Engineering Simulation, 1995, Vol. 13, pp. 115-122
Reprints available from the Publisher
Photocopying permitted by license only

Implementation of Compilation Logic Modeling with Delays

A. I. ANDRYUKHIN
Institute of Applied Mathematics and Mechanics, National Academy of Sciences of the Ukraine, Donetsk

(Received March 24, 1993)

This paper considers a method for processing delays in compilation logic modeling that extends the field of application of the latter.

The development of technology in the manufacture of LSI circuits has necessitated research in sufficient detail into the response of the device under design to the arrival time of the input signals and their delays in its structural components. There are a great many ways to describe the dynamic behavior of various devices, including the construction of time diagrams, models with Boolean time functions, automata models [1], continuous-logic models [2], [3], an so on. The need to process vast data arrays when designing VLSI circuits has aggravated the requirements for the automation of this process, and logic modeling has become a fairly important tool in the design and testing of such circuits.

Logic modeling with delays is most accurate in reflecting the behavior of logic circuits in digital computing and control systems, but imposes heavy demands on the speed of modeling. The asynchronous event simulation algorithm in interpretative implementation takes account of the couplings between signals in the communication lines of the device being modeled by means of two lists of data: a queue of the currently active elements and a queue of future events [4], [5].

The queue of future events is necessary to take account of signals occurring after some delay with respect to the current clock period. This queue is a dynamic structure, and there are numerous ways to organize and process it [5], [8]. A specific feature of handling the queue of future events is the processing of what is known as the overflow wheel. The processing difficulties are due to limited computer memory capacities and long propagation delay times.

The development of the Turbo family compilers has made it possible to cut down the time spent in the compilation approach to the implementation of logic modeling systems in the case of changes made in the design of the device in hand. The possibility of obtaining rapidly the necessary data structures to process elements with new properties by means of an invariable modeling algorithm materially eases the introduction of alterations in the system. The principal work in that case is associated with designing the operation

programs for the elements.

The compilation modeling technique is faster than its interpretative counterpart, but the difficulties arising with this technique in accounting for delays restrict its field of application to combinatorial and synchronous sequential circuits [9].

Let us consider a method allowing these difficulties to be largely circumvented. The method is based on the well-known representation of logic circuit elements in the form of a model comprising an ideal logic element and an inertial delay element connected in series [10], [11]. The essence of the method is the automatic construction of an intermediate description of the logic circuit in hand on the basis of its initial description by way of introduction of special elements whose operation under the control of an event-driven algorithm monitor program allows various types of delays to be simulated, and the generation of a program that is a compilation implementation of the event simulation algorithm for the modified circuit.

It should be noted that it is only the program model of the circuit that is the compilation type model. The program modules of the circuit elements are designed individually and are connected together at the editing stage. This approach has been realized in the IKSM compilation system described in [12]. At the present time there is no clear distinction between purely compilation- and purely interpretative-type approaches in the implementation of modeling systems. Most efficient are therefore those modeling systems which make use of the advantageous properties of the both types of models [14].

The IKSM hierarchical compilation modeling system is a compilation type modeling system because it lacks any common interpreter program whose input data are the description of the device and the input stimuli. In this system, a program is written in C language for each device, which implements the modeling of only this particular device and uses its tabular description. A modification of the design requires no substantial expenditure of time in recompilation, for the alteration of the description of the device is effected by a common semantic-syntactic analyzer. So, the system is an interpretative type modeling system.

The automatic modification of the initial circuit for an inertial delay and propagation delays is illustrated in Figure 4, *b* presented in [9]. The change of the initial ELEMENTS and CONNECTIONS tables corresponding to the circuit modification for propagation delays is described as follows.

We designate the data of each entry in the CONNECTIONS table describing some circuit by the symbols *D*, *K*, *DI*, and *KI*, where *D* (*DI*) is the name of the predecessor (successor) element and *K* (*KI*), the serial number of the predecessor (successor) contact, and execute the following operations for each entry in the CONNECTIONS table.

1. Form the unique delay element name *DK* for the element *D* at the output contact *K* by concatenating the identifiers *D* and *K*.

2. Add to the ELEMENTS table the element *DK* of universal propagation delay type.

3. Replace each entry $<D, K, DI, KI>$ in the CONNECTIONS table by the two entries $<D, K, DK, 1>$ and $<DK, 2, DI, KI>$.

The condition of delay elements is defined by the structure of the internal parameters described below, each particular structure corresponding to a particular type of delay. Changing the variable *F* causes the control monitor to put the delay element on the active

element queue, its input X and output Y remaining unchanged. The input data of the program modules of the delay elements are their internal parameters, the input X, and the serial number M of the model clock period.

The condition of a propagation delay element is defined by the following internal variables: T - the model Y-entry time; YT - the value of the output Y at the instant T; $P1$ ($P2$) - the indicator of the first (last) element in the chain of events for the given delay element; F - the flag indicating that the delay element is put on the active event queue; and DT - the delay value.

The chain of events for the delay element at the instant M of the model time, which describes the variation of the input X during the time $(M-DT, M-1)$ is illustrated in Figure 1 and is a unidirectional list consisting of the data structures $E = (TE, XE, PE)$, where TE is the model Y-variation time, XE is the value of Y at the instant TE, and PE is the indicator of the next event E.

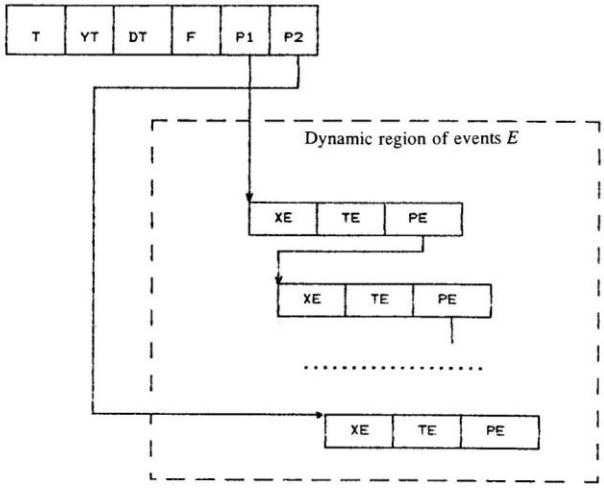

Figure 1

Let us define the operation ALLOCATE (DEALLOCATE) (X), the function of which is to allocate (deallocate) a computer memory space for the variable X. We use the symbol $P \rightarrow X$ to designate the variable X whose address is defined by the indicator P. The indefinite value of the indicator and the address of the variable X will be designated N and

$A(X)$, respectively. Consider the following assumptions as to the operation of the delay element: the element changes its output ($Y = YT$) at the model instants of time T, $P1 \rightarrow TE$, $(P1 \rightarrow PE) \rightarrow TE$,..., and so on; the events are processed on the FIFO principle; the element finishes its self-activation ($F = 0$) with an empty chain ($P1 = N$) and $T = M$ and then initiates itself again ($T = 0$) and changes its output ($Y = YT$); with a nonempty chain and $T = M$, the given events $P1 \rightarrow E$ are transferred to the static memory space of the delay element; with a nonempty chain, a change of the input X of the delay element causes a new event to be added at the end of the queue.

Let us write down the operation algorithm of a propagation delay element with due regard for the notation adopted.

1. If $P1$ is not equal to N, go to 7.
2. If $T = 0$, then $F = 1$, $T = M+DT$, $YT = X$, and output.
3. If $T = M$, then $Y = YT$; otherwise go to 5.
4. If $X = YT$, then $T = 0$, $F = 0$, and output; otherwise $YT = X$, $T = M + DT$, $F = 1$, and output.
5. If $X = YT$, then $F = 1$ and output.
6. Allocate (E). Set $P1$, $P2 = A(E)$, $P1 \rightarrow XE = X$, $P1 \rightarrow TE = M + DT$, $F = 1$, and output.
7. If $X = P2 \rightarrow XE$, go to 11.
8. Allocate (E).
9. Set $P2 \rightarrow PE = A(X)$.
10. Set $P2 = P2 \rightarrow PE$, $P2 \rightarrow XE = X$, $P2 \rightarrow TE = M+DT$.
11. If M is not equal to T, then $F = 1$ and output.
12. Set $Y = YT$, $YT = P1 \rightarrow XE$, $T = P1 \rightarrow TE$.
13. If $P1 = P2$, then deallocate $(P1 \rightarrow E)$, set $P2$, $P1 = N$, $F = 1$, and output.
14. Set $P2 \rightarrow PE = P1$, $P1 = P1 \rightarrow PE$.
15. Deallocate $(P2 \rightarrow PE) \rightarrow E$.
16. Set $F = 1$ and output.

The indicator $P2 \rightarrow PE$ is used as a working field in point 15.

Let us explain the essence of the main items of the above algorithm. An element will be referred to as active if it changes the value of its output. This occurs at the instants $M = T$, where T is replaced in the course of processing with $M+DT$, $P1 \rightarrow TE$, and so on. Otherwise use will be made of the term passive. In point 2, there takes place the processing of the start of operation of the delay element and its initialization. In points 3 and 5, there occurs the processing of the passive element, its input remaining constant, and in points 3 and 4, the processing of the active element, depending on its input value. To illustrate, with the input remaining unchanged, setting $F = 0$ instructs the event-driven monitor to eliminate this element from the further processing. In points 8 through 10, a new event is put on the queue, and in points 12 through 15, the inverse operation – deletion – is performed.

Figure 2 shows changes in the input X (solid line) of the delay element and the corresponding values of its output Y (dashed line) in the case of binary modeling alphabet, and Figure 3 illustrates schematically the condition of the data and event queue of the delay element.

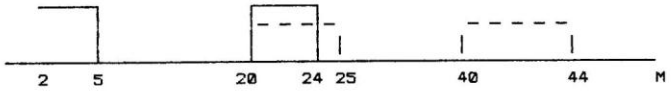

Figure 2

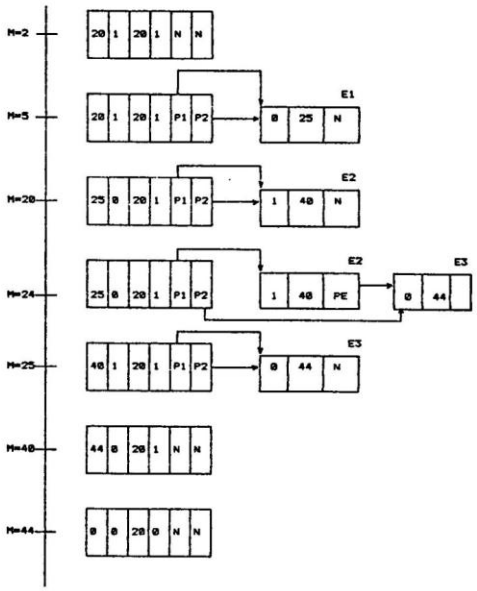

Figure 3

Thus, the operation of the universal delay element consists in its regular self-activation

(by way of its being put on the active element queue at $F = 1$), which continues for as long as the element stores in its own dynamic memory the varying values of its input and the instants at which they are sent to the output. The event-driven monitor starts processing that part of the circuit which follows the delay element once the latter has changed its output. The delay functions which were previously simulated by means of a next event queue are now being performed by the delay element itself. This causes the queue to vanish (it is replaced with the dynamic chains of events of the delay elements) and makes it possible to use a simpler event-driven synchronous algorithm to model the modified circuit.

The result of modeling with inertial delays depends on the location of the delay, i.e., is it located at the input or the output of the element. In both cases, use is made of a universal inertial delay element, whose operation consists in ignoring the input signal X if it is applied in less than DT model clock periods. The condition of this delay element is defined by the following internal parameters: T - the model Y-entry time; YT - the value of the output Y at the instant T; Y - the current value of the output Y; and DT - the inertial delay value.

The operating algorithm of the inertial delay element has the following form.

1. If $X = Y$, go to 7.
2. If $X = YT$, go to 5.
3. If $T = M$, set Y equal to YT.
4. Set $YT = X$, $T = M+DT$, $F = 1$, and output.
5. If T is equal to M, set $Y = YT$, $F = 0$, and output.
6. Go to 4.
7. If X is equal to YT, set $F = 0$ and output.
8. Go to 3.

The approach described differs from interpretative implementation in that the asynchronous modeling with delays reduces to the synchronous modeling of the appropriate modified circuit. There are no more dynamic structure (time wheel and overflow wheel) organization and processing problems. The construction and operation of such structures are replaced by the software implementation of independent delay element modules and their operation in the course of modeling. The simplicity of the software implementation enables one to change delay values in the course of modeling by using different static data structures in defining the delay elements.

Among the disadvantages of this approach is the time wasted to check the operation of the delay elements in each model clock cycle, which is similar to the software inquiry by a processor of the condition of the ports [13], [14]. This is essential where the active element queue only contains long-delay elements. Such a situation being typical, software implementation does not rule out the possibility of detecting it and prolonging automatically the model time by so much clock cycles as will make up a period not too long to allow any changes to occur in the model in its course. The minimum duration of this period is $T-M$ for all the elements in the active queue. This procedure is very important for the efficiency of modeling where use is made of the event simulation principle. One can also note a faster putting of an event on the dynamic list of delay elements, compared to the time wheel, for no time is wasted in the search for the beginning of the list for a certain

instant of time.

The evaluation testing of the implementation has shown that modeling with delays prolongs the modeling time by no more than 270%. Taking into account the high speed of synchronous modeling (it takes some 20 s for the IKSM system using a 16-MHz IBM AT 286 computer to model on 110 patterns a circuit including 70 Series 155, 561, and 586 50-input elements), the approach described above can be considered quite a success.

REFERENCES

1. V. V. Karibskii, P. P. Parkhomenko, E. S. Sogomonyan, and V. F. Khalchev, *Osnovy tekhnicheskoi diagnostiki* (Fundamentals of technical diagnostics)(Moscow: Energiya, 1976)(in Russian).
2. V. I. Levin, *Dinamika logicheskikh ustroistv i sistem* (Dynamics of logic devices and systems) (Moscow: Energiya, 1980)(in Russian).
3. V. I. Levin, *Avtomatika i Telemekhanika* No. 8: 3-22(1990).
4. P. V. Saveliev and V. V. Konyakhin, *Funktsionalno-logicheskoe proektirovanie BIS* (Functional-logic design of large-scale integration circuits)(Moscow: Vysshaya Shkola, 1990)(in Russian).
5. S. S. Badulin, Yu. M. Barnaulov, V. A. Bardyshev *et al.*, *Avtomatizirovannoe proektirovanie tsifrovykh ustroistv* (Computer-aided design of digital devices)(Moscow: Radio i Svyaz, 1981) (in Russian).
6. K. Pradhan, *Fault-Tolerant Computing* (Prentice Hall, 1986).
7. V. V. Litvinov and T. P. Marianovich, *Metody postroyeniya imitatsionnykh sistem* (Methods for constructing simulation systems)(Kiev: Naukova Dumka, 1991)(in Russian).
8. E. A. Avramchuk, A. A. Vavilov, and S. V. Yemelianov, *Tekhnologiya sistemnogo modelirovaniya* (Systems simulation technology)(Moscow: Mashinostroyenie, 1988)(in Russian).
9. K. Kinoshita, K. Asada, and O. Kratsu, *Logicheskoe proektirovanie SBIS* (Logic design of VLSIs)(Moscow: Mir, 1988)(Russian translation).
10. R. Miller, *Teoriya pereklyuchayushchikh skhem* (Switching circuit theory)(Moscow: Nauka, 1971)(Russian translation).
11. F. A. Mikael and A. O. Timofeev, *Avtomatika i Telemekhanika* No. 8: 186-188(1990).
12. A. I. Andryukhin and D. V. Speranskii, in: *Proc. XI Intercollege School-Seminar "Metody i sredstva tekhnicheskoi diagnostiki"* (Methods and means of technical diagnostics)(Ivano-Frankovsk, 1992): 99-102(in Russian).
13. V. G. Artyukhov, A. A. Budnyak, Yu. V. Lapii *et al.*, *Proektirovanie mikroprotsessornoi elektronno-vychislitelnoi apparatury. Spravochnik* (Handbook of microprocessor-based computer design)(Kiev: Tekhnika, 1988)(in Russian).
14. G. M. Yasinavichene, B. V. Burgis, E. A. Metsaev, and I.-A. K. Greblikas, *Testovyi kontrol mikroprotsessornykh BIS na proizvodstve* (Testing microprocessor large-scale integration circuits under industrial conditions)(Moscow: Radio i Svyaz, 1989) (in Russian).