THE SYTER PROJECT
SOUND PROCESSOR DESIGN AND
SOFTWARE OVERVIEW

Jean-François Allouis
Jean-Yves Bernet

INA • GRM • September 1982

Abstract
This document presents a general description of the Syter audio
digital processor. This processor has been incorporated in the GRM digital
studio. Second part of the paper gives more concise information about the
outlines of the experimental software and its development towards a higher
level program for use by electroacoustic musicians. This part should be read
as a progress report.

1 - Introduction

The main purpose of the Syter project is to provide electroacoustic
musicians with new tools for sound manipulation in real time, using the digital
technology. As "concrete" approach to electroacoustic music is a rather practical
and empirical one, a special benefit is expected from real time interaction and
 audition.

Specific design goals may be related to this particular musical
environment:

- Capability of treating and transforming natural recorded sounds
  with good audio quality; this feature looks to us more important
  than the synthesis performances.

- Special attention paid to manipulation of continuously varying
  control parameters. The musicians whom we work with may sometimes
  use notes, are dealing often with discrete events, but they do have
  a particular interest in evolving phenomena.

Of course, we have to consider more common constraints: costs, co-
veniently and also ease of programming and size, keeping in mind a later possible
realization for live performance (actual system is mainly intended for studio use).
2 - Audio processor design

2.1 - General architecture

The sound processor uses a kind of Harvard architecture: data and instruction memories are separated for increasing processing speed. It is essentially a three operand machine as the arithmetic unit can be considered to have three inputs and one output (figure 1). This choice has allowed us to design multipurpose arithmetic operators (as we shall see later) while reducing the number of data read/write cycles.

A control memory (1024 x 24) drives the computation. It provides addresses for reading out operands, codes to the control sequences and the arithmetic unit. In fact this control memory houses the synthesis program which determines nature of operators to compute and way of connecting them via the scratchpad memory. Since the whole machine is highly pipelined, the synthesis program must execute sequentially (jumps are forbidden). Recursion occurs at identical intervals, according to the length of the program and in correspondence with the basic sampling rate.

Microprogramming of the processor must be understood in a restrictive sense. This means that most of the control signals (including those of the arithmetic unit) are generated by special memories. These memories are ROM's; therefore applying minor changes is rather easy but (fortunately?) cannot be done by the user.

2.2 - Operand scratchpad and array memories

Operand scratchpad memory is duplicated: three memories (labeled A, B and C) are physically present in the machine, each of them having the same content at any time; but for the external world they are looking as a single memory. As it can be easily seen (figure 2) this feature leads to a 2-microcycle operation (one for reading out the three operands and one for writing back the result); instead of 4 microwcycles with the single memory solution. From now we consider a single scratchpad; but it should be clear this is only a practical convention for easier explanation.

A larger memory has been included for table look-up and buffering. It can be used for DMA transfers of sound files from/to disk units and for generation of audio delays. This array memory has a size of 64K words -16 bit wide - and is logically divided in sixteen arrays of 4K words (Provision has been made for 256K extension). 4K arrays can be directly addressed by the output of arithmetic unit; anyway smaller arrays can also be used with some extra address computation.

2.3 - Operators and how to use them

How we have to introduce operators and basic cycles. Such a cycle lasts four microwcycles (in the present realization 4 x 70ns), during which the machine computes two different operators (figure 3).

The first one combines (A+B) operand pair with the help of an add/subtract unit. The result is written back to the scratchpad and also serves as an index to access an array in which operand (C1) can be written.
1. Audio processor architecture
1. M(a) → ALU
2. M(b) → ALU
3. M(c) → ALU
4. ALU → M(d)

1-memory system

1. \( M_A(a) \rightarrow \text{ALU} \)
2. \( M_B(b) \rightarrow \text{ALU} \)
3. \( M_C(c) \rightarrow \text{ALU} \)
4. ALU \( \rightarrow M_A(d) \)
5. ALU \( \rightarrow M_B(d) \)
6. ALU \( \rightarrow M_C(d) \)

3-memories system

2. Duplicated scratchpad memories and what they emulate

(C1) (B1) (A1) (A2) (B2) (C1)

\( \frac{4}{-} \)

in

4 K array

out

index

shifter

multiplier

\( \frac{4}{-} \)

Operator 1

Operator 2

(4n)

(4n+2)

3. Operators for basic cycle \( \frac{4}{n} \)
The second operator multiplies operand (B2) by operand (A2) or by the value read in the same array with the same index. The multiplicand may be shifted twelve bits left for interpolation purposes. Result of multiplication is then added to operand (C2).

Operators are dealing with 24-bits fixed point 2' complement numbers with the except of multiplication (until now 16 x 16 = 24, than to 32). Results are written back to fixed addresses of the scratchpad memory (in 4n and 4n+2 where n is the rank of variant basic cycle). This rather severe restriction will probably be removed in a later version. Writing back can be conditionalized according to the result of an arithmetic test. At last, read/modify/write operations on arrays are possible in one single basic cycle.

An important point to be emphasized is that the "add/subtract" unit is a little more clever than a common adder. In fact, this unit allows use of saturation arithmetic, absolute value and extremum computation, and conditional interrupt generation to an external device.

Because of the pipelined structure, execution of operators is overlapped and may extend to four basic cycles if arrays are used. This shows that current output of operators #n may not be available before operands fetching of operators #m (n+4). Although this is of little consequence for several synthesis algorithms, the problem can be hidden by renumbering the operators in an interleaved way, as follows: \(1 \cdot 1, 2 \cdot 5, 3 \cdot 9, \ldots\)

Control word for each operator is contained in two 24 bits words of the control memory, with a sixteen bit field for operation code.

Let us now consider the canonical example of figure 4, in order to check efficiency of the two operator types as building blocks for audio processing. Tone generation oscillator with amplitude control (4a) requires one pair of operators, as well as ramp generator (4b) or exponential interpolator (4c). Comb filter (4b) is one and a half, and interpolation oscillator (4e) two and a half.

2.4 - Interfacing

The sound processor has to communicate with host computer and with other similar units or DAC's and ADC's.

Host interface uses the system bus for asynchronous read/write access to control, scratchpad and array memories which are looking to the host as part of its own main memory. Because of the large size of the array memory, a paging scheme was adopted in order to save computer address space. As the actual interface is for DEC PDP 11 (a sixteen-bit machine), complete access to the 24-bit memories is achieved through a special extension register for the eight least significant bits.

A different kind of interface allows the input/output of at least sixteen sampled data streams. The output interface is quite simple; any operator result can be loaded into the output register with an arbitrary channel number. The input data is stored in the sixteen upper positions of scratchpad with an exact address depending upon channel number. Here a 16 word FIFO memory provides a partial level of timing flexibility. In particular, this mechanism eliminates synchronization needs when connecting with devices operating at identical sampling rates.

Both host and streamed input requests are served by stealing basic machine cycle. It is the reason why cycles are spared (even if not used) at a fixed but strappable ratio, actually 1:6.

236
4a. truncation oscillator

4b. ramp generator

4c. exponential interpolator

4d. comb filter

4e. interpolation oscillator
2.5 - Prototype implementation and overall performance

The prototype processor was wirewrapped on a standard PDP 11 computer board (390 mm x 200 mm) with TTL logic, using advanced Schottky technology for the most critical paths. Memories are high-speed MOS devices. Power consumption is a quite reasonable 5V/7A.

Overall performance may be estimated through the evaluation rate \( F \) of the pair of operators introduced in section 2.3. If \( f \) is the main clock frequency and \( r \) the interface ratio (section 2.4.)

\[
F = rf / (1 + r)
\]

For \( f = 14 \) MHz and \( r = 6 \), then \( F = 3 \) MHz. With a 32 kHz sampling rate, this is equivalent to 94 truncation oscillators on ramp generators, 47 comb on two pole filters on 31 interpolation oscillators connected at will.

The prototype version of the sound processor has been included in the GRM digital studio (built around a DEC PDP 16/60 minicomputer with 160 kbytes of disk storage for sound files). Dedicated to real time are a graphic digitizer and four output audio converters (input conversion is accomplished via the studio regular converters).

3 - The software

We did not attempt to solve the general problem of synthesizer control by writing one unique and definitive program. Too many problems would have to be solved simultaneously. Instead, we chose to split the project into two phases. The first phase includes almost all the features of the final system, but differs in the level of concepts involved, integration of tasks and user interface.

3.1 - Step one : getting started

This first step can be seen as a pre-release, useful for debugging and performance evaluation. It also made it possible to build some instruments for musical production. It consists of two programs : an interactive program editor and an offline assembler.

The INTERACTIVE PROGRAM EDITOR/LOADER is capable of loading single synthesizer instructions and data, activating "host procedures", saving and restoring the state of the whole system, connecting sound files and converters. Host procedures are subroutines whose periodical execution is triggered by the presence of their name in a list. The play mode displays periodically system state information and a set of selected variables, with their name, unit (dB, Hz, ms) and value expressed in that unit. A graphic digitizer, with a built-in ten keys numeric keypad, emulates ten virtual two-dimensional control devices. Each of them can be dynamically scaled, recorded and updated. The system deals with three kinds of files: sounds (input and output), editfiles (update), and memory-image files.

The OFFLINE ASSEMBLER has been written as a preliminary assistance in building complex instruments. Input is a symbolic file which describes the arithmetical operations performed by the synthesizer, in a syntax very close to the structure of a SYTER operator. Output is a binary file loadable with the interactive editor. The assembler allows symbolic naming of operators and variables, 24-bit constant generation, automatic feeding of default values for partially used operators.

238
3.2 - Step two: getting smarter

Although this phase of the project has just begun, we will enumerate the trends determining the writing of the second version.

First: we want to solve the problem of generating a program for two tightly coupled machines (the synthesizer and the controlling computer).

Second: we want a higher-level user interface, hiding the technical constraints as far as they are not musically meaningful.

Third: we want better control and feedback on the performance, with sophisticated edit capabilities.

The new system will integrate the functions of the two previous programs in two distinct modes: the BUILDING MODE, which involves code production through a definition language, and the PERFORMANCE MODE, in which control is exercised on the synthesizer through non-alphanumeric inputs.

It is essential that switching between the two modes can be done at any moment. Writing a hundred lines of code and hearing “blip” from the computer is one of the most frustrating experiences of non-interactive programs users. A benefit of real time is to make possible the progressive building of complex manipulations with immediate auditive feedback. Real interactivity means no restriction on switching between listening and programming.

3.2.1 - The synergy problem

The SYNERGY problem arises from the fact that synthesis programs are not only synthesizer code but two-parts programs, involving both synthesizer instructions and controlling computer instructions. The two interacting in an often complex fashion. The intricate symbiosis between the two machines is not a musical constraint, but the result of an architectural choice. As such, it must be logically hidden from the musical end of the system.

To meet that, each elementary program must be treated as a vector whose one coordinate is a set of synthesizer instructions, and the other coordinate a set of computer “activations.” Synthesis instructions are symbolic operations which, after (possibly interactive) assembling and linking, can be executed by SYTER hardware. Computer activations are symbolic instructions which can be interpreted by the kernel of the control program running on the computer.

As we have defined the set of synthesis instructions by hardware (actually microprogram), we have to define the set of activations as the minimal and combinable operations we need in most sound-processing algorithms, but does not exist in synthesizer hardware. This will extend the definition of the synthesis language and will unify the definition of modules as co-programs working in two cooperating machines.

3.2.2 - The building mode

We have to choose the LOWEST-LEVEL OBJECT seen by the user through the system. Additions and multiplications are adequate for debugging purpose, but not meaningful enough to build instruments. Although they are not musical objects, signal processing functions, like filters or oscillators, seem more appropriate. As they are well-known and easy to implement, we choose to provide them in the base level environment.

Since we do not want each construction to start from this level, the user must not be bound to the same set of basic objects. We need a multi-level BUILDING PROCESS, allowing definition and use of new objects in terms of already defined objects. We also want that user-defined objects be used exactly as system-defined objects, with no visible difference, including the syntax.

239
This also leads to the definition of USER WORKSPACE, saving all or selected definitions written in the course of a session. Simply naming a workspace, gives access to all the objects built during the sessions referring to the same workspace. These features, often found in large interpreters (e.g. Interlisp, Smalltalk), makes it easy to write complex functions by breaking them down into smaller ones easily tested in an interactive context.

3.2.3 - The performance mode

Playing synthesizers in real time has been used for a long time in the GRM. Sticks and pots are widely used on analog machines, both for studio and live music. There is first a problem due to the mechanical nature of such controls, as opposed to a piano keyboard for example. They have no resistance, no feedback, and detect only bare coordinates.

It is not our concern to solve this problem inside the SYTER project. But it is important to notice that musicians are not completely satisfied with the input hardware existing on most synthesizers. For the moment, we use a graphic tablet as a general two-dimension input device. It could be a stick or a trackball, as well, since an input device is controlled by a specific driver supplying normalized values to the system.

From this unique input medium, we can control an arbitrary number of logical channels. A LOGICAL CHANNEL is a connection between an internal variable and an external data flow subject to scaling and edition. Scaling includes gain and offset setting, interpolation, amplitude quantification and selection of linear or exponential response. Edition takes place between the logical input channels and the synthesizer. Each channel is recorded on disk and can be selectively updated in real time. A relative mode suppresses the necessity to match the levels.

All these functions, except interpolation, are held by the control computer, whose processing power places a limit on the number of channels we can safely use. Beside this limit, the system does not stop or crash, but lowers the the sampling rate of logical channels. We have thus implemented an "elastic response" of the system to an overflow of external events.

At the other end of the system, convenient output must be available, along with the audible one, for MONITORING purposes. A high-resolution graphic display will be added to the system. It is of prime importance in the editing process because a large number of channels must be monitored, each with its name, numeric value and an analog representation of the current value. Absolute synthesis time, relative position in the sound files, and miscellaneous informations, like system load and memory usage, will also be displayed.

Extensive use will be made of the graphic display as a SELECTING DEVICE (by pointing at an object with a cursor), thus disabling the alphanumeric terminals in performance mode. The goal here is to build an environment in which all the possible actions at one moment are displayed by their name or pictogram, with a simple and unique selection scheme to start any of them (à la Xerox Smalltalk). This achieves a fast and natural way of interacting with the system, allowing the specialist users to start without any knowledge of commands or syntax.