Chapter 6

Wallaceツリーを用いたマルチプライヤの高速化

マルチプライヤには大きく分けて，二つの構成法があります．一つはフラッシュ・タイプといって，組み合せ回路だけで積算を実行するものです．積項を一度に発生させるので，積項の総数に比例したハードウェアが必要になります．計算速度が速いので，現在この方法が主流であり，ここでもフラッシュ・タイプについて解説します．

もう一つはシーケンシャル・タイプと言われているもので，積算をいくつかに分割して少しずつ実行します．小さなハードウェアで大きな計算ができるので，昔はよく使われました．シーケンシャル・タイプの計算方法としてはブースのアルゴリズムが有名です．

フラッシュ・タイプのマルチプライヤでは，まずすべての積項を発生させてしまいます．そのため計算の順序としては，積項を減らすためのWallaceツリーと，Wallaceツリーの出力を加算するバイナリ・アダーに分けられます．

6-1 Wallaceツリーの作りかた

Wallaceツリーはフル・アダーを使って項数を減らすプロセスです．フル・アダーを使うと，あるけたの項数を［0，3］→［1，1］というようにけた上げを行って，項数の総量を少なくしてくれます（**図6-1**）．各けたについて差分をとると，［＋1，－2］となり，あるけたの項数を2減らしてその上のけたの項数を1増やしていることがわかります．当然，項の総数は3から2に減っています．また，フル・アダーの入力の一つが定数になっている場合，論理圧縮ができて計算が速くなります．なぜなら，

入力：［a，b，1］
出力：［s1，s0］

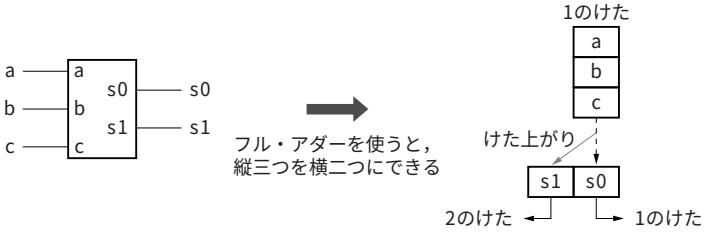


図6-1
フル・アダーによる項数の削減
フル・アダーを用いると，項数の総量を少なくしてくれる．

ここで，s0＝－(a⊕b)，s1＝a＋b
だからです．

このパターンはしばしば出てくるので，本書の回路図ではHAP1（half adder plus 1；ハーフ・アダーの出力に1を足す）という記号で表している場合があります．HAP1をVHDLのfunctionに定義しておけば，VHDLファイルの中で利用できます．VHDLの論理合成はかなりうまいのでここまで指定する必要はないことが多いのですが，論理合成の結果に納得がいかないときは試してみてください（首尾良く回路が改善されなくても，EDAツールの理解が深まる）．

各位のビット数を2以下にして高速アダーで加算

Wallaceツリーは，次のようなルールにのっとって作ります．

1）同じ位に3*n*ビット以上の変数がある場合，これを*n*個のフル・アダーの入力にする

2）フル・アダーの出力s0（和）は同じ位に，s1（キャリ）は一つ上の位になる

1）と**2）**を各位のビット数が2以下になるまで続け，その後，バイナリ・アダーで加算します．

Wallaceツリーを作る過程でリプル・キャリ（けた上げ信号）が走ることがあります．これは，各位の

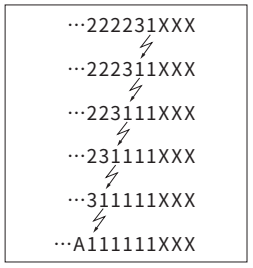


図6-2
Wallaceツリーの途中でリプル・キャリが発生

図6-3　4×4マルチプライヤ

ビット数が“…222223XXX”のパターンになったときに起こります（ただし，XXXはすべて‘0’でないとする）．ここでは，一つの位に3ビット以上の変数がある場所は1ヵ所しかないので，そこにフル・アダーを入れていくと，**図6-2**のようにリプル・キャリが走ってしまいます．この場合はハーフ・アダーを用いて，フル・アダーのキャリを待たずに上位ビットにキャリを上げてしまいます．すると，

・・・122231・・・
・・・222211・・・

というように，リプル・キャリを待たずに各位のビット数を2以下にすることができます．ただ，ハーフ・アダーを使うと次のバイナリ・アダーの入力が増えるので，高速化の必要性に応じて使い分けてください．

Wallaceツリーを使ってマルチプライヤを作る

では，上述した手順で各種マルチプライヤ（積算器）を作ってみましょう．

まずは，4ビット×4ビットのマルチプライヤですが，これは**図6-3**に示すように一義的にWallaceツリーを組むことができます．4×4マルチプライヤの各位（けた）のビット数（加算する項数）は［1，2，3，4，3，2，1］なので，最初に［…3，4，3…］のけたをそれぞれフル・アダーで加算します．フル・アダ

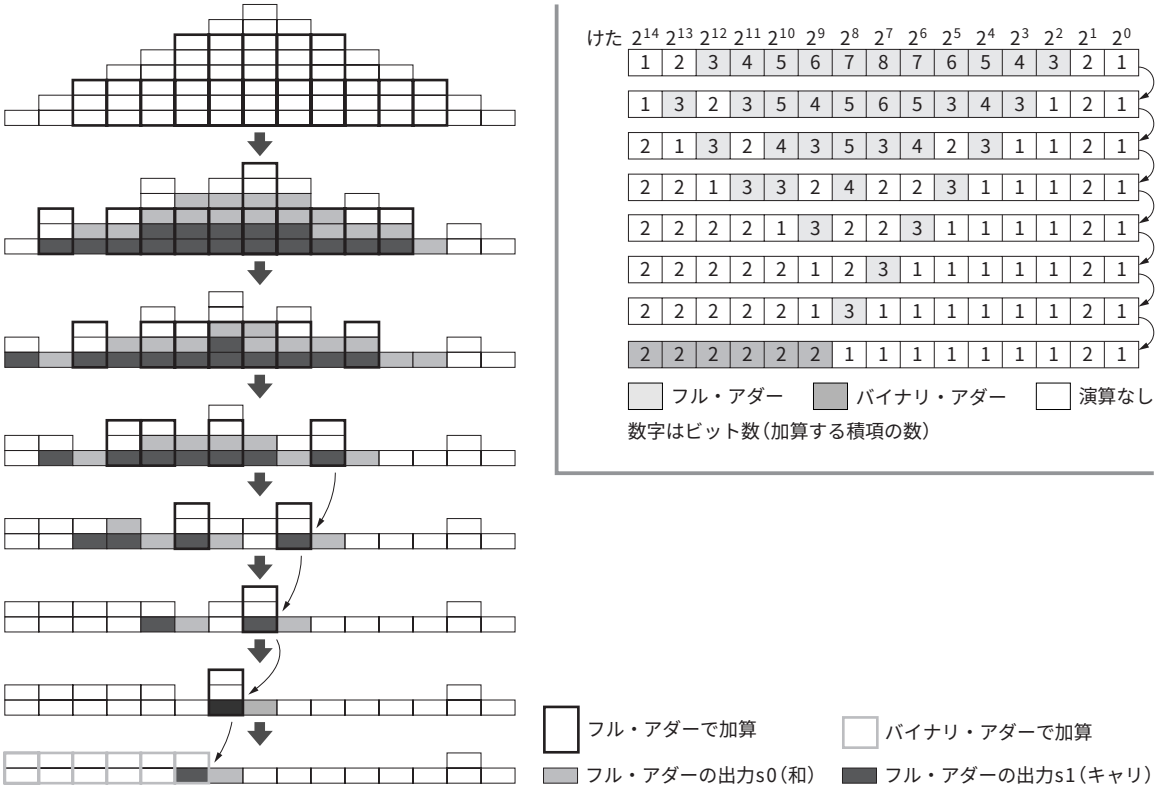
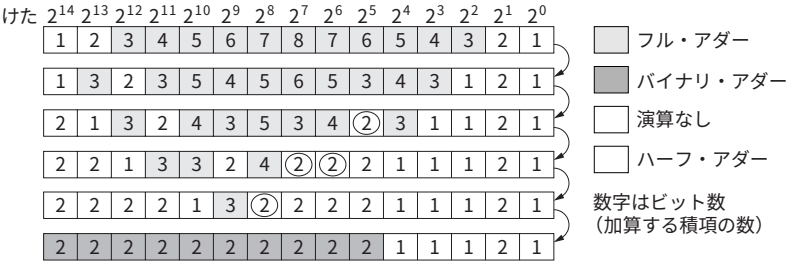


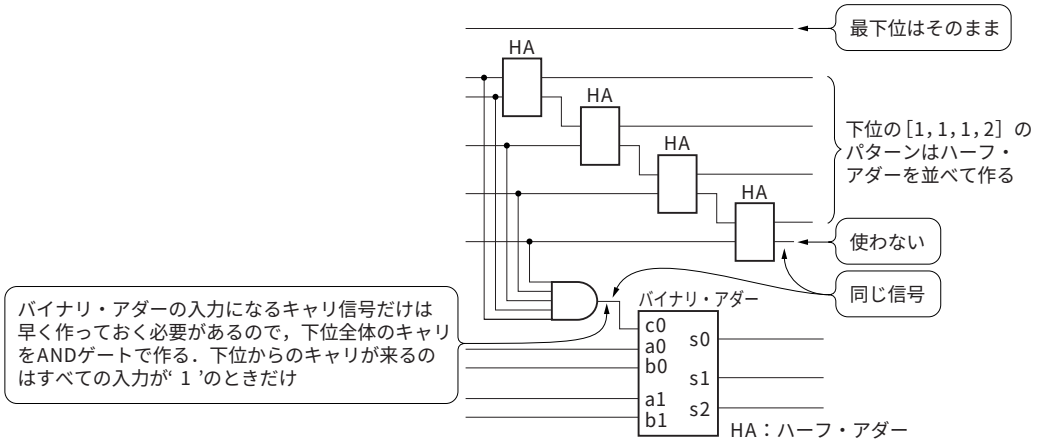
図6-4　8×8マルチプライヤ

この場合，Wallaceツリーで各位のビット数を2以下にするには7段の遅延が必要．32の位から4段にわたってリプル・キャリが走っているのが気になる．

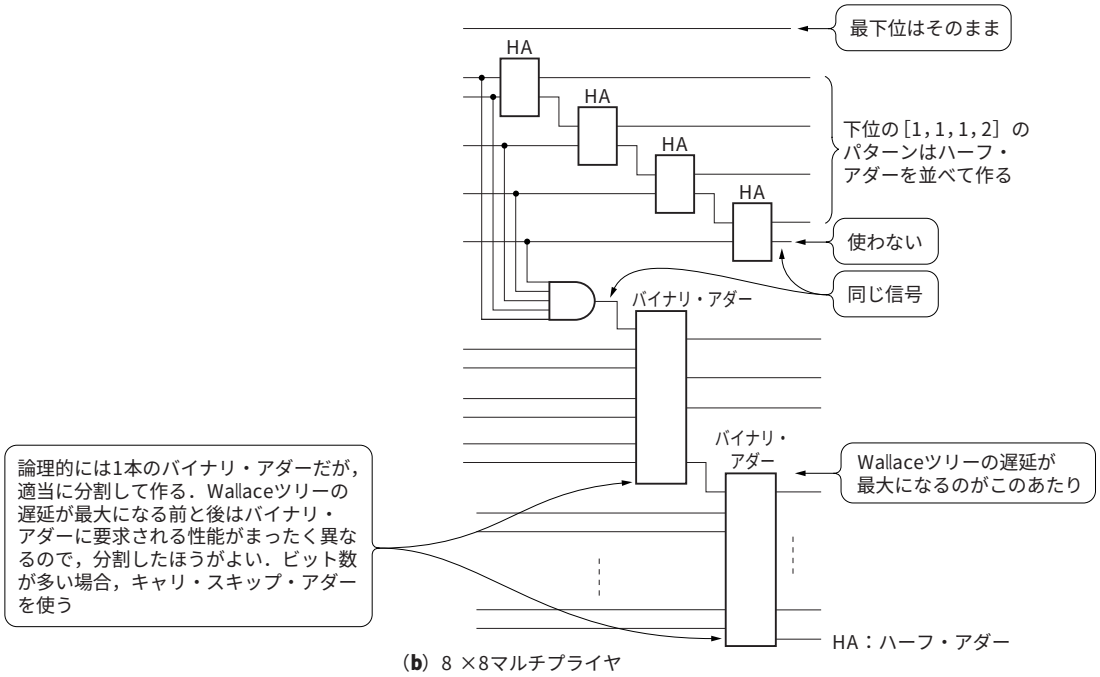
図6-5
8×8マルチプライヤ
（ハーフ・アダーを利用）

ハーフ・アダーを用いることで（つまり，下位からのキャリを処理しないことで），遅延を5段に抑えた．ハーフ・アダーを使う代償として加算するビット数が増えるが，加速の効果は大きい．





（**a**）4×4マルチプライヤ

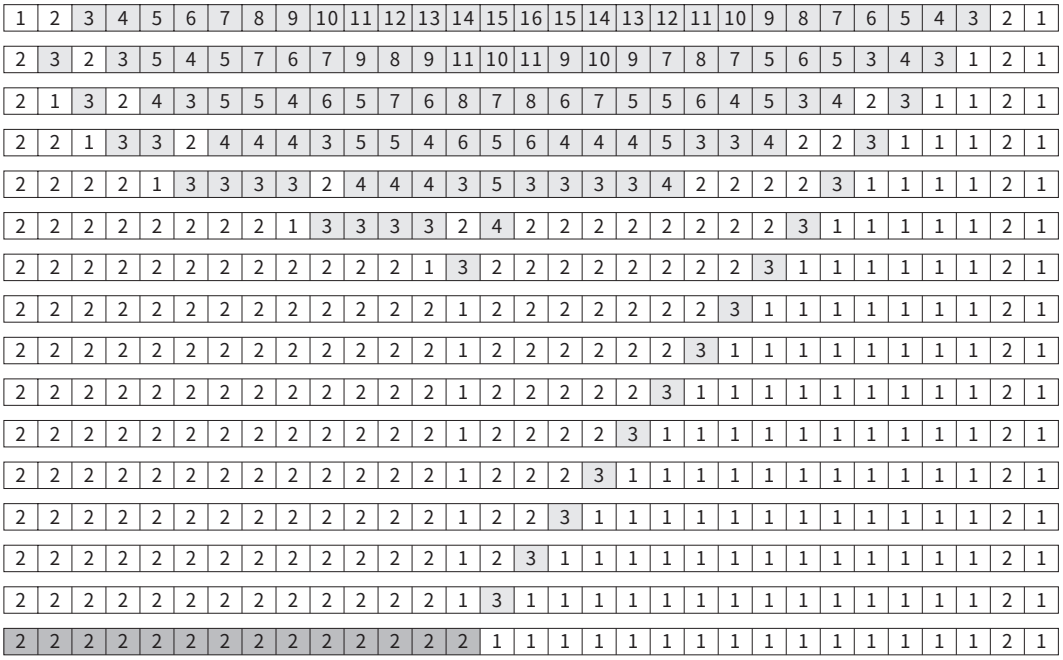


（**b**）8 ×8マルチプライヤ

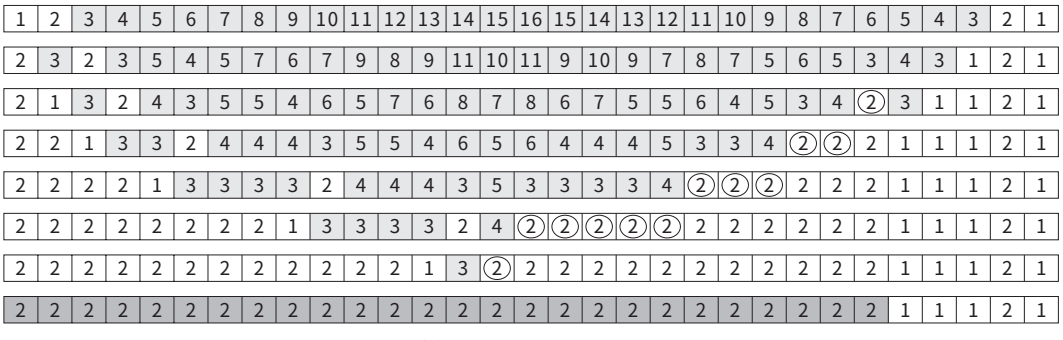
図6-6　マルチプライヤ回路例

ーの出力(和とキャリ)によって，各位の変数は[1，3，2，3，1，2，1]となります．まだ3ビット以上の位があるので，その位をフル・アダーで加算します．同じ要領でフル・アダーによる加算を行っていくと4段目で各位のビット数が[2，2，1，1，1，2，1]となるので，ここでバイナリ・アダーで加算することになります．ハーフ・アダーを用いると3段目で各位のビット数が2以下になります．

次は8ビット×8ビットのマルチプライヤですが，やりかたは同じです．この場合，Wallaceツリーで各位のビット数を2以下にするには7段の遅延が必要になります(**図6-4**)．このとき，32の位から4段にわたってリプル・キャリが走っているのが気になります．そこで，上述のハーフ・アダーによるけた



(**a**) フル・アダーとバイナリ・アダーのみ



(**b**) ハーフ・アダーを使用

フル・アダー　バイナリ・アダー　ハーフ・アダー　演算なし　数字はビット数(加算する積項の数)

図6-7　16×16マルチプライヤ

(a)では，32の位から12段にわたってリプル・キャリが走っている．(b)のようにハーフ・アダーを用いて，遅延を7段に抑える．