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 simulations\textsuperscript{1-3} written in procedural lan-
guages (e.g., C, FORTRAN, or PASCAL) simulate sys-
tems of equipment by tracking signals through the compon-
ents. 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 desti-
nation 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 imped-
ance, and control settings of gain and ON/OFF switch posi-
tion. Its output would include the output signal level, and
perhaps some parameter indicating the quality of the out-
put. When the input values change, the values of the output
parameters change accordingly. The destination of the out-
put 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.\textsuperscript{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 sepa-
raten 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 "black-
board" data structure\textsuperscript{5}), 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 out-
put 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 ("brown-
out"), 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 fea-
ture 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 gener-
ator/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 sig-
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: 11-02-99 LAD
  --
  -- This has ADA procedures as formal parameters and therefore
  -- must be instantiated with procedures which implement the transfer
  -- functions of the equipment used.
  -- In particular:
  -- Procedure Set Off Values
  -- Procedure Set_INITIAL_ON Values: * values for equipment just turned on;
  -- Procedure Set_RUNNING Values: * values for equipment on a running
  -- package GENERIC_EQUIPMENT is
    task SWITCH_CONTROL is
      entry CLOSED_SWITCH;
      entry OPEN_SWITCH;
      and SWITCH_CONTROL;
      procedure GENE C U P_ENT;
      and GENERIC_EQUIPMENT;
    end GENE C U P_ENT;
  -- procedure SET OFF Values
  -- package GLOBAL VARIABLES
  -- entry OPE$_IT$
  -- end GENERIC_EQUIPMENT;
```

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 procedure.

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 procedure.

```
--Package GLOBAL VARIABLES is a global database where shared info is
-- stored and where connections between signals are made.
-- Last update: 11-02-99 LAD
--
--package GLOBAL VARIABLES is
--
--procedure RETURN_QUANTITY is
--  --From Source Quantities:
--  --Globa'd Database: Signal Values and Line Voltage
--  --Function Generator Signal and Level
--  --Pulse Generator Signals and Levels
--end GLOBAL VARIABLES;
```

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 (---) = data; dashed line (——) = commands.

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

<table>
<thead>
<tr>
<th>AMPLITUDE : float</th>
<th>renames GLOBAL_VARIABLES.FUNCTION_GENERATORAMPLITUDE;</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT : float</td>
<td>renames GLOBAL_VARIABLES.FUNCTION_GENERATOR_OUTPUT;</td>
</tr>
<tr>
<td>TIP : float</td>
<td>renames GLOBAL_VARIABLES.PULSE_GENERATOR_TIP;</td>
</tr>
<tr>
<td>BASE_TIME : calendar.day_duration</td>
<td>calendar.seconds</td>
</tr>
<tr>
<td>INCREMENT : float</td>
<td>renames GLOBAL_VARIABLES.PULSE_GENERATOR_INCREMENT;</td>
</tr>
<tr>
<td>INITIAL_VALUE : float</td>
<td>renames GLOBAL_VARIABLES.PULSE_GENERATOR_INITIAL_VALUE;</td>
</tr>
<tr>
<td>OUTPUT_VALUE : float</td>
<td>renames GLOBAL_VARIABLES.PULSE_GENERATOR_OUTPUT_VALUE;</td>
</tr>
<tr>
<td>PULSE_GENERATOR : package</td>
<td>renames PULSE_GENERATOR_PACKAGE;</td>
</tr>
</tbody>
</table>

### TABLE III. ADA code for the procedure SIMULATE which displays the sensor and signal values for the Illustrative Example equipment.

```ada
package PULSE_GENERATOR is
    package FUNCTION_GENERATOR is
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
            end;
            package SIMULATE is
                with GLOBAL_VARIABLES;
                with PULSE_GENERATOR;
            end;
        procedure SIMULATE is
            with GLOBAL_VARIABLES;
            with PULSE_GENERATOR;
            with test is
                use test;
                -- Do interface package for Alanya or Meridian compilers
                with OFF is Alanya compiler
                with ON is Alanya compiler
```
**TABLE III. (Continued.)**

<table>
<thead>
<tr>
<th>X2</th>
<th>INTEGER; PULSE GENERATOR OUTPUTS</th>
<th>( \text{ABSIS} )</th>
<th>( \text{ABSIS} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td>( \text{ABSIS} )</td>
<td>( \text{ABSIS} )</td>
</tr>
<tr>
<td>2.4</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.8</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>PULSE GENERATOR AMPLITUDE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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.**

**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-referencing switches. There are upwar...
FIG. 4. Block diagram of the starboard space-to-ground subsystem of the Communications and Tracking System of Space Station Freedom. The numbers are sensor identifiers. BSP: baseband signal processor; HDR: high data rate recorder; TDRSS: Tracking and Data Relay Satellite System.

34 tasks for the total system (which includes the port system and contingency communications equipment not shown here). At present, the simulator transforms signal levels and sets sensor readings to typical values. Once the detailed electronic characteristics for this equipment (under development) are established, they can be incorporated into the corresponding tasks.

The simulator has been compiled under a variety of ADA compilers (including Alsys, DEC, Meridian, and Verdix), and runs under MS-DOS on PCs (80286 and 80386 CPU's), and DEC VMS and Ultrix operating systems. Although it is unlikely that all of the communications equipment on the Space Station would be ON simultaneously, the current simulator has been run in that state 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., modifiable) 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 \( V_m > 0 \). Suppose the present state of the circuit is as indicated in the schematic. One input to the multiplier is 8 V, the other is \( V_m \). The multiplier output is thus \( + 8 \times V_m \), and so...