Hamid Zehtab 9912762541 Spring 2022

‘clk’ is the system clock,‘enable’ activates the multiplication operation,‘A’ is the 8-bit unsigned multiplicand input,‘B’ is the 8-bit unsigned multiplier input, and‘C’ is the 16-bit unsigned product output.

When designing multipliers there is always a compromise to be made between

how fast the multiplication process is done and how much hardware we are

using for its implementation.

A simple multiplication method that is slow‚ but efficient in use of hardware

is the shift-and-add method. In this method‚ depending on bit *i* of operand *A‚*

either operand *B* is added to the collected partial result and then shifted to the

right (when bit *i* is **1)**‚ or (when bit *i* is **0**) the collected partial result is shifted

one place to the right without being added to *B.*

This method is justified by considering how binary multiplication is done

manually. Figure 11.2 shows manual multiplication of two 8-bit binary

numbers.

We start considering bits of *A* from right to left. If a bit value is **0** we select

00000000 to be added with the next partial product‚ and if it is a **1**‚ the value of

*B* is selected. This process repeats‚ but each time 00000000 or *B* is selected‚ it

is written one place to the left with respect to the previous value. When all bits

of *A* are considered‚ we add all calculated values to come up with the

multiplication results.

Understanding hardware implementation of this procedure becomes easier

if we make certain modifications to this procedure. First‚ instead of having to

move our observation point from one bit of *A* to another‚ we put *A* in a shiftregister‚

always observe its right-most bit‚ and after every calculation‚ we move

it one place to the right‚ making its next bit accessible.

Second‚ for the partial products‚ instead of writing one and the next one to

its left‚ when writing a partial product‚ we move it to the right as we are writing

it‚ and the next one will not have to be shifted.

Finally‚ instead of calculating all partial products and at the end adding

them up‚ when a partial product is calculated‚ we add it to the previous partial

result and write the newly calculated value as the new partial result.

Therefore‚ if the bit of *A* that is being observed is **0‚** *00000000* is to be

added to the previously calculated partial result‚ and the new value should be

shifted one place to the right. In this case‚ since the value being added to the

partial result is *00000000‚* adding is not necessary‚ and only shifting the partial

result is sufficient. This process is called *shift.* However‚ if bit of *A* being

observed is **1**‚ *B* is to be added to the previously calculated partial result‚ and