2005年ハードウェア構成法中間試験の解説

石井 康雄*

平成 17 年 12 月 6 日

間違いに気づいた人は**こっそり**教えてください。お願いします。

1　ビットスライスアダー

問題 1

押しボタンが 15 個あって、そのうちの過半数が押されたことを判定する回路を示せ。

1.1　入出力ビット数

まず、入出力ビットに関して検証します。入力に関しては $a_0, a_1, \cdots, a_{14}$ の 15 本をとることします。出力については s の 1 本をとるとします。従って、入出力のビット数に関しては以下のようになります。

入力 $a_0, a_1, \cdots, a_{14}$

出力 s

従って、この 15 本の入力ビットを適当に加工して 1 ビットの結果をもとめればよいということが分かるでしょう。

1.2　解法

まず、ビットスライスアダーを用いて入力中の 1 の本数を判定し、その本数が 8 以上なら $s = 1$ そうでないなら $s = 0$ とする回路を構成します。

1.2.1　ビットスライスアダー

まずは、ビットスライスアダーで 15 本の入力のうちの何本で 1 がたっているかを調べましょう。ビットスライスアダーはフルアダーを多数組み合わせて作ることができます。今回のビットスライスアダーは出力は 0 から 15 ですから、4 ビットの符号なし整数であらすことができます。従って、$15 - 4 = 11$ 個程度のフルアダーを利用すれば十分です。

あとは、体力勝負です。桁をそろえることに注意をして、ビットスライスアダーを構成しましょう。

実は授業中に一度 15 ビットのビットスライスアダーは書いた気がするので、ノートを読み返してそれをそのまま書いても OK でしたね。

*yishii@is.s.u-tokyo.ac.jp

1.2.2 出力を求める

ビットスライスアダーを構成できたら、それが 8 以上あるかそうでないかを判定します。このビットスライスアダーの出力は符号なし整数ですから、その値を $S = (s_3, s_2, s_1, s_0)$ とすると $S = \sum_{i=0}^{3} 2^i s_i$ です。従って、$S \geq 8 \Leftrightarrow s_3 = 1$ なので、出力は s_3 をそのまま出力すれば十分です。

1.2.3 最適化

ここから先は最適化の話です。解答には書いていても書いていなくても OK です。

前節までの議論の結果、ビットスライスアダーの出力のうち、答えを求めることに利用したものは s_3 の 1 本だけでした。そこで、s_3 の出力に関係のない回路を取り除きます。

ビットスライスアダー中のフルアダーのうち s_2, s_1, s_0 を出力するのは 3 つです。この回路の桁上がり部分は s_3 の計算に必要ですが、同じ桁の出力をする XOR 回路部分は不要ですので消去します。要するにキャリーの生成部分だけの回路にします。

2 積和演算

問題 2

4 bit の整数 $A(a_3, a_2, a_1, a_0)$、4 bit の整数 $B(b_3, b_2, b_1, b_0)$、がある
$(A + 1) \times (B + 1) - 1$ を出力する回路を求めよ

2.1 入出力ビット数

1 章と同じく入出力のビット数に関して検証します $-8 \leq A \leq 7, -8 \leq B \leq 7$ ですので $-7 \leq A + 1 \leq 8, -7 \leq B + 1 \leq 8$ です。よって、$-57 \leq (A + 1) \times (B + 1) - 1 \leq 63$ が分かります。従って、この計算の出力は 7 ビットの符号付整数値を用いれば十分に表せるということが分かります。

以上より、入出力のビットをあらわすと以下のようになります。(このときにワラスツリーの出力を計算できます、詳しくは後述)

入力 $a_3, a_2, a_1, a_0, b_3, b_2, b_1, b_0$

出力 $s_6, s_5, s_4, s_3, s_2, s_1, s_0$

以降ではどのようにこの 7 ビットの出力を得ればよいのかを考えます。

2.2 解法

まず整数乗算に関する一般的なアルゴリズムを述べておきます。その後に今回の問題である積和演算の解法に関して考えます。

2.2.1 整数乗算の一般的なアルゴリズム

2004 年度の解説の使い回しです。あしからず。

n bit の整数 A と m bit の整数 B の乗算 AB を求めることを考えます。
A と B のビット列の表す整数値は以下のようになります。

$$A = -2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

$$B = -2^{m-1}b_{m-1} + \sum_{j=0}^{m-2} 2^j b_j$$

従って、その積 AB は以下のように表せます。

$$\begin{aligned} AB &= 2^{n+m-2}a_{n-1}b_{m-1} + \sum_{i=0}^{n-2}\sum_{j=0}^{m-2} 2^{i+j}a_i b_j \\ &- \sum_{j=0}^{m-2} 2^{n+j-1}a_{n-1}b_j - \sum_{i=0}^{n-2} 2^{n+i-1}a_i b_{m-2} \end{aligned} \tag{1}$$

ただし、この式 (1) では負の整数が多数含まれていて厄介ですのでこれを消去することを考えます。$-a_{n-1}b_j$ などを $(1 - a_{n-1}b_j) - 1$ と置換してみます。これを (1) に代入して整理してみると以下のような式になります。

$$\begin{aligned} AB &= 2^{n+m-2}a_{n-1}b_{m-1} + \sum_{i=0}^{n-2}\sum_{j=0}^{m-2} 2^{i+j}a_i b_j \\ &+ \sum_{j=0}^{m-2} 2^{n+j-1}(1 - a_{n-1}b_j) + \sum_{i=0}^{n-2} 2^{n+i-1}(1 - a_i b_{m-2}) \\ &- \sum_{j=0}^{m-2} 2^{n+j-1} - \sum_{i=0}^{n-2} 2^{n+i-1} \end{aligned} \tag{2}$$

となります。$1 - a_{n-1}b_j = \overline{a_{n-1}b_j}$ と置くことができますのでさらにこの式は以下のように整理できます。

$$\begin{aligned} AB &= 2^{n+m-2}a_{n-1}b_{m-1} + \sum_{i=0}^{n-2}\sum_{j=0}^{m-2} 2^{i+j}a_i b_j \\ &+ \sum_{j=0}^{m-2} 2^{n+j-1}\overline{a_{n-1}b_j} + \sum_{i=0}^{n-2} 2^{n+i-1}\overline{a_i b_{m-2}} \\ &- \sum_{j=0}^{m-2} 2^{n+j-1} - \sum_{i=0}^{n-2} 2^{n+i-1} \end{aligned} \tag{3}$$

ついでに $-\sum_{j=0}^{m-2} 2^{n+j-1} - \sum_{i=0}^{n-2} 2^{n+i-1}$ はただの定数ですから K とでもおいてしまいましょう。最終的に AB の値は以下のように表せます。

$$\begin{aligned} AB &= 2^{n+m-2}a_{n-1}b_{m-1} + \sum_{i=0}^{n-2}\sum_{j=0}^{m-2} 2^{i+j}a_i b_j \\ &+ \sum_{j=0}^{m-2} 2^{n+j-1}\overline{a_{n-1}b_j} + \sum_{i=0}^{n-2} 2^{n+i-1}\overline{a_i b_{m-2}} + K \end{aligned} \tag{4}$$

この式を筆算形式で書くことを考えて見ましょう ($n \geq m$ を仮定します)。

$$
\begin{array}{cccccccccccc}
 & & & & & \overline{a_{n-1}b_0} & a_{n-2}b_0 & & \cdots & & a_0b_0 \\
 & & & \cdots & & \cdots & & & & \cdots & \\
 & & \overline{a_{n-1}b_{m-2}} & a_{n-2}b_{m-2} & & & \cdots & & a_0b_{m-2} & & \\
 & a_{n-1}b_{m-1} & \overline{a_{n-2}b_{m-1}} & & & \cdots & & \overline{a_0b_{m-1}} & & & \\
k_{n+m-1} & k_{n+m-2} & & & & \cdots & \cdots & \cdots & & & k_0 \\
\hline
s_{n+m-1} & s_{n+m-2} & & & & \cdots & \cdots & \cdots & & & s_0
\end{array}
$$

こんな具合になります。そして、通常はワラスツリーを用いて項数を減らしてから加算をします。

2.2.2 今回の課題の場合

乗算をしてから減算を行ってもいいのですがせっかくハードウェアでやるのですすべての結果を畳み込みましょう。すると、$(A+1)\times(B+1)-1 = A\times B + A + B$ は筆算形式では以下のようになります。A, B に関しては $A+8, B+8$ として (つまり最上位 bit を反転させて) 最終的に定数の畳み込みで解決しています。

$$
\begin{array}{ccccccc}
 & & & \overline{a_3} & a_2 & a_1 & a_0 \\
 & & & \overline{b_3} & b_2 & b_1 & b_0 \\
 & & & \overline{a_3b_0} & a_2b_0 & a_1b_0 & a_0b_0 \\
 & & \overline{a_3b_1} & a_2b_1 & a_1b_1 & a_0b_1 & \\
 & \overline{a_3b_2} & a_2b_2 & a_1b_2 & a_0b_2 & & \\
a_3b_3 & \overline{a_2b_3} & \overline{a_1b_3} & \overline{a_0b_3} & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

最後の定数に関しては $A = B = 0$ のときに 0 を返すように設定します。…今回の課題ではその値がゼロですね。ラッキーでした。あとは、これを計算する回路をがんばって書くのみです。あとは体力勝負 !! がんばりましょう。

2.2.3 別解

上記の解放は最も一般的な解法です。多少、式の変更があってもそのまま使えます。しかし、今回の課題ではもっと最適な解法があります。

$-A = \overline{A} + 1 \Leftrightarrow \overline{A} = -A - 1$ ですから、

$$
\begin{aligned}
(A+1)\times(B+1)-1 &= (-A-1)\times(-B-1)-1 \\
&= \overline{A}\times\overline{B}-1
\end{aligned}
$$

と変形できます。

$$
\begin{array}{ccccccc}
 & & & \overline{\overline{a_3}\,\overline{b_0}} & \overline{a_2}\,\overline{b_0} & \overline{a_1}\,\overline{b_0} & \overline{a_0}\,\overline{b_0} \\
 & & \overline{\overline{a_3}\,\overline{b_1}} & \overline{a_2}\,\overline{b_1} & \overline{a_1}\,\overline{b_1} & \overline{a_0}\,\overline{b_1} & \\
 & \overline{\overline{a_3}\,\overline{b_2}} & \overline{a_2}\,\overline{b_2} & \overline{a_1}\,\overline{b_2} & \overline{a_0}\,\overline{b_2} & & \\
\overline{a_3}\,\overline{b_3} & \overline{\overline{a_2}\,\overline{b_3}} & \overline{\overline{a_1}\,\overline{b_3}} & \overline{\overline{a_0}\,\overline{b_3}} & & & \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
s_6 & s_5 & s_4 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

今回の場合は最後の定数値はゼロになりませんでしたのでちゃんと足します。

この場合には積項の数が最小化されるので最も良い解法であるといえます。何人かの学生はこの解法で解いてくれました。

2.2.4 補足:ワラスツリーに関して

乗算ではワラスツリーを用いて回路を最適化します。このときには最終型を想像しながら行いましょう。これは非常に簡単なアルゴリズムで考えられます。たとえば今回のケースでは (ありえない話ですが) 上の全ての変数項が 1 であったと仮定します。すると上の筆算形式の演算は以下のように計算できます。

$$
\begin{array}{cccccccc}
 & & & & 1 & 1 & 1 & 1 \\
 & & & 1 & 1 & 1 & 1 & \\
 & & 1 & 1 & 1 & 1 & & \\
 & 1 & 1 & 1 & 1 & & & \\
+) & & & & 1 & 1 & 1 & 1 \\
\hline
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
+) & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
\hline
 & FA & FA & FA & HA & HA & HA & HA
\end{array}
$$

このときにはワラスツリーを用いて 7 本の出力と 4 本の出力を得ることができます。各ビットに関して 1 が 2 つたっているところは最終的にフルアダーを用いて、1 が 1 つしかたっていないところは最終的にハーフアダーで最終結果を計算することになりますですから、上の筆算式での FA はフルアダー、HA はハーフアダーを示しています。一番下の桁は下からのキャリーがないので HA にします。

また、この式の結果はこのように馬鹿正直に筆算をする必要はなく、概念さえ分かっていれば変域の計算の際に求めることができます。今回の答えが高々7 ビットであることに注意をして、$\overline{A}$、$\overline{B}$ が全て正数であったと仮定をしましょう。また、畳み込む定数も正数化します (0001111 = 15 ですね)。また、7 ビットの正数の最大値は 127(= 1111111) であることに注意すると。

$$(A+1)\times(B+1)-1 = 15\times 15+15 = 240 = 127+113 = 1111111+1110001$$

となることが分かります。この計算で上の筆算での最終結果の 2 つの数字と同等の結果が得られます。これによりワラスツリーの出力結果を比較的簡単に得ることができます。ワラスツリーは結構複雑になりがちですからこのようにして最終の入力ビットの本数をあらかじめ計算しておくとケアレスミスを避けることができます。

ついでにハーフアダー部分のキャリーは AND ゲートを用いて一度に判定することができますので、それを行った回路が今回の最適回路といえそうです。

3　アンケート結果

今回の試験のアンケート結果で多かった疑問の回答を掲載します。

- D-FF(D ラッチ) の役割
- FPGA と ASIC の違い
- 浮動小数点数に関して

3.1　D ラッチの役割

アンケートや授業の質問などで最も多かったものです。

まず、D ラッチは 2 種類存在しています。1 つはレベルセンシティブラッチで、もう一つがエッジセンシティブラッチです。現実の同期式回路ではエッジセンシティブラッチを使います。D ラッチはさまざまな場所で使われています。例えば、SRAM(CPU のキャッシュメモリなどに利用されます) は D ラッチのアレイです。

このラッチの使用目的と効果ですが、下のようなものがあります。

データの保存 そのままです。SRAM などですね。あるいはステートマシンの「状態の保持」も、この利用目的です。

回路規模の節約 例えば、回路面積の都合上、今回の課題のような乗算器が実装できなかったとしましょう。仕方がないので繰り返し演算により実現することにします。回路規模を最小化するなら 1 つの加算器を使いまわすことになるでしょう (興味があれば、どんな回路になるか書いてみてください)。

この例では、従来の組み合わせ回路では実現できなかった機能を実現できることになります。

動作周波数の向上 (パイプライン化) ある回路で、実装面積などの都合上、1 つの乗算器が 1 秒間に 1 億回の乗算をする必要があったとします。そして、組み合わせ回路 (今回の課題のような回路) で実装した場合には 1 秒間に 7000 万回が限度だったとします。

このときに演算をいくつかのステップに分けて演算することをパイプライン化といいます。各ステップの出力は次のステップの出力に必要な入力を全て生成します。上手く、構成すれば目標のスループットを実現できるでしょう。

ただし、このパイプライン化という処理にはラッチにデータをセットするための処理が余分にかかるため全体のレイテンシは遅くなります。例えば、図 1 の回路は動作周波数は 100MHz から 166MHz に上がっていますが、全体のレイテンシは 10ns から 12ns に下がっています。

3.2　FPGA と ASIC の違い

FPGA は回路を再構成できるように設計された回路です。その内容は D-FF といくつかの組み合わせ回路から構成されます。

図 2 に AND ゲートの FPGA 上の実装例を書きました。図は 2 入力 1 出力ですが、現実の FPGA では主に 4 入力 1 出力の回路を採用しています。これをルックアップテーブル (LUT) と呼びます。LUT

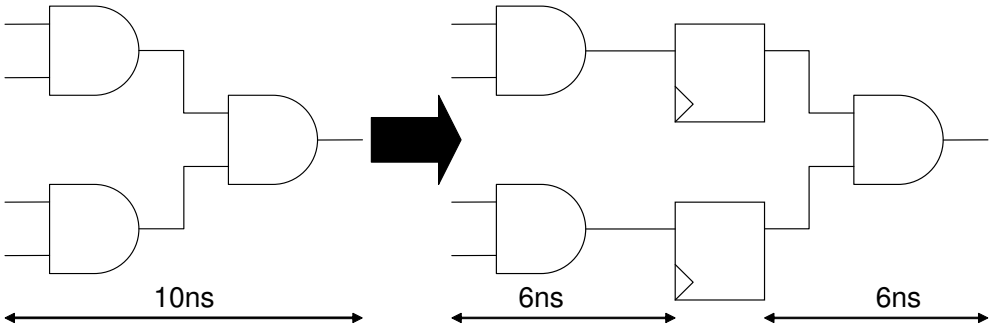

図 1: パイプライン化の例

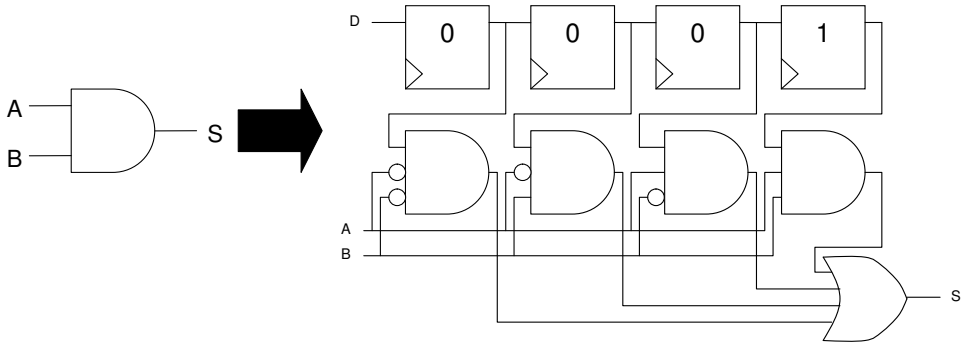

図 2: AND ゲートの FPGA 上の実装

は D の配線からデータを流し込み、そのデータを保持しておくことで任意の回路を構成することができることがわかると思います。例えば 0001 を 0111 に変更すれば OR と等価な回路が構成できます。図から一目瞭然ですが FPGA は ASIC と比較して動作速度は遅くなります。図 2 とほぼ同様の手法で配線や D ラッチも再構成できるようになっています。

なお、最近の FPGA だと、もっと多様な機能が付いています。詳しく知りたい人がいれば、Xilinx 社か Altera 社のホームページからマニュアルを入手して読んでみると良いでしょう。

3.3 浮動小数点数に関して

丸め時に桁上がりが発生してで指数部に影響が及んだ場合に、もう一度丸め操作をする必要があるんじゃないのか? という質問がいくつかありました。これは結論から言うと**必要ありません**。

その理由は丸め時に桁上がりが発生するのは、仮数部が $1.11\cdots1 = 1 - ulp$ の時だけだからです。この場合には**先に ulp を左に 1 つずれた点を見て丸めを行なっても、ulp を足した後に左にずれた値を見ても**結果は 2 になります。

よく考えれば当たり前で、$1.11\cdots1$ は **2 よりも小さい値の中で最も大きい値**で、それに誤差の補正をかけるので 2 にはなってもそれ以上の値には絶対にならないわけです。