I-CHIP PS 1

SHRAVANI MAMIDWAR (22095059)
KARANAM SRIMAYI (22095051)

The Processing Element(PE) contains a floating point multiplier and a floating point adder.

DUT:

Firstly, we started with defining the size of the arrays and the initial number in PE, i.e; 10.0. Then we declared 2 arrays, ‘P’ and ‘Q’ (of size ‘N’) of input ports, with each input of size 32 bits and an output port ’R’ of reg type with size of 32 bits.
In the module, we declared two more arrays ‘pro’ of size N and each element of size 32 bits (with its ‘i’th index storing the product ‘i’th elements of the arrays) and ‘sum’ of size N-1 whose ‘i’th index stores the sum of products upto that index.
If the arrays contain only one element, the floating point multiplication is just the product of the two elements, and sum is the sum of this product and the initial value in PE(10.0). Else, the floating point multiplication is the product of elements at each index and sum is the sum of products upto that element, for which a for loop is iterated.

MODULE DEFINITIONS:
Now the module “FloatingMultiplication” is defined such that it multiplies two numbers in IEEE 754 standard and returns result in IEEE 754 single precision.
Inputs ‘A’ and ‘B’ each of 32 bits and an output reg of 32 bits is defined for 2 numbers in IEEE 754 standard form.
After extracting sign, Exponent, Mantissa from both the numbers, we obtained the temporary exponent and mantissa for the product by simply adding the exponents(with bias 127) and multiplying the mantissa.
If Temp_Mantissa[47]==1 that means our actual mantissa is from [46:24] and 1 will be added exponent else our exponent remains the same and original mantissa lies in [45:23]. So, we can find out the mantissa and exponent of the floating point multiplication.
The sign of our result it is just XOR of sign of A and B. So, we can obtain the final result in IEEE 754 standard form.

The module for floating point addition, “FloatingAddition” is defined by firstly declaring the similar variables as of floating point multiplication.
After deciding which of the two numbers is greater (using a variable “comp”), we stored the greater of the two numbers in A and smaller one in B.
If A and B are equal in magnitude and opposite in sign, their sum is zero.

The variable diff_exponent gives the magnitude of difference of exponents. The mantissa of smaller number is right shifted by difference of the exponents.
The sum or difference of mantissas is calculated depending on the sign difference between A and B. If A_sign is different from B_sign, it adds the mantissas (A_Mantissa + B_Mantissa). Otherwise, it subtracts (A_Mantissa - B_Mantissa). The result is stored in Temp_Mantissa, and carry indicates if there was a carry-out during addition or borrow during subtraction.
The variable exp_adjust is initialized with the exponent of A (A_Exponent). This variable will be adjusted later based on the result of addition or subtraction.
If there is a carry, it indicates overflow during addition. In this case, Temp_Mantissa is right-shifted by 1 bit to normalize it, and exp_adjust is incremented by 1 to adjust the exponent.
When there is no carry, it implies no overflow, so it enters a loop to left-shift Temp_Mantissa and decrement exp_adjust until the leading bit (bit 23) of Temp_Mantissa becomes 1, effectively normalizing the mantissa.
Sign of the final result, i.e, the floating point addition is simply the sign of the greater of the two numbers. So, the final result of floating point addition is determined.

TESTBENCH:

Firstly, the size of arrays is defined. Input arrays A and B(reg type) of size N, each element of 32 bits and output of 32 bits are declared. Then, DUT instantiation followed by stimulus generation is done. Final result is displayed.