**Example 1**

If there is a sequence of 0's in the multiplier, the multiplication is easy as all 0's can be skipped.

\begin{displaymath}
\begin{tabular}{cccc\vert\vert cccccccccccc}
& & 2 & 2 & & ...
...4 & & & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
\end{tabular}\end{displaymath}

**Example 2**

However, it does not help if there is a sequence of 1's in the multiplier. We have to go through each one of them:

\begin{displaymath}
\begin{tabular}{cccc\vert\vert cccccccccccc}
& & 2 & 2 & & ...
... & 8 & & & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0
\end{tabular}\end{displaymath}

How can we enjoy a sequence of 1's as well as a sequence of 0's? We first Realize that $001110=010000-000010$,

or in general a string of 1's in the multiplier A can be written as: 

\begin{displaymath}
\begin{tabular}{ccccccccccccc}
& & d & d & \underline{0} & ...
...ts$ & 0 & \underline{1} & \underline{0} & 0 & 0
\end{tabular}\end{displaymath}

where d is ``don't care'' (either 0 or 1). If we define the first part of the above as $A_{leftend}=dd10\cdots 00dd$ and the second part as $A_{rigtend}=0000\cdots
1000$,

then the multiplication becomes: 

\begin{displaymath}
B\times A=B\times (A_{leftend}-A_{rightend})=B\times A_{leftend}-B\times A_{rightend}
\end{displaymath}

In other words, only the two ends of a string of 1's in the multiplier need to be taken care of. At the left end the multiplicand is added to the partial product, while at the right end the multiplicand is subtracted from the partial product. The above multiplication can therefore be written as: 

\begin{displaymath}
\begin{tabular}{ccccccccccc\vert\vert ccccccccccccc}
& & & ...
...0 &&& & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
\end{tabular}\end{displaymath}

On the right side above the subtraction is carried out by adding 2's complement. We observe that there is a sequence of 1's in the multiplier, only the two ends need to be taken care of, while all 1's in between do not require any operation. The Booth's algorithm for multiplication is based on this observation.

To do a multiplication $B\times A$, where

* $B=b_{n-1} b_{n-2}\cdots b_1 b_0$ is the multiplicand
* $A=a_{n-1} a_{n-2}\cdots a_1 a_0$ is the multiplier

we check every two consecutive bits in $A$ at a time: 

\begin{displaymath}
\begin{tabular}{c\vert c\vert c\vert\vert l} \hline
$a_i$ &...
... 1. \\
& & & Add B to partial product  \hline
\end{tabular}\end{displaymath}

where $i=0, 1, \cdots, n-1$, and when $i=0$, $a_{i-1}=a_{-1} \equiv 0$.

**Why does it work?** What we did can be summarized as the following

|  |  |  |  |
| --- | --- | --- | --- |
| $\displaystyle Product$ | $\textstyle =$ | $\displaystyle (a_{-1}-a_0)\times B\times 2^0+(a_0-a_1)\times B\times 2^1+ (a_1-a_2) \times B\times 2^2+$ |  |
|  |  | $\displaystyle \cdots +(a_{n-2}-a_{n-1})\times B\times 2^{n-1}$ |  |
|  | $\textstyle =$ | $\displaystyle B \times [\;-a_{n-1}\times 2^{n-1}+ \sum_{i=0}^{n-2} a_i\times 2^i\;]$ |  |
|  | $\textstyle \stackrel{*}{=}$ | $\displaystyle B \times Val(A)$ |  |

\* Recall that the value of a signed-2's complement number (either positive or negative) can be found by:

\begin{displaymath}Val(A=a_{n-1} \cdots a_{0})
=-a_{n-1}\times 2^{n-1}+\sum_{i=0}^{n-2} a_i\times 2^i \end{displaymath}

**Another Example:**

Assume $n=7$ bits available. Multiply $B=22=(0010110)_2$ by $A=-34=-(0100010)_2$. First represent both operands and their negation in signed 2's complement:

\begin{displaymath}
\begin{tabular}{ccccc}
22: & 0010110, & -22: & 1101010 \\
34: & 0100010, & -34: & 1011110
\end{tabular}\end{displaymath}

Then carry out the multiplication in the hardware: 

\begin{displaymath}
\begin{tabular}{c\vert c\vert\vert ccccc} \hline
& & [M] & ...
...right shift & & 1111010 & & 0010100 & 1  \hline
\end{tabular}\end{displaymath}

The upper half of the final result $1111010 \;\; 0010100$ is in register [A] while the lower half is in register [Q]. The product is given in signed 2's complement and its actual value is negative of the 2's complement:

\begin{displaymath}B\times A=-\overline{11110100010100}=-00001011101100=-748_{10} \end{displaymath}

Also note that:

* As the operands are in signed 2's complement form, the *arithmetic shift* is used for the right shifts above, i.e., the MSB bit (sign bit) is always repeated while all other bits are shifted to the right. This guarantees the proper sign extension for both positive and negative values represented in signed 2's complement.
* When the multiplicand is negative represented by signed 2's complement, it needs to be complemented again for subtraction (when the LSB of multiplier is 1 and the extra bit is 0, i.e., the beginning of a string of 1's).