基礎から学ぶ Verilog HDL & FPGA 設計

第15回
最終回：コンパイラの設計(2)

中野浩嗣，伊藤靖朗

デバイスの記事　ビギナーズ

前回(2009年2月号，pp.93-99)は，学習用のC言語風言語「MICROC」のコンパイラを設計した．今回はこれを拡張し，if文やwhile文，do文などの制御文をサポートするC言語風言語「TINYC」のコンパイラを設計する．特にやっかいなif文の構文解析についても，分かりやすく解説する．最終的に，数学の未解決問題を計算するTINYCプログラムをFPGA上で動作させる．(筆者)

● C言語と同様の学習用言語「TINYC」コンパイラを設計

前回(2009年2月号，pp.93-99)設計した学習用のC言語風言語「MICROC」では，if-goto文とunless-goto文に対応していました．今回のC言語風言語「TINYC」では，これらを廃止し，代わりに，通常のC言語のif文，if-else文，while文，do文に対応します．また，式で用いることのできる演算子も大幅に拡張します．TINYC言語の仕様を**表1**に示します．また式としては具体的には，**表2**の演算をサポートします．

コンパイラは，入力された高級言語プログラムに対して，字句解析，構文解析，最適化，コード生成を順に行い，ア

表1　C言語風言語「TINYC」の言語仕様

項　目	説　明
名前	英字で始まり，英数字が続く．変数名やラベル名に用いられる．
整数	10進数で表される整数．
ラベル	「ラベル:」の形で，goto文などの分岐先になる．
変数宣言	変数とその初期値を宣言する．型は16ビットの整数(2の補数)のみ．
goto文	「goto ラベル;」の形で，指定したラベルに分岐する．
if文	「if(式){文の列1}」または「if(式){文の列1}else{文の列2}」の形で，意味はC言語と同じ．前者は式の計算結果が0でない(真)とき，文の列1を実行する．後者は式の計算結果が0でない(真)とき，文の列1を実行し，0(偽)のとき文の列2を実行する．
while文	「while(式){文の列}」の形をとる．式の計算結果が0でない(真)とき文の列を実行する．これを式が0(偽)になるまで繰り返す．
do文	「do{文の列}while(式);」の形をとる．文の列を実行し，式の計算を行う．計算結果が0(偽)なら終了する．これを式の計算結果が0(偽)になるまで繰り返す．
halt文	機械語プログラムの実行を停止する．
out文	out(式)の形をしており，式を計算しその結果を出力する．
代入文	「変数 = 式 ;」の形をしており，式の計算結果を変数に代入する．
式	変数，整数，算術論理演算からなる式．算術論理演算は，TINYCPUの算術論理演算回路がサポートするものとする．

表2　TINYCの算術論理演算子

優先順位	演算子	トークン	結合	演算の内容
1	!		右	論理否定
	~		右	ビットごとの反転
	-	NEG	右	符号反転
2	*		左	乗算
3	+		左	加算
	-		左	減算
4	<<	SHL	左	左シフト
	>>	SHR	左	右シフト
5	>=	GE	左	大なりイコール
	<=	LE	左	小なりイコール
	>		左	大なり
	<		左	小なり
6	==	EQ	左	等しい
	!=	NEQ	左	等しくない
7	&		左	ビットごとの論理積
8	^		左	ビットごとの排他的論理和
9	|		左	ビットごとの論理和
10	&&	AND	左	論理積
11	||	OR	左	論理和

Keyword　C言語，コンパイラ，字句解析，構文解析，還元，if，else，while，do，Flex，Bison，コラッツの問題，Spartan-3A，Spartan-3E

センブリ言語プログラムを出力します．今回設計するTINYCのコンパイラでは前回のMICROCと同様，簡単のため最適化は行わず，字句解析，構文解析，コード生成のみ行います．コンパイラを作成するために，字句解析を行うプログラムを自動生成する「Flex」と，構文解析プログラムを自動生成する「Bison」の二つを使用します．構文解析プログラムの中でコード生成も同時に行うことにします．

● TINYCの字句解析定義を作成

まず，字句解析プログラム生成ツールFlexで使用するTINYCの字句解析定義を作成します．**リスト1**はTINYCの字句解析定義tinyc.lです．1行目から5行目が宣言部，7行目から27行目が文法規則部，29行目が追加のCプログラム部です．この字句解析定義は，MICROCの字句解析定義microc.lを少しだけ拡張したものです．以下，microc.lと異なる点を説明します．

宣言部で，int型変数のnを宣言しています．このnは，文法規則部で定義するトークンIF，WHILE，DOに通し番号を意味値として割り当てるために用います．文法規則部では，2文字以上の演算子，例えば，==や!=に対して，トークンEQやNEを返します．また，doやelseなどの予約語に対しても，DOやELSEを返します．

トークンの意味値は，2通りのデータ型をとります．一つは，yylval.sで，MICROCと同様に，文字列へのポインタです．もう一つは，yylval.nで，整数値をとります．トークンNUMBERとNAMEを返すときは，マッチした文字列をyylval.sに代入します．一方，if，while，doにマッチする場合は，トークンIF，WHILE，DOをそれぞれ返しますが，このときの意味値は，yylval.nに代入されます．代入される値は1から始まる通し番号になります．トークンがとる意味値の型宣言は，構文解析定義で行われます．

● TINYCの文法規則

TINYCの構文解析定義はMICROCのものに比べて複雑になります．まず，文法規則を文脈自由文法として記述しましょう．**図1**はTINYCの文法規則を文脈自由文法として記述したものです．MICROCの文法規則から追加・拡張されるのは，if文，while文，do文，式の四つです．

表2は，TINYCの式で用いられる演算子の優先順位と結合規則をまとめたものです．基本的にC言語のサブセットになっています．ここで用いられる2項演算はすべて左結合です．また，「!」などの単項演算は並列する場合に右側から計算されるので，右結合となります．

表2を基に，式exprについて構文解析の定義を行います．**リスト2**はその定義です．宣言部で，演算子の優先順位と結合規則を定義します．2文字以上の演算子の場合は，字句解析規則tinyc.lにより返されるトークンが定義に用い

リスト1　TINYCの字句解析定義（tinyc.l）

```
 1  %{
 2  #include <string.h>
 3  #include "y.tab.h"
 4  int n=0;
 5  %}
 6  %%
 7  [ ¥t¥n¥r]
 8  &&      {return(AND);}
 9  ¥|¥|    {return(OR);}
10  ==      {return(EQ);}
11  !=      {return(NE);}
12  ¥>=     {return(GE);}
13  ¥<=     {return(LE);}
14  ¥<¥<    {return(SHL);}
15  ¥>¥>    {return(SHR);}
16  do      {yylval.n=++n;return(DO);}
17  else    {return(ELSE);}
18  goto    {return(GOTO);}
19  halt    {return(HALT);}
20  if      {yylval.n=++n;return(IF);}
21  in      {return(IN);}
22  int     {return(INT);}
23  out     {return(OUT);}
24  while   {yylval.n=++n;return(WHILE);}
25  [0-9]+  {yylval.s=strdup(yytext);return(NUMBER);}
26  [a-zA-Z0-9]+
        {yylval.s=strdup(yytext);return(NAME);}
27  .       {return(yytext[0]);}
28  %%
29  int yywrap(){ return(1);}
```

宣言部　文法規則部　Cプログラム部

文の列	statements	→ statement | statements statement
文	statement	→ label | intdef | goto | if | while | do | halt | out | assign
ラベル	label	→ NAME :
変数定義宣言	intdef	→ int intlist ;
変数定義リスト	intlist	→ integer | intlist integer
変数定義	integer	→ NAME | NAME = NUMBER | NAME= - NUMBER
goto文	goto	→ goto NAME;
if文	if	→ if(expr){statements} | if(expr){statements}else{statements}
while文	while	→ while(expr){statements}
do文	do	→ do{statements}while(expr);
代入文	assign	→ NAME = expr ;
halt文	halt	→ halt ;
out文	out	→ out(expr) ;
式	expr	→ NAME | NUMBER | IN | ! expr | ~expr | -expr | expr + expr | expr - expr | expr * expr | expr && expr | expr || expr | expr & expr | expr | expr | expr ^ expr | expr << expr | expr >> expr | expr == expr | expr != expr | expr >= expr | expr <= expr | expr > expr | expr< expr | (expr)

図1　文法規則定義