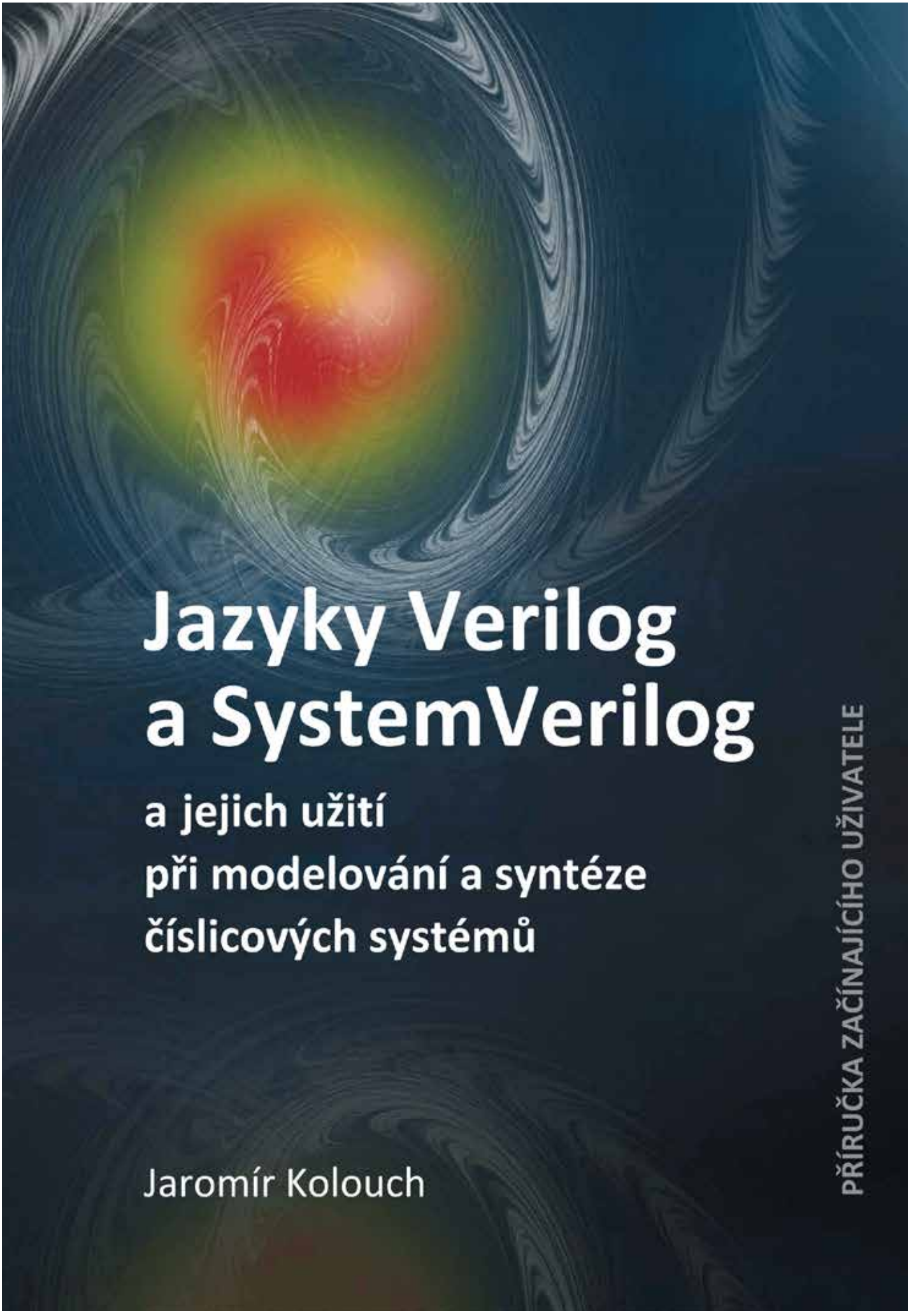

Jazyky Verilog a SystemVerilog

a jejich užití při modelování a syntéze číslicových systémů

PŘÍRUČKA ZAČÍNAJÍCÍHO UŽIVATELE

Jaromír Kolouch

Jazyky Verilog a SystemVerilog

a jejich užití

při modelování a syntéze číslicových systémů

Příručka začínajícího uživatele

Jaromír Kolouch

Lektorovali:
doc. Ing. Martin Poupa, Ph.D.
Ing. Marcela Šimková

ISBN 978-80-214-5216-9

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
NAKLADATELSTVÍ VUTIUM

Obsah

Zkratky

ASIC	Application Specific Integrated Circuits
CAD	Computer Aided Design
CPLD	Complex Programmable Logic Devices
DPI	Direct Programming Interface
FPGA	Field Programmable Gate Arrays
HDL	Hardware Description Language
IP	Intellectual Property
LUT	Look-up Table
PAL	Programmable Array Logic
PLA	Programmable Logic Array
PLD	Programmable Logic Devices
RTL	Register Transfer Level
SOP	Sum of Products

1 Předmluva

Tento text je určen pro čtenáře, který se chce seznámit s jazykem Verilog a s jeho novější zdokonalenou a rozšířenou verzí – či správnější by snad bylo říci nadstavbou – s jazykem SystemVerilog, a s jejich použitím při návrhu aplikací programovatelných obvodů známých jako **obvody PLD** (*Programmable Logic Devices*) a **FPGA** (*Field Programmable Gate Arrays*), které budeme dále souhrnně (i když ne zcela přesně) označovat vžitým názvem programovatelné obvody. Měl by čtenáře připravit pro praktickou aplikaci těchto obvodů v číslicových konstrukcích. Důraz je zde proto kladen především na syntézu; simulace je probrána v rozsahu, který je potřebný pro ověření funkce takových aplikací. Do značné míry je text použitelný i pro čtenáře, který se zaměřuje na obvody ASIC (*Application Specific Integrated Circuits*), i když tam jsou určité odlišnosti, má-li být výsledek implementace optimální. Aby byl jeho rozsah udržen v přijatelných mezích, není v něm cílem úplnost výkladu syntaxe. Není v něm zahrnuta například většina nesyntetizovatelných částí jazyka nebo úplný výčet různých variant hodnot operandů v popisu funkce operátorů (zejména reálných nebo nedefinovaných hodnot), nebo také podrobnosti o příkazech skupiny `generate`. Nejsou zde také podrobně uvedeny informace o organizaci výpisů na konzolu při simulaci, které odpovídají příslušným výpisům při práci s jazykem C (systémová úloha `$display`, řídící řetězce formátu pro výpisy apod.). V některých případech se pak spoléhá na intuici čtenáře. Zájemce o podrobnější a úplnější zpracování najde informace ve speciální literatuře, především v referenčních manuálech jazyků Verilog [1], popř. [2] a SystemVerilog [6]; někdy může být užitečné vědět i o starších verzích standardu [4], [5]. Standard pro syntézu modelů zapsaných ve Verilogu je předmětem publikace [3]. Referenční manuál jazyka SystemVerilog [6] verze 2012 lze bezplatně stáhnout z www stránky http://standards.ieee.org/getieee/1800/download/1800-2012.pdf. V něm není rozlišeno použití jazyka SystemVerilog k verifikaci a k syntéze, tj. není vyznačena syntetizovatelná část jazyka, což lze pokládat za určitý nedostatek. Použití SystemVerilogu k syntéze je podrobně zpracováno v knize [11], jejíž autor patří k duchovním otcům a předním popularizátorům tohoto jazyka, která však u čtenáře předpokládá znalost jazyka Verilog. Poznatky i příklady v ní uvedené vhodně doplňují referenční manuály a zde se na ni budeme často odkazovat. V některých případech se však vyskytují drobné odlišnosti mezi touto knihou a referenčními manuály, a na ně v našem textu upozorníme, protože je možné, že verze uvedená v knize bude lépe odpovídat současným syntetizérům. O standardech pro syntézu se ještě zmíníme v **odst. 2.1**. Použití jazyka SystemVerilog k verifikaci je podobně zpracováno například v knize [14].

U čtenáře se předpokládá znalost číslicové techniky v rozsahu odpovídajícím základním kurzům technických vysokých nebo i středních škol, shrnutém například v publikaci [8] nebo [9]. Bude také užitečná (i když ne nezbytná) aspoň orientační znalost obvodových struktur a funkčních bloků používaných v obvodech PLD a FPGA. Dále se předpokládá (i když to není bezpodmínečně nutné), že čtenář dokáže pracovat s některým návrhovým systémem, v němž jsou jazyky Verilog a SystemVerilog podporovány, například se systémem Quartus II firmy Altera; další známá firma Xilinx má oba jazyky implementovány v systému Vivado, starší systém ISE podporuje jen jazyk Verilog. Základní verzi systému Quartus – tzv. Web Edition – může zájemce bezplatně stáhnout z www stránek firmy Altera, a podobné je to i u systémů firmy Xilinx. Výhodné, i když nikoli nutné, pro něj bude, když bude mít zkušenosti s některým u nás rozšířeným jazykem HDL (*Hardware Description Language*), například s jazykem VHDL – pro seznámení s tímto jazykem v češtině lze doporučit například knihu [8]. V angličtině jsou jazyky Verilog i VHDL i s ohledem na syntézu zpracovány přístupnou formou například v publikaci [7].

Jazyky HDL jsou dnes nejpoužívanějším a většinou nejefektivnějším prostředkem pro modelování vyvíjených číslicových systémů a pro jejich následnou syntézu a implementaci do programovatelných obvodů i do obvodů ASIC. Nenajdeme dnes prakticky používaný návrhový systém, který by tyto jazyky, především VHDL a Verilog, nepodporoval. U nás se dosud především z historických důvodů pracovalo převážně s jazykem VHDL, který byl nejrozšířenějším jazykem HDL v evropských zemích, zatímco jazyk Verilog dříve dominoval v asijských zemích, a v zemích amerického kontinentu byly zhruba stejně rozšířeny oba tyto jazyky. Tomu také odpovídá výuka v českých školách i literatura psaná v češtině, jako např. [8]. Ve světě se ale stále více prosazuje jazyk Verilog, který je jednodušší na pochopení a stručnější ve vyjadřování, poskytuje však prakticky stejné možnosti jako jazyk VHDL; zhruba od začátku tohoto desetiletí přebírá vedoucí úlohu jazyk SystemVerilog, který je nadstavbou nad Verilogem. V praxi se proto konstruktéři číslicových systémů často setkají se zdrojovými texty napsanými ve Verilogu nebo v SystemVerilogu, které získají například z internetu

nebo z literatury, a měli by být schopni tyto texty převzít, porozumět jim a upravit je podle vlastní potřeby. Proto lze očekávat, že uvítají příručku, která může sloužit i jako učebnice jazyků Verilog a SystemVerilog a může představovat vodítko, jak psát zdrojové texty pro návrhové systémy představující syntetizovatelné modely vytvářených číslicových systémů.

Prostředky jazyků Verilog a VHDL dovolují efektivně modelovat číslicové bloky na úrovni modulů. Oba tyto jazyky podporují ovšem taky hierarchičnost projektů, při projektech rozsáhlých systémů se však popisy zapsané v těchto jazycích stávají dosti nepřehlednými. SystemVerilog, podobně jako jazyk SystemC, obsahuje nové jazykové konstrukty (jako příklad uveďme rozhraní – *interface*), které výrazně zlepšují přehlednost popisů takových systémů a usnadňují práci s nimi, a jsou příslibem jednotné metodiky pro konstrukci a verifikaci celých systémů na čipu (*System-on-Chip*, SoC) od konstukce na úrovni modulů a od simulace na úrovni hradel až po verifikaci na systémové úrovni. Odtud vychází název jazyka. SystemVerilog dále obsahuje proti Verilogu řadu zlepšení, která jej činí přátelštější pro uživatele, a to i u menších projektů.

Textové formy popisu modelů nejsou jediné. Kromě nich se v návrhových systémech setkáme i s dalšími formami popisu, zejména grafickými – jsou to například schémata nebo stavové diagramy. Práce s nimi je značně odlišná u různých návrhových systémů a nebudeme ji zde podrobněji diskutovat. Jen jako příklad alternativy strukturálního popisu v jazyku Verilog uvedeme v **odst. 5.1.5** ukázku blokového schématu vytvořeného v editoru schémat systému ISE.

Velká většina zdrojových textů zapsaných v jazyku SystemVerilog, které jsou uvedeny jako příklady v této knize, byla ověřena z hlediska syntaktické správnosti a možnosti simulace a syntézy systémem Quartus II firmy Altera. Odkazy na tento systém i na syntetizér XST (*Xilinx Synthesis Technology*) systému ISE se týkají jeho verze aktuální v době zpracování knihy, tj. verze 11. Příklady textů zapsaných ve Verilogu byly ověřovány také v systému ISE.

Terminologii věnují oficiální dokumenty jazyka Verilog, na který SystemVerilog navazuje, výrazně menší pozornost než odpovídající dokumentace k jazyku VHDL. Některé pojmy jsou v tomto jazyku používány víceméně volně, a často nejsou definovány termíny obdobné termínům jazyka VHDL, i když by se to mohlo jako účelné očekávat. Je to zřejmé zejména u označení typů signálů – například pojem *typ* není ve Verilogu vůbec oficiálně zaveden, i když se často používá ve smyslu dosti podobném jako v jazyku VHDL. V SystemVerilogu se přes zjevnou snahu o nápravu nepodařilo tento nedostatek zcela odstranit, přičemž dosti značnou nesnáz zřejmě představoval požadavek dodržení kompatibility s Verilogem. Pro čtenáře zvyklého na přesnější názvosloví je matoucí (hlavně při zápisu textů určených k syntéze) také označení *register*, používané ve starších anglických publikacích o Verilogu, jako je například **[16]**, pro proměnné typů `reg`, `integer`, `time`, `real` a `realtime`. Toto označení nemá nic společného s obvodovým prvkem – klopným obvodem či registrem, a myslí se jím paměťové místo v paměti počítače, na němž běží simulace kódu zapsaného ve Verilogu. Označení proměnné slovem *register* bylo ve Verilogu-2001 opuštěno a zbylo po něm jen klíčové slovo `reg`. V SystemVerilogu je definován nový typ `logic` s vlastnostmi shodnými s typem `reg`, jehož název nevyvolává nesprávné asociace (význam obou je však v SystemVerilogu poněkud posunut). Někteří autoři píšící v angličtině se snaží terminologické mezery vyplnit, často právě na základě terminologie zavedené v jazyku VHDL. Terminologie používaná v této knize vychází z dokumentu **[6]** a z doplnění tam zavedené terminologie ponejvíce podle podle pramenu **[7]**, s některými drobnými odchylkami – například pro popisy zapsané v jazycích HDL se v **[7]** (často i jinde) používá označení „program“, kdežto autor se tomuto označení vyhýbá, protože podle jeho názoru existuje dosti zásadní rozdíl mezi počítačovým programem a popisem hardwarového uspořádání, kterému odpovídá popis v jazyku HDL; za vhodnější považuje například výrazy „zdrojový text“, „kód“ nebo prostě „popis“. (V SystemVerilogu má slovo `program` nově definovaný význam – je klíčovým slovem s významem blízkým počítačovému programu, s použitím hlavně při verifikaci.) Dále budeme termínem „objekt“ rozumět datové objekty, nikoliv instance tříd, jak je to obvyklé u objektového programování, které SystemVerilog ve své nesyntetizovatelné části podporuje.

Obvykle se v literatuře o Verilogu a o SystemVerilogu také nerozlišuje mezi definicí a deklarací, zejména ve vztahu k funkcím, úlohám, uživatelsky definovaným typům a k rozhraním. Tyto pojmy se často používají záměnně. V této knize se budeme snažit o rozlišení těchto pojmů.

Termíny používané v dalším textu a vztahující se k práci s návrhovým systémem odpovídají terminologii zavedené v systému ISE firmy Xilinx, v případě potřeby doplněné o terminologii používanou v systému Quartus firmy Altera. U jiných návrhových systémů mohou být (a také bývají) odchylky.

Autor bude vděčný laskavým čtenářům za upozornění na chyby a nepřesnosti, které najdou v tomto textu. Zašlete je prosím na adresu:

kolouch@feec.vutbr.cz

Opravy chyb, které budou nalezeny v této knize (autor věří, že jich nebude mnoho), budou dostupné na stránce:

http://www.urel.feec.vutbr.cz/~SystemVerilog/

1.1 Klíčová slova jazyků Verilog a SystemVerilog

KLÍČOVÁ SLOVA jazyka *Verilog-95* s doplněním o klíčová slova verze *Verilog-2001* a *Verilog-2005* (kurziva), [1]:

always	**endmodule**	***liblist***	**rcmos**	**tranif1**
and	**endprimitive**	***library***	**real**	**tri**
assign	**endspecify**	***localparam***	**realtime**	**tri0**
automatic	**endtable**	**macromodule**	**reg**	**tri1**
begin	**endtask**	**medium**	**release**	**triand**
buf	**event**	**module**	**repeat**	**trior**
bufif0	**for**	**nand**	**rnmos**	**trireg**
bufif1	**force**	**negedge**	**rpmos**	***unsigned***
case	**forever**	**nmos**	**rtran**	***use***
casex	**fork**	**nor**	**rtranif0**	***uwire *)***
casez	**function**	***noshowcancelled***	**rtranif1**	**vectored**
cell	***generate***	**not**	**scalared**	**wait**
cmos	***genvar***	**notif0**	***showcancelled***	**wand**
config	**highz0**	**notif1**	***signed***	**weak0**
deassign	**highz1**	**or**	**small**	**weak1**
default	**if**	**output**	**specify**	**while**
defparam	***ifnone***	**parameter**	**specparam**	**wire**
design	***incdir***	**pmos**	**strong0**	**wor**
disable	***include***	**posedge**	**strong1**	**xnor**
edge	**initial**	**primitive**	**supply0**	**xor**
else	**inout**	**pull0**	**supply1**	
end	**input**	**pull1**	**table**	
endcase	***instance***	**pulldown**	**task**	
endconfig	**integer**	**pullup**	**time**	
endfunction	**join**	***pulsestyle_onevent***	**tran**	
endgenerate	**large**	***pulsestyle_ondetect***	**tranif0**	

***) Poznámka:** Klíčové slovo `uwire` je zavedeno ve verzi Verilog-2005 (jediné nové klíčové slovo zavedené v této verzi).

KLÍČOVÁ SLOVA jazyka nově zavedená ve verzi *SystemVerilog-2005* s doplněním o klíčová slova verze *SystemVerilog-2009* (kurziva), [4], [11]:

accept_on	**dist**	**import**	**randsequence**	**type**
alias	**do**	**inside**	**ref**	**typedef**
always_comb	***endchecker***	**int**	***reject_on***	**union**
always_ff	**endclass**	**interface**	***restrict***	**unique**
always_latch	**endclocking**	**intersect**	**return**	***unique0***
assert	**endgroup**	**join_any**	***s_always***	***until***
assume	**endinterface**	**join_none**	***s_eventually***	***until_with***
before	**endpackage**	***let***	***s_nexttime***	***untyped***
bind	**endprogram**	**local**	***s_until***	**var**
bins	**endproperty**	**logic**	***s_until_with***	**virtual**
binsof	**endsequence**	**longint**	**sequence**	**void**
bit	**enum**	**matches**	**shortint**	**wait_order**
break	***eventually***	**modport**	**shortreal**	***weak***
byte	**expect**	**new**	**solve**	**wildcard**
chandle	**export**	***nexttime***	**static**	**with**
checker	**extends**	**null**	**string**	**within**
class	**extern**	**package**	***strong***	
clocking	**final**	**packed**	**struct**	
const	**first_match**	**priority**	**super**	
constraint	**foreach**	**program**	***sync_accept_on***	
context	**forkjoin**	**property**	***sync_reject_on***	
continue	***global***	**protected**	**tagged**	
cover	**iff**	**pure**	**this**	
covergroup	**ignore_bins**	**rand**	**throughout**	
coverpoint	**illegal_bins**	**randc**	**timeprecision**	
cross	***implies***	**randcase**	**timeunit**	

Poznámka: Klíčová slova všech verzí jazyka Verilog zůstávají klíčovými slovy jazyka SystemVerilog. Direktivami `` `begin_keywords `` a `` `end_keywords `` lze v SystemVerilogu nastavit platnou sadu klíčových slov, viz **odst. 2.1.2**.

KLÍČOVÁ SLOVA jazyka nově zavedená ve verzi *SystemVerilog-2012* [6]:

implements	**interconnect**	**nettype**	**soft**

1.2 Slovníček termínů užívaných v jazycích Verilog a SystemVerilog

Termíny používané v anglických textech o Verilogu a SystemVerilogu nemají dosud v četných případech všeobecně akceptovaný český překlad. Autor se pokusil doplnit českou terminologii, a pro snadnější orientaci čtenáře je zde připojen malý slovníček těchto termínů. Pro stručnost v něm nejsou obsaženy termíny, jejichž překlad je totožný či téměř totožný s původním slovem, a termíny, jejichž překlad je všeobecně známý a akceptovaný.

array	pole	bitový výběr	bit-select
assignment	přiřazení	brána	port
association	přidružení	citlivostní seznam	sensitivity list
bit-select	bitový výběr	částečně typový	semi-strongly typed
built-in	vestavěný	částečný výběr	part-select
code	kód (zdrojový text)	hlavička	header
concatenation	sjednocení	kód (zdrojový text)	code
concurrent statement	souběžný příkaz	nesbalený	unpacked
connection	připojení	odskočený identifikátor	escaped identifier
context-determined	s kontextovým určením	ořízntutí (zkrácení)	truncation
enumerated type	výčtový typ	pole	array
escaped identifier	odskočený identifikátor	proměnná	variable
expression	výraz	propojení (typ signálu)	net
full	úplný (příkaz `case`)	prostor (kompilační jednotky)	scope
handle	úchyt	prostor jména	space of a name
header	hlavička	přenositelnost	portability
inferencing	rozpoznání	převod typu	type casting
instantiation	vložení	přidružení	association
interface	rozhraní	příkaz	statement
justification	zarovnání	připojení	connection
layout (e.g., of an array)	uspořádání (např. pole)	přiřazení	assignment
loop	smyčka	rozhraní	interface
nested module	vnořený modul	rozpoznání	inferencing
net	propojení (typ signálu)	rozsah (v deklaraci)	range
package	sloha	rozsah působnosti	scope
packed	sbalený	rozšíření (doplnění)	padding
padding	rozšíření (doplnění)	s kontextovým určením	context-determined
part-select	částečný výběr	s vlastním určením	self-determined
port	brána	sbalený	packed
portability	přenositelnost	sjednocení	concatenation
range	rozsah (v deklaraci)	sloha	package
scope	rozsah působnosti, prostor (kompilační jednotky)	smyčka	loop
		souběžný příkaz	concurrent statement
self-determined	s vlastním určením	šablona	template
semi-strongly typed	částečně typový	úchyt	handle
sensitivity list	citlivostní seznam	úloha	task
side effect	vedlejší účinek	úplný (příkaz `case`)	full
source text	zdrojový text	uspořádání (např. pole)	layout (e.g., of an array)
space of a name	prostor jména	vedlejší účinek	side effect
statement	příkaz	vestavěný	built-in
task	úloha	vložení	instantiation
template	šablona	vnořený modul	nested module
truncation	ořízntutí (zkrácení)	výčtový typ	enumerated type
type casting	převod typu	výraz	expression
unpacked	nesbalený	zarovnání	justification
variable	proměnná	zdrojový text	source text

2 Syntaxe jazyků Verilog a SystemVerilog a modelování číslicových systémů

2.1 Úvodní poznámky

Jazyky Verilog a SystemVerilog patří do skupiny jazyků HDL (*Hardware Description Language*), které se používají pro návrh aplikací obvodů PLD a FPGA. Dalším jazykem tohoto druhu, v současnosti zhruba stejně rozšířeným, je jazyk VHDL. Jak jazyk VHDL, tak i Verilog (na který navazuje SystemVerilog) byly původně vyvinuty pro účely simulace a dokumentace, nikoliv pro syntézu. Na mnoha jejich konstruktech je to znát, mnohé z nich nemají pro syntézu vůbec význam. Zde se budeme zabývat především použitím jazyků Verilog a SystemVerilog k vytváření modelů určených pro syntézu číslicových systémů; jak již bylo řečeno v úvodu, většinu nesyntetizovatelných konstruktů zde nebudeme diskutovat. Simulaci budeme využívat hlavně pro ověření správnosti funkce syntetizované konstrukce. Lze ji užít i pro zjištění časových parametrů, k čemuž v jazycích HDL slouží speciální jazykové konstrukty, zde však budeme k tomuto účelu předpokládat především použití statické časové analýzy.

České termíny budou v prvním výskytu zapsány **tučně**. Často tyto termíny nejsou ustálené, a proto budeme uvádět i jejich anglické ekvivalenty, které jsou většinou používány standardně. Bude-li to pro orientaci v textu účelné, mohou tak být vyznačeny termíny i na více místech (opakovaně). Anglické termíny a často užívané fráze jsou v textu uváděny *kurzivou* (zpravidla v závorce). Tučně a modrou barvou jsou také psána klíčová slova. Slova podobného charakteru, která však nejsou klíčovými slovy ve smyslu jazyků Verilog a SystemVerilog (jako jsou například direktivy nebo názvy systémových funkcí), budou psána fontem odpovídajícím zápisu zdrojových textů, avšak kurzivou a nikoliv tučně.

Snahou autora bylo podchytit v textu jak to, co se týká Verilogu, tak i to, co je specifické pro SystemVerilog, a to bez opakování, které by bylo nutné, pokud by se měly popsat konstrukty platné ve Verilogu i v SystemVerilogu samostatně. V některých případech by tak mohlo být nesnadné poznat, co platí pro Verilog a co jen pro SystemVerilog, a to zejména v **odst. 2.3.1**, a také v **odst. 2.3.9** a v **odst. 2.3.10**. Proto jsou v těchto podkapitolách uvedeny části platné jen v SystemVerilogu hnědou barvou. Jiné kapitoly či podkapitoly, kde tento problém není tak závažný, jsou však psány běžným způsobem (černým písmem).

Předmět popisu konstrukce, ať již textového, grafického či smíšeného, budeme označovat za její **model**. Pro popis modelu zapsaný v jazycích Verilog i SystemVerilog budeme také používat termín **zdrojový text** nebo stručněji **kód**, což odpovídá často užívaným anglickým termínům *source text* (*code*).

Základní vlastnosti jazyků Verilog a SystemVerilog (totéž platí i pro jazyk VHDL):

- Představují **otevřený standard** (*open standard*). K jejich použití pro sestavení návrhových systémů není třeba licence jeho vlastníka, jako je tomu u některých jiných jazyků HDL (například u jazyka ABEL). Asi se nenajde návrhový systém, který by jazyky Verilog a VHDL nepodporoval, a v dohledné době to zřejmě bude platit i pro SystemVerilog.
- Umožňují pracovat na návrhu, aniž je předtím zvolen cílový obvod (*device-independent design*). Ten může být zvolen až v době, kdy jsou známy definitivní požadavky na prostředí, v němž má navrhovaná konstrukce pracovat, a je možno jej ve značné míře měnit podle potřeby při zachování popisu v jazyku HDL. Může být zvolen obvod PLD nebo FPGA, aniž se volba dotkne popisu, pokud ovšem tento obvod obsahuje bloky, které popis předpokládá.
- Je možno provést simulaci navrženého obvodu na základě téhož zdrojového textu, který pak bude použit pro syntézu a implementaci v cílovém obvodu. Zdrojový text je možno zpracovávat v různých simulátorech a v syntetizérech různých výrobců (opět s určitým omezením daným zejména vnitřní strukturou cílových obvodů). Odsimulovaný text může být použit v dalších projektech s různými cílovými obvody, což je umožněno tím, že jazyky Verilog i SystemVerilog podporují hierarchickou strukturu projektů, kde je vytvářený systém složen z dílčích bloků. Tyto bloky mohou představovat typické funkční celky, jako jsou například převodníky kódu, čítače, aritmetické obvody, procesory a podobně, které mohou být

opakovaně využity v dalších projektech a mohou být implementovány do jiných cílových obvodů, případně zpracovávány na jiných návrhových systémech. Této vlastnosti jazyka se říká **přenositelnost** (*portability*) kódu.

- V případě úspěšného zavedení výrobku na trh lze popis konstrukce v jazycích HDL použít jako podklad pro její implementaci do obvodů ASIC vhodných pro velké série.

Základní verze jazyka Verilog byla přijata jako standard IEEE číslo 1364 v roce 1995. Konstrukty odpovídající tomuto standardu se označují jako konstrukty jazyka Verilog-95. Na základě zkušeností s touto verzí byla v roce 2001 přijata nová verze, Verilog-2001, [1]. V této verzi je provedena řada úprav a zdokonalení a podporuje ji většina současných návrhových systémů. Následující – poslední verze Verilogu [2] z roku 2005 obsahuje jen malé korekce, jejichž podpora ve vývojových nástrojích není příliš rozšířená. Pro tyto příručky i pro podobné dokumenty platné pro SystemVerilog se často používá označení LRM (*Language Reference Manual*).

V roce 2002 přijala organizace IEEE standard IEEE Std 1364.1™-2002 [3], který definuje podmnožinu konstruktů jazyka Verilog-2001 vhodnou pro syntézu na úrovni RTL (význam této zkratky je vysvětlen v **odst. 2.1.1**). Tento standard zajišťuje přenositelnost i pro výsledky syntézy u syntetizérů, které jej splňují.

Jazyk SystemVerilog navazuje na Verilog. Byl vyvíjen organizací Accelera s úmyslem doplnit standard Verilogu o nové možnosti potřebné zejména pro konstrukci rozsáhlých systémů. O procesu vývoje tohoto jazyka referují podrobně prameny [26], [27].

V roce 2005 byly organizací IEEE přijaty dva standardy: Verilog-2005 (*IEEE 1364-2005*) a SystemVerilog-2005 (*IEEE 1800-2005*). Tyto standardy byly sjednoceny v roce 2009 pod označením SystemVerilog-2009 (*IEEE 1800-2009*). Jazyk Verilog-2005 zde představuje podmnožinu jazyka SystemVerilog-2009. SystemVerilog je tedy zpětně kompatibilní s Verilogem a **nástroje pro SystemVerilog zpracovávají zdrojové texty zapsané ve Verilogu bez problémů.** Renomovaní autoři, jako je například Stuart Sutherland, tvrdí, že se další vývoj již bude týkat jen jazyka SystemVerilog, viz [26], [27], a že v současnosti všichni, kdo pracují s Verilogem, používají ve skutečnosti SystemVerilog (i když si to mnohdy neuvědomují). Další verze, SystemVerilog-2012, [6], přináší pro syntézu jen nevelké úpravy. Všechny změny provedené ve verzi 2012 jsou přehledně uvedeny v [28].

V manuálech SystemVerilogu [3], [5], [6] není syntetizovatelná podmnožina jazyka vyznačena. Do značné míry nahrazují tento nedostatek články [29], [30] a kniha [11].

V dalším textu budeme označením Verilog myslet verzi Verilog-2001 podle standardu [1], která je v současnosti nejrozšířenější verzí. Pokud bychom měli na mysli jinou verzi, bude to uvedeno jejím explicitním číselným označením. Podobně označení SystemVerilog se bude týkat verze SystemVerilog-2012 [6].

Za základní dokument, který by měl definovat chování nástrojů, jako jsou simulátory, syntetizéry a podobně, se většinou považují manuály popisující standardy IEEE, tedy v současnosti publikace [1], [6] (standard pro Verilog je patrně možno považovat za konečný, u SystemVerilogu dochází přibližně v pětiletých intervalech k inovacím). V některých případech však jejich text není zcela vyčerpávající a srozumitelný, což se týká zejména syntetizovatelnosti, jak již bylo zmíněno. Pro SystemVerilog je pak vhodné doplnit informace z manuálu další literaturou. Pro naše účely je k tomu vhodná zejména kniha [11], jejíž autoři patří ke tvůrcům jazyka. Stává se občas (sice zřídka, ale přece jen), že se manuál a uvedená kniha v některých podrobnostech rozcházejí, viz např. **odst. 3.1** nebo **odst. 3.3.7.1**. V takových případech se autor tohoto textu snažil na to upozornit – může se stát, že chování některých nástrojů bude spíš odpovídat tomu, co je psáno v knize [11] než v manuálu [6], nemusí však tomu tak být vždy. Informace z literatury, zejména z internetu, je však třeba přebírat s opatrností, nezřídka se tam vyskytují nepřesnosti či dokonce chyby. Proto se autor snažil v tomto textu uvádět časté odkazy na to, odkud jsou informace doplňující referenční manuály převzaty, aby si čtenář mohl jejich věrohodnost snadněji ověřit.

Inovace v SystemVerilogu jsou zaměřeny především na usnadnění práce s rozsáhlými projekty, i když mnoho zlepšení zavedených v tomto jazyku je užitečné i u menších projektů. Jedním z důležitých zlepšení je omezení redundance zápisu. Ve Verilogu je například nutné při vkládání komponent strukturálním stylem

vypsat seznam bran vkládané komponenty do příkazu vložení. U malých projektů to příliš nevadí a může se to zdát jako drobnost. V moderních konstrukcích však komponenty často mívají stovky i více bran, jejichž seznam je nutno ve všech vloženích znovu vypisovat. Dojde-li při vývoji ke změně v branách, je nezbytné všechna vložení opravit. V SystemVerilogu je vylepšenou syntaxí dosaženo toho, že většinou stačí opravit seznam bran jen v jednom místě, není potřebné tuto opravu provádět u každého vložení této komponenty. Obecně je tam kladen důraz na zásadu, že malé změny v konstrukci by neměly vyvolávat nutnost oprav na mnoha místech.

Jazyk Verilog tedy můžeme v současné době považovat za podmnožinu jazyka SystemVerilog, Pokud nebude výslovně uvedeno jinak, platí v následujícím textu vše, co bude psáno pro Verilog, i v SystemVerilogu (samozřejmě však ne naopak). Přesto však někdy, když to bude účelné, budeme o Verilogu a o SystemVerilogu mluvit jako o dvou jazycích, i když to není zcela přesné.

V jazycích Verilog i SystemVerilog je možno vytvořit neefektivní konstrukce, přičemž efektivnost (nebo její nedostatek) nemusí být na první pohled ze zdrojového textu patrná. To však platí i o jiných jazycích vyšší úrovně. Výsledná efektivnost konstrukce závisí jak na kvalitě programových návrhových prostředků, tak i na zkušenosti konstruktéra (návrháře) – výhodu zde mají konstruktéři, kteří prošli přípravou s jazykem nižší úrovně, jako je například ABEL, kde je nižší stupeň abstrakce a je více zřejmá souvislost mezi popisem a jeho fyzickou realizací. Jak vyzrávají návrhové systémy, přebírají na sebe stále větší díl potřebné dovednosti a „chytrosti“, dříve nezbytných u konstruktéra. Tyto vlastnosti samozřejmě bude konstruktér nadále potřebovat, budou však nasměrovány poněkud jinak než dříve.

Vztah k jazyku VHDL: Jak již bylo uvedeno, vedle jazyků Verilog a SystemVerilog (v tomto odstavci budeme o obou dále mluvit jako o Verilogu) se setkáme také s jazykem VHDL, který má podobné použití. Jazyk Verilog (a zejména SystemVerilog) má syntaxi podobnou jazyku C, zatímco jazyk VHDL odpovídá spíše jazykům skupiny Ada. Podobnost se u jazyka SystemVerilog projevuje především u pokročilejších jazykových konstruktů, jako jsou struktury nebo uniony. Verilog je aspoň ve svých základních konstruktech poněkud jednodušší a zápisy v něm jsou kratší než ve VHDL, kde zvláště práce s typy dat může začátečníkovi činit obtíže. Uvádí se, že z praktického hlediska jsou jazyky Verilog i VHDL zhruba stejně použitelné, i když zejména podpora rozsáhlých modelů se zdá být u SystemVerilogu dokonalejší, a že pokud návrhář zvládne jeden z nich, není pro něj velký problém přejít na druhý. O všech se říká, že je vcelku snadné naučit se jim, ale je obtížné zvládnout je mistrovsky. Pro běžného uživatele není aspoň z počátku příliš smysluplné usilovat o podrobné a aktivní zvládnutí obou jazyků Verilog i VHDL. Nejnápadnější rozdíl mezi těmito jazyky pravděpodobně spočívá v tom, že jazyk VHDL má značně přísnější pravidla syntaxe než Verilog. V textech zapsaných ve Verilogu se počítá s tím, že se simulátor nebo syntetizér bude snažit správně pochopit, co mu návrhář chce textem říci. V jazyku VHDL však mívají i drobné odchylky od předepsané formy zápisu za následek přerušení syntézy či simulace a hlášení chyby, zatímco ve Verilogu bývají častěji různé formy zápisu tolerovány. To může být pro uživatele Verilogu příjemné, ale může se také stát, že systém porozumí textu jinak, než jak to návrhář myslel, a ten pak může být překvapen výsledkem. Toto překvapení bývá tím nepříjemnější, čím v pozdější fázi práce na projektu se projeví. Příkladem typické chyby tohoto druhu je, když vynecháme rozsah v deklaraci proměnné, které se dále přiřazuje celočíselná hodnota – syntakticky je vše v pořádku, ale tato proměnná se pak chápe jako jednobitová, což neumožňuje rozlišit více hodnot než nula a nenula. Odhalení takové chyby nemusí být ve složitějších konstrukcích snadné. Lze také doporučit, aby se návrháři při psaní textu ve Verilogu příliš nesnažili experimentovat s jazykovými konstrukty a používali osvědčený styl zápisu, u něhož si ověří, jak mu systém rozumí, a pracovali s ním třeba i za cenu poněkud složitějšího vyjadřování, než by mohlo být nezbytné. Druhou možností je důkladnější ověření funkce simulací nebo jinými metodami, než jak je to nutné u jazyka VHDL, to však obvykle bývá pracnější cesta.

V dále uvedeném výkladu různých jazykových konstruktů se občas vyskytují slova a fráze jako „obvykle“, „měl by být“ a podobně. Tím je většinou myšlena skutečnost, že současně existující návrhové systémy jsou na různé úrovni vyzrálosti a inteligence. Systém, který je možno označit jako vyspělý, bude správně sestavené popisy schopen zpracovat a optimalizovat bez problémů. Ne všechny současné systémy to však zvládnou stejně úspěšně. Obecně lze říci, že čím blíže odpovídá popis fyzické realizaci popisované struktury v cílovém obvodu, tím pravděpodobnější je optimální výsledek syntézy. Například některé způsoby popisu s algoritmickým charakterem mohou působit při syntéze problémy, i když jsou syntakticky správné,

protože zapojení, které má být výsledkem syntézy, žádný algoritmický charakter ve skutečnosti nemá. Potíže tohoto druhu se vyskytovaly zejména u starších verzí programových nástrojů, u současných moderních verzí jsou již většinou překonány. Začneme-li však pracovat s novým systémem, bývá vždy vhodné zjistit rozsah syntetizovatelných konstruktů jazyka v dokumentaci systému, jako je například **[48]** nebo **[58]**, a ověřit si na jednoduchých příkladech, zda syntéza probíhá podle očekávání. Někdy se může stát, že zdánlivě jasný text je syntetizován do struktury, která obsahuje nadbytečnou logiku, případně její část může být nevhodná (mohou například vznikat problémy s hazardy) nebo zcela nefunkční. Lze sice čekat, že syntetizéry budou stále dokonalejší a podobných problémů bude čím dál méně, dovednost a zkušenosti konstruktéra však sotva kdy plně nahradí.

K přenositelnosti kódu je nutno připojit poznámku. Jazyk SystemVerilog je velmi rozsáhlý, a do značné míry to platí i pro jazyk Verilog, který byl původně určen především pro simulaci číslicových systémů. Jeho použití pro syntézu se začalo rozpracovávat později a využívá se při něm jen část jeho prostředků. V počátečních fázích tohoto směru využití jazyka Verilog byly vytvářeny jednoduché syntetizéry se značně omezeným rozsahem jazykových konstruktů, které jimi byly podporovány. Postupně, jak syntetizéry vyspívaly, se tento rozsah rozšiřoval. Proto je možno očekávat, že základní konstrukty jazyka budou různými syntetizéry zpracovány stejně, ale neobvyklé způsoby popisu mohou dávat různé výsledky při zpracování různými syntetizéry – obvykle sice ne z hlediska funkčního, což by byla vyloženě chyba systému (ale i ty se stávají), ale hlavně z hlediska kvality výsledků, tedy z hlediska spotřeby strukturních prvků, zpoždění, odběru z napájecího zdroje a podobně. Je také nutno počítat s tím, že různé syntetizéry podporují odlišný rozsah prostředků jazyka, což samozřejmě platí i pro SystemVerilog. Postupem doby, jak se budou syntetizéry zdokonalovat, se patrně bude rozšiřovat oblast jazykových konstruktů využitelných v syntéze, které budou dávat výsledky syntézy skutečně nezávislé na použitých návrhových systémech. Dále je nutno poznamenat, že syntetizéry používají často volnější pravidla syntaxe než simulátory. Hlásí-li například simulátor chybu, syntetizér někdy vydá pouze varování a pokračuje dále v syntéze, pokud může odhadnout úmysl konstruktéra.

I když tedy v principu by měl být kód zapsaný v jazycích Verilog a SystemVerilog přenositelný na různé systémy a cílové obvody, není tato přenositelnost úplná. Pro jednoduché konstrukce, kde se vystačí se základními logickými bloky, nečiní obvykle přenositelnost problémy. Ve složitějších konstrukcích, zejména pokud se v nich využívají speciální bloky typické pro současné obvody FPGA dovolující výrazně zlepšit vlastnosti výsledku, je přenositelnost omezená, a k dosažení optimálního výsledku musí mít konstruktér podrobnější znalosti i o funkci návrhových systémů a o struktuře cílových obvodů. Například v obvodech CPLD řady CoolRunner se vyskytují klopné obvody reagující na obě hrany hodinového signálu. Odpovídající popis (viz **odst. 2.6.9**) je syntetizovatelný pouze do těchto cílových obvodů. Podobné poznatky jsou v předkládané publikaci zmíněny jen okrajově. Podrobněji se s nimi může čtenář seznámit v katalogových listech obvodů FPGA a CPLD, v uživatelských příručkách a v podobných publikacích výrobců programovatelných obvodů.

Kromě zápisu kódu v jazycích Verilog a SystemVerilog bývá účelné zejména v hierarchických konstrukcích s menším rozsahem používat k popisu také schémata (hlavně bloková, která představují grafickou podobu strukturálního popisu). Tento popis může být výhodnější než textový strukturální popis zejména pro lepší přehlednost, nebývá však přenositelný do jiných návrhových systémů a u rozsáhlejších konstrukcí přestává být použitelný. K prohlížení schématu musíme použít editor z návrhového systému, nestačí k tomu např. běžné textové editory. V současné době většinou nebývá smysluplné popisovat schématem takové obvodové bloky, které je možno popsat textově behaviorálním stylem (schémata tohoto druhu se používala k popisu konstrukcí zejména pro obvody FPGA v době, kdy ještě nebyly k dispozici prostředky pro syntézu z jazyků HDL, a někteří starší konstruktéři jsou na tento způsob popisu zvyklí, i když jej lze dnes pokládat za překonaný). Výhodou textového strukturálního popisu je možnost použití některých vyšších jazykových konstruktů (například smyček, příkazů `generate` a podobně), které mohou textový popis výrazně zkrátit. V SystemVerilogu jsou zavedeny nové konstrukty – například rozhraní, které u rozsáhlých strukturálních popisů výrazně zlepšují jejich efektivitu a přehlednost. (Pojmy behaviorální a strukturální popis budou vysvětleny v **odst. 2.1.1**.)

V dalším textu se budou vyskytovat termíny kompilace, analýza a elaborace. Připomeneme stručně jejich význam. **Kompilace** (*compilation*) obvykle probíhá ve dvou fázích: analýza a elaborace. **Analýza** (*analysis*) představuje převod zdrojového textu či schématu do vnitřní formy popisu akceptované systémem,

kterou jsou schopny zpracovat navazující programové nástroje (simulátor, syntetizér) a která zpravidla není uživateli přístupná. Přitom se zpravidla provádí kontrola syntaktické správnosti (*parsing*) textu. **Elaborace** (*elaboration*) navazuje na analýzu: zpracovává se při ní hierarchická struktura projektu – vytvářejí se vazby mezi dílčími bloky konstrukce (komponentami) a jejich vloženími do vyšších jednotek (instancemi), vyčíslují se hodnoty parametrů a podobně. Výsledkem je tzv. netlist, který je pak podroben simulaci nebo syntéze. Některé nástroje obě tyto fáze slučují do jednoho procesu, u jiných jsou fáze více či méně odděleny, někdy se také názvy fází posouvají nebo zaměňují.

V dalších kapitolách budeme často používat pojem **signál**. Budeme tím zpravidla rozumět veličinu, jejíž hodnota se může v čase měnit, na rozdíl od konstant, parametrů a podobně. Signál může být skalární (jednobitový) nebo vektorový (vícebitový), o signálech však budeme mluvit i v souvislosti se složitějšími datovými objekty, jako jsou například pole nebo struktury.

Poznámka k označení `v,sv`: Příklady v dalším textu budou většinou psány v syntaxi SystemVerilogu, tedy v deklaracích budou uváděny proměnné typu `logic` místo typu `reg` z Verilogu a příkazy SystemVerilogu `always_comb`, `always_latch` a `always_ff` budou používány místo příkazů `always` (Verilog). Rovněž budou psány v syntaxi SystemVerilogu příkazy pro smyčky `for` a v definicích funkcí nebudou uváděny nadbytečné příkazové závorky `begin-end`. Tam, kde to bude účelné, budou příkazy `case` doplněny příslušným modifikátorem (`unique`, popř. `priority`). Pokud budou tyto texty po jednoduchém návratu k syntaxi Verilogu, tedy po náhradě typu `logic` typem `reg` (typem `wire` v deklaracích vstupních bran a u signálů, jimž se přiřazuje hodnota v příkazech `assign`, a také u signálů připojených k výstupům vložené jednotky ve strukturálních popisech), po náhradě tří dalších příkazů příkazem `always` (u prvních dvou z nich je třeba doplnit ještě citlivostní seznam), po úpravě syntaxe smyček (viz **odst. 2.6.6**) a funkcí a po vynechání modifikátorů odpovídat syntaxi Verilogu, budou označeny poznámkou `v*,sv`; bude-li stačit pouhá změna typů nebo nebude-li zapotřebí žádná úprava, nebude v označení hvězdička. Bude-li však syntaxe v SystemVerilogu od syntaxe ve Verilogu natolik odlišná, že by převod do syntaxe Verilogu byl příliš komplikovaný, popřípadě má-li text sloužit k ilustraci konstruktů, které nejsou platné ve Verilogu (například při použití uživatelských typů, struktur, rozhraní nebo pokročilých konstruktů pro zpracování polí), bude u nich poznámka `jen sv` (pokud platnost nebude zřejmá z okolního textu). Případné další rozdíly mezi oběma druhy syntaxe budou uvedeny individuálně v okolním textu nebo formou komentáře. Některé texty budou psány v syntaxi Verilogu-2001, a ty budou označeny poznámkou `v` (ty budou samozřejmě vyhovovat i syntaxi SystemVerilogu). Autor věří, že převod textů s poznámkou `v*,sv` do syntaxe Verilogu nebude čtenáři činit velké obtíže.

2.1.1 Způsoby sestavení modelu a styly popisu

Rozeznáváme tři základní **způsoby sestavení modelu:**

Postup **zdola nahoru** (*bottom-up*): nejprve se vytvoří dílčí bloky modelu (moduly, případně další konstrukční prvky, viz **odst. 2.2**) a ty se pak skládají do větších celků. Tento postup byl charakteristický pro sestavování číslicových konstrukcí z pevně zapojených číslicových obvodů, které představovaly bloky nejnižší úrovně, nebo pro vytváření popisu modelů pomocí schémat propojováním schematických značek základních bloků, což bylo typické pro počátky využívání obvodů FPGA.

Postup **shora dolů** (*top-down*): definuje se funkce navrhovaného systému jako celku. V něm se vyčlení bloky, jejichž funkce se specifikuje spolu s vzájemnou návazností jednotlivých bloků. U velkých konstrukcí mohou jednotlivé bloky zpracovávat různí konstruktéři. Tento proces pokračuje tak dlouho, až se na nejnižší úrovni získají dostatečně jednoduché bloky, jejichž funkci je možno popsat prostředky, jako je behaviorální popis v jazycích HDL na úrovni RTL (viz dále). Tak se zpravidla pracuje při syntéze s použitím moderních systémů CAD, kde starost o podrobné zapojení na ještě nižších úrovních přebírají návrhové systémy.

Přístupy zdola nahoru a shora dolů může být účelné v různých fázích zpracování modelu určité konstrukce kombinovat. Tyto způsoby sestavení modelu se označují jako **hierarchické**. Rozumně zvolená hierarchie usnadňuje orientaci v popisu modelu navrhované konstrukce a dovoluje opětné použití odladěných dílčích bloků. Jazyk Verilog podporuje hierarchičnost modelů, v SystemVerilogu je podpora hierarchie ještě značně zdokonalena – je zde zavedeno mnoho nových konstruktů, které usnadňují práci na rozsáhlých hierarchicky uspořádaných projektech.

Model **plochého typu** (*flat*) neobsahuje hierarchické členění. Představuje konstrukci jako jeden monolitický blok. Současné syntetizéry a optimalizační programy optimalizují takový model jako celek, zatímco u hierarchických popisů se zpravidla optimalizace provádí jen v rozsahu jednotlivých hierarchických bloků. Návrhové systémy proto před optimalizací často převádějí hierarchické modely na ploché, aby bylo možno provést optimalizaci v celém rozsahu. To však znamená, že orientace v takovém popisu, například nalezení a testování vnitřních signálů při simulaci, je obtížnější. Proto návrhové systémy obvykle dovolují zakázat rozvinutí hierarchického modelu do plochého, což může být výhodné ve fázi ladění a testování, a po odladění se povolením rozvinutí do plochého modelu může získat lépe optimalizovaný výsledek s lepšími parametry.

V souhlasu s terminologií výrobců návrhových systémů budeme označovat souhrn souborů týkajících se konstrukce vytvářené v návrhovém systému názvem **projekt** (*project*).

Příkazy v modulu mohou jeho funkci popisovat **behaviorálním stylem** (*behavioral style*) nebo **strukturálním stylem** (*structural style*). Behaviorální styl popisuje chování částí (komponent) modulu, zatímco strukturální styl představuje soupis komponent obsažených v modulu a jejich propojení (*netlist*). Pro pochopení funkce modelu (či jeho části) popisovaného behaviorálním stylem v podstatě stačí znát syntaxi jazyka, zatímco funkci komponenty ve strukturálním popisu může napovídat jen její název a ostatní údaje o komponentě musí uživatel znát (popřípadě si je najít v popisu komponenty). Behaviorální popis je tedy obvykle pro člověka srozumitelnější, a budeme jej zde, pokud to bude možné, používat. Použití strukturálního stylu je typické například u interních souborů generovaných návrhovým systémem, které jsou určeny pro další strojové zpracování (počítač nemusí popisu rozumět, na rozdíl od člověka mu stačí syntaktická správnost). Označení těchto stylů však není součástí definice jazyka a slouží jen pro orientaci uživatele. V jednom modulu můžeme styly libovolně kombinovat, pokud dodržíme syntaktická pravidla jazyka. Někteří autoři zavádějí ještě pojem **stylu popisujícího tok dat** (*dataflow style*), který zhruba řečeno spočívá v použití kontinuálních přiřazovacích příkazů (klíčové slovo `assign` – viz **odst. 2.5.1**), zde však budeme popis tohoto druhu počítat k behaviorálnímu stylu.

Setkáme se také s pojmem **behaviorální modelování** (modelování na **behaviorální úrovni**), čímž se rozumí modelování na vysoké úrovni abstrakce, kde například není ještě definována šířka dat a simulací se ověřuje chování takového abstraktního modelu. Modely určené k syntéze se nejčastěji popisují na **úrovni RTL** (*Register Transfer Level*), kdy se v modelu vyčlení paměťové prvky (registry) a kombinační obvody mezi nimi, a ty se popíší – nejčastěji behaviorálním stylem. Popis na úrovni RTL pak specifikuje přenos signálů mezi registry, včetně jejich případné modifikace kombinačními obvody. Ještě nižší úroveň abstrakce je u strukturálního modelování, kde již jde o popis na **úrovni hradel** (*gate level*) – jde obvykle o popisy generované návrhovými systémy, které slouží například k časové analýze.

Terminologické poznámky: Pojmy jako „behaviorální styl“ nebo „strukturální styl“ nejsou v oficiální terminologii Verilogu a SystemVerilogu (obsažené např. v dokumentech **[1]**, **[6]**) zavedeny, jsou to jen pomocné vyjadřovací prostředky, které mají za účel usnadnit orientaci ve způsobech popisu.

Podobně není v této terminologii zaveden pojem **komponenty** – tou budeme rozumět dílčí část konstrukce (například modul) vkládané strukturálním stylem do jiné jednotky. Místo něj se někdy užívá výraz „*child module*“; modul, do něhož je komponenta vkládána, se označuje výrazem „*parent module*“. Autorovi se nepodařilo najít vhodné české výrazy pro tyto termíny, a proto budeme pro vkládaný modul používat podobně jako v jazyku VHDL označení „komponenta“. Nicméně se i v anglické literatuře týkající se Verilogu a SystemVerilogu dosti často setkáme s označením *component* místo výrazu *child module*. V SystemVerilogu mohou být kromě modulů vloženy do vyšší jednotky také rozhraní a programy. V dokumentech SystemVerilogu se označení *parent – child* v tomto smyslu používá jen zřídka.

2.1.2 Lexikální jednotky

Zdrojový text ve Verilogu i v SystemVerilogu je tvořen sledem **lexikálních jednotek** (*lexical tokens*). Mezi tyto jednotky patří zejména:

- prázdné znaky;
- klíčová slova, atributy, direktivy pro kompilátor, názvy systémových funkcí a úloh;
- uživatelské identifikátory;
- operátory;
- číselné, řetězcové, strukturní literály a literály pro zápis hodnot polí;
- komentáře.

Prázdné znaky (*white spaces*), tedy mezery, tabulátory a znaky nového řádku slouží podobně jako v jiných jazycích jako oddělovače, nemají další význam a můžeme je používat podle libosti k rozčlenění dalších lexikálních jednotek v textu tak, aby byl text přehledný (s malými výjimkami, například je-li mezera součástí textového řetězce). Někdy mohou být tyto oddělovače vynechány, lze-li hranice lexikálních jednotek rozpoznat jinak (jak je to například před seznamem bran v textu `TXT2.1`).

Vyhrazená (klíčová) slova (*reserved words*, *keywords*) musí být ve Verilogu i v SystemVerilogu psána malými písmeny. Jsou to slova, jejichž význam je stanoven v definici jazyka. Jejich seznam je uveden v **odst. 1.1**. Editory návrhových systémů tato slova obvykle vybarvují, čemuž odpovídá dříve uvedená poznámka, že zde je budeme zapisovat tučně modrou barvou.

Jednotlivé verze Verilogu definují klíčová slova, přičemž vyšší verze přebírají klíčová slova nižších verzí. V SystemVerilogu je situace poněkud odlišná: je zde možno zvolit verzi jazyka, jejíž klíčová slova jsou v určitém bloku zdrojového textu rozeznávána (považována za platná klíčová slova), přičemž jiná slova jsou považována za běžné identifikátory, i když jsou ve vyšších verzích klíčovými slovy. K tomu jsou zde zavedeny direktivy `` `begin_keywords `` a `` `end_keywords ``, přičemž první z nich je následována specifikátorem verze. Platné specifikátory jsou: `1800-2012`, `1800-2009`, `1800-2005`, `1364-2005`, `1364-2001`, `1364-1995` (je ještě definován specifikátor `1364-2001-noconfig`, kde jsou z verze Verilog-2001 vynechána některá klíčová slova související s konfigurací, viz např. **[6]**). Tyto direktivy musí být umístěny mimo konstrukční prvky (jako jsou například moduly, podrobněji v **odst. 2.2**), čili v celém konstrukčním prvku musí být platná stejná sada klíčových slov. Není-li tímto způsobem verze specifikována, odpovídají rozeznávaná klíčová slova výchozímu stavu příslušné implementace.

Pragmata (*pragmas*) jsou podle standardu pro syntézu jazyka Verilog 1364.1-2002 **[3]** konstrukty, které nemají ve standardu jazyka definovaný sémantický význam, jsou však užívány pro řízení programových nástrojů, jako jsou syntetizéry, při zpracování zdrojových textů. Uvedený standard připouští jedinou formu zápisu pragmat ve zdrojových textech zapsaných ve Verilogu určených k syntéze ve formě **atributů** (*attributes*). Ve standardech Verilogu i SystemVerilogu slouží pro ohraničení atributů dvě speciální dvojice znaků `(*` a `*)`. Mnoho syntetizérů však připouští také použití pragmat zapsaných jako metakomentáře. V SystemVerilogu má termín *pragma* specifický význam – patří mezi direktivy; nebudeme jej však zde rozebírat. Slovem „atributy“ budeme také označovat vlastnosti objektů, jako je znaménkovost nebo rozsah signálů a podobně.

V publikaci **[3]** je uvedeno 15 atributů, které mají být podporovány syntetizéry Verilogu odpovídajícími standardu 1364.1-2002. O některých z nich se zmíníme později. Je však dovoleno, aby syntetizéry podporovaly ještě další vlastní specifické atributy, jejichž význam a způsob použití je nutno najít v dokumentaci k těmto nástrojům.

O **direktivách pro kompilátor** bude řeč později (**odst. 2.9**). Znakem dolaru (`$`) začínají názvy **systémových funkcí a úloh**, o nichž budeme mluvit v **odst. 2.7**.

Uživatelské identifikátory (uživatelské symboly, *user identifiers*, stručněji často jen identifikátory) jsou názvy sloužící k označení signálů, parametrů, konstant, funkcí a podobných objektů. Volí je návrhář (uživatel), a samozřejmě nesmějí být totožné s klíčovými slovy. Budeme také někdy mluvit v užším smyslu