**EDN design ideas**

**Double-speed interpolation**

[MICHAEL MAHON](http://www.edn.com/user/mjmahon) - September 18, 2013

It is sometimes necessary to perform linear interpolation or “blending” of two vectors. For example, this is done to “mix” two audio waveforms or blend two images.  The equation is:

***out*  = a•*in1* + (1-a)•*in2***

where ***out*** is the output vector, ***in1*** and ***in2*** are the two input vectors, and **a** is the fraction of ***in1*** to be blended with the complementary fraction of ***in2***.

For each element of the vectors, this computation usually requires two multiplications and an addition. While this is not particularly onerous for a processor with a fast multiply operation, it can be a significant performance issue for processors that must synthesize multiplication by repeatedly shifting and adding. This Design Idea simplifies the computation on processors that do not have a fast multiply by evaluating the complete equation at the cost of just one shift-and-add multiply loop, as shown below in 6502 code for 8-bit operands:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* BLEND \*

\* \*

\* Called with blend fraction (0..1-) in Accumulator \*

\* X register = vector subscript i \*

\* (IN1 and IN2 are input vector base addresses) \*

\* \*

\* Upon exit, Accumulator = OUT(i) \*

\* factor = low 8 bits of OUT(i) (unused) \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

blend lsr ; Low bit of factor to CARRY

 sta factor ; and save shifted result.

 lda IN2,x ; Init sum to IN2(i)

 ldy #8 ; 8 iterations

blender bcc addIN2 ; bit off: add IN2(i)

 clc ; bit on: add IN1(i)

 adc IN1,x ; (after clearing CARRY)

 jmp shift

addIN2 adc IN2,x ; add IN2(i) (CARRY already clear)

shift ror ; (shifts in any carry out)

 ror factor ; Next bit of factor to CARRY

 dey

 bne blender ; 8 iterations

 rts ; Return with OUT(i) in Accumulator.

The following is a brief summary of the 6502 operations used in this algorithm (with the exception of the Store, Load, and Jump instructions, which do just what you’d expect):

|  |  |  |
| --- | --- | --- |
| lsr | Logical Shift Right | Shift the accumulator right one bit, shifting a zero into the MSB and shifting the LSB out into the CARRY toggle. |
| bcc | Branch if Carry Clear | Branch if CARRY toggle is 0 |
| clc | CLear Carry | CARRY = 0. |
| adc | ADd with Carry | Accumulator = Accumulator + operand + CARRY.  Therefore, CARRY is typically cleared prior to doing 8-bit adds. |
| ror | ROtate Right | Shift the operand (either a memory byte or the accumulator if no operand is specified) one bit to the right, shifting the CARRY toggle into the MSB and the LSB out into the CARRY toggle. |
| dey | DEcrement Y | Decrement the Y register by 1, setting the Equal toggle if result is 0. |
| bne | Branch Not Equal | Branch if the Equal toggle is set. |
| rts | Return from SubRoutine | Returns from a subroutine that has been entered by a Jump to SubRoutine instruction. |

The hash prefix “#” is used to indicate a literal value.

The operation of the algorithm is straightforward.  For every bit of the blend factor that is one, the first input is added to the sum, and for every bit that is zero, the second input is added.  Because **a** and **(1-a)** are complements, the loop directly computes the sum of the two products in the time normally required by a single multiply loop.

The blend fraction, **a**, is an 8-bit unsigned fraction.  **IN1**, **IN2**, and **OUT** are all 8-bit unsigned integers.

Note that the Accumulator is initialized to **IN2(i)** so that if the blend factor is zero, the value returned is equal to **IN2(i)**.

**Also see**:

* [Bit-shifting method performs fast integer multiplying by fractions in C](http://www.edn.com/design/systems-design/4327349/Bit-shifting-method-performs-fast-integer-multiplying-by-fractions-in-C)
* [The product of all fears: binary multiplication](http://www.edn.com/design/systems-design/4341285/EDN-Access-04-10-97-The-product-of-all-fears-binary-multiplication)
* [In the days of old, when engineers were bold](http://www.edn.com/electronics-blogs/tales-from-the-cube/4315210/In-the-days-of-old-when-engineers-were-bold)
* [Retro-emulation](http://www.edn.com/electronics-blogs/fpga-gurus/4306486/Retro-emulation)
* [HP Virtualizes Its Calculators for the iPhone and PC](http://www.edn.com/electronics-blogs/other/4307064/HP-Virtualizes-Its-Calculators-for-the-iPhone-and-PC)