# Homework 04: Word-Based Parallel Multiplier in Verilog

**Due on April 12 23:59pm**

## Objective

In this assignment, we are going to practice several tricks in Verilog:

1. Behavior (or functional) description, which is a higher-level design description than the Boolean expression in the last assignment, using the `always` block with `for` and `if-else` statements inside;
2. Hierarchical design style (building up a larger design with smaller sub-blocks);
3. Validation of the simulation results in the test stimulus automatically.

## Problem Description

It is a common design practice to partition a large problem into smaller pieces. One of the two major reasons doing so is that, sometimes, only the smaller problems can be affordable in terms of design complexity and implementation cost. The other reason is that existing designs can be reused to build up larger systems without reinventing the wheel every time. Therefore, most of the realistic designs are ***hierarchical***.

The parallel multiplier calculates the multiplication of two values at once. A large multiplier can be divided into sub-blocks in different ways. Among them, the word-based multiplier consists of smaller parallel multipliers that perform the multiplication of two smaller pieces of the inputs. (Usually the smaller pieces of the inputs are known as *words*.)

In this homework assignment, you are going to design a multiplier of two 8-bit unsigned integers in a hierarchical manner. (All the integers are unsigned in this assignment. So we will not specify it explicitly in the rest of the description.)

1. First of all, design a 4-bit Multiply-and-ACcumulate (MAC) block which computes
$$out = a * b + c$$
where $a$ and $b$ are two 4-bit integers, $c$ is a 5-bit integer, and $out$ is an 8-bit integer. Implement it in Verilog. How do you verify the design?
Hint:
```
module mac (out, a, b, c);
```
2. Use the 4-bit MAC blocks as the basic building blocks to construct the multiplier of two 8-bit integers.
Hint:
```
module multiplier (out, a, b);
```

What is the bit width of the out when both a and b are 8 bits?

In addition to the MAC blocks, will it help if you are allowed to use the adder blocks? If yes, design the adder module to complete the multiplier. What kind of the adders, and how many of them will you use to implement the 8-bit multiplier together with the MAC blocks?

**Note:**

a. The simpler and fewer the adders/MACs, the better the design.
b. Only the 4-bit MAC and adder instances can be used at the top level (inside the module `multiplier`). Neither other kind of block, nor high-level operator is allowed. You may use multiple MAC instances. But they must remain the same specification in (1). If you need two or more adder blocks of different bit widths, you may use different module references. Or you may create a single adder module and take advantage of the `parameter` statement to parametrize different instances.

**Optional:**

Is it possible to use the MAC blocks only to complete the multiplier without the adders and any other blocks/operators?

3. Draw the block diagram of your design. Discuss if the I/O bit widths of the MAC block defined in (1) is sufficient (and correct) or not, to implement the 8-bit multiplier.
4. Implement the Verilog design, and verify it with your own test patterns. Show the waveforms of at least 6 test patterns and their output responses in nWave.
5. Applying exhaustive test patterns to the 8-bit multiplier is feasible. But for that many patterns ($2^8 \times 2^8 = 2^{16}$ in total!!), we would like to have an auto-verification. Complete the following test template to verify the design.

```
`timescale 1ns/1ns
module test;

parameter pattern = (1 << 16); // 2^16

reg [7:0] a, b;
integer i, error;

...  // Complete the missing part

initial begin
  a = 0;
  b = 0;
  error = 0;
  for (i = 0; i < pattern; i = i + 1) begin
```

```
        a = i[15:8];
        b = i[7:0];
        golden = a * b;
        #10
        $display("<%5d> a=%d | b=%d | out=%d (%b) [%d (%b)]",
          i, a, b, out, out, golden, golden);

        // Insert if statement to display "Mismatched!" when
        // the out is not equal to the golden.
        // And increase the error count when it happens.

      end

      // Insert if statement.
      // Assume there is/are any error(s), e.g., 54 errors,
      // display something like
      //   "\n<<< 54 ERROR(S)!! >>>\n"
      // If no error, display
      //   "\n<<< PERFECT!! >>>\n"

      #100 $finish;
    end
    endmodule
```

6. Write a report to summarize all the discussions.

## Note

1. **Raise the discussion for any questions when in doubt.**
2. **Submit the source code and the electrical report based on TA's instructions.**