設計自動化　45—6
（1988．12．16）

ハードウェア記述言語CDTとその設計支援への応用

Hardware Description Language CDT and It's Applications to Computer Aided Design

中川　圭介
Keisuke NAKAGAWA
電気通信大学　情報工学科
Department of Computer Science and Information Mathematics
The University of Electro-communications

あらまし　計算機などの論理回路を記述し設計，製作を行う目的で，ハードウェア記述言語を設計し，それを使った設計システムを開発している．CDT はプログラミング言語 Pascal に似た記法を持ち，並列実行を記述できる．また，動作記述とともに，その部分集合である CDT-L を使い，回路記述を行なえる．したがって，設計の過程は，CDT で書かれたプログラムを CDT-L で書かれたプログラムにすることになる．CDT-L の記述は，ゲート，フリップフロップ，そしていくつかのくりかえし構造を記述する機能によってなされる．CDT，CDT-L の機能，設計システムの概要とともに，シミュレーションについても少し述べる．

Abstract. In this paper, a hardware description language CDT and a design system using it are described. CDT is a language whose form is rather similar to Pascal, and has features for describing parallel execution of statements. The behavioral descriptions of the circuits are written using all it's features, while their circuit descriptions are written using it's subset CDT-L, which include only logical operations, gates, flip flops and some simple features for expressing repeated structures. So, to design is to transform programs in CDT to programs in CDT-L. We present the features of CDT-L and CDT, the design system which we are now developing, and at the end, discuss about simulation briefly.

1．はじめに

計算機などの論理回路の設計と製作に使う目的で，動作から回路の構造までを記述する言語 CDT を設計し，それを使った設計支　システムを製作している．以下に，回路記述言語 CDT-L，CDT，設計の過程，シミュレーションの順に概要を示す．

2．回路記述言語 CDT-L

次に示すプログラムは，図1.1．の回路の記述である．ここには circuit 名前；変数の宣言；複合文．という基本の書式が示してある．

```
1  circuit R1;
2  input s,d;
3  output q;
4  module DFF: f;
5  (f.D:=(f.Q AND NOT(s)) OR (d AND s);
6   q:=f.Q).
```

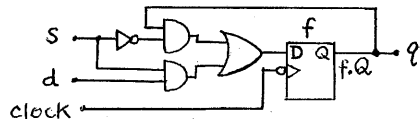

図1.1　R1の回路図

構造を代入文だけで表わすと記述が大きくなるので，くり返し構造の記述と階層構造の記述のための機能が重要である．

2.1. 繰り返し構造の記述

CDT-L の変数は基本的に1ビットを表わし，演算も1ビットのデータにたいする論理演算しか許されていないが，2進数を表わす変数などのために，ビット配列が記述できる．さらに，配列の各々のビットに同じ演算を行うときのために for 文が導入され，その制御変数には，整数を利用できるようにした．たとえば，nビットのレジスタは，次のように書ける．

```
1  circuit R(n);
2   input d[0..n-1], s;
3   output q[0..n-1];
4   module DFF: f[0..n-1];
5    (ns:=NOT(s);
6     for i:=0 to n-1 do
7      (f[i].d:=(f[i].Q AND ns) OR (d[i] AND s);
8       q[i]:=F[i].q)).
```

5行目と7行目に現われる ns は中継のための変数で，記憶の能力を持たない．

for 文の意味は，7，8 行にある構造が n 回くりかえして表われるということである．すなわち，実際の回路に展開したときには，2n 行の並列実行される代入文となる．

また，回路の名前とともに，長さをパラメータとして書くことで，柔軟な記述ができる．

2.2. モジュール

回路の構造を階層的に表現するために，モジュールによって部分の構造を記述できるようになっている．その表現の基本形式は

```
define module 回路名;
    変数の宣言;
    構造の記述;
```

であり，circuit の場合と同様である．モジュールを使った例として，2つのレジスタを使った回路 DR(n) を示そう．

```
1  circuit DR(n);
2   define module R(n);
3    input d[0..n-1], s;
4    output q[0..n-1];
5    module DFF: f[0..n-1];
6     (ns:=NOT(s);
7      for i:=0 to n-1 do
8       (f[i].d:=(f[i].Q AND ns) OR (d[i] AND s);
9        q[i]:=F[i].Q));
10   input d[0..n], s;
11   output qa[0..n-1];qb[0..n-1];
12   module R(n):a, b; DFF: f;
13    (f:=NOT(f) AND s;
14     for i:=0 to n-1 do
15      (a.d[i]:=d[i]; b.d[i]:=d[i];
16       qa[i]:=a.q[i]; qb[i]:=b.q[i])).
```

このプログラムの2～9行が n ビットレジスタの定義で，それを使い，2つの n ビットレジスタ a と b が 12行目で宣言され，実際に存在するようになる．3，4行目の宣言は主回路とのインタフェースで，主回路の構造内では，a.s，b.s，a.d[i] などと回路の名前と一緒に書かれる．図1.1の DFF はシステム内にすでに定義されたモジュールとして扱われていたのである．

2.3. その他の機能

高速の加算器などは，基本的に木の形をしているので，これらの回路を効率よく表現するため，モジュールを再帰的に定義できるようにした．すなわち，自分自身をモジュールの宣言に使うことを許している．同時に，パラメータに関する判断をする if 文も利用できる．

3. ハードウェア記述言語 CDT

CDT は以下に示す特徴を持つ記述言語である．

(1) 基本の型は，ビット（bit），整数（integer），文字定数であり，これらを要素とする配列，集合を定義できる．

(2) ビット，ビット配列にたいして，算術演算，論理演算，連結（concatenation），抽出（extraction）が定義されている．整

整数はビット配列に準ずる。

(3) 制御構造は、while文, repeat文, for文, if文, case文などを使い構成する。

(4) 複合文は、並列に実行されるものと逐次実行されるものがあり、前者は複数の文を"("と")"で囲むことにより、後者はbeginとendで囲むことによって表現する。

(5) 全体の形式はCDT-Lと同じである。

このあとにつづく各々の節でCDTについて概説するが、CDTの文の意味は、CDT-Lの文で表わすことによって説明する。

3.1. 変数とその宣言

変数は名前によって表わし、inputなどの種類、型、変数名をつづけて書いて宣言する。たとえば、

input bit : d[0..n−1], s;

は、nビットのビット配列dと1ビットの変数sを宣言する。また、フリップフロップやレジスタを表わすための変数の種類として、storeがある。

store bit : a[0..n−1], s;

はnビットのレジスタaとフリップフロップsを表現する。

なお、bit: は省略してもよい。

3.2. 式

式は定数と変数を演算記号+, −, ×, ÷, AND, OR, NOT, @ (連結)で結んで構成する。演算順序は、演算の強さ、( )によってきまる。

2つの変数aとbの連結a@bは、aを上位にしてaとbとを並べたものである。

また、配列aの2〜7ビットをとりだして作った6ビットのデータは、

a[2..7]

である。この抽出演算[..]は、式にも適用できる。

式に含まれる演算記号は、それを計算する論理回路モジュールとなり、式はこれらのモジュールを実行順に結合した回路となる。たとえば、

0@r[0..n−2]

はrの1桁右シフトであるが、rを入力とする1桁シフト回路を表わす。また、

0@(a+b)

は、aとbがともにnビットの整数であれば、2つのnビットの2進数を加える回路の後に、加算回路からのn+1ビットの出力を入力となるように1桁右シフト回路をつないだ回路として実現されると考える。各回路モジュールの出力は、精度を保持するために必要な桁の配列として宣言される必要がある。

3.3. 文

文には、代入文, if文, case文, for文, while文, repeat文, wait文, 空文があり、これらで書かれたプログラムが制御回路と演算回路の構造をきめる。式と代入文が演算回路を、複合文が代入文に相当する指令を出力する順序回路を記述しているとおおまかに考えてよい。

3.3.1. 代入文

代入文は

変数:=式;

の形式で表わされるが、左辺がstore変数であれば1単位時間(これを1τと呼ぶことにする)で実行される。そうでなければ、0単位時間かかる。すなわち、遅れがないものと見做す。

逐次実行される複合文

begin s1; s2; ··· ; sn end;

の実行時間はS1～Snの実行時間の和である。一方，並列実行される複合文

(S1; S2; ---; Sn)

の実行時間は，S1～Snのうち最長の実行時間に等しいと考える。

3.3.2. 分岐

分岐はif文とcase文によって表わす。これらの文自体には遅れはなく，それに含まれる代入文によって遅れ時間がきまる。

```
if c then S1 else S2;
```

は，S1かS2の遅れ時間に等しく，

```
if c then S1;
```

の場合の遅れ時間は，cが真ならS1の遅れ時間に等しく，cが偽なら0である。

case文

```
case c of
    c1: s1;
    :
    ck: sk;
end
```

は，条件cの値によって，c1～ckのうちの何れか1つを実行する。ci: siを分岐要素と呼ぶことにする。

分岐と関連して，wait文と空文（continue）が利用できる。空文は1τの遅れの生成に，wait文はif文の中で，くりかえしを表現するために使われる。たとえば，

```
if c then S1 else wait;
```

は，cが真ならS1を実行してif文を終るか，cが偽であれば1τの遅れの後にif文を再実行することを意味する。

3.3.3. いろいろな表現

1つの論理回路は，これまでに述べた機能を使っていろいろに表現できる。たとえば，2.2節で例に示した回路DR(n)は，さらに2つの異なるレベルでの表現が可能である。

まず，動作記述として，次のものが考えられる。

```
1  circuit DR(n);
2  input d[0..n-1], s;
3  output qa[0..n-1], qb[0..n-1];
4  store a[0..n-1], b[0..n-1];
5   (repeat
6     begin
7      if s then a:=d else wait;
8      b:=d
9     end
10    until off;
11    repeat
12    (qa:=a;
13     qb:=b)
14    until off).
```

これがCDTの記述の中で最もレベルの高いものであり，2.2節のCDT-Lの記述は最もレベルの低いものと考えられる。

なお，上記のプログラム中のrepeat --- until offは，---の部分をスイッチが入っている間くりかえし実行することを意味している。

動作記述とCDT-Lによる回路記述の間に，もう1つの記述がある。たとえば，

```
1  circuit DR (n);
2  input d[0..n-1], s;
3  output qa[0..n-1], qb[0..n-1];
4  store a[0..n-1], b[0..n-1], state;
5   (repeat
6     case state of
7     0: if s then       (a:=d; state:=1 )else state:=0;
8      1: (b:=d; state:=0)
9     end
10    until off;
11    repeat
12    (qa:=a;
13     qb:=b)
14    until off).
```

がDR(n)にたいするもので，順序回路記述と呼べるものである。すなわち，ここでは状態を表わす変数stateが導入され，逐次実行される複合文は見られない。各状態の動作はstateを条件とするcase文中の分岐要素によって表わされ，状態の変化は状態変数への代入文によ

って記述されている．5～　行の repeat 文が図3.1　の状態遷移図を表現していることはすぐにわかる．

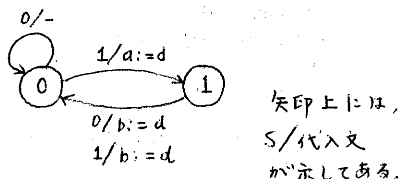

図3.1. DR(n) の遷移図

さらに，store 変数を2進符号化し，それぞれのビットに DFF を使えば，スイッチング理論の与えてくれる方法により回路記述へ移行できる。

それで，if 文を再びとりあげ，順序回路記述で表わしてみる．

まず，

```
if c then s1 else s2; s3;
```

が逐次実行されるときには，case 文に，

```
  i: (if c then s1 else s2; state:=i+1);
i+1:  s3;
```

を追加する．

3.3.4. while 文と repeat 文

while 文は if 文を使って表現できる．たとえば，

```
while c do begin s1;s2;s3 end;
s4; s5;
```

は，s1～s5 が代入文であれば，

```
   i: if c then (s1; state:=i+1);
           else (s4; state:=i+4);
i+1: (s2; state:=i+2);
i+2: (s3; state:=i);
i+3: (s4; state:=i+4);
i+4: (s5; ---       );
```

となる．

repeat 文も while 文と同様にして if 文を使って表現される．なお，repeat 文の書式は

```
repeat 複合文 until 条件式;
```

である．

3.3.5. for 文

for 文は，くりかえされた文が逐次実行か並列実行かによって，2つの形式がある．

まず，

```
for 変数:=初期値 to 最終値
    do 文;
```

は，初期値値から最終値までのくりかえしで実行される複数個の文は，並列に実行される．

また，

```
for 変数:=初期値 to 最終値
    do serially 文;
```

は，直列実行される．

4. 設計の過程

与えられたプログラムを CDT-L のプログラムに書き直すことが論理設計であると考え，その過程を議論する．

まず，全過程を次の段階に分けて考える．

(1) 演算回路と制御回路への分離
(2) 制御回路の設計
 (2.1) 順序回路記述の作成
 (2.2) CDT-L 記述の作成
(3) 演算回路の設計
 (3.1) 構造の決定
 (3.2) 各モジュールの順序回路記述の作成
 (3.3) CDT-L 記述の作成

以下につづく節でCDT-L記述の作成を除く各段階について概説するが，理解を助けるため，DR(n)より複雑な例を考える。

```
circuit ps(n);
  input bit: a[0..n-1]; s;
  output bit: z;
  store bit: r[0..n-1]; c: integer;

   begin
     repeat
       begin
         if s then (r:=a; c:=0) else wait;
         while c<=n-1 do (r:=0@r[0..n-2]; c:=c+1; z:=r[n-1])
       end
     until off
   end.
```

この回路は時刻tにs=1とすると，入力aを受け取り，時刻t+τ, t+2τ, ..., t+nτの(クロック)時間にa[n-1], ..., a[0]をzから出力する，並列→直列変換回路である。

4.1. 演算回路と制御回路への分離

ここで行なう仕事は，式の計算と代入を行なう部分をまとめて演算回路モジュールを定義し，それと主プログラムとのインタフェースを記述することである。ps(n)については，上述したプログラムから，右上に示すプログラムを求めることが目的である。

これを実行する手順は次のようになる。

```
circuit ps(n)
  define set control= { c0, c1, c2 };
  define module am(n);
   input bit: a[0..n-1]; com: control;
   output bit: z,.q1;
   store bit: r[0..n-1]; c: integer;
     (repeat
       (case com of
           c1 : (r:=a; c:=0);
           c2 : (r:=0@r[0..n-2]; c:=c+1; z:=r[n-1]);
        end;
        q1:=(c<=n-1) );
      until end );

  input bit: a[0..n-1]; s;
  output bit: z;
  module A: am(n);

 (repeat
   begin
     if s then A.com:=c1; else wait;
     while A.q1 do A.com:=c2;
   end;
  until off;
  repeat
   (A.a:=a; z:=q1);
  until off ).
```

(1) ( )で囲まれた代入文からなる複合文を見出し，それぞれをf1, f2, ..., fkとする。

(2) 集合 control = {c0, c1, ..., ck} を定義する。

(3) 演算回路モジュールamの定義

(3.1) モジュール名を書く。

(3.2) 原プログラムのstore変数の宣言をコピーする。(そして原プログラムから削除する。)

(3.3) 式中に現われる入力変数の宣言をコピーする。そして，

com: control

を追加する。

(3.4) 代入文の左辺の出力変数の宣言をコピーする。條件式に対して，出力変数q1, q2, ... を宣言する。

(3.5) 以下の複合文を書く。

```
(case com of
     c1:  f1;
     c2:  f2;
     ...
     ...
     ck:  fk
  end ;
  q1:= ex1 ;
  q2:= ex2 ;
  ...
  ...
  qm:= exm ) ;
```

(4) module A: am を宣言する。

(5) f1, f2, ... を A.com:=c1, A.com:=c2, ... に書きかえる。

(6) 條件式を，A.q1, A.q2, ... に書きかえる。

(7) Aの入力と原入力，Aの出力と原出力を結ぶ代入文を書く。

4.2. 制御回路の設計

順序回路の設計は，逐次実行と並列実行の解析をして必要な状態を見出し，順序回路記述を作り，それをもとに回路記述を作成することである．

この段階の作業により，PS(n) の制御を表わす主プログラムの複合文は，以下のようになることが期待される．

```
—
(repeat
  case state of
   0: if s then (A.com:=c1; state:=1)
        else state:=0;
   1: if A.q1 then (A.com:=c2; state:=1)
        else state:=0
  end;
until off;
repeat
  (A.a=a; z:=q1);
until off);
—
```

これを行なう手続きの内容の概要は3.3節からわかるように，以下のようになる．

(1) 順序回路記述の作成

(1.1) 3.3節 の方針に従って必要な状態を見出す．

(1.2) 集合 $S=\{0,1,\cdots,n-1\}$ を定義する．ただし n は必要な状態の数である．

(1.3) state:S を store 変数として宣言する．

(1.4) case 文を作成する．

(2) 回路記述の作成

(2.1) ビットまたはビット配列でない変数に2進符号を与える．

(2.2) 1ビットの変数にたいする式に分解する．

(2.3) 論理式を生成する．

この段階では，いくつかの問題点がある．まず，並列実行複合文にwhile 文のような実行回数が予測できない文が含まれている場合には，すべての文の終了（して停止したこと）を検出する回路を付加する．また，複合文の中の複合文の処理は，その部分の展開をして状態を定義するが，その部分を分離する方法もある．

4.3. 演算回路の設計

この章の最初に示したように，この段階は3つの段階に分解できるが，最後の段階は前節の(2)と同じであるので，省略する．

(1) 構造の決定

構造の決定はこの段階で最も難しく，また大切な部分である．もともと，各store変数が左辺にある式を集め，それらを独立に回路を作っても，正しい動作をするが，それでは無駄が多いため，よりよい構造をさがすのが仕事である．ここでは，次のような方法によっている．

(1.1) 代入文を，左辺が同じstore変数であるようなグループに分ける．また，条件式はそれぞれ別のグループとする．

(1.2) 主プログラムを解析して，同時に実行できない組合せをきめる．

(1.3) 同時に実行できる組合せについて，式を同じ回路で計算するのが得かどうかを調べ，グループの併合を試みる．このとき，加算回路の兼用など，高価な回路の兼用から試みる．このとき，演算は，単項演算か2項演算であるので，基本構成は，1個または2個の選択回路が演算モジュールの入力に，演算モジュールの出力に選択回路がつながった形となり，そこからレジスタなど別の変数へつながる．

(2) モジュールの順序回路記述の作成．

構造がきまると，4.1と同様の方法で演算モジュール，選択回路を1つずつモジュールとして記述する．そして最後にモジュール間の結線を表わす代入文が主プログラムとして残る．

選択回路は段階(3)で1ビットの記述に分解され回路記述が求められるが，演算回路は，システム内に定義をしておかざるをえない．

5. シミュレーション

我々の処理システムは，パーサーから得られる構文木をもとに，設計プログラム，シミュレータを開発している。

CDT は並列実行が行なえるので，計算の順序は，データの流れを解析して決めなければないが，現在は，木と実行順序表とを使ってシミュレーションを行なう方法をとっている．従って，実行はインタプリタが行なうことになり，あまり速くない．現在我々の使っているシミュレータは，順序回路記述までを扱うことができるが，簡単な計算機の場合で，1秒数十命令である．（SUN3-50上）この速度でも，モジュールの検証，アーキテクチャに関する実験を通して，言語，システムの有効性を確かめることはできるが，やはり，機能の拡張とともに，高速化が課題である。

6. おわりに

このシステムの開発は，CDT-L とそのシミュレータの開発に始まり，CDT への拡張，そして現在設計システムの開発を行なっている。これまでの使用経験から，CDT は，設計システム用の言語として十分使えると考えている。また，機能はそれ程大きくなく，小人数でのシステム開発用という目的も満たされているように考えている．

さいごに，CDT-L の処理系を開発した小野田実，鈴木英泰，CDT のシミュレータを開発した土居晋三の諸氏は，その製作と利用を通じて CDT の設計に大きな貢献があった。ここに記して感謝する。