Let us first consider the general task of multiplication. Two binary numbers can be multiplied using the same method as we use for decimal numbers. In the following, we will simply assume that both numbers are \( n \)-bit unsigned binary numbers. Figure 1 shows how multiplication is performed manually, using 4-bit numbers. Each multiplier bit is examined from right to left. If a bit is equal to 1, an appropriately shifted version of the multiplicand is added to form a \emph{partial product}. If the multiplier bit is equal to 0, then nothing is added. The sum of all shifted versions of the multiplicand is the desired product. Note the product occupies 8 bits. (In general, multiplying two \( n \)-bit unsigned numbers results an \( 2n \)-bit unsigned number).

\[
\begin{array}{c}
\text{Multiplicand A (14)} & 1 \ 1 \ 1 \ 0 \\
\text{Multiplier B (11)} & \times 1 \ 0 \ 1 \\
\text{Partial product 0} & 1 \ 1 \ 1 \ 0 \\
\text{Partial product 1} & 1 \ 0 \ 1 \ 0 \ 1 \\
\text{Partial product 2} & 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \\
\text{Product P (154)} & 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \\
\end{array}
\]

Figure 1: Multiplication by hand.

The same scheme can be used to design a multiplier circuit. Let us stay with 4-bit numbers to keep the discussion simple. Let the multiplicand, multiplier, and product be denoted as \( A = a_3a_2a_1a_0 \), \( B = b_3b_2b_1b_0 \), and \( P = p_7p_6p_5p_4p_3p_2p_1p_0 \), respectively.

**A Sequential Addition Approach**

One simple way of implementing the multiplication scheme is to use a sequential approach, where an 8-bit adder is used to compute partial products. As a first step, the bit \( b_0 \) is examined. If \( b_0 = 1 \), then \( A \) is added to the initial partial product, which is initialized to 0. If \( b_0 = 0 \), then 0 is added to the partial product. Next \( b_1 \) is examined. If \( b_1 = 1 \), then the value \( 2 \times A \) is added to the partial product. The value \( 2 \times A \) is created simply by shifting \( A \) one bit position to the left. Similarly, \( 4 \times A \) is added to the partial product if \( b_2 = 1 \), and \( 8 \times A \) is added if \( b_3 = 1 \).

**Array Multiplier for Unsigned Numbers**

The sequential approach leads to a relatively slow circuit, primarily because a single 8-bit adder is used to

\[
\begin{array}{c}
\text{Multiplicand A (14)} & 1 \ 1 \ 1 \ 0 \\
\text{Multiplier B (11)} & \times 1 \ 0 \ 1 \\
\text{Partial product 0} & 1 \ 1 \ 1 \ 0 \\
\text{Partial product 1} & 1 \ 0 \ 1 \ 0 \ 1 \\
\text{Partial product 2} & 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \\
\text{Product P (154)} & 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \\
\end{array}
\]

Figure 2: Multiplication for implementation in hardware.
perform all additions needed to generate the partial products and the final product. A much faster circuit can be obtained if multiple adders are used to compute the partial products.

Figure 2 indicates how multiplication may be performed by using multiple adders. In each step a 4-bit adder is used to compute the new partial product. Note that as the computation progresses, the least significant bits are not affected by subsequent additions; hence they can be passed directly to the final product, as indicated by the arrows. Of course, these bits are a part of the partial products as well.

A faster multiplier circuit can be designed using an array structure that is similar to the organization in Figure 2. Consider a $4 \times 4$ example, where the multiplicand and multiplier are $A = a_3a_2a_1a_0$ and $B = b_3b_2b_1b_0$, respectively. The partial product 0, $PP_0 = pp03pp02pp01pp00$, can be generated using the AND of $b_0$ with each bit of $A$. Thus

$$PP_0 = a_3b_0 \ a_2b_0 \ a_1b_0 \ a_0b_0$$

Partial product 1, $PP_1$, is generated using the AND of $b_1$ with $A$ and adding it to $PP_0$ as follows

$$PP0 : \begin{array}{cccccc}
0 & pp03 & pp02 & pp01 & pp00 \\
+ & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & 0
\end{array}$$

$$PP1 : \begin{array}{cccccc}
pp14 & pp13 & pp12 & pp11 & pp10
\end{array}$$

Similarly, partial product 2, $PP_2$, is generated using the AND of $b_2$ with $A$ and adding to $PP_1$, and so on.

A circuit that implements the preceding operations is arranged in an array, as shown in Figure 3. There are two types of blocks in the array. Figure 4 shows the details of the blocks in the top row, and Figure 5 shows the block used in the second and third rows. Observe that the shifted versions of the multiplicand are provided by routing the $a_k$ signals diagonally from one block to another. The full-adder included in each block implements a ripple-carry adder to generate each partial product.

Figure 3: The structure of a $4 \times 4$ multiplier circuit.
Figure 4: A block in the top row in Figure 3.

Figure 5: A block in the bottom two rows in Figure 3.