Using ADA tasks to simulate operating equipment

Louis A. DeAcetis

Oron Schmidt and Kumar Krishen

Reprinted from COMPUTERS IN PHYSICS, September/October 1990 © American Institute of Physics
Using ADA tasks to simulate operating equipment

Louis A. DeAcetis
Physics Department, Bronx Community College/CUNY, Bronx, New York 10453

Oron Schmidt and Kumar Krishen
NASA, Johnson Space Center, Houston, Texas 77058

(Received 15 November 1989; accepted 5 January 1990)

A method of simulating equipment using ADA tasks is discussed. Individual units of equipment are coded as concurrently running tasks that monitor and respond to input signals. This technique has been used in a simulation of the space-to-ground Communications and Tracking subsystem of Space Station Freedom.

INTRODUCTION

Many computer simulations1–3 written in procedural languages (e.g., C, FORTRAN, or PASCAL) simulate systems of equipment by tracking signals through the components. Although this may represent a logical solution to the problem, it usually requires that a piece of equipment know not only what it is connected to for input, but also the destination of its output. This is contrary to the way equipment generally operates. For example, an amplifier may have inputs of line voltage, signal level (volts), and load impedance, and control settings of gain and ON/OFF switch position. Its output would include the output signal level, and perhaps some parameter indicating the quality of the output. When the input values change, the values of the output parameters change accordingly. The destination of the output signal is of no concern to the amplifier, and it therefore does not know (or care) what is connected to its output. (Note that we are treating loading as the input parameter “load impedance.”)

The ADA language is especially suited to simulating a piece of equipment because of the “task” construct.4 Each piece of equipment can be modeled as a concurrent free-running task that constantly monitors its input values and adjusts the outputs accordingly. As each unit reacts to changes in its input values, one can monitor the signal flow through a collection of components by placing “sensors” at strategic locations. Issues of signal transition delays, and other equipment characteristics can be addressed as needed.

I. METHOD

Each unit of equipment is modeled as an ADA task. The individual equipment characteristics are supplied in separate ADA packages (one for each unit of equipment), which include the appropriate transfer function(s) for the input signal(s). All of the values that are external to a piece of equipment are stored in a global database (or “blackboard” data structure5), and the individual components link their internal values to these blackboard values. Each component task then monitors those blackboard values that serve as input to the equipment it is simulating, and while the equipment is ON and running, appropriate output values for that equipment are generated and written out to the “blackboard” where they can then be monitored for input by those tasks using them. For example, if all of the equipment is plugged into the same power source, then each monitors the blackboard value of the line voltage. If the line voltage vanishes (“blackout”) or is low (“brownout”), then this can be incorporated into the determination of the appropriate output signal(s).

In order to prevent a task from “running” when the equipment is in the OFF state, an algorithm is used that requires that the equipment be turned ON in order for it to process input information. Table I contains the structure of this ON/OFF algorithm, which is coded in a generic ADA package so that it can be implemented (“instantiated” in ADA terminology) for each piece of equipment. One of the features of the algorithm is that it will accept and ignore any ON/OFF requests that are redundant, rather than queue them as might normally be the case with ADA task rendezvous (i.e., if an ON command is sent to a task that is already ON, then the command is discarded). Another feature is that the task of a piece of equipment in the OFF state does no processing, but merely waits for a rendezvous to turn it ON. This minimizes the use of CPU resources.

II. ILLUSTRATIVE EXAMPLE

Consider the equipment depicted in Fig. 1, which consists of a saw-tooth function generator attached to a pulse generator/amplifier. The circled numbers refer to sensor or test points whose values are to be monitored. The waveform produced by the function generator is used by the pulse generator to determine the pulse width as follows: While the value of the input to the pulse generator is at or beyond a certain threshold value (taken as 0.5 V), the value of the pulse generator output is +10.0 V; when the value of the input is below threshold, then the pulse generator output is 0.0 V. We thus have a waveform transformer that converts a saw-tooth signal into a rectangular pulse. The frequency of both active signals is the same, and the pulse width can be varied by changing the amplitude of the saw-tooth signal. In addition, effects of a “brownout” (line voltage less than a nominal value of 120 V) have been incorporated in
TABLE I. ADA package with task code for algorithm used to simulate an equipment unit.

```
--Package Generic_Equipment
-- Package containing task to implement specific instances of equipment
-- Last update: 11-09-89

generic
  procedure Set Initial_OV (Values); * values for equipment just turned on;*
  procedure Set Running_Values; * values for equipment on & running;*
end GENERIC_EQUIPMENT;

package GENERIC_EQUIPMENT is
  task Control is
    entry Close_Switch;* entry OPEN_Switch;*
  end Control;
  procedure OFF_values;
end GENERIC_EQUIPMENT;

package body GENERIC_EQUIPMENT is
  switch OPEN Switch is
    entry Closed: -- delay Switch Closed;*-- print Switch Closed;*-- start Switch Control Loop;*-- Single Switch Control Loop;*-- Should only get here if Switch was Closed;*-- Set Running_Values;*-- Delay;*-- Loop Switch Closed;*
  end
end switch;

procedure OFF Values is
  delay 0.001;*-- delay Switch Opened;*-- start Switch Control Loop;*-- Single Switch Control Loop;*-- Should only get here if Switch was Opened;*-- Set Initial_OV Values;*-- Delay;*-- Loop Switch Opened;*
end OFF Values;
end GENERIC_EQUIPMENT;
```

that the output levels of each device will decrease until a "minimum operating voltage" for the equipment is reached. Output from the function generator vanishes when the line voltage drops below 90 V, and for line voltages below 80 V, the output from the pulse generator also ceases. Figure 2 is a dataflow diagram for this equipment setup.

Table II contains ADA code that implements the above for each piece of equipment and includes the structure of the global database. Table III is a main ADA proce-

FIG. 2. Flow diagram of data and command information flowing between the blackboard global database, the ADA equipment simulator tasks, and the main procedure of the Illustrative Example. Straight line (3/M) = data; dashed line (-----) = commands.

TABLE II. ADA code for Illustrative Example equipment, including the Global Database definition and the packages for the function and pulse generators.

```
--Package GLOBAL_VARIABLES is a global database where shared info is stored and where connections between signals are made.
-- Last update: 11-09-89

package GLOBAL_VARIABLES is
  type Source_Variables is
    VOLTAGE: float range 0.0 to 1.0;
    INDEX: integer range 0 to 10;
  end Source_Variables;

  signal VOLTAGE: source Variables;*-- cycle source var;*-- cycle at end of cycle;*-- Global Database*-- Global Database
end GLOBAL_VARIABLES;
```

that the output levels of each device will decrease until a "minimum operating voltage" for the equipment is reached. Output from the function generator vanishes when the line voltage drops below 90 V, and for line voltages below 80 V, the output from the pulse generator also ceases. Figure 2 is a dataflow diagram for this equipment setup.

Table II contains ADA code that implements the above for each piece of equipment and includes the structure of the global database. Table III is a main ADA proce-

FIG. 1. Illustrative Example equipment setup consisting of a (saw-tooth) function generator whose output drives a pulse generator. Circled values indicate sensors or test points.
TABLE II. (Continued.)

```
AMPLITUDE : float rename GLOBAL_VARIABLES.FUNCTION_GENERATOR_AMPLITUDE;
OUTPUT : float rename GLOBAL_VARIABLES.FUNCTION_GENERATOR_OUTPUT;
---
BEGIN
PROCEDURE OFF_VALUES (begin
  -- Set output values for equipment in OFF state:
  PROBOUT(2) := 0.0;
  PROCEDURE_OUTPUT_SIGNAL := OUTPUT);
end;
---
PROCEDURE INITIAL_ON_Values (begin
  -- Set output values for equipment just turned on:
  PROCEDURE_OUTPUT_SIGNAL := (OUTPUT := 0.0);
end;
---
PROCEDURE OUTPUT_VALUE (begin
  -- Set output values for equipment on and tuning:
  PROCEDURE_OUTPUT_VALUE := FUNCTION_GENERATOR_OUTPUT;
end;
---
```

```
package PULSE_GENERATOR is
  with
  -- Pulse Generator Simulator
  end
  --*
END PULSE_GENERATOR;
```

```
COMPUTER IN PHYSICS, DAW
```

```
end
```

```
---
```
```
PROBE : package
procedure INITIAL_ON_Values;
procedure INITIAL_ON_Values;
procedure PROBLEM(OUTPUT_VALUE (begin
  begin -
  TRIGGER
  PULSE_GENERATOR;
end
```

```
GLOBAL VARIABLES; -- of
generates a pulse
counter := COUNTER
end
```

```
AYE
```

```
end
```

```
begin
```
```
FUNCTION GENERATOR AMPLITUDE;
```

```
```
GLOBAL VARIABLES; -- Global Database where signal values are
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
TABLE III. (Continued.)

<table>
<thead>
<tr>
<th>X2</th>
<th>INTEGER; PULSE GENERATOR PROBE OUTPUTS</th>
<th>( x_{2} ) INTEGER; PULSE GENERATOR PROBE OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{WINDOW ARBITRARY (LAST LINE)} )</td>
<td>( \text{WINDOW ARBITRARY (LAST LINE)} )</td>
<td>( \text{WINDOW ARBITRARY (LAST LINE)} )</td>
</tr>
<tr>
<td>( \text{SET CURSOR ABSSA (FIRST, LAST, LINES)} )</td>
<td>( \text{SET CURSOR ABSSA (FIRST, LAST, LINES)} )</td>
<td>( \text{SET CURSOR ABSSA (FIRST, LAST, LINES)} )</td>
</tr>
<tr>
<td>( \text{NEW LINE} )</td>
<td>( \text{NEW LINE} )</td>
<td>( \text{NEW LINE} )</td>
</tr>
</tbody>
</table>

---


drawn, called SIMULATE, which interfaces with the simulator and produces output similar to that in Fig. 3. Although this is a simple application, it does illustrate the method and suggests how greater sophistication is possible.

### III. IMPLEMENTATION

Figure 4 is a block diagram of the starboard portion of the proposed Space-to-Ground subsystem of the Communications and Tracking System on Space Station Freedom. As above, the circled numbers represent sensors whose values can be monitored. The oval enclosed numbers represent sensors whose values indicate the ON/OFF state of the equipment. This system has been simulated using the above paradigm: Each rectangular box is represented by a task that, when “ON,” monitors the values of its input signals and sets the values of the output values and sensors accordingly. The simulator may be controlled by other programs or using a keyboard interface program that permits the asynchronous entry of commands to turn equipment ON/OFF and set cross-strapping switches. There are upward of

**FIG. 3.** Sample output of procedure SIMULATE which displays probe readings and signal levels for the Illustrative Example. The outputs of the function and pulse generators are displayed graphically and scroll upward to show the changes with time. The "cycle count" indicates the number of cycles completed by each running task and procedure SIMULATE.
with no major problems. (As more tasks are switched "ON," they do slow execution somewhat, especially on an IBM PC-AT.)

Additional refinements of the model presented here are possible, most especially in the area of information hiding. One of the major problems with a blackboard approach is that signal information is not only visible to all of the equipment tasks, but accessible (i.e., modificable) as well. It is thus possible for the Function Generator to get access to the output value of the Pulse Generator even though it has no electrical connection to that output. However, from the code in Table II, it can be seen that one would deliberately have to associate a local variable with an improper global variable in order to accomplish such a connection. There are ADA constructs that can be used to prevent any unwarranted access to normally inaccessible signal levels, but the level of abstraction and programming complexity required would obscure what is basically a simple concept and implementation, and these were deemed beyond the scope and intent of the present article.

IV. CONCLUSIONS

One of the unique aspects of the ADA programming language is the ability to do logically "parallel" processing using the task construct. This is especially useful in simulating concurrently running equipment. Satisfactory results are readily obtainable for situations where transient states can be ignored (e.g., where we are not concerned with the output of the pulse generator during the transitions between its minimum and maximum voltage states). When the latter are important, timing considerations can greatly increase the complexity of the problem. Real-time simulations, which require timing and interrupt considerations, constitute a further challenge in ADA, which is not considered here.

ACKNOWLEDGMENT

This work was done while the first author held a National Research Council–NASA Senior Research Associateship.

REFERENCES

Modeling superconducting networks containing Josephson junctions by means of PC-based circuit simulation software

James A. Blackburn
Department of Physics and Computing, Wilfrid Laurier University, Waterloo, Ontario N2L 3C5, Canada

H. J. T. Smith
Department of Physics, University of Waterloo, Waterloo, Ontario N2L 3C5, Canada

(Received 30 January 1990; accepted 18 May 1990)

Software packages are now available with which complex analog electronic circuits can be simulated on desktop computers. Using Micro Cap III it is demonstrated that the modeling capabilities of such software can be extended to include superconducting networks by means of an appropriate equivalent circuit for a Josephson junction.

INTRODUCTION

Superconducting circuits, containing Josephson devices, inductances, capacitors, and resistors, have many important practical applications. These include SQUID magnetometers, high-speed superconducting computer elements, and voltage standards. The usual procedure for predicting the behavior of such circuits has been to solve the corresponding sets of nonlinear differential equations numerically. However, as we shall demonstrate, superconducting electronics can be included within the modeling capabilities of presently available circuit simulation software, and this provides a powerful and flexible alternative method of analysis.

With the advent of computer-aided engineering (CAE) software, analog circuits can be simulated on a computer before a hardware prototype is constructed. A well-known mainframe oriented software package is SPICE, which was developed at UC Berkeley in the 1970s. The appearance of high-performance personal computers based on the 80386, and most recently 80486, chips has made CAE simulation of relatively large circuits feasible on desktop machines. Micro Cap III (which was selected for the present work) is a leading simulation package for use on PC's. It has an extensive library of standard devices. Each specific component, such as a 2N2222 transistor or an LM741 op-amp, is modeled so as to replicate accurately that device's static and dynamic characteristics. As will be shown, this library can be extended by creating an equivalent circuit for a Josephson junction.

I. JOSEPHSON JUNCTION SIMULATION

The circuit for simulating a current-biased resistively shunted Josephson device is shown in Fig. 1. The principal elements are an operational amplifier (op-amp) and a voltage-controlled oscillator (VCO). MicroCap III does not provide a VCO in its component library, and so a separate macro, described below, was designed for this purpose.

The VCO shown schematically in Fig. 2 contains three separate submacros:

1. SPDT—a voltage-controlled switch set to act as a zero crossing detector; this is formed by combining two voltage-controlled single-pole/single-throw switches provided within MicroCap III.

2. X—a simple voltage multiplier created from a voltage-controlled voltage source in MicroCap III.

3. SINECONV—a four-diode triangle-to-sine wave converter as shown in Fig. 3. This type of sine converter possesses the important attribute of not introducing any phase shift in the waveform.

The input voltage to the VCO is applied to PIN 1, and then is passed to the control terminal of the first SPDT (which enables the VCO to handle both positive and negative input voltages) and to the multiplier. The action of the circuit may be followed by assuming for the moment that $V_{in} > 0$. Suppose the present state of the circuit is as indicated in the schematic. One input to the multiplier is 8 V, the other is $V_{in}$. The multiplier output is thus $+ 8 \times V_{in}$, and so...