ELEC6016 Digital System Design

SystemVerilog Design of an Embedded Processor

Data formats for the affine transformation algorithm

Affine transformation and fixed-point representation

For the two-dimensional affine transformation implemented in your picoMIPS:

$$\begin{vmatrix} x_2 \\ y_2 \end{vmatrix} = A \times \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} + B$$

use the following data formats. Pixel coordinates [x1,y1] and [x2,y2] and coefficients of vector B are 8-bit 2's complement signed integers, i.e. their values are in the range -128..+127.

The coefficients of matrix A are 2's complement signed fixed-point fractions in the range -1 .. +1 – 2^(-8), i.e. they are 2's complement fractional numbers with the radix point positioned after the most significant bits. Therefore the weights of the individual bits are:

Bit position	Weight
7 (MSB)	-2^0
6	2^(-1)
5	2^(-2)
4	2^(-3)
3	2^(-4)
2	2^(-5)
1	2^(-6)
0 (LSB)	2(-7)

When coefficients of the matrix A are multiplied by pixel coordinates, a double-length 16-bit product is obtained which is a 2's complement number with the radix point positioned after the 9-th bit. Note however that the result [x2,y2] must be an 8-bit 2's complement whole number.

Binary multiplication examples

Binary multiplication of 2's complement 8-bit numbers yields a 16-bit result. As one of the numbers is represented in the range -1..+1-2^{-7} and the other in the range -128..127, it is important to determine correctly which 8-bits of the 16-bit result represent the integer part which should be used for further calculations. The following examples might help.

Example 1. Multiply 0.75 x 6.
In 2's complement 8-bit binary representation these two operands are:

weights:	-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
0.75 =	0.	1	1	0	0	0	0	0

weights:	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
6 =	0	0	0	0	0	1	1	0.

The 16-bit result is:

-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
0	0	0	0	0	0	1	0	0.	1	0	0	0	0	0	0

which represents the value of 4.5. The shaded area shows which bits need to be extracted when the representation is truncated to 8 bits. Note that the fraction part is discarded entirely, so the 8-bit result is now 4. Also note that when the leading bit is discarded, the weight of the new leading bit must now change from 2^7 to -2^7. Why? The importance of the correct interpretation of the leading bit's weight is evident in the following example, where the result is negative.

Example 2. Multiply -0.25 x 20.
The two operands in signed binary are:

weights:	-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
-0.25 =	1.	1	1	0	0	0	0	0

weights:	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
20 =	0	0	0	1	0	1	0	0.

The 16-bit result is:

-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
1	1	1	1	1	1	0	1	1.	0	0	0	0	0	0	0

which is -5 in decimal representation. Again, the shaded area shows which 8 bits to extract when the result is truncated from 16 to 8 bits for further calculations. The truncated 8-bit result has the weight of -2^7 on the most significant bit so that it still correctly represents -5:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	0	1	1.

If you would like to experiment more with binary multiplication of signed numbers, run some SystemVerilog simulations of your multiplier in Modelsim, which is what you should do anyway to test your multiplier module before synthesis. If you would like to practice signed binary multiplication by hand, I recommend you use Booth's algorithm (and rather fewer than 8-bits!):
en.wikipedia.org/wiki/Booth's_multiplication_algorithm

The original paper where Andrew Booth published his algorithm back in 1951 can be found here: http://qjmam.oxfordjournals.org/content/4/2/236.short

tjk, 4 Feb'14