ハードウェア記述言語トランスレータの設計

1H−4　櫻井 俊之　豊嶋 久道
神奈川大学工学部

1 研究の目的

通常ハードウェア設計に用いられるハードウェア記述言語はソフトウェア言語と大きく異なる。

- ハードウェア記述言語で用いられる端子は回路のノードを表しているためにソフトウェア言語の変数のように再帰的に利用できない。
- 端子自体に様々な属性があり、ソフトウェア言語のような自由度の高い記述は難しい。
- 制御構造が貧弱で、ソフトウェア言語のように条件によって動的に下位モジュールをを構成できない。

そこで本研究ではソフトウェア言語によってシミュレーションしたアルゴリズムをそのままハードウェア設計できるようにするため、また、ソフトウェア言語特有の動的で柔軟な記述を生かした設計を行えるようにするためにC++のクラス機能を用いて2進演算クラスを設計した。

また、それを用いて記述されたプログラムを論理合成システム PARTHENON で用いられる SFL (Structured Function description Language) に変換するトランスレータを設計した。

2 SFLについて

SFL とは NTT が開発した論理合成システム PARTHENON で用いられるハードウェア記述言語の一種である。

ハードウェア記述言語では VerilogHDL や VHDL が有名だが、SFL は記述対象を同期回路に限定して、動作メカニズムの記述を全て手続きの中で行うようにしたハードウェア記述言語である。

このため動作記述と接続記述を分離でき、設計の負担を軽減する事が出来る。また、従来のハードウェア記述言語と比べてオブジェクト指向化した言語でもあり、C++からの変換に向いているため、本研究では SFL を対象とする事とした。

3 C2SFLの構成

C2SFL とは本研究で作成されたハードウェア記述言語トランスレータの名称である。

C2SFL は以下のような構成となっている。

- 2進演算クラス (bit クラス)
  C++で端子を表現するのに使用する。
- 翻訳系
  C++の字句解析、構文解析、中間コードの生成およびラベル参照の解決をする。
- 実行系
  翻訳系が生成した中間コードを実行して SFL ファイルを出力する。

C++から SFL に変換する為には、端子の概念のない C++で端子を表現しなければならない。そこで C2SFL では C++で端子を表現する為に bit クラスと言う2進演算クラスを設計しそれを用いて記述する事とした。

Design of Hardware Description Language Translator
Toshiyuki SAKURAI,Hisamichi TOYOSHIMA
Faculty of Engineering, Kanagawa University

また、C++では存在するが SFL には存在しない概念もあるので使用できる文法も制限した。

以下に bit クラスと C2SFL で使用できる文法を示す。

bit(2進演算) クラスの仕様

- 宣言方法は
  1. bit x;
  2. bit x(8);

  とすると
  1. x という名前のビット幅1の端子
  2. x という名前のビット幅8の端子

  となる。
- 論理演算は2項演算の場合、お互い bit 型でないと演算できない。
  利用可能な論理演算子は表1のようになる。

AND	&
OR	|
EOR	^
NOT	~
連結	+
左シフト	≪
右シフト	≫
ビット切り出し	.elem(n)

表 1: 論理演算子

C2SFL で使用可能な文法

- 使用可能な変数の型は、int,long のみ。端子は bit のみ。
- SFL に変換する際に端子の属性が必要となるので予め bit クラスの中で属性を定義してある。属性は、input(入力端子) 、bit(内部端子) 、output(出力端子) の3つである。
- 使用可能な文は論理演算文と for 文、if 文のみである。
- 関数は使えるが、各関数の結果を返す方法は参照渡しとする。

4 C2SFLによる変換の流れ

C++の記述から条件分岐や繰り返し等の制御構造に応じて動的に SFL ファイルを出力するためには単なる1対1の変換では不可能である。

そこで C2SFL では C++の入力ファイルを翻訳系が一旦中間コードに変換して、その中間コードを実行系が逐次実行する構成にする事によって動的に SFL ファイルを出力する構成とした。

C2SFL に C++のファイルが入力されると、

- 翻訳系が構文解析を行い、中間コードを生成し、中間コード領域に格納する。
- 実行系は変数と端子の格納領域を利用して中間コードを実行する。スタック領域は、変数の演算や端子の論理演算の中間結果を格納するのに用いられる。また、次に実行される中間コードを示すプログラムカウンタとスタック領域の先頭を示すスタックポインタがある。

以上の処理の流れを図示すると図1のようになる。

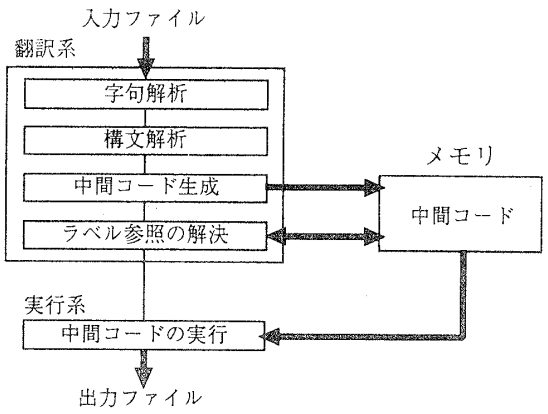

図 1: C2SFL による処理の流れ

5 C2SFL の変換アルゴリズム

通常、ソフトウェア記述言語の変数には次のような属性がある。

- 変数の型
- 値

しかし、ハードウェア記述言語の端子には次のような属性がある。

- 端子の型 (入力端子、出力端子、内部端子)
- ビット幅

これを単純に変換するだけではソフトウェア記述言語特有の再帰的な変数の利用に対応できないので、C2SFL では端子の属性に

- 代入された回数

を追加して、中間コードが利用する端子の格納領域に格納する事にした。そして、内部端子が代入される度に自動的に別の端子名をつけて再帰的に端子を使わせないようにしている。

しかし、入力端子や出力端子は、C++の関数の引数のように決まっているので内部端子と違い代入できないようにして構文解析の段階でエラーを出すようにした。

また、C2SFL では、端子と変数が共存している為に変数の演算と端子の論理演算を各々のスタック領域で実行するようにしている。図 2に実行系の仮想マシンの構成を示す。

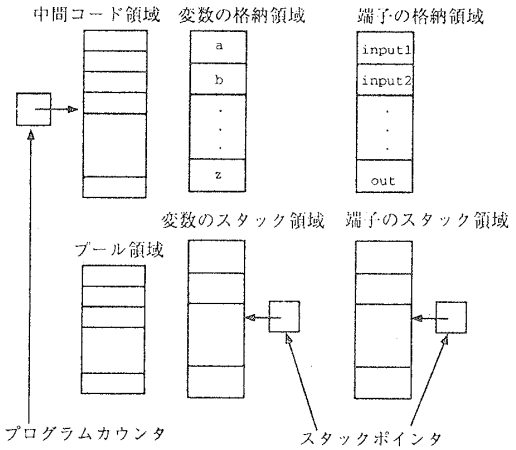

図 2: 実行系の仮想マシンの構成

端子のスタック領域では端子名を文字列としてスタックし、論理演算をする毎に論理演算子をつけてスタック領域に格納し、代入されるとプール領域に格納する。

変数のスタック領域は変数の数値をスタックし、演算をする毎に演算結果をスタック領域に格納し、代入されると変数の格納領域に格納する。図 3に C2SFL による簡単な変換の例を示す。

```
int i,n=3;
input in1,in2,c_in;
bit t_out,c_out;
output sum;

fa(in1.elem(0),in2.elem(0),c_in,t_out,c_out);
c_in=c_out;
out=t_out;
for(i=1;i<n;i++){
    fa(in1.elem(i),in2.elem(i),c_in,t_out,c_out);
    c_in=c_out;
    out=t_out+out;
}
sum=out;
```

↓ C2SFL

```
t_out=fa1.do(in1<0>,in2<0>,c_in).out;
c_out=fa1.do(in1<0>,in2<0>,c_in).cout;
c_in_tmp1=c_out;
out=t_out;

t_out_tmp1=fa2.do(in1<1>,in2<1>,c_in_tmp1).out;
c_out_tmp1=fa2.do(in1<1>,in2<1>,c_in_tmp1).cout;
c_in_tmp2=c_out_tmp1;
out_tmp1=t_out_tmp1 | out;

t_out_tmp2=fa3.do(in1<2>,in2<2>,c_in_tmp2).out;
c_out_tmp2=fa3.do(in1<2>,in2<2>,c_in_tmp2).cout;
c_in_tmp3=c_out_tmp2;
out_tmp2=t_out_tmp2 | out_tmp1;

sum=out_tmp2;
```

図 3: 変換の例

図 3は全加算器モジュール (fa) を用いて桁上げ循環加算器を構成している例である。これを見ると C2SFL が自動的に端子名を割り当てている事がわかる。

6 むすび

本研究では、C++から SFL へのトランスレータを設計した。これにより C++の記述から直接的に SFL 記述を得る事が可能になった。また、ソフトウェア言語の動的な条件分岐構造や繰り返し構造を用いて設計できる為、柔軟な記述が可能になり設計の負担が軽減された。

今後の課題としては、ハードウェア特有の概念である順序回路、制御入力端子等をサポートし、より実用性の高いツールにする事である。

参考文献

[1] 五月女 健治, “yacc/lex プログラムジェネレータ on UNIX ”, 啓学出版 (1992)

[2] 中村行宏, 小野定康, ”ULSI の効果的な設計方法”, オーム社 (1994)

[3] Niklaus Wirth, ”Hardware Compilation: Translating Programs into Circuits”, IEEE Computer Magazine,pp25-31,June,1998