Chapter 16
FUNS

The name "FUNs," although it sounds like a pun, is simply the best abbreviation for "Function." You'll definitely have a good time with them, however, if you're looking for a variety of ways to control your sounds.

We've discussed various control sources throughout this manual, from the physical controls like the Mod Wheel to the software control sources like LFOs and attack velocity. You can assign them to affect your sounds in all sorts of ways.

The FUNs take the control sources one level further. By setting up a FUN as a control source, you can mix the signals of two control sources, and perform one of 50 functions on the combined signals. The result of that function becomes the new control source value. Because the FUNs can radically change the combined input values, the FUNs can have a profound effect on your sounds.

You may find that experimenting with the various FUN equations gives you a better idea of their effects than reading the explanations. Although there's a great deal of mathematics behind the FUNs, the most important consideration is how they affect your sounds. The more you play around with them, the better you'll understand how powerful they are.

The Mechanics of Control Sources

We'll return for a minute to the notion that the K2500 is an integrated system consisting of a MIDI-driven sound engine and a MIDI-driven effects processor. The sound engine responds to MIDI messages received at the MIDI In port and from the front panel, as does the effects processor.

The K2500's control sources use their own internal signal format for interpreting control messages and communicating them to the sound engine. Every control source sent from your MIDI controller to the K2500's sound engine is translated to a value in the range from -1 to +1. This consistency enables the sound engine to process control source signals very efficiently. Conversely, the K2500's internal control source signals are translated to MIDI values before being sent to the MIDI Out port.

A control signal value of 0 represents minimum effect; it's equivalent to the control source being turned off or disconnected. A control signal value of +1 represents the maximum positive effect of a control source, while a value of -1 represents the maximum negative effect of a control source.

Unipolar and Bipolar Control Sources

There are two kinds of control source signals: unipolar and bipolar. A unipolar signal has a value between 0 and +1. A bipolar signal has a value between -1 and +1.

A switch pedal is unipolar; its control signal value will never go below 0. Since it's a switch control, it has only two possible values: 0, which corresponds to off or minimum, and +1, which corresponds to on or maximum. When you depress your MIDI controller's sustain pedal, for example, it sends a control signal value of +1 to the K2500's sound engine.

Continuous controls can be unipolar or bipolar. Consider your MIDI controller's Mod and Pitch Wheels as examples. Normally, the Mod Wheel affects the K2500 as a unipolar control

source; it sends a control signal value that's interpreted as 0 when it's fully down, and values interpreted between 0 and +1 as you push it up. When fully up, it sends a value that's interpreted as +1. It can be used as a bipolar control source by assigning a value of Bi-Mwl to any control source parameter.

The Pitch Wheel is normally bipolar; it sends a control signal value that's interpreted as 0 when it's centered, values interpreted between 0 and -1 as it's pulled downward, and values interpreted between 0 and +1 as it's pushed upward. It can be used as a unipolar control source by assigning a value of AbsPwl to any control source parameter.

The FUNs can act as unipolar or bipolar control sources; it depends on the values of the input signals and the nature of the function you choose. Depending on the function you choose to process the input signals, the output signal value can exceed +1 or -1. Normally the signal merely pins at +1 or -1; that is, it won't go any higher or lower. In some cases, however, the output signal value is wrapped around instead of pinning; we'll mention these cases as we get to them. You can assume that the output signal values of the functions listed below will pin at -1 or +1, unless specified otherwise.

Programming the FUNs

Start by entering the Program Editor, then use the soft buttons to select the FUN page. Setting up a FUN as a control source is a two-step process: assigning a FUN as the value for one or more control source parameters in the Program Editor, then programming the FUN on the FUN page, by assigning control sources to two inputs—**a** and **b**, and choosing a function (equation) that will process the combined signals from **input a** and **input b**.

```
EditProg:FUN                      <>Layer:1/1

              Input a:  Input b:  Function:
FUN1:         OFF       OFF       a+b
FUN2:         OFF       OFF       a-b
FUN3:         OFF       OFF       (a+b)/2
FUN4:         OFF       OFF       a/2+b
       <more    LFO      ASR      FUN      VTRIG    more>
```

There are four FUNS; you can combine and process four different pairs of control source signals. FUNs 1 and 3 are always local, that is, they affect each note in their respective layers independently. FUNs 2 and 4 are local by default, but they can be made global by setting a value of On for the Globals parameter on the COMMON page in the Program Editor. A global FUN affects all notes in its layer equally and simultaneously.

The best way to understand the use of the FUNs is to set up a simple test model, then plug in the different equations and listen to their effects. We'll walk you through the programming of a FUN and assigning it to control pitch. Then you can scroll through the list of equations at your leisure.

Start in Program mode and select program 199. Press EDIT to enter the Program Editor. Select the KEYMAP page, and change the keymap to 152 Dull Sawtooth. Then select the PITCH page, and assign a value of FUN1 for the Src1 parameter (a shortcut is to press 1, 1, 2, ENTER on the alphanumeric pad). Select the Depth parameter and change the value to 1200 cents. Next, select the FUN page, and select the **Input a** parameter for FUN1. Assign a value of MWheel (the quickest way is to hold the K2500's ENTER button and move your MIDI controller's Mod Wheel). Next, select the **Input b** parameter for FUN1, and assign a value of Data. This assumes your MIDI controller either has a data slider, or a programmable control that you set to send Data messages (MIDI 06). If you don't have a data slider or a programmable control, you can set the value of **Input b** to AbsPwl, and use your Pitch Wheel to control **Input b**. If you do this,

you'll need to go to the LAYER page and set the PBMode parameter to a value of Off to keep Pitch Wheel messages from interfering with the test model.

Now select the Function parameter, and scroll through the list of equations. Move your MIDI controller's Mod Wheel and Data slider as you play, and listen to their effects. Actually listening to the various effects while reading the explanations below will help your understanding. In the model we've set up here, **inputs a** and **b** are both unipolar. The effect of each equation will differ depending on the type of controls you assign to the inputs. There are four possible combinations: both inputs unipolar; both inputs bipolar; **input a** unipolar with **input b** bipolar; **input a** bipolar with **input b** unipolar.

The FUN Equations

In this section we'll describe how each of the FUN equations works. In some cases, a small graph will accompany the explanation. Here's how to interpret the graphs.

Each graph shows a curve illustrating the effect of the equations on the input signals. The horizontal axis represents the possible values of the input to the FUN (the combined control signals of **inputs a** and **b**). The vertical axis represents the possible values of the FUN's output signal. The four elements in the diagram below show you how to read these graphs:

a   the curve representing the effect of the FUN's equation on every possible input value

b   one point on that curve, representing a single input value and the corresponding output value generated by the FUN's equation

c   the input value represented by the point

d   the output value represented by the point
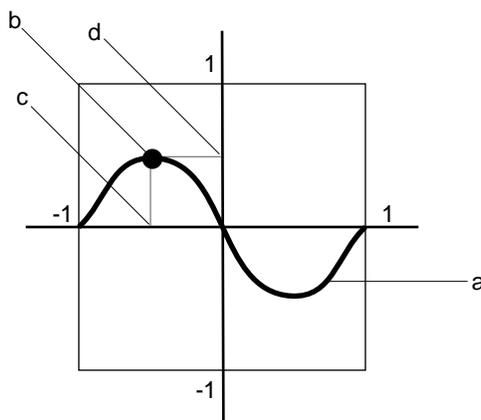
For any point on the equation's curve, you can determine the input value by tracing a line from the point to the horizontal axis. Similarly, you can determine the output value by tracing a line from the point to the vertical axis. For the point shown in the example above, the combined values of the control signals of **inputs a** and **b** equal about -.5, which translates to an output value of +.5. An input value of -1 gives an output value of 0, as do input values of 0 and +1. An input value of +.5 gives an output value of -.5.

The List of Equations

The first six equations are weighted sums and differences—that is, the signal values of **inputs a** and **b** are added to or subtracted from each other, and are divided in turn by various amounts to alter their effects relative to each other. These equations give you several different types of mixers for combining the signals of the two inputs.

a + b

The values of **inputs a** and **b** are added, creating a simple mixer. For example, you might have LFO1 assigned for the Src2 parameter on a layer's PITCH page, and a FUN assigned for the DptCtl parameter. On the FUN page, if you set **input a** to a value of MWheel, and **input b** to a value of MPress, then this equation will let you modulate the depth of the LFO's pitch modulation with your MIDI controller's Mod Wheel or with mono pressure. You could set a fixed initial depth with the Mod Wheel and alter it further with mono pressure. In this case the output signal would pin at +1 or -1 fairly quickly.

a - b

This operates similarly to the previous equation, but the value of **input b** is subtracted from the value of **input a**. This equation will reverse the normal effect of the control source assigned to **input b**. For example, if **input a** is off, and **input b** is assigned to a unipolar control source like MWheel, then the Mod Wheel will generate a control signal of -1 when fully down, and 0 when fully up.

(a + b) / 2

The values of **inputs a** and **b** are added, and the sum is divided by 2. This gives you the same kind of control as the previous two equations, but the output signal will reach +1 or -1 half as often as with the equation a + b.

a / 2 + b

The value of **input a** is divided by 2, and the result is added to the value of **input b**. **Input a** has half the effect of **input b**.

a / 4 + b / 2

The value of **input a** is divided by 4, and the value of **input b** is divided by 2. The two results are added to give the output value. **Input a** has half the effect of **input b**, and the total result has half the effect of the previous equation.

(a + 2b) / 3

The value of **input b** is multiplied by 2, and the result is added to the value of **input a**. This sum is then divided by 3. **Input a** has half the effect of **input b**, and the total result has somewhat more effect than the previous equation, but less effect than a + b.

a * b

The values of **inputs a** and **b** are multiplied. If you like using Src2 and DptCtl, this equation can be used to create a similar type of control source (it's equivalent to the Src2/DptCtl pair with the MinDpt parameter set to 0).
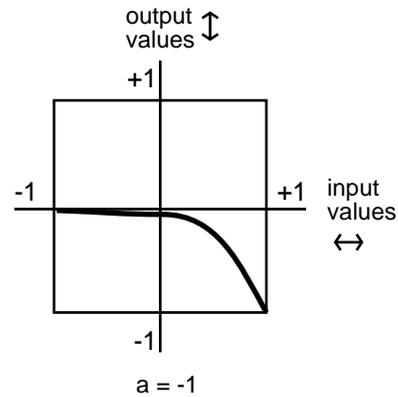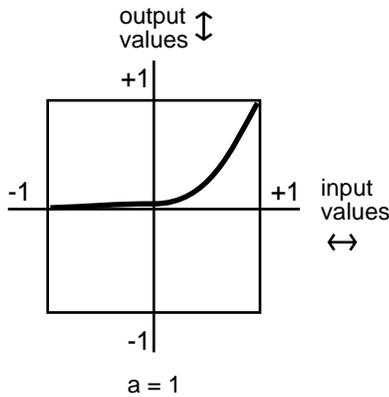
-a * b

The value of **input a** is multiplied by -1, then multiplied by the value of **input b**. This will reverse the normal effect of the control source assigned to **input a**. This equation also produces an effect like that of Src2 and DptCtl with the MinDpt parameter set to 0.

a * 10^b

The actual equation is: $\mathbf{a} \times \left(10^{(2 \times b)} \div 100 \right)$. This is an exponential curve. 10 is raised to the (2 x **b**) power, then divided by 100. This result is then multiplied by **a**. Another way to express this is as follows: a change of 1 in the value of input **b** results in a hundredfold change in the output value. Here are a few possible output values:

Input values		Output values
a = +1,	**b** = +1	+1
a = +1,	**b** = 0	.01
a = +1,	**b** = -1	.0001
a = 0,	**b** = +1	0
a = 0,	**b** = 0	0
a = 0,	**b** = -+1	0
a = -1,	**b** = +1	-1
a = -1,	**b** = 0	-.01
a = -1,	**b** = -1	-.0001

output values ↕
+1
-1   +1 input values ↔
-1
a = 1

output values ↕
+1
-1   +1 input values ↔
-1
a = -1

| a + b |

The values of **inputs a** and **b** are added, and the absolute value of the sum is taken. If the sum is negative, it is multiplied by -1. This makes the FUN a unipolar control source. (See the illustration on the following page.)

| a - b |
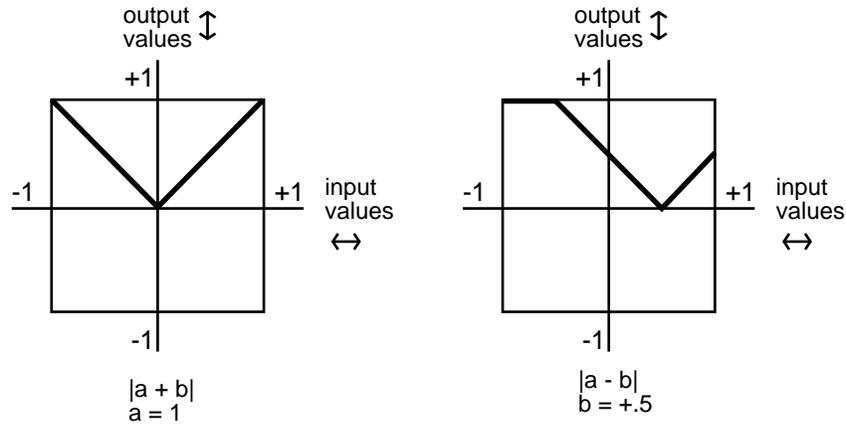
The value of **input b** is subtracted from the value of **input a**, and the absolute value is taken. If the difference is negative, it is multiplied by -1. This also makes the FUN unipolar.

min (a, b)

The values of **inputs a** and **b** are compared, and the smaller value becomes the output value. This can be used to limit the value range of a control source. If, for example the value of the control source assigned to **input b** is left at +.5, then when the value of the control source assigned to **input a** is between -1 and +.5, its value will be used. As soon as its value exceeds +.5, the value of **input b** is used.

max (a, b)

This is the opposite of the previous equation. The values of **inputs a** and **b** are compared, and the larger value becomes the output value.

Quantize b to a

This turns the control source assigned to **input b** into a stepped control source. Instead of smooth transitions from minimum to maximum, it will jump from minimum to maximum in some number of equal steps. The number of steps is determined by the value of **input a**. The normal realtime application of this is to set a stationary value for input a to set the number of steps in the effect. Then use the control source assigned to **input b** as a realtime control to induce the stepped effect. Changing the value of **input a** in realtime will produce an extraneous (but possibly useful) effect.

Range of values for **input a**		Total number of steps as **input b** moves from min to max	
from	to	(when **input b** is bipolar)	(when **input** b is unipolar)
0	.0625	1 (no effect)	1
.0625	.125	2	1*
.125	.1875	3	2
.1875	.25	4	2*
.25	.3125	5	3
.3125	.375	6	3*
.375	.4375	7	4
.4375	.5	8	4*
.5	.5625	9	5

.5625	.625	10	5*
.625	.6875	11	6
.6875	.75	12	6*
.75	.8125	13	7
.8125	.875	14	7*
.875	.9375	15	8
.9375	1	16	8*

As an example, consider the FUN we set up at the beginning of the previous section: the Mod Wheel was assigned as **input a**, and the data slider as **input b**. The FUN was assigned as Src1 on the PITCH page, and the depth of Src1 was set to 1200 cents. If you push the Mod Wheel all the way up, the value of **input a** will be +1. This will set the number of steps at 8, since the data slider sends a unipolar control signal. With your MIDI controller's data slider at minimum, play and sustain a note. Then move the data slider slowly up. The pitch of the note will jump up an octave in 8 steps as you move the data slider all the way up.

If the value of **input a** is negative, it's multiplied by -1, so its value always falls within the ranges above. When **input b** is bipolar and the resulting number of steps is an odd number, the steps are centered around a value of 0—that is, the center step is equivalent to no effect from **input b**. When the number of steps is even, a value of zero is not included in the steps. This is also true for the values marked by an asterisk when input b is unipolar.

lowpass (f = a, b)

This equation might be called a lag equation. Its effect is to introduce a delay in the K2500's response to changes in the value of **input b**. It works by filtering (reducing) higher values of **input b**. The value of **input a** determines the degree to which the values of **input b** are filtered. Low values for **input a** will induce a long lag when the value of **input b** changes. High values will shorten the lag. When **input b** remains constant at a high level, low values of **input a** will cause the FUN to sweep up slowly from 0 to the value of **input b**. Higher values for **input a** will cause the FUN to sweep more rapidly.

The four graphs below show the effect of different values for **input a** on the change of **input b**. In each graph, the value of **input b** jumps from 0 to +1. In graph 1, the value of **input a** is +1. Each successive graph represents the same change in the value of **input b**, at successively lower values for **input a**.

This equation works as intended only when the value of **input a** is 0 or positive. Negative values for **input a** will result in a much less predictable response than positive values. You might like the effect, but it won't be anything like what we've just described.
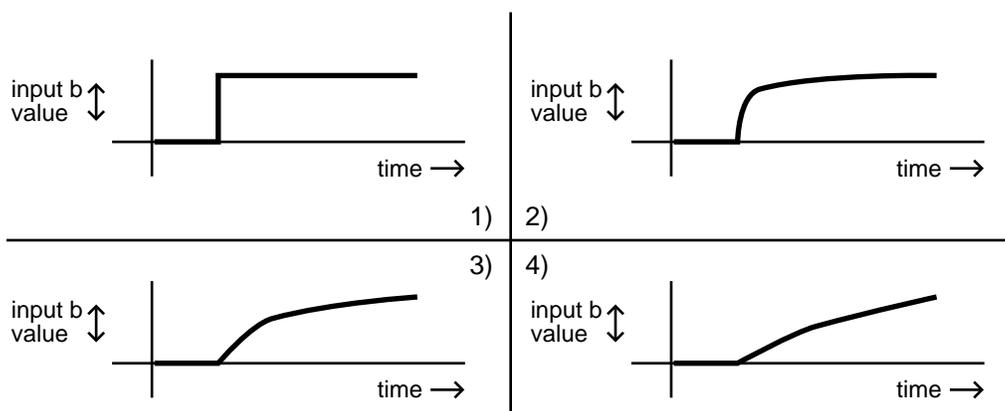
hipass (f = a, b)

With this equation the low values of **input b** are filtered according to the value of **input a**. This causes somewhat different results compared with the lowpass equation above. At low values for **input a**, low values for **input b** will have little effect, while high values for **input b** will cause the FUN to quickly reach full effect then slowly sweep down to its starting level. At high values for **input a**, a rapid change in the value of **input b** will have little effect. At low values for **input a**, rapid changes in the value of **input b** will cause the FUN to respond quickly to the change, then slowly fade back to minimum effect. Listening to the effects at different values for each input will give you the best understanding.

The four graphs below show the effect of different values for **input a** on the change of **input b**. In each graph, the value of **input b** drops from +1 to 0. In graph 1, the value of **input a** is +1. Each successive graph represents the same change in the value of **input b**, at successively lower values for **input a**.
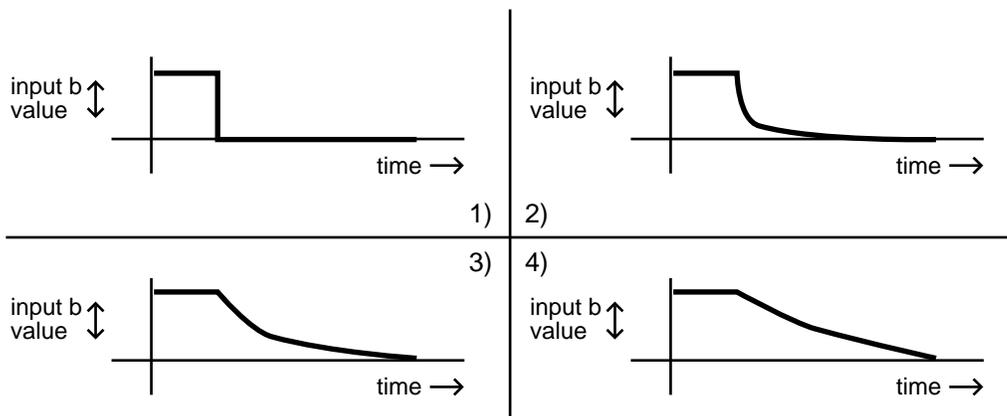

b / (1 - a)

This is another weighted difference equation similar to the first six. The value of **input a** is subtracted from 1. The value of **input b** is then divided by the difference. You'll get considerably different results for different input values of **a** and **b**.

a(b-y)

Think of this equation as reading "y is replaced by the result of the function a(b-y)." The value of y indicates the value of the FUN's output signal. Every 20 milliseconds, the K2500 takes the current value of y, runs the equation, calculates a new value of y, and inserts the new value into the equation. Consequently the value of y will change every twenty milliseconds. Here's an example. When you play a note, the K2500 starts running the FUN. The first value for y is always 0. We'll assume the value of **input a** is +.5, and the value of **input b** is +1. The first time the K2500 evaluates the FUN, the result of the equation is .5 x (+1 - 0), or .5. So the FUN's output value after the first evaluation is .5. This becomes the new value for y, and when the K2500 does its next evaluation of the FUN, the equation becomes .5 x (+1 - .5), or .25. The resulting output value is .25, which becomes the new value for y. For the next evaluation, the equation is .5 x (+1- .25), or .375.
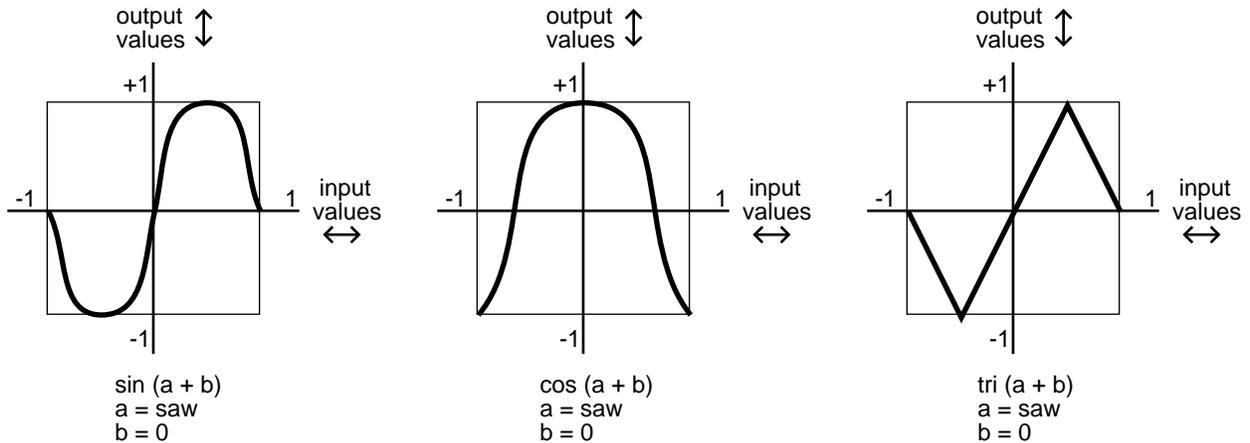
(a + b)^2

The values of **inputs a** and **b** are added, and the result is squared (multiplied by itself). This will change the linear curve of a unipolar control signal into a curve that's lower at its midpoint (by a factor of 2). Bipolar control signals will generate curves that are high at both ends, and 0 in the middle.
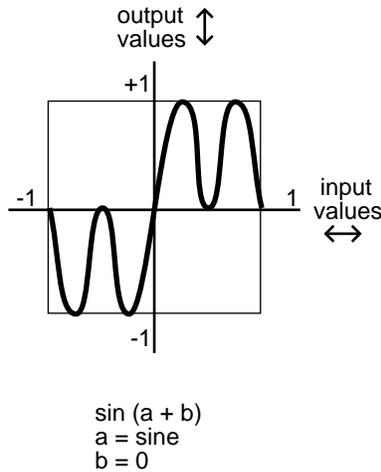
b = 0

a = b

a is unipolar
b = .5
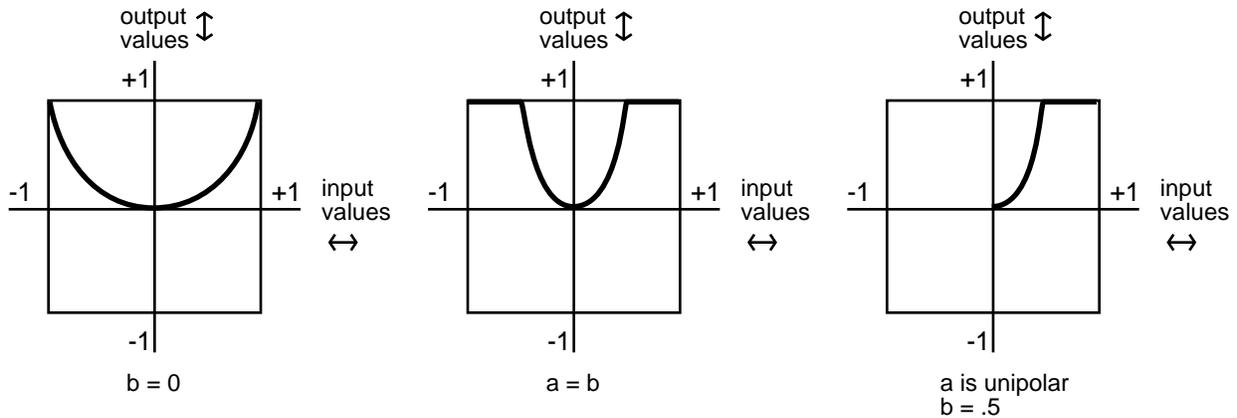
sin (a + b), cos (a + b), tri (a + b)

These equations are intended to be used with inputs that are sawtooth waves—for example, **input a** might be LFO1 with its shape set as a sawtooth. Each equation will map a sawtooth-shaped input into a sine-, cosine-, or triangle-shaped output. Other input waveform shapes will result in outputs with more complex waveform shapes.

Other ways to get sawtooth shapes as inputs to these FUNs are to use other FUNs as the inputs, with their equations set as any of the ramp equations described later in this section (see the note at the end of this chapter about the evaluation order of the FUNs). You could also use LFOph1 or LFOph2 as inputs. The first three graphs below show the result of these functions when **input a** is a rising sawtooth wave, and the value of **input b** is 0. The fourth shows the result of the sin (f=a + b) equation when the value of **input b** is 0 and **input a** is a sine wave.

sin (a + b)
a = saw
b = 0

cos (a + b)
a = saw
b = 0

tri (a + b)
a = saw
b = 0

sin (a + b)
a = sine
b = 0

Warp Equations

The next five equations all behave similarly, and are intended to be used as follows: the value of **input a** is the controlling value, and normally remains constant, although it doesn't have to. The value of **input b** is expected to change over time; **input b** might be an LFO, for example. The value for **input a** affects how the FUN calculates its output value while the value of **input b** changes.

warp1(a, b)

We call this the Vari-slope™ equation. The value of input a controls the mapping of values for **input b**. If **input b** is a sawtooth wave, different values for **input a** will change it into a triangle wave. If **input b** is a more complicated waveform, the output waveform is also more complicated.

a = 0, b rising
from -1 to 1

a = .5, b rising
from -1 to 1

a = 1, b rising
from -1 to 1
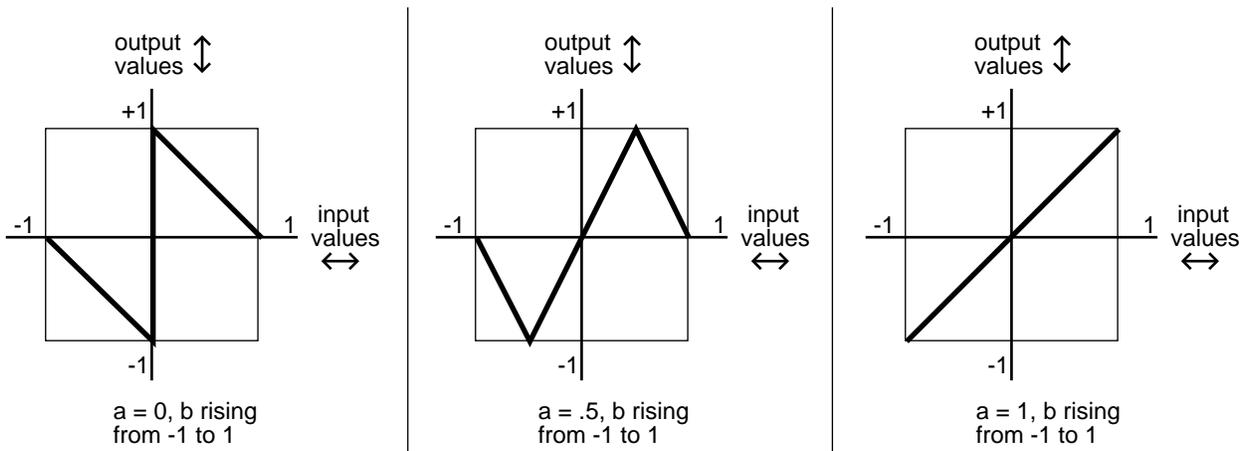
warp2(a, b)

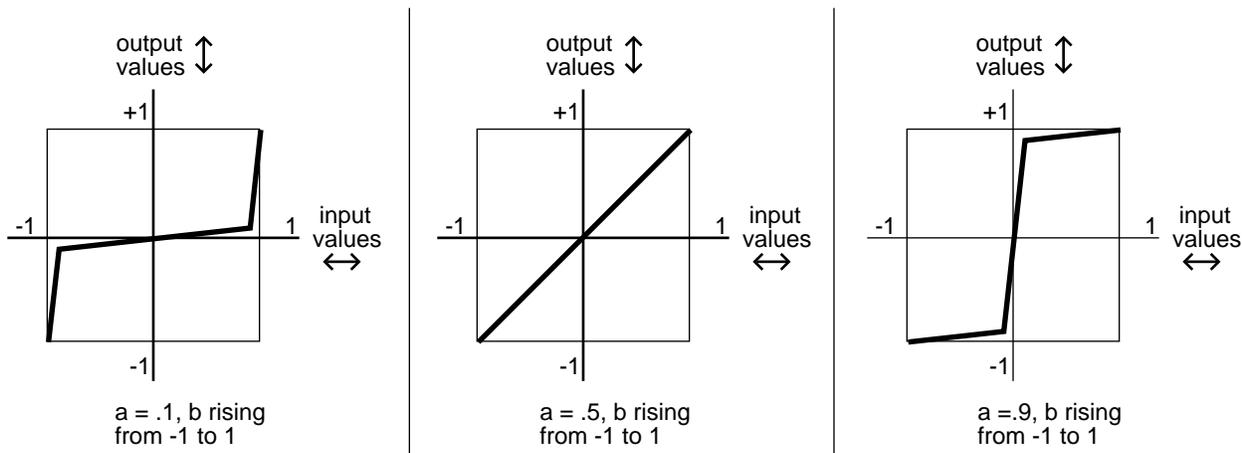We call this equation Slant-square.™ Again, the value of **input a** controls the mapping of values for **input b**. If **input b** is a sawtooth wave, different values for **input a** will turn it into a number of variations on square waves.

a = .1, b rising from -1 to 1

a = .5, b rising from -1 to 1

a =.9, b rising from -1 to 1

warp3(a, b)

We call this one the Variable Inverter.™ It looks at the binary numbers that represent the values of **inputs a** and **b**, compares the corresponding bits in each number, and performs an XOR operation on them (we'll explain that below). The resulting number is converted into the output value. This can produce some erratic results, but if variety is what you're after, this equation will give it to you. You'll get your best results when **input b** is an LFO with a slow rate.

The XOR operation is a subprogram that applies a truth table to each of the digits in the binary numbers that represent the values of **inputs a** and **b**. Each of these numbers is a string of 16 digits (bits); each bit is either a 0 or a 1. The subprogram looks at the first bit of each number. If they're both 0s, the resulting value is 1. If one is a 0 and the other is a 1, the result is 1. If they're both 1s, the result is 0. This process is repeated for the remaining 15 bits of each number, and a new 16-bit number is generated. This number represents the output value of the FUN.
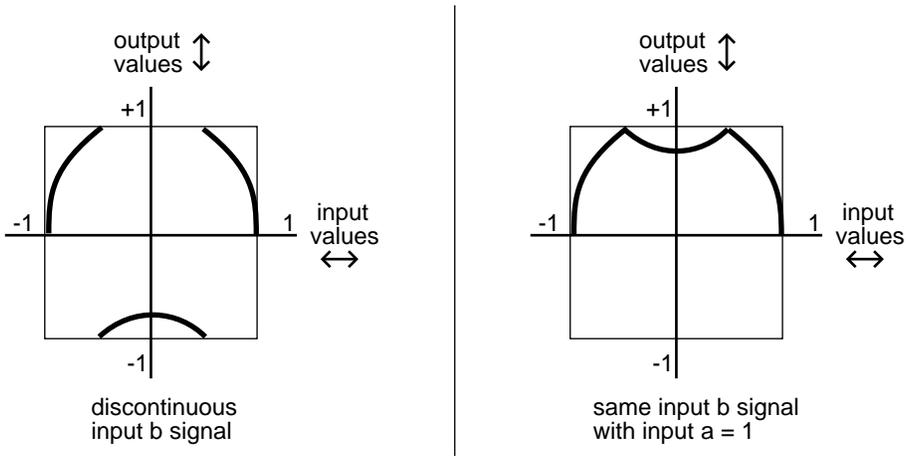
a = -1, b rising from -1 to 1

a = .5, b rising from -1 to 1

a =1, b rising from -1 to 1
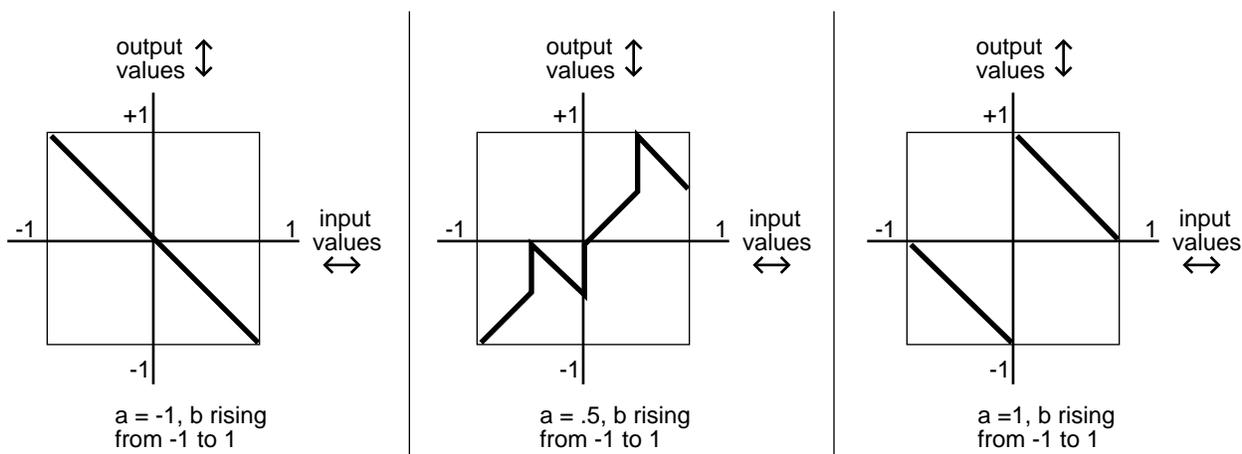
warp4(a, b)

This equation, the Period Inverter,™ is based on repeated evaluations of the value of **input b**. The K2500 compares each new value of **input b** with the value from the previous evaluation. If the absolute value (always a positive number) of the difference between the two is greater than the value of **input a**, the current value of **input b** is multiplied by -1.

The primary feature of this equation is that it will take a discontinuous signal and make it continuous. If, for example, FUN1 uses an equation like a(y + b), its output can wrap around from +1 to -1, or vice versa. You could set FUN1 as **input b** for FUN2, set **input a** of FUN2 to ON (+1), and FUN2 would remove the discontinuity from the signal. The first graph below shows a hypothetical output signal with such a discontinuity, and the second shows how FUN2 in this case would make the signal continuous without drastically changing its shape.

If, on the other hand you *want* the signal to become discontinuous, you can use the warp4(a, b) equation in a single FUN, with **input a** set to OFF (0), and the signal would be multiplied by -1 with each evaluation of **input b**.

output
values

+1

-1   1   input
                           values

-1

discontinuous
input b signal

output
values

+1

-1   1   input
                           values

-1

same input b signal
with input a = 1

warp8(a, b)

This relatively simple equation is **a** x **b** x 8. If the result is beyond the range of -1 to +1, it wraps around from +1 to -1 (or vice versa), until it's within the allowable range. The table below shows some examples of how this works.

a x **b** x 8 =	final output value
-7.4	.4
-4.2	-.2
-1.8	.8
-.6	-.6
.4	.4
1.2	-.2
2.6	.6
5.4	-.4

a AND b

The values of **inputs a** and **b** are interpreted as logical quantities—they're considered TRUE if they're greater than +.5, and FALSE otherwise. This turns the FUN into an on/off switch. In the

model we set up in the previous section, FUN1 was set to control Src1 on the PITCH page, and Src1's depth was set to 1200 cents. With this equation, both **input a** (the Mod Wheel in this case) and **input b** (the data slider in this case) would have to be more than halfway up for the FUN to switch on. The pitch would jump 1200 cents as soon as both control sources moved above their halfway points. As soon as one of them moved below its halfway point, the pitch would jump back to its original level.

This equation can be used to trigger ASRs, or as a layer enable control, or for any control source that toggles on and off. If you set one of the inputs to an LFO, the FUN would switch on and off every time the LFO's signal went above +.5 (as long as the other input was also above +.5).

a OR b

This equation is very similar to a AND b. The only difference is that the FUN will switch on when the value of either **input a** *or* **input b** moves above +.5.

Sawtooth LFOs

The next six equations case the FUN to generate a sawtooth LFO as its output signal. Each performs a different operation on the values of **inputs a** and **b**, and the resulting value is multiplied by 25. The result determines the frequency of the LFO. If the value is a positive number, the LFO has a rising sawtooth shape. If the value is negative, the LFO has a falling sawtooth shape. When the resulting values are large (above 10 or so), the output waveform is not a pure sawtooth; a bit of distortion occurs.

ramp(f=a + b)

The values of **inputs a** and **b** are added, then multiplied by 25.

ramp(f=a - b)

The value of **input b** is subtracted from the value of **input a**, and the difference is multiplied by 25.

ramp(f=(a + b) / 4)

The values of **inputs a** and **b** are added, and the sum is divided by 4. This value is multiplied by 25.

ramp(f=a * b)

The values of **inputs a** and **b** are multiplied, and the result is multiplied by 25.

ramp(f=-a * b)

The value of **input a** is multiplied by -1, then multiplied by the value of **input b**. The result is multiplied by 25.

ramp(f=a * 10^b),

10 is raised to the power of **b**, then multiplied by the value of **input a**. The result is multiplied by 25.

Chaotic LFOs

The next five equations function somewhat like the equation a(b-y) described earlier, in that they start with a value of 0 for y, evaluate the equation, and use the result as the new value of y for the next evaluation. Although they all can function as LFOs (they can have a repeating cycle of output values), they can become chaotic depending on the input values.

a(y + b)

The values of y and **b** are added, then multiplied by the value of **a**.

ay + b

The values of **a** and y are multiplied, then added to the value of **b**.

(a + 1)y + b

1 is added to the value of **a**. The sum is multiplied by the value of y. The result is added to the value of **b**.

y +a(y + b)

The values of y and **b** are added. The sum is multiplied by the value of **a**. The result is added to the value of y.

a |y| + b

The absolute value of y is taken (if it's a negative value, it's multiplied by -1). The absolute value of y is multiplied by the value of **a**. the result is added to the value of b.

Sample b On a

This is a sample and hold function. The values of **inputs a** and **b** are interpreted as logical quantities, as described for the equations a AND b, a OR b. When the value of **input a** changes from FALSE to TRUE (goes above +.5), the value of **input b** at that moment is sampled (recorded), and becomes the FUN's output value. This value remains constant until **input a** makes another transition from FALSE to TRUE.

Sample b On ~a

This works like the previous equation, but the value of **input b** is sampled whenever the value of **input a** makes a transition from TRUE to FALSE.
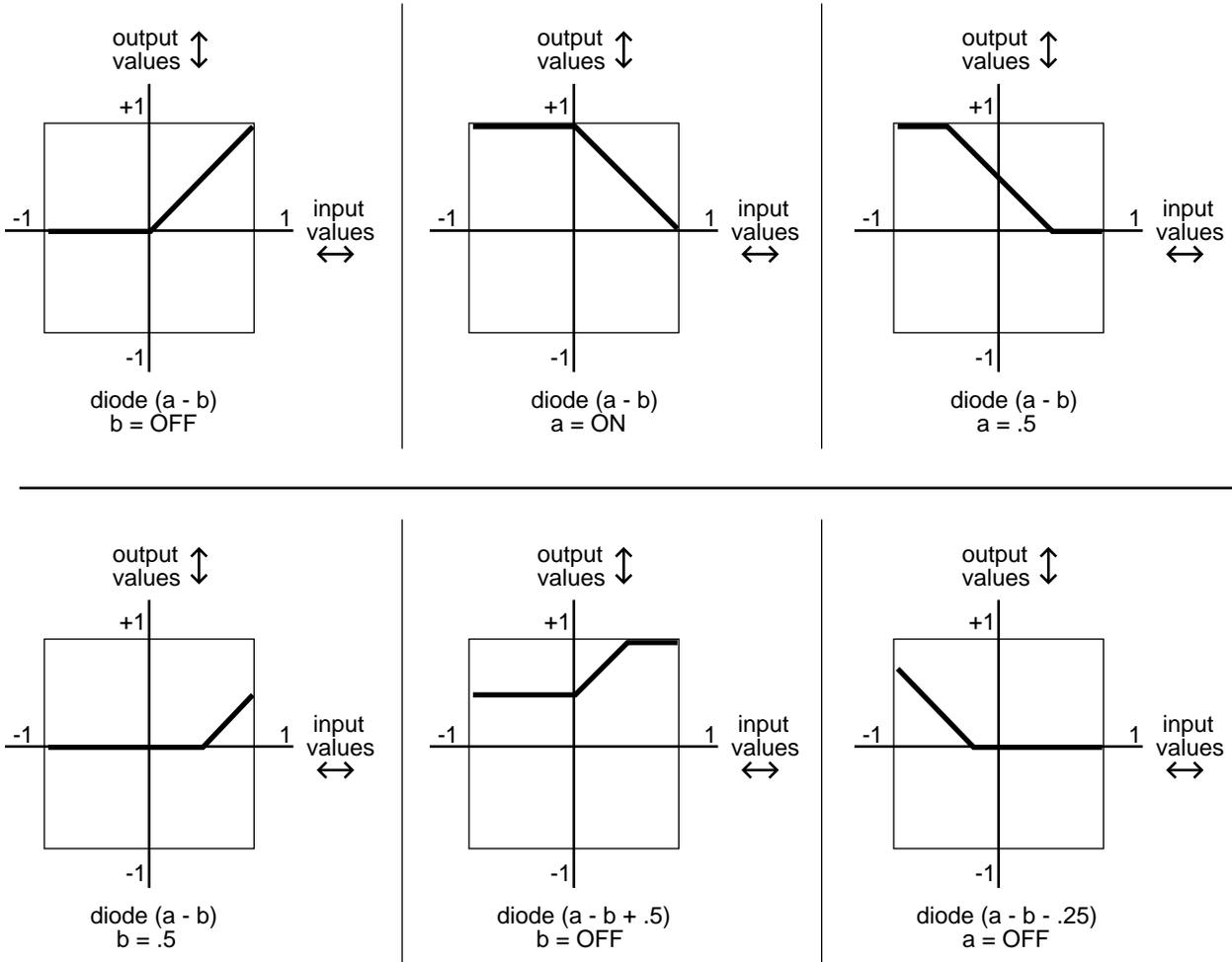
Track b While a

This equation also interprets the values of **inputs a** and **b** as logical quantities. While the value of **input a** is TRUE, the value of **input b** is used as the FUN's output value. The output value changes exactly as the value of **input b** changes. When the value of **input a** goes FALSE, the FUN's output value freezes and remains constant until the value of **input a** becomes TRUE again. The FUN's output value then continues to track the value of **input b**.

Track b While ~a

This is the opposite of the previous equation. The FUN's output value tracks the value of **input b** as long as the value of **input a** is FALSE.

Diode Equations

The remaining equations perform a diode-like function; only positive input values are significant. If the result of the equation is negative, the FUN's output value is 0. You can use these equations to limit bipolar control signals to unipolar values. Normally you'll use these by setting input a or b to ON or OFF, and assigning some control source to the other input. These will enable you to produce a variety of output curves.

output
values ↕

+1

-1   1   input
                                           values
                                           ↔

-1

diode (a - b)
b = OFF

output
values ↕

+1

-1   1   input
                                           values
                                           ↔

-1

diode (a - b)
a = ON

output
values ↕

+1

-1   1   input
                                           values
                                           ↔

-1

diode (a - b)
a = .5

output
values ↕

+1

-1   1   input
                                           values
                                           ↔

-1

diode (a - b)
b = .5

output
values ↕

+1

-1   1   input
                                           values
                                           ↔

-1

diode (a - b + .5)
b = OFF

output
values ↕

+1

-1   1   input
                                           values
                                           ↔

-1

diode (a - b - .25)
a = OFF

Diode (a - b) simply subtracts the value of **input b** from the value of **input a**. If the difference is less than 0, the output value is 0.

Diode (a - b + .5) adds a constant of +.5 to the difference of (a - b), then maps all negative values to 0. The curve is the same shape as Diode (a - b), but shifted upward 1/4 of the range between -1 and +1.

Diode (a - b - .5) is the same curve as diode (a - b), shifted downward 1/4 of the range.

Diode (a - b + .25) shifts the curve up 1/8 of the range, and diode (a - b - .25) shifts the curve down 1/8 of the range.

The Order of Evaluation for FUNs

The K2500 is a computer, and processes information at very high speeds. Every 20 milliseconds, it checks the condition of every active parameter, evaluates any changes, and processes the new information. This is done according to a rigid set of priorities that determines the sequence in which the parameters are evaluated.

The status of each control source is evaluated in turn, according to the sequence in which they appear in the Control Source list. In the case of the FUNs, they are evaluated in the following order: FUN1, FUN2, FUN3, FUN4 (although there are a few control sources that get evaluated between FUN2 and FUN3).

This sequence of evaluation becomes significant if you assign a FUN as the input for another FUN. You should always assign FUNs as inputs for higher-numbered FUNs. For example, you might assign FUN1 as input a for FUN2. Since the K2500 needs to know the value of FUN1's output before it can evaluate FUN2, assigning the FUNs in this way will ensure that the K2500 can evaluate both FUNs as quickly as possible.

If you were to assign FUN2 as an input for FUN1, then when the time came for the K2500 to evaluate FUN1, it wouldn't know the current value for FUN2's output, so it would evaluate FUN1 according to the previous value of FUN2. There would be a delay of one evaluation cycle before a change in FUN2 would be reflected in FUN1. This might have an adverse affect on the start of notes as you play. You won't harm anything, but you might not hear what you expect to hear.