Thoughts on Software for the Systems Concepts Digital Synthesizer
Peter Samson — Feb. 11, 1976

SYSTEM DESIDERATA

- Reliability
- Good programming practices:
    - Software modularity (with respect to devices supported)
    - Well-commented code
    - Comprehensive documentation for User and Maintainer
- Minimize time around cycle of use:
    - Edit source file;
    - Compile;
    - Save compiled output and/or merge with other saved files;
    - Run (play)
        - Possible real-time input (performance)
        - Save sampled-date stream and/or merge with others;
    - Debug (with source symbols, at selected breakpoints).

LANGUAGE DESIDERATA

- Behavior well defined in all circumstances
    - Valid statements -- defined actions
    - Invalid statements -- defined error messages
- All likely errors checked for
    - Omitted parameters detected (strictly limits "defaults")
- Gives access to all hardware features of Digital Synthesizer
- Real-time input (e.g. keyboard) supported
- Permits building constructs from primitives
    - -- like function or macro definitions
    - -- to any depth needed
    - -- recursively if necessary
    - -- used with the same syntax as the primitives
- Repeat/indefinite repeat feature
- Format-free (not tied to "card columns")
- Viable with 64-graphics subset of ASCII
- No language distinction as to different passes of compilation
- No need to go to another language for computational processes
- No GOTO
- "Style" library feature
- Ordering of numeric parameters not required in function call
    - (eliminates prime source of clerical errors)
- Adaptable to various approaches to digital music:
    - Computer music orientation;
    - Live instrument orientation;
    - Compositional algorithms.

LANGUAGE FEATURES

The following describes various desired semantic capabilities. A syntax is given in order to show examples embodying the semantic features.

Symbol Types:

- Fixed symbol
  - throughout program is synonym for a given number
  - form is a name preceded by a period
  - can be used anywhere in place of a number
  - definition is a declaration:
    - `.name = value`
  - .T is built in and (exceptionally) has the running value of the elapsed time
- Running variable
  - value is a function of time into piece
  - form is a name preceded by a dollar sign
  - definition is an action:
    - `$name = value`
    - `$name = ? * value + value`
    - `$name = 2 ** (? + value) + value`
    - (In the above possible forms, ? is a dummy symbol representing elapsed time since the defining action.)
  - alternative definition form uses : instead of =
    - (Difference is that all = actions take place before any other actions called for at same moment of time; all : actions take place after any other actions called for at the same moment.)
- Function
  - value of a function call depends on definition
  - call delimits arguments:
    - `[name,val0; arg1,val1;argn,valn]`
    - (There may be any number of arguments. The first is unnamed and is accessed in the definition as the character # . Other arguments are denoted by name in the call (arg1 and argn in the example above).)
  - definition is declaration
    - `name = value`
    - (The value may be any expression, including function calls and running variables. The expression is evaluated when the function is called.)
  - built-in functions include SIN, COS, LOG, EXP, URAN (uniform random number), [AMPL,n] (amplitude of generator n).
- Argument
  - value is that assigned by function call or action (see Invocation below)
  - form is name
- Instant name
  - has no numeric value
  - can be invoked in an action (see Invocation)
  - is an instant, hence can invoke consequent actions
  - form is name

- Reserved words to flag statements of special types
- Command action name
  - denotes element of Digital Synthesizer
  - instant names must not conflict
- Hardware parameter name or mode name
  - denotes quantity in an element of Synthesizer
  - (mode name also conveys value)

Numeric Values:

- Real numbers (decimal radix);
- Fixed symbols;
- Running variables;
- Arguments;
- Function calls;
- Constructs with +-*/\ () (\ is modulo, ** is exponentiation);
- Conditional expressions
  - ex.: keyval<lim1,val1; lim2,val2; limn,valn>
  - (This works as follows: keyval modulo limn (or whatever the last lim is) is computed and the result is compared to the successive lim terms. If less than lim1, then val1 is returned as the value of the expression; if greater than or equal to lim1 but less than lim2, than val2 is returned; and so on.

Invocation:

(The primary programming interface to the Digital Synthesizer is the command stream. Hence it is natural that the most important primitives of the language are of the form: "At a given instant, perform a given command.")

- Forms:
  - instant => action
  - action <= instant
- Instants:
  - named instant
    - ex.: NAME
  - value of elapsed time
    - ex.: 1.03
  - list of instants
    - ex.: NAME; 1.03; 1.17
  - arithmetic progression
    - ex.: <1,3,...,21>
  - function applied to any of the above
    - ex.: [U,<1,3,...,21>] is equivalent to [U,1]; [U,3]; [U,5]; etc.
  - moment when a relation involving a running variable (or [AMPL,n]) becomes true (having been false). Relations include greater than, equal, less than, and combinations.
    - ex.: [AMPL,15] "BLE" 0 means when the amplitude of generator 15 goes down to 0. It denotes the set of discrete moments when the relation becomes true, not an interval during which it is true. (BLE means Becomes Less than or Equal to.)

state-change of external input

ex.: KEYDN is invoked when a key is is pressed (becomes down) on a keyboard. Depending on the features of the keyboard, # may be the key number and KVEL its velocity, for example.

any of the above delayed by an amount of time

ex.: NAME ++ 1.03

any of the above conditioned by a relation involving one or more running variables (or [AMPL,n]).

ex.: NAME & $VAR "LE" 0 means any time the named instant NAME is invoked AND the running variable $VAR has a value less than or equal to 0.

(Note that an instant is, in general, multiple-valued. This is akin to a subroutine which can be called from different places.)

Actions:

set running variable (see above)

compile Digital Synthesizer commands

ex.: GEN,3; FREQ,440; SIN; AMPL,.5

The most common command actions are GEN, MOD, and DLY, followed by the unit number and one or more parameter phrases.

A parameter phrase is either a word that sets a mode: RUN, STOP; SIN, SAW; LIN, EXP (for envelope); AM, 2POL, MIX (modifier mode); etc. or a parameter name (FREQ, AMPL, K0, K1, etc.) and the value to set the parameter to.

Such an action compiles one command for each parameter phrase, in the order given.

Certain parameters have alternative names, to permit giving the value in several convenient scales: FREQ in hertz, FFREQ as a fraction of the sample rate; AMPL for amplitude in linear mode, EAMPL to denote the amplitude that comes out of the exponential table.

invoke named instant

ex.: INST,val0; ARG1,val1; ARG2,val2

The unnamed argument (which may be omitted) is given the value of val0, and can be accessed by the character # in any action invoked by the name INST. There may be any number (including 0) named arguments, given the values indicated and accessed by name (e.g. ARG1), not only in any action invoked by the name INST, but also in any action invoked by such an action, and so on.

(It is entirely possible that at the same moment of time the same named instant or command action name (GEN, MOD, etc.) will be invoked more than once by different paths. These invocations are not merged, but are processed independently in a well-defined order.)

Indefinite Repeat

to repeat statements with tabular entries for arguments

```
ex.: IRP; arg1,<1;2;3>; arg2,<5,6,7>
     one or more statements using arg1, arg2
     ENDIRP
```

(In this example, the block of statements between the IRP and the ENDIRP will be repeated three times: the first time, arg1 will have the value 1 and arg2 the value 5; the second time, arg1 will have the value 2 and arg2 the value 6; etc.)

Miscellaneous Declarations

declare sample rate, number of update ticks
give beginning and ending times of piece
declare real-time inputs to be used
give name of source-language file to "insert"

Comments

!begun by exclamation point, ended by carriage return

DEBUGGER FEATURES

Use of source-program symbols of all types

Breakpoints on various conditions:

- Named instants
- Time value
- (highly desired: All instants that can be specified in language)
- Named function call
- Hardware overflow

Ability to proceed from one breakpoint to another

- Sequences in defined order through breakpoints which occur at same moment of time

Examination and modification of values of:

- Running variables
- Arguments (if at breakpoint where argument defined)

(desired: Ability to make some kinds of modifications to program without recompilation)

SOME SIMPLE LANGUAGE EXAMPLES

```
1.03 => GEN,12; STOP; FREQ, 440; IFM,0; SIN; AMPL,.5; ASYMP,0;
    LIN; DRATE,0; RUN
```

Above is a statement using only primitives of the language to set up an oscillator and start it running 1.03 seconds into the piece. Spaces and tabs have no function except to improve program readability. A line ending with a comma or semicolon is automatically continued. Since STOP is the first parameter phrase and RUN is the last, the other parameter changes will cause no momentary undesired output even if they are not all done at the same sample time. The statement can be made parametric on instant, generator number, frequency, and amplitude as follows:

```
DAH => GEN,#; STOP; IFM,0; FREQ,FRQ; SIN; AMPL,AMP; ASYMP,0;
    LIN; DRATE,0; RUN
```

and called, for instance, as follows:

```
1.03 => DAH,12; FRQ,440; AMP,.5
```

The symbol FRQ is used to pass down the frequency. FREQ could have been used, with a parameter phrase FREQ,FREQ in the GEN action, with no ambiguity: the two uses of FREQ (argument and hardware parameter name) can always be distinguished by the context. However, such usage should probably be deprecated on the grounds that it is confusing to the programmer.

A similar example using a decaying envelope follows:

```
PING => GEN,#; STOP; IFM,0; FREQ,FRQ; SIN; EAMPL,AMP; ASYMP,0;
    EDRATE,-.1; EXP; RUN
```

Here EDRATE is used to specify an exponential decay rate such that the amplitude falls to half its original value in a tenth of a second (a rate of about 60 dB per second).

To provide a finite duration to the note initiated by DAH, the following can be used:

```
DIT => DAH,#
DIT ++ DUR => GEN,#; STOP
```

with a call such as:

```
1.03 => DIT,12; FRQ,440; AMP,.5; DUR,.333
```

An elementary "compositional algorithm" could be implemented as follows:

```
0 => $N=0
ZOT => DIT,30; FRQ,$N<1,440; 2,550; 3,660>; AMP,$N<1,.75; 2,.5>
ZOT => $N:$N+1
```

In this example, each time ZOT is invoked generator number 30 sounds a note whose frequency is taken cyclically from the sequence 440, 550, 660 and whose amplitude is either .75 or .5 (alternately). Since ZOT does not specify the DUR argument used by DIT, the call to ZOT has the responsibility of declaring its value. The = form is used to initialize $N and the : form to set its new value after any use of the old value.

A minimal program for real-time performance could be:

```
IRP; N,<1,2,3,4,5,6,7,8,9,10,11,12>;
     M,<.C4,.CS4,.D4,.DS4,.E4,.F4,.FS4,.G4,.GS4,.A4,.AS4,.B4>
  0 => RESET,N; FRQ,M
ENDIRP
! Just 1 Octave for Illustration
RESET => GEN,#; STOP; IFM,0; FREQ,FRQ; SAW; DRATE,0; AMPL,.9;
    ASYMP,0; LIN
KEYDN => GEN,#; RUN
KEYUP => GEN,#; STOP
```

The fixed symbols for frequencies of the tempered scale are assumed to be in a "style" file inserted by a declaration not shown.