APPENDIX C:
SPECIFICATIONS AND BLOCK DIAGRAMS FOR SIGNAL PROCESSING MODULES AND SYSTEM CONTROL ALGORITHMS USED IN *CORT LIPPE'S MUSIC FOR CLARINET AND ISPW*

The following series of signal processing module block diagrams and pseudo-code algorithm outlines are intended to supplement the analysis given in chapter 8. For the block diagrams, I have adopted a set of symbols based those found in many standard textbooks on computer music and digital synthesis. A key to the symbols used in this appendix is shown in figure C.1. Examples of Max/MSP signal processing patchers with corresponding diagrams are given in figures C.2 – C.4. Detailed specifications and signal routing schematics are given for the DSP modules (Sampler, Harmonizer, Frequency Shifter, Flange, Frequency/Amplitude Modulation, Reverb, Noise Modulation, and Spatializer) along with pseudo code outlines of their associated control algorithms.

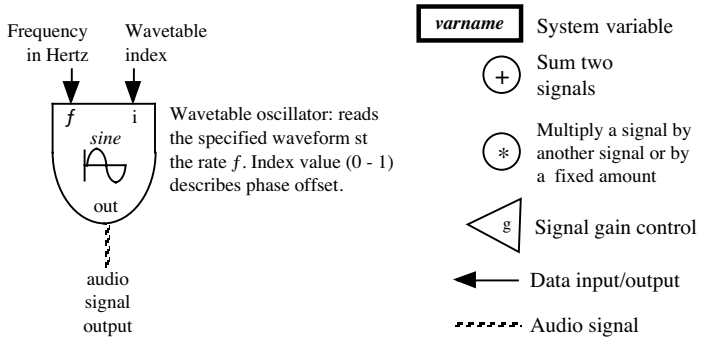


Figure C.1. Basic symbols used in the block diagrams of Cort Lippe's signal-processing software instruments

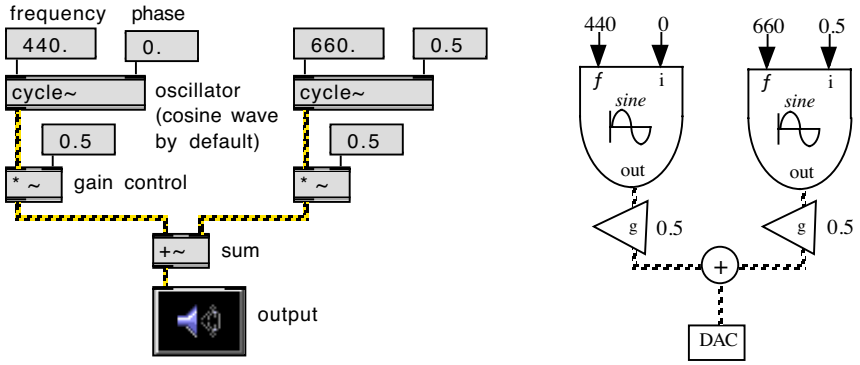


Figure C.2. A simple Max/MSP instrument for additive synthesis and its corresponding block diagram

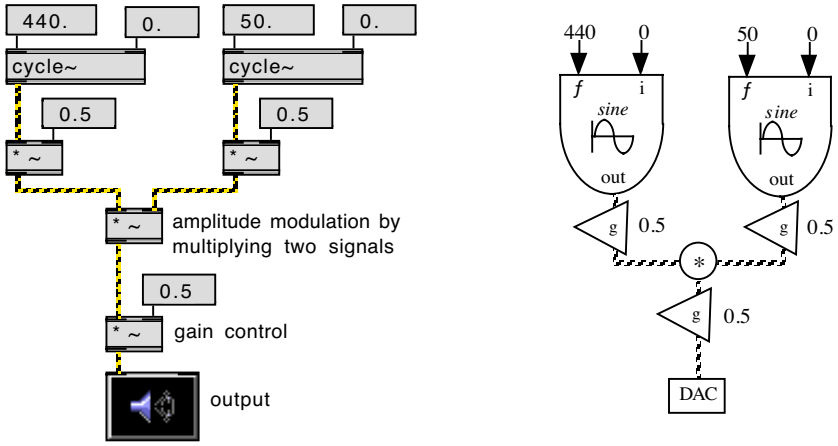


Figure C.3. A Max/MSP instrument for amplitude modulation (AM synthesis) and its corresponding block diagram

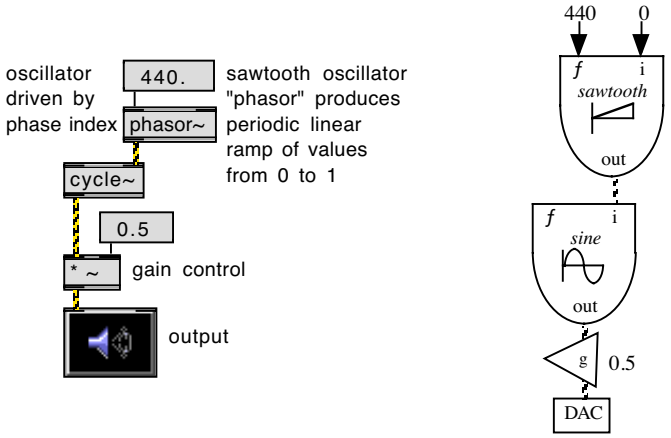
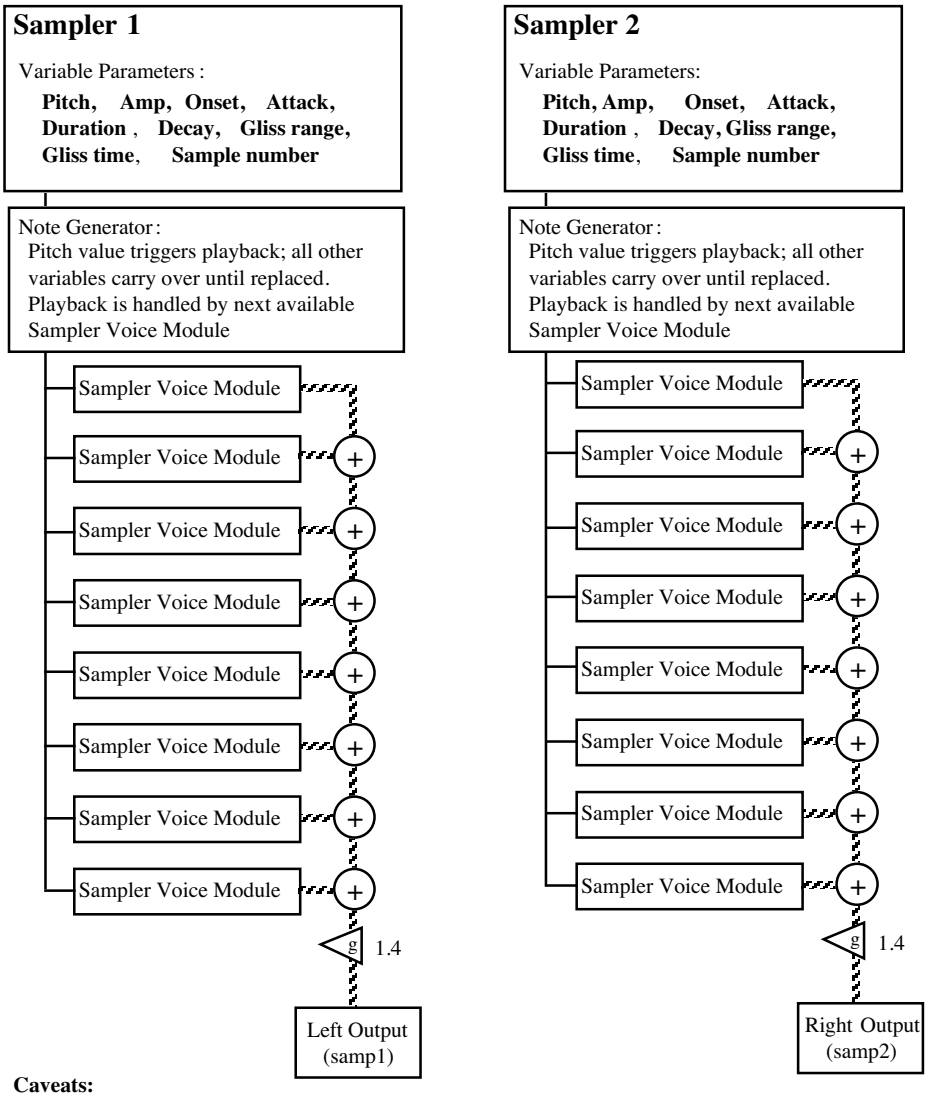


Figure C.4. A phase-driven wavetable oscillator in Max/MSP and its corresponding block diagram

The following are block diagrams of Lippe's signal processing modules and pseudo-code versions of their associated control algorithms, if applicable. The sampler is the central component of the system, and is therefore subject to multiple complex control algorithms. Figures C.6 and C.6 show the sampler instrument. Figures C.7, C.8, and C.9 show pseudo-code outlines of the three main granular sampling algorithms "Trevor," "Trevor-Back" and "PLAY_RAND." Eight more sampler control algorithms are associated with specific score events, and these are shown in figures C.10 – 17. The remaining DSP instruments,

SAMPLER



Caveats:

1. Note Generators: list of parameter values for sampler note data is sent to a Sampler Voice Module when a pitch value is received from the event list or from another processing module. Previously stored values (for parameters other than pitch) will be carried over unless a new value is received.

2. Eight-voice polyphonic voice allocation: each note generated is automatically allocated to an avalable Sampler Voice Module. "Voice Stealing" grabs the least recently used module if more than eight voices are needed simultaneously.

3. Sampler sounds are generated as MIDI notes: Note number determines pitch, velocity determines amplitude. Duration is determined by sending a corresponding note witha velocity (amp) of 0 at the appropriate time.

Figure C.5. Samplers 1 and 2: playback is controlled by formulating MIDI-style note packets of variable parameters for pitch, velocity, duration, envelope, and glissando

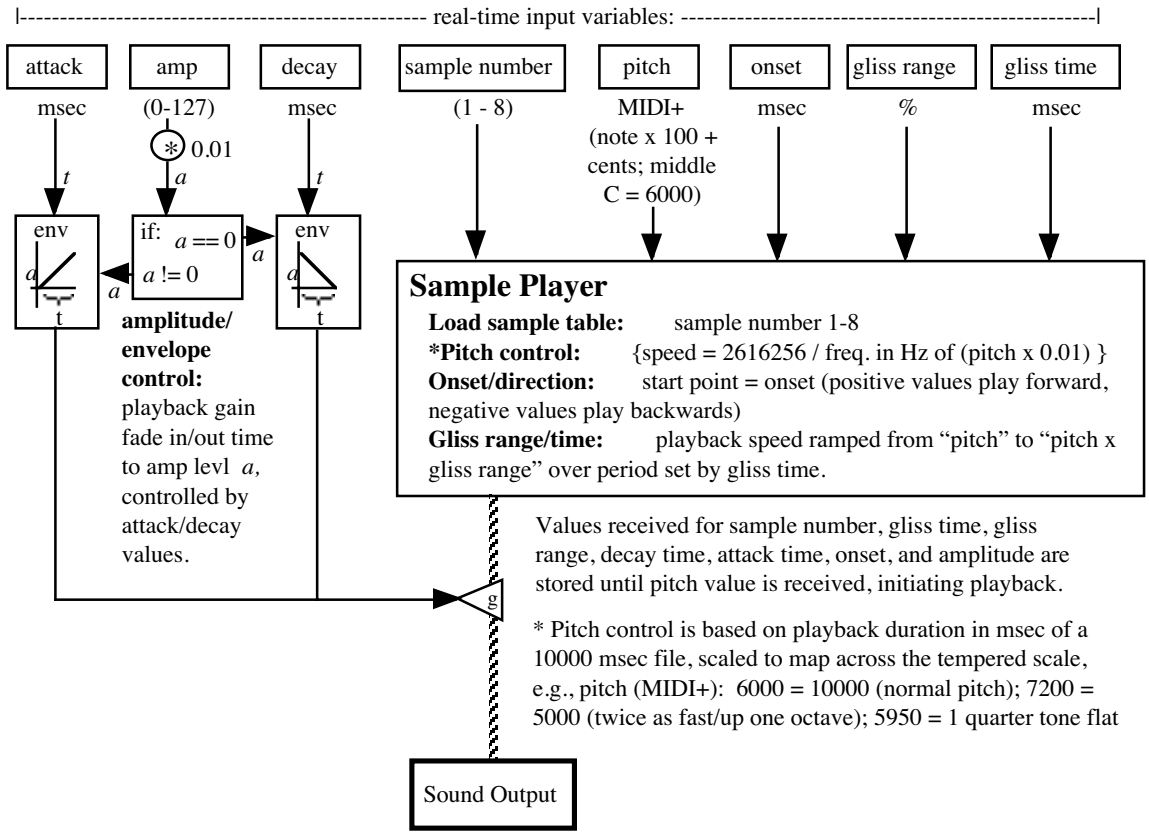


Figure C.6. Sampler Voice Module: each of the 16 sampler voice modules can read from any of the buffered sample files. Input variables control pitch, amplitude, file onset, envelope attack and decay rates, and glissando (range and duration).

```
START is set by one of the following variables received:
  {
  START = t-start = 0; OR
  START = t-start1 = 0; OR
  START = t-start2 = 200000; OR
  START = t-start3 = 400000; OR
  START = t-start4 = 600000; OR
  START = t-start5 = 800000; OR
  START = tt-start = (0 – 10000) * 100
  }

LOOP every 10 to 20 milliseconds (10 + random number from 0 to 10)
UNTIL t-stop message is received {
  onset = (START + (value of t-precess * 11)) / 100
  velocity = 75
  duration = 50
  pitch = current value of t-transpose (may be changed while the
    loop is running)
  PLAY note on Sampler1 (previous values for sample table, attack,
    decay, gliss, and gliss-duration are maintained)
  }
```

Figure C.7. “Trevor.” Granular sampling playback initiated by a “t-start” variable in the event list

```
START is set by one of the following variables received:
  {
  START = b-start = 200000; OR
  START = b-start1 = 200000; OR
  START = b-start2 = 400000; OR
  START = b-start3 = 600000; OR
  START = b-start4 = 800000; OR
  START = b-start5 = 970000; OR
  START = bb-start = (0 – 10000) * 100
  }

LOOP every 10 to 20 milliseconds (10 + random number from 0 to 10)
UNTIL b-stop message is received {
  onset = (START - (value of b-precess * 11)) / 100
  velocity = 75
  duration = 50
  pitch = current value of b-transpose (may be changed while the
    loop is running)
  PLAY note on Sampler2 (previous values for sample table, attack,
    decay, gliss, and gliss-duration are maintained)
  }
```

Figure C.8. “Trevor-back.” Backwards granular sampling initiated by a “b-start” variable in the event list

```
IF play-rand message is received THEN {
  CALL FUNCTION play-rand1
  CALL FUNCTION play-rand2
}

FUNCTION play-rand1 {
  LOOP every 10 to 60 milliseconds (10 + random number from 0 to 50
    (SEED: value of play_rand_metro1)) {
      CALL FUNCTION rnd-gliss
      CALL FUNCTION playnote-1
  }
} END of FUNCTION

FUNCTION play-rand2 {
  LOOP every 10 to 60 milliseconds (10 + random number from 0 to 50
    (SEED: value of play_rand_metro2)) {
      CALL FUNCTION playnote2
  }
} END of FUNCTION

FUNCTION Playnote-1 {
  onset = play-rand-ondur1 + random number from 0 to 4000 (SEED:
    value of play-rand-onset1)
  velocity = 80
  duration = 10 + random number from 0 to 400 (SEED: value of play-
    rand-dur1)
  pitch = (value of play-rand-pit1 + random number from 0 to 100
    (SEED: play-rand-pchval1)) – (play-rand-pchval1 / 2)
  PLAY note on sampler1
} END of FUNCTION

FUNCTION rnd-gliss {
  IF value of rand-gliss-gate is 1, THEN {
      N = value of counter % 8
      counter = counter + 1
      IF value of N is {
         0: gliss = 0.8
         1: gliss = 1.05
         2: gliss = 0.7
         3: gliss = 1.1
         4: gliss = 0.6
         5: gliss = 1.15
         6: gliss = 0.5
         7: gliss = 1.2
      }
  }
  IF value of rand-bang8-gate is 1 THEN {
      CALL FUNCTION playnote-1 8 times in immediate succession
         using current parameters
  }
  IF value of ranp-bang4-gate is 1 THEN {
      CALL FUNCTION playnote-1 8 times in immediate succession
         using current parameters
  }
} END of FUNCTION
```

Figure C.9. “PLAY_RAND.” Granular sampling controlled by random processes

```
IF value of runsgate is 1 THEN
{
   glisstot = RAMP from previous value to value 1 of glisstot over
     duration specified by glisstot value 2
   N = value of metro_playtot_val
   LOOP every N msec
   {
      PLAY note on Sampler1
      {
         Duration = 261 / frequency in Hz of (glisstot + random
            number from 0 to 50) * 9500
         Pitch = glisstot + (random number from 0 to 50)
         Onset = 0
         Velocity = 75
      }
   }
}
```

Figure C.10. Sampler control algorithm 1: Section I, events 5 – 6 and 9

```
IF the value of metro_evt5 is 1, THEN
   SET sampler1 onset to 0;
   LOOP: every N milliseconds
   {
      N = rspeed_evt5 + a random number from 0 to 400 (seed:
      speed_evt5)
      PLAY Sampler1
      {
        Pitch = 6000 + value of glis_evt5
        Velocity = 120
        Duration = value of speed_evt5_dur
        }
   }
```

Figure C.11. Sampler control algorithm 2: section I, events 5 – 10

```
IF the value of ichgate is 1, THEN
{
  SET the value of tt02, sto2, tto4, and sto4 to 127; (full volume)
  IF the value of pitch-track-out is between 50 and 52, THEN
  {
     TRIGGER spatXY spatialization algorithm;
     SET b-precess value to [pitch-track-out * b-precess-val];
     SET b-transpose value to [pitch-track-out – 3];
     TRIGGER granular sampling module Trevor-back with onset value
     set by brevor-onset;
     }
  IF the value of pitch-track-out is between 76 and 79, THEN
  {
     SET t-precess value to [pitch-track-out * t-precess-val];
     SET t-transpose value to value of pitch-track-out;
     TRIGGER granular sampling module Trevor with onset value set
     by Trevor-onset;
     }
```

Figure C.12. Sampler control algorithm 3: "Trevor" in section I, events 12 – 16 (also shown in figure 8.11)

```
IF value of evt11-sec2 is 1, THEN
{
   PLAY Sampler2
   {
      pitch = 6000
      duration = 5000
      onset = 0
      attack = 10
      decay = 50
      gliss = 1.0
   }
   LOOP every (150 + random number from 0 to 700) msec
   {
      PLAY Sampler2
      {
         pitch = 5300 + random number from 0 to 1500
         (Dur., Onset, etc. carry over from previous)
      }
   }
}
```

Figure C.13. Sampler control algorithm 4: Section II, events 11 and 26

```
IF start-23 = 1 THEN
{
   LOOP every 0 to 2000 msec (randomized)
   {
      N = random number from 0 to 2000
      IF value of N is 0, THEN t-stop (stop Trevor)
      IF value of N is between 1 and 999, THEN
      {
          LOOP every 20 to 520 msec
          {
            t-precess = random number from 0 to 256
            t-transcent = 3000 + (random number from 0 to 3000)
            t-start (start Trevor granular sampling module)
            SEND t-stop after 500 msec delay
          }
      }
      IF value of N >= 1000
      {
          Play-rand2 = 1 (see PLAY_RAND)
          Stop PLAY_RAND after 100 msec delay
      }
   }
}
```

Figure C.14. Sampler control algorithm 5: Section III, event 23

```
IF value of ichgate3 is 1 THEN
{
   IF (value of pitch_track_out – 6) is between 64 and 76, THEN
   {
      t-precess = (pitch_track_out – 6) * 10
      t-transcent = (pitch_track_out – 6) * 100
      SEND tt-start message to Trevor
   }
   ELSE IF (value of pitch_track_out – 6) is between 77 and 92,
   THEN
   {
      N = random number from 0 to 5
      IF N is {
         0: t-precess = 840; t-transcent = RAMP to 8400 in 400 msec
         1: t-precess = 830; t-transcent = RAMP to 8300 in 400 msec
         2: t-precess = 820; t-transcent = RAMP to 8200 in 400 msec
         3: t-precess = 810; t-transcent = RAMP to 8100 in 400 msec
         4: t-precess = 800; t-transcent = RAMP to 8000 in 400 msec
         5: t-precess = 790; t-transcent = RAMP to 7900 in 400 msec
         }
      SEND tt-start to Trevor
      }
      }
   }
}
```

Figure C.15. Sampler control algorithm 6: Section III, event 26

```
IF bang4-line = 1 THEN
{
   LOOP every 1000 to 4000 msec (1000 + random number from 0 to
   3000 (seed: bang-ran)
   {
      START RAMP: go from previous value to new value over ramp-
      time, incrementing at 30 msec intervals
      {
          Ramp-time = bang-ran + random 3000 (seed: bang-ran)
          FOR each RAMP increment
          {
            Pitch = bang-pit + random number from 0 to 3000
          }
      }
   }
}
```

Figure C.16. Sampler control algorithm 7:
send ramping pitch values to PLAY_RAND in Section IV, event 3

```
IF value of runsgate is 1 THEN
{
   FOR every pitch-track-out value received
   {
      PLAY note on Sampler1
      {
         Duration = 261 / frequency in Hz of (value of pitch-track-
            out) * 9500
         Pitch = pitch-track-out * 100
         Onset = 0
         Velocity = 75
      }
   }
}
```

Figure C.17. Sampler control algorithm 8: play Sampler1
in response to pitch tracker values in Section V, events 3 - 6

HARMONIZER

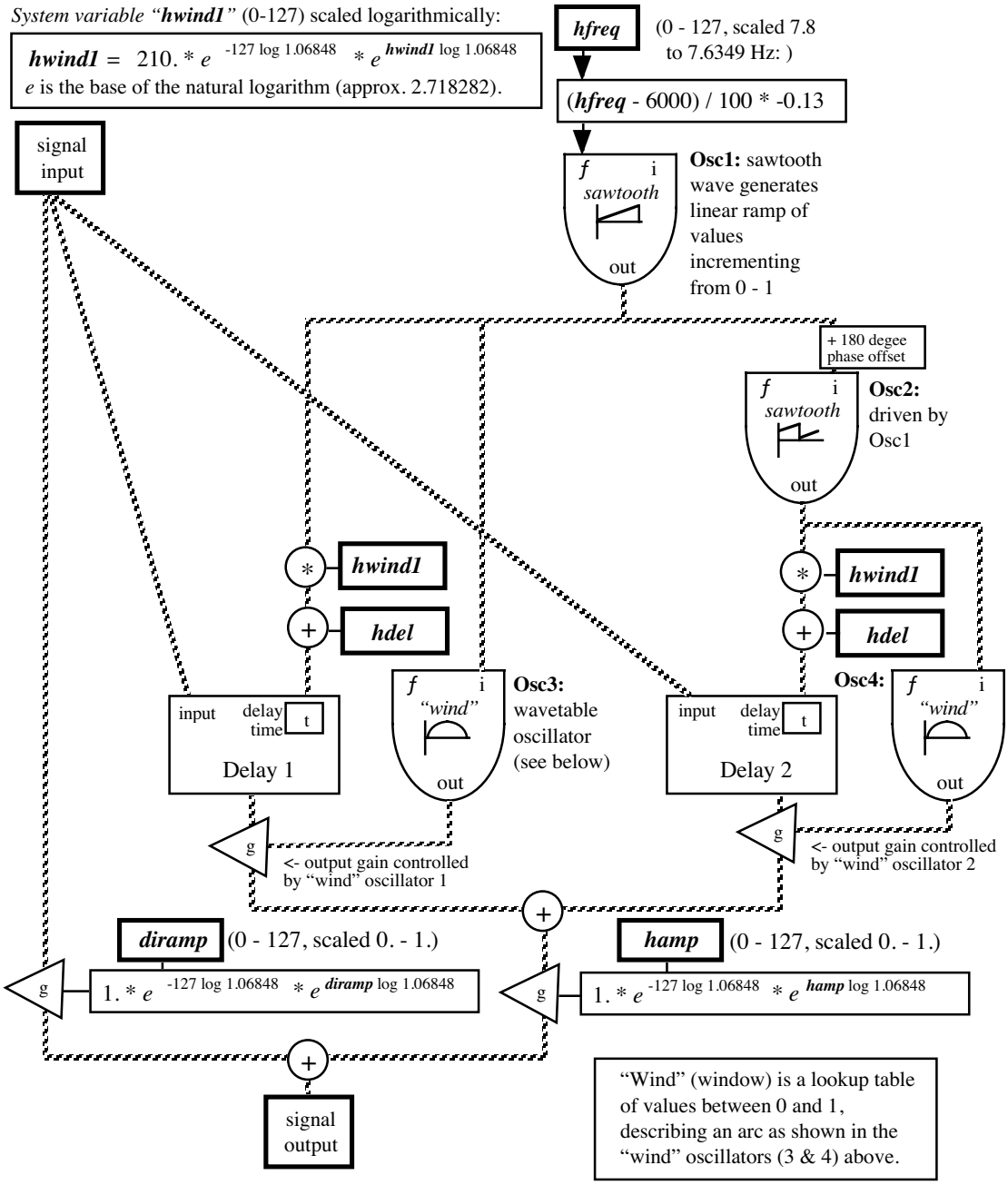


Figure C.18. Two delay lines with delay time increased at a constant linear rate. Delays are synchronized by osc1, but delay time increments are out of phase by 180 degrees. Amplitude window ("wind" oscillators) controls envelope for delay output, cross-fading between the two delay lines.

FREQUENCY SHIFTER

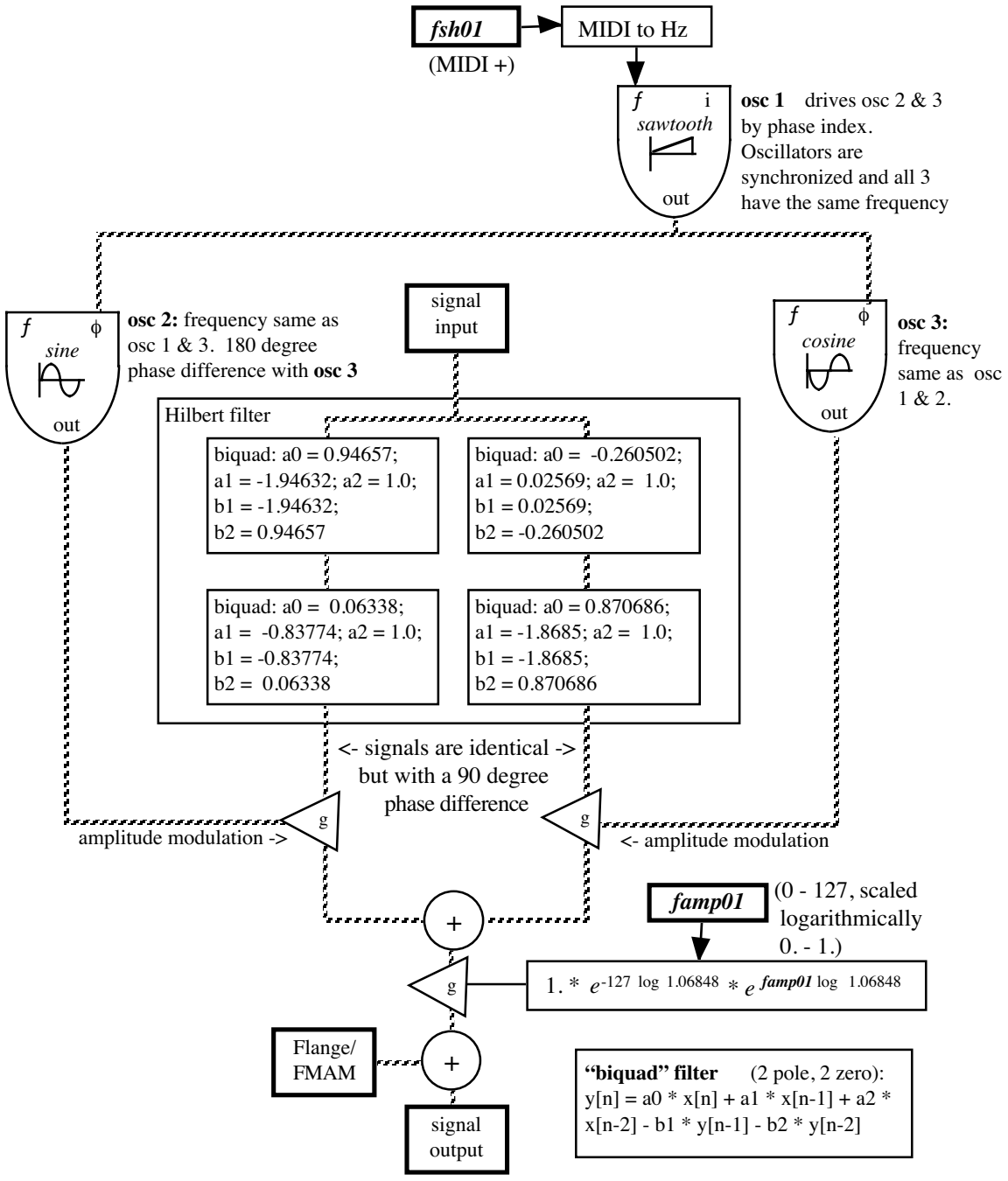


Figure C.19. Frequency shifter: all harmonic components are shifted individually up or down by a fixed frequency interval

FLANGE

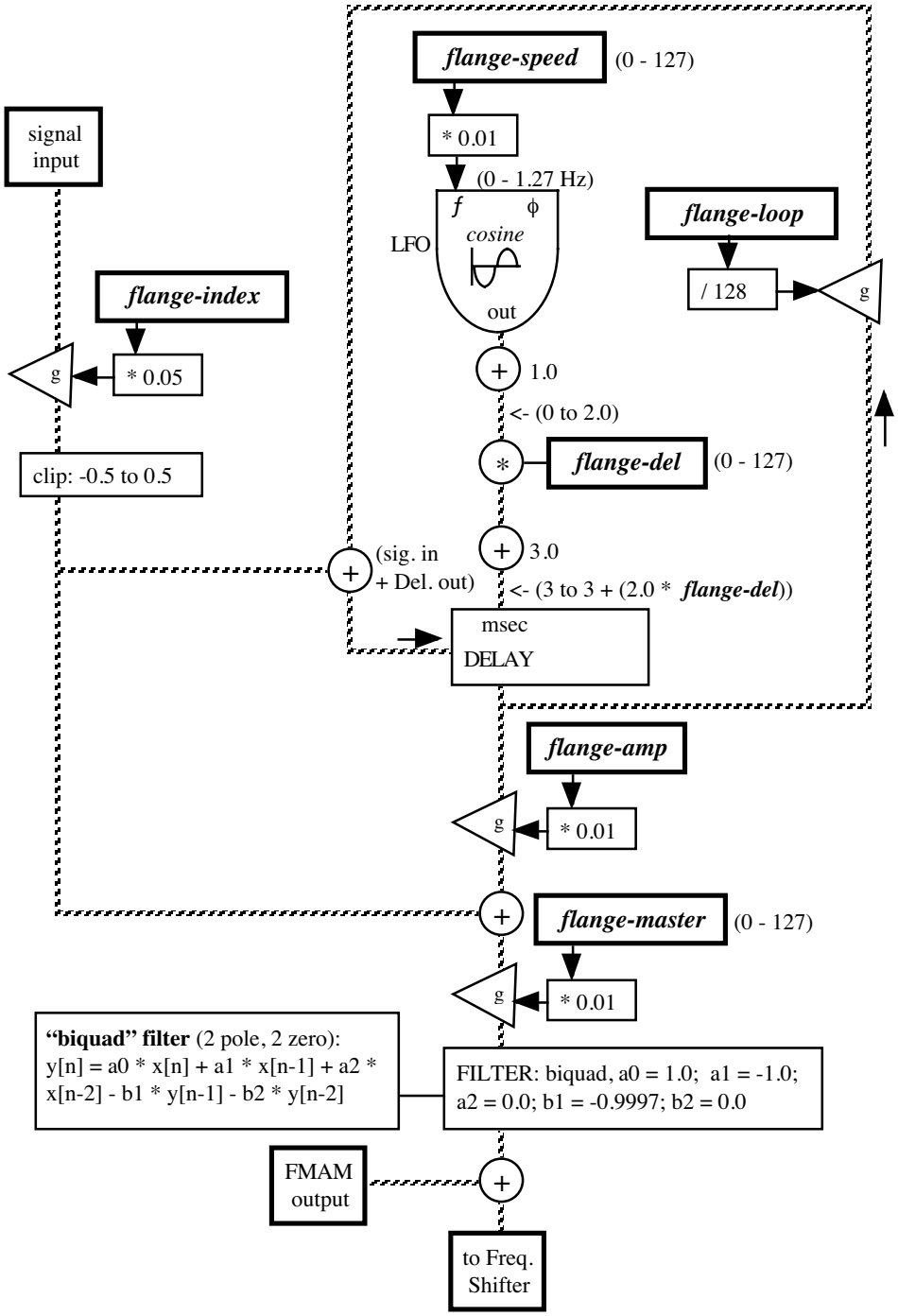


Figure C.20. A typical flange effect created by a short delay with LFO modulation

FREQUENCY/AMPLITUDE MODULATION

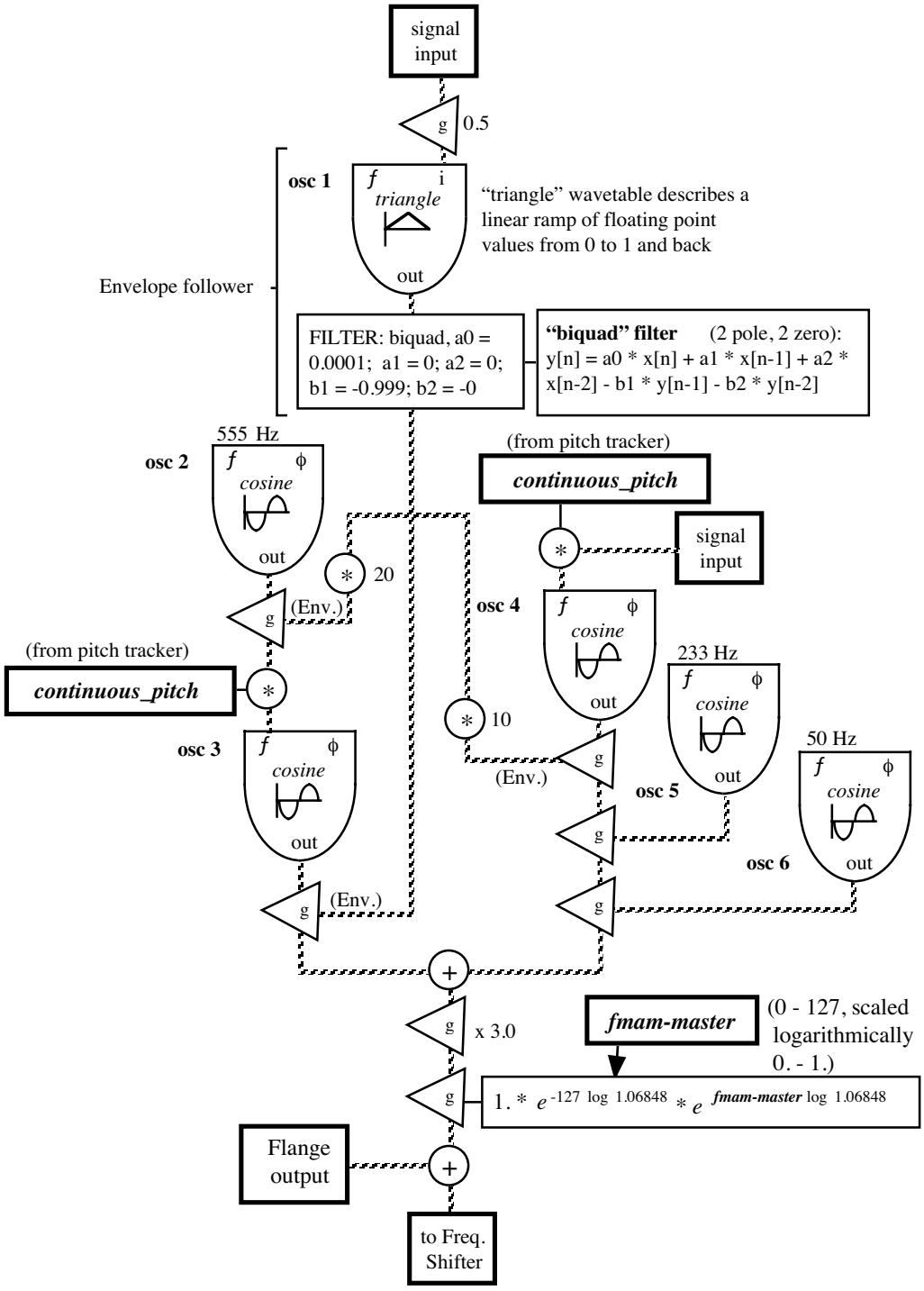


Figure C.21. Combination of FM and AM synthesis based on real-time analysis of clarinet pitch and amplitude.

```
IF the value of fsgate18 is 1, THEN:
  IF pitch-track-out value is 77,
     THEN {
     SET fsh01 value to a random number from –2000 to –6499.
     }
  IF the value of pitch-track-out is between 50 and 59,
     THEN {
     SET hfreq value to a random number from 40 to 63;
     SET ptoh value to 122; (clarinet signal vol. to harmonizer)
     }
  ELSE, SET ptoh value to 0.
  IF pitch-track-out value is between 71 and fstend18 value,
     THEN {
     SET ptof value to 127; (clar. signal vol. to freq. shifter)
  ELSE, SET ptof value to 0.
```

Figure C.22. Harmonizer/Frequency Shifter control:

Section I, event 18 and Section II, event 1

REVERB

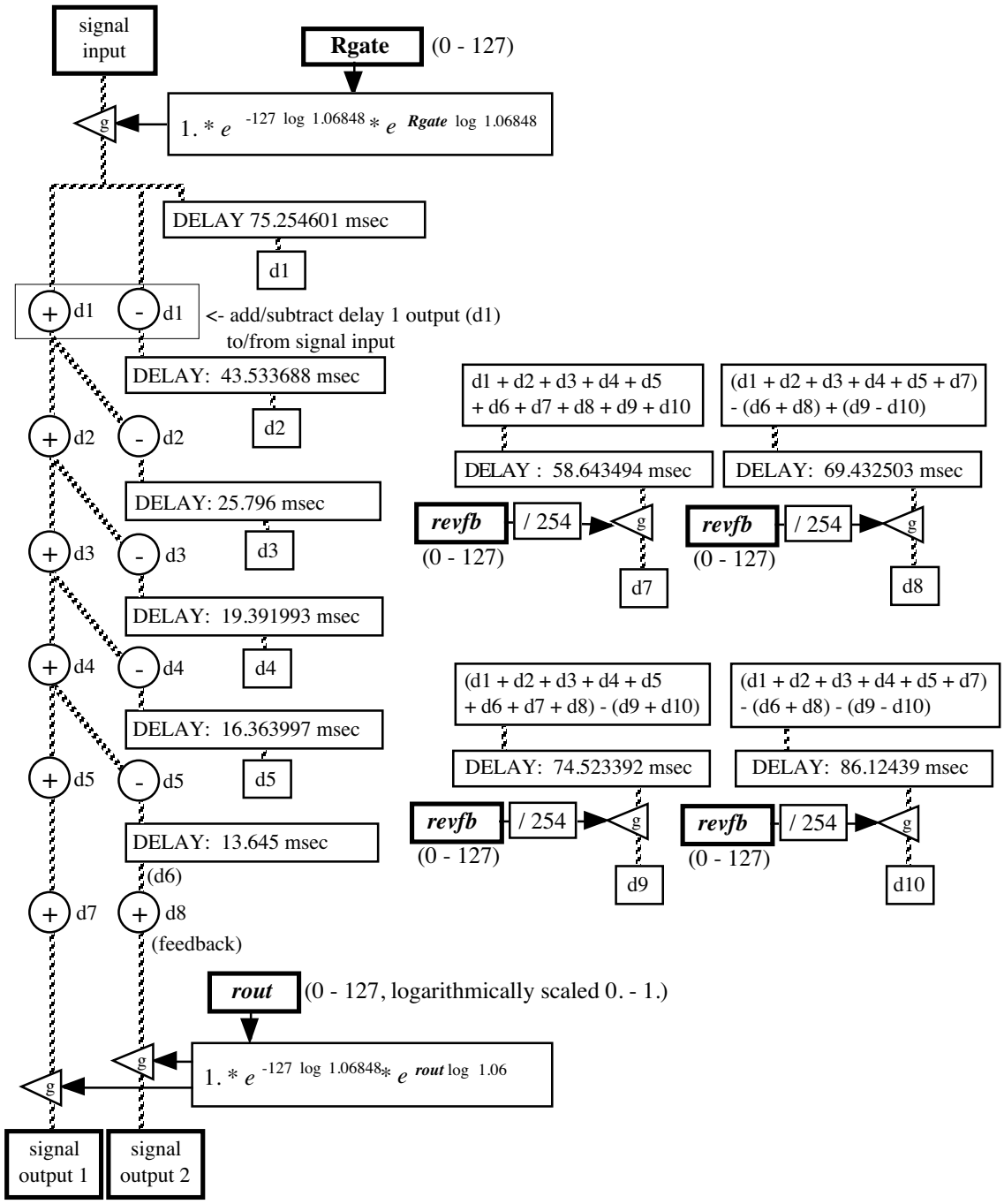


Figure C.23. Reverb: the input signal is subjected to a series of delays with variable feedback

NOISE MODULATION

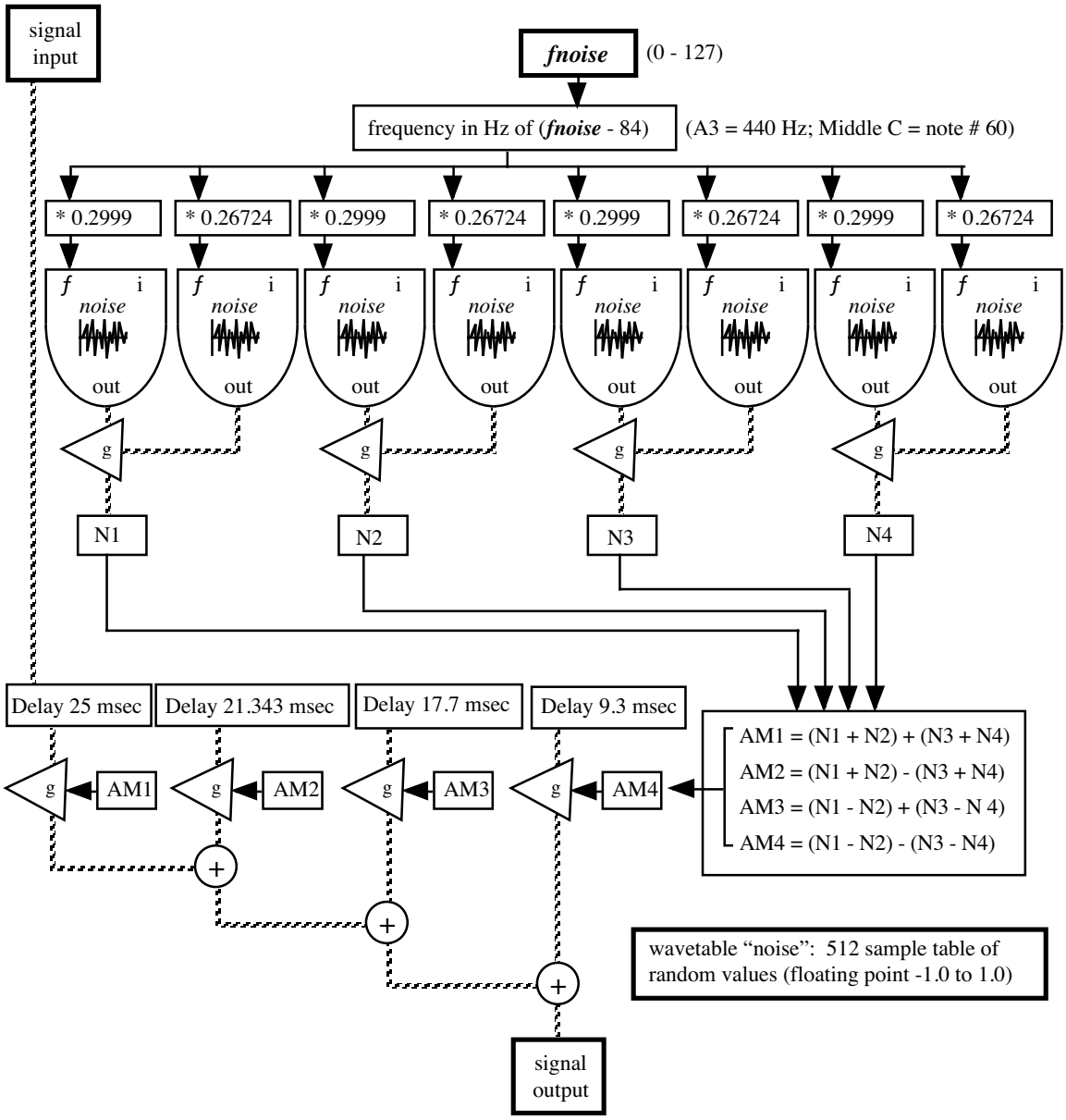


Figure C.24. Noise Modulation: random amplitude envelopes generated by multiple noise wavetable LFOs

SPATIALIZER

Input Crossbar

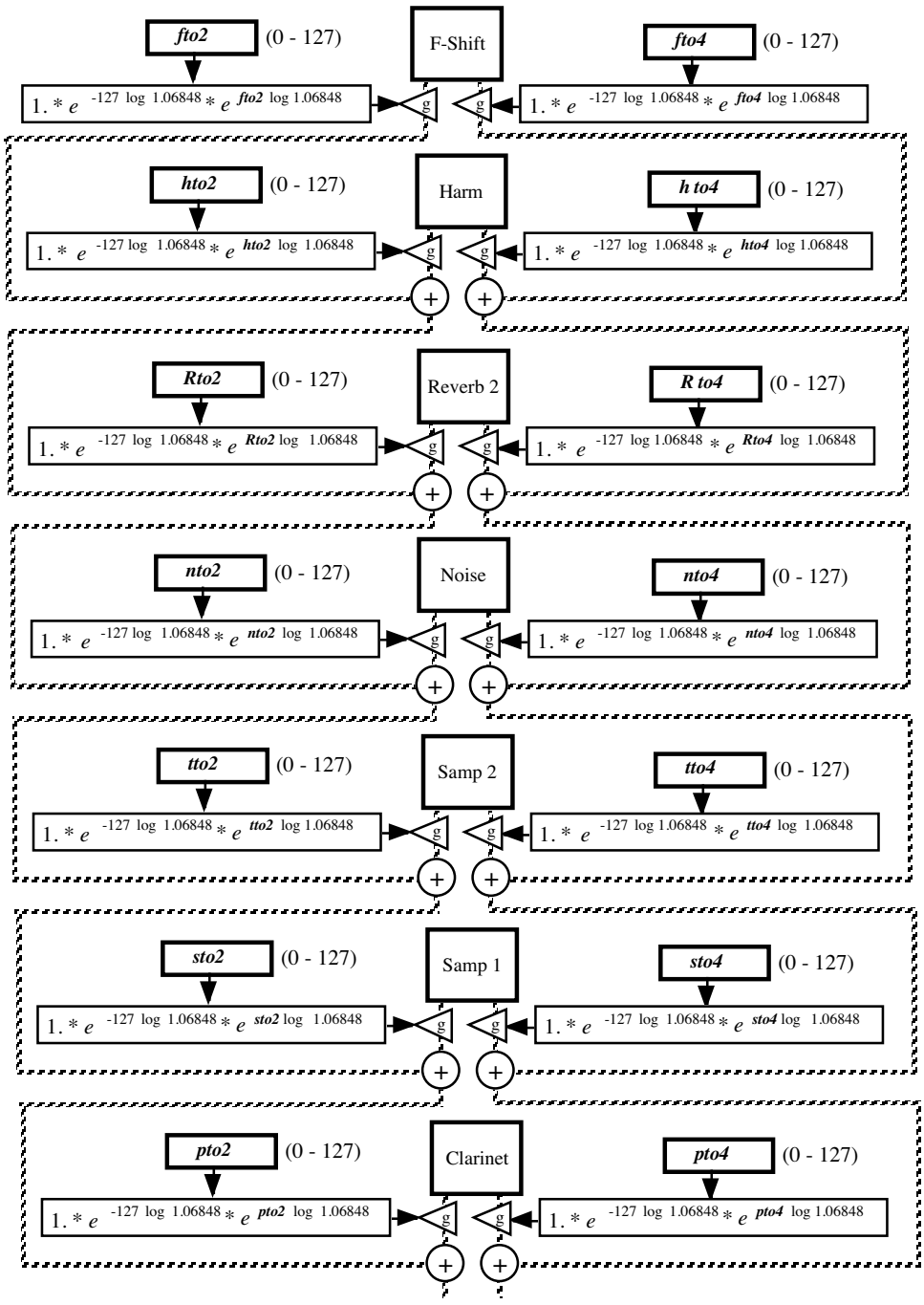


Figure C.25. Spatializer input crossbar: inputs from all DSP modules are mixed individually for left and right channel output.

Left-Right placement of DSP and Sampler output is controlled per output channel by individual variables for each DSP module. The input crossbar method shown in figure C.25 is used to manage input to all of the individual DSP modules as well (except the Sampler). Final output to the loudspeakers is shown in figure C.26.

Spatializer Output

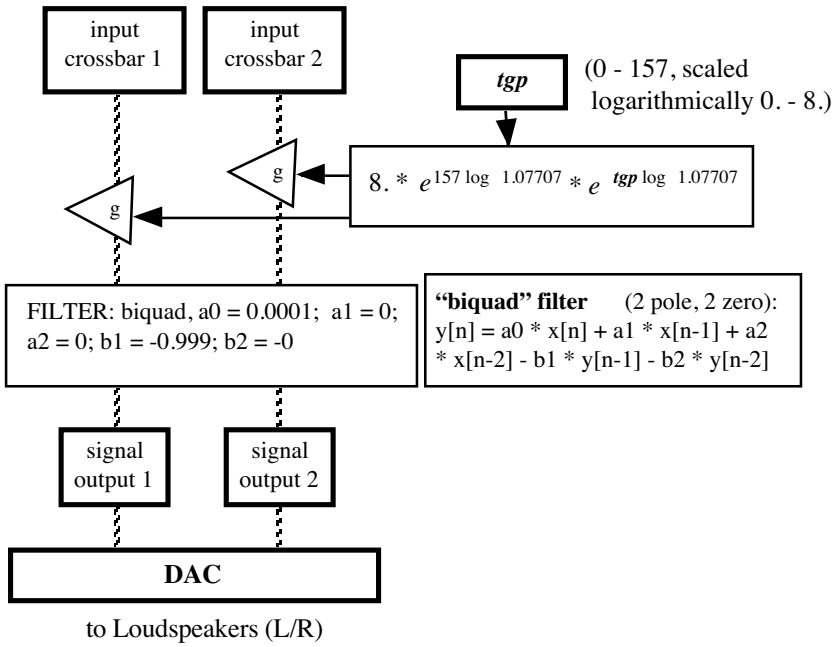


Figure C.26. Spatializer output: signal from the input crossbar is scaled and sent to the Digital to Analog Converters (DAC) for amplification via loudspeakers

```
IF value of ichgate is 1 THEN {
   IF value of pitch_track_out is between 50 and 62 THEN {
      IF spagate is 1 THEN {
        N = random number from 0 to 1
        IF value of N is 0, THEN spatx = 127 AND spaty = 0
        ELSE spatx = 0 AND spaty = 127
      }
      IF s-spatR is 1, THEN stoR = spatx
      IF t-spatR is 1, THEN ttoR = spaty
      IF f-spat is 1, THEN fto2 = spatx AND fto4 = spaty
      IF h-spat is 1, THEN hto2 = spatx AND hto4 = spaty
      IF R-spat is 1, THEN Rto2 = spatx AND Rto4 = spaty
      IF n-spat is 1, THEN nto2 = spatx AND nto4 = spaty
      IF s-spat is 1, THEN sto2 = spatx AND sto4 = spaty
      IF t-spat is 1, THEN tto2 = spatx AND tto4 = spaty
   }
}
```

Figure C.27. Spatializer control algorithm: section I, events 12- 16. Hard-left or hard-right placement of the computer generated sound is chosen randomly each time a note between 50 and 62 is played and is detected by the pitch tracker.

```
IF value of spatXY-19 is 1 THEN {
  LOOP every 1000 to 6000 msec (1000 + random value 0 to 5000
    (seed: spatXY-metro))
  {
      N = random value from 0 to 1
      IF N is 0, THEN {
          spatx = 127
          spaty = 0
      }
      ELSE {
          spatx = 0
          spaty = 127
      }
      IF s-spatR is 1, THEN stoR = spatx
      IF t-spatR is 1, THEN ttoR = spaty
      IF f-spat is 1, THEN fto2 = spatx AND fto4 = spaty
      IF h-spat is 1, THEN hto2 = spatx AND hto4 = spaty
      IF R-spat is 1, THEN Rto2 = spatx AND Rto4 = spaty
      IF n-spat is 1, THEN nto2 = spatx AND nto4 = spaty
      IF s-spat is 1, THEN sto2 = spatx AND sto4 = spaty
      IF t-spat is 1, THEN tto2 = spatx AND tto4 = spaty
  }
}
```

Figure C.28. Spatializer control algorithm: random left-right placement of DSP modules' output in section II, event 19 through two seconds after the onset of event 20

```
IF value of spaton is {
   1: N = 0;
      REPEAT every 20 msec {
         "spaf1.t" table index = (N & 255) + (spatinc – 64)
         "spaf2.t" table index = ((N & 255) + (spatinc – 64)) >> 1
         N = (N & 255) + (spatinc – 64)
      }
   2: N = 0;
      REPEAT every 20 msec {
         "spaf3.t" table index = (N & 255) + (spatinc – 64)
         "spaf4.t" table index = ((N & 255) + (spatinc – 64)) >> 1
         N = (N & 255) + (spatinc – 64)
      }
   3: N = 0;
      REPEAT every 20 msec {
         "spaf5.t" table index = (N & 255) + (spatinc – 64)
         "spaf6.t" table index = ((N & 255) + (spatinc – 64)) >> 1
         N = (N & 255) + (spatinc – 64)
      }
   4: N = 0;
      REPEAT every 20 msec {
         "spaf7.t" table index = (N & 255) + (spatinc – 64)
         "spaf8.t" table index = ((N & 255) + (spatinc – 64)) >> 1
         N = (N & 255) + (spatinc – 64)
      }
   }
}
```

Figure C.29. Algorithmic control of spatializer using event list variable *spaton*

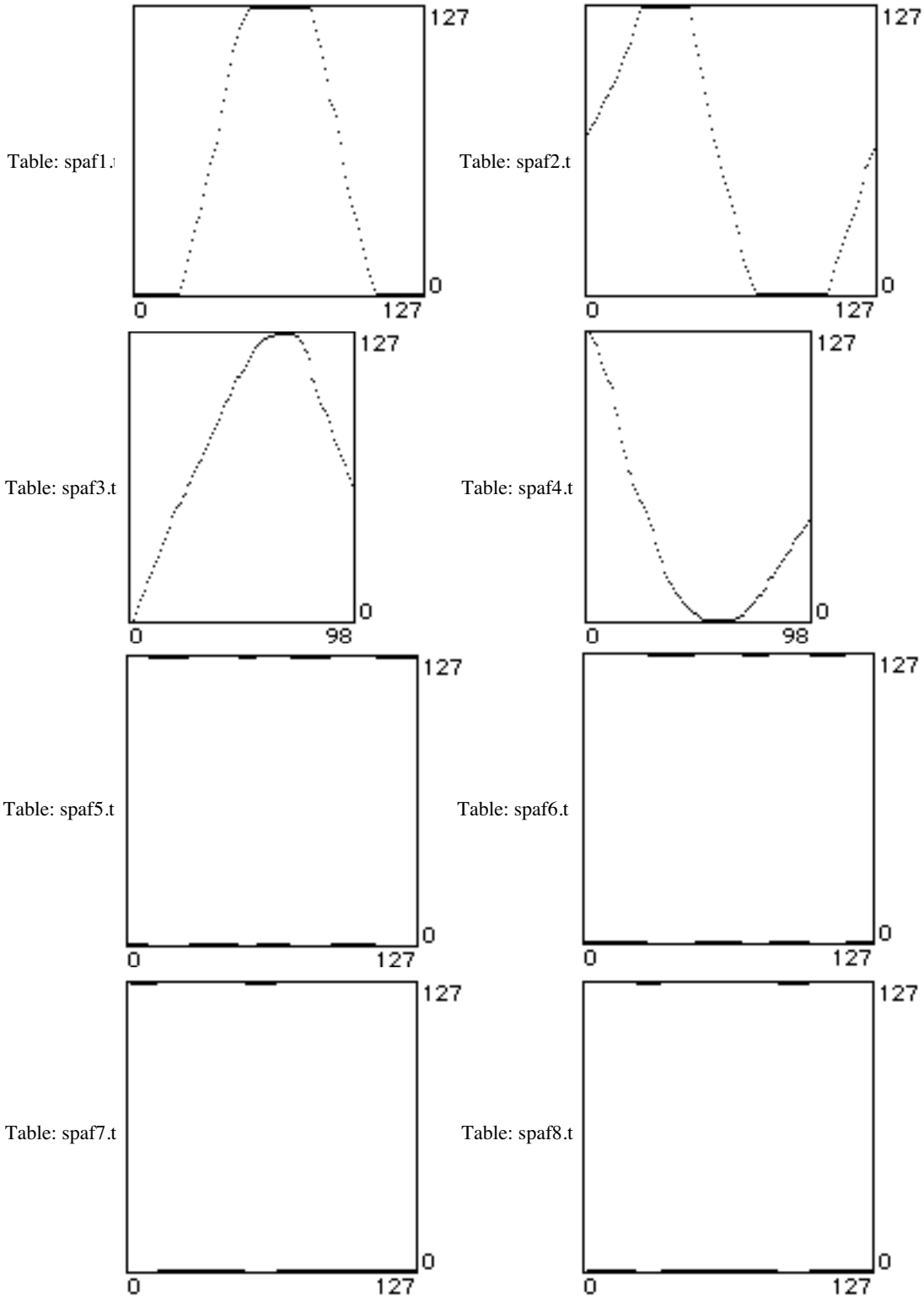


Figure C.30. Amplitude tables used by "spaton" algorithm