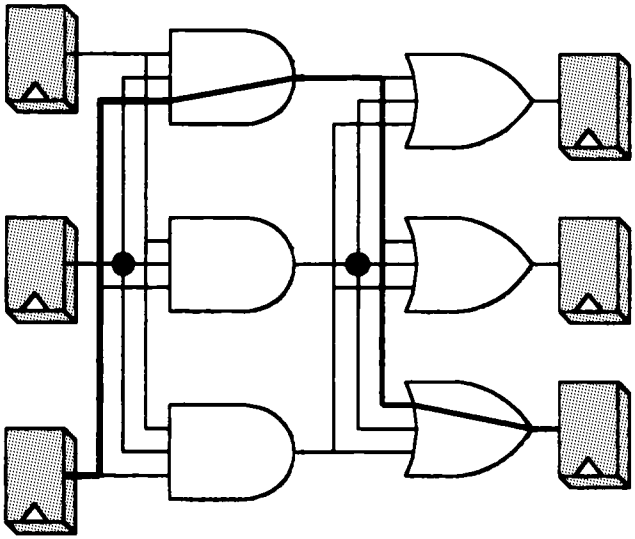

3 Timing in ASICs

3.1 Introduction

The number of ASICs designed increases every year. Advances in technology allow more transistors to be packed onto a single die which expands the applications where they can be used and accelerates development. Successful development of an ASIC depends on accurate modeling of its operation. Designing a circuit to be logically correct is simple. Producing an accurate timing model is critical to successful development. Current methodologies for generating accurate timing models for ASIC designs are described here.

Integrated circuits start as computer representations of a physical device. The designer's goal is to model the device characteristics with sufficient accuracy that actual silicon behaves as the model predicts, assuming the computer simulations exercise the model in the same way the device is expected to operate in the real world. Modeling a device's logical operation is relatively simple, and the translation from the model to the physical would be easy if it were not for the major difference introduced during fabrication: timing delays. The conversion of a logic statement to a model of its physical implementation is shown in Figure 3.1. The operation of the circuit

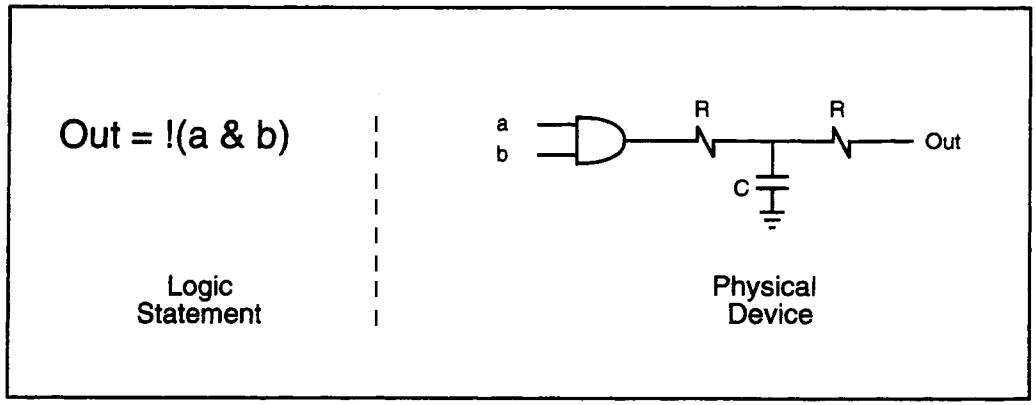

Fig. 3.1 Accurate Models Require Inclusion of Parasitic Capacitors

in Figure 3.1 is affected by the charging and discharging of the parasitic capacitor through resistors, both of which are inherent to silicon physical implementation. The stray capacitance and resistance can have such a great and deleterious effect that the physical operation is nothing like the simulated logical model.

A circuit's correct operation can be assured only if the timing of the simulated model is a close approximation of the final device. The accurate modeling of delay is of major importance. As process geometry shrinks and the number of transistors per die increases, the task of modeling the effects of parasitic capacitance and resistance makes it more challenging to correlate prelayout to postfabrication timing. Fortunately, CAD tools exist to accurately estimate delays before layout and extract the capacitance and resistance once layout is complete. Modeling estimated and extracted delays plays an important part in guaranteeing the timing and operation.

Any delay value used before the device is fabricated is merely an estimate. The four sources of delay are shown in Figure 3.2. Gate delay is determined by input slew rate and the inherent RC loading of the gate. Delay through a line depends on the RC load the gate drives. The fanout load simply increases the capacitance the driver must charge and discharge. Methodologies for predicting delay are well established. Gate delay is measured from fabricated test struc-

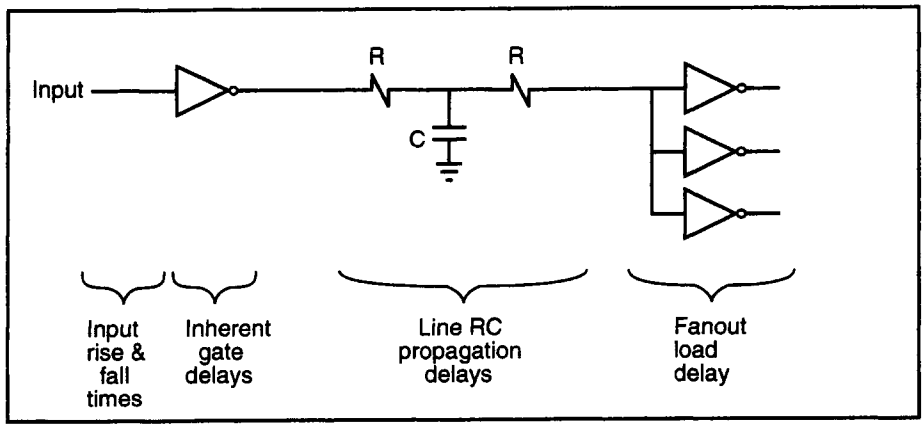

Fig. 3.2 Components of Circuit Delay: Input Slew Rate, Inherent Gate Delay, Line Propagation Delay, Fanout Load

tures tested at specific operating points. A transistor's speed, and therefore the inherent delay in a gate, is affected by its dimensions, the supply voltage, doping levels, input slew rate, operating temperature, and fanout load. The data measured from the test structure provides a device model that extrapolates to estimate delay under all operating and fabrication conditions.

The delay due to signal lines may be modeled in two stages: prelayout and postlayout. In either case, the physical characteristics of fabricated traces are known, having been measured from test structures. In the absence of layout, the unknown elements that affect timing are the trace's length, width, and surrounding signals. Figure 3.3 shows the parasitic capacitors seen by a metal trace. Parasitic capacitance is explored in detail in section 3.3. Before the layout is completed, any delay attributed to a signal line is an estimate based on probable length and width of the trace. Since the actual path is not known, the length is simply a guess based on the size of the overall circuit and the probability of placing the output of one gate close to the input terminals of the gates it drives. Another unknown aspect of the trace is the topology over which it passes. Once the layout is finished, the trace lines, and therefore their

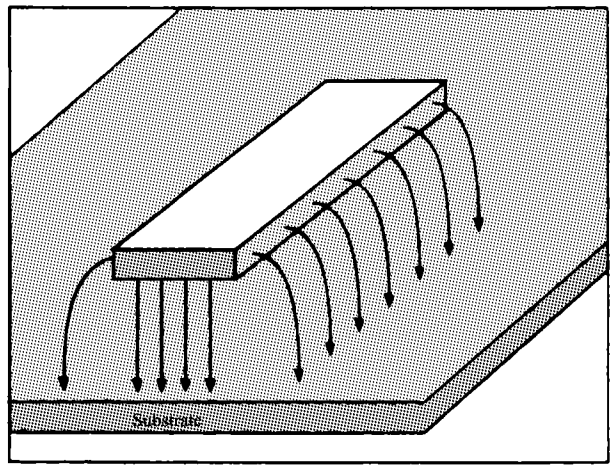

Fig. 3.3 Parasitic Capacitance of a Metal Line

delay, can be accurately modeled. The layout fixes their length and reveals what lies under the trace, whether it is substrate, transistors, or other layers.

Delay estimations are made in all stages of design: prelayout, synthesis, and postlayout. The most common methods used to estimate delays at all stages of the design cycle are explored.

3.2 PRELAYOUT TIMING

The design environment and methodology determine the accuracy and ease of modeling delays before the layout is finished. HDL languages, such as Verilog and VHDL, make it possible to add both gate and interconnect delays; however, except in situations where the layout is regular and known, such as in memories or decoders, the effects of delays due to interconnect are ignored until after synthesis or layout. The ease of including gate delays also depends on the type of model used. HDL modeling can also be done at two different levels: RTL and gate level.

3.2.1 RTL vs. Gate-Level Timing

RTL code models a logic function without regard to its implementation, whereas gate-level code specifies the exact gates required. Both the RTL and gate-level code for the logic function shown in Figure 3.4 are given in Example 3.1.

Example 3.1

RTL Code

```
out = ((a & b) | !a);
```

Gate-Level Code

```
and(a, b, s2);
not(a, s1);
or(s1, s2, out);
```

Modeling delay at the gate level is straightforward. The delay of each gate is found in the technology library. The appropriate delay can be assigned to every gate in the code and the propagation delay of signals estimated to provide a fairly accurate representation. However, manually implementing HDL code with delays for each gate is time consuming. At the prelayout stage, most design methodologies use synthesis to provide a gate model with delays while RTL code is used to model the circuit's behavior.

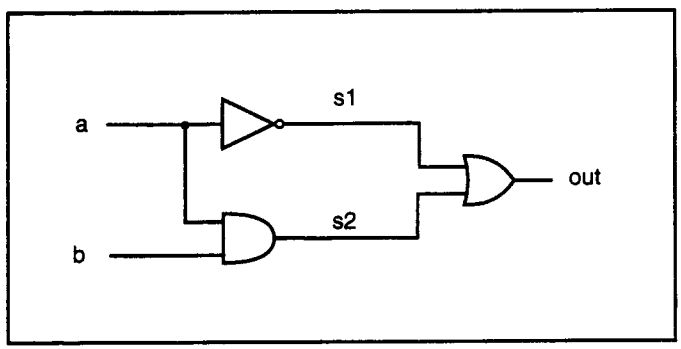

Fig. 3.4 Circuits Can Be Represented as RTL Code or Instantiated Gates

A clear method of accounting for delay is to determine the delay through each gate. The technology library already has delay information for every gate. Accurate modeling requires the assignment of the appropriate delay to each gate as described below. Estimating the delay of the RTL code is more difficult because of its level of abstraction. Until synthesis is complete, there is no straightforward way to correlate RTL code to actual gate delays.

The level of coding used affects the delays that can be modeled. Generally, RTL code is used to determine correct logical operation without regard for delays. A design at the gate level not only checks for correct operation, it also ensures that delays meet the required timing. Most designs start with an RTL code, then use synthesis to generate the gates needed to verify timing. Furthermore, few designs start at the gate level because the simulations, especially when timing is included, are very slow. Design at the RTL level offers a fast method to ensure that the logic is properly implemented. Synthesis then converts the design to gates that include delays from gates, estimated routing, and fanout.

3.2.2 Timing in RTL Code

Although it is impractical to assign delays to individual lines of RTL code, it is feasible to assign delays to entire modules. In RTL code, timing should be applied to any module or port that has a known response such as:

- Bus models
- Memories
- I/O ports
- Setup and hold times

A high-level system is shown in Figure 3.5. Each block is implemented as RTL. The RAM and the EPROM will not be synthesized. They are both modeled as an array of memory indexed by the address. The processor comes from a vendor's library. Its model reflects only the bus transactions that take place. The address-

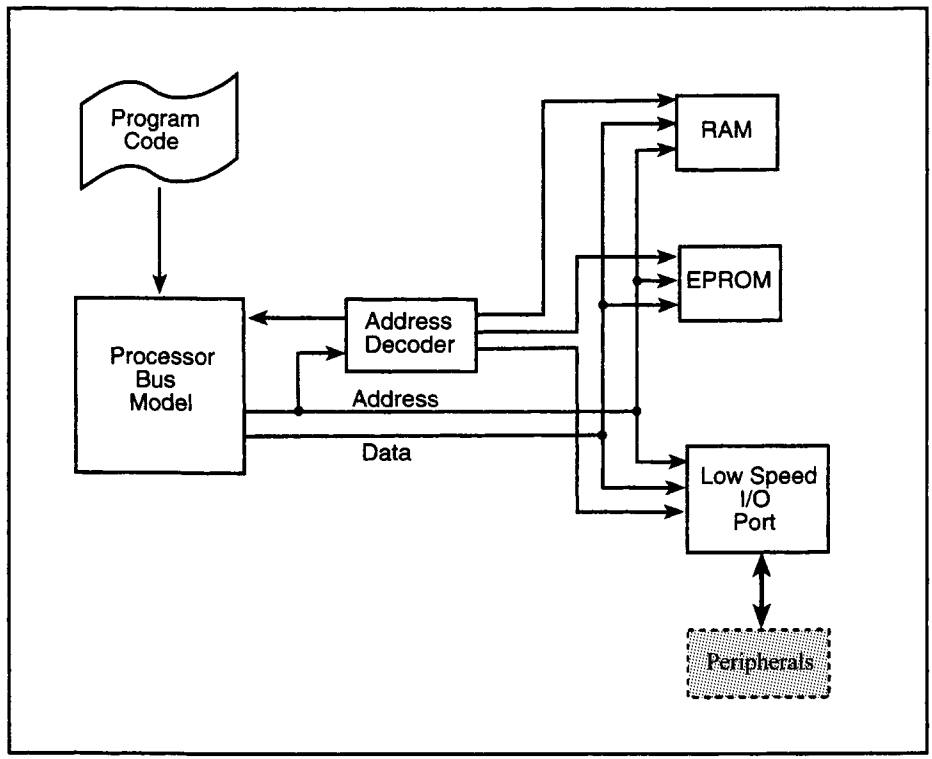

Fig. 3.5 Delays of Entire Modules Are Easy to Implement in an HDL

decode and low-speed I/O port will be synthesized and include any logic and flip-flops needed to perform their functions. Timing is important in the system simulation. At the RTL level, it is possible to see if the processor bus timing matches the RAM and EPROM timing. It can be determined if the decoder has too much delay or if the read/write timing of the I/O port meshes with the processor's requirements. The timing response of each block can be added to the model.

The read timing of the RAM is given in Figure 3.6. When the RTL model detects a read cycle, it can instantaneously get the data from its memory array and present it on the bus, but a fast response does not correspond to reality. The delay, shown in Figure 3.6 as Tvavd must be implemented in the model to reflect the time actually needed for the RAM to access and present valid data. The

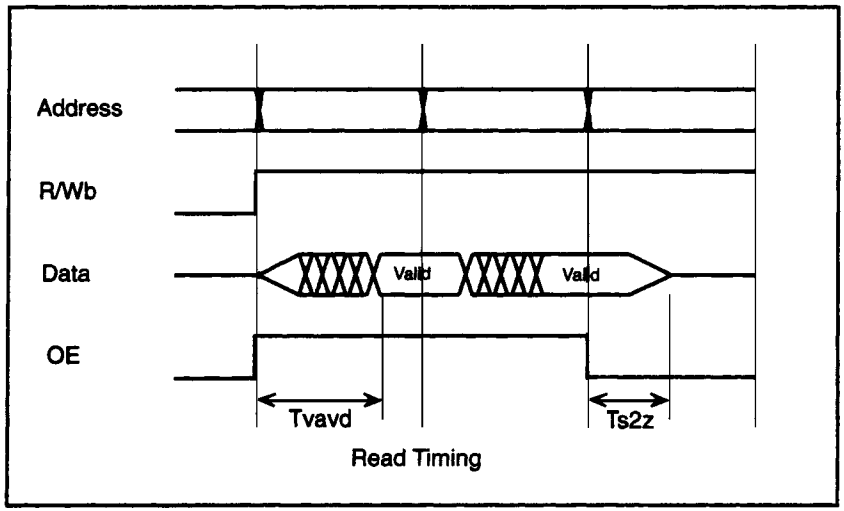

Fig. 3.6 RAM Timing Diagram

response time of the address decode cannot be instantaneous, but should reflect a delay based on the maximum delay it can have and still work in the system. The I/O port also needs bus timing to match the processor's characteristics. The processor model comes from the vendor with timing that matches the processor's real operation. The processor cycle time provides a check of the timing of all the other blocks. If a block meets the bus cycle time, it will work when fabricated.

A snippet of Verilog code, shown in Example 3.2, demonstrates how to implement the Tvavd and Ts2z delays in the memory model.

Example 3.2

```
1. 'define Tvavd   10   // data delay out of memory
2. 'define Ts2z     5   // delay of deselect to tristate
3. module RAM (addr, data, sel, rw);
4.     input [15:0]   addr;
5.     inout [15:0]   data;
6.     input      sel, rw;
7.     reg [15:0]   mem_array [0:65536], data_internal;
8.     // data bus tristate. Bi-directional.
9.     assign #Ts2z data = (sel) ? data_internal : 16'bz;
```

Example 3.2 (Continued)

```
10.    always @ (addr, rw)
11.    begin
12.       // read memory
13.       if ((rw === 1'b1) && (sel === 1'b1))
14.          #Tvavd data_internal = mem_array [addr[15:0]];
15.       // write memory
16.       if ((rw === 1'b0) && (sel === 1'b1))
17.          mem_array [addr[15:0]] = data;
18.    end
19. end module
```

Note when the memory is read, the assignment of the data from the array to the bus is delayed by the time Tvavd. The data bus response to the sel signal is also delayed by Ts2z. Whenever the timing of a module is known, it should be implemented in the RTL model; however, HDL languages offer different types of delays. It is important to understand how the delay is applied to ensure the model mirrors the real world. In Verilog, the two main default types are regular and intra-assignment. The effects of both types on continuous blocking, and nonblocking assignments are discussed below.

3.2.3 Delay with a Continuous Assignment Statement

The regular delay applied to the continuous assignment statement provides an inertial delay. An inertial delay means that the inputs must change and remain at their new values longer than the specified delay before the output is affected. A continuous assignment statement with a regular delay of 5 time units is shown in Example 3.3.

Example 3.3

```
Assign #5 sel = address15 | address16 | address17;
```

The output, sel, is simply the OR of the inputs address15, address16, and address17. Logically, whenever one of the address signals goes high, sel goes high; however, delay changes that fundamental assumption slightly. The relationship between the input and the output signals is shown in Figure 3.7.

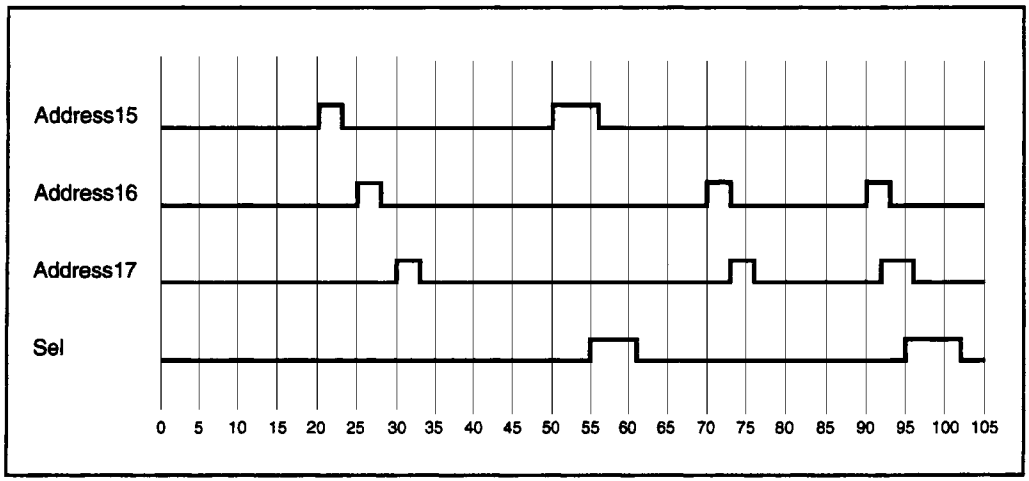

Fig. 3.7 Signals Corresponding to Example 3.3

At 20ns, each input sequentially goes high for 3ns. Each input stays high for less time than the specified delay of 5. The output does not change because the delay is inertial and no input is high-longer than the delay. At 50ns, address15 goes high for 6ns. After the input signal has been high for 5ns, the output responds and produces a pulse 6ns wide. At 70ns, both address16 and address17 go high for 3ns, but they are coincident and do not satisfy the inertial delay requirement, so the output does not change. At 90ns, a 3ns-wide pulse on address16 overlaps a 4ns-wide pulse from address17. The simulator interprets the overlap as meeting the delay requirement and a 7ns pulse occurs on the output.

Both the continuous assignment statement and the regular delay operate like combinatorial logic. Just as the delay through a gate suppresses glitches, so does the regular delay when used with a continuous assignment statement.

3.2.4 Delay in a Process Statement

Process statements, such as always or initial, support two types of assignment statements: blocking and nonblocking. The effects of