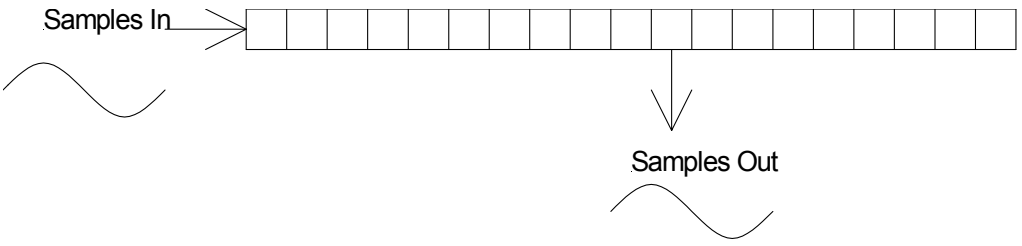
Basics of the LFOs in the FV-1

Frank Thomson
OCT – Los Angeles, CA

The Spin Semiconductor FV-1 contains a total of four LFOs; two SIN LFOs (SIN0 and SIN1) and two ramp LFOs (RMP0 and RMP1). These LFOs allows the user to create modulated effects like chorus and flange or to pitch shift a signal up or down.

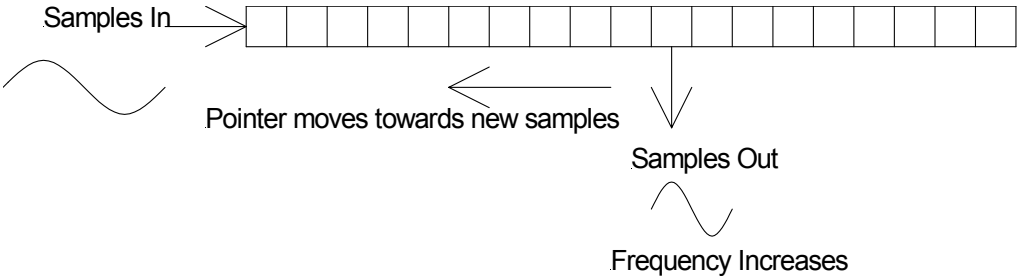
To generate a simple chorus effect in the chip requires the use of a SIN LFO. The purpose is to modulate a pointer in the delay memory to pitch shift the original signal up and down so when the result is mixed back with the original signal we hear a chorusing effect.

To illustrate this to make it clearer to the user, first take a delay where samples are inserted at the start and a pointer reads them from a particular point like:

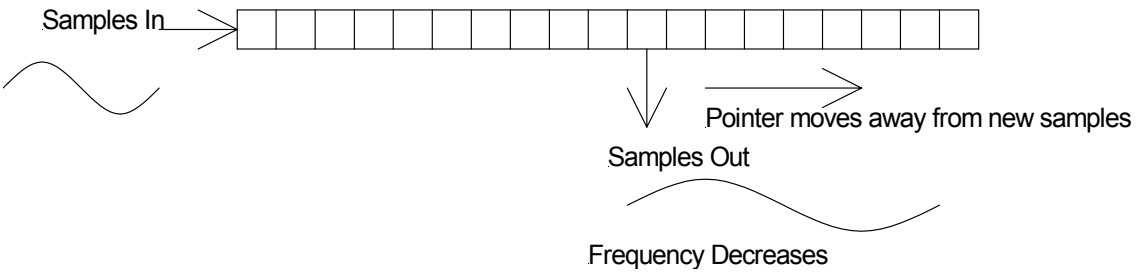


If the pointer we use for reading does not move, the output is simply the input delayed by the number of samples between the input and read pointer.

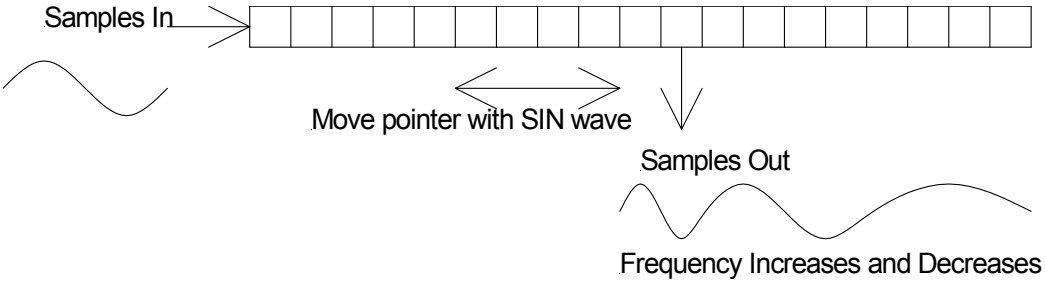
If we move the read pointer towards the incoming samples, then we have a rise in pitch as we are basically playing the samples faster than they are coming in:



Conversely, if we move the pointer away from the incoming samples then we will have a fall in pitch as we are basically playing the samples slower than they are coming in:



If we modulate the pointer with a wave form like a SIN wave then the output will shift up and down in frequency as the read pointer moves towards and away from the incoming samples:



To accomplish this in the FV-1, we need to do the following:

1. Define an area in memory to use as our delay memory
2. Set a read pointer to the middle of this delay
3. Modulate the position of the read pointer using the LFO

When we define the area in memory to use for delay, we must make sure it is long enough so that the pointer, when modulated by the LFO, does not go past the start or end of the delay. This length must be the same as the amplitude of the SIN wave. For example, if we select the SIN wave to go from -1024 to +1024 then the memory must be 2049 locations long.

We set the read pointer to the middle of the delay since the LFO goes both positive and negative, so if the delay goes from 0 to 2048 and we set the read pointer to 1024 then when we add the +/-1024 from the SIN wave we will not exceed either end of the delay.

Now we also need a way to place samples into our delay and move these samples through the delay. The FV-1 does this by using a down counter that is decremented once per sample period at the start of each new sample period. This down counter is added to the address for all memory accesses there by moving the samples through the delay. As a result of using a down counter, we write to delays to their lower address and read from a higher address (i.e. for a 20 sample delay write to address 0 and read from address 20). So for the case of the above chorus, we would write to address 0 and read from address 1024.

There is one small remaining item to deal with and that is the fact that we will normally not be landing on whole samples while modulating the pointer with the SIN wave. This means we will want to interpolate between adjacent points to generate the chorus output, the SIN generator can assist in that its output has bits below those added to the address

pointer and we can use these bits as interpolation coefficients for linear interpolation. The linear interpolation is performed by:

Sample[N]*(K-1) + Sample[N+1]*K

where K is the “fractional” bits (those below the ones used for addressing) from the SIN generator and are considered to be <= 1.0

Now, using the above information we can write a program to take an input, place it in a delay, read from a pointer modulated by one of the SIN LFOs and generate an output by linearly interpolating a result.

The equation to calculate the coefficient for a particular LFO frequency for the SIN LFO for use in a WLDS instruction is:

$$Kf = 2^{17} * \left(\frac{2\pi f}{R} \right)$$

Where
f: Desired LFO frequency in Hz
R: Sample rate in Hz
Valid values are in the range 0 to 511 for Kf which results in a frequency range of about 0 to 20Hz at a sample rate of 32,768Hz

The equation to calculate the coefficient for the amplitude for a given delay length for use in a WLDS instruction is:

$$Ka = \frac{N * 32767}{16385}$$

Where:
N: Delay length in samples
Valid values are in the range of 0 to 32767 for Ka.

```
; From Application note AN-0001
; Program AN0001-1.spn
;
; Memory declarations
delayl mem 8193 ; Left delay
delayr mem 8193 ; Right delay
; Initialization, only run on first execution of code
; Skip to the START label if NOT the first time
skp run,START
; Load up a sin LFO, this is about 0.2Hz
; and +/-4096 samples for a total delay requirement of 8193
wlds SIN0,5,16384
; End of skip/initiaization
; Main program code
; First, read in the sample and write to the start of the delay
; Read in the current left sample, ADCL -> ACC
START: ldax ADCL
; Write it to left delay and clear ACC
```

```
wra delayl,0
; Read in the current sample, ADCR -> ACC
ldax ADCR
; Write it to right delay and clear ACC
wra delayr,0
; We use the middle of the sample memory block as the
; address since we are using a signed sine wave
; that ranges -1.0 to 1.0 (-4096 to +4095 in this case)
; The following instruction will read LFO 0, "REG" it in a temp register
; in the LFOblock (we don't want it to change while we are doing calculations)
; add the "integer" portion to the address of the middle of the delay
; and down counter and finally multiply the sample from the delay
; memory by (1-K). That's a lot for one instruction! The "rda"
; tells the ALU to add the result of the multiplication to the ACC,
; similar to a normal rda instruction. You can combine rda, wra, sof
; or rdal with a chorus instruction.
; (1-k)*sample[addr]
cho rda,SIN0,SIN|REG|COMPC,delayl^
; Now we have sample[N] * (1-interpolation coeff) in the accumulator,
; now need to generate sample[N+1] * (interpolation coefficient) and
; add it to the value in the accumulator. Note that the address used
; in the instruction is 1 greater than the preceding instruction.
;k*sample[addr+1] + ACC
cho rda,SIN0,SIN, delayl^+1
; Interpolated sample in ACC, write it to DACL and clear ACC
wrax DACL,0
; Repeat for right channel. Since we are using the same LFO and output
; we do not need to REG the LFO again.
cho rda,SIN0,SIN|COMPC,delayr^
cho rda,SIN0,SIN,delayr^+1
wrax DACR,0
; That's it!
```

Note that the values used here for the SIN generator of 0.2Hz and +/-4096 are extreme to illustrate use of the LFO. To really hear the chorus effect, the result of the chorus operation should be mixed back in with the original signal.

One of the most powerful aspects of the FV-1 chip is the ability to change the frequency and amplitude coefficients in real time by writing to the SINX_RATE, SINX_RANGE, RMPX_RATE and RMPX_RANGE registers within a program (replace 'X' with 0 or 1 to select the appropriate SIN or RMP LFO). This makes it possible to use the potentiometer inputs (or even the input signal its self) to control an LFO.

The key to this is understanding how the accumulator is mapped to the control values for the SIN LFO:

ACC[22:14] is mapped to the 9-bits that control frequency
ACC[22:8] is mapped to the 15-bits that control amplitude

To convert between the calculated Kf and Ka values and their equivalent decimal values:

Kf(decimal) = Kf/512

Ka(decimal) = Ka/32768

As an example, we will write a program that will use the POT0 input to control amplitude and POT1 to control frequency of LFO 0 and to output the sine wave on the left output. We will want amplitude to cover the entire range from 0 to +/-16384 (Ka=0 to 32767) and frequency to range from 5Hz to 20Hz (Kf=125 to 500 at a sample rate of 32768Hz)

Ka(decimal) = 0 to (32767/32768) = 0 to 0.999969482421875
Since a pot input ranges from 0 to 0.99… we will just use the POT0 input directly.

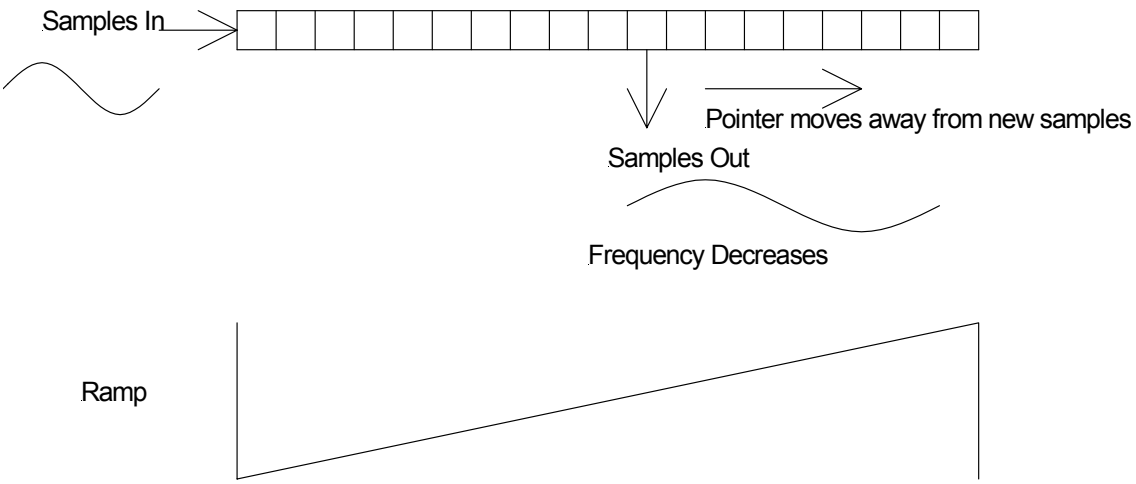
Kf(decimal) = (125/511) to (500/511) = 0.2446… to 0.9784…
As we will want the value from POT1 to fall in this range, we can read the POT1 value, multiply it by 0.7338 (0.9784… - 0.2446…) and add 0.2446 to it.

```
; From Application note AN-0001
; Program AN0001-2.spn
;
; Initialization, only run on first execution of code
; Skip to the START label if NOT the first time
skp run,START
; Initialize sin LFO 0 for 5Hz and +/- 0
wlds SIN0,125,0
; End of skip/initiaization
; Main program code
; First, read in POT0 and write it to LFO0_RANGE
; POT0 -> ACC
START: ldax POT0
; Write it to LFO0 amplitude register and clear ACC
wrax SIN0_RANGE,0
; Read in POT1 and multiply it by 0.7338, POT1 * 0.7338 -> ACC
rdax POT1,0.7338
; Add 0.2446 to the value in the accumulator
sof 1.0,0.2446
; Write it to the LFO frequency register and clear ACC
wrax SIN0_RATE,0
;
; We now use the cho rdal instruction which will read an LFO into the accumulator
cho rdal,SIN0
;
; Wave is now in ACC, write it to DACL
wrax DACL,0
; That's it!
; NOTE: When you view the output on a scope you will see amplitude variation
; due to the high pass filtering in the DAC.
```

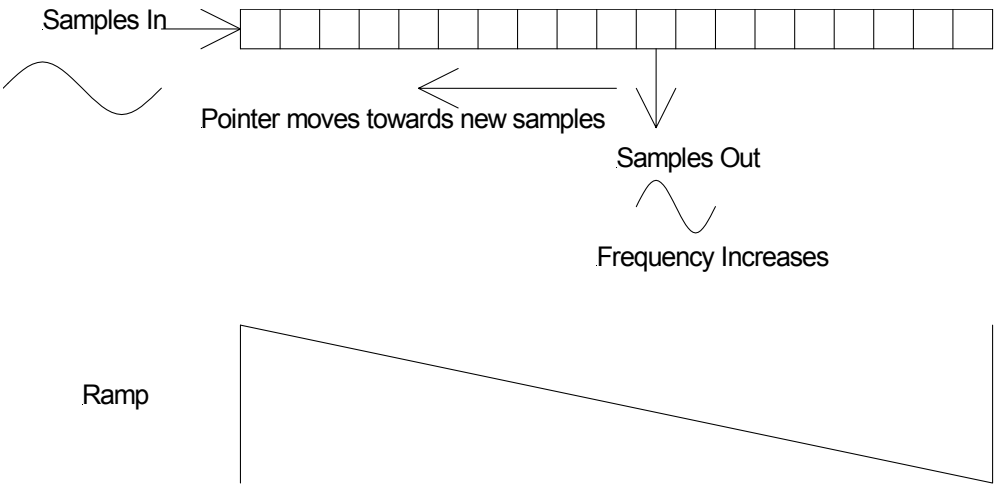
The ramp LFOs are primarily used to pitch shift a signal up or down. As we learned in the above example, if a read pointer is moved towards or away from the incoming samples in a delay, the pitch will shift up or down accordingly. The problem is that the pointer eventually would run off one end of the delay so we would need to make sure it is brought back to the other end. The ramp LFO can do this for us, unlike the SIN LFO the ramp LFO is always positive, going from 0 to a max value (selectable as 4095, 2047, 1023 or 511) so by adding this offset to the base address of a memory delay, the read pointer can move through memory. The ramp can be an increasing ramp so that the read pointer is moving away from the

in coming samples and as a result creates a pitch down or a decreasing ramp so that the pointer is moving towards the incoming samples and creating a pitch up.

Increasing ramp:



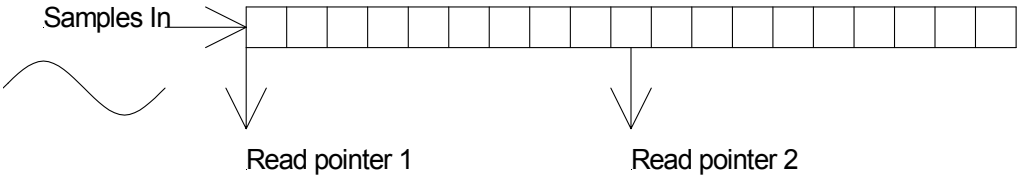
Decreasing ramp:

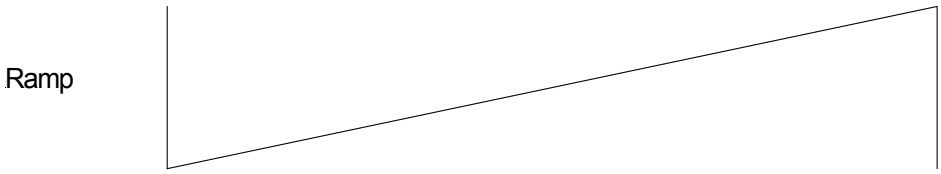


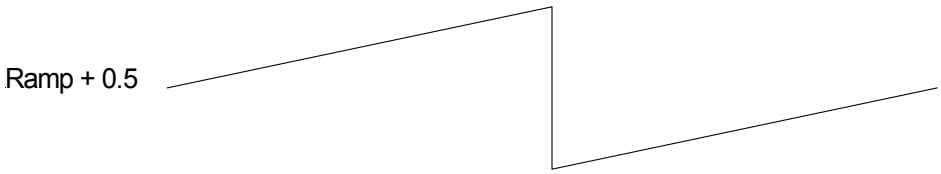
While these ramp waves can be used to pitch shift the signal, we will hear a discontinuity or "click" when the pointer wraps from one end of the delay to the other. To avoid this, we would want to use two pointers spaced apart by half the delay length and read the samples out from the pointer closest to the middle of the delay (farthest from the ends) but this would

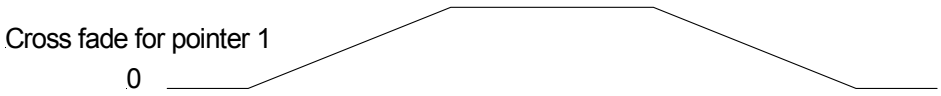
still have a similar problem as we jump between the pointers so we would additionally like to cross-fade between the pointers so that we smoothly transition from one pointer to the other.

The ramp LFO can provide both the second pointer and the cross fade coefficients to transition between the pointers. If we imagine that the ramp LFO goes from 0 to 1.0 and wraps back to 0 again like: 0, 0.1, 0.2, …, 0.8, 0.9, 1.0, 0, 0.1, 0.2,… then we can create the second pointer by simply adding 0.5 to the current ramp like: The cross fade coefficient can be derived from one pointers position in the delay, at the ends it is 0 and in the middle it is 1.0. This can all be shown in the following illustration:









As we can see, when read pointer 1 is at an end of the delay (the ramp wave is wrapping from 1.0 to 0) read pointer 2 is at the middle of the delay by using ramp+0.5. Additionally, the cross fade coefficient is 0 when read pointer one is at the end of the delay and 1.0 when it is in the middle of the delay. The coefficient to use with the second pointer is simply (1.0 – cross fade for pointer 1)

Now, there is one final issue to deal with and that is the same issue that we had with the SIN LFO and that is that we will normally not be landing on whole sample points as we move along the delay so we will want to interpolate at each read pointer like we did in the SIN LFO. Similarly to the SIN LFO there are fractional bits below those used for address offset that can be used for linear interpolation.

So to summarize the steps required for a pitch shift:

1. Define a delay in memory that is 4096, 2048, 1024 or 512 samples long

2. Set ramp amplitude (4096, 2048, 1024 or 512, must match length in 1 above) and pitch shift coefficient.

3. Read from first pointer and linearly interpolate a result

4. Read from second pointer by adding 0.5 to the ramp and linearly interpolate a result

5. Use the cross fade coefficient to cross fade between the results from the ramps for the final result.

While this is more complex that the chorus as we need to perform two linear interpolations and the cross fade, the LFOs will again do most of the work.

The equation to calculate the coefficient to send to the LFO for a given amount of pitch shift is:

For pitch shifting down:

$$C = -2^{14} * \left(1 - \frac{1}{2^N}\right)$$

N = Desired amount of pitch shift in octaves (N is positive)

For pitch shifting up:

$$C = 2^{14} * \left(2^N - 1\right)$$

N = Desired amount of pitch shift in octaves

NOTE: Coefficient values must be in the range of -16384 <= C <= 32767

```
; From Application note AN-0001
; Program AN0001-3.spn
;
delayl mem 4096 ; Left delay
delayr mem 4096 ; Right delay
temp mem 1 ; Temp location for partial calculations
; Initialization, only run on first execution of code
; Skip to the START label if NOT the first time
skp run,START
; Load up a ramp LFO. Since we need to be able to create both increasing
; and decreasing ramps so we can pitch up or down, we use the sign of the
; coefficient to tell us what to do. A positive coefficient means
; pitch up while a negative coefficient means pitch down.
;
; The amplitude of the ramp is fixed at 4096, 2048, 1024 or 512
; We will use the long delay range and a coefficient of 16384 for a
; shift up of one octave.
wldr 0,16384,4096
; End of skip/initiaization
; Main program code
; First, read in the sample and write to the start of the delay
```

```
; Read in the current left sample, ADCL -> ACC
START: ldax ADCL
; Write it to left delay and clear ACC
wra delayl,0
; Read in the current sample, ADCR -> ACC
ldax ADCR
; Write it to right delay and clear ACC
wra delayr,0
; We use the start of the sample memory block as the
; address since we are using a positive only ramp
; that ranges 0 to 1.0 (4095 in this case)
; We are linearly interpolating at the first read pointer
; (1-k)*sample[addr]
; We register the output of the ramp LFO into an internal LFO
; register so it won't change as we use it.
cho rda,RMP0,REG|COMPC,delayl
; k*sample[addr+1] + ACC
cho rda,RMP0,,delayl+1
; We now have the result for the first read pointer, we save it to
; memory for now. So here we are doing ACC -> memory, ACC*0 -> ACC
; to save the result and clear the accumulator
wra temp,0
; Do the second read pointer. We tell the LFO to add ½ to the ramp
; and basically do the same as above.
; (1-k)*sample[addr+ half ramp]
cho rda,RMP0,RPTR2|COMPC,delayl
; k*sample[addr+ half ramp + 1] + ACC
cho rda,RMP0,RPTR2,delayl+1
; Now in the ACC we have the result of the linear interpolation
; around the second read pointer. We now want to cross fade between
; this and the result of the first read pointer. Since this is from
; the second read pointer we want to use 1-XFADE coefficient. Note
; that XFADE' means 1-XFADE to the assembler.
; ACC*(1-XFADE) + 0
cho sof,RMP0,NA|COMPC,0
; Result in ACC is now read [pointer 2] * (1-XFADE)
; Add in earlier value that was saved in memory, multiply saved value
; by XFADE coefficient
; memory[addr] * XFADE + ACC
cho rda,RMP0,NA,temp
; Final result is in ACC, write it to DACL and clear ACC
wrax DACL,0
; Now repeat it for the right channel. Since we are using the same ramp LFO as above,
; we do not need to REG it.
cho rda,RMP0,COMPC,delayr
cho rda,RMP0,,delayr+1
wra temp,0
cho rda,RMP0,RPTR2|COMPC,delayr
cho rda,RMP0,RPTR2,delayr+1
cho sof,RMP0,NA|COMPC,0
cho rda,RMP0,NA,temp
```

```
wrax DACR,0
```

Like the SIN LFOs, we can directly write to the rate and range registers for ramp LFOs. In general we will only write to the rate register since that is all we need to change to control the pitch. Like the SIN registers we need to determine how the coefficient is mapped from the accumulator into the ramp rate register. In this case, the coefficient can be positive or negative, so:

ACC[23:8] are mapped to the 16-bits that control frequency

Therefore to convert between the calculated C value and its decimal equivalent:

C(decimal)=C/32768

Since C may range from -16384 to +32767, C(decimal) will range from -0.5 to + 0.999969482421875

In the following example, we will use POT0 to adjust the frequency of the ramp so that the input signal can be pitched up or down by an octave. The signal will be at normal pitch when the pot is in the middle position.

C(-1 octave) = -8192, C(-1 octave decimal) = -0.25
C(+1 octave) = 16384, C(+1 octave decimal) = +0.5

Since the range of values are not symmetrical about 0 for +1 octave and -1 octave, we will need to use slightly different calculations if POT0 is greater than or less than 0.

```
; From Application note AN-0001
; Program AN0001-4.spn
;
; Memory declarations
delayl mem 4096 ; Left delay
delayr mem 4096 ; Right delay
temp mem 1 ; Temp location for partial calculations
; Initialization, only run on first execution of code
skp run,START
; Load up a ramp LFO.
; We will use the 4096 delay range and a rate coefficient of 0
wldr RMP0,0,4096
; End of skip/initiaization
; Main program code
; Read in the current left sample, ADCL -> ACC
START: ldax ADCL
; Write it to left delay and clear ACC
wra delayl,0
; Read in the current sample, ADCR -> ACC
ldax ADCR
; Write it to right delay and clear ACC
wra delayr,0
; (1-k)*sample[addr]
cho rda,RMP0,REG|COMPC,delayl
; k*sample[addr+1] + ACC
cho rda,RMP0,,delayl+1
```

```
; ACC -> memory, ACC*0 -> ACC
wra temp,0
; (1-k)*sample[addr+ half ramp]
cho rda,RMP0,RPTR2|COMPC,delayl
; k*sample[addr+ half ramp + 1] + ACC
cho rda,RMP0,RPTR2,delayl+1
; ACC*(1-XFADE) + 0
cho sof,RMP0,NA|COMPC,0
; memory[addr] * XFADE + ACC
cho rda,RMP0,NA,temp
; Final result is in ACC, write it to DACL and clear ACC
wrax DACL,0
; Now repeat it for the right channel
; Since we will use the same ramp for right, we do not need to "REG" it
cho rda,RMP0,COMPC,delayr
cho rda,RMP0,,delayr+1
wra temp,0
cho rda,RMP0,RPTR2|COMPC,delayr
cho rda,RMP0,RPTR2,delayr+1
cho sof,RMP0,NA|COMPC,0
cho rda,RMP0,NA,temp
wrax DACR,0
;
; Rate adjustment
;
; Read POT0 into ACC
ldax POT0
; Subtract 0.5 so ACC ranges from -0.5 to +0.5
sof 1.0,-0.5
; If ACC >= 0, skip the ACC*0.5 instruction
skp GEZ,pos
; ACC < 0, scale it to be -0.25 to 0
sof 0.5,0
; Load ACC into ramp rate register and clear ACC
pos: wrax RMP0_RATE,0
; Done
```

Notice

Spin Semiconductor reserves the right to make changes to, or to discontinue availability of, any product or service without notice.

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

Spin Semiconductor make no warranty, expressed or implied, of the fitness of any product or service for any particular application.

Contact Information

Spin Semiconductor
Phone: (310) 417-4956
Web: www.spinsemi.com

Mailing:
Spin Semiconductor
c/o OCT Distribution
6504 1/2 Arizona Ave.
Los Angeles, CA 90045