 28) * lupa - _Popisky.Height));
// Výpis a poloha názvu prvku.
            e.Graphics.DrawString(_Velicina, _Popisky, _VyplnTextu, new
PointF((_PlochaPrvku.X + 20) * lupa, (_PlochaPrvku.Y + 52) * lupa));
// Výpis a poloha veličiny u prvku
            break;
          }
        case 90:  { …// Vykreslení v rotaci 90° }
        case 180: { ... // Vykreslení v rotaci 180° }
        case 270: { … // Vykreslení v rotaci 270° }
    }   }
```

Zdrojový kod 4: Vykreslení rezistoru

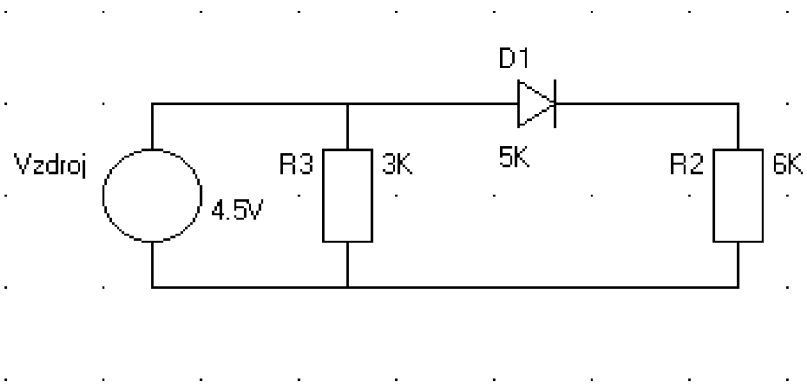
Ukázka samotného vykreslení jednotlivých prvků na plátně:



2.5 Ukázka generovaných schémat

Ukázka 1:

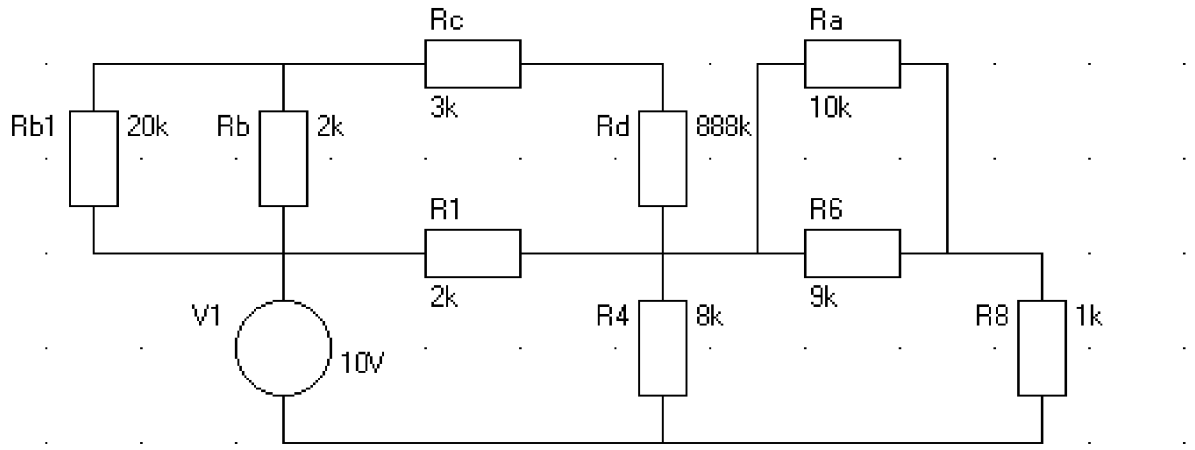
```
Name: Obvod
;Verze: 1.0
;Datum: 27.5.2008
;Autor:
;--------------------------------------------------------------------------
;Popis obvodu:

Vzdroj 1 0 4.5V
R3 1 0 3K
D1 1 2 5K
R2 2 0 6K

;--------------------------------------------------------------------------
;Simulace:

.OP
.TRAN 1u 1m SKIPBP
.PROBE V(Cp) V(Ls)

;--------------------------------------------------------------------------
.end;
```



Obr. 12: Výsledek generování Ukázky 1

Ukázka 2:

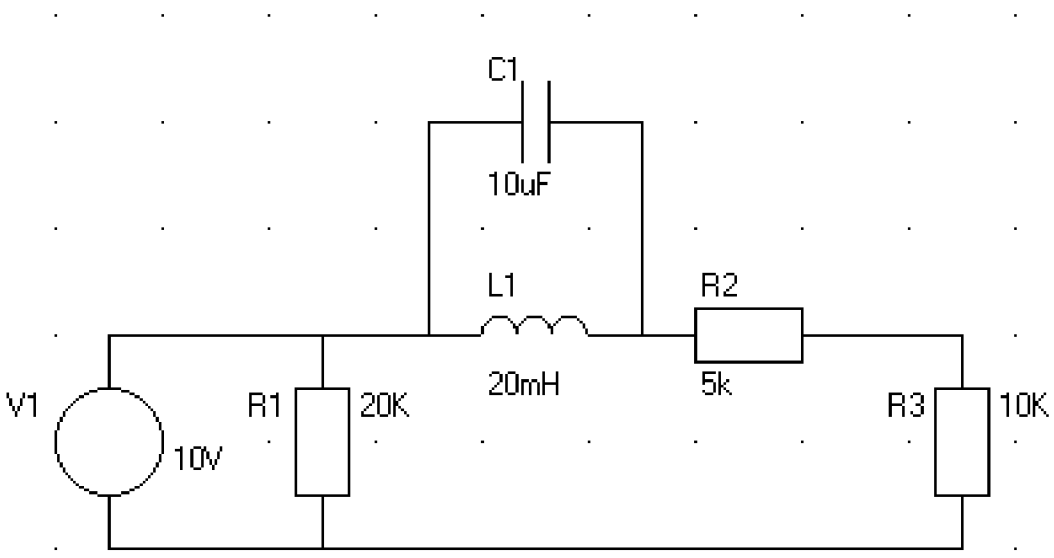
```
Name: Ukazka2
;Verze: 1.0
;Datum: 22.5.2008
;Autor:
;--------------------------------------------------------------------------
;Popis obvodu:

V1 1 0 10V
R1 1 3 2k
Rb 1 a 2k
Rc a b 3k
Rd b 3 888k
Rb1 1 a 20k
R4 3 0 8k
R6 3 5 9k
;R7 4 5 10k
R8 5 0 1k
Ra 3 5 10k
;--------------------------------------------------------------------------
;Simulace:


;--------------------------------------------------------------------------
.end;
```



Obr. 13: Výsledek generování Ukázky 2

Ukázka 3:

```
Name: Ukazka3
;Verze: 1.0
;Datum: 27.5.2008
;Autor:
;-------------------------------------------------------------------------
;Popis obvodu:

V1 1 0 10V
R1 1 0 20K
L1 1 2 20mH
C1 1 2 10uF
R2 2 3 5k
R3 3 0 10K

;-------------------------------------------------------------------------
;Simulace:


;-------------------------------------------------------------------------
.end;
```



Obr. 14: Výsledek generování Ukázky 3

2.6 Výstupy z programu

Program umožňuje několik druhů výstupu. Základním výstupem z programu je možnost uložení netlistu do libovolného textového souboru bez formátování. V programu jsou předefinované tyto koncovky: CIR-oficiální označení netlistu, TXT – libovolný textový soubor. Netlist je možné rovněž vytisknout. Pro tisk netlistu jsem využil objekt PrintDocument, u kterého se definuje událost PrintPages a v ní provedu kompletní převod textu s richTextBox1 na plátno. Pro velikost fontu jsem zvolil velikost 12.

Vygenerované schéma je možné uložit do rastrového obrázku. Program umožňuje uložení schématu do rastru typu JPG, PNG a BMP. Schéma je rovněž možné tisknout. Pro výtisk jsem zvolil komponentu printPreviewDialog1, která slouží jako náhled před tiskem. S komponentou se pracuje velmi podobně jako s objektem PrintDocument. Události PrintPages jsem přiřadil tiskovou metodu, ve které jsem převzal plátno z komponenty panelu.

3. Závěr

Konverze schématu zapsaného v jazyce PSpice netlist do grafické podoby je velice náročná. Prvním krokem pro správnou grafickou reprezentaci bylo docílit přesné analýzy netlistu. Dosáhlo se úplného rozboru jednotlivých prvků ve schématu, vyfiltrování všech nepotřebných příkazů pro simulaci, které nemají vliv na schéma. Přesná specifikace spojení všech součástek v obvodu zajišťuje jasné podklady pro rozmisťovací algoritmus. Vytvoření algoritmu pro rozmisťování součástek na plátno bylo na celém projektu nejtěžší. Pro rozmisťování součástek bylo vytvořeno a otestováno několik algoritmů. Nejvhodnější algorimus spočívá ve vyhledávání smyček v obvodu. Před začátekem hledání se stanoví výchozí prvek a k němu se hledá nejkratší cesta obvodem. Po nalezení nejkratší cesty systém začne jendotlivé součástky rozmisťovat. Rozmisťovací algoritmus obsahuje velké množství podmínek a výpočtů pro rozmístění součástek. Systém umí rozmísťit jedotlivé prvky tak, aby se navzájem nepřekrývaly a nedocházelo ani k překrývání vodíčů, nebo průchodem vodiče přes součástku. Grafická reprezetace netlistu je rovněž optimalizovaná pro zabrání minimální velikosti plátna. Člověk při návrhu schématu se snaží o maximální přehlednost, jednoduchost a intuitivní rozmístění všech součástek. V software CPS Netlist se dosahuje stejného výsledku. Jako každý systém tohoto typu není dokonalý a může se vyskytnou schéma, které algoritmus nemusí umět správně rozmístit. Aplikaci tohoto typu je možné neustále zdokonalovat a doplňovat o nové a nové věci, které člověku pomohou v jeho práci.

4. Použitá literatura

[1] BIOLEK, Dalibor, *Modelování a simulace v mikroelektronice*. Brno: Vysoké učení technické v Brně, 2005. 136 stran.

[2] PUŠ, Petr. Poznáváme C# a Microsoft .NET [online]. 2004 [cit. 2008-05-28]. Dostupný z WWW: <http://www.zive.cz/Clanky/Poznavame-C-a-Microsoft-NET--1dil/sc-3-a-120978/default.aspx>

5. Příloha

[1] CD s verzí programu pro konverzi SPICE netlist do grafické podoby