ALGORITHM:
LINK --> Concatenation action
SHy L --> Lefshift by y bits
SzE --> Extension to z bits

Logic for sign extending a n-bit number to (n + h) bits :
1) A n-bit unsigned number can be represented as a positive (n + h)-bit signed number having all the h additional bits set to 0.
2) A n-bit signed number can be represented as a signed (n + h)-bit number having all the h additional bits equal to the nth.

We use the following identities given in the research paper for doing the multiplication with different precision:
(1) A[31:0] * B[31:0] = SH16L(A[31:0] * B[31:16]) + S64E(A[31:0] * B[15:0])

(2) A[31:0] * B[31:0] = (A[31:16] * B[31:16]) LINK (A[15:0] * B[15:0])
+ SH16L (S64E(A[31:16] * B[15:0]) + S64E(A[15:0] * B[31:16]))

(3) A[31:0] * B[31:0] = SH16L (SH8L(A[31:0] * B[31:24]) + S48E(A[31:0] * B[23:16]))
+ S64E (SH8L(A[31:0] * B[15:8]) + S48E(A[31:0] * B[7:0]))

Using the Fig. 1 of the research paper, the block diagram of the new variable precision SIMD multiplier is depicted. It can be seen that its main building modules are: a control unit (CU) needed to properly set the computational flow on the basis of the required precision; four 32*9 CBPxS multipliers (M1, M2, M3, and M4) each generating 40-bit output;2 two 48-bit and one 64-bit carry propagate adders (CPA1, CPA2, and CPA3).

The CU receives the signals prec1, prec0, SA, and SB as input and generates all the signals needed to orchestrate extensions and dataflow as summarized in Table I of research paper. The signal SA indicates if the operand A is a signed (SA = 1) or an unsigned (SA = 0) number. The same occurs for the operand B on the basis of SB . Conversely, the signals prec1 and prec0 serve to establish the required precision, as shown in Table I.

In order to orchestrate the extension of the operands, subwords and the dataflow through the blocks MUXj , with j = 1; . . . ; 6 the CU generates appropriate control signals. samsb, sammsb, samlsb, salsb, salmsb, and sallsb indicate if the modules for extension S32E have to treat the subwords A[31:16]; A[31:24]; A[23:16]; A[15:0]; A[15:8] , and A[7:0] , respectively, as signed or unsigned numbers. Analogously, sbmmsb, sbmlsb, sblmsb, and sbllsb indicate if the modules S9E must consider the subwords B[31:24]; B[23:16]; B[15:8], and B[7:0], respectively, as signed or unsigned numbers. The CU also generates the signals Sj and SAMi . The former are used as the select input of the multiplexing blocks MUXj , whereas the latter indicate if the data InAi input to the multipliers Mi are signed or unsigned numbers.

The above control signals allow the required operation to be matched and the appropriate data flow to be set as summarized in Table I. When 32*32 multiplications are executed, the multipliers M1–M4 calculate in parallel the products between the operand A and the subwords B[31:24]; B[23:16]; B[15:8], and B[7:0] all extended to 9 bits. Then, the 40-bit results PR1 and PR3 are left shifted by 8-bipositions, whereas the 40-bit products PR2 and PR4 are extended to 48 bits. The two 48-bit words Ris1 and Ris2 obtained in this way are added by means of the carry-propagate adder CPA1, which generates the 48-bit word SUM1. At the same time, Ris3 and Ris4 are added by CPA2 forming the word SUM2. Then, SUM1 is left shifted by 16-bit positions, thus forming the 64-bit partial result PPR1, whereas SUM2 is extended to 64-bit, thus forming the partial result PPR2. To generate the final 64-bit result OUT[63 : 0], PPR1 and PPR2 are added by CPA3. It can be noted, that the hardware used could support the execution of two 32 * 16 multiplications. However, a 64-bit output register is used, thus only one 32*16 multiplication result can be accommodated in this register. When this operation is required just two multipliers must operate, whereas the others can be stopped. We have chosen to make the multipliers M3 and M4 able to go into action and to stop M1 and M2. To control this particular case the signals able1 and able2 are used. When able1 is low, all the 40 bits of PR1 and PR2 (and consequently, also the 48 bits of Ris1 and Ris2) are forced to zero. Therefore, the whole result OUT[63 : 0] is equal to PPR2, which is directly outputted by means of MUX5 and MUX6, without passing through CPA3. When operations on 16-bit data are executed, MUX4 and MUX6 allow the less significant 32-bit of the independent results SUM1 and SUM2 to be directly outputted without passing through CPA3. Analogously, when multiplications on 8-bit data are requested, MUX4 and MUX6 allow the less significant 16-bit of the independent results PR1, PR2, PR3, and PR4 to be directly outputted without passing through the addition modules CPA1, CPA2, and CPA3.

Thus instead of sequentially multiplying pairs of numbers , we can simultaneously multiply different pairs of numbers depending on their size. Hence we can save time. This is kind of an example of parallel processing in an FPGA based processor.

* We have implemented each of the elements in a separate module.

Group Members :
A. S. Akil Arif Ibrahim(15116001)
Arshdeep Singh(15116009)
Ashutosh Parija(15116010)
Gourav Dhar(15116021)