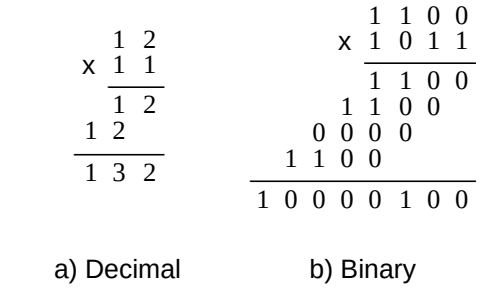
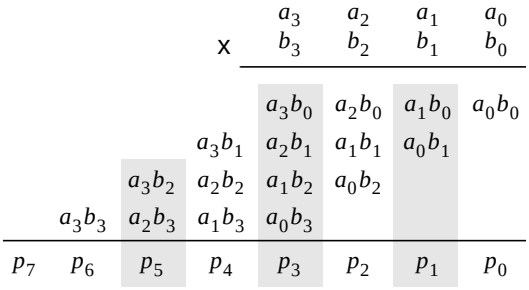
Part V

Figure 5a gives an example of the traditional paper-and-pencil multiplication $P = A \times B$, where $A = 12$ and $B = 11$. We need to add two summands that are shifted versions of A to form the product $P = 132$. Part b of the figure shows the same example using four-bit binary numbers. Since each digit in B is either 1 or 0, the summands are either shifted versions of A or 0000. Figure 5c shows how each summand can be formed by using the Boolean AND operation of A with the appropriate bit in B.



a) Decimal b) Binary



c) Implementation

Figure 5. Multiplication of binary numbers.

A four-bit circuit that implements $P = A \times B$ is illustrated in Figure 6. Because of its regular structure, this type of multiplier circuit is usually called an *array multiplier*. The shaded areas in the circuit correspond to the shaded columns in Figure 5c. In each row of the multiplier AND gates are used to produce the summands, and full adder modules are used to generate the required sums.

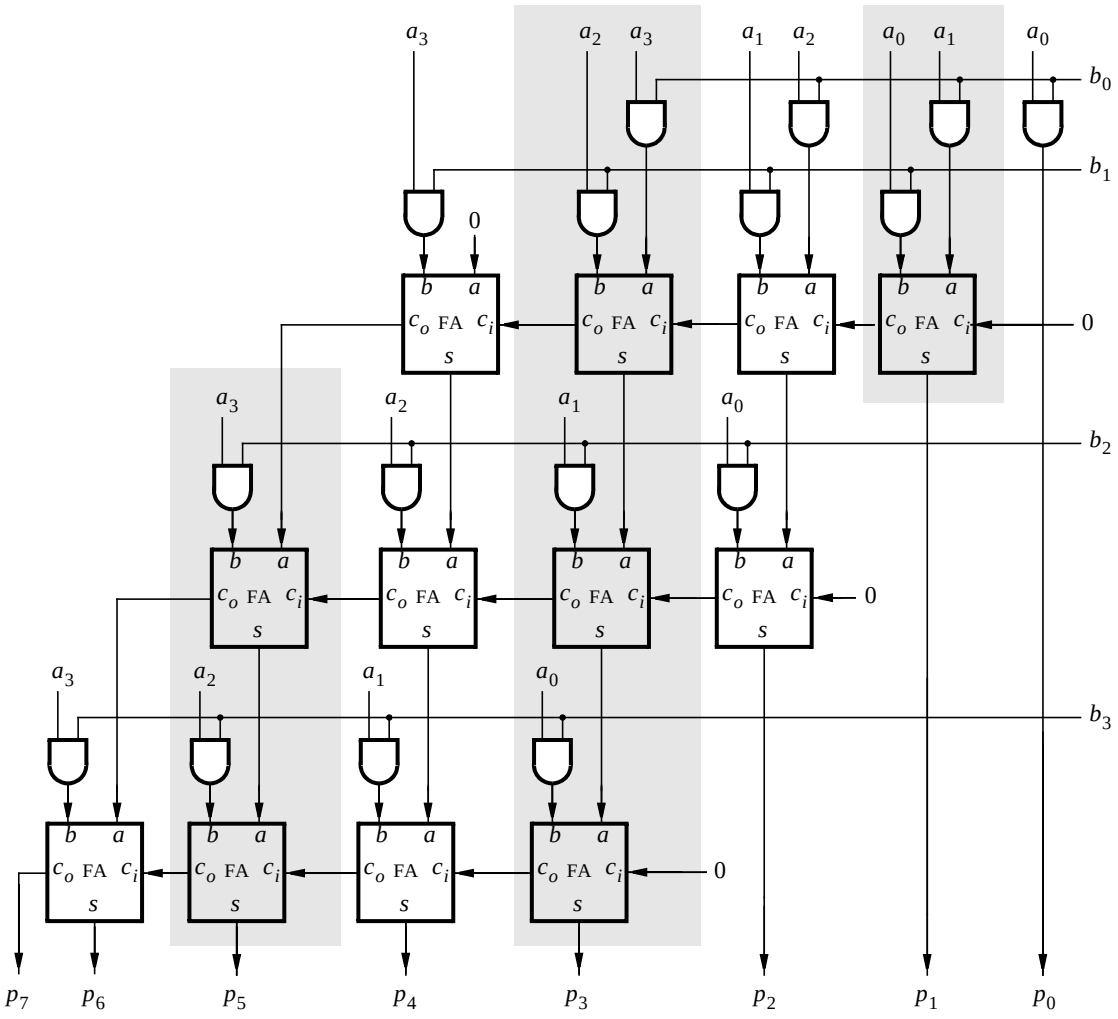


Figure 6. An array multiplier circuit.

Use the following steps to implement the array multiplier circuit:

1. Create a new Quartus II project which will be used to implement the desired circuit on the Altera DE2 board.
2. Generate the required VHDL file, include it in your project, and compile the circuit.
3. Use functional simulation to verify that your code is correct.
4. Augment your design to use switches SW_{11-8} to represent the number A and switches SW_{3-0} to represent B. The hexadecimal values of A and B are to be displayed on the 7-segment displays *HEX6* and *HEX4*, respectively. The result $P = A \times B$ is to be displayed on *HEX1* and *HEX0*.
5. Assign the pins on the FPGA to connect to the switches and 7-segment displays, as indicated in the User Manual for the DE2 board.
6. Recompile the circuit and download it into the FPGA chip.
7. Test the functionality of your design by toggling the switches and observing the 7-segment displays.

Part VI

Extend your multiplier to multiply 8-bit numbers and produce a 16-bit product. Use switches SW_{15-8} to represent the number A and switches SW_{7-0} to represent B. The hexadecimal values of A and B are to be displayed on the 7-segment displays *HEX7−6* and *HEX5−4*, respectively. The result $P = A \times B$ is to be displayed on *HEX3−0*. Add registers to your circuit to store the values of A, B, and the product P, using a similar structure as shown for the registered adder in Figure 1.

After successfully compiling and testing your multiplier circuit, examine the results produced by the Quartus II Timing Analyzer to determine the fmax of your circuit. What is the longest path in terms of delay between registers?

Part VII

Change your VHDL code to implement the 8 x 8 multiplier by using the *lpm_mult* module from the library of parameterized modules in the Quartus II system. Complete the design steps above. Compare the results in terms of the number of logic elements (LEs) needed and the circuit fmax.

Part VIII

It many applications of digital circuits it is useful to be able to perform some number of multiplications and then produce a summation of the results. For this part of the exercise you are to design a circuit that performs the calculation

$$S = (A \times B) + (C \times D)$$

The inputs A, B, C, and D are eight-bit unsigned numbers, and S provides a 16-bit result. Your circuit should also provide a carry-out signal, C_{out}. All of the inputs and outputs of the circuit should be registered, similar to the structure shown in Figure 1*b*.

1. Create a new Quartus II project which will be used to implement the desired circuit on the Altera DE2 board. Use the *lpm_mult* and *lpm_add_sub* modules to realize the multipliers and adders in your design.

2. Connect the inputs A and C to switches SW_{15-8} and connect the inputs B and D to switches SW_{7-0}. Use switch SW_{16} to select between these two sets of inputs: A, B or C, D. Also, use the switch SW_{17} as a *write enable* (WE) input. Setting *WE* to 1 should allow data to be loaded into the input registers when an active clock edge occurs, while setting *WE* to 0 should prevent loading of these registers.

3. Use KEY_0 as an active-low asynchronous reset input, and use KEY_1 as a manual clock input.

4. Display the hexadecimal value of either A or C, as selected by SW_{16}, on displays *HEX7-6* and display either B or D on *HEX5-4*. The sum S should be shown on *HEX3-0*, and the C_{out} signal should appear on $LEDG_8$.

5. Compile your code and use either functional or timing simulation to verify that your circuit works properly. Then download the circuit onto the DE2 board and test its operation.

6. It is often necessary to ensure that a digital circuit is able to meet certain speed requirements, such as a particular frequency of a signal applied to a clock input. Such requirements are provided to a CAD system in the form of *timing constraints*. The procedure for using timing constraints in the Quartus II CAD system is described in the tutorial *Timing Considerations with VHDL-Based Designs*, which is available on the *DE2 System CD* and in the University Program section of Altera's web site.

   For this exercise we are using a manual clock that is applied by a pushbutton switch, so no realistic timing requirements exist. But to demonstrate the design issues involved, assume that your circuit is required to operate with a clock frequency of 220 MHz. Enter this frequency as a timing constraint in the Quartus II software, and recompile your project. The Timing Analyzer should report that it is unable to meet the timing requirements due to the lengths of various register-to-register paths in the circuit. Examine the timing analysis report and describe briefly the timing violations observed.

7. One way to increase the speed of operation of a given circuit is to insert registers into the circuit in a way that shortens the lengths of its longest paths. This technique is referred to as *pipelining* a circuit, and the inserted registers are often called *pipeline registers*. Insert pipeline registers into your design between the multipliers and the adder. Recompile your project and discuss the results obtained.

Part IX

The Quartus II software includes a predesigned module called *altmult_add* that can perform calculations of the form $S = (A \times B) + (C \times D)$. Repeat Part VIII using this module instead of the *lpm_mult* and *lpm_add_sub* modules. Test your circuit using both simulation and by downloading the circuit onto the DE2 board.

Briefly describe how the implementation of your circuit differs when using the *altmult_add* module. Examine its performance both with and without the pipeline registers discussed in Part VIII.