LAB NOTES: MUS 1 with the new miracle ingredient – stg

by John S. Simonton, jr.

With the exception of the bare-bones listing of POLY 1.0 that ran in the last issue, we haven't looked at any software- mainly because there was little to examine.

But MUS1 was just recently finalized, so that situation is beginning to change.

MUS1, for the benefit of those of you who haven't been waiting for it for the last six months, is what many would call "system firmware"- and since that has the sort of technical ring to it that tends to make things interesting, we'll call it that, too.

In almost any computer application there are some programs which, for one reason or another, are best handled as firmware- a name that these days means not software (which must be loaded from some storage media external to the computer) and not hardware (a permanently wired collection of gates, etc. which cause a specific, set sequence of actions to take place) but something betwixt and between; most usually, software that is contained in a PROM somewhere.

The most obvious firmware is a monitor program such as PIEBUG. Since this program is the thing that allows for the entry of data and instructions into the memory of the computer in the first place (as well as usually providing whatever de-bugging and editing features the designer thought were important and/or had room for), it is at least inconvenient to have to load it every time it is needed. Much better to have it in a dedicated PROM where it is always available for immediate use.

The firmware of MUS1 is roughly analogous. These are universally useful routines that, with rare exceptions, will be used with everything we do musically. It's a waste of time and resources to have to load them to RAM from tape (or worse yet, manually) every time they're needed. A PROM is their happiest home. In our 8700 Computer/Controller, MUS 1 is a 1702A PROM that occupies the address range $D00-$DFF (IC-17).

Examples? OK, the keyboard reading routine (LOOK). It isn't particularly long or complicated (a little over 30 bytes) but we're going to need it every time we turn on the system- even if it isn't used to read the keyboard, it's the thing that our protocols dictate will be the tempo-determining element in the system (based on the clock rate of the encoder). At some future date the occasion may arise when we can examine this in detail. Today, it's not the point.

The QuAsh drivers (called NOTE)- same thing- we're going to need them for almost everything we do. Why bother to load them?

In addition to these two routines, MUS1 also contains:

INIT: an initialization routine that takes care of setting various variables and buffer areas to a known, acceptable state (as opposed to the random numbers they will contain when power is first applied.)

POLY: essentially the polyphonic (I still prefer polytonic) allocation algorhythm from POLY1.0, except refined somewhat to take less memory space.

TRGN: The new miracle ingredient- Software Transient Generators (STG). A routine that will serve as a software substitute for ADSRs.

OPTN: A very simple option selecting program that allows the remaining firmware of MUS1 to be tied together into a 16 voice polyhonic synthesizer with or without software transient generators- without having to lead any additional software (though several parameters will need to be initiallized manually).

All of this is pretty straight-ahead code that should be understandable from the documented listing that appears at the end of this article- you may need to refer back to previous articles in this series for background information; "In Pursuit of the Wild QuAsh" (reference Polyphony, July '77) and "What the Computer Does" (reference Polyphony 4/76) would be particularly useful ones.

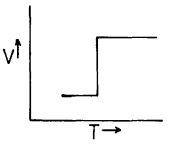
Two exceptions, NOTE and TRGN, need some additional explanation - they introduce some new ideas.

In an embryonic form, NOTE was a part of POLY1.0. It is the responsibility of this routine to take individual entries from the output buffer area (NTBL), add to it the corresponding entry from the Transpose buffer area (TTBL)and output the results to the QuAsh channels. Some aspects of the significance of the addition that takes place will be seen when we look at TRGN- for now, it will suffice to say that this will be an extraordinarily handy convention in a number of cases.

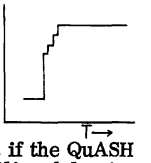
A more important function of NOTE is to make sure that what comes out of the QuAsh channels has no annoying glitches that may be artifacts of the D/A and multiplexing process. In an earlier story, we looked at one of the annoyances - the fact that our 8780 D/A, though quick, takes a finite amount of time to change from one value to the next and if appropriate settling time is not allowed between writes to the QuAsh channels we will be able to hear the changes as a slight "buzz" in each of the channels. The solution here is to output the data first to a "dummy channel" that is occupied solely by the D/A, with no corresponding QuAsh, followed immediately by a write of the identical information to an output which does correspond to a QuAsh channel. The first write allows the D/A to settle while the second strobes the settled output into the appropiate QuAsh channel.

And here we come face to face with the next problem; the QuAsh really need some settling time since they are at their heart nothing more than an RC circuit.

As long as we are thinking in terms of small systems(8 output channels or less) this is not a big problem since it can be dealt with simply by delaying after writing to one QuAsh but before setting up the next. If the delay is not long enough, we will hear changes from one value to the next not as an instantaneous change, but rather as a series of steps from the initial value to the final one:



We want this figure a



But if the QuASH settling delay is not long enough, will get this.

In larger systems, this constant delay approach is not a practical solution because there is not enough time during alternate "dummy" scans of the keyboard (the time which our conventions allow for processing, output driving, allocations, etc.) to allow all of the output channels the luxury of a delay. The time comes for the keyboard to be read again (or other things to happen) and the processor is still busy waiting for all of those QuAsh to get to the right value.

The key to the solution of this problem is to notice that there is really only one set of circumstances under which the long QuAsh-settling delay is required, and that's when the output of one of these channels must change from one value to another (which happens only a small percentage of the time) and then, only when the glide of the channel is turned off. (if

the glide is on, its integrating action will smooth out the steps; and, in fact, a short write time is preferable here since it will serve to increase the time required for the glide.)

The actual solution is what I feel we should call "DYNAMIC QuAsh DRIVERS" -a small block of programming, more or less in the middle of NOTE.

This part of the program first checks to see if the glide control bit (the most significant bit of the data just written to D/A and S/Hs) was turned on or not. If we are in "glide mode," no delay is required so the program immediately goes to see if there are any channels left to write; if there are, it services them.

If the glide is not on, we have a candidate for dynamic operation so the dynamic mode switch is checked (more later) and if this option is selected the current data is compared to the data that was previously written to this channel (requires a new table that we've generated called "LAST") and if they're different (a change), the program goes into the delay that allows the output of the QuAsh to instantaneously (apparently) step from its previous value to the next one. The new value is saved in LAST (for use next time) and if there are more channels to do- it does them.

SOFTWARE TRANSIENT GENERATORS

Here we begin, for the first time, to replace some of the elements that constitute traditional synthesizer hardware with software that performs the same function (hopefully as well, or better) with less costly hardware. STGs are a good place to start because they're not super difficult to implement.

Just like their hardware equivalents, STGs respond to a note which has just been triggered (pressed on the keyboard) by producing a voltage that rises at a controlled Attack rate. After reaching some peak value, the voltage then drops at a Decay rate until it reaches a pre-set Sustain level where it stays as long as the note remains triggered. When the key is released, the voltage drops to its lowest level at the Release rate.

Computing the number which represents the current value of the transient is only slightly more complicated than adding, subtracting and comparing.

Unlike an ADSR, an STG has no knobs to set, in their place you enter numbers setting Attack rate, etc. into the computer.

Perhaps the biggest problem having to do with STGs is deciding where they should come out. Oh, the QuAsh channels, obviously; but which ones? Of the numerous possibilities, we've selected the convention of having pitch setting voltages (those that correspond to notes) and transient voltages come from alternate QuAsh channels, primarily because this will work nicely with some stuff under development (or consideration, at least), without making obsolete all of the hardware that we've accumulated up to now.

This implies two distinct modes of operation; the first in which the STGs are not asserted and POLY assigns notes to sequential QuAsh channels; and, the second mode (STGs on), in which notes are assigned by POLY* to the odd number QuAsh channels (first, third, etc.) while transients are produced at the even number outputs (second, fourth, etc.).

The note produced at the first QuAsh output has a corresponding transient happening at the second output, and so on. Just as if the trigger from the first channel were patched to the input of an ADSR whose output was somehow tied to the output of the second QuAsh channel.

This would seem a good place to mention (in case it's not already obvious) that in this implementation all of the STGs produce the same kind of transient, and for the kinds of things that we're doing now, this is how it should be. It may also be worth mentioning that while the transients are all the same, they are totally independent where following the triggered and released states of their respective note channels is concerned.

There are also some internal details which muddy the STG waters. For instance, a key that is currently down may require a transient function that is either in the Attack cycle (increasing) or Decay/Sustain cycle (decreasing or holding) depending on its past history (had it already peaked?). Somewhere we need to save information on which cycle the transient is actually in.

Another, somewhat interrelated, problem concerns the smoothing of the transient waveform. Under most conditions, the glide of the QuAsh channels that are being used as transient outputs should be turned on so that a smoothly increasing or decreasing function is produced. But, the glide can't always be on because that would limit the maximum attack rate.

Without having the space to cover it entirely, I can only state that the solution to both of these difficulties lies in the use of the Transpose table and remembering that the data stored in TTBL entries is added to the output parameter in NTBL (where we're storing the actual current value of the transient) before the output operation takes place. Note also that while the data in NTBL is manipulated extensively by POLY and TRGN (as they calculate, allocate, - regurgitate?) TTBL is untouched by computer hands, and this makes it an ideal place to save control type functions. Not only transpositions, but a place that glide and trigger bits and such can be permanently set.

These locations are so handy for this application that in TRGN they have been re-named CWRD (Control-Words...but do not be confused, this is still our old friend TTBL and has no relationship at all to the System Control Word-CTRL) and it is here that we keep track of the A/D/S state of each of the transient channels.

Also, to help me keep things straight in my own mind, the NTBL bytes that are used to store the current value of the transient have been re-named PARM (parameter); but, again, this is the same physical area as NTBL.

* Note that POLY checks to see if the STGs are turned on as it assigns notes to outputs.

NOW, HOW DO WE USE ALL THIS?

Perhaps the best way to begin an essay on how to use MUS1 is to state one of the functions that it was devised to perform

As you are no doubt beginning to realize, we've carefully developed a system that will have applications far beyond what we've discussed to this point. It's complex; and while the complexity implies unmatched versatility, it undeniably has its intimidating aspects.

At one level of use, MUS1 should reduce this intimidation by giving the user an instrument with a specific (though within certain limits alterable) personality the instant that it's turned on, without having to hassle around with loading any additional programs (success) or variables (well...)

Also, these program modules should be written so that they easily interface with future expansions of the system, either hardware or software, so that, when needed, they can be accessed by programs offering distinctly different personalities (success here maybe- only time will really tell).

While we've reduced the intimidation, we've not eliminated it entirely because even when using MUS1 as a stand-alone personality there are some variables which must be initialized before you begin to play- some information that the system must have in order to operate properly. This data could be part of the PROM, but not without significantly compromising versatility.

For instance, we've mentioned in passing a couple of times the System Control Word-CTRL. This is a single word in the com-

puter's RAM memory at location $0E8.

It is most helpful to visualize CTRL as a collection of eight "switches", each bit representing one switch. To MUS1, only two of these switches have any significance- D7, which turns the STGs on and off, and D6, which enables or disables the dynamic mode option. The rest are reserved.

Every time you power up the system, CTRL must be set so that the desired options are selected- there is no default setting that is part of MUS1. If you want dynamic mode (which you should, for now) then bit 7 should be turned on. If you want STGs, bit 8 must be set.

The 4 possible combinations of these 2 bits then have the following significance:

binary	hex	action
00000000	$00	STGs off; dynamic mode off
01000000	$40	STGs off; dynamic mode on
10000000	$80	STGs on; dynamic mode off
11000000	$C0	STGs on; dynamic mode on

CTRL is not the only variable which must be initialized manually. There's also:

```
EXTERNALLY INITIALIZED VARIABLES
================================
 LOC. LABEL          USE
--------------------------------
 0E8  CTRL  SYSTEM CONTROL WORD
            D7 SET TURNS ON
            TRANSIENT GENERATORS
            D6 SETS DYNAMIC MODE

 0E9  ODLY  SETS OUTPUT DELAY;
            IN DYNAMIC MODE $20
            RECOMMENDED

 0EA  OUTS  NUMBER OF HARDWARE
            SUPPORTED CONTROL
            CHANNELS AVAILABLE

 -- AND TRANSIENT PARAMETERS --

 0BA  ATCK  ATTACK RATE
 0BB  DCY   DECAY RATE
 0BC  SUST  SUSTAIN LEVEL
 0BD  RELS  RELEASE RATE
 0BE  PEAK  PEAK VALUE -SEE TEXT

            RATES: $01 (SLOW)
                   $3F (FAST)
            LEVEL: $01 (MINIMUM)
                   $3F (MAXIMUM)
--------------------------------
```

Most of these are easily understood or have been examined in the past, so we won't go into any great detail. A few points are worth mentioning, however.

ODLY- this is a number that represents the delay that the QuAsh drivers will use, when required. For normal use, a value in the range of $20-$30 is most appropriate.

OUTS- this variable tells the POLY subroutine how many output channels it has to work with, so that notes don't get lost; we talked about this last time. Now we need to notice that when the STGs are asserted we should think of the QuAsh channel that is producing the transient as simply an extension of the channel producing the note. In other words, the two QuAsh channels constitute a single "hardware supported" channel. A single QuAsh represents two such channels.

ATCK/DCY/SUST/RELS- When the transient generators are turned on, we also need to enter the attack, delay, sustain and release parameters that we want produced. These four entries should need little explanation other than the examples which follow shortly; their range is from $01-$3F, with $01 representing the lowest rate or level and $3F the highest.

PEAK- this fifth transient parameter needs a little extra attention. PEAK has only one use; it determines whether the transient produced is going to be percussive (quickest possible attack and full ADSR segments) or non-percussive. In the non-percussive mode, the glide is on for all segments of the transient and the Decay and Sustain states of the transient are eliminated entirely.

In fact, there is only one bit in the word PEAK that is changed to select one of these two options- the most significant bit. The remaining seven bits should (for now- until you have a real feel for what's happening) be set to $3F (00111111 in binary). If the most significant bit of this word is cleared, you're in percussive mode. If the bit is set (so that PEAK contains $BF - 10111111 in binary) you are in non-percussive mode.

The differences between the two are great. Assume for a moment that we have set the ADSR parameters at $3F/$04/$20 /$01 respectively (fastest attack/moderated decay/medium sustain/slowest release) and that we are only going to change the PEAK parameter. If PEAK contains $3F(percussive mode), a 'scope display of the transient will look something like this:

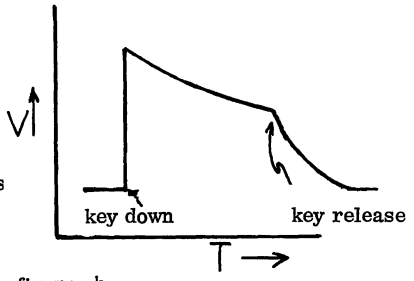


figure b

Setting PEAK to $BF (non-percussive) produces this result:

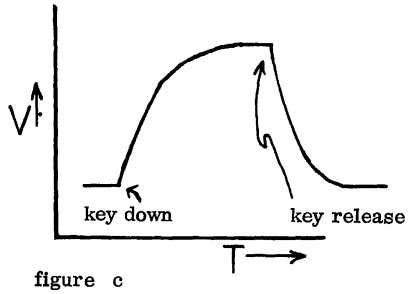


figure c

Because the glide is now on during the entire attack cycle and the Decay and Sustain portion of the transients are eliminated. Straightforward stuff, really.

We need to cover an example of system set-up before we wind up, but first must notice that the effect of having the PEAK parameter are far more far-reaching than we've been able to cover in detail. A quick example:

ADSR parameters set to $10/$04/$20 /$01 and PEAK containing $3F will produce this kind of transient:

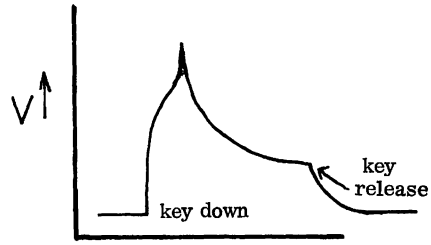


figure d

which, when heard, starts out with a non-percussive kind of "swell" with a percussive "pip" added at the last instant before the transition to the Decay and Sustain cycles. This would seem to be a unique and useful transient that isn't produced by traditional ADSRs.

Along the same lines, the TSGs can be considered to be "better" than our hardware ADSRs in that they need not finish the Attack cycle before transitioning to the Release state. If a key is released before its transient has gone all the way to PEAK, the transient immediately switches to the release state. This is frequently called "muting' and it offers the possibility of effective control of espression directly from the AGO keyboard.

A SUMMARY, OF SORTS

So, we've gotten our hands on a MUS1 PROM and are ready to start doing things. What has to be done first? Really very little.

First, the System Control Word, Output Delay and number of hardware channels available must be set. For example:

keystrokes	explanation
0-E-8-DISP	sets monitor pointer to $E8-CTRL
C-0-ENT	sets $E8-asserts STGs dynamic mode
3-0-ENT	sets ODLY value
0-2-ENT	sets output channels at 2

these entries define the personality of the instrument as a 2 voice polyphonic synthesizer (notes from channels A & C) with software transient generators (which appear at QuAsh channels B & D) .

Next, we must set the transient parameters to the desired values:

keystrokes	explanation
0-B-A-DISP	sets monitor pointer to $BA- ATCK
3-F-ENT	sets shortest attack
0-4-ENT	sets moderate decay
2-0-ENT	sets moderate sustain
0-1-ENT	sets slowest release
3-F-ENT	percussive mode

and you may recognize these parameters as being those that we examined in the illustration earlier.

Finally, we simply begin running the program:

keystrokes	explanation
D-0-0-DISP	sets monitor pointer to beginning of OPTN
RUN	presto- the program runs

A typical patching configuration that would be consistent with these entries would look something like this:

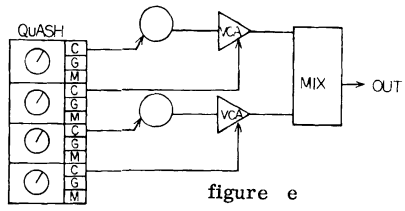


figure e

Oh, yes- I almost forgot. OPTN, like POLY 1.0, uses the 8700 keyboard to control two important functions. While OPTN is running, touching key 0 of the control keyboard will cause the entire system to be re-initialized. Not the entries that we made manually- those remain unchanged, but all the notes and transients go immediately to zero level.

Similarly, touching key #1 produces a tuning function that makes the synthesizer respond as if all the channels were seeing the second C on a three octave keyboard held down. The transients go, the notes play, etc. After tuning, be sure to re-initialize the system by touching control key #0.

I prefaced one of the earlier paragraphs with "at one level of use." In all of the preceding words, that's all that we've examined- one level of use (the simplest and most obvious level, at that.). I've also referred in the past to "software modules" which can be strung together in different ways (just as can hardware modules) to produce different effects and personalities. MUS1 is the first set of these modules.

With more regret than you can imagine, I haven't the space here to go into all of the implications of this (even if I knew them all, which I'm sure I don't).

Providing you're more than just casually interested, you should spend some time trying to understand how MUS1 works internally (there are numerous different entry points to the routines that we haven't covered - for instance). I believe that the time investment will be wisely made.

```
                        ;**********************************
                        ;*                                *
                        ;*             MUS1               *
                        ;*                                *
                        ;*       BY JOHN SIMONTON         *
                        ;*(C) 1978 PHIA ELECTRONICS, INC*
                        ;*      ALL RIGHTS RESERVED       *
                        ;*                                *
                        ;*                                *
                        ;*   SYNTHESIZER SUBROUTINES      *
                        ;*         ++ AND ++              *
                        ;*  MULTIPLE OPTION POLYPHONIC    *
                        ;*   ALLOCATION PROGRAM WITH      *
                        ;*     SOFTWARE TRANSIENT         *
                        ;*         GENERATION             *
                        ;*                                *
                        ;**********************************
                        ;
                        ;          OPTN
                        ;POLYPHONIC SYNTHESIZER / OPTION
                        ;       SELECTION
                        ;**********************************
                        ;
                        ;
                        INIT .DL 0D21
                        POLY .DL 0D71
                        TRGN .DL 0DC3
                        NOTE .DL 0D2B
                        DECD .DL 0F00
                        FILL .DL 0D52
                        DISP .DL 0820
                        CLCK .DL 00BF
                        ;
                        ;OPTION TIES MUS1 FIRMWARE
                        ;TOGETHER INTO A POLYPHONIC SYNTH
                        ;WITH OR WITHOUT TRANSIENT GENER-
                        ;TION; W/WO DYNAMIC QUASH DRIVERS
                        ;
                        ;ALSO USES PIEBUG DECODE AND
                        ;ASSIGNS KEY #0 AS SYSTEM CLEAR
                        ;AND #1 AS TUNE - EQUIVALENT TO
                        ;ALL CHANNELS 2ND "C" ON KBD DOWN
                        ;
0D00-   20 21 0D  OPTN  JSR INIT     ;ZERO ALL BUFFS
0D03-   20 71 0D  LOOP  JSR POLY     ;ALLOCATE CHANS
0D06-   20 C3 0D        JSR TRGN     ;NEW TRANSIENTS
0D09-   20 2B 0D        JSR NOTE     ;OUTPUT-READ AGO
0D0C-   A5 BF           LDA *CLCK    ;GET CLOCK VALUE
0D0E-   8D 20 08        STA DISP     ;RAZZ-MA-TAZZ
0D11-   20 00 0F        JSR DECD     ;CHECK COMMANDS
0D14-   C9 01           CMP 01       ;0?  1?  >1?
0D16-   30 E8           BMI OPTN     ;0; CLEAR ALL
0D18-   D0 E9           BNE LOOP     ;>1; KEEP ON
0D1A-   A0 5C           LDY 5C       ;1; TUNE 2ND C
0D1C-   20 52 0D        JSR FILL     ;KEYS ALL DOWN
0D1F-   F0 E2           BEQ LOOP     ;BRANCH ALWAYS
                        ;
                        ;          INIT
                        ;  INITIALIZATION ROUTINE
                        ;**********************************
                        ;
                        CTRL .DL 00E8
                        TBEG .DL 00BF
                        ;
                        ;INIT CLEARS INPUT BUFFER (KTBL)
                        ;OUTPUT BUFFER (NTBL) AND TRANS-
                        ;POSE BUFFER/CONTROL WORDS (TTBL)
                        ;HEXADECIMAL MODE IS SELECTED
                        ;
                        ;ENTER AT INT0 TO FILL TABLES
                        ;WITH CHARACTER FROM ACCUMULATOR
                        ;
0D21-   A9 00     INIT  LDA 00         ;PREPARE TO ZERO
0D23-   A2 28     INT0  LDX 28         ;SET POINT/COUNT
0D25-   D8              CLD          ;SET HEX MODE
0D26-   95 BF     INT1  STA *TBEG,X    ;ZERO BUFFER
0D28-   CA              DEX          ;POINT TO NEXT
0D29-   D0 FB           BNE INT1     ;SOME LEFT -LOOP
                        ;
                        ;       NOTEOUT/LOOK
                        ;16 CHANNEL QUASH DRIVERS AND AGO
                        ;   KEYBOARD READING ROUTINE
                        ;
                        ;**********************************
                        ;
                        CTRL .DL 00E8
                        ODLY .DL 00E9
                        KTBL .DL 00DF
                        NTBL .DL 00CF
                        TTBL .DL 00BF   ;ALSO CLCK
                        LAST .DL 00A9
                        S/H  .DL 09EF
                        D/A  .DL 0900
                        KBD  .DL 0810
                        ;
                        ;       *** NOTEOUT ***
                        ;    DYNAMIC QUASH DRIVERS
                        ;GETS NOTES TO BE PLAYED FROM THE
                        ;OUTPUT BUFFER (NTBL) AND ADDS
                        ;TRANSPOSING VALUE FROM TRANSPOSE
                        ;BUFFER (TTBL).  OUTPUTS RESULT
                        ;
0D2B-   A2 10     NOTE  LDX 10         ;SET POINTER
0D2D-   B5 CF     NO0   LDA *NTBL,X    ;GET NOTE
0D2F-   18              CLC          ;PREPARE AND
0D30-   75 BF           ADC *TTBL,X  ;ADD TRANSPOSE
0D32-   8D 00 09        STA D/A      ;LET D/A SETTLE
0D35-   9D EF 09        STA S/H ,X   ;WRITE TO S/H
                        ;
                        ;NOW THE DYNAMIC PART; IF GLIDE
                        ;IS ON, DELAY IS SKIPPED.  IF NOTE
                        ;IS SAME AS LAST PLAYED (IGNORING
                        ;CONTROL BITS D6 & D7) DELAY IS
                        ;SKIPPED.   IF NOT IN DYNAMIC MODE
                        ;AND NO GLIDE, DELAY ALWAYS TAKEN
                        ;
0D38-   30 0F           BMI NO2      ;GLIDE? NO DELAY
0D3A-   09 80           ORA 80       ;IGNORE FLAGS
0D3C-   24 E8           BIT *CTRL    ;DYNAMIC MODE ?
0D3E-   50 04           BVC DLAY     ;NO, JMP TO DELAY
0D40-   D5 A9           CMP *LAST,X  ;COMPARE TO LAST
0D42-   F0 05           BEQ NO2      ;SAME:SKIP DELAY
0D44-   A4 E9     DLAY  LDY *ODLY    ;GET DELAY VALUE
0D46-   88        NO1   DEY          ;DECREMENT DELAY
0D47-   D0 FD           BNE NO1      ;LOOP TIL DONE
0D49-   95 A9     NO2   STA *LAST,X  ;FOR NEXT TIME
```

```
0D4B-   CA            DEX          :POINT TO NEXT
0D4C-   D0 DF         BNE NO0      :SOME LEFT -LOOP
                      :
                      :LOOK WAITS FOR THE BEGINNING OF
                      :AN "ACTIVE" SCAN-BEGINS PUTTING
                      :THE NUMBERS OF KEYS DOWN IN SE-
                      :QUENTIAL IN-BUFF WORDS.  WHEN
                      :SCAN DONE REMAINING IN-BUFF IS
                      :ZERO'D.
                      :
0D4E-   E6 BF     LOOK INC *TTBL   :INCREMENT CLOCK
0D50-   A0 00         LDY 00       :PREPARE FOR CLR
0D52-   A2 08     FILL LDX 08      :SET UP POINTER
0D54-   AD 10 08  LK2  LDA KBD     :WAIT FOR
0D57-   30 FB         BMI LK2      :"ACTIVE" SCAN
0D59-   AD 10 08  LK3  LDA KBD     :GET KEY
0D5C-   30 0F         BMI DONE     :END SCAN? -CLR
0D5E-   2A            ROL          :STROBE TO D7
0D5F-   10 F8         BPL LK3      :D7=0,NO STROBE
0D61-   6A            ROR          :RESTORE DATA
0D62-   95 DF         STA *KTBL,X  :TO IN BUFFER
0D64-   CD 10 08  LK4  CMP KBD     :NOW WAIT FOR
0D67-   F0 FB         BEQ LK4      :NEXT KEY
0D69-   CA        LK0  DEX          :PNT TO NEXT BUF
0D6A-   D0 ED         BNE LK3      :SOME LEFT -LOOP
0D6C-   60        RTN  RTS          :LEAVE
0D6D-   94 DF     DONE STY *KTBL,X :ZERO IN-BUFFER
0D6F-   30 F8         BMI LK0      :BRANCH ALWAYS
                      :
                      :          POLY
                      : A LIMITED RESOURCE ALLOCATION
                      :         ALGORHYTHM
                      :**********************************
                      :
                      OUTT .DL 00EB
                      OUTS .DL 00EA
                      CTRL .DL 00E8
                      KTBL .DL 00DF
                      NTBL .DL 00CF
                      :
                      :POLY-FIRST HALF OF ALGORHYTHM
                      :IN THIS BLOCK DE-ACTIVATED CHANS
                      :ARE REACTIVATED IF THE DATA THEY
                      :CONTAIN APPEARS IN THE IN BUFFER
                      :
                      :D7 IN CTRL SET - ALTERNATE MODE
                      :D7 "   "   CLR - SEQUENTIAL MODE
                      :
0D71-   A5 EA     POLY LDA *OUTS   :# OF OUT CHANS
0D73-   85 EB         STA *OUTT    :USE AS COUNTER
0D75-   A2 10         LDX 10       :PREPARE PNT/CNT
0D77-   B5 CF     POL0 LDA *NTBL,X :GET NOTE
0D79-   F0 27         BEQ NWKY     :0-OLD KEYS DONE
0D7B-   29 7F         AND 7F       :CLEAR D7
0D7D-   09 40         ORA 40       :SET D6
0D7F-   A0 09         LDY 09       :PREPARE PNT/CNT
0D81-   88        LP0  DEY          :POINT NEXT KEY
0D82-   F0 12         BEQ NEXT     :DONE -NEXT NOTE
0D84-   D9 DF 00      CMP KTBL,Y   :SAME AS KEY?
0D87-   D0 F8         BNE LP0      :NO -NEXT KEY
0D89-   95 CF         STA *NTBL,X  :SAVE NOTE D6=1
0D8B-   C6 EB         DEC *OUTT    :ONE LESS OUTPUT
0D8D-   F0 33         BEQ OUT      :NONE LEFT-LEAVE
0D8F-   A9 00         LDA 00       :OR PREPARE AND
0D91-   99 DF 00      STA KTBL,Y   :ELIMINATE KEY
0D94-   F0 04         BEQ LP1      :& BRANCH ALWAYS
0D96-   29 BF     NEXT AND 0BF     :CLEAR TRIG (D6)
0D98-   95 CF         STA *NTBL,X  :& RESTORE NOTE
0D9A-   24 E8     LP1  BIT *CTRL   :ALTERNTE MODE?
0D9C-   10 01         BPL SKP1     :NO -DEC. ONCE
0D9E-   CA            DEX          :YES-DEC. TWICE
0D9F-   CA        SKP1 DEX         :POINT NEXT NOTE
0DA0-   D0 D5         BNE POL0     :SOME LEFT -LOOP
                      :
                      :NEWKEY - SECOND HALF.  KEYS DOWN
                      :ARE ASSIGNED TO OUTPUT BUFFER
                      :LOCATIONS WHICH ARE STILL DE-
                      :ACTIVATED
                      :
0DA2-   A2 10     NWKY LDX 10      :NTABLE PNT/CNT
0DA4-   A0 09         LDY 09       :KTABLE PNT/CNT
0DA6-   A9 40     NK1  LDA 40      :PREPARE MASK
0DA8-   35 CF         AND *NTBL,X  :NOTE TRIGGERED?
0DAA-   D0 0E         BNE NK3      :YES -GO TO NEXT
0DAC-   88        NK2  DEY          :POINT NEXT KEY
0DAD-   F0 13         BEQ OUT      :NONE LEFT-LEAVE
0DAF-   B9 DF 00      LDA KTBL,Y   :KEY NEEDS HOME?
0DB2-   F0 F8         BEQ NK2      :NO -GET NEXT
0DB4-   95 CF         STA *NTBL,X  :YES-PUT IN NOTE
0DB6-   C6 EB         DEC *OUTT    :ONE LESS OUTPUT
0DB8-   F0 08         BEQ OUT      :NONE LEFT-LEAVE
0DBA-   24 E8     NK3  BIT *CTRL   :ALTERNATE MODE?
0DBC-   10 01         BPL SKP2     :NO -DEC ONCE
0DBE-   CA            DEX          :YES-DEC TWICE
0DBF-   CA        SKP2 DEX         :POINT NEXT NOTE
0DC0-   D0 E4         BNE NK1      :SOME LEFT -LOOP
0DC2-   60        OUT  RTS         :RETURN
                      :
                      :          TRGN
                      :TRANSIENT GENERATOR PROGRAM
                      :
                      :*****************************
                      :
                      CTRL .DL 00E8
                      ATCK .DL 00BA
                      DCY  .DL 00BB
                      SUST .DL 00BC
                      RLS  .DL 00BD
                      PEAK .DL 00BE
                      NTBL .DL 00CF
                      PARM .DL 00CE
                      TTBL .DL 00BF
                      CWRD .DL 00BE
                      :
                      :       NTBL 00D0-00DF
                      :       TTBL 00C0-00CF
                      :
0DC3-   A5 E8     TRGN LDA *CTRL   :DO TRANSIENTS?
0DC5-   10 38         BPL RTN1     :NO  -RETURN
0DC7-   A2 10         LDX 10       :NTABLE PNT/CNT
                      :
                      :   A/D/S/R DETERMINATION
                      :ROUTINE PREPARES Y TO USE AS
                      :CONTROL WORD, GETS NOTE AND
                      :SHIFTS TRIG. TO CARRY, GETS
                      :CURRENT STATE (CS) PARAMETER.
                      :IF NOTE TRIG. NOT SET STATE IS
                      :RELEASE.  IF CS PARA IS POSI-
                      :TIVE STATE IS DECAY/SUSTAIN
                      :OTHERWISE, STATE IS ATTACK
                      :
0DC9-   A0 40     ADSR LDY 40      :PREPARE CWRD
0DCB-   B5 CF         LDA *NTBL,X  :GET NOTE AND
0DCD-   2A            ROL          :ROTATE TRIGGER
0DCE-   2A            ROL          :TO CARRY BIT
0DCF-   B5 CE         LDA *PARM,X  :GET CS PARA.
0DD1-   90 19         BCC RELS     :NO TRIG? -RLS
0DD3-   10 0B         BPL DS       :CS>0? -DECAY/S
                      :
                      :    ATTACK ROUTINE
                      :ADDS ATTACK PARAMETER TO CS PARA
                      :AND IF GREATER THAN PEAK
                      :SUBSTITUTES $3F AND SETS CONTROL
                      :WORD TO $40 (D6 SET - NO GLIDE).
                      :NOTE THAT CS PARA WILL BE >0
                      :WHEN NEXT CHECKED.
                      :
0DD5-   18        ATTK CLC          :PREPARE
0DD6-   65 BA         ADC *ATCK    :ADD ATTACK PARA.
0DD8-   C9 BF         CMP 0BF      : >PEAK
0DDA-   90 1B         BCC NEXT     :NO -PLACE PARA.
0DDC-   A5 BE         LDA *PEAK    :YES-PEAK VALUE
0DDE-   D0 17         BNE NEXT     :BRANCH ALWAYS
                      :
                      :    DECAY AND SUSTAIN ROUTINE
                      :NOTE THAT CARRY IS SET.  DECAY
                      :PARAMETER IS SUBTRACTED FROM
                      :CURRENT STATE PARAMETER.  IF
                      :RESULT IS LESS THAN SUSTAIN
                      :PARAMETER THEN SUST. PARA.
                      :BECOMES CURRENT STATE PARA.
                      :D6 & D7 OF CONTROL WORD SET
                      :
0DE0-   A0 C0     DS   LDY 0C0      :PREPARE CWRD
0DE2-   E5 BB         SBC *DCY     :SUBTRACT DCY
0DE4-   C5 BC         CMP *SUST    :>SUSTAIN?
0DE6-   10 0F         BPL NEXT     :PLACE PARA
0DE8-   A5 BC         LDA *SUST    :CS PARA=SUST
0DEA-   10 0B         BPL NEXT     :PLACE PARA
                      :
                      :    RELEASE ROUTINE
                      :MAKES SURE THAT CURRENT STATE
                      :GLIDE BIT IS SET (NOTE-MAKES
                      :CS NEGATIVE).  SUBTRACTS RELEASE
                      :PARA. FROM CURRENT STATE.  IF
                      :RESULT >0, MAKES CS & CWRD =80
                      :
0DEC-   38        RELS SEC          :PREPARE
0DED-   09 80         ORA 80       :SET CS GLIDE
0DEF-   E5 BD         SBC *RLS     :SUBTRACT RLS
0DF1-   30 04         BMI NEXT     :CS<0 -PLACE CS
0DF3-   A0 00         LDY 00       :CS>0 -DONE MAKE
0DF5-   A9 80         LDA 80       :CS=80,CWRD=0
                      :
                      :    NEXT
                      :PLACES CS PARA AND CWRD IN
                      :PROPER CONTROL CHANNEL OUTPUTS
                      :DECREMENTS POINTER (TWICE) AND
                      :IF NOT YET DONE LOOPS FOR MORE
                      :
0DF7-   94 BE     NEXT STY *CWRD,X :PLACE CONTROL
0DF9-   95 CE         STA *PARM,X  :PLACE CS PARA
0DFB-   CA            DEX          :DECREMENT POINT
0DFC-   CA            DEX          :AND AGAIN
0DFD-   D0 CA         BNE ADSR     :SOME LEFT -LOOP
0DFF-   60        RTN1 RTS          :RETURN
```

AFTERTHOUGHTS .....

It has been pointed out that some perhaps pertinent details have been omitted from the preceeding explanation of MUS 1.

The most prominent example is "why would you ever want to not have dynamic mode". The most probable reason is for special effects.

In general, the difference between special effects and noise is imagination. Contemporary musical lore is full of instances where a special effect resulted from an unsuccessful attempt to do something entirely different. Phil Spector' s original "flanging" effect, so popular today, was supposed to be voice doubling, but didn' t work.

In this same manner, there will be those who will be able to use the "step glissando" that results from too short a QuASH settling delay as a valid musical device.

Also, the dynamic mode requires an additional 16 byte table area that might easily be put to better use in some programs.

This same philosophy of maximizing versatility is responsible for the QuASH settling delay being an externally initialized variable. For the purpose of effect, there may be times when you want a short delay.

In addition to this, we have seen systems which were marginal in their power supply complement which would have a discernible pitch "blip" when keys were pressed with long delays (in the $30 - $40 range) - caused by the relatively heavy charging current producing a momentary dip in supply voltage. In these systems, a short term solution has been to decrease the QuASH driver delay to something on the order of $10. The long term solution is more power.

SEVERAL POINTS RELATIVE TO THE OPERATION OF THE STGs SHOULD BE MENTIONED.

QuASH GLIDE CONTROLS. The setting of the QuASH glide pots have an effect on the transients produced. In most cases, these controls will need to be advanced only slightly from their fully counterclockwise "off" position.

The most noticeable effect of different settings of the glides will be observed when the STGs are set to the percussive mode by PEAK.

When the most significant bit of PEAK is cleared, it will effect only on the last increment of the attack cycle. For all increments other than the last, the glide will be set. A detailed example will best illustrate this.

Assume that we have set the STG parameters as follows:

ATCK - 08
DCY - 04
SUST - 20
RELS - 04
PEAK - 3F

or in our more or less standard notation, $08/$04/$20/$04/$3F.

If we were able to disable glide entirely, and then scope'd the transient we'd see this:

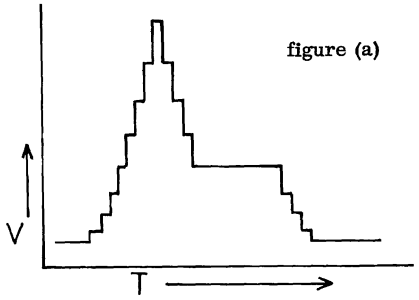


figure (a)

as the STG program counted up to the peak and then down to the sustain level before counting down to the base level when the key was released.

If we then enabled the glide and set them to a slightly advanced position and examined the same output we would find that this change had taken place.

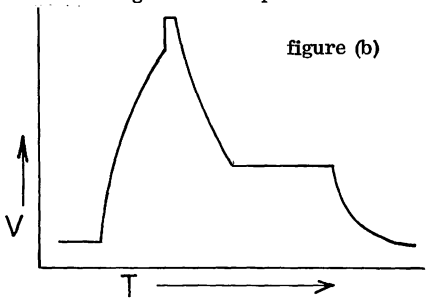


figure (b)

The integrating action of the glide circuitry has smoothed the steps of the Attack, Decay and Release, with the exception of the last Attack step where (as we have already stated) the glide is off under all percussive circumstances. In this specific case, the last glide-less increment will be hardly noticeable.

If, on the other hand, the glide is set to a long value (fully advanced, for instance) an examination of the waveform will show this:

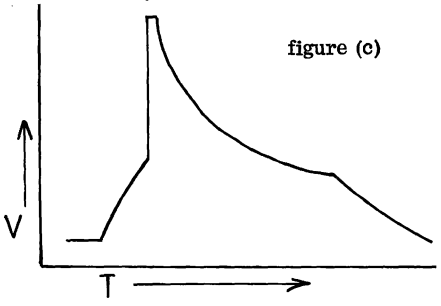


figure (c)

The heavy glide has slowed the waveform to the point that when the glide-less final increment comes it takes a much greater step (one that is completely noticeable, and unique). The heavy glide also has the effect of slowing the decay and release rates as shown.

It would also be appropriate to mention at this point that the instantaneous steps produced by the STG/QuASH combination is much faster than the maximum attack rate available from a 4740, or in fact from most ADSRs. Whereas a typical ADSR may have a minimum attack time of more or less 5 milliseconds, the QuASH in dynamic mode can step in a fraction of a millisecond.

This means that if there are any tendencies on the part of the VCA being used to have interaction between control and signal channels it will be aggravated when using STGs. We may hear "pops" and "thumps" that were not objectionable before. Probably the best solution here is to limit the response of the control voltage inputs of the VCA. In a 4710 Balanced Modulator, this means the addition of a small capacitor. Like this:

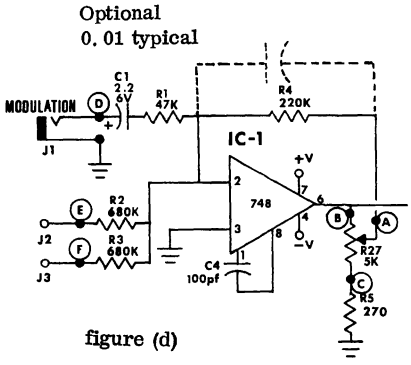


figure (d)

Another point to be considered is the fact that the output voltage from a QuASH channel doesn't go to zero - there may be some leakage from the VCA when it is supposed to be off. The easiest fix here is to re-adjust the Modulation rejection control of the VCA being used. In the 4710 this is R25. Be aware that this also limits somewhat the useable range of the D/A's TUNE control since wide variations in the setting of this control will affect the leakage from the VCA. Tuning changes on the order of 1/4 octave should not present any particular difficulties.

There is a point dealing with this which may not be immediately exploitable by many, but which should be mentioned in any case; the action of the trigger outputs of the QuASH channels which are being used as STG channels.

The trigger outputs QuASH channels which are being used to produce pitch setting voltages behave in the normal manner. When the AGO key

corresponding to that QuASH channel is being held down, the trigger is at a high state. When the key is released, the trigger goes low. A standard "gate" type response.

In a similar manner, the triggers of transient channels also go high as soon as the AGO key to which they correspond is pressed; but unlike the normal trigger, this level remains high until the software has completed the last increment of the Release cycle. In future hardware this will drive a "noise gate", a simple semi-conductor switch which completely quiets a channel that is inactive.