Using the INTERP Instruction

When adjusting delay length in an algorithm it is rare that the result of the calculations land directly on a sample, generally the result is a value that lands between samples. A programmer must then decide whether to use just the integer portion of the result to point at a sample but this can have undesired effects or to interpolate between samples but this adds a great deal of programming overhead. The FXCore addresses this issue by providing a dedicated interpolation instruction called INTERP.

INTERP will interpolate between two adjacent samples in delay memory and put the result in the ACC32 register. INTERP requires two pieces of information: the base address of the delay line and the offset into the delay line including fractional portion. The base address is coded into the instruction and the offset with fractional portion is placed in a register. The instruction in FXCore assembly would look something like:

interp r0, delay

The base address is generally from a .mem statement but the register value is a bit more complex, the upper bits in the register are the sample number and the lower are the fractional portion used as the interpolation coefficient. Using r0 as an example r0[31] is 0 as the address range is limited to 15-bits for 32K addressing, r0[30 - 16] contain the sample address offset and r0[15 – 0] contains the fractional portion.

Below is an example of using interp to create an adjustable delay of length 16384 samples and using the smoothed value of POT0 to control the length:

```
// Example using the INTERP instruction
// This example creates an adjustable delay on channel 0 using
// POT0 to control the delay from 0 to the length of the define
// delay. We calculate the depth into the delay by multiplying
// POT0 by the length, add it to the base of the delay and
// interpolate between adjacent samples

// First define a delay line of some length
.mem delay 16383

// Next read in ADC channel 0 to core register R0
cpy_cs r0, in0

// Now write it to the head of the delay
wrdel delay, r0

// Read the POT0 smoothed value into core register R1
cpy_cs r1, pot0_smth


// Put the length of the delay into the upper 16 bits of core register R2
// Remember that adding a “!” to the end of a memory block name returns the
// length of the delay not the base address and that wrdld put the value in
// the upper portion of the register and 0 in the lower 16-bits
wrdld  r2, delay!

// Multiply core register R1 and R2, result is in ACC32
multrr r1, r2

// At this point the upper 16-bits of acc32 are the depth into the delay
// and the lower 16-bits are the interpolation coefficient so interpolate
// between samples, result in acc32
interp acc32, delay

// We still have the input in core register R0 so simply add it to
// the interpolated result in ACC32 to add dry to delayed signal
adds acc32, r0

// Finally write it to DAC0
cpy_sc out0, acc32
```

As can be seen the program is very simple as we just read the ADC, write to the delay line, multiply POT0 by the length of the delay line to calculate where we want to read from, get the interpolated result, add the dry signal to the delayed one and put it out the DAC.

Additionally, there is only one number, the delay length, we need to deal with. All the rest is handled by the assembler or in the FXCore so it makes it very easy to change the delay length if you need to use memory for other effects.

With a few minor changes we can add feedback to the program:

```
// Example 2 using the INTERP instruction – an-1_b.fxc
// This example creates an adjustable delay on channel 0 using
// POT0 to control the delay from 0 to the length of the defined
// delay. We calculate the depth into the delay by multiplying
// POT0 by the length, add it to the base of the delay and
// interpolate between adjacent samples. We use POT1 as a feedback
// control of the delayed signal to add repeats

// First define a delay line of some length
.mem delay 16383
// Next read in ADC channel 0 to core register R0
cpy_cs r0, in0

// Add in the feedback, result in ACC32
adds r0, r3

// Now write it to the head of the delay
wrdel delay, acc32

// Read the POT0 smoothed value into core register R1
cpy_cs r1, pot0_smth
```

```
// Put the length of the delay into the upper 16 bits of core register R2
// Remember that adding a “!” to the end of a memory block name returns the
// length of the delay not the base address and that wrdld put the value in
// the upper portion of the register and 0 in the lower 16-bits
wrdld  r2, delay!

// Multiply core register R1 and R2, result is in ACC32
multrr r1, r2

// At this point the upper 16-bits of acc32 are the depth into the delay
// and the lower 16-bits are the interpolation coefficient so interpolate
// between samples, result in acc32
interp acc32, delay

// Copy the result to core register R3 for use in feedback
cpy_cc r3, acc32

// We still have the input in core register R0 so simply add it to
// the interpolated result in ACC32 to add dry to delayed signal
adds acc32, r0

// Finally write it to DAC0
cpy_sc out0, acc32

// Get POT1 smoothed value into R0
cpy_cs r0, pot1_smth

// We really want to limit it from 100% feedback to it doesn't just
// build up forever so put 0.9 in r1 but wrdld expects a 16-bit number
// not a decimal so multiply by 32767
wrdld r1, 0.9*32767

// Multiply the pot1 value by the limit, result in ACC32
multrr r0, r1

// Multiply by the delayed signal
multrr acc32, r3

// move it to R3 to be added to the dry next time
cpy_cc r3, acc32
```

Experimental Noize Inc.

Scottsdale, AZ USA

www.xnoize.com

sales@xnoize.com