Chapter 6

The Parallel Technique

6.1 The Basics of the Algorithm

In Chapter 5 we learned that oblivious unit- and multi-delay simulation may require several gate simulations to be performed for a single gate. These extra gate-simulations significantly increase simulation time over oblivious zero-delay simulation. The Parallel Technique was introduced as a method of speeding up these additional gate simulations. Chapter 2 introduced the concept of bit-parallel simulation, using the extra bits in a word to perform useful work. The Parallel Technique uses bit-parallel simulation in a more sophisticated way to perform the additional gate simulations required by unit-delay and multi-delay simulation.

Rather than treating a word as a simple collection of bits, the parallel technique indexes each bit in a word, starting on the right with index 0. Figure 6-1 illustrates this indexing, assuming that a word contains eight bits. A longer word will have a correspondingly larger number of indices. Each index corresponds to one unit of simulated time. Bit zero corresponds to time zero, bit one corresponds to time one, and so forth. As in Event-Driven Simulation, these time units are of unspecified length, and are assumed to occur during the processing of a single input vector.

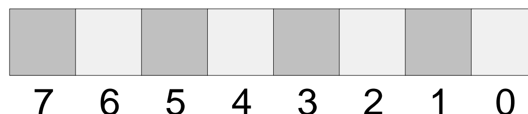

Figure 6-1. Assigning Times to Bit Positions.

Because the principles of the Parallel Technique do not depend on word-length, the term "Bit-Field" will be used to denote an indexed sequence of bits such as that pictured in Figure 6-1. For each circuit, there is a minimum acceptable bit-field width, M+1,

where M is the maximum level number over all gates in the circuit. Any bit-field width can be used as long as a bit-field is at least as wide as the minimum acceptable width. Bit-Fields are used to represent the history of a single net at all times during the simulation of a single input vector. The parallel technique requires one bit-field to be created for each net in the circuit. When a gate is simulated, the bit-fields corresponding to the gate-inputs are combined using bit-parallel operations, and the results are placed in the bit-fields corresponding to the gate outputs. To simplify matters, the discussion will focus on the Unit-Delay version of the Parallel Technique. Extensions to the multi-delay model are straightforward.

In most cases, the bit-parallel operations required to simulate a gate are nothing more than the operations used for zero-delay simulation. Figure 6-2 illustrates one such simulation. Note, however, that the low-order bit of the resultant bit-field corresponds to time 1 rather than time 0. In such cases, the index of the low-order bit is called the *alignment* of the bit-field. The change in alignment is due to the delay of the AND gate. Since the gate has a delay of 1, the input values at time 0 combine to form the output value at time 1, and so forth.

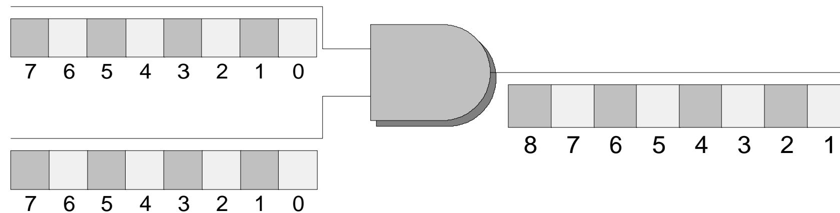

Figure 6-2. Parallel Technique Gate Simulation.

Because all bit-fields must be aligned to zero, it is necessary to correct the alignment of the resultant bit-field after simulating a gate. Bit-Field correction is illustrated in Figure 6-3. After AND gate is simulated by performing a bitwise AND operation on the two input bit-fields, it is necessary to shift the bit-field to the left one bit, and introduce a new value into the low order bit. The new value must correspond to the value of the net at time zero. Since time zero also corresponds to the value of the net before the first gate-simulation is performed, this new value must be the value of the net from the previous input-vector. For practical reasons, it is convenient to initialize the bit-field of each net, so that the value from the previous input vector is placed in bit-position zero. The remaining bit positions are initialized to zero. When a gate simulation is performed, the result of the bit-parallel operations is stored in a temporary variable. After the left-shift, the temporary bit field can be combined with the output bit-field by performing a bit-wise OR operation. The complete code for simulating the gate of Figure 6-2 is illustrated in Algorithm 6-1.

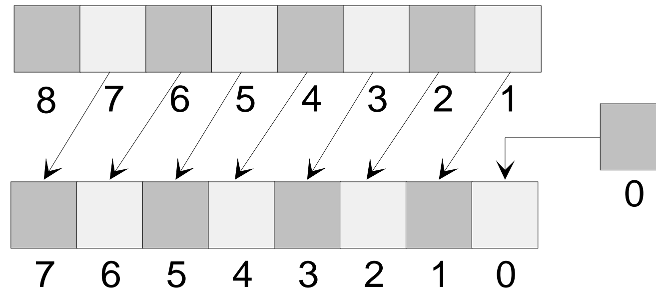

Figure 6-3. Bit-Field Realignment.

```
Temp := B And C
Temp := Temp ShiftLeft 1
A := A Or Temp
```

Algorithm 6-1. Parallel Technique Gate Simulation.

Gate simulations such as that pictured in Algorithm 6-1 are generated for each gate in the circuit in levelized order. The gate simulation code must be preceded by the net initialization code. Although net initialization appears complicated, it can be accomplished quite simply. When the simulation of an input vector is complete, the high-order bit of each bit-field is guaranteed to contain the final value of the net. This bit can be shifted into the low-order bit position. If a logical rather than an arithmetic shift is used, the vacated high-order bits will be automatically filled with zeros. Algorithm 6-2 illustrates the process of initializing a net, assuming that the bit-field contains eight bits.

```
A := A RightShift 7
```

Algorithm 6-2. Initializing Nets.

Algorithm 6-2 is not used for primary input nets. Primary inputs are assumed to achieve their new values at time zero and do not change their values during simulation. When a new value is assigned to a primary input, it is necessary to propagate the new input value through all bits of the bit-field. This is done using Algorithm 6-3.

```
If A ≠ 0 Then
   A := -1
EndIf
```

Algorithm 6-3. Initializing Primary Inputs.

Simulation using the parallel technique is illustrated in Figure 6-4 through Figure 6-7. It is assumed that this circuit has already been simulated with the input vector (A=0, B=1, C=1), and will now be simulated with the input vector (A=1, B=1, C=0). Since the maximum level number over all gates is two, a bit-field width of 3 is sufficient. It is possible to use a larger bit-field, but if this is done, all bits above the third will have the same value as the third bit.

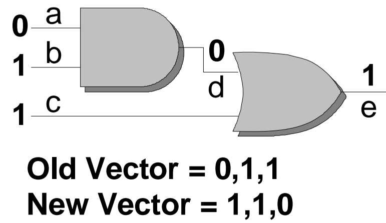

Figure 6-4. A Sample Circuit.

Figure 6-5 illustrates the bitfield initialization, which is the first step in the simulation of the circuit of Figure 6-4.

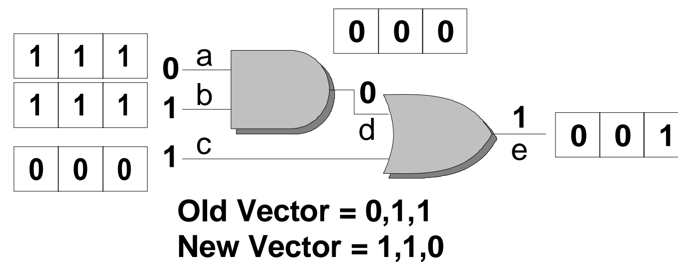

Figure 6-5. Initializing the Bit-Fields.

Figure 6-6 illustrates the simulation of the AND gate, the generation of the intermediate bit-field and the final adjustment of the bit-field.

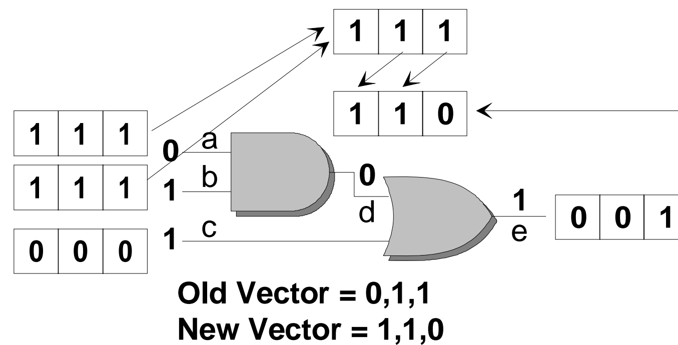

Figure 6-6. Simulating the AND Gate.

Figure 6-7 illustrates the simulation of the OR gate, which is the final step in the simulation..

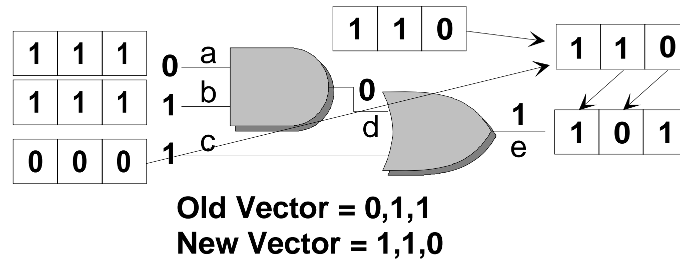

Figure 6-7. Simulating the OR Gate.

Note that the result of the simulation in Figure 6-7 shows a static hazard in the output of the OR gate. The generated code for the circuit of Figure 6-4is given in Algorithm 6-4. It is assumed that the input values have already been read, and placed in the low-order bit of the variables corresponding to the primary inputs.

```
if a <> 0 then
    a = -1;
endif
if b <> 0 then
    b := -1;
endif
if c<> 0 then
    c := -1;
endif
temp = a AND b;
d = (temp ShiftLeft 1) OR d;
temp = d OR c;
e = (temp ShiftLeft 1) OR e;
```

Algorithm 6-4. Simulation of a Simple Circuit.

6.2 Bit-Field Alignments

Apart from the **shift** and **or** operations, the simulation code generated for the parallel technique is identical to that of Zero-Delay Levelized Compiled Code simulation. If it were somehow possible to eliminate the shift operations, unit-delay simulation could be performed as quickly as zero-delay simulation. To investigate this concept further, consider the diagram of Figure 6-8. This diagram represents the output bit-field right after simulation is performed, but before the shift has been done. The intermediate output must be shifted to the left one bit to make room for the time-zero value of the net. But, is this value really needed? If the answer is NO, then the shift and OR operations can be dropped, and the alignment of the bit-field can be left at one.

It is useful to list the conditions under which the time zero value would be needed. If there was a change in the net at time one, it would be necessary to retain the time-zero value to show the change. Also, if some other gate simulation used the net as an input, the time-zero value might be required to properly simulate a subsequent gate. If the net

does not change at time one, then the time zero value might not be required. To determine whether the net will change at time one, it is sufficient to compute the PC-Set of the net, as described in Chapter 5. The minimum value of the PC-Set, also known as the minlevel of the net, is the earliest time at which it is possible for the value of the net to change. If a change does indeed occur at this time, it is necessary to retain the net-value from the previous time-unit, so that the change is visible.

The maximum value of the PC-Set of a net corresponds to the level of the net. Using the level and the minlevel of a net, it is possible to compute both the alignment and width of the bit-field. The alignment of the bit-field for the net must be no larger than the *minlevel*-1. The width of the bit-field must be no smaller than *level*-*minlevel*+2. If these two requirements are met, then the alignment and width of the bit-field will allow all changes in the output net to be simulated.

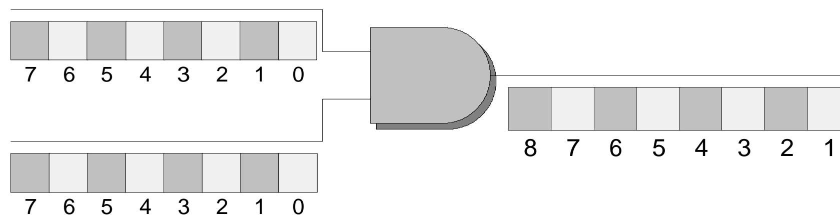

Figure 6-8. Changing Alignments.

To eliminate all shift operations from a simulation, the following conditions must be met.

1. **The alignment of the bit-field for a net must be no larger than *minlevel*-1.**
2. **The width of the bit-field for a net must be no smaller than *level-minlevel*+2.**
3. **For a particular gate, the alignments of all input nets must be the same.**
4. **The alignment of the output of a gate must be one larger than the alignment of the inputs.**

Unfortunately, as Figure 6-9 illustrates, it is not always possible to enforce conditions 3 and 4 simultaneously.

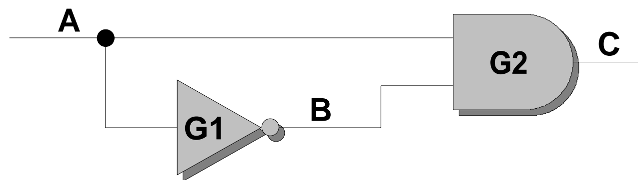

Figure 6-9. Alignment Conflicts.

Condition 3 requires net A and net B to have the same alignment, while condition 4 requires the alignment of net B to be one larger than that of net A. These two conditions cannot both be true. Because conditions 3 and 4 cannot be enforced for this circuit, it is impossible to eliminate all shifts. Most circuits require the retention of some shift operations. To determine whether a circuit requires shifts, it is necessary to create a graph known as the undirected network graph. Each vertex in the undirected network graph

represents either a net or a gate. There is an edge between the vertex for net N and the vertex for gate G, if N is either an input or output of G. The edge is undirected. The following is the undirected network graph for the circuit of Figure 6-10.

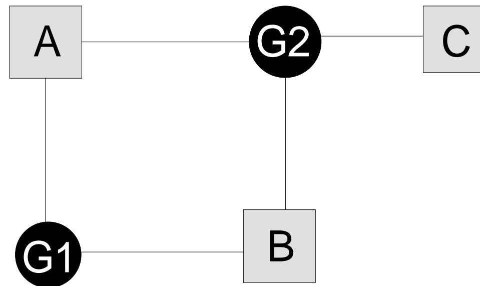

Figure 6-10. The Undirected Network Graph.

Note that the graph of Figure 6-10 is cyclic. Cycles in the undirected network graph are a necessary, but not sufficient condition for the retention of shifts. If the undirected network graph is acyclic, all shifts can be eliminated from the circuit. If the graph is cyclic, it is necessary to compute the weight of each cycle to determine whether it is possible to eliminate all shifts. The weight of a cycle is calculated by traversing the cycle in either direction, starting with a net-vertex. A weight of zero is assigned to each net-vertex. The weight of a gate-vertex must be calculated dynamically during the traversal of the cycle, and must be recomputed for each new cycle.

A traversal of a gate vertex is a sequence of three vertices N1, G, N2, starting and ending on two different net vertices. N1 and N2, with the gate vertex, G, in the middle. In Figure 6-10, an example of a gate traversal is A,G1,B. The nets N1 and N2 must be inputs or outputs of G. If N1 and N2 are both inputs or both outputs then G has the weight 0. If N1 is an input and N2 is an output then G has the weight 1. If N1 is an output and N2 is an input then G has the weight -1. The total weight of a cycle is the combined weight of all gate traversals in it. In Figure 6-10, the cycle A,G2,B,G1,A has weight -1, because in the traversal A,G2,B, G2 has weight 0, and in the traversal B,G1,A, G1 has weight -1.

If the undirected network graph of the circuit contains a cycle of non-zero weight, then it will be impossible to remove all shifts from the circuit. The converse is also true. If all cycles in the undirected network graph have zero weight, then it is possible to eliminate all shifts from the circuit. Because the undirected network graph of Figure 6-10 has a cycle of non-zero weight, it will be impossible to eliminate all shifts from the simulation of the associated circuit.

The cycle in the graph of Figure 6-10 is due to the reconvergent fanout in the circuit of Figure 6-9. However there are circuits without reconvergent fanout that have cycles of non-zero weight in their undirected network graphs. Figure 6-11 gives an example of such a circuit.

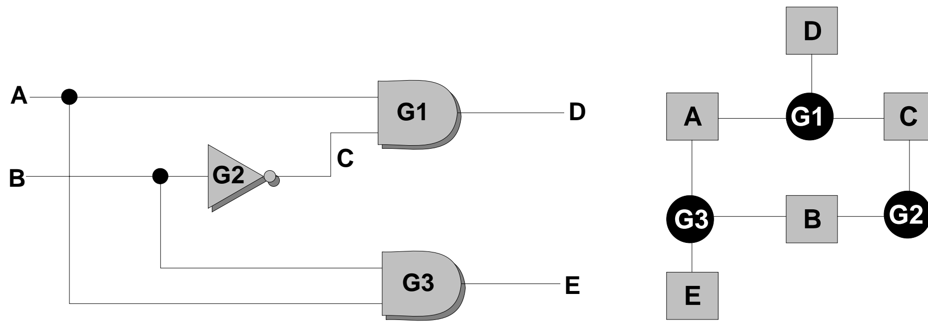

Figure 6-11. A One-Shift Circuit.

6.3 Eliminating Shifts

The trick to eliminating shifts is adjusting the alignments and widths of bit-fields to reduce the number of bit-field realignments. For any circuit, it is advantageous to have all bit-fields the same width, so it is necessary to compute the maximum of the required bit-field width over all nets N. This is the maximum of $\mathbf{L_N}$-$\mathbf{M_N}$+2, where $\mathbf{L_N}$ is the level of net N, and $\mathbf{M_N}$ is the minlevel of net N. This value will be used as the bit-field width for all bit-fields.

Net alignments are set using a procedure called the *path-tracing* algorithm. Gates are processed starting with the primary outputs, and proceeding backwards through the circuit to the primary inputs. For each primary output net **N**, the alignment is set to $\mathbf{M_N}$ - 1. Gates are processed in reverse order by level. When a gate is processed, the alignment of each input net is set to one less than the alignment of the output net. In some cases, a net will already have an alignment when this procedure is executed. If this is the case, the new alignment replaces the old alignment *only* if the new alignment is less than the old alignment. Algorithm 6-1 shows the path-tracing algorithm, while Figure 6-12 gives an example of computing alignments using the path-tracing algorithm. The alignment initialization value for non-primary outputs can be any number larger than the maximum level in the circuit, such as the total number of nets and gates in the circuit.

```
Compute N.PCSet for Each Net N in the circuit.
For Each primary output P do
    S := P.PCSet;
    L := Max(S);
    M := Min(S);
    P.Align := L-M+2;
EndFor
For Each non-primary output net N in the circuit do
    N.Align := GateCount+NetCount;
EndFor
For Each Gate G in Descending order by Level do
    N := G.OutputNet;
    NewAlign := N.Align - 1;
    For Each Input X of G do
        If X.Align > NewAlign Then
            X.Align := NewAlign;
        EndIf
    EndFor
EndFor
```

Algorithm 6-5. The Path Tracing Algorithm.

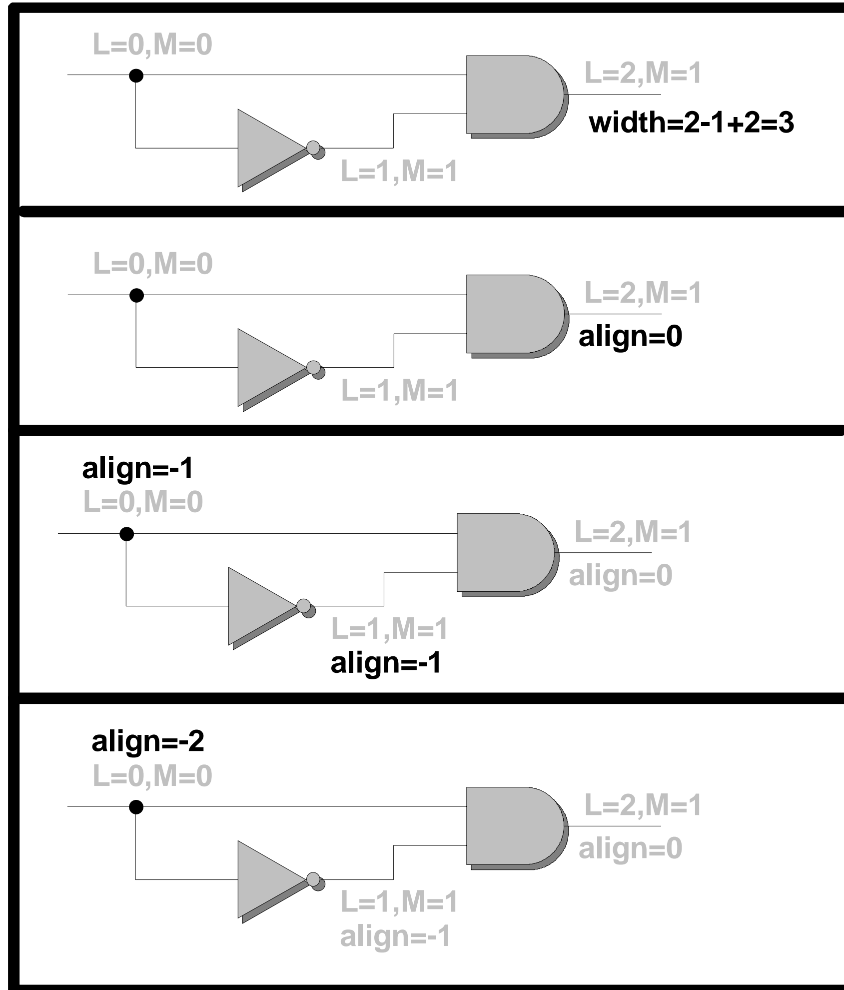

Figure 6-12. The Path-Tracing Algorithm.

A retained shift could be either to the right or to the left, and multiple-bit shifts may be necessary. It is also possible to create alignments that are less than zero. Since any time-unit less than zero represents a time before the simulation begins, all bit-positions that have negative indices must contain the final value of the net from the previous input vector. This is true for primary inputs as well as other nets. Because primary inputs are assumed to change at time zero, any bit-positions with negative indices must contain values from the previous input vector. When new primary input values are read, the new value is propagated through all non-negative bit positions of the primary-input bit-field. The value read from the previous input vector is placed in all negative bit-positions of the bit-field.

The Path-Tracing algorithm has several desirable properties. If G is a gate with output N, and the alignment of N is A, then the alignment of the inputs of G will be A-1 or less. This implies that the bit-fields for the inputs of G will contain bits corresponding to time N-1, which are the values needed to compute the value of the low order bit of N. This implies that the initialization of intermediate net values is unnecessary. Another important property, is that the width of the bit-fields will never be larger than those for the unoptimized circuit. When path-tracing is used, it is never necessary to shift the bit-

field after performing a gate simulation, but it is often necessary to align some input bit-fields before a gate simulation occurs. If the alignment of a gate output is A, all inputs must be aligned to A-1. If the any input has a different alignment, a shift must be performed to correct it. Figure 6-13 illustrates simulation with negative alignments.

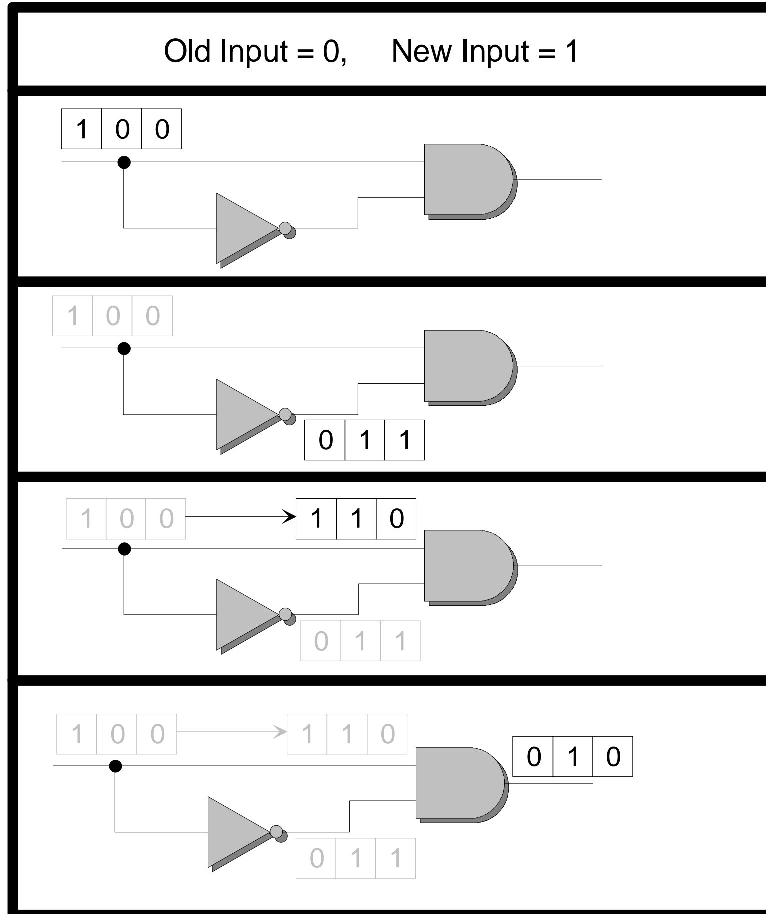

Figure 6-13. Simulating with Negative Inputs.

Arithmetic right shifts are used to align bit fields so the high-order bit of the bit-field is replicated throughout all vacated bit positions. Left-shifts are never required. For intermediate nets, it is never necessary to retain values from the previous input vector. If such values are required, they will be recomputed as needed.

Although the path-tracing algorithm can eliminate many shifts from a parallel-technique simulation, it is not optimal with respect to removing shifts. The circuit of Figure 6-14 has an acyclic undirected graph, so all shifts can be eliminated from its simulation. As Figure 6-15 shows, the path tracing algorithm aligns nets so that one shift is required. When simulating G6, it is necessary to shift input I1 by two bits. Nevertheless, it is possible to eliminate all shifts from this circuit, as the alignments of Figure 6-16 show. Eliminating this extra shift is not necessarily desirable, because the choice of alignments can affect the width of the bit-field. The path tracing algorithm is guaranteed not to expand the bit-field beyond the maximum of $\mathbf{L_N}$-$\mathbf{M_N}$ +2 over all nets N, however this is not true if other alignment procedures are used. The maximum bit-field

width in both Figure 6-15 and Figure 6-16 can be determined from the alignment of the net I2. For a primary input, there must be at least one bit position with a non-negative index, so in Figure 6-15 the bit-field of I4 requires four bits, but in Figure 6-16 it requires six. In practical applications, the expansion of the bit-field can be a serious matter. On a computer with 32-bit words, expansion of the bit-field from a value less than 32 to a value greater than 32 will have a significant impact on the performance of the simulation. Each bit-field will require two words instead of one, doubling the number of required bit-parallel operations. Shifts will be significantly more complicated because bits will need to be shifted from one word to another.

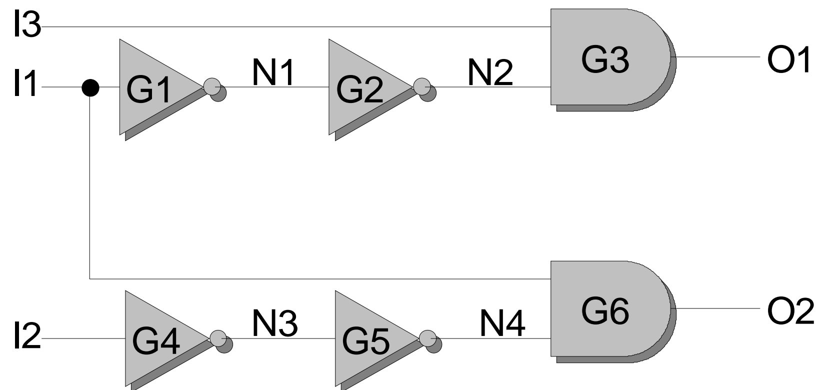

Figure 6-14. Non-Optimal Path-Tracing.

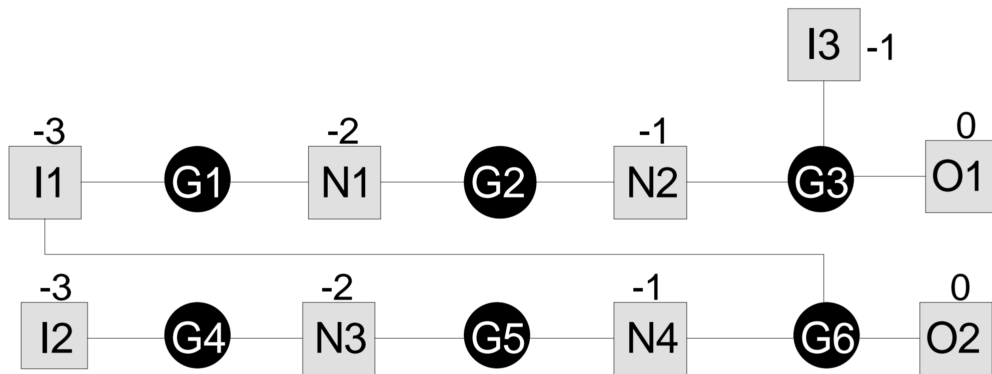

Figure 6-15. An Acyclic Undirected Network Graph.

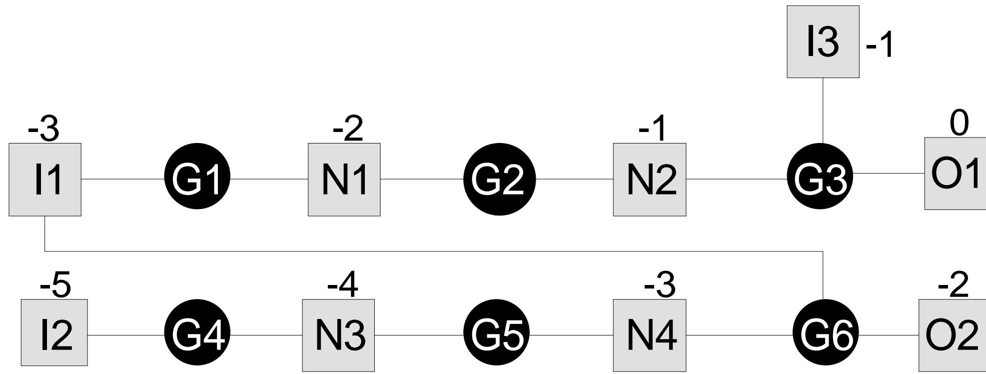

Figure 6-16. Expanding the Bit-Field.

6.4 Bit-Field Trimming

For circuits with bit-fields wider than the maximum word-size of the processor (usually 32 or 64 bits), bit-field trimming can be used to enhance simulation performance. Bit-field trimming exploits the relationship between the bit-field and the PC-Set of a net. Suppose a circuit has a bit-field width of 32 bits, and that net A has a PC-Set of {3,7,9,22,27}. The bit positions 3, 7, 9, 22, and 27 are called *PC-Set Representatives*. Although all 32 bit-positions are simulated during a Parallel-Technique simulation, only the PC-Set Representatives are important, because this is where changes occur. After the final value of net A has been computed, bit positions 0-2 will have identical values, as will positions 3-6, 7-8, 9-21, 22-26, and 27-31. xxx illustrates this idea.

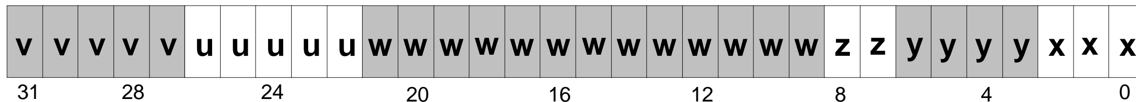

Figure 6-17. PC-Set Representatives.

When the bit-field of a net requires more than one word, those words that contain no PC-Set representatives are called *bit-field gaps*. There are three types of gaps, *low-order gaps* which contain the initial value of the net in every bit, *high-order gaps*, which contain the final value of the net in every bit, and *intermediate gaps* which contain an intermediate value in every bit.

No simulation code of any kind is required for high-order gaps. The high-order words can usually be eliminated from the simulation. Low-order gaps can be initialized, but no simulation code is needed for them. If the next highest word also contains the initial value of the net in the low-order bit, then it may also be possible to eliminate the initialization code. The value of an intermediate gap can be computed by propagating the high-order bit of the next lowest word through every bit, by using n arithmetic right-shift of 31 bits. This technique can be used to recover the value of high-order gaps, if needed. No simulation code is required, and the word does not need to be shifted after simulation. The value of a gap is needed only when the bit-field is being combined with another bit-

field that does not have gaps in the same position. If two gaps are being combined, the result will be another gap, and no simulation code is required.

6.5 Asynchronous Sequential Circuits

Because the parallel technique uses levelization, it is essentially a combinational technique. The algorithm can be extended to synchronous combinational circuits by breaking the circuit at the flip-flops as outlined in Chapter 2. This has the effect of converting the circuit into a combinational circuit which can be handled in the obvious manner. This technique will not work for asynchronous sequential circuits, because the changes in the flip-flops are not synchronized with a clock pulse and may change state more than once during the simulation of an input vector.

Asynchronous circuits can be handled using a technique similar to the forced levelization technique of Chapter 2. The first step in this procedure is to identify the strongly connected components of the circuit, which is a set of gates S, such that for any two gates G_1 and G_2 in S, the output of G_1 depends on the output of G_2 and the output of G_2 depends on the output of G_1. Stated more simply, two gates are in the same strongly connected component if there is a circular logic path that includes both gates. Finding strongly connected components is a standard problem in the theory of directed graphs, and the graph-theoretical algorithms for this problem can be adapted to finding the strongly connected components of a circuit.

The graph of Figure 6-18 has four strongly connected components, namely {A1, A2, A3}, {B1, B2, B3}, {C1, C2, C3, C4, C5} and {D1, D2, D3, D4, D5}. Once the strongly connected components of a graph have been identified, each component can be collapsed into a single vertex, as illustrated in Figure 6-19. If there is an edge between the vertices of two different strongly connected components, then there must be an edge between the corresponding vertices in the collapsed graph. The collapsed graph is always guaranteed to be acyclic.

The graph-based algorithms can easily be adapted to handle logic-level circuits. After assigning each gate to a strongly connected component, each of these components can be collapsed into a sub-circuit. (Many of these sub-circuits will contain a single gate.) The circuit is levelized with respect to the sub-circuits, and simulation code for each sub-circuit is generated in levelized order.

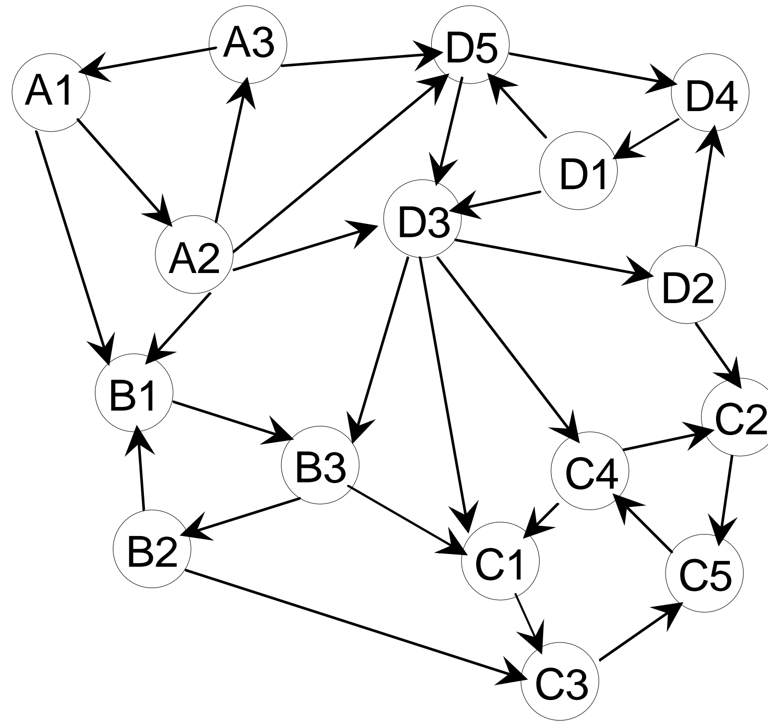

Figure 6-18. Strongly Connected Components: An Example.

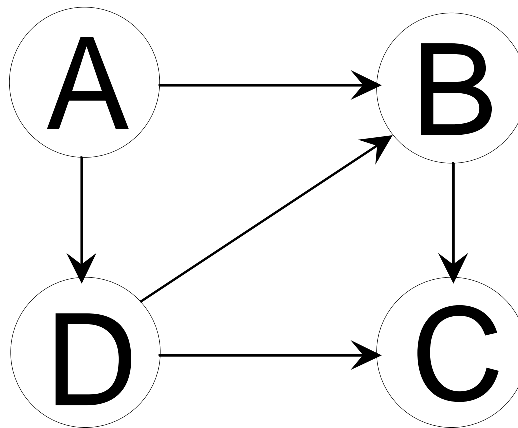

Figure 6-19. A Collapsed Graph.

The gates of a strongly connected component are all contained in one or more logic-loops, or *cycles*. The simulation process is best illustrated through an example. Consider the circuit of Figure 6-20 and the strongly connected component consisting of gates G3 and G4. Inside this component, there are two different types of inputs, those that originate outside the component (X1 and X2), and those that originate inside the component (Q1 and Q2). When the component is simulated, the values nets coming from the outside will be known for all simulation times. The values of the other nets will be unknown, except for the final value from the previous input vector in bit 0. A typical scenario is given in Figure 6-21.

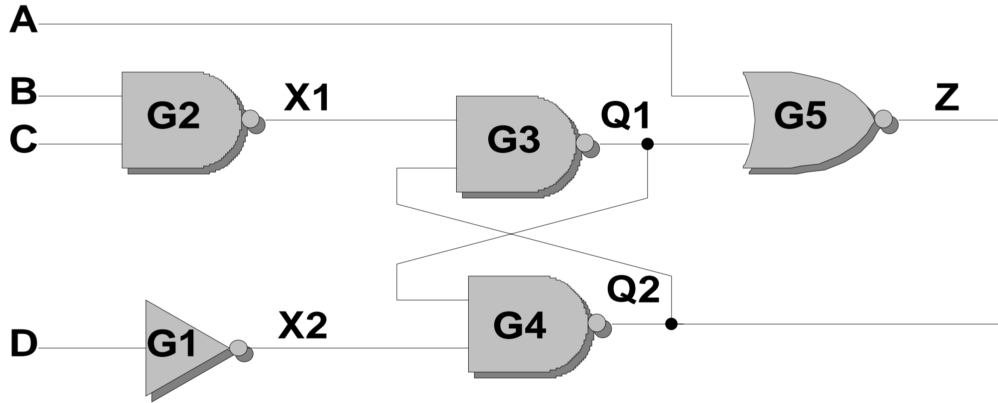

Figure 6-20. A Circuit Containing a Cycle.

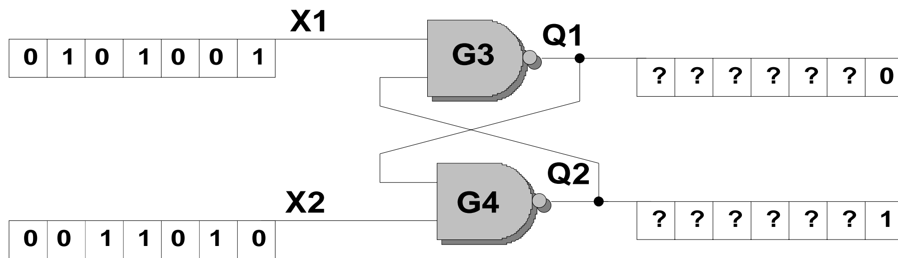

Figure 6-21. Simulation of Strong Components.

To simulate the component of Figure 6-21, it is necessary to simulate all gates in levelized order. Forced levelization must be used, since the component contains a cycle. It doesn't matter which gate is simulated first, but the ordering should be consistent for each simulation of the component. Let us assume that G3 is simulated first. In this case, the simulation proceeds as shown in Figure 6-22.

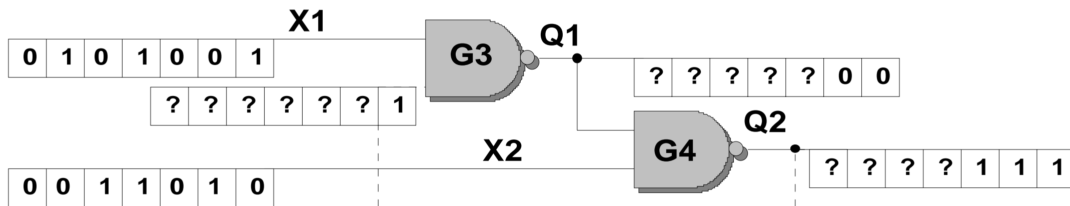

Figure 6-22. A Simulation Cycle of a Strong Component.

In Figure 6-22, the input, Q2, to G3 has a single correct input in the low-order bit position. A bit-wise NAND is performed on the two bit-fields, and the result is shifted to the left one bit. The existing low-order bit from the Q1 bit-field will be placed in the vacated low-order bit position. This gives two correct low-order bits for net Q1. The same procedure gives three correct low-order bits for Q2. If more than three correct bits are required, it is necessary to simulate the component again, as in Figure 6-23, to accumulate more bits. The component can be simulated as many times as necessary to produce the required number of output bits.

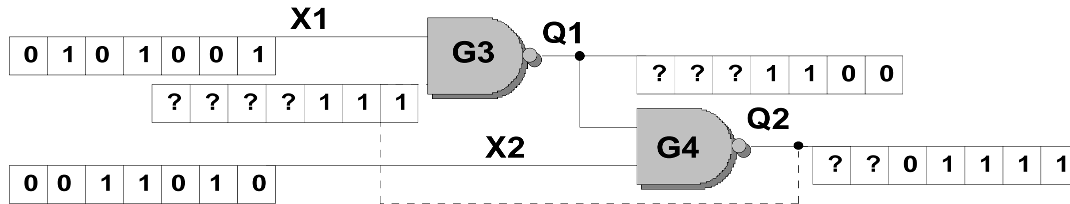

Figure 6-23. Second Simulation of a Strong Component.

As Figure 6-22 and Figure 6-23 illustrate, each simulation of this component accumulates two correct bits in the Q2 bit-field. The general rule is that the number of correct bits accumulated for each simulation of a strongly connected component is equal to the length, in gates, of the shortest cycle. If there is more than one cycle in the component, different bit-fields may accumulate correct bits at different rates.

It is possible to detect complex oscillations in the outputs as long as a sufficient number of correct bits is generated. The number of correct bits depends on the stability of the input signals and the number of feedback arcs. In Figure 6-23 there is a single feedback arc, but complex circuits can have many more. Suppose there are k feedback arcs. To detect any oscillation it is necessary to generate 2^k+1 additional correct bits after the inputs stabilize. It is then necessary to examine the bit-fields for duplicate output patterns. If the duplicate patterns are adjacent, the circuit has stabilized, otherwise it is in oscillation. Since there can be at most 2^k different patterns on k feedback arcs, generating 2^k+1 additional bits will guarantee that any oscillation has been captured.

The main drawback of the asynchronous parallel technique is the extremely wide bit-fields that are required to produce the requisite number of output bits. In an oblivious simulation, a worst-case scenario must be assumed, and this leads to extremely long bit-fields, particularly when several strongly connected components must be simulated in sequence. Because of this, the asynchronous parallel technique is most appropriate for small circuits or for large synchronous circuits with some asynchronous elements.

6.6 Conclusion

The parallel technique is extremely effective for simulating unit-delay circuits, giving an average of 90% savings in execution time over conventional event-driven simulation. Even greater savings can be realized by using the path-tracing and bit-field trimming techniques. Multi-delay simulations are less effective because much larger bit-fields are required. Synchronous sequential circuits can be simulated using the parallel technique by breaking the circuit at each synchronous flip-flop and then simulating the circuit as a combinational circuit. There are techniques that permit asynchronous circuits to be simulated, but these are effective only for reasonably small circuits, or for larger circuits with only a few asynchronous elements.

The parallel technique can be combined with other techniques to produce efficient compiled simulations of highly complex circuits.

6.7 Exercises