HOKKAIDO UNIVERSITY

Title	ICOSS-1 : An Interactive Digital continuous System Simulator
Author(s)	Koyama, Shoichi; Sano, Takashi; Miura, Ryoichi
Citation	北海道大學工學部研究報告, 87, 97-107
Issue Date	1978-06-05
Doc URL	https://hdl.handle.net/2115/41455
Type	departmental bulletin paper
File Information	87_97-108.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

Bulletin of the Faculty of Engineering,
Hokkaido University, No. 87 (1978)

ICOSS-1 : An Interactive Digital Continuous System Simulator

Shoichi KOYAMA* Takashi SANO**
Ryoichi MIURA*

(Received September 30, 1977)

Abstract

This paper describes ICOSS-1 designed as a pilot-system of the ICOSS (Interactive COntinuous System Simulator) project, which tries to establish a software system for a new-type digital simulator using the multiprocessor system being proposed by the authors [7], [8]. An "easy to operate simulator" is the basic design concept of the ICOSS project. Main feature of ICOSS-1 is the thorough interactiveness from program editing to its execution. The set of statements and commands while being simple are sufficient to carry out simulation so that even a novice can easily learn and operate it. The system can be used like an analog computer.

1. Introduction

In the field of continuous system simulation, analog computers have been playing very important roles. They provide rapid computation and intimate man-machine interaction which are essential for efficient trial-and-error processes in simulation. There are, however, several difficulties in analog computation, e. g., their maintenance, repeatability and accuracy of solutions, etc. Among those, the burdens of patching and scaling are often the cause of considerable trouble in the preparation of simulation, and at times, even cause despair in using analog computers.

On the other hand, many attempt have been made to make general purpose digital computers into useful simulation tools. Many languages or software systems for simulation of continuous dynamical systems have been developed up until now [1]～[6]. They provide us with very convenient patching-free and scaling-free simulation tools which give highly accurate solutions and complete repeatability.

Because of sequential calculation, however, digital computers usually consume much more time than analog computers in their execution of simulation. In addition to that, in digital simulation with a batch-type language [1], [3], [4], we can not expect an intimate man-machine interaction like those of in analog simulation. While, with interactive simulation languages [2], [5], [6], it is almost impossible to expect quick responses from the simulators, since the line-by-line interpretation processes working in these simulators consume still more time.

Thus, new types of computer systems for efficient simulation which has quick responses, highly accurate solutions, and intimate man-machine interaction are desired.

* 精密工学科 自動制御工学講座 Automatic Control Engineering Laboratory, Department of Precision Engineering.
** FUJITSU Limited.

We have previously proposed a multiprocessor system for simulation of continuous systems [7], and realized it [8] on a universal multiprocessor system "HARPS (Hokkaido university ARray Processor System)" [9] in which eight microprocessors work in parallel.

The proposed simulator has a special memory and bus structure for a stored interconnection scheme which relieves the burdens of patching. And a flexible control structure of this system enables us to use rather complicated numerical methods than tranditional digital differential analyzers (DDAs) to obtain solutions of high accuracy. The availability of these facilities has been verified by the experimental results with the system carried out using HARPS [8].

Now, ICOSS (Interactive COntinuous System Simulator) project proposed in this paper is an attempt to establish a software system to make such a computer system as mentioned above be a powerful and "easy to operate tool" for simulation. In designing the ICOSS system, several new aspects should be taken into consideration. In particular, the most important point is the intimate and efficient man-machine interaction.

ICOSS-1 is a pilot-system for ICOSS which will be a software system for the simulator on HARPS. Its main feature is thorough interactiveness from program editing to its execution. The set of system instructions implemented here is still a minimum set sufficient to perform simulation, since our major interest at the moment is to examine the feasibility of a good man-machine interaction.

The ICOSS-1 system is described with one of the interactive procedural languages; BASIC, which, we believe, reduces our system developing efforts, and facilitates further extension of the system. ICOSS-1 is working now on the minicomputer ECLIPSE S/200 which is the host processor of HARPS, with 96 KB memory and standard peripherals, i. e., Disk, Card Reader, Line Printer, CRT Terminals, etc.

In the next chapter, we present the external specifications of ICOSS-1. The internal structure of the ICOSS-1 system is outlined in chapter 3 to show how the system processes simulation programs. An example is shown in chapter 4 to illustrate how ICOSS-1 as a whole works. Chapter 5 is the conclusion of this paper.

2. External Description of ICOSS-1

2.1. General Scope of ICOSS-1

As the pilot-system of the ICOSS project, the capability of interactive usage throughout the whole simulation processes is the main advantage of ICOSS-1. The trial-and-error processes essential in simulation can be carried out in the interactive manner with this system.

The simulation processes are performed through two system interactive modes, i. e., program editing mode and its execution mode. And the system mode can be easily changed from the user's terminal whenever necessary ; from editing-to-execution or execution-to-editing.

The system instructions are divided into two different types ; statements for describing simulation problems, and commands for operation of the system. The statements are almost based on the proposal of the SCi's CSSL in 1967 [10]. The set of these statements implemented on this version, however, is still a minimum set,

simple but sufficient to carry out simulation processes, and less attention is paid to flexibility of the system for sophisticated programming, such as block structure, procedural programming and macro facilities, etc.

The command set consists of program editing commands and execution control commands. These commands are used for interactive operation of the system; the program editing commands help the user to edit or modify his program in the program editing mode, and with the execution control commands the user can control the execution of simulation in the execution mode. Some of statements can be used like commands for interactive use.

Above all, the following points are considered of paticular importance on this version:

(1) The user should not be required to have a detailed knowledge of the system. He just thinks of his simulation problem in his mind and concentrates on his simulation.

(2) It should be easy for the user to describe, to understand, and to modify his program. It should also be easy even for a novice to learn how to use or operate the system.

2.2. Statements

The statements are classified into two categories:

(1) Representation statements,

(2) Control statements.

These statements can be used in any order in the program regardless of the sequence of computations, which is automatically arranged by the system using a sorting technique. And further, all statements are examined line-by-line on syntaxes when they are put into the system. If there is a syntactical error, immediately the error message and its diagnosis are displayed on the terminal.

Representation statements

The format of the representation statements and operators used here are as follows:

Format: 〈variable〉 = 〈expression〉

Operators: Arithmetic operators +, −, *, /, ↑, (,)

Simulation operator INTEG (ic, variable or expression)

where "ic" is an initial condition.

Example:

```
X1D = X2
X2D = −A1*X1−A2*X2
X1 = INTEG(IC1, X1D)
X2 = INTEG(IC2, X2D)
```

Control statements

The control statements are further divided into three types, that is;

(a) data statements,

(b) output control statements,

(c) execution control statements.

The functions of these statements are tabulated in table 1.

Table 1. Control Statements

	statement	function and example
data statement	CONST	This statement specifies the values of constants. ex. CONST A=0.1, B=0.5
	INCON	This statement specifies the values of initial conditions. ex, INCON IC1=IC2=0.0, IC3=1.0
	PARAM	This statement specifies the parameters whose values are keyed in on the TSS terminal at the beginning of execution. ex. PARAM C, D
output control statement	TITLE	This statement specifies the content of header title. ex. TITLE EXAMPLE 1
	DISP	This statement specifies an output variable displayed in a picture image. (scaling of figure is possible) ex. DISP 15+10*YDD (15 : positioning factor 10 : scaling factor)
	PRINT	This statement specifies output variables displyed in a tabular form. ex. PRINT YDD, YD, Y
execution control statement	TIMER	This statement specifies the integration step, the output interval, and the termination time of simulation. ex. TIMER DELT=0.5, PRDEL=1.0, ETIME=20.0
	FINISH	This statement specifies the termination conditions. ex. FINISH Y=0.4

These functions of the control statements are almost like those of CSMP's [4]. To avoid redundancy in data statements, the system allows such a brief notation as

$$\text{INCON IC1} = \text{IC2} = \text{IC3} = 0.0,\ \text{IC4} = 0.5$$

instead of

$$\text{INCON IC1} = 0.0,\ \text{IC2} = 0.0,\ \text{IC3} = 0.0,\ \text{IC4} = 0.5\,.$$

For interactive use, these control statements can be used like commands for editing or modifying a simulation program in the program editing mode. Especially DISP and PRINT statements are also used to change output variables even in the execution mode. The control statements are used with cares as follows :

When a new control statement which has the same keyword to some statement already used is put in,

(1) for data statements, every new contents is appended to the previous ones,
(2) for output control and execution control statements, the latest contents remain in their place, and
(3) for all control statements, the statement with no content deletes all the variables and their values which have been already specified by the previous statement with the same keyword.

2.3. Program Editing Commands

Table 2 shows the program editing commands which can be used on ICOSS-1 in the program editing mode.

Table 2. Editing Commands

command	function and example
LIST	Displays user's program already put in. Representation statements are displayed with their line numbers. ex. LIST
KILL n	Erases a representation statement whose line number is appointed. (n: line number) ex. KILL 4
INSERT	Appends or replaces a representation statement by a new one with the same leading variable. ex. INSERT YDD=INTEG (IC2, Y)
RFILE	Specifies an input file name and ICOSS-1 starts reading from this file. ex. RFILE EXAMPLE (where EXAMPLE is file a name).
PFILE	Specifies an output file name to which the contents of current program is transferred. ex. PFILE PROG1 (where PROG1 is a file name.)
CLEAR	Erases all the contents of a current program. ex. CLEAR
END	Tells the end of the user's program to the system. ex. END

To correct or modify a control statement, as previously mentioned, it is sufficient to put in the corrected or modified one from the terminal. On the other hand, KILL command or INSERT command is used to correct representation statements. For example, if the user wants to correct a wrong representation statement, he deletes it by using the KILL command with its line number, or replaces it with the corrected one by the INSERT command. Representation statements are automatically numbered and are always displayed with their line numbers when they are displayed using the LIST command.

Program input media are CRT terminal, and/or a card reader, and/or a paper tape reader, and/or disk files. Since each input medium is treated as a kind of files, the RFILE command is used for choosing an input medium to be used, for instance, if the card reader is an input device;

RFILE $CDR↵.

2.4. Execution Control Commands

To provide a good man-machine communication, ICOSS-1 has on-line execution control commands for parameter control, output variable control, and execution control. Those commands can be used on the system in the execution control mode. Table 3 shows the on-line execution control commands and their functions.

With these commands, the user can start his simulation (RUN) and stop ("esc") while watching the results being provided by ICOSS-1. He can change the values of parameters (SET), or examine the current values of parameters or variables (TYPE), or change output variables (DISP, PRINT). He can also continue his simulation from

Table 3. On-line Execution Control Commands

command	function and example
SET	Changes the value of any appointed variable into the value indicated. ex. SET IC2=0.4
TYPE	Displays the current value of any appointed variable. ex. TYPE IC1
DISP	Changes the output variable to be displayed in a picture image. ex. DISP 15+20*YD
PRINT	Changes the output variables to be displayed in a tabular form. ex. PRINT Y, YD, YDD
RUN	Starts execution. ex. RUN
CON	Continues execution after the user's interruption. ex. CON
RUND	Starts Problem Checker. ex. RUND
COND	Starts Problem Checker after the user's interruption. ex. COND
"esc"	A key board switch which interrupts program execution.
EDIT	Changes the system mode from execution to edit. ex. EDIT

the point where he interrupted a moment ago (CON). On the other hand, if the user cannot agree with the result, he can start the problem checker (RUND, COND) which displays the current values of all variables with their user-defined names. Thus, all values of the variables he uses are displayed at every output interval which is defined by the PRDEL parameter in the TIMER statement in his program.

To change the system mode from the execution control mode to the program editing mode, the EDIT command is used.

3. System Structure of ICOSS-1

ICOSS-1 consists of three subsystems; program editing subsystem, language processing subsystem, and execution control subsystem. The structure of ICOSS-1 is shown in figure 1. All subsystems are described with BASIC at this version.

Progrom editing subsystem

This subsystem is named CSSL. It allows the user to use program editing commands to put in or modify his program. It always examines syntaxes of the statement line-by-line when it is put in, and stores it to a certain core area in a form of text strings. The END command tells the end of the user's program to this subsystem and the language processing subsystem follows the execution.

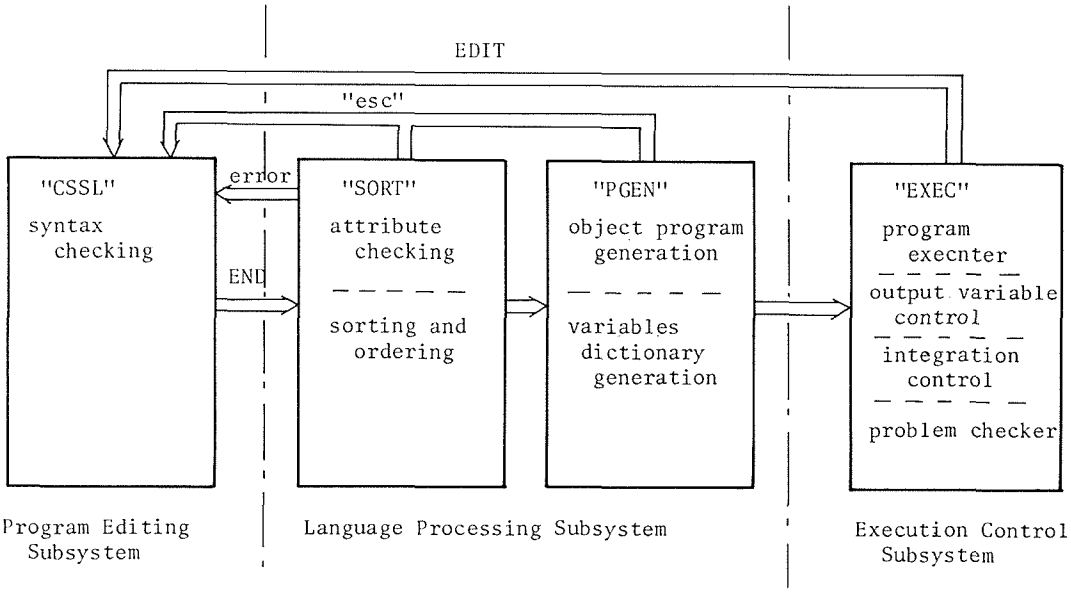

Fig. 1. The structure of ICOSS-1.

Language processing subsystem

This subsystem is divided into two programs; SORT and PGEN. The SORT program, at first, examines all attributes of the user-defined symbolic names. Each attribute of variables is to be indicated by the corresponding data statement or representation statement. When a symbolic name whose attribute has not been defined yet in a program is found, an error message with the symbolic name is displayed on the terminal. Then the control of this subsystem is automatically passed to the program editing subsystem. In case no error is found, SORT starts sorting process of representation statements to arrange calcaulation sequence of simulation in regular order. Therefore, the user does not have to bother himself to think over the correct sequence of calculation at his programming. He only describes his simulation model in his mind. In sorting process, if any algebraic loop is found, an error message "SORTING ERROR LINE-n" is displayed and the control is returned to the program editing subsystem again.

The PGEN program generates an object program which is a workable form of the user's program, according to the results of sorting and ordering in the SORT program. The object program on this version is still in a form of BASIC, however it will be extended in the near future to the form which the microprocessors of "HARPS" can carry out. To allow the user a free format variable name, a complete dictionary of the user-defined variable names is generated. It correlates the user-defined symbols with working variables of the system. This dictionary is handed over to the execution control subsystem and is referred at the execution time with *parameter* or *output variable control commands*.

Execution control subsystem

This subsystem follows the PGEN program in the language processing subsystem. When the object program is completely generated by the PGEN, this subsystem

```
 RUN"CSSL
INPUT FILENAME ? VANDERPOL
OUTPUT FILENAME ? POL
TITLE ********** VAN DER POL'S EQUATION ************************
INCON XDIC=0.5,XIC=0.0
TIMER DELT=0.01,PRDEL=0.25,FTIME=10.0
DISP 30+10*X
PRINT X,XD
CONST LAMDA=1.2
XD=INTEG(XDIC,LAMDA*(1.0-X↑2)*XD-X)
X=INTEG(XIC,XD)
# END
SORTING COMPLETED
USER'S PROGRAM LOADED
SYSTEM IS READY !

# RUN
********** VAN DER POL'S EQUATION ************************
 0            0                                          *
 .25          .13691707                                  I*
 .5           .31740465                                  I  *
 .75          .5312168                                   I    *
 1            .76018238                                  I      *
 1.25         .97138763                                  I        *
 1.5          1.1277732                                  I          *
 1.75         1.2064612                                  I           *
 2            1.2054384                                  I           *
 2.25         1.1338167                                  I          *
 2.5          .99941554                                  I        *
 2.75         .80174334                                  I       *
 3            .52845805                                  I    *
 3.25         .1531887                                   I*
 3.5         -.35668691                                 *  I
 3.75        -.98484706                               *    I
 4           -1.5603563                            *       I
 4.25        -1.8740893                          *         I
 4.5         -1.9453587                        *           I
 4.75        -1.8929666                         *          I
 5           -1.7852271                          *         I
 5.25        -1.6465736                           *        I
 5.5         -1.4823725                            *       I
 5.75        -1.2892731                             *      I
 6           -1.057386                                *    I
 6.25        -.76841446                                 *  I
 6.5         -.39122052                                *   I
 6.75         .11927409                                  I*
 7            .78578735                                  I      *
 7.25         1.4748911                                  I            *
 7.5          1.9039276                                  I                *
 7.75         2.0278791                                  I                 *
 8            1.993824                                   I                *
 8.25         1.8960883                                  I               *
 8.5          1.7679318                                  I              *
 8.75         1.6177757                                  I             *
 9            1.4446009                                  I            *
 9.25         1.242006                                   I          *
 9.5          .99776966                                  I        *
 9.75         .69071523                                  I     *
 10           .28585295                                  I *
SIMULATION TERMINATED
```

Fig. 2. A sample and its results for the van der Pol equation.

```
# EDIT
# LIST
TITLE ********** VAN DER POL'S EQUATION **********************
PRINT X,XD,
DISP 30+10*X
TIMER DELT=.01,PRDEL=.25,FTIME=10.02
INCON XDIC=0.5,XIC=0.0,
CONST LAMDA=1.2,
PARAM
FINISH
  1. XD=INTEG(XDIC,LAMDA*(1.0-X↑2)*XD-X)
  2. X=INTEG(XIC,XD)
# END
SORTING COMPLETED
USER'S PROGRAM LOADED
SYSTEM IS READY !

# SET LAMDA=1.8

# RUND
********** VAN DER POL'S EQUATION **********************
 0            0                                        *


 XDIC= .5  XIC= 0      EEEEE1= .5 EEEEE2= 0
 XD= .5    X= 0



 .25          .14713905                                I*


 XDIC= .5  XIC= 0      EEEEE1= .75901335      EEEEE2= .15461229
 XD= .74732413         X= .14713905



 .5           .37088665                                I  *


 XDIC= .5  XIC= 0      EEEEE1= 1.073081       EEEEE2= .38148994
 XD= 1.0603293         X= .37088665
SIMULATION INTERRUPTED AT TIME= .5
```

Fig. 2 (continued)

immediately starts off with loading it. After that this subsystem then allows the user to control the execution of his simulation with a help of the on-line execution control commands. This subsystem possesses a program executer routine, an output variable control routine, an integration control routine and a problem checker routine. When this subsystem recieves the EDIT command from the user, the control of the system is passed to the program editing subsystem.

4. Example

To show how ICOSS-1 works, the van der Pol equation defined by

$$\ddot{x}-\lambda\cdot(1-x^2)\cdot\dot{x}+x=0$$

is picked up as an example. Figure 2 shows the program and its results displayed on the terminal

The operation of ICOSS-1 starts when the command RUN"CSSL is keyed in on the user's terminal. Then, ICOSS-1 asks the user which input medium he uses, with a message "INPUT FILE NAME?". In this example, the user's program is already stored in a disk file named "VANDERPOL", so VANDERPOL is replied in this case. Since ICOSS-1 uses an over-lay technique, the user is then requested to put in an output file name after "OUTPUT FILE NAME?". In this case, it is "POL". Then the system starts to load the program from the file "VANDERPOL".

After the END command is put in, ICOSS-1 starts attribute checking and then sorting. In case no attribute and sorting error is found, a message "SORTING COMPLETED" is displayed; otherwise the control of the system is returned to the program editing subsystem. When the "EXEC" subsystem finishes loading the object program, messages "USER'S PROGRAM IS LOADED" and "SYSTEM IS READY" are displayed. Then, after the prompt mark "#" from the "EXEC" subsystem, the user can start his simulation using on-line execution commands. And so his simulation proceeds.

A simulation result of this example follows the RUN command in the figure. There is also shown how the problem checker works for the same example with a different value of the parameter LAMDA.

5. Conclusion

We have demonstrated how ICOSS-1 works. ICOSS-1 is a pilot-system of the ICOSS project which tries to establish a new-type all-digital simulator for continuous systems. The most important point in designing the ICOSS system is the intimate and efficient man-machine interaction which enables to carry out the effective trial-and-error processes in simulation. And the main feature of ICOSS-1 is the thorough interactiveness from program editing to its execution.

The command set is easy even for the novice to learn and use intuitively. Most of the execution control commands correspond to the operation of analog computers, such as SET to POTSET, CON to COMPUTE, RUN to RESET and COMPUTE, "esc" key to HOLD, and RUND, COND to PROBLEM CHECK. Therefore, ICOSS -1 can be used like an analog computer, however the accuracy and repeatability of

solutions, and handiness in its operation are far superior than in analog computers.

Since our major interest at the moment is in examining the feasibility of a good man-machine interaction, we employed here a minimum set of statements, and paid less attention to sophisticated programming. Those facilities for sophisticated programming will be considered in the succeeding versions of the ICOSS project.

Because of using BASIC, one of the high-level interactive language, for system description, it did not take so long time to implement ICOSS-1. Five months were necessary for a person. BASIC, however, is a redundant language for system description, so that, much more efficient high-level interactive language is desireable for our purpose.

References

1) Harnett, R. T., Sansom, F. J., Warshawsky, L. H.: "MIDAS—an Analog Approach to Digital Computation", SIMULATION, Vol. 3, No. 3, May 1964.
2) Brennan, R. D., Sano, H.: "PACTOLUS—A Digital Analog Simulator Program for the IBM 1620", Proc. AFIPS FJCC, 1964, pp. 299-312.
3) Chu, Y., Sansom, F. J., Petersen, H. E.: "Digital Simulation for Continuous Systems" (Book), McGraw-Hill Book Company, 1969.
4) "Continuous System Modeling Program III (CSMP III) Program Reference Manual", IBM Data Center Services.
5) Liebert, A. T.: "DARE II: Fast On-line Digital Simulation on a Small Computer", J. of AICA, No. 4, Oct. 1971, pp. 72-78.
6) Worth, G. A.: "SIMEX-Simulation Executive and Compiler", Proc. SCSC, July 1974, Houston, Texas, pp. 61-67.
7) Koyama, S., Miura, R.: "A Multiprocessor System for On-line Simulation of Dynamical Systems", Simulation of Systems, 8th AICA Congress, Delft, Aug. 1976, pp. 263-269.
8) Koyama, S., Isurugi, Y., Miura, R., Sakurama, M., Sano, T.: "A Realization of a DDA System for Continuous Dynamical System Simulation with a Universal Multimicroprocessor System 'HARPS'", EUROMICRO Newsletter, Vol. 3, No. 4, Oct. 1977.
9) Tanaka, Y., Miyashita, K., Koyama, S., Miyamoto, E., Tsuda, T.: "HARPS (Hokkaido university ARray Processor System): A New Hierarchical Array Processor System", 2nd EUROMICRO Sym. on Micro Architecture, Venice, Oct. 1976.
10) The SCi, "The SCi Continuous System Simulation Language (CSSL)", SIMULATION, Dec. 1967, pp. 281-303.