HDL設計ステップアップ講座（VHDL編）（3）

遅延のあれこれ

小林 優

回路記述やテストベンチ記述で欠かせない概念に「遅延」があります．料理でいえば調味料に相当するでしょう．脇役ではありますが，ソフトウェアにはないハードウェアならではの概念です．Verilog-HDLでの遅延表現は，＃と@だけですんでしまいそうなので，今回はVHDLを中心に「遅延」を解説します．（筆者）

1 遅延のおさらい

◆遅延表現の基本は2種類

復習をかねて遅延の記述方法を整理してみましょう．ここでは，テストベンチなどで遅延を直接記述する方法を考えます．

遅延を表現するには，**図1**に示す方法があります．wait文とafter節です．両者は文法上，異なる概念ですので，使用できる場所が次のように異なります．

◎wait文は，process文や，procedureの中
◎after節は，代入の右辺

◆wait文はバリエーションが豊富

wait文は，文法上，順次処理文（sequential statement）なので，architecture直下に書くことができません．書ける場所は，processやprocedure内にかぎられます．テストベンチのように，順次遅延を与えて入力を変化する場合に適しています．

wait文は，for，until，onなどと組み合わせて，さまざまな条件で遅延を作ることができます．たとえば，

```
wait for 100 ns;--100nsの遅延
wait until CLK'event and CLK='1';
                --クロックの立ち上がりを待つ
wait on A, B, C;--A, B, Cのどれかが変化するまで待つ
```

さらにこれらを組み合わせて，

```
wait until CLK'event and CLK='1' for 100ns;
          --クロックが立ち上がるか100ns経過するのを待つ
```

とできます．組み合わせた場合は，各条件のORになります．

余談ですが，

```
process begin
    wait until CLK'event and CLK='1';
    Q <= D;
end process;
```

と記述しても，論理合成可能なDフリップフロップ（D-FF）になります．

さらに余談ですが，この記述ってVerilog-HDLの

```
always
     @（ posedge CLK）
     <= D;
```

とまったく同じですね．

◆after節は代入文で使う

もう一つの遅延表現は，after節です（**図1(b)**）．after節は，architecture直下の代入文（同時処理代入文）と，process文内の代入文（順次処理代入文）とのどちらでも記述することができます．代入できるのは信号（signal）だけです．時間の概念を変数（variable）に持ち込むことはできません．

信号には「波形」を代入することができます．「波形」は，このafter節を用いて表現します．たとえば同時処理代入文で，

```
CLK <= '0', '1' after 25ns,
       '0' after 50ns,
       '1' after 75ns,
       '0' after 100ns;
```

基本	wait for	「時間」が経過するまで待つ
	wait until	「条件」が成立するまで待つ
	wait on	「信号」が変化するまで待つ
	waitのみ	無条件に待つ
複合	wait on 信号 for 時間	「信号」が変化するか「時間」が経過するまで待つ
	wait until 条件 for 時間	「条件」が変化するか「時間」が経過するまで待つ

(a)wait文

```
信号 < = 定数式または定数式 after 時間，… ;
```

(b)after節

〔図1〕遅延を表現する構文

〔リスト1〕相対遅延と絶対遅延

(a)相対遅延

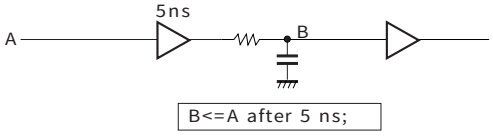
```
process begin
                    res <= '0';
   wait for 10 ns; res <= '1';
   wait for 10 ns; res <= '0';
   wait for 10 ns; res <= '1';
   ...
   wait;
end process;
```

(b)絶対遅延

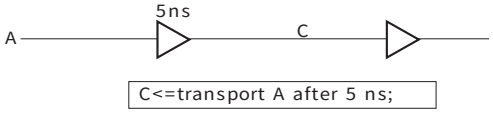
```
architecture SIM of...
...
begin
   ...
   res <= '0', '1' after 10 ns,
               '0' after 20 ns,
               '1' after 30 ns;
end SIM;
```

(c)クロック・エッジ相対遅延

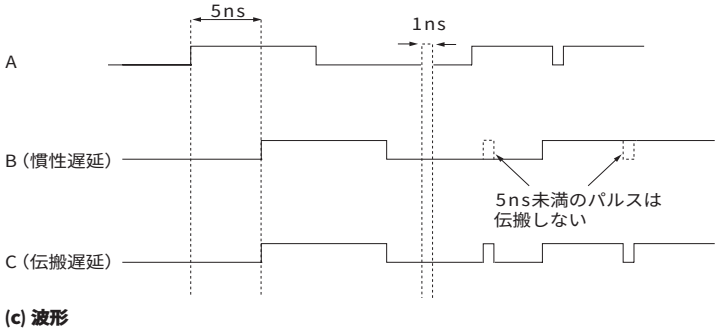
```
process begin
   ...
   wait until CLK'event and CLK='1';
        wait for 10 ns; DATA <= din;
        wait for 10 ns; WEB <= '0';
        wait for 25 ns; WEB <= '1';
   wait until CLK'event and CLK='1';
   ...
   wait;
end process;
```



(a) 慣性遅延



(b) 伝搬遅延



(c) 波形

〔図2〕慣性遅延と伝搬遅延

とすれば，1周期50nsで2周期分のCLK波形を記述したことになります．さらに同時処理代入文で，

```
   CLK <= '0', not CLK after 25ns;
   -- 周期50nsのクロック
```

とすれば，process文を使わなくても繰り返し波形のクロックを記述できます．

◆絶対遅延と相対遅延

テストベンチで検証対象への入力を作成する基本は，「遅延を与えて入力を変化する」です．**リスト1(a)**はres信号に対し1ステップずつ遅延を与えて変化させています．この場合の遅延であるwait文は，一つ前の文からの相対遅延となります．

一方，同時処理代入文中のafter節では，シミュレーション開始時刻からの絶対遅延を示します(**リスト1(b)**)．**リスト1**の**(a)**と**(b)**は同じ波形です．

システム・クロックのエッジから若干遅延して変化する信号は，**リスト1(c)**のように記述すると確実です．これはCLKのクロック・エッジから相対的に遅延して，DATAやWEB信号が変化しています．CLKの周期を変えたとしても，エッジからの遅延量を一定にすることが可能です．

◆慣性遅延と伝搬遅延

同時処理代入は，ハードウェア上での「接続」を表現したものです．したがって，代入時の遅延では，ハードウェアの世界でのできごとを忠実に表現しています．**図2(a)**の代入は，5nsの遅延を付加した何の変哲もない代入です．しかし，実際のハードウェアの世界に近い挙動をします．

図2で二つのバッファ間の遅延は5nsです．実際のバッファ間の配線には抵抗分，容量分が存在します．この5nsはこれらを考慮した遅延ですが，単純に遅れるだけではありません．**図2(c)**の波形Bのように，微小パルスは積分されて伝搬しません．正確にいえば，記述した遅延量以下の幅のパルスは，伝搬しないのです．これを**慣性遅延**と呼びます．

after節は単純に遅れるだけと思いがちですが，信号への代入は，物理的な動作も考慮しているのです(余計なお世話!?)．

単純に遅延させるには，**図2(b)**のように右辺の先頭にtransportを付加します．これを**伝搬遅延**と呼び，**図2(c)**の波形Cのように微小パルスも伝搬します．VHDLの信号代入では，慣性遅延がデフォルトになっているので要注意です．

2 デルタ遅延をきわめる

VHDLで「遅延」といえば**デルタ遅延**にふれないわけにはいきません．Verilog-HDLにはない重要な概念だからです．しかし，設計や検証の最中には，あまり気にしたことはないと思います．ここできちんと整理してみたいと思います．