Delay Line Tips for the FXCore

Introduction

The native delay line format in the FXCore is a 16-bit linear value, this format is fine for 99% of applications as a 16-bit number is a 96db dynamic range and most applications using a long delay line are processing and mixing in the delay as in reverb or delay programs.

However there is a small group of applications that can make use of a larger dynamic range or resolution. Here we look at 16-bit, 32-bit and floating point formats, any issues to be aware of and how to implement them on the FXCore.

16-bit delay line

As mentioned above, most applications can use the default 16-bit format. This is fine for reverb, chorus and most delays. This is also the simplest format to deal with in the FXCore:

```
.mem        sixteen     1024
cpy_cs      r0, in0           ; get left in

; save as a 16-bit value to memory
wrdel       sixteen, r0 ; save 16-bit to memory

; output the 16-bit value
rddel       r0, sixteen#      ; read in from delay memory
cpy_sc      out0, r0
```

As we can see we just read in an ADC channel, write to the delay line (the upper 16-bits are written to the delay memory) then read the tail of the delay line and write it to a DAC channel.

32-bit delay line

In some rare cases an algorithm may need to save 32-bit vales, in most cases this is just a few and MREGs will provide all you need. But there may be a case where you need a longer delay line of 32-bit values, this can be done with just a few instructions:

```
.mem        linearh     1024
.mem        linearl     1024

cpy_cs      r0, in0           ; get left in

; save as a 32-bit by writing both high 16 and low 16 bits to memory
```

```
wrdel       linearh, r0
sl          r0, 16
wrdel       linearl, acc32

; now output the 32-bit value
rddel       r0, linearh#      ; get the upper 16-bits
rddel       r1, linearl#      ; get the lower 16-bits
sr          r1, 16            ; push lower bits back down
or          r0, acc32         ; combine back to 32-bit word
cpy_sc      out0, acc32       ; output to out0
```

We basically define 2 16-bit delay lines where one will hold the upper 16-bits of each delay and the other the lower 16-bits. We read in the ADC, write the upper 16-bits to linearh, shift the lower 16-bits up into the upper 16-bits then write them to linearl.

To output the values we read back both linearh and linearl to CREGs, shift the lower 16-bits back down into position and OR the 2 CREGs together to make the 32-bit value.

While this preserves both the dynamic range and resolution it does use twice as much memory so you can only save 16384 samples to the delay memory.

Floating point delay line

While the 16-bit format uses the least memory it is also the smallest dynamic range. The 32-bit format give both a larger dynamic range and resolution but at a cost of using twice the memory.

Floating point falls between these two, it provides a shifting dynamic range while only using the 16-bit memory. Basically floating point maps larger ranges to a smaller number but trades resolution for range. Larger numbers have less resolution but we don't notice this as the samples are loud and as numbers get smaller we gain resolution which is desired for quieter samples.

If you are familiar with the FXCore instruction set you may think "but there is no floating point conversion in there". The trick is to realize that LOG2 is a specific type of floating point format and can be used to our advantage.

First let's consider the limits of the LOG2 function and how to deal with them:

1. **LOG2 is undefined for the value 0 like any other log**
   The LOG2 function in FXCore deals with 0 by automatically mapping it to the smallest value 0x00000001 so the user does not need to worry about this.

2. **LOG2 only works with positive numbers while samples are positive and negative**
   LOG2 will always take the absolute value of a number prior to conversion so the user does not need to worry about that aspect of it but they will need a way to remember the

sign of the number.

3. **As the programmer needs to track the sign of a number the programmer needs to use more memory to store this information.**
   Actually the sign can be saved in the LOG2 value its self. First we need to realize that all results of the LOG2 instruction in FXCore are negative. This is because all the numbers are 0 < X < 1.0 and all numbers < 1.0 have a negative LOG2 value. As a result the sign bit in the result of LOG2 is redundant meaning it will always be negative so we can actually use it to store sign information for the sample. In this case we will use it to save sign information for the sample.

4. **LOG2/EXP2 use lots of instruction cycles**
   This is true and as a result this method should ONLY be used when a user requires an expanded dynamic range, must use all memory and will only be doing a few memory accesses in the program.

Taking the above into account we can develop a short bit of code that can convert a linear sample into a floating point type number and another piece of code to convert it back:

```
.mem  float 1024

cpy_cs      r0, in0           ; get left in
; save a floating point format version to memory
wrdld       r1, 0x8000
and         r0, r1            ; isolate sign bit to acc32
cpy_cc      r2, acc32         ; save sign bit to r2
log2        r0                ; compress audio, sign bit will be set to 1
xor         acc32, r2   ; save NOT sign bit to sign bit of compressed data
wrdel       float, acc32

; now the convert it back
rddel       r0, float#        ; get the floating point version
wrdld       r1, 0x8000        ; set the sign bit of r1
xor         r0, r1            ; getting the sign bit
and         acc32, r1         ; isolate the result of sign bit extraction
cpy_cc      r2, acc32         ; save it
or          r0, r1            ; make sure sign bit is set
exp2  acc32                   ; expand from float to linear

; if r2 is 0 it means the original value was positive and we are done so jump
to output
; if r2 is not 0 then original value was negative so we need to negate it
jz          r2, out
neg         acc32       ; sample was negative originally so 2's comp the
result

out:
```

```
cpy_scout0, acc32       ; output to out 0
```

While we need to use a few instructions to implement it we do get the advantage of a larger dynamic range mapped into a 16-bit number

Conclusion

In most cases 16-bit linear is the best option as it provide a 96db dynamic range in a 16-bit value, but for the cases where greater resolution or greater dynamic range are required a little bit of code can allow a programmer to implement 32-bit or floating point delay lines..

Experimental Noize Inc. reserves the right to make changes to, or to discontinue availability of, any product or service without notice.

Experimental Noize Inc. assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using any Experimental Noize Inc. product or service. To minimize the risks associated with customer products or applications, customers should provide adequate design and operating safeguards.

Experimental Noize Inc. make no warranty, expressed or implied, of the fitness of any product or service for any particular application.

In no event shall Experimental Noize Inc. be liable for any direct, indirect, consequential, punitive, special or incidental damages including, without limitation, damages for loss and profits, business interruption, or loss of information arising out of the use or inability to use any product or document, even if Experimental Noize Inc. has been advised of the possibility of such damage.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Experimental Noize Inc. products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”). Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Experimental Noize Inc. products are not designed nor intended for use in military or aerospace applications or environments. Experimental Noize Inc. products are not designed nor intended for use in automotive applications.

Experimental Noize Inc.

Scottsdale, AZ USA

www.xnoize.com

sales@xnoize.com