A lock-step synchronization algorithm for logic simulation in a multi-solver environment

X. WANG, T. J. KAZMIERSKI and Z. MRCARICA

Department of Electronics and Computer Science,
University of Southampton,
Southampton SO9 5NH,
U.K.

Tel. (0044)(703)(593116)
Fax. (0044)(703)(593045)

Email: wxg91r@ecs.soton.ac.uk

Abstract

A mini-backplane for integrating logic simulators has been developed. Solver synchronization is done by means of a novel lock-step algorithm with no global time. This contribution presents details of the synchronization and backplane communication interface. The solver engines are linked into the backplane to form a single executable process. The example of a 32-bit adder, partitioned between two instances of an identical solver, illustrates the method.

Keywords

CAD/CAE Techniques and Methodologies as well as Application Simulation

INTRODUCTION

The complexity of the simulation tools has led to the concept of simulation backplane, where third party solvers can be integrated and work on a single design, appropriately partitioned between the solvers. A general-purpose standard for simulation backplane has recently been proposed by the CAD Framework Initiative (CFI) [1]. The mini-backplane described here is not an implementation of the CFI standard but it follows the CFI's guidelines.

Part of the standard is an Application Programming Interface (API) which specifies functions needed to integrate a simulator to the backplane which is a common control program. The backplane is responsible for controlling and synchronizing the simulation.

A subset of the CFI API definitions has been developed to allow integration of multiple logic simulators into the mini-backplane and a future integration of the analogue engine of the behavioral simulator ALFA [2].

SIMULATION MINI-BACKPLANE AND ITS API

The mini-backplane API is a collection of communication protocols and formats that allow simulators to operate using their own algorithms and to share data both during and after a simulation run. Figure 1 shows a general model of the mini-backplane. Pending the development of a user interface, communication with the users is performed via simulators' own input processors and a configuration file which contains data about the partitions. The backplane creates a design data base which reflects the structure of the partitions. The synchronizer controlling the simulation is part of the backplane.

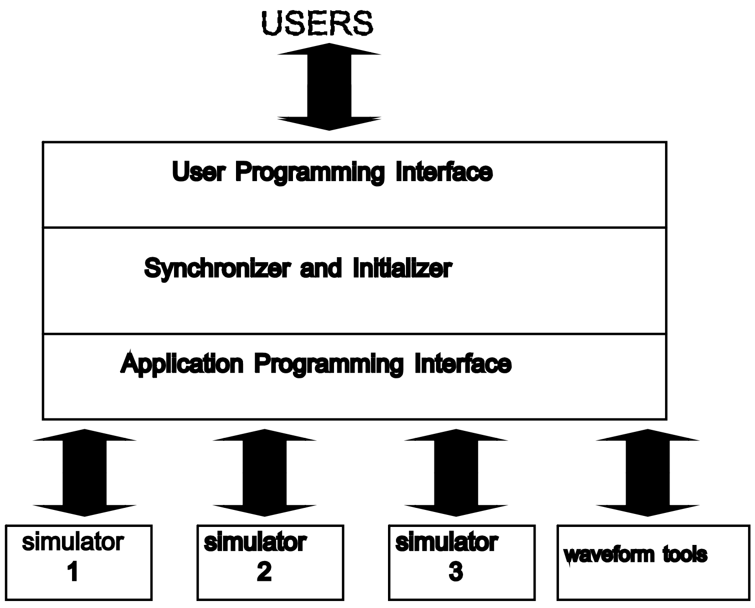
USERS
User Programming Interface
Synchronizer and Initializer
Application Programming Interface
simulator 1
simulator 2
simulator 3
waveform tools

Figure 1 A General Model of the Simulation Backplane

The synchronization algorithm has been tested on logic simulation engines but it has been developed with a view to work in a mixed-signal simulation environment.

GLOBAL NETS AND SYNCHRONIZATION

The signals that pass between simulators are carried by global nets. These global nets might be resolved by more than one simulator. If a global net is driven by more than one simulator, this would require that the backplane resolve the global net as a mixed-mode simulator. A solution method adopted here only allows each global net to be able to connect two partitions (simulators), one of which owns the net. This means that a general global net is divided into two smaller terminal nets, called ports. The resolution of a port is therefore performed within one and only one simulator. The approach ensures that the backplane is only responsible for the communication and synchronization between simulators.

There are at least two kinds of algorithms for the synchronisation between a backplane and simulators. One, optimistic simulation allows each simulator to advance freely in time until all its internal events have been processed. If an event from another simulator is produced before the end of this optimistic time interval, then all events generated after the event are discarded. This means that optimistic simulator engines should have a capability of backtracking.

An alternative approach, adopted in this paper, is the lock-step synchronization. Each simulator generates a next event time but only the simulator returning the smallest value of next event time is allowed to advance until the smallest next event time returned by other simulators. This approach ensures that no results are thrown away and there is no need for backtracking. Most existing logic simulators cannot backtrack and therefore no fundamental changes are required if such simulators are integrated to the mini-backplane.

Two API functions, aaisSendSimNextTime and aaisSimRunThruTime, are very crucial for implementing the lock-step method. The algorithm developed for the purpose of the mini-backplane can be summarized as follows:

(1) The function aaisSendSimNextTime is called for each simulator, sending the next activity time (next event time) to the backplane.

(2) The minimum of the next activity time values is known as the start_time. The simulator, with the time, is the next simulator to be run. The target_time, defined as the next smallest value, is also calculated. If more than one simulator returns the same minimum value, the start_time is the same as the target_time. The simulator, less recently used, is chosen as a next one.

(3) The backplane calls the function aaisSimRunThruTime for the

next simulator, instructing the simulator to run from the start_time to the target_time. The simulator can stop executing before the target_time, if a global event is produced by the simulator before this time. The event is passed to another partition as defined in the configuration file. This means that global events can reduce the target_time and create a new target_time for the backplane to avoid possibly backtracking. During the simulation interval, signal changes on the traced signals are sent to a display waveform tool by the backplane to be displayed graphically.

Steps (1) to (3) are repeated until the event lists for current target_time are exhausted or until the user-specified maximum time is reached.

The lock-step synchronisation algorithm can be described in the form of pseudo-code as follows:

```
      set idle_count for all simulators to zero;
      start_time <- 0;
   while (start_time <= end_time) do
    begin
          start_time <- INFINITY;
          target_time <- INFINITY;
          selected_simulator <- NONE;
/* this for loop selects one simulator to be run */
      for (i=1 to number_of_simulators) do
        begin
           calling function aaisSendSimNextTime to obtain
            next_activity_time;
          if (next_activity_time <= start_time)
            begin
               target_time <- start_time;
                 start_time <- next_activity_time;
                 /* find the minimum next activity time */
```

```
                if(start_time < target_time or
                  idle_count(ith_simulator)              >
idle_count(selected_simulator)
                selected_simulator <- ith_simulator;
              end
            else if (next_activity_time < target_time)
                 target_time = next_activity_time;
      end /* end of the for loop */
/* then execute the selected simulator */
    calling function aaisRunThruTime for selected_simulator until
    target_time;
    set idle_count for the selected_simulator to zero;
    increase idle_count for all other simulator by 1;
    distribute events on the global nets generated by the
selected_simulator to other simulators;
   end /* end of the while loop */
```

The lock-step algorithm can cause a deadlock, i.e. a situation in which the simulation time can never advance, when two or more simulators share a zero-delay path and generate unresolvable events. In most cases, delta cycles caused by zero-delay paths can be resolved but, in general, sharing zero-delay paths between solvers should be avoided.
If the maximum allowed number of delta cycles is exceeded, the algorithm treats this situations as deadlock and stops.

SIMULATION RESULTS

The mini-backplane has been written in C and tested on an 486 IBM PC clone. An experimental logic simulator has been developed, two instances of which have been into the mini-backplane. As an example, a 32-bit adder with ripple carry is simulated by the backplane and logic simulators. Figure 2 shows the 32-bit ALU. The circuit is partitioned into two halfs simulated by the two engines respectively. The partitioned circuit is shown in figure

3. First, the ALU adds the numbers 3 and 2 and then the result is added to the number 3. The clock period is 200ns and the accumulator register propagation delay is 10ns (from the clock edge). The simulation results are shown in figure 4. It can be seen that the sum becomes 3 at 210ns, 5 at 410ns, and 8 at 610ns.

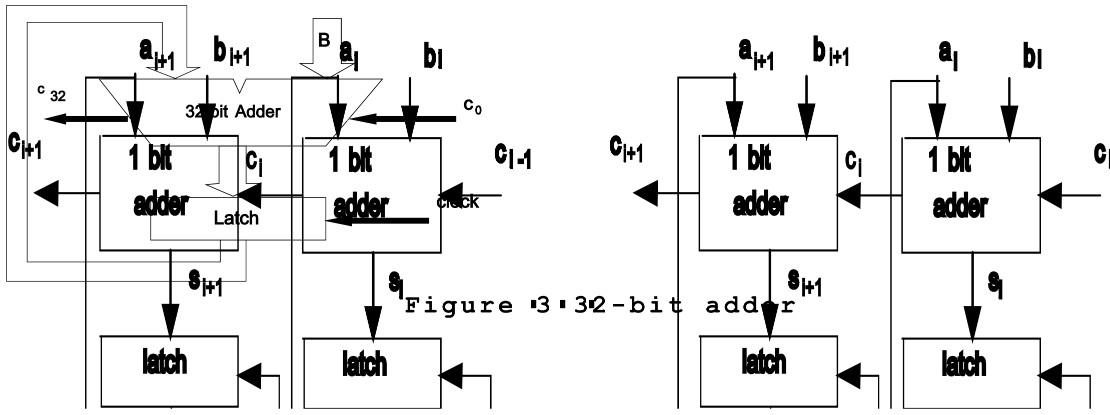

Figure 3 32-bit adder

Figure 4 One Partition Instance (16-bit adder)

CONCLUSION

A simulation mini-backplane and a lock-step synchronization algorithm has been developed. A one-to-one connection method is used to describe global nets. The simulators are directly linked into the backplane through function calls. The principle of the simulation backplane has been demonstrated and a correct operation of the synchronization algorithm has been shown. So far, only logic simulators have been integrated and work is under way to integrate the analogue engine of the behavioral simulator ALFA [2] for mixed-signal multi-solver simulations.

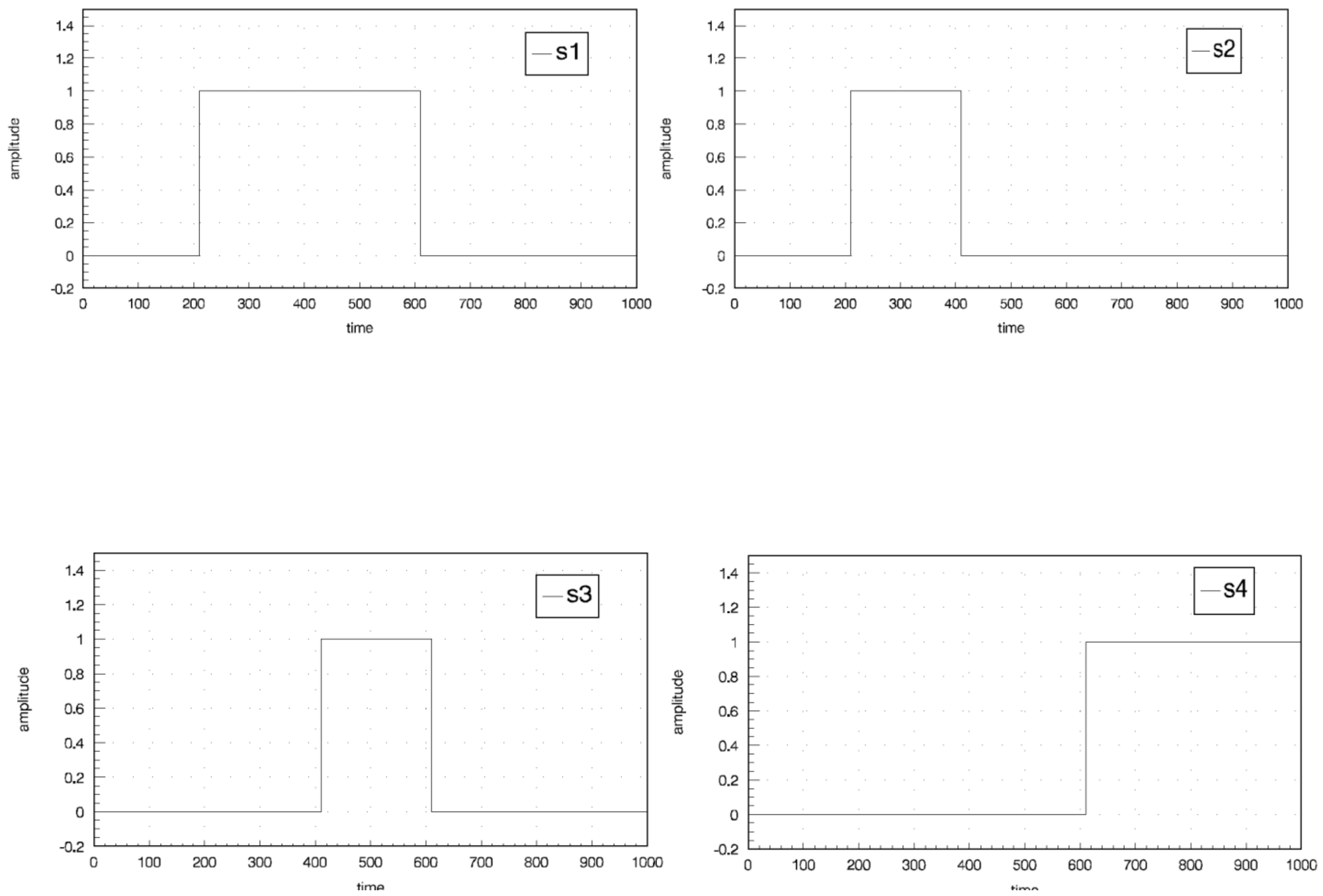

Figure 5. Simulation Results

REFERENCES

[1]T. Kemp, " A proposal for a modest prototype simulation backplane interface", IBM, Revision 1.0, March, 1993.

[2]T. J. Kazmierski, A.D. Brown, K.G. Nichols, M. Zwolinski, "A general-purpose network solving system", IFIP Transactions A-1, VLSI'91, North-Holland 1992, pp.147-156.

[3]S. A. Szygenda and E. W. Thompson, "Digital logic simulation in a time-based, table-driven environment", IEEE Computer, March 1975, pp. 24-36.

[4]Z. Mrcarica, "The ALFA simulation backplane", Research Report, University of Southampton, July 1993.