ハードウェア動作記述言語：ALHARD（2）
-- 言語仕様とC言語への変換処理 --

2U-4

小島　泰三*　杉本　明*　阿部　茂*　鶴　薫**　加藤　幸男**
*三菱電機(株)中央研究所　**同コンピュータ製作所

1．はじめに

ハードウェアの仕様・動作レベルシミュレーションを目的として、ハードウェア動作記述言語 Alhard の開発を行った[1)2)]。本稿では、言語仕様の特徴とC言語への変換処理について述べる。

2．言語仕様の概要

Alhard の言語仕様の設計では、まず、シミュレータ作成用の問題向き言語としてC言語の拡張を行い、その後、ハードウェア動作記述容易化のため、標準的なハードウェアリソースや、これに対する構文を用意した。以下に主な特徴を述べる。

(1)オブジェクト指向機能

ハードウェア記述のモジュール性向上のため、オブジェクト指向方式のクラス定義や継承、メソッドなどの導入をおこなった。継承は単一継承であり、メッセージとメソッドは、C言語変換時に静的に束縛する。シンタックスは、動作表現はC言語に準拠することとし、クラス定義はドキュメントとしての可読性を考慮して、図1の例のようにした。図1では、継承（inherit）や、このクラスから生成されるオブジェクト（ハードウェアブロック）が使用するリソース（resource）、オブジェクトの動作を定義するメソッド（method）の記述を例示している。このほか、定数や変数の定義、ビット位置やメモリのバイト順序の指定などがオブジェクトの属性として記述できる。

(2)デーモン機能

あるイベントが発生した時に行うデーモン動作を記述する機能として、event 型データと when 文の導入を行った。図1のクラスが継承している SingleClockScheme の記述例を図2に示す。図2行2 に clock が event 型として定義されている。また、図1の fetchメソッド（行16-24）にデーモン記述の例を示す。

clock に対して、when 文により実行文を記述すると（図1行17)、この実行がデーモンとして clock イベントに登録される。後に clock イベントが発生すると、

```
class Example is                                    -- 1
inherit SingleClockScheme;                          -- 2
resource                                            -- 3
    memory    MEM[#4000](32);                       -- 4
    register  PC(18),                               -- 5
              MBUF(32),                             -- 6
              IR(32),                               -- 7
                  OP(8) == IR(31,24),               -- 8
                  DR(24)  == IR(23,0),              -- 9
              ACC(32),                              -- 10
              C(1);                                 -- 11
method                                              -- 12
    init is                                         -- 13
        fetch;                                      -- 14
        end                                         -- 15
    fetch is                                        -- 16
        when(clock) {                               -- 17
            MBUF = MEM[pc];                         -- 18
            PC += 4;                                -- 19
            if(MBUF(31)) fetch;                     -- 20
            IR <= MBUF;                             -- 21
            exec;                                   -- 22
        }                                           -- 23
        end                                         -- 24
    exec is                                         -- 25
       when(clock) {                                -- 26
           decode(OP) {                             -- 27
               #00: ACC := DR;                      -- 28
               #01: ACC(23,0) = DR;                 -- 29
               #02: C @ ACC = ACC + DR;             -- 30
          ....
    ....
end Example
```

図1　Alhard 言語の記述例

```
class SingleClockScheme is                          - 1
resource  event clock;                              - 2
method                                              - 3
    step is                                         - 4
        signal(clock);                              - 5
        end                                         - 6
end SingleClockScheme                               - 7
```

図2　クロック操作の記述例

clock に登録されたすべてのデーモンが起動され、図1行18-22の文が実行される。行20のfetchメソッドの再帰的呼び出しは、次の clock イベント発生に対する新たなデーモン登録となる。この場合、行22 の exec の呼び出しも同様に exec メソッド内のデーモン登録を行う。従って、次の clock イベント発生時点には、2つのデーモンが実行される（同期並行動作の表現、ただし、シミュレーションでは登録順に実行を行う）。行21は特殊な遅延転送デーモンの登録である。現在のイベントによるす

ALHARD: A Behavioral Hardware Description Language (2)
Taizo KOJIMA, Akira SUGIMOTO, Shigeru ABE, Kaoru TSURU, Sachio KATOU
MITSUBISHI ELECTRIC CORPORATION

べてのデーモン実行が終了した後、右辺から左辺への値の代入が行われる。

イベントの発生は、event 型変数に対する signal 操作により起こる。本図の例では、図2の step メソッドにより clock に対して signal操作が行われている（図2行5）。このほか、event 型の巡回リストとして timer 型があり、順序付けられた遅延の記述に使用される。

ハードウェア動作記述上のクロックスキームには種々のものが有り得る。このため、クロックを基本データタイプとはせず、event と timer を組合わせることにより、対象ハードウェアやシミュレーションの目的に応じたクロックスキームを構成できるようにしている。

(3) C言語演算子の拡張

任意ビット幅のビットストリングを表現するため、新たに bint(bounded int)型を導入した。bint 型データに対する演算はそのデータのビット幅に従って行われる。C言語の演算子が（++と--を除いて）すべて使用できるほか、表1に示す演算子を追加した。現在の実現では、255 ビット幅までの演算をサポートしている。また、C言語の int 型との型変換も定義しているので、int 型と bint 型を自由に混在させることができる。

(4) ハードウェアリソース

標準的なハードウェアリソースとして、レジスタやバス、メモリを用意している。レジスタは、bint 型データと、2種類の event 型データ（読出し時に発生する参照イベントと書込み時に発生する更新イベント）を組み合わせたものである。また、メモリは、シミュレーション時に32ビット幅までのアドレスが扱えるよう、所定のサイズを越えるとハッシュテーブルにより実現している。このほか、レジスタ内のビットフィールドやフィールド間の結合を仮想的なレジスタとして扱うための仮想レジスタ（図1行7-8）や、バイトメモリをワードメモリとしてアクセスする場合に使用する仮想メモリなどがある。

(5) 動作記述用構文

C言語の構文はすべて使用できるほか、switch 文をより簡単な表記にしたdecode 文を用意している（図1行27）。また、レジスタなどの参照時、更新時のデーモン登録構文として、ifget 文や ifset 文が使用できる。例えば、A,B,C をレジスタとする。

```
ifget(A) A = B + C;
```

により A の参照デーモンを設定すると、A の値が読出される毎にまず B と C から A を計算してその値が返される（on demand evaluation)。また、逆に、

```
ifset(B,C) A = B + C;
```

により B と C に更新デーモンを設定すると、B や C に値が書込まれると A の値も更新される（forward propagation)。

(6) 部品化支援機能

新たなハードウェアリソースを部品として定義するため、クラスの記述にポートを定義することができる。ポートとして定義されたリソースは、インスタンス生成時にパラメータとして与えられるリソースと同一化され、部品間のリソースの共有に用いられる。

3．C言語への変換処理

任意ビット幅の演算やリソースのアクセスなどは、C言語により作成した実行支援ライブラリの関数により行う。従って、C言語への変換では、

(1) Alhard 構文のC言語構文への展開
(2) デーモン実行文の関数としての切り出しと、必要な登録時環境の保存処置
(3) データタイプによるライブラリ関数の選択と型変換
(4) bint 型演算のための作業変数の割当
(5) シミュレーションインタフェースのための、クラス構造情報の保存

などが、主な処理内容である。また、Alhard による記述はクラス定義単位や、さらにメソッド定義単位で分離コンパイルが可能である。

4．おわりに

本稿では、言語仕様と変換処理について概要を述べた。ソフトウェア量は、Alhard-C トランスレータがC言語による記述が約 8000 行、YACC,LEX 部分が約 800 行である。また、実行支援ライブラリは、約 3000 行である。今後の課題として、並行プロセスの表現が残されている。

参考文献

1) 杉本，小島，阿部，鶴，加藤：ハードウェア動作記述言語 ALHARD(1)，情処37全大(1988).
2) 鶴，加藤，杉本，小島，阿部：ハードウェア動作記述言語 ALHARD(3)，情処37全大(1988).

表1　追加演算子

:=	代入（符号拡張）
<=	遅延転送
@	結合
:>>	算術シフト（右）
<:<	ローテート（左）
>:>	ローテート（右）
( )	ビットフィールド
[ ]	メモリアクセス