# Lab 3

## A Large Integer Multiplier Architecture

In this lab, you are required to design and implement a large integer multiplier hardware with a 16-bit datapath (an arithmetic unit with 16-bit inputs and output). The input operand length may vary from 16-bit to 128-bit (It is guaranteed that input operands will have the same length and be multiple of 16 bits.). In this lab, you will learn how to use Block RAM and DSP blocks in Xilinx FPGAs.

Algorithm for hardware-friendly integer multiplication operation is shown in Algorithm 1. The Algorithm divides input operands into 16-bit pieces and performs each 16-bit×16-bit multiplication operation separately. Then, resulting partial products are added using adders. For example, for $k = 3$, $A$ and $B$ inputs will be 48-bit integers where $A_0$, $A_1$, $A_2$, $B_0$, $B_1$, $B_2$ are 16-bit pieces (note that $A_0$ and $B_0$ are least-significant 16-bits). For $k = 3$, $C$ output will be 96-bit integer where $C_0$, $C_1$, $C_2$, $C_3$, $C_4$, $C_5$ are 16-bit pieces. $T_H$, $T_L$, $carry$ variables in Algorithm 1 have 18-bit, 16-bit and 1-bit lengths, respectively.

**Algorithm 1** Hardware-friendly Integer Multiplication Algorithm

```
Input: $A = \sum_{i=0}^{k-1} A_i \cdot 2^{16.i}$, a $(16.k)$-bit positive integer where $A_i < 2^{16}$ and $1 \le k \le 8$
       $B = \sum_{i=0}^{k-1} B_i \cdot 2^{16.i}$, a $(16.k)$-bit positive integer where $B_i < 2^{16}$ and $1 \le k \le 8$
Output: $C = \sum_{i=0}^{m-1} C_i \cdot 2^{16.i}$, a $(16.m)$-bit positive integer where $C_i < 2^{16}$ and $m = 2k$
 1: $C = 0$
 2: for $i$ from 0 to $(k-1)$ do
 3:     $carry = 0$
 4:     $P = 0$
 5:     for $j$ from 0 to $(k-1)$ do
 6:         $\{T_H, T_L\} = (A_i \times B_j) + P + carry$        ▷ DSP#1
 7:         $\{carry, C_{i+j}\} = T_L + C_{i+j}$                ▷ DSP#2
 8:         $P = T_H$
 9:     end for
10:     $C_{i+k} = P + carry$                                 ▷ DSP#2
11: end for
12: return $C$
```

Your design should take inputs first according to I/O protocol which will be detailed later in this document. Each input operand should be stored in one simple dual-port Block RAM (see *Xilinx_ISE_BRAM_Tutorial.pdf* document for generating Block RAM) with 16-bit data length and 8 address location. In other words, each 16-bit of the operands will be stored in one Block RAM address. In your design, you will use one Block RAM for each input operand. After storing inputs in Block RAMs, your design should perform multiplication operation according to Algorithm 1. Each operation in steps 6 and 7 of Algorithm 1 can be implemented using one DSP block (the operation in step 10 can also be implemented using the same DSP block with step 8). In your design, you are allowed to use only 2

DSP blocks (see *Xilinx_ISE_DSP_Tutorial.pdf* document for generating DSP block). In your design, you should store your output in one Block RAM (with 16-bit data length and 16 address location) as well. After storing your output in Block RAM, your design should output the result starting from least-significant 16-bit.

The multiplier has the following I/O protocol (please note that *rst* signal is not shown for simplicity):

1. Multiplier accepts new inputs if *busy* signal is 0.
2. If multiplier accepts new inputs, length of the operands should be given as inputs first. For providing input operand length, *len_in* signal should be 1 for one clock cycle and *len* signal should be entered in the next clock cycle.
3. As *len* signal is given as input, *busy* signal should be set as 1 (and remains as 1 until multiplication is done.).
4. Arbitrary length of time will pass (0 to infinite clock cycles).
5. For providing input operands, in_valid signal will be 1 for a single clock cycle. Then, first operand will be entered as 16-bit pieces continuously after this clock cycle using 16-bit *data_in* signal (least-significant 16-bit first). Then, the second operand will be entered continuously using *data_in* signal (least-significant 16-bit first).
6. Multiplier will start its operation as soon as it receives the last input. Multiplier will not accept any more input until it is done multiplying and outputs the result. An example timing diagram for input operation is shown in Figure 1.

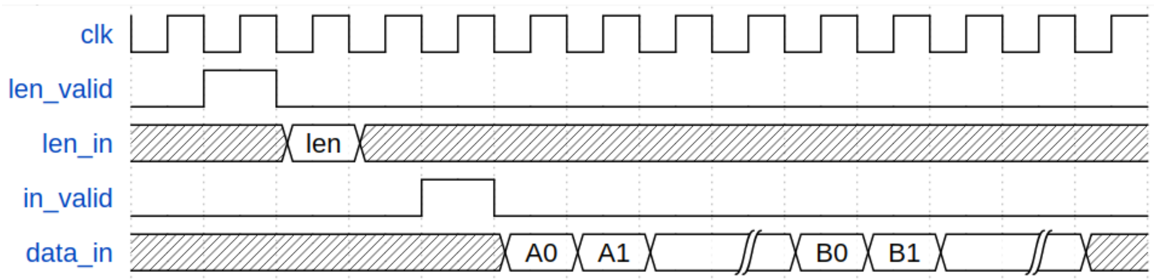

Figure 1: Input Protocol

7. As multiplier finishes its operation, *out_valid* signal will be 1 for a single clock cycle.
8. Result will be at the output continuously after this clock cycle (least-significant 16-bit first).
9. After the last 16-bit output is given, *busy* signal should be brought to 0 and multiplier should accept new operands to be multiplied. An example timing diagram for output operation is shown in Figure 2.

# Lab 3

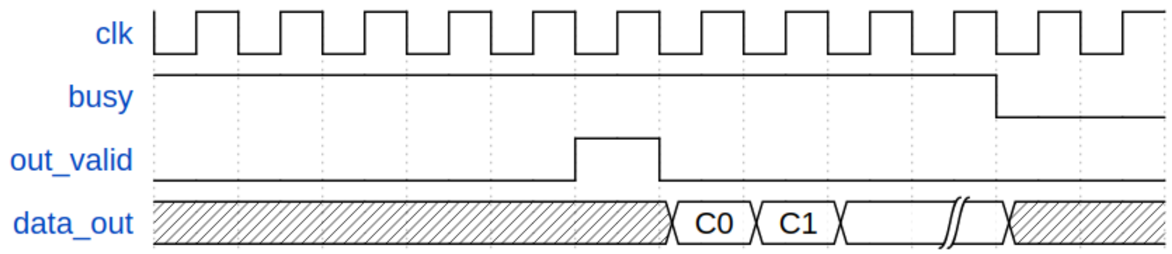

Figure 2: Output Protocol

Write a behavioral Verilog model for your hardware design. In your Verilog model, your Verilog module should have the following interface:

```
module IntMul(clk, rst, len_valid, len_in, in_valid, data_in,
                     busy, out_valid, data_out);
    input clk, rst;
    input len_valid, in_valid;
    input [2:0] len_in;
    input [15:0] data_in;
    output reg busy, out_valid;
    output reg [15:0] data_out;

    // ...

endmodule
```

You should also write a Verilog testbench that verify the correctness of your Verilog model. In the testbench, you should apply input values to your model and compare its outputs with the expected results. You should test your design with 25 test cases (make sure that you test your design for all possible operand lengths). You can either create your test vectors using a script (and read these inputs/outputs from **.txt* files) or generate test vectors in testbench. Please note that you should use your own test cases. Using your friend's test cases will be considered as plagiarism. After verifying the correctness of your design, synthesize and implement your design for Artix-7 FPGA (Family:Artix 7, Device:XC7A100T, Package:CSG324, Speed:-3) using Xilinx ISE WebPack 14.7 (**You will not implement your design on an FPGA board. You will just report implementation results.**). Then, you should write a lab report (in .pdf format) with *readable* English and good structure (dumping many figures and raw synthesis/implementation reports into a pdf file is not a lab report.). Your report should include the following information:

1. Write a summary of the assignment and your design.
2. What is the hardware area-cost (slices, register, LUTs, DSPs, BRAMs, etc.) of your design? What is the maximum achievable frequency of your design?
3. How many clock cycles does it take to finish one multiplication operation for each operand length?

Finally, put all of your Verilog projects, your script to generate test vectors (if you use a script for generating test vectors) and your report into one ".zip" file named Lab3_username(s).zip (e.g. Lab3_berkea_erdinco.zip) and submit this zip file using EE310 SUCourse website. Please note that we will use a plagiarism detection software for your lab assignments.