Verification of Timing Constraints in Large Digital Systems

Thomas M. McWilliams

This Paper Was Prepared For Submittal To
17th Design Automation Conference
Minneapolis, Minnesota
June, 1980

January 4, 1980

This is a preprint of a paper intended for publication in a journal or proceedings. Since changes may be made before publication, this preprint is made available with the understanding that it will not be cited or reproduced without the permission of the author.
DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.
Verification of Timing Constraints in Large Digital Systems *

by

Thomas M. McWilliams

December 1979

Computer Science Department
Stanford University

and

Lawrence Livermore Laboratory
University of California

P.O. Box 808, L276
Livermore, Ca. 94550
415-422-0758

*Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48, with support from the Naval Electronics Systems Command and the Office of Naval Research.
Abstract

A technique to automatically verify all the timing constraints in large digital systems, taking into account the component timing properties, wire delays, and any designer-specified timing constraints, is described. This system has been used to verify the timing constraints in the S-1 Mark IIA processor, which consists of 10,000 ECL chips, and is comparable in performance to the Cray 1 CPU.

1 Introduction

The SCALD Timing Verifier takes in the design of a digital system given in the SCALD hardware description language, and analyzes the circuit, looking for timing errors. SCALD (Structured Computer Aided Logic Design) is a complete CAD system that inputs a graphics-based, hierarchical description of a design, and generates a complete set of low level documentation and magnetic tapes to implement the design [2,3,4]. The Timing Verifier does a complete verification based on the minimum and maximum propagation delays of the circuit components, their setup and hold times, minimum pulse width constraints, and wire delays.

Since the Verifier does not perform a full simulation of the design, it does not know the values of most signals during the analysis, but only knows whether they are stable or changing. In case the design is incompletely specified, or the Verifier is unable to determine the value of a signal, the designer can specify signal timing assertions in the design. This information not only allows the Verifier to analyze the timing without resorting to full simulation, but becomes an integral part of the design specification, and increases the understandability of the design. The ability to specify timing information in the design also allows the designer to state explicitly what timing assumptions he is making as he proceeds. These specifications are then automatically checked by the Timing Verifier when the design is completed.

One of the main features of the SCALD Timing Verifier is the ability to verify designs by modules. This not only permits the use of computers with limited memory size, but also allows timing constraints to be checked as a design progresses, on a day-by-day basis. This is particularly important in that it allows timing errors to be corrected before they have a chance to propagate their effects throughout the design, or to induce major design changes late in the design. It also supports an accurate estimate of the cycle time of a machine before the design is completed.

The Timing Verifier can find timing errors in any type of logic, although it works best for clocked systems. In non-clocked systems, it can check delays through specific logic paths, but there is no way of checking the timing between different independent asynchronous units.

In general, the philosophy of the Verifier is to warn against the slightest possibility of an error, thereby alerting the engineer to analyze the circuit in detail to determine whether there is a real problem. Because the Verifier does not know the value of many signals in the network, it in many cases will assume that things are worse than they really are. If a particular path is indicated
to be a problem and it really is not, then information can be added to the circuit to help the Verifier, so that the circuit will be analyzed correctly the next time. For example, if the design counts on correlations that the Verifier is not aware of, false errors can be generated.

2 Previous Approaches

There have been a number of previous approaches to the verification of timing constraints in digital systems, which can be grouped into two main categories: logic simulation [1,5] and worst-case path analysis [6].

The logic simulation approach poses several problems. It requires either a complete design including any microcode and diagnostics, or some way of generating patterns to drive the undefined signals. Waiting until the design is completed to start simulation presents the problem that bugs are not found until late in the design cycle. Generating patterns to drive undefined signals is very time-consuming and difficult, especially when the patterns need to go through the "worst-case" set of states. Simulation is also a very inefficient way of finding timing errors, because of the need to run through a large number of states in order to test all of the "worst-case" timing paths. In fact, for most large digital systems, it is impossible to have a high degree of confidence that the worst-case states have all been tested.

The worst-case path analysis approach examines all paths through the combinatorial logic between registers or latches, searching for the longest and shortest paths. This approach poses problems when the timing of the circuit is a function of the value of some signals, requiring logic simulation in order to determine the timing of the circuit. It is also computationally expensive, but does provide feedback to the designer early during the design cycle, without the need to generate detailed test patterns.

In contrast, the SCALD Timing Verifier eliminates most of the problems with these approaches, allowing the design to be verified as it proceeds, without the need to generate test patterns. It also uses a computationally efficient algorithm to cover all of the states needed to find all of the timing errors. Handling circuits where logic simulation of parts of the circuit is needed to understand the detailed timing is another of its capabilities.

3 Operation of the SCALD Timing Verifier

Consider checking the timing of the simple circuit shown in Figure 1. As input, the Verifier needs a detailed timing model of the components used in the circuit, the timing properties of the signals which are not generated within the circuit, and a set of evaluation directives. These evaluation directives specify how to handle conditions which the Verifier has trouble with because it doesn’t know whether a signal is true or false.

Given a set of input values to a component with any needed evaluation directives, the Verifier can calculate a new set of output values. If an output changes, then all of the primitives that it drives are reevaluated, possibly generating more output changes. This process is continued until all of the signals stop changing, at which point the Verifier knows the timing behavior of all
of the signals. It then proceeds to check all of the setup, hold, and minimum pulse width errors.

The Timing Verifier is a symbolic logic simulator which does case analysis to handle the different timing cases that need to be checked within a design. It is a symbolic logic simulator in that it represents most signals as either stable or changing, rather than as true or false. This greatly reduces the number of cases that need to be checked to find all possible timing errors. Since the Verifier is not trying to check the correct logical operation of the circuit, knowing the actual value of most signals is unnecessary.

To understand the value of using a symbolic representation for most signals, consider checking the timing of a 16-bit adder where the values of the signals are specified. Either an analysis has to be made to determine the worst-case add to test, or all possible combinations need to be tried, because the addition of different pairs of numbers can take a different amount of time. Trying all possible combinations gives \(2^{16} \times 2^{16} = 2^{32}\) cases, representing all the pairs of two 16-bit numbers. Determining the worst-case test to try in general is a quite difficult problem. If, instead of setting the inputs of the adder to a particular value, the inputs go from a changing state to a stable state, then the simulation can not count on a particular set of numbers being added, and the worst case is automatically tested.

Where it is essential to know whether a signal is a zero or a one for a particular cycle, the Verifier must resort to case analysis. This is because the Verifier only knows one value for a signal during the analysis of a particular cycle or case.

Consider the design of a processor that has a variable cycle time. Depending on the operation to be done, a different amount of time is required to execute it, because different paths through the execution unit having different delays are used. To analyze the timing, a number of fundamentally different cases need to be checked, and which case is being analyzed is determined by the value of a number of the control signals. The Verifier will then enumerate the cases, checking them one by one. This could be a very time consuming process, but in practice has turned out not to be so. In the verification of the S-1 Mark IIA processor, it has been found that very few cases (on the order of half a dozen) need to be checked, and that most cases tend to change only a few local signals, making them very quick to analyze. It has also been found that the number of places that need detailed case analysis are small, and that simulating just the main cases is normally quite satisfactory. This allows the verifications that are performed on a daily basis to be executed very rapidly. Periodically, complete checks are done that go through all of the cases.

3.1 Circuit Periodicity

The clock period of a circuit being analyzed is declared before a particular case is analyzed, and for most designs, is the same for all cases. If different parts of the system run at different clock periods, then the least common multiple is declared to be the period of the circuit. For example, in the Mark IIA processor, the arithmetic unit runs at twice the clock rate of the instruction unit, in order to increase the vector processing rate, and so the period declared for the entire machine is that of the instruction unit.
Within a given case, the intervals of stability and change for each signal are assumed to occur periodically at the clock frequency. Consider an edge-triggered register. Within a window determined by the minimum and maximum propagation delay of the register and by the skew on the clock, the output of the register may be changing, but for the rest of the cycle, it must be stable. For a given case, the clock will always occur at the same place within the cycle, and so the behavior of the output of the register will be periodic if one looks only at change versus stability, not at logical truth or falsity.

The Verifier then analyzes a case via an event-driven simulation, taking all signals to be periodic with period equal to the cycle time. It then repeatedly evaluates the circuit elements until all signal values converge.

### 3.2 Values of Signals

At any instant, a signal has one of seven values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>false, or zero</td>
</tr>
<tr>
<td>1</td>
<td>true, or one</td>
</tr>
<tr>
<td>S or STABLE</td>
<td>signal is stable, i.e., not changing</td>
</tr>
<tr>
<td>C or CHANGE</td>
<td>signal may be changing</td>
</tr>
<tr>
<td>R or RISE</td>
<td>signal is going from a zero to a one</td>
</tr>
<tr>
<td>F or FALL</td>
<td>signal is going from a one to a zero</td>
</tr>
<tr>
<td>U or UNKNOWN</td>
<td>the state of the signal is unknown, which is the initial value of most signals</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Values</td>
</tr>
</tbody>
</table>

The value of a signal over a clock period is represented by a linked list, each node of which specifies a value and the duration of that value. The sum of the durations of all the nodes in the list must equal the period of the circuit being analyzed.

When a signal propagates through a gate or wire where it is delayed by a variable amount of time, then skew is added to the signal, representing the uncertainty in when the signal will change. This skew is kept separately in the representation of the signal to preserve information about the width of pulses, to avoid false timing errors arising from not meeting minimum pulse width requirements. If two or more changing signals are combined, then the skew cannot be simply represented separately, and so it has to be incorporated into signals by using the CHANGE, RISE, and FALL values.
3.3 Assertions on Signals

To handle partially designed circuits, the Verifier needs timing assertions on undefined signals. Undefined signals with no assertions are taken to be always stable, to keep them from causing numerous spurious error messages. Two types of assertions are used for specifying clocks, and one is used for defining the behavior of control and data signals.

There are two categories of clock signals: precision and non-precision. The difference between precision and non-precision clock specifications is the default skew used when none is explicitly given. The skew is generated by the variation in the wire delay to the different parts of a system and by the variation in delay between the different buffers used in the clock generation. In the design of a large digital system, these variations can become quite large, and degrade performance unacceptably. To reduce this skew, the short clock paths can have extra delay inserted into them. Because the delay in a clock distribution system varies between machines, in many cases it must be adjusted by hand, by using some type of adjustable delay for each of the clock lines. Using this technique, the skew can be reduced to below some specified amount. In order to verify the timing in a design that has been so de-skewed, it is necessary to describe how the clocks will be adjusted in detail within the design specification. A number of features which will be described in this and the next section have been provided to make this task as easy as possible.

If a clock signal is adjusted to some specified skew, then an assertion can be given within the signal name denoting that fact.Assertions are given at the end of signal names and are preceded by a period. They are considered part of the signal name by the rest of the SCAMD system, which makes sure that all of the assertions for a given signal are consistent.

The format for the assertions for the precision and non-precision clocks are:

\[
\text{SIGNAL NAME.P <value specification> <skew specification>}
\]

and

\[
\text{SIGNAL NAME.C <value specification> <skew specification>}
\]

where:

- \(<\text{value specification}>\) := \(<\text{time range}> | <\text{time range}>, <\text{value specification}>\)
- \(<\text{time range}>\) := \(<\text{time}> | <\text{time}> - <\text{time}>\)
- \(<\text{time}>\) := \(<\text{real number}>\)
- \(<\text{skew specification}>\) := \(| <\text{minus skew}>, <\text{plus skew}> |\)
- \(<\text{minus skew}>\) := \(<\text{negative real or zero}>\)

An example clock specification is

\[
\text{XYZ.C4-6 L}
\]

which says that the signal goes from a high to a low at time 4, and from a low to a high at time 6. The time units that the clocks are specified in are normally set to a fraction of the cycle time. For
example, one eighth of the cycle is the period used in the design of the Mark IIA processor. Specifying clocks as fractions of the cycle time rather than in absolute time units allows the relative timing within the design to scale nicely if the cycle time is changed. The signal

\[ \text{XYZ}_C2-3,5-6 \]

is a high from 2 to 3 and from 5 to 6, and is a low for the rest of the time. If a single time is given instead of a range, then a range of one clock unit is assumed. For example,

\[ \text{XYZ}_C2.5 \]

is equivalent to the previous signal.

The other type of assertion states when a given signal is stable, and when it may be changing. Its general form is

\[ \text{SIGNAL NAME } S <\text{value specification}> \]

For example, the name XYZ .S4-8 says that the signal is stable from time 4 to time 8, and may be changing during the rest of the cycle.

This type of assertion has several uses. First, it allows the designer to specify his assumptions about when signals are valid (i.e., not changing) as he creates them, and those assumptions will be used by the Verifier until the signals are generated. For generated signals, the assertion is checked against the timing of the actual generated signal, and an error message is generated if the assertion is violated. In the design of the Mark IIA processor, most signal names have stable assertions in them. This greatly improves the readability of the logic, since a signal name very explicitly states when the signal is valid.

Putting the stable assertion on interface signals is the key to the ability to verify a design in sections. After each section is verified, SCALD checks to see that the interface signals have the same assertions. If no section has an error, and all of the interface signals have the same assertions, then the entire system must be free of errors.

3.4 Evaluation Directives

Evaluation directives tell the Timing Verifier how to evaluate certain gates. They can also specify the exact point in a circuit at which a clock is adjusted to some specified time.

Because the Verifier is not doing full logic simulation on a completed design, it does not know the logic level of most signals, but only whether the signals are stable or changing. Consider the circuit shown in Figure 1. The clock signal "CK .P2-3 L" is being ANDed to the control signal "WRITE .S0-6 L" to generate a write enable pulse for the RAM array. If the data is stable every cycle during the period that the RAM is to be written, then the most efficient way to check for timing errors is just to analyze the case in which the signal "WRITE .S0-6 L" enables a write. The
"&H" directive shown at the end of the clock signal says to ignore the value of the "WRITE S0-6 L" signal, allowing the clock signal always to propagate through the gate. In addition, it says the timing specified by the clock signal is to be adjusted so that it refers to the time at which the output, rather than the input, of the gate changes. The "&H" directive also specifies to check that the control signal "WRITE S0-6 L" is stable during the period that the clock is asserted, to ensure that the write will be either solidly enabled or solidly disabled.

There are a number of different directives along the lines of the "&H" directive. For example, the "&Z" directive on the signal "CK .P0-4" states that the clock timing refers to the time the output of the gate changes. If multiple directives are given after a signal, such as "&HZ", then the first letter refers to the first level of gating and the second refers to the second level of gating after the directive. There is no limit on the length of a directive string.

4 Design Specification

Figure 1 shows a circuit example specified in the SCALD Hardware Description Language. The circuit consists of a 16-word by 32-bit RAM, a 32-bit register, a 2-input multiplexer and several gates. This design description is entered into the computer via an interactive graphic editor, and forms the data base that drives the entire SCALD system.

A detailed description of the basic SCALD language can be found in [2,3,4], and will not be repeated here. The main points of interest to the Timing Verifier in Figure 1 are the assertions on signals, evaluation directives, and the specification of possible wire delays. The assertion on the signal "W DATA S0-6<0:31>" says that it is stable from time 0 to time 6, allowing the Verifier to check the timing of this circuit without knowing how the signal is generated. The assertion on the clock signal "CK .P2-3 L" says that it is low between times 2 and 3, and high for the rest of the cycle. The signal "ADR<0:31> [0.06:0.07]" states that the address wires on the RAM can be between 0 and 6 nsec long. The evaluation directives "&H" and "&Z" have already been described.

5 Chip Definitions

For each chip used in a design, a definition of its timing and logical properties is given in the SCALD Hardware Description Language.

A chip is defined in terms of a set of primitive functions that the Verifier understands. These primitives include AND, OR and CHANGE gates, registers, latches, multiplexers, a setup and hold checker, and a minimum pulse width checker. Two example chip definitions are shown in Figures 2 and 3. Figure 2 shows the definition of a 10145A, a 16-word RAM. Figure 3 shows the definition of a 10158, a 2-input multiplexer. The 10145A model is a timing model only. The "3 CHG" and "4 CHG" gates are CHANGE gates, which output the value CHANGE when any of their inputs change, and output the value STABLE the rest of the time. Using CHANGE gates has been found to be invaluable in modeling complex functions for which knowledge of the exact logical operation is unnecessary. The model for the 10158, on the other hand, is an accurate model, which could be used to do full logic simulation. For the 10158, the model of its complete logical operation is necessary to verify timing constraints in many circuits.
6 Wire Delay Estimates and Calculations

Before the actual wire delays are known, the Timing Verifier uses a rule to estimate them, except where they have been specified by the designer. After the routing of the wires is known, accurate wire delays are calculated. This is done by the SCALD Physical Design Subsystem [3, 4], which then feeds them back into the Timing Verifier, which does a detailed check of the timing.

7 Circuit Verification Example

Figures 4 and 5 show the results of running the circuit shown in Figure 1 through the Timing Verifier, with a specified cycle time for the circuit of 50 nsec, and a default wire delay of 0 to 2 nsec. Figure 4 gives a complete listing of all of the signals, showing their value as a function of time. Consider the first signal in the list, "ADR<0:3>". It has the same value for all of its bits, and so has only one value given. It is stable at the beginning of the cycle, and starts changing 0.5 nsec into the cycle. It is then changing from 0.5 nsec to 5.5 nsec, at which point it goes stable until 25.5 nsec. It is then changing from 25.5 nsec into the cycle until 30.5 nsec. It then goes stable from 30.5 nsec until the end of the cycle.

Figure 5 lists the setup and hold time errors. Because of the long wire specified on the signal "ADR<0:3> [0.0.6.0]", two setup time errors are generated. The first error message shows the address on the RAM just going stable at time 11.5, the same time as the write enable (WE) signal starts rising. Since a RAM requires a setup time of 3.5 nsec, the wire delay on the address signal must be reduced to 2.5 nsec in order to eliminate the error. The second error message lists the data being read out of the RAM going stable at time 47.5 nsec, and the clock starting to rise at time 49.0 nsec, giving only 1.5 nsec of setup time instead of the required 2.5 nsec.

8 Correlations Within Digital Systems

Consider constructing an 8-bit shift register using two 4-bit shift register chips. The shift output of one chip must be connected to the shift input of the other chip. If the minimum delay from the clock to the shift output is not greater than the hold time on the shift input by at least the maximum skew possible between the two clock inputs, then there is a timing problem. The key to checking this timing constraint is to calculate the maximum skew between the two clock inputs, taking into account any correlations within the circuit.

Now consider the case where there is a large amount of skew on the time at which the clock signal will occur, but that the two clock inputs are wired together with a short wire. The skew that each chip sees is large, but the maximum possible skew between the two inputs is quite small, because of the correlation between the two clock inputs. To analyze this case correctly, the Verifier needs to understand the correlation between the two clocks. The approach that the Timing Verifier has taken is to make the designer explicitly declare this correlation in the design specification, relieving the system of the burden of automatically finding it. Since this type of correlation tends to occur in only a few simple macros (defining items such as shift registers and counters), it seems a small burden to declare when a circuit is counting on a correlation to work properly.
Verification of the S-1 Mark IIA Processor Timing

The Timing Verifier has been used to check all of the timing constraints in the S-1 Mark IIA processor design. The design was checked in two parts, consisting of the instruction and operand preparation unit and the instruction execution unit. Each of these units consists of approximately 5,000 MSI and LSI ECL chips. The verification of one of these units takes about 12 to 15 minutes of execution time and 6 to 8 megabytes of storage on the S-1 Mark I processor. The Mark I processor is the first generation S-1 processor which was designed with SCALD, and is comparable in performance to an IBM 370/168.

The verification of the timing constraints proceeded on a daily basis. Each day the Timing Verifier was run to find any timing errors which might have been introduced during the previous day’s work into the design. This allowed errors to be found and fixed as they were introduced into the design, before their effects could propagate throughout the design. This early continuous feedback was found to be invaluable in timely completion of a design with a high degree of confidence in its operability.

Conclusions

The SCALD Timing Verifier has been a very efficient way of discovering timing errors in large digital systems. It is efficient from the standpoint that it requires little more effort from the designer beyond what is required to do the basic design. It is also computationally efficient, allowing a large design to be verified in a relatively small amount of computer time.

Once the timing constraints have been verified, then a simple logic simulator which does not have to worry about timing can find the highly probable logic errors. The less probable logic errors can be found either by a hardware simulator or prototype. The timing in both the prototype and the final implementation can then be checked with the Timing Verifier.

Acknowledgments

I would like to thank Forest Baskett and Bill vanCleemput for the constant support and guidance they have provided throughout the course of this research. The Fannie and John Hertz Foundation’s gracious support has provided me the freedom to pursue this research. Bill Bryson, Mike Farmwald, and Jeff Rubin have been the first users of the SCALD Timing Verifier and their patience and many suggestions for improvement are most appreciated. My thanks also go to the Office of Naval Research and the Naval Electronics System Command for the support for the S-1 Project which has provided the necessary environment for this research, and to Curt Widdoes and Lowell Wood, whose tireless efforts have made the S-1 Project possible.

References


Figure 1 – Example circuit to be verified.
Figure 2 - Definition of the 10145A, a 16-word random access memory chip (RAM).
Figure 3 – Definition of the 10158, a two input multiplexer chip.
Values of all signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Value</th>
<th>Constant Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADR&lt;0:3&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK.P0-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK.P2-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK.P4-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTPUT&lt;0:31&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM&lt;0:31&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ ADR.S4-9&lt;0:3&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REG CLK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W DATA.S0-6&lt;0:31&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE.S0-6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE ADR.S0-6&lt;0:3&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

S:0.0, C:0.5, S:5.5, C:25.5, S:30.5 (constant value)
R:0.0, I:1.0, F:24.0, 0:26.0, R:49.0 (constant value)
0:0.0, R:11.5, I:13.5, F:17.8, 0:19.8 (constant value)
F:0.0, 0:1.0, R:24.0, 0:26.0, F:49.0 (constant value)
S:0.0, C:0.5, S:7.5
S:0.0, C:5.0, S:20.5, C:30.0, S:45.5
S:0.0, C:6.3, S:25.0
R:0.0, I:1.0, F:24.0, 0:26.0, R:49.0
S:0.0, C:37.5
0:0.0, R:11.5, I:13.5, F:17.8, 0:19.8
S:0.0, C:37.5
S:0.0, C:37.5

Figure 4 - Output from the Timing Verifier showing values of signals.
Setup, Hold and Minimum Pulse Width errors . . . .

Setup time error; Setup Time = 3.5, Hold Time = 1.0
CK INPUT = WE
DATA INPUT = ADR

Setup time error; Setup Time = 2.5, Hold Time = 1.5
CK INPUT = REG CLK
DATA INPUT = RAM

Figure 5 - Setup and hold errors found by the Timing Verifier.