Phaser Program

The other app notes for the FXCore are designed to detail a specific operation or function of the chip, this app note differs by combining information from the other app notes and the instruction documentation to present an entire program.

For this example we have elected to create a phase shifter program, we will assume it is running at 32KHz sample rate, it is selectable 2, 4, 6 or 8 stages and is using 4 of the POT inputs to control rate, range, resonance and mix.

First we need to generate a wave form to modulate the coefficient for the app-pass filters, we elect to use one of the built in LFOs and to allow a range of about 0 to 5Hz. From app note an-3 we know the equation to calculate the coefficient for the LFO for a given frequency is:

$C = (2^{31} - 1) * (2*pi*F)/F_S$

F = Target frequency in hertz
F_S = Sample rate

Solving this for 5Hz at 32KHz sample rate we get C = 2058874 so we need the rate coefficient for the LFO to range from 0 to 2058874. This can be accomplished with a simple multiply of the POT by this value.

Next we need to consider the range for the K coefficients for the all pass filters, from experience we feel that a range of 0.45 to 0.95 will provide a useful range. Given that the LFOs raw output is -1.0 to +1.0 we need to shift and offset this to generate the desired 0.45 to 0.95 sweep, the code below give an explanation of each step to do this. One thing to remember is phasors like to use all-passes with a positive feed forward and a negative feedback so we will need to invert the value as the apma and apmb instruction do the reverse of that. We save the coefficient in r14.

Speaking of apma and apmb, this set of all-pass instruction is perfect for phase shifting, these instructions do a single delay element all-pass using the MREG registers so all your delay memory is available for other algorithms that could run along with the phase shifter. Additionally it would be convenient to save the results of every two all-pass stages to MREGs as well and to make it easy to select the desired one based on the switch inputs. We decide to use MREG0 – MREG7 to store the delay for the 8 all-pass stages and store the result of every other stage to MREG8 - MREG11. By doing this we can simply read the S0 and S1 switch inputs, treat them as the LSBs of a value that we then add 8 to so the result ranges 8 to 11, we can then use the cpy_cmx instruction to use this value to select the MREG to read.

We want to limit the feedback so we do not overload the system so we decide to limit the resonance (feedback) to 0.875 of the output. We will save the feedback in r13

Finally we do the mix of the phased signal and dry signal as a simple interpolation using the POT as the interpolation coefficient, fully CCW is 100% dry and fully CW is 100% wet.

```
// Example combining the app notes to a full phase shifter program - an-5.fxc
// S1:S0 select 2,4,6 or 8 stages
// POT0 : rate
// POT1 : sweep range
// POT2 : resonance (feedback)
// POT3 : depth (mix)
// mr0 - mr7 are the AP delays
// mr8 - mr11 hold the result from each 2 stages for later selection
// r13 holds the feedback for resonance
// r14 has the K coefficient for ths APs
// r15 is used by the apma and apmb instructions so leave alone
// remaing registers are available for general use

.equ lfomax 2058874  // LFO max about 5Hz at 32K, see an-3 for equations
.equ maxfb  0.875    // Max feedback level
.equ rangebase -0.45*32767  // We want the K for the APs to range from about
-0.5 to -0.95
                            // but the "wrld" instruction want an unsigned
16-bit value so we
                            // play a trick with the assembler, we calculate
the 16-bit signed
                            // number but use the ".l" extension when we use
the value in wrdld
                            // as adding .l forces the assembler to treat the
number as an
                            // integer and mask off the lower 16-bits as an
unsigned number

// First set up the rate pot
cpy_cs r0, pot0_smth  // read POT0 into r0
multrr r0, r0         // square it so more control at lower range, result in
acc32
wrdld r0, lfomax.u    // since wrdld uses just a 16-bit value use upper 16
bits, we could
                      // load the full 32-bit number but the max lfo speed is
not critical
                      // in this program so losing the lower 16-bits is not
an issue
multrr acc32, r0      // multiply pot by max range
cpy_sc lfo0_f, acc32  // write it to LFO0 frequency control

// get the sin wave range -0.45 to -0.95 save in r14
// NOTE: Phasers want a positive feed forward and negative feed back
// so make K negative as apma inverts in the feedback path and apmb does not
cpy_cs r0, lfo0_s      // read in sin wave ranges -1.0 to +1.0 (well, almost)
sra r0, 2              // divide by 4 via right shift so ranges +/- 0.25
addsi acc32, -0.25     // now ranges 0 to -0.5
cpy_cs r0, pot1_smth   // get the range pot
multrr r0, acc32       // scale, result in acc32
wrdld r0, rangebase.l  // load in the base of -0.45 that we force the
assembler to treat as unsigned
adds r0, acc32         // add base so range can go from -0.45 to -0.95
cpy_cc r14, acc32      // save K in R14 for the APs
```

```
// get source and add feedback
cpy_cs r12, in0         // read from channel 0, put in r12 as we will want it
later and working from core
                        // registers is faster than mregs
adds r12, r13           // add feedback from r13

// shift 6 bits down to avoid clipping APs, recover at end.
sra acc32, 6            // divide by 64 for headroom

// Do the 8 all passes saving the result every 2 APs for later selection
apma r14, mr0           // AP 1
apmb r14, mr0

apma r14, mr1           // AP 2
apmb r14, mr1

cpy_mc mr8, acc32       // Save result for 2 stages

apma r14, mr2           // AP 3
apmb r14, mr2

apma r14, mr3           // AP 4
apmb r14, mr3

cpy_mc mr9, acc32       // Save result for 4 stages

apma r14, mr4           // AP 5
apmb r14, mr4

apma r14, mr5           // AP 6
apmb r14, mr5

cpy_mc mr10, acc32       // Save result for 6 stages

apma r14, mr6           // AP 7
apmb r14, mr6

apma r14, mr7           // AP 8
apmb r14, mr7

cpy_mc mr11, acc32       // Save result for 8 stages


// Look at the switches and select the number of stages accordingly
cpy_cs r0, switch       // read in the switch sfr
andi r0, 0x0003         // only keep S0 and S1
cpy_cc r0, acc32        // save them
andi acc32, 0           // clear acc32
ori  acc32, 0x0008      // put 8 into acc32
add  acc32, r0          // add the switchs to acc32 so ranges 8 to 11 which
happens
                        // to be the MRs used to save the AP results
cpy_cmx r0, acc32       // use acc32 as the pointer to the mreg to read
```

```
      sls r0, 6              // multiply by 64 to recover from the initial
shift, result in acc32

// use POT3 for mix level, full CCW is all dry (in0), full CW is full phase
shifter output (fpo)
subs acc32, r12         // acc32 = fpo - in0
cpy_cs r0, pot3_smth    // POT3 placed in r0
multrr r0, acc32        // acc32 = acc32 * POT3 = POT3*(fpo - in0)
adds acc32, r12         // acc32 = acc32 + in0 = POT3*(fpo - in0) + in0

// write to output
cpy_sc out0, acc32      // output it!

// adjust feedback level
cpy_cs r1, pot2_smth    // Read POT2
multrr acc32, r1        // Multiply the output by the feedback
wrdld r0, maxfb*32768   // We defined maxfb above as a decimal but need it to
be unsigned 16-bit for wrdld
multrr acc32, r0        // Multiply by limit
cpy_cc r13, acc32       // save it in r13 for next time
```

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 even 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