State Machine Design Considerations and Methodologies

The use of state machines provides a systematic way to design complex sequential logic circuits—an increasingly popular approach since the advent of PLD (Programmable Logic Device) circuitry. This application note describes the many options encountered during the state machine design cycle. By exhaustively walking through the PLD-based design example presented here, you can weigh the merits of several design approaches.

Definitions of Commonly Used Terms

External input vector—External signals (stimulus) applied to the state machine.

System outputs—Signals generated by the state machine that are explicitly designed for availability to the external system (hardware outside of the state machine). Registered system outputs can also be fed back into the state machine as part of the State Vector, which is then used in the decode of the state machine's next state.

State registers—Registers used exclusively for determining the next state of the machine (feedback).

State outputs—Outputs of the state registers that are available to the external system. (They are typically available to the external machine for debug or due to the lack of buried registers.)

State vector or machine state—The registered feedback information defining the present state of the machine and required to determine the next state of the machine.

State path—The transitional condition that must be met for the state machine to progress from one state to another. The state path typically consists of one or more product terms generated from external inputs, although other state paths are possible.

Total input vector—The combination of the external input vector and the state vector. The total input vector is decoded to generate the next state of the machine.

State Machine Entry Methods

There are many ways of describing a state machine, each with distinct advantages and disadvantages. Three popular description methods are state diagrams, state tables, and high-level languages (HLLs). The state diagram provides an easily observable flow description of the state machine. Because the ability to view the flow of states provides distinct documentation advantages, state diagrams will be used throughout this application note to describe the example state machine.

Upon completing a state diagram, you can easily convert the diagram's visual information into the other types of state machine description or directly into Boolean equations. Several available software programs accept their own forms of state table, HLL, and/or Boolean entry. You can enter all these formats easily via your favorite text editor. The software then translates the inputs into suitable forms (usually a JEDEC map) for hardware implementation.

Another method of describing a state machine, the state table, offers perhaps the most concise description. Its major advantage over the other entry methods is the availability of state table reduction methods (see Reference 1). When applied to your state table definition, a reduction program generates a minimal model for the function. The software used for state machine synthesis throughout this application note uses the state table method of entry. The program is called LOG/iC™ from Isdata Corporation.

Finally, high-level language (HLL) state machine entry is probably the most popular form of state machine design. HLLs typically offer C-language-like instructions (e.g., case, if-then-else, etc.) to describe the machine.

A Sample State Machine

The sample state machine is a clock generator for a pipelined (three system execution stages), bit-slice-based, central processing unit (CPU). Each of the three system execution stages contains two clocks for a total of six system clocks for every instruction execution. With pipelining enabled, each instruction takes an average of two clock periods. Further, external hardware unaffected by CPU wait and stop states (e.g., cache memory) needs both polarities of an additional free-running clock.

To minimize clock edge skew, the state machine provides both versions of the clock. To put the timing of this application into perspective, executing each pipeline stage in an 80-ns period (or 12.5 MHz) requires the state machine to run at 25 MHz. This speed is well within the range of the available PALs, EPLDs and PROMs that can be used to implement the state machine.

Each of the pipeline's three execution stages has a specific function. Briefly, the first stage of the pipeline accesses the Writable Control Store (WCS) RAM. The Arithmetic Logic Unit (ALU) execution occurs during the second stage of the pipeline. Finally, the third pipeline stage clocks status and memory address registers. The function(s) performed during each of the three stages are described in greater detail in the State Machine Output Definition section of this application note.

If this design only generates a simple set of pipelined clocks, why not use shift registers and miscellaneous glue logic instead of a state machine? There are two reasons to consider a state machine. First, it is usually desirable to minimize the number of chips required; the state machine in PLD form might need external glue logic, but significantly less than the shift register solution.

The second reason for considering a state machine is that this application requires more then just a simple set of pipeline clocks. The function of the clock signals is to provide control

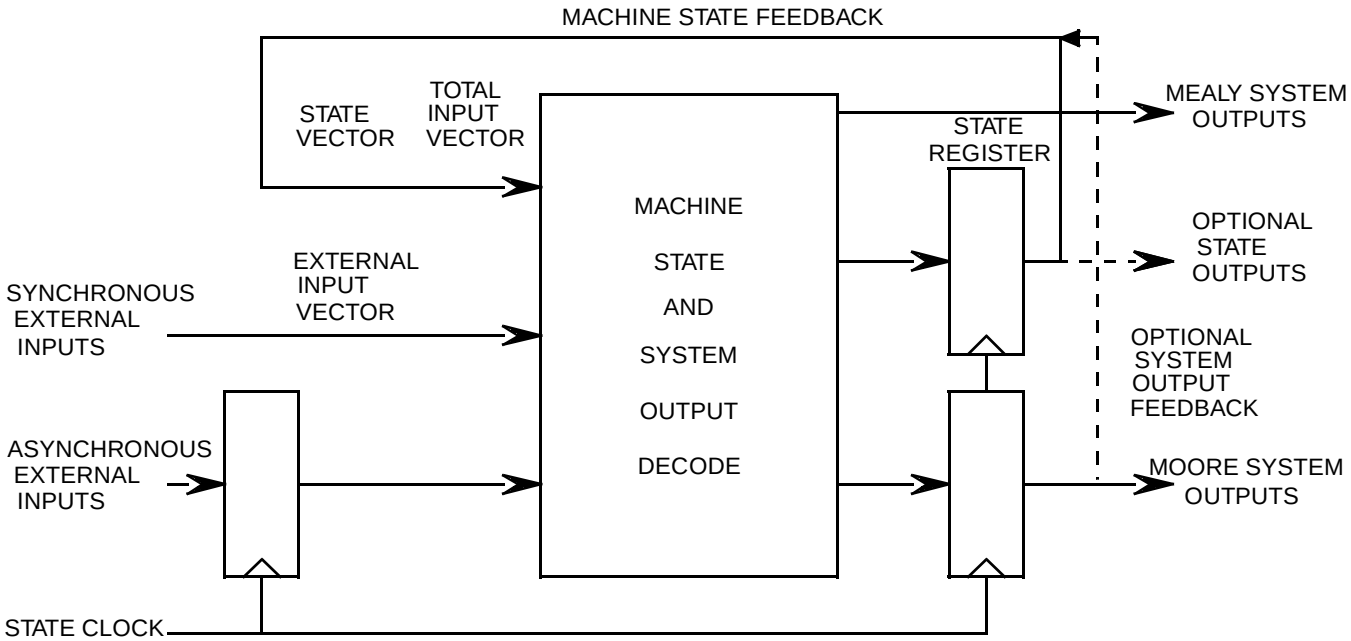


Figure 2. Synchronous State Machine Block Diagram

of the CPU in multiple modes of operation. The desired modes of operation follow.

PIPELINED RUN Mode

In this mode, the CPU simultaneously performs the instructions in all three stages of the pipeline. For example, while instruction n does an ALU operation, instruction n+1 accesses WCS, and instruction n–1 clocks ALU status.

NONPIPELINED RUN Mode

NONPIPELINED RUN mode performs all three stages of instruction execution without overlap. The time to complete one nonpipelined instruction equals the average of three pipelined instructions.

CPU STOP

The system must have a way to perform an orderly stop of CPU execution from both of the above run modes. This stop might be the result of several possible conditions, including a utility stop from a system control unit, a single step, a breakpoint, or a response to external hardware (e.g., a logic analyzer). The free-running clocks continue to run during the CPU STOP mode and remain running at all times, except during a reset condition.

CPU WAIT

In CPU WAIT mode, an external condition causes a delay in an instruction's execution. The instruction pauses until the external condition is removed. One application for the CPU WAIT mode is to handle a cache miss. When a cache miss occurs, the CPU remains in the CPU WAIT mode until the cache completes its memory transfer.

SINGLE STEP

The ability to execute one instruction at a time is needed to debug the CPU. You can easily implement SINGLE STEP external to the clock state machine by pulsing the RUN signal. SINGLE STEP mode is described further in the State Machine Input Definition section of this application note.

INTERRUPT

A variety of system conditions can interrupt the CPU out of its normal execution sequence and immediately start the execution of the interrupt handler. The influence of the INTERRUPT mode on the system clocks will be discussed in greater detail later in this application note.

REPEAT INSTRUCTION

The REPEAT INSTRUCTION mode is a CPU debug feature. It is a good idea to implement this mode external to the clock state machine. By dubbing the clock to the instruction register and the interrupt line to the clock state machine, the CPU continually executes the instruction in the instruction register.

Synchronous vs. Asynchronous Machine

At this point in the state machine design, an appropriate type of state machine must be chosen to match the application. Two major types are the asynchronous and the synchronous implementations. The asynchronous machine changes state when one or more of its inputs changes from a previously stable input state. After a state change, the outputs of the state machine settle, while the machine stabilizes once again. A basic example of an asynchronous state machine would be a simple SR latch built from two NAND gates (*Figure 1*). For the clocking application considered in this application note, the asynchronous state machine implementation would be a poor choice, due to the instability of the system outputs.

The synchronous state machine offers a better choice. A synchronous state machine block diagram appears in *Figure 2*. Generally, a synchronous state machine samples the total input vector at specific periods to determine the machine's next state. When designing synchronous state machines, it is important to avoid state register metastability. External inputs to the machine must be synchronized to guarantee stable state register inputs, and the feedback time plus data set-up time to the state register clock must be less then or equal to the state clock period.

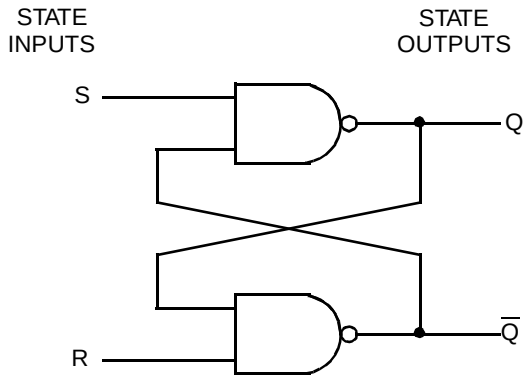


Figure 1. SR Latch, Asynchronous State Machine Example

The modern theory of synchronous state machines was pioneered by Mealy and Moore (see Reference 1). Mealy and Moore machines differ slightly from each other in the way they control the system outputs. During a specific machine state, a Mealy machine allows the input conditions to alter the system outputs (the outputs depend on the "total" input state). In contrast, a Moore machine system outputs depend only on the present machine state. Thus, the system outputs remain stable until the next time period, when the Moore machine samples the total input vector to determine the next state. If all design conditions are met (external inputs are stable prior to the next state clock), the Moore machine provides glitch-free system outputs-a desirable characteristic for the CPU system clock. The design described here is therefore implemented as a Moore machine.

Clock Generator Output Definition

As explained earlier, each of the three system execution stages contains two clocks for a total of six system clocks for every instruction execution. The naming convention for these clocks is

CLK_xy

where x = 1, 2, or 3, representing the first, second, or third stage of the instruction execution and y = A or B, representing the first or second half of the execution stage.

Following this convention, the state machine's two free-running clocks are named CLK_A and CLK_B. These clocks run at half the state clock frequency and 180 degrees out of phase. The free-running clocks occur at the same time as their respective CLK_xA and CLK_xB clocks.

The major clock functions for this application are:

CLK_1B: The leading edge of this clock updates the instruction register.

CLK_2A: This clock's leading edge marks the start of ALU execution. The information on the ALU input bus clocks into the appropriate input registers at this time. The instruction cycle is considered recoverable up through and including CLK_2A (i.e., the status of the machine from the previous instruction has not been altered).

CLK_2B: Used to control the second half of the ALU execution stage, this clock initiates a write to RAM, triggers counters, gates ALU output into its latch, and clocks the ALU output information into any of the distributed destination registers.

CLK_3A: On this clock the memory address register can be updated. The ALU output bus status and ALU status is also clocked into the CPU status register.

Clock Generator Inputs

A set of inputs (external stimulus to the state machine) controls the state machine. The clock state machine described here has eight external inputs, including the state machine clock. These inputs are:

STATECLK: The state machine clock.

RESET: An asynchronous or synchronous reset input that can be connected directly to the state registers' preset or clear or to all clocked register inputs (D or T input). If connected to the preset or clear, RESET need not be synchronized. In this case, RESET forces the state machine into the machine's initial state, regardless of the present state. RESET can result from any combination of the following sources:

- Power up circuit (system reset)
- System controller software decodes system reset
- System controller software decodes module reset
- CPU software decodes module reset

RUN: This signal controls the start and stop sequence of the CPU clocks. In PIPELINE RUN mode, the start sequence generates the proper clock progression to fill up the pipeline registers, and the stop sequence empties the pipeline. RUN is externally manipulated to implement the single step and breakpoint functions.

NPL: Used to select NONPIPELINED RUN vs. PIPELINED RUN modes, this signal must be set to the selected mode prior to activating the RUN signal. Setting NPL = 1 selects NONPIPELINED RUN mode, and NPL = 0 selects PIPELINED RUN mode. The single step function operates properly in NONPIPELINED RUN mode only.

INTR: This signal indicates an external interrupt. When INTR is received, and IEN (interrupt enable, described below) is active, the CPU executes its interrupt handler. An interrupt inhibits the instruction register update clock (CLK_1B) and the ALU update clock (CLK_2B). CLK_1A for the interrupt instruction executes on the next cycle. The interrupt condition has priority over a wait condition and therefore starts generating clocks to permit execution of the interrupt instructions.

IEN: This interrupt enable signal qualifies INTR. IEN is likely to be a bit in the instruction word, allowing the user to define sections of uninterruptable code.

WAIT: The wait condition is initiated when both WAIT and WEN (wait enable, described below) are active. The CPU remains in the wait condition until WAIT goes inactive.

WEN: This wait enable signal qualifies WAIT for entrance into the wait condition. Like IEN, WEN is usually a bit in the instruction word, allowing the user to define sections of wait-sensitive code.

State Machine Partitioning

When architecting a state machine, it is generally a good practice to break up large machines into workable blocks, with each of the smaller machines containing states that require common inputs and generate common outputs. The example

clock state machine is small enough to be designed as a single state machine, although it would be trivial to design logic to generate the free-running clocks as a separate machine from the rest of the clock state machine. Equations for the free-running clocks are:

CLK_A := $\overline{\text{RESET}}$ • $\overline{\text{CLK\_A}}$

CLK_B := $\overline{\text{RESET}}$ • CLK_A

where ":=" indicates a registered output.

By examining these output equations, you can see that the free-running clocks have only two dependencies in common with the remaining portion of the clock state machine, i.e., RESET and STATECLK. The free-running clocks are required as inputs to the other state machine to synchronize the additional system outputs, however.

The example presented here implements the free-running clocks and the other system outputs within the same state definition. The resulting output equations can be verified against the equations for the free-running clocks alone.

The Initial Machine State

Regardless of the preferred state machine entry method, attacking the problem starts with defining the initial state of the machine. This initial state (INIT in the example) must be consistent with the power-on condition and/or an external input used to initialize the machine (RESET).

The state of the machine can be decoded from the present values of the system outputs, state registers, or a combination of the two. (The advantages and disadvantages of the state definition options will be discussed in greater detail later in this application note.) The initial machine state is generally, but not always, a decode of all 0s or all 1s. In the example design, INIT is the decode of all 0s.

Naming the States

With the exception of INIT, each state in the example design is named to indicate the active system clocks occurring during that state. For example, during state A, only CLK_A is active. Similarly, state 123B has only CLK_1B, CLK_2B, CLK_3B, and CLK_B active. Additionally, an "N" suffix designates a nonpipelined state and a "W" suffix designates a wait condition state; this convention differentiates between states with identical active system outputs.

CPU Inactive States

The RESET input causes the state machine to enter the INIT state from any state in the machine. From the INIT state, the machine unconditionally starts to generate the free-running clocks. As shown in *Figure 3*, a line pointing from the INIT state to the A state, with a path equation equal to 1, indicates an unconditional branch. The state machine progression continues from the A state unconditionally into the B state. In the B state a multi-branch condition exists. If the RUN input remains inactive, then the A and B states continue to toggle, generating only the free-running clocks. Hence the INIT, A, and B states are referred to as "CPU inactive states."

Nonpipelined States

If the NPL input is active while the RUN input becomes active, the state machine operates in NONPIPELINED RUN mode and follows the model portrayed in *Figure 4*.

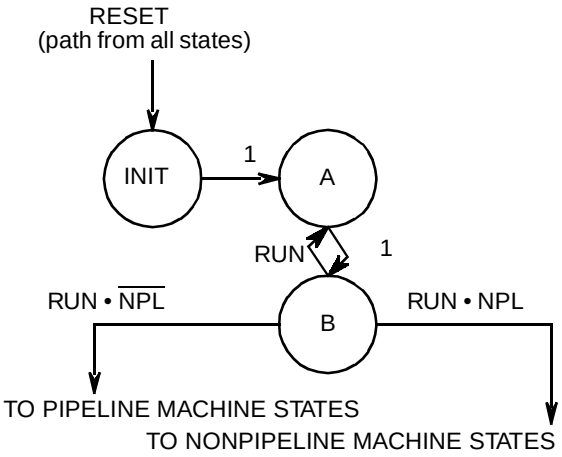


Figure 3. CPU Inactive States

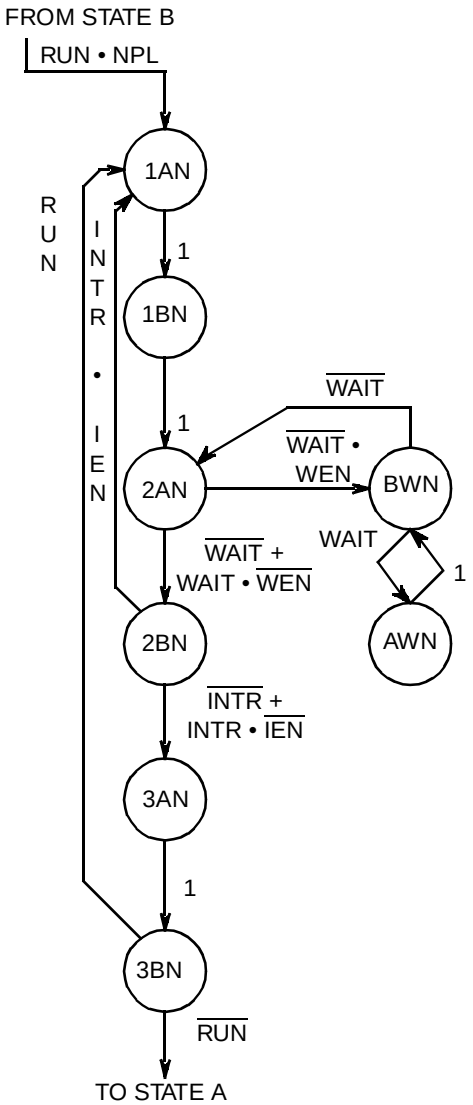


Figure 4. Non-Pipelined States

Pipelined States

If the NPL input is inactive when the RUN input goes active, thus indicating PIPELINED RUN mode, the state machine operates as depicted in *Figure 5*.

Unique States

When the RUN input goes active, the next state executed is either the 1A or the 1AN state, depending upon the value of the NPL input (refer to *Figures 4* and *5*). Notice that the active system outputs in these two states are identical. Why generate two identical states—when an additional state register might be required to differentiate between the states? (This assumes you use the system outputs to decode the machine's states.) The redundant states are not a problem because the additional state register needed to differentiate between the states is not an issue. There are two reasons for this. First, if you eliminate the redundant states, the state machine would require at least one additional state register anyway to differentiate between the B and the BW or BWN states, which would be needed without 1A and 1AN. (Separation of states BW and BWN from state B is required for correct functionality.) Second, adding another state only increases the number of state registers if the new total number of states exceeds an additional binary boundary (2, 4, 8, 16,...). This is not a problem here.

You might also choose to widen your state machine (increase the number of state registers) to reduce the number of product terms to the state or system output registers. This decision should take into account the desired circuit implementation (PLDs, PROMS, discrete hardware, etc.) and is often an

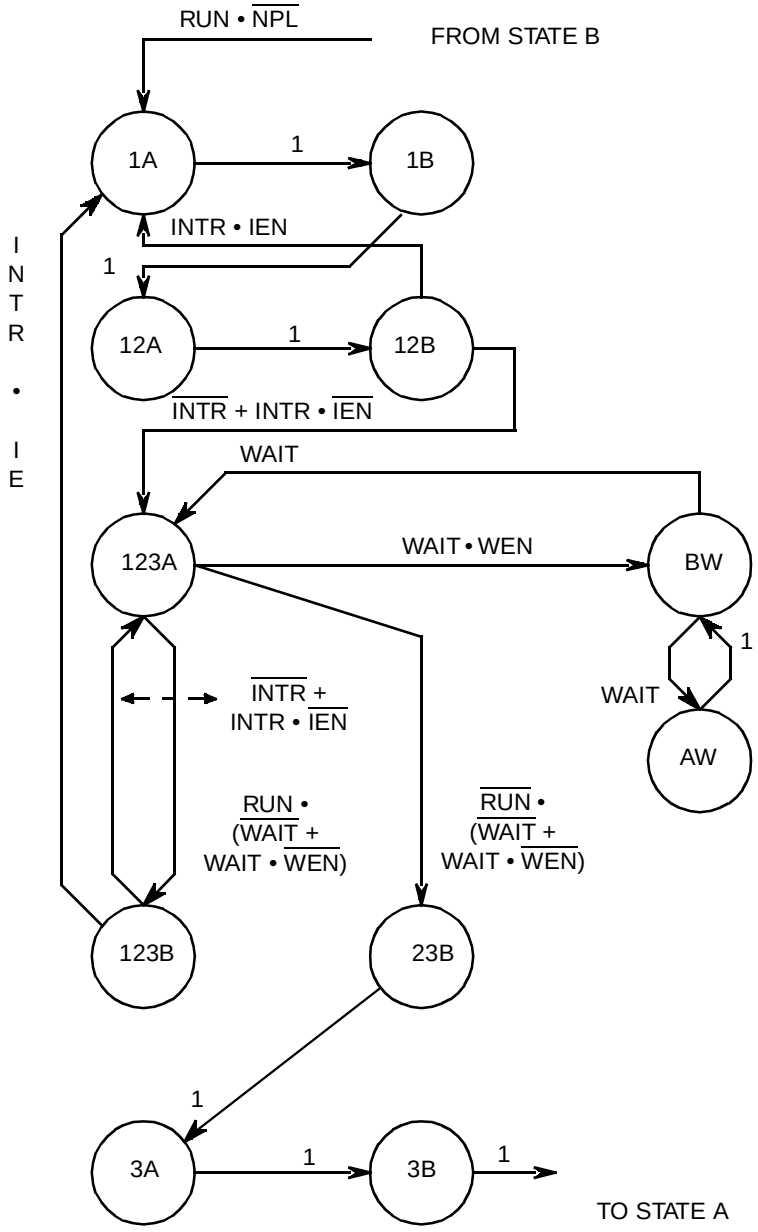


Figure 5. Pipelined States

iterative process. In general, you can initially architect the state machine in the manner that is the easiest for you to understand, then make additional changes or small adjustments later if they become necessary.

State Description Verification

Now that all the pieces of the state machine are functionally defined (refer to *Figure 6* for the completed state diagram), consider methods for verifying the validity of the design. Some software you can use to describe and implement state machines would already offer verification at this point in a design. For other methods, read on!

One way to verify a state machine design is to recognize a rule of thumb: Out of every state, there should be a state path to another state for every possible combination of relevant external inputs. For example, there are two paths out of state 123B, with INTR and IEN as the relevant external inputs:

Path 1 = INTR • IEN

Path 2 = $\overline{\text{INTR}}$ + INTR • $\overline{\text{IEN}}$

If there are no known restrictions on the external inputs, a simple method of verifying the above rule of thumb is to generate an equation where all of the paths out of a state are ORed together as follows:

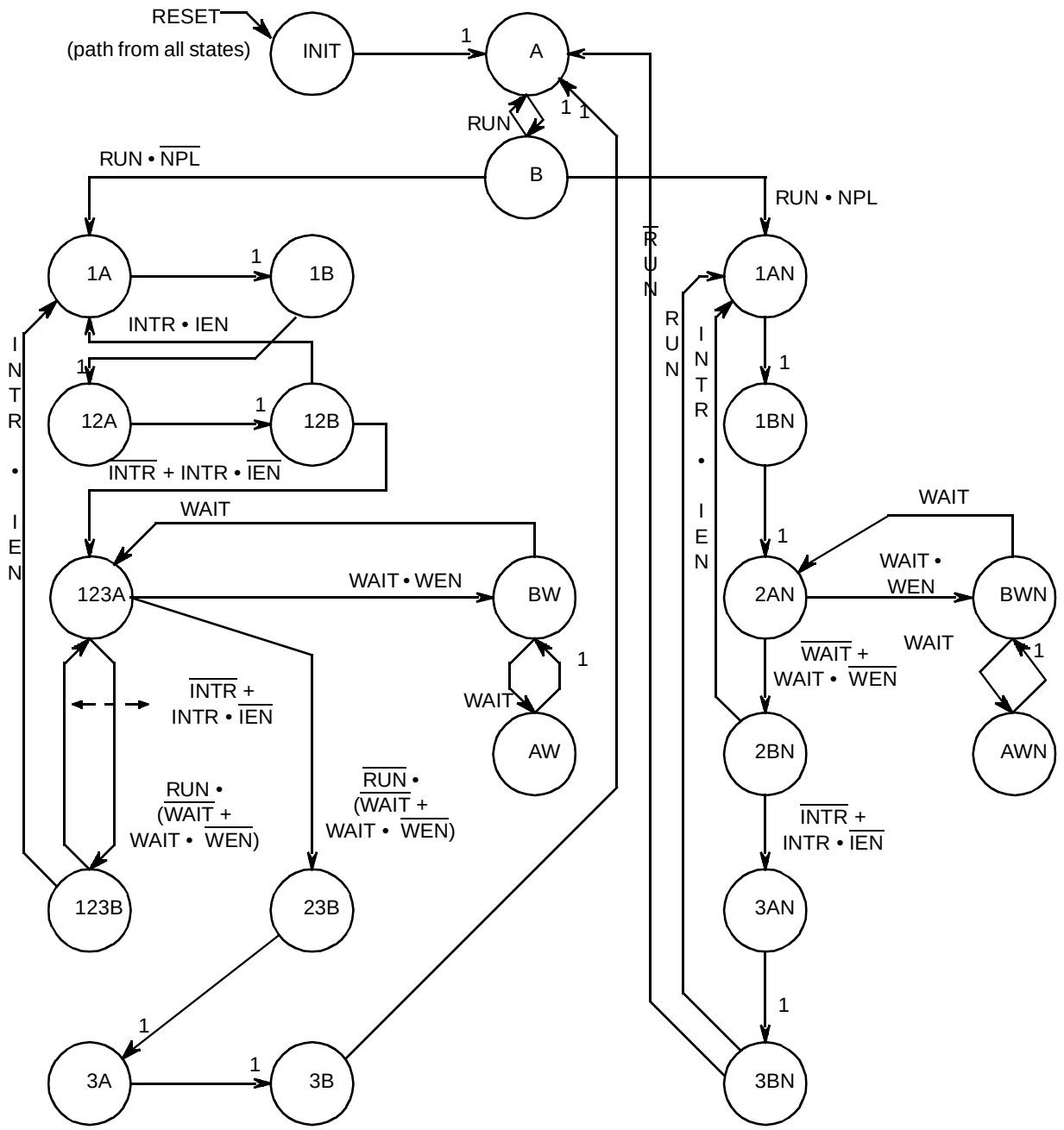


Figure 6. CPU Clock State Machine

```
OUT_STATE_123B      = Path 1 + Path 2;
OUT_STATE_123B      = (INTR • IEN)
                    + INTR
                    + (INTR • IEN);
= OUT_STATE_123B    = 1
```

If the equation's terms equal 1 after Boolean reduction, then every state path out of the state is accounted for. The main advantage to this verification method is that you can easily do it using readily available Boolean reduction software.

If there are known restrictions to the external inputs, you can use this information to reduce the complexity of the machine. If it is impossible for the INTR • $\overline{\text{IEN}}$ condition to occur externally, for example, then you can leave this condition out of the Path 2 equation. In that case, the reduction of the OUT_STATE_123B equation yields a non-1 result.

Because the method of verification just described does not detect redundant path equations, it is useful to revise the original rule of thumb to: Out of every state, there should be one and only one state path to another state for every possible combination of relevant external inputs.

This revised condition is not as easily verified as the original statement. The easiest way to verify the more restrictive case is to simulate the state machine. To do this, you must generate a test vector for every possible external input that is relevant to each state simulated. Automatic test vector generation programs are available that produce every possible combination. After running the vectors against the design, you must visually inspect the output to verify that the machine never enters an illegal state.

System and State Register Output Generation

The model defining the clock state machine is complete, but there are still quite a few important decisions to be made regarding the final circuit implementation. Some of the major alternatives for final implementation are:

- System output vs. exclusive state register state decode
- D flip-flop vs. T flip-flop implementation
- PLD vs. PROM implementation

To gain some insight into these choices, consider how the output or feedback equations are assembled. Take, for example, the generation of CLK_3A using a D flip-flop (FF) implementation. By referring to *Figure 6*, you can find all the states in which CLK_3A is active. These are 123A, 3A, and 3AN. The CLK_3A output is generated by ORing the state decodes that, when ANDed with their respective state paths, advance the state machine into the three states listed above. Specifically:

```
CLK_3A :=
(Decode of 12B)•(INTR+INTR•IEN)   ;-123A
+(Decode of BW)•(WAIT)            ;-123A
+(Decode of 23B)•(1)              ;-3A
+(Decode of 2BN)•(INTR+INTR•IEN)  ;-3AN
```

When you define the state decodes, the CLK_3A equations are completely specified in terms of the state machine inputs (state path), state registers, and/or system outputs (state decode). Typically, you then multiply the equation out to form a sum of products. This format provides for easy implementation in a PLD, which has a sum-of-products architecture, and also provides a useful foundation for further equation reduction.

State Decode

As discussed earlier, the next state of the machine can be decoded from the present values of the system outputs, the state registers, or a combination of the two. The choice typically comes down to weighing the maximum number of product terms verses the maximum number of flip-flops available in an implementation. For a Moore machine with registered system outputs, using the system outputs to uniquely define the states uses the smallest number of flip-flops to define the state machine. However, it is often necessary to add one or more state registers to uniquely define the states.

State assignment for this state decoding method is quite simple, but also rigidly defined, allowing limited flexibility when assigning the additional state registers. After reduction, the feedback and output equations of this "narrow" state machine might contain too many product terms to be implemented in a specific PLD, although product term complexity is never a problem with a PROM implementation.

Exclusive State Registers

Another consideration in state machine design is that you might be able to distribute the number of product terms more evenly among the equations implementing the state machine by using state registers exclusively to decode the states. Because the state decodes in the state registers can be selected to assist in Boolean reduction, proper state assignment enables the more complex equations to fit into a specific implementation.

This type of decode is useful in a PLD implementation, where there is a shortage of product terms for a specific state flip-flop, but extra flip-flops are available. Adding an extra state register can simplify the decode logic enough to fit the design in a single PLD.

The total number of exclusive state registers required to implement a state machine varies from a minimum of LOG(2)X (rounded up to the nearest integer) to a maximum of X, where X is the total number of states in the machine. You can iteratively change this number, along with the state assignment, to obtain a suitable solution.

The state assignment itself is a non-trivial issue, with almost limitless possibilities and no known method of obtaining the optimal solution. There are, however, some guidelines that can be used to obtain workable solutions:

1. Two or more states that potentially enter the same state with identical path equations should be adjacent (their binary codes differ in exactly one position). As an example, refer to *Figure 5*. States 12B and 123B both proceed into state 1A if the path condition INTR • IEN is true. When generating the CLK_1A equation, two of the terms of the equation look like this:

```
CLK_1A :=
   (Decode of 12B) • (INTR • IEN)     ;-1A
+ (Decode of 123B) • (INTR • IEN)     ;-1A
   .  .  .
```

If the decode of 12B and 123B differ in exactly one position, then Boolean reduction (which uses the A•B + $\overline{\text{A}}$•B = B relationship) converts the two product terms into one smaller product term.

2. Two or more states that might proceed into different states with identical path equations, and an identical active out-

put, should be adjacent. This situation occurs in the previous CLK_3A equation, shown again here:

```
CLK_3A :=
(Decode of 12B)•(INTR+INTR•IEN)      ;-123A
+(Decode of BW)•(WAIT)               ;-123A
+(Decode of 23B)•(1)                 ;-3A
+(Decode of 2BN)•(INTR+INTR•IEN);    -3AN
```

Note that if states 12B and 2BN are adjacent, then you can reduce the CLK_3A equation to three product terms.

Clock Generator Implementation

As mentioned earlier, there are many ways to implement state machines. The following sections discuss some of the pros and cons associated with some of the more common state machine implementations.

D Flip-Flop Implementation

There are more products available that support a D flip-flop solution than any other implementation. Therefore, it is usually the most cost-effective solution for a state machine.

Table 1 lists the number of product terms per output obtained by compiling the clock generator state machine definition with the LOG/iC software, using D flip-flops. The compiler input file appears in Appendix A. Optimizing the design (*Table 2*) significantly reduces the number of product terms needed.

Table 1. Non-optimized Results for Clock Generator: D Flip-Flop Implementation

Log/IC Optimization Summary (FACT)				
CPU Time Quota per Function: 100 sec				
Function	**INV**	**P-Terms**	**CPU-Time**	**Flags**
CLK_1A.D	No	12	<1	N
	Yes	27	<1	N
CLK_1B.D	No	5	<1	N
	Yes	34	1	N
CLK_2A.D	No	8	<1	N
	Yes	31	<1	N
CLK_2B.D	No	7	<1	N
	Yes	32	<1	N
CLK_3A.D	No	8	<1	N
	Yes	31	<1	N
CLK_3B.D	No	6	<1	N
	Yes	33	<1	N
CLK_A.D	No			NT
	Yes			NT
QQ1.D	No	6	<1	N
	Yes	5	<1	N
QQ2.D	No	10	<1	N
	Yes	9	<1	N

N: No Optimization
T: Trivial Function
FACT Minimization: 11 sec

Table 2. Optimized Results for Clock Generator: D Flip-Flop Implementation

Log/IC Optimization Summary (FACT)				
CPU Time Quota per Function: 100 sec				
Function	**INV**	**P-Terms**	**CPU-Time**	**Flags**
CLK_1A.D	No	6	1	
	Yes	11	2	
CLK_1B.D	No	3	1	
	Yes	4	<1	
CLK_2A.D	No	4	1	
	Yes	7	<1	
CLK_2B.D	No	3	1	
	Yes	4	<1	
CLK_3A.D	No	4	1	
	Yes	9	1	
CLK_3B.D	No	3	<1	
	Yes	3	1	
CLK_A.D	No	1	<1	
	Yes	2	<1	
CLK_B.D	No	1	1	
	Yes	2	<1	
QQ1.D	No	3	<1	
	Yes	3	1	
QQ2.D	No	6	16	
	Yes	6	2	

FACT Minimization: 29 sec

T Flip-Flop Implementation

Even though D flip-flop solutions are more widely available, there are times when the logic needed for this implementation is prohibitively complex. Under these circumstances, a T flip-flop implementation might be more cost effective, because using T flip-flops reduces the logic significantly.

The best example of this situation is a simple synchronous binary counter. While the most significant bit (MSB) of an N-bit counter in a D flip-flop implementation requires N product terms, the T flip-flop solution requires only one product term. Note that the Cypress family of CY7C33x devices offers you a configurable T- or D-type implementation if you place an XOR gate prior to the D flip-flop; route the AND/OR array to one of the XOR's inputs and the flip-flop's Q output (via an additional product term) to the other XOR input.

It isn't clear from simple observation, however, whether the T flip-flop implementation is beneficial for the clock generator state machine. One way to clarify this question is to change three command lines in the state machine description shown in Appendix A and recompile to produce a T flip-flop implementation. *Table 3* contains the product term results using T flip-flops. A quick study of the results reveals that the optimized version using D flip-flops (*Table 2*) requires fewer product terms than the T flip-flop version.

Table 3. Optimized Results for Clock Generator: T Flip-Flop Implementation

LOG/iC Optimization Summary (FACT)				
CPU Time Quota per Function: 100 sec				
Function	**INV**	**P-Terms**	**CPU-Time**	**Flags**
CLK_1A.T	No	6	<1	
	Yes	7	1	
CLK_1B.T	No	4	1	
	Yes	3	1	
CLK_2A.T	No	5	1	
	Yes	4	<1	
CLK_2B.T	No	4	1	
	Yes	3	<1	
CLK_3A.T	No	5	<1	
	Yes	6	2	
CLK_3B.T	No	4	<1	
	Yes	2	<1	
CLK_A.T	No			C
	Yes			C
CLK_B.T	No	2	1	
	Yes	1	<1	
QQ1.T	No	3	<1	
	Yes	5	1	
QQ2.T	No	6	<1	
	Yes	11	2	

PLD Implementation

With the LOG/iC PLD Database option, the software assists in selecting a PLD, and it shows that the non-optimized version of the clock state machine fits in a PALCE22V10 without further reduction. If the equations are reduced using Boolean reduction, however, a lower-cost solution is available. The results shown in *Table 3* indicate that the less expensive PLDC20G10 would work. Appendix A shows the listing for the 20G10 LOG/iC implementation. Waveforms for the completed design appear in Appendix B. You can verify the CLK_A and CLK_B equation results against the equations generated in the State Machine Partitioning section of this application note.

PROM Implementation

You can obtain very high speed solutions by implementing state machines using PROMs. A PROM uses a look-up table to decode the machine's next state, as opposed to the AND/OR array in a PLD. The main advantage of using a look-up table to decode the next state is that every combination of the inputs can be decoded. Thus, you can create an extremely complex machine, without equation reductions.

The look-up table's drawback is that the PROM's depth grows exponentially (2N, where N = # of inputs to the look-up table) with every additional input to the look-up table. To determine the depth required, notice that the present total input vector provides the inputs to the look-up table. The clock generator state machine has seven external inputs, six system outputs, and two state outputs, which indicates a feasible implementation using the CY7C277 (32K x 8) registered PROM.

Using a registered PROM such as the CY7C277 to implement the machine also helps to reduce the parts count, because the PROM implements both the state and system output registers. LOG/iC offers support for implementing state machines in PROMs, and only a few minor changes to the state machine description shown in Appendix A are required. *PROM replaces the *PAL command, some simple statements indicating the CY7C277 architecture (INPUTS = 15 AND OUTPUTS = 8) replaces the TYPE = statement, and PROGFORMAT = INTEL–HEX.

Reference

1. Donald D. Givone, Introduction to Switching Circuit Theory (New York: McGraw-Hill, Inc., 1970)

Appendix A. LOG/iC PLD Source Code: Clock State Machine

```
LOG/iC-PAL             Rel 3.2/2-2328-1721/00034 #32-5955 90/03/15  23:49:45
-------------------------------------------------------------------------------
LOG/iC - COPYRIGHT (C) 1985,1988 BY ISDATA GMBH, 7500 KARLSRUHE WEST-GERMANY
Cypress Semiconductor                                 LICENCE FOR IBM-PC/XT/AT
Data Set: OD20G10.DCB
  1     1: *IDENTIFICATION
  2     2: PIPELINED CLOCKING SYSTEM OD20G10
  3     3: ERIC B.  ROSS
  4     4: CYPRESS SEMICONDUCTOR
  5     5: NAMING CONVENTION
  6     6:  OD      = SYSTEM OUTPUTS ARE DFLOPS AND ARE USED FOR STATE DEF
  7     7:  20G10   = PALC20G10 IMPLEMENTATION
  8     8: *PAL
  9     9:  TYPE=PALC20G10
 10 I  10:
 11    11: *X-NAMES
 12 I  12: ;-------------------------------------------------------------------
 13 I  13: ;INPUT DEFINITIONS :
 14 I  14: ; RUN    = START & STOP EXECUTION OF OUTPUT CLOCKS (NORMAL, SINGLE
 15 I  15: ;          STEP, & BREAK PT.  EXECUTION
 16 I  16: ; NPL    = PIPELINED VS NON-PIPELINED MODE OF EXECUTION
 17 I  17: ; INTR   = EXTERNAL INTERRUPT CONDITION (TLB MISS, PARITY ERROR,...)
 18 I  18: ; IEN    = INTERRUPT ENABLE
 19 I  19: ; WAIT   = WAIT ENABLE (CACHE MISS)
 20 I  20: ; WEN    = WAIT ENABLE
 21 I  21: ;-------------------------------------------------------------------
 22 I  22: ;
 23    23:  RUN, NPL, INTR, IEN, WAIT, WEN, RESET;
 24 I  24:
 25    25: *Z-NAMES
 26 I  26: ;-------------------------------------------------------------------
 27 I  27: ;OUTPUT DEFINITIONS :
 28 I  28: ;
 29 I  29: ; 3 CLOCK STAGES 1, 2, 3
 30 I  30: ; 2 CLOCKS PER STATE A, B
 31 I  31: ;    CLK_XX WHERE XX = 1A,1B,2A,2B,3A,3B
 32 I  32: ;
 33 I  33: ; 2 FREE RUNNING CLOCKS
 34 I  34: ;    CLK_A, CLK_B
 35 I  35: ;
 36 I  36: ; ADDITIONAL REGISTERS FOR STATE DEFINITION
 37 I  37: ;    QQ1, QQ2
 38 I  38: ;-------------------------------------------------------------------
 39 I  39: ;
 40    40: CLK_1A, CLK_1B, CLK_2A, CLK_2B, CLK_3A, CLK_3B, CLK_A, CLK_B, QQ1, QQ2;
 41 I  41:
 42    42: *Z-VALUES
 43 I  43:
 44 I  44: ;                        ADDITIONAL OUTPUTS
 45 I  45: ;      SYSTEM OUTPUTS    FOR STATE DEFINITION
 46 I  46: ;      ______________    ____________________
 47 I  47: ;
 48 I  48: ;      C C C C C C C C   Q Q
 49 I  49: ;      L L L L L L L L   Q Q
 50 I  50: ;      K K K K K K K K   1 2
 51 I  51: ;      1 1 2 2 3 3 A B
 52 I  52: ;      A B A B A B
 53 I  53:
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
 54    54:  S1  = 0 0 0 0 0 0 0 0   - -           ; INIT  COMMON STATES
 55    55:  S2  = 0 0 0 0 0 0 0 1 0   0 -         ; SA       - INACTIVE
 56    56:  S3  = 0 0 0 0 0 0 0 0 1   0 -         ; SB         MODE STATES
 57 I  57:
 58    58:  S4  = 1 0 0 0 0 0 1 0   - 0           ; S1A    PIPELINE STATES
 59    59:  S5  = 0 1 0 0 0 0 0 1   - 0           ; S1B
 60    60:  S6  = 1 0 1 0 0 0 1 0   - -           ; S12A
 61    61:  S7  = 0 1 0 1 0 0 0 1   - -           ; S12B
 62    62:  S8  = 1 0 1 0 1 0 1 0   - -           ; S123A
 63    63:  S9  = 0 1 0 1 0 1 0 1   - -           ; S123B
 64   64:  S10 = 0 0 0 1 0 1 0 1  - -       ; S23B 65   65:  S11 = 0 0 0 0 1 0 1 0  - 0       ; S3A
 66    66:  S12 = 0 0 0 0 0 0 1 0 1   - 0         ; S3B
 67    67:  S13 = 0 0 0 0 0 0 0 1 0   1 0         ; SAW
 68    68:  S14 = 0 0 0 0 0 0 0 0 1   1 0         ; SBW
 69 I  69:
 70    70:  S15 = 1 0 0 0 0 0 1 0   - 1           ; S1AN   NON-PIPELINE
 71    71:  S16 = 0 1 0 0 0 0 0 1   - 1           ; S1BN
 72    72:  S17 = 0 0 1 0 0 0 1 0   - -           ; S2AN
 73    73:  S18 = 0 0 0 1 0 0 0 1   - -           ; S2BN
 74    74:  S19 = 0 0 0 0 1 0 1 0   - 1           ; S3AN
 75    75:  S20 = 0 0 0 0 0 0 1 0 1   - 1         ; S3BN
 76    76:  S21 = 0 0 0 0 0 0 0 1 0   1 1         ; SAWN
 77    77:  S22 = 0 0 0 0 0 0 0 0 1   1 1         ; SBWN
 78 I  78:
 79    79: *STRING
 80    80:  INIT    = 1                           ;  COMMON STATES
 81    81:  SA      = 2                           ;   -INACTIVE MODE
 82    82:  SB      = 3                           ;    STATES
 83 I  83:
 84    84:  S1A     = 4                           ;  PIPELINE STATES
 85    85:  S1B     = 5                           ;
 86    86:  S12A    = 6                           ;
 87    87:  S12B    = 7                           ;
 88    88:  S123A   = 8                           ;
 89    89:  S123B   = 9                           ;
 90    90:  S23B    = 10                          ;
 91    91:  S3A     = 11                          ;
 92    92:  S3B     = 12                          ;
 93    93:  SAW     = 13                          ;
 94    94:  SBW     = 14                          ;
 95 I  95:
 96    96:  S1AN    = 15                          ;  NON-PIPELINE
 97    97:  S1BN    = 16                          ;
 98    98:  S2AN    = 17                          ;
 99    99:  S2BN    = 18                          ;
100   100:  S3AN    = 19                          ;
101   101:  S3BN    = 20                          ;
102   102:  SAWN    = 21                          ;
103   103:  SBWN    = 22                          ;
104   104:  LASTSTATE = 22;
105 I 105:
106   106: *FLOW-TABLE
107 I 107: ;
108 I 108: ;-------------------------------------------------------------------------
109 I 109: ;RESET STATE
110 I 110: ;ALL STATES MUST RESET TO THE INITIAL STATE (ALL OUTPUTS REGISTERS 0)
UPON
111 I 111: ;AN ACTIVE RESET INPUT.  SINCE THE 20G10 HAS NO GLOBAL OR INDIVIDUAL
112 I 112: ;RESETS TO THE OUTPUT REGISTERS, RESET TO INITIAL STATE MUST BE EMBEDDED
113 I 113: ;INTO THE STATE MACHINE
114 I 114: ;
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
115   115: RELEVENT = RESET                          ;
116   116: S[1..'LASTSTATE'], X 1     , F 'INIT'  ;ALL STATE > INIT UPON RESET
117   138: RELEVENT = RESET = 0                      ;
118 I 139: ;
119 I 140: ;-------------------------------------------------------------------
120 I 141: ;INACTIVE MODE STATES
121   142: RELEVANT = RUN, NPL                       ;
122   143: S 'INIT'   , X - -         , F 'SA'    ;INITIAL STATE AFTER RESET
123 I 144:
124   145: S 'SA'     , X - -         , F 'SB'    ;INACTIVE MODE STATE, ONLY
125 I 146:
126   147: S 'SB'     , X 0 -         , F 'SA'    ;FREE RUN CLKS A & B ARE ACTIVE
127   148:             X 1 0         , F 'S1A'   ; PIPELINE VS.
128   149:             X 1 1         , F 'S1AN'  ; NON-PIPELINE DECISION
129 I 150:
130 I 151: ;-------------------------------------------------------------------
131 I 152: ;PIPELINE MODE STATES
132 I 153:
133   154: RELEVANT = INTR, IEN                      ;*PRIMING THE PIPELINE*
134   155: S 'S1A'    , X - -         , F 'S1B'   ;
135 I 156:
136   157: S 'S1B'    , X - -         , F 'S12A'  ;
137 I 158:
138   159: S 'S12A'   , X - -         , F 'S12B'  ;
139 I 160:
140   161: S 'S12B'   , X 1 1         , F 'S1A'   ;   INTERRUPT CONDITION ? YES
141   162:             X 1 0         , F 'S123A' ;                         NO
142   163:             X 0 -         , F 'S123A' ;                         NO
143 I 164:
144   165: RELEVANT =  RUN, INTR, IEN, WAIT, WEN; *FULL PIPELINE*
145   166: S 'S123A'  , X - - - 1 1 , F 'SBW'     ;   WAIT CONDITION
146   167:             X 0 - - 0 - , F 'S23B'    ;   /RUN COND., EMPTY PIPELINE
147   168:             X 0 - - 1 0 , F 'S23B'    ;   /RUN COND., EMPTY PIPELINE
148   169:             X 1 - - 0 - , F 'S123B'   ;   RUN CONDITION
149   170:             X 1 - - 1 0 , F 'S123B'   ;   RUN CONDITION
150 I 171:
151   172: S 'S123B'  , X - 1 1 - - , F 'S1A'     ;   INTERUPT CONDITION
152   173:             X - 0 - - - , F 'S123A'   ;   RUN CONDITION
153   174:             X - 1 0 - - , F 'S123A'   ;   RUN CONDITION
154 I 175:
155   176: RELEVANT = RUN                            ; *EMPTY PIPELINE*
156   177: S 'S23B'   , X -           , F 'S3A'   ;
157 I 178:
158   179: S 'S3A'    , X -           , F 'S3B'   ;
159 I 180:
160   181: S 'S3B'    , X -           , F 'SA'    ;   BACK TO INACTIVE STATE
161 I 182:
162   183: RELEVANT = WAIT                           ; *PIPELINE WAIT STATES*
163   184: S 'SBW'    , X 1           , F 'SAW'   ;   WAIT
164   185:             X 0           , F 'S123A' ;   /WAIT
165 I 186:
166   187: S 'SAW'    , X -           , F 'SBW'   ;
167 I 188:
168 I 189: ;-------------------------------------------------------------------
169 I 190: ;NON-PIPELINE MODE STATES
170 I 191:
171   192: S 'S1AN'   , X -           , F 'S1BN'  ;
172 I 193:
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
173   194: S 'S1BN'   , X -           , F 'S2AN'  ;
174 I 195:
175   196: RELEVANT = WAIT, WEN                   ;
176   197: S 'S2AN'   , X 1 1       , F 'SBWN'    ; WAIT CONDITION
177   198:              X 0 -       , F 'S2BN'    ; /WAIT CONDITION
178   199:              X 1 0       , F 'S2BN'    ; /WAIT CONDITION
179 I 200:180   201: RELEVANT = INTR, IEN                 ;
181   202: S 'S2BN'   , X 1 1       , F 'S1AN'    ; INTERRUPT CONDITION
182   203:              X 0 -       , F 'S3AN'    ; /INTERRUPT CONDITION
183   204:              X 1 0       , F 'S3AN'    ; /INTERRUPT CONDITION
184 I 205:
185   206: RELEVANT = RUN                         ;
186   207: S 'S3AN'   , X -           , F 'S3BN'  ;
187 I 208:
188   209: S 'S3BN'   , X 1           , F 'S1AN'  ;
189   210:              X 0           , F 'SA'    ; BACK TO INACTIVE STATE
190 I 211:
191   212: RELEVANT = WAIT                        ;*NON-PIPELINED WAIT STATES*
192   213: S 'SBWN'   , X 1           , F 'SAWN'  ;  REMAIN IN WAIT
193   214:              X 0           , F 'S2AN'  ;  END OF WAIT CONDITION
194 I 215:
195   216: S 'SAWN'   , X -           , F 'SBWN'  ;  REMAIN IN WAIT
196 I 217:
197   218: *STATE-ASSIGNMENT
198   219:  Z-VALUES
199 I 220:
200 I 221:
201   222: *PIN
202   223:  STATECLK = 1, RUN = 2, NPL = 3, INTR = 4, IEN = 5, WAIT = 6, WEN = 7,
203   223:  RESET = 8, CLK_1A = 14, CLK_1B = 15, CLK_2A = 16, CLK_2B = 17,
204   223:  CLK_3A = 18, CLK_3B = 19, CLK_A = 20, CLK_B = 21, QQ1 = 22, QQ2 = 23;
205 I 224:
206   225: *RUN-CONTROL
207   226:  LISTING= LONG,SYMBOL-TABLE,EQUATIONS,PINOUT;
208   227:  PROGFORMAT= L-EQUATIONS
209   228:  OPTIMIAZATION= P-TERMS;
210   229: *END
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

LOG/IC SYMBOL TABLE				
SYMBOL	TYPE	REG	LEVEL	PIN/NODE
GND	LOCAL	-	HIGH	
VCC	LOCAL	-	HIGH	
RUN	X-VARIABLE	-	HIGH	2
NPL	X-VARIABLE	-	HIGH	3
INTR	X-VARIABLE	-	HIGH	4
IEN	X-VARIABLE	-	HIGH	5
WAIT	X-VARIABLE	-	HIGH	6
WEN	X-VARIABLE	-	HIGH	7
RESET	X-VARIABLE	-	HIGH	8
CLK_1A	X-VARIABLE	-	HIGH	14
CLK_1B	X-VARIABLE	-	HIGH	15
CLK_2A	X-VARIABLE	-	HIGH	16
CLK_2B	X-VARIABLE	-	HIGH	17
CLK_3A	X-VARIABLE	-	HIGH	18
CLK_3B	X-VARIABLE	-	HIGH	19
CLK_A	X-VARIABLE	-	HIGH	20
CLK_B	X-VARIABLE	-	HIGH	21
QQ1	X-VARIABLE	-	HIGH	22
QQ2	X-VARIABLE	-	HIGH	23
CLK_1A.D	Z-VARIABLE	DFF	HIGH	14
CLK_1B.D	Z-VARIABLE	DFF	HIGH	15
CLK_2A.D	Z-VARIABLE	DFF	HIGH	16
CLK_2B.D	Z-VARIABLE	DFF	HIGH	17
CLK_3A.D	Z-VARIABLE	DFF	HIGH	18
CLK_3B.D	Z-VARIABLE	DFF	HIGH	19
CLK_A.D	Z-VARIABLE	DFF	HIGH	20
CLK_B.D	Z-VARIABLE	DFF	HIGH	21
QQ1.D	Z-VARIABLE	DFF	HIGH	22
QQ2.D	Z-VARIABLE	DFF	HIGH	23

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
EXPANDED FUNCTION TABLE (INCLUDING LOCAL VARIABLES):
---------------------------------------------------
                             :  CCCC CC
                             :  LLLL LLCC
              CCC CCC        :  KKKK KKLL
             RLLL LLLC C     :  ____ __KK QQ
      I W    EKKK KKKL L     :  1122 33__ QQ
   GVRN NIAW S___ ___K KQQ   :  ABAB ABAB 12
   NCUP TEIE E112 233_ _QQ   :  ....  ....  ..
   DCNL RNTN TABA BABA B12   :  DDDD DDDD DD
   ---------------------------------------
   ---- ---- 1000 0000 0--   :  0000 0000 --;     1/  116
   ---- ---- 0000 0000 0--   :  0000 0010 0-;     2/  143
   ---- ---- 1000 0001 00-   :  0000 0000 --;     3/  117
   ---- ---- 0000 0001 00-   :  0000 0001 0-;     4/  145
   ---- ---- 1000 0000 10-   :  0000 0000 --;     5/  118
   --0- ---- 0000 0000 10-   :  0000 0010 0-;     6/  147
   --10 ---- 0000 0000 10-   :  1000 0010 -0;     7/  148
   --11 ---- 0000 0000 10-   :  1000 0010 -1;     8/  149
   ---- ---- 1100 0001 0-0   :  0000 0000 --;     9/  119
   ---- ---- 0100 0001 0-0   :  0100 0001 -0;    10/  155
   ---- ---- 1010 0000 1-0   :  0000 0000 --;    11/  120
   ---- ---- 0010 0000 1-0   :  1010 0010 --;    12/  157
   ---- ---- 1101 0001 0--   :  0000 0000 --;    13/  121
   ---- ---- 0101 0001 0--   :  0101 0001 --;    14/  159
   ---- ---- 1010 1000 1--   :  0000 0000 --;    15/  122
   ---- 11-- 0010 1000 1--   :  1000 0010 -0;    16/  161
   ---- 10-- 0010 1000 1--   :  1010 1010 --;    17/  162
   ---- 0--- 0010 1000 1--   :  1010 1010 --;    18/  163
   ---- ---- 1101 0101 0--   :  0000 0000 --;    19/  123
   ---- --11 0101 0101 0--   :  0000 0001 10;    20/  166
   --0- --0- 0101 0101 0--   :  0001 0101 --;    21/  167
   --0- --10 0101 0101 0--   :  0001 0101 --;    22/  168
   --1- --0- 0101 0101 0--   :  0101 0101 --;    23/  169
   --1- --10 0101 0101 0--   :  0101 0101 --;    24/  170
   ---- ---- 1010 1010 1--   :  0000 0000 --;    25/  124
   ---- 11-- 0010 1010 1--   :  1000 0010 -0;    26/  172
   ---- 0--- 0010 1010 1--   :  1010 1010 --;    27/  173
   ---- 10-- 0010 1010 1--   :  1010 1010 --;    28/  174
   ---- ---- 1000 1010 1--   :  0000 0000 --;    29/  125
   ---- ---- 0000 1010 1--   :  0000 1010 -0;    30/  177
   ---- ---- 1000 0101 0-0   :  0000 0000 --;    31/  126
   ---- ---- 0000 0101 0-0   :  0000 0101 -0;    32/  179
   ---- ---- 1000 0010 1-0   :  0000 0000 --;    33/  127
   ---- ---- 0000 0010 1-0   :  0000 0010 0-;    34/  181
   ---- ---- 1000 0001 010   :  0000 0000 --;    35/  128
   ---- ---- 0000 0001 010   :  0000 0001 10;    36/  187
   ---- ---- 1000 0000 110   :  0000 0000 --;    37/  129
   ---- --1- 0000 0000 110   :  0000 0010 10;    38/  184
   ---- --0- 0000 0000 110   :  1010 1010 --;    39/  185
   ---- ---- 1100 0001 0-1   :  0000 0000 --;    40/  130
   ---- ---- 0100 0001 0-1   :  0100 0001 -1;    41/  192
   ---- ---- 1010 0000 1-1   :  0000 0000 --;    42/  131
   ---- ---- 0010 0000 1-1   :  0010 0010 --;    43/  194
   ---- ---- 1001 0001 0--   :  0000 0000 --;    44/  132
   ---- --11 0001 0001 0--   :  0000 0001 11;    45/  197
   ---- --0- 0001 0001 0--   :  0001 0001 --;    46/  198
   ---- --10 0001 0001 0--   :  0001 0001 --;    47/  199
   ---- ---- 1000 1000 1--   :  0000 0000 --;    48/  133
   ---- 11-- 0000 1000 1--   :  1000 0010 -1;    49/  202
   ---- 0--- 0000 1000 1--   :  0000 1010 -1;    50/  203
   ---- 10-- 0000 1000 1--   :  0000 1010 -1;    51/  204
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
    ---- ---- 1000 0101 0-1  :  0000 0000 --;    52/  134
    ---- ---- 0000 0101 0-1  :  0000 0101 -1;    53/  207
    ---- ---- 1000 0010 1-1  :  0000 0000 --;    54/  135
    --1- ---- 0000 0010 1-1  :  1000 0010 -1;    55/  209
    --0- ---- 0000 0010 1-1  :  0000 0010 0-;    56/  210
    ---- ---- 1000 0001 011  :  0000 0000 --;    57/  136
    ---- ---- 0000 0001 011  :  0000 0001 11;    58/  216
    ---- ---- 1000 0000 111  :  0000 0000 --;    59/  137
    ---- --1- 0000 0000 111  :  0000 0010 11;    60/  213
    ---- --0- 0000 0000 111  :  0010 0010 --;    61/  214
    REST                     :  ---- ---- --;    62
    ---------------------------------------
    1234 5678 9012 3456 789     1234 5678 90


  STATE ASSIGNMENT:
-----------------
    CCCC CC
    LLLL LLCC
    KKKK KKLL
    ____ __KK QQ
    1122 33__ QQ
    ABAB ABAB 12
    ------------
    0000 0000 --; 1
    0000 0010 0-; 2
    0000 0001 0-; 3
    1000 0010 -0; 4
    0100 0001 -0; 5
    1010 0010 --; 6
    0101 0001 --; 7
    1010 1010 --; 8
    0101 0101 --; 9
    0001 0101 --; 10
    0000 1010 -0; 11
    0000 0101 -0; 12
    0000 0010 10; 13
    0000 0001 10; 14
    1000 0010 -1; 15
    0100 0001 -1; 16
    0010 0010 --; 17
    0001 0001 --; 18
    0000 1010 -1; 19
    0000 0101 -1; 20
    0000 0010 11; 21
    0000 0001 11; 22
EXPANDED FUNCTION TABLE (LOCAL VARIABLES REMOVED):
--------------------------------------------------
                           :  CCCC CC
                           :  LLLL LLCC
            C CCCC C       :  KKKK KKLL
           RL LLLL LCC     :  ____ __KK QQ
      I  W EK KKKK KLL     :  1122 33__ QQ
    RNNI AWS_ ____ _KKQ Q  :  ABAB ABAB 12
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
    UPTE IEE1 1223 3__Q Q  :  ....  ....  ..
    NLRN TNTA BABA BAB1 2  :  DDDD DDDD DD
    ------------------------------------
    ---- --10 0000 000- -  :  0000 0000 --;     1/  116
    ---- --00 0000 000- -  :  0000 0010 0-;     2/  143
    ---- --10 0000 0100 -  :  0000 0000 --;     3/  117
    ---- --00 0000 0100 -  :  0000 0001 0-;     4/  145
    ---- --10 0000 0010 -  :  0000 0000 --;     5/  118
    0--- --00 0000 0010 -  :  0000 0010 0-;     6/  147
    10-- --00 0000 0010 -  :  1000 0010 -0;     7/  148
    11-- --00 0000 0010 -  :  1000 0010 -1;     8/  149
    ---- --11 0000 010- 0  :  0000 0000 --;     9/  119
    ---- --01 0000 010- 0  :  0100 0001 -0;    10/  155
    ---- --10 1000 001- 0  :  0000 0000 --;    11/  120
    ---- --00 1000 001- 0  :  1010 0010 --;    12/  157
    ---- --11 0100 010- -  :  0000 0000 --;    13/  121
    ---- --01 0100 010- -  :  0101 0001 --;    14/  159
    ---- --10 1010 001- -  :  0000 0000 --;    15/  122
    --11 --00 1010 001- -  :  1000 0010 -0;    16/  161
    --10 --00 1010 001- -  :  1010 1010 --;    17/  162
    --0- --00 1010 001- -  :  1010 1010 --;    18/  163
    ---- --11 0101 010- -  :  0000 0000 --;    19/  123
    ---- 1101 0101 010- -  :  0000 0001 10;    20/  166
    0--- 0-01 0101 010- -  :  0001 0101 --;    21/  167
    0--- 1001 0101 010- -  :  0001 0101 --;    22/  168
    1--- 0-01 0101 010- -  :  0101 0101 --;    23/  169
    1--- 1001 0101 010- -  :  0101 0101 --;    24/  170
    ---- --10 1010 101- -  :  0000 0000 --;    25/  124
    --11 --00 1010 101- -  :  1000 0010 -0;    26/  172
    --0- --00 1010 101- -  :  1010 1010 --;    27/  173
    --10 --00 1010 101- -  :  1010 1010 --;    28/  174
    ---- --10 0010 101- -  :  0000 0000 --;    29/  125
    ---- --00 0010 101- -  :  0000 1010 -0;    30/  177
    ---- --10 0001 010- 0  :  0000 0000 --;    31/  126
    ---- --00 0001 010- 0  :  0000 0101 -0;    32/  179
    ---- --10 0000 101- 0  :  0000 0000 --;    33/  127
    ---- --00 0000 101- 0  :  0000 0010 0-;    34/  181
    ---- --10 0000 0101 0  :  0000 0000 --;    35/  128
    ---- --00 0000 0101 0  :  0000 0001 10;    36/  187
    ---- --10 0000 0011 0  :  0000 0000 --;    37/  129
    ---- 1-00 0000 0011 0  :  0000 0010 10;    38/  184
    ---- 0-00 0000 0011 0  :  1010 1010 --;    39/  185
    ---- --11 0000 010- 1  :  0000 0000 --;    40/  130
EXPANDED FUNCTION TABLE (LOCAL VARIABLES REMOVED)- continued :
    ---- --01 0000 010- 1  :  0100 0001 -1;    41/  192
    ---- --10 1000 001- 1  :  0000 0000 --;    42/  131
    ---- --00 1000 001- 1  :  0010 0010 --;    43/  194
    ---- --10 0100 010- -  :  0000 0000 --;    44/  132
    ---- 1100 0100 010- -  :  0000 0001 11;    45/  197
    ---- 0-00 0100 010- -  :  0001 0001 --;    46/  198
    ---- 1000 0100 010- -  :  0001 0001 --;    47/  199
    ---- --10 0010 001- -  :  0000 0000 --;    48/  133
    --11 --00 0010 001- -  :  1000 0010 -1;    49/  202
    --0- --00 0010 001- -  :  0000 1010 -1;    50/  203
    --10 --00 0010 001- -  :  0000 1010 -1;    51/  204
    ---- --10 0001 010- 1  :  0000 0000 --;    52/  134
    ---- --00 0001 010- 1  :  0000 0101 -1;    53/  207
    ---- --10 0000 101- 1  :  0000 0000 --;    54/  135
    1--- --00 0000 101- 1  :  1000 0010 -1;    55/  209
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
      0--- --00 0000 101- 1  :  0000 0010 0-;    56/  210
      ---- --10 0000 0101 1  :  0000 0000 --;    57/  136
      ---- --00 0000 0101 1  :  0000 0001 11;    58/  216
      ---- --10 0000 0011 1  :  0000 0000 --;    59/  137
      ---- 1-00 0000 0011 1  :  0000 0010 11;    60/  213
      ---- 0-00 0000 0011 1  :  0010 0010 --;    61/  214
      REST                   :  ---- ---- --;    62
      ---------------------------------------
      1234 5678 9012 3456 7     1234 5678 90
  PIPELINED CLOCKING SYSTEM OD20G10
  CYPRESS SEMICONDUCTOR
  90/03/15  23:49:45
  ***************************************************
  ***  NET DESCRIPTION TABLE FOR AND/OR STRUCTURE  ***
  ***************************************************
                         :  CCCC CC
                         :  LLLL LLCC
        C CCCC C         :  KKKK KKLL
       RL LLLL LCC       :  ____ __KK QQ
     I  W EK KKKK KLL      :  1122 33__ QQ
RNNI AWS_ ____ _KKQ Q  :  ABAB ABAB 12
UPTE IEE1 1223 3__Q Q  :  ....  ....  ..
NLRN TNTA BABA BAB1 2  :  DDDD DDDD DD
---------------------------------------
                     INV  ....  ....  ..
                     REG  DDDD DDDD DD
---------------------------------------
---- 0-0- --0- 0-11 0  :  A...  ....  ..  ;   1
1--- --0- --0- 1--- 1  :  A...  ....  ..  ;   2
---- --0- 1--- ---- 0  :  A...  ....  ..  ;   3
---- --0- 1-1- ---- -  :  A...  ....  ..  ;   4
--11 --0- --1- 0--- -  :  A...  ....  ..  ;   5
1--- --0- 0-0- 0-10 -  :  A...  ....  ..  ;   6
---- --01 ---0 ---- -  :  .A..  ....  ..  ;   7
1--- -001 ---- ---- -  :  .A..  ....  ..  ;   8
1--- 0-01 ---- ---- -  :  .A..  ....  ..  ;   9
---0 --0- 1--- ---- -  :  ..A.  ....  ..  ;  10
--0- --0- 1--- ---- -  :  ..A.  ....  ..  ;  11
---- 0-0- --0- 0-11 -  :  ..A.  ....  ..  ;  12
---- --0- 1-0- ---- -  :  ..A.  ....  ..  ;  13
---- 0-0- -1-- ---- -  :  ...A ....  ..  ;  14
---- -00- -1-- ---- -  :  ...A ....  ..  ;  15
---- --01 -1-0 ---- -  :  ...A ....  ..  ;  16
---0 --0- --1- ---- -  :  ....  A...  ..  ;  17
--0- --0- --1- ---- -  :  ....  A...  ..  ;  18
---- --0- 0-1- 1--- -  :  ....  A...  ..  ;  19
---- 0-0- 0-0- 0-11 0  :  ....  A...  ..  ;  20
---- 0-0- ---1 ---- -  :  ....  .A..  ..  ;  21
---- -00- ---1 ---- -  :  ....  .A..  ..  ;  22
---- --0- -0-1 ---- -  :  ....  .A..  ..  ;  23
---- --0- ---- -0-- -  :  ....  ..A.  ..  ;  24
---- --0- ---- -1-- -  :  ....  ...A ..  ;  25
---- ---- ---- -1-1 -  :  ....  ....  A.  ;  26
---- ---- ---- 0-11 -  :  ....  ....  A.  ;  27
---- ---- -1-- ---- -  :  ....  ....  A.  ;  28
---- ---- --0- 1--- -  :  ....  ....  .A ;  29
---- ---- 0-1- 0--- -  :  ....  ....  .A ;  30
---- ---0 -1-- ---- -  :  ....  ....  .A ;  31
---- ---- -0-- -1-- 1  :  ....  ....  .A ;  32
-1-- ---0 0--0 0--0 -  :  ....  ....  .A ;  33
---- ---- 00-- 0--1 1  :  ....  ....  .A ;  34
---------------------------------------
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

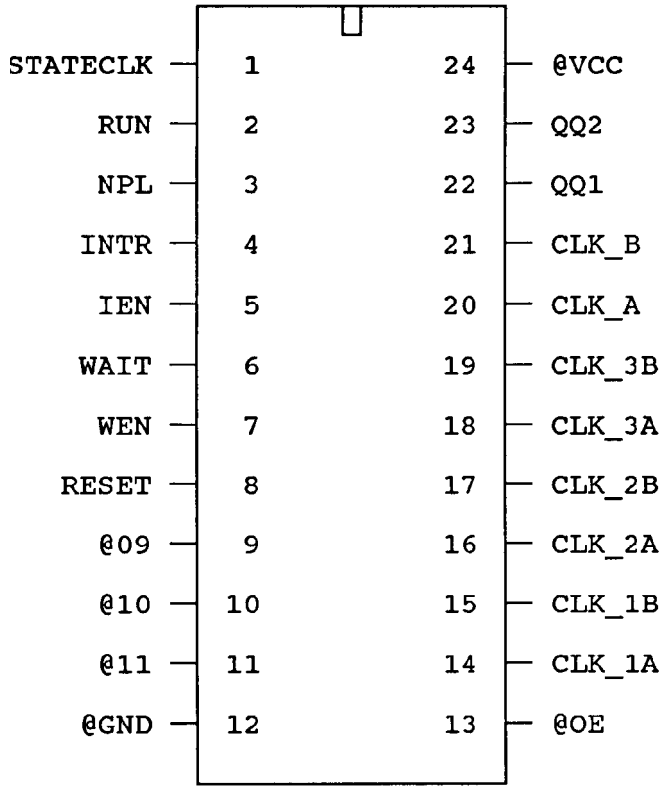
```
1234 5678 9012 3456 7  :  1234 5678 90
   PIPELINED CLOCKING SYSTEM OD20G10
   CYPRESS SEMICONDUCTOR
   90/03/15  23:49:45
   ************************************************
   ***              BOOLEAN EQUATIONS             ***
   ************************************************


  CLK_1A.D     :=
           /WAIT   & /RESET  & /CLK_2B       & /CLK_3B       &  CLK_B
                   &  QQ1    & /QQ2
         +  RUN    & /RESET  & /CLK_2B       &  CLK_3B       &  QQ2
         + /RESET  &  CLK_1B       & /QQ2
         + /RESET  &  CLK_1B       &  CLK_2B
         +  INTR   &  IEN    & /RESET  &  CLK_2B       & /CLK_3B         +  RUN    & /RESET  &
/CLK_1B       & /CLK_2B       & /CLK_3B
                   &  CLK_B  & /QQ1    ;
 CLK_1B.D     :=
           /RESET  &  CLK_1A       & /CLK_3A
         +  RUN    & /WEN    & /RESET  &  CLK_1A
         +  RUN    & /WAIT   & /RESET  &  CLK_1A       ;
 CLK_2A.D     :=
           /IEN    & /RESET  &  CLK_1B
         + /INTR   & /RESET  &  CLK_1B
         + /WAIT   & /RESET  & /CLK_2B       & /CLK_3B       &  CLK_B
                   &  QQ1
         + /RESET  &  CLK_1B       & /CLK_2B       ;
 CLK_2B.D     :=
           /WAIT   & /RESET  &  CLK_2A
         + /WEN    & /RESET  &  CLK_2A
         + /RESET  &  CLK_1A       &  CLK_2A       & /CLK_3A       ;
 CLK_3A.D     :=
           /IEN    & /RESET  &  CLK_2B
         + /INTR   & /RESET  &  CLK_2B
         + /RESET  & /CLK_1B       &  CLK_2B       &  CLK_3B
         + /WAIT   & /RESET  & /CLK_1B       & /CLK_2B       & /CLK_3B
                   &  CLK_B  &  QQ1    & /QQ2    ;
 CLK_3B.D     :=
           /WAIT   & /RESET  &  CLK_3A
         + /WEN    & /RESET  &  CLK_3A
         + /RESET  & /CLK_2A       &  CLK_3A       ;
 CLK_A.D      :=
           /RESET  & /CLK_A  ; CLK_B.D      :=
           /RESET  &  CLK_A  ;
 QQ1.D  :=  CLK_A  &  QQ1
         + /CLK_3B       &  CLK_B  &  QQ1
         +  CLK_2A       ;
 QQ2.D  := /CLK_2B       &  CLK_3B
         + /CLK_1B       &  CLK_2B       & /CLK_3B
         + /CLK_1A       &  CLK_2A
         + /CLK_2A       &  CLK_A  &  QQ2
         +  NPL    & /CLK_1A       & /CLK_1B       & /CLK_3A
                   & /CLK_3B       & /QQ1
         + /CLK_1B       & /CLK_2A       & /CLK_3B       &  QQ1    &  QQ2    ;
```

Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

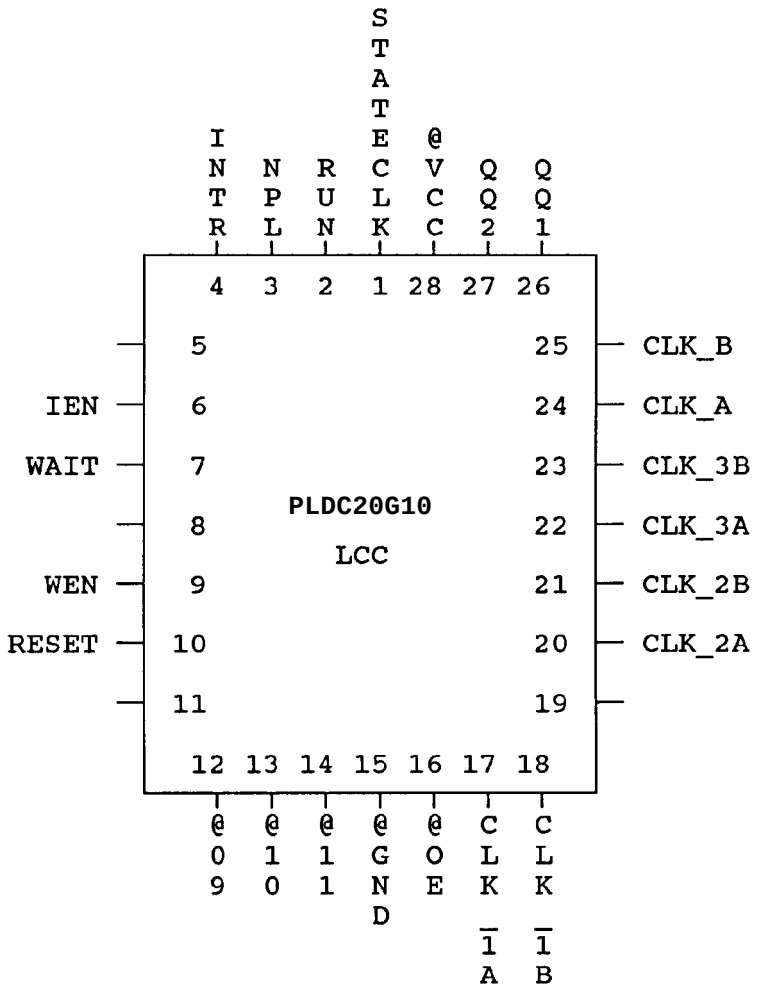
```
 PIPELINED CLOCKING SYSTEM OD20G10
CYPRESS SEMICONDUCTOR
90/03/15  23:49:45
```

PLDC20G10



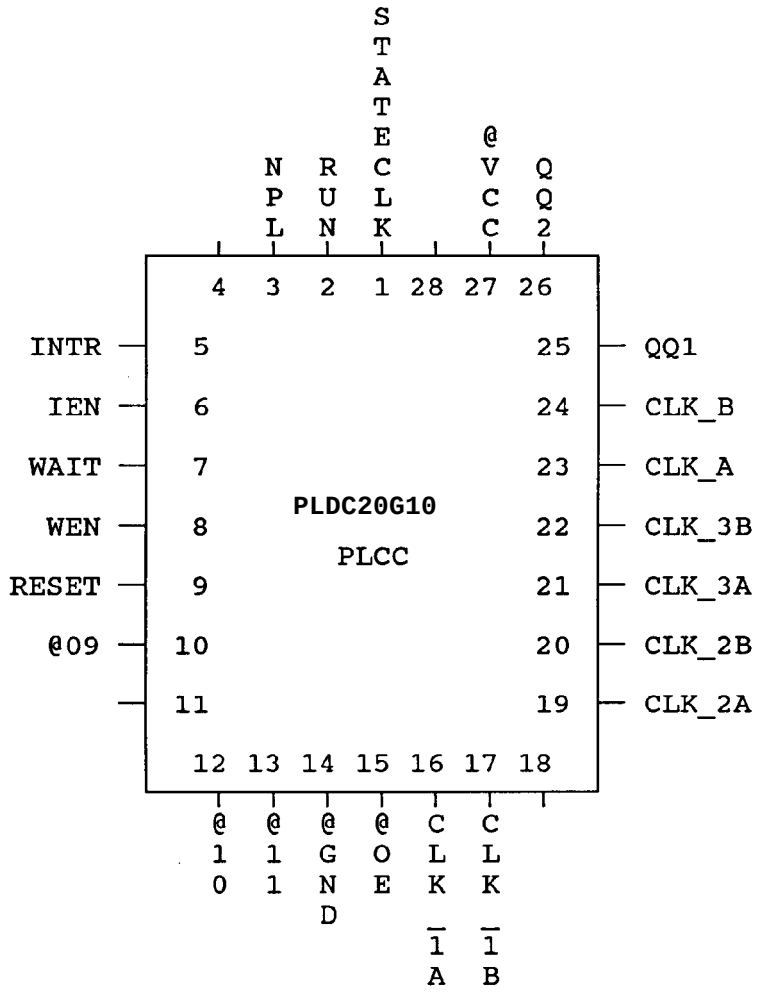
Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
PIPELINED CLOCKING SYSTEM OD20G10
CYPRESS SEMICONDUCTOR
90/03/15  23:49:45
```



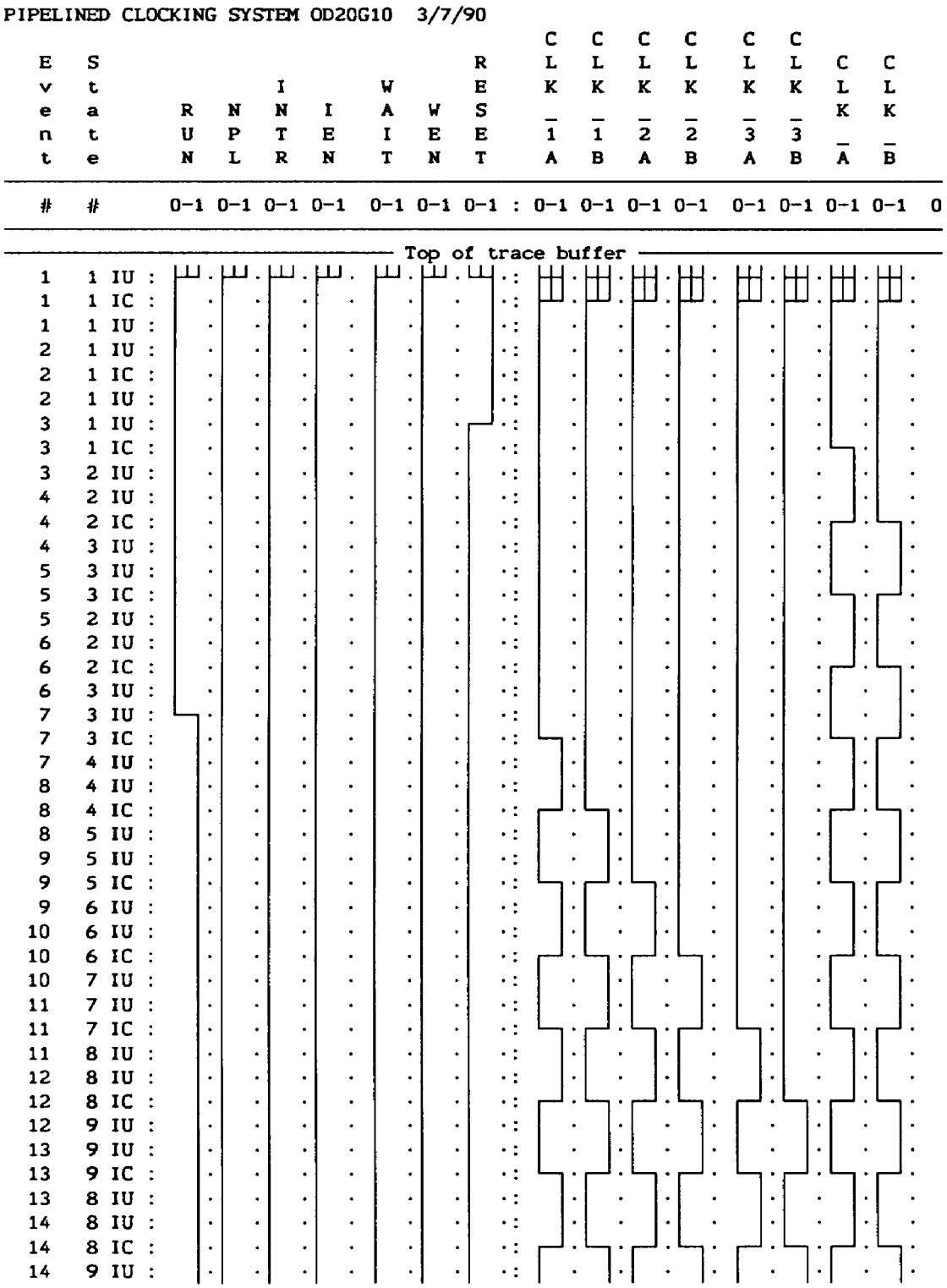
Appendix A. LOG/iC PLD Source Code: Clock State Machine (continued)

```
PIPELINED CLOCKING SYSTEM OD20G10
CYPRESS SEMICONDUCTOR
90/03/15  23:49:45
```




```
-------------------------------------------------
LOG/iC  -  PAL            CPU TIME USED:      45 SEC
```

Appendix B. LOG/iC Simulation: Clock State Machine



Appendix B. LOG/iC Simulation: Clock State Machine (continued)

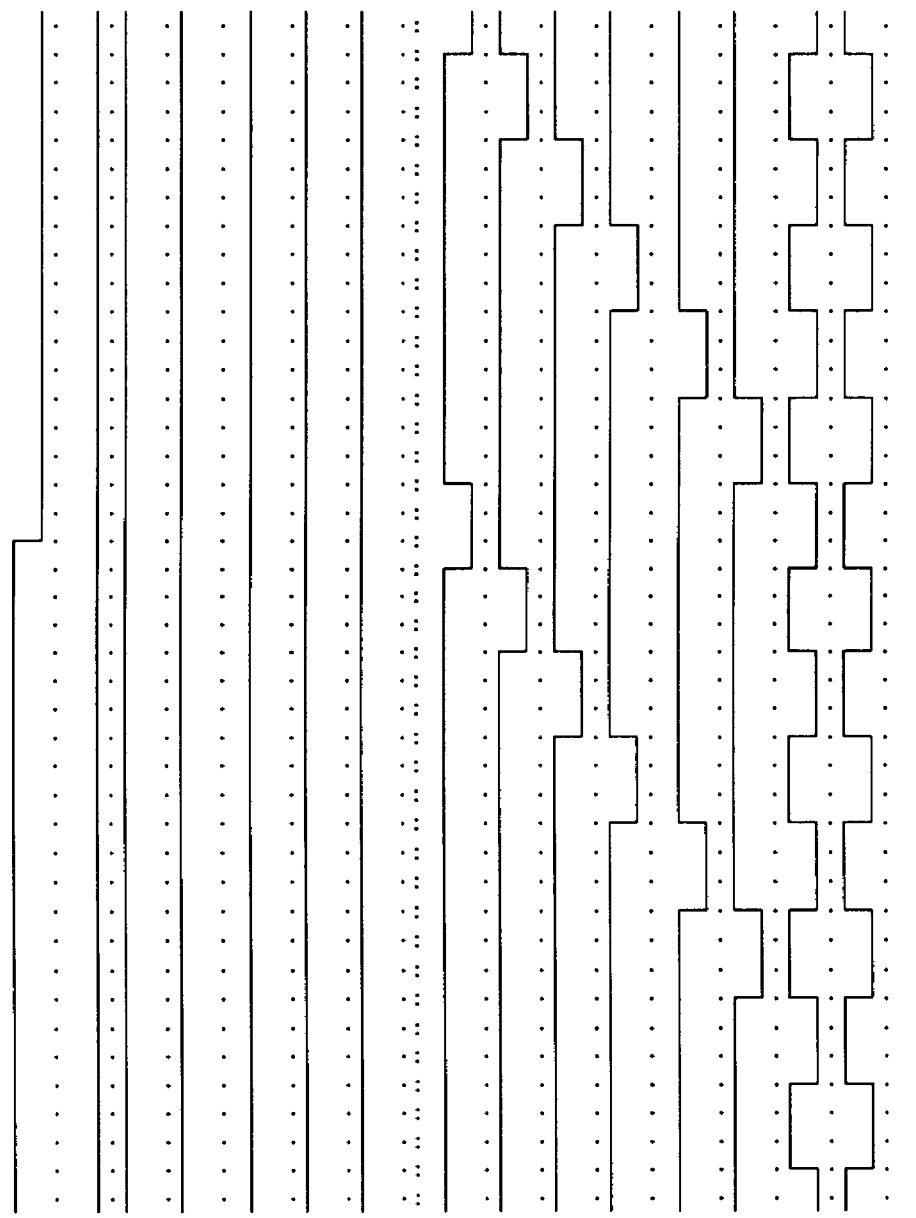
PIPELINED CLOCKING SYSTEM OD20G10 3/7/90

Event #	State #		RUN	NPL	INTR	IEN	WAIT	WEN	RES	:	CLK_1A	CLK_1B	CLK_2A	CLK_2B	CLK_3A	CLK_3B	CLK_A	CLK_B	
#	#		0-1	0-1	0-1	0-1	0-1	0-1	0-1	:	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0
15	9	IU																	
15	9	IC																	
15	8	IU																	
16	8	IU																	
16	8	IC																	
16	10	IU																	
17	10	IU																	
17	10	IC																	
17	11	IU																	
18	11	IU																	
18	11	IC																	
18	12	IU																	
19	12	IU																	
19	12	IC																	
19	2	IU																	
20	2	IU																	
20	2	IC																	
20	3	IU																	
21	3	IU																	
21	3	IC																	
21	2	IU																	
22	2	IU																	
22	2	IC																	
22	3	IU																	
23	3	IU																	
23	3	IC																	
23	15	IU																	
24	15	IU																	
24	15	IC																	
24	16	IU																	
25	16	IU																	
25	16	IC																	
25	17	IU																	
26	17	IU																	
26	17	IC																	
26	18	IU																	
27	18	IU																	
27	18	IC																	
27	19	IU																	
28	19	IU																	
28	19	IC																	
28	20	IU																	
29	20	IU																	
29	20	IC																	
29	15	IU																	

Appendix B. LOG/iC Simulation: Clock State Machine (continued)

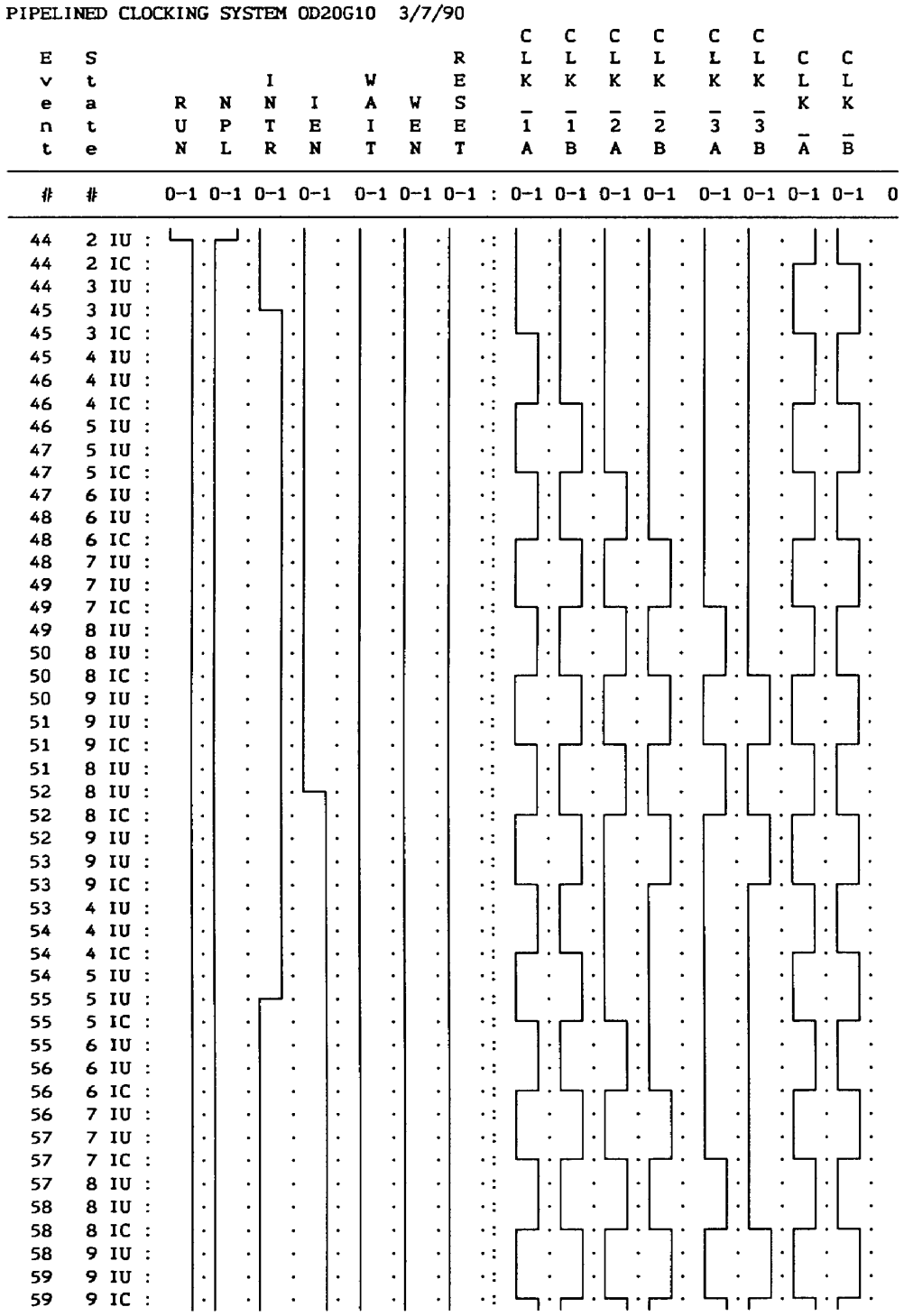
```
PIPELINED CLOCKING SYSTEM OD20G10   3/7/90
                                                C   C   C   C    C   C
E  S                                    R       L   L   L   L    L   L   C   C
v  t            I           W           E       K   K   K   K    K   K   L   L
e  a     R   N  N   I       A   W       S       _   _   _   _    _   _   K   K
n  t     U   P  T   E       I   E       E       1   1   2   2    3   3   _   _
t  e     N   L  R   N       T   N       T       A   B   A   B    A   B   A   B
---------------------------------------------------------------------------------
#  #    0-1 0-1 0-1 0-1   0-1 0-1 0-1 : 0-1 0-1 0-1 0-1   0-1 0-1 0-1 0-1   0
---------------------------------------------------------------------------------
```

Event #	State #		
30	15	IU	:
30	15	IC	:
30	16	IU	:
31	16	IU	:
31	16	IC	:
31	17	IU	:
32	17	IU	:
32	17	IC	:
32	18	IU	:
33	18	IU	:
33	18	IC	:
33	19	IU	:
34	19	IU	:
34	19	IC	:
34	20	IU	:
35	20	IU	:
35	20	IC	:
35	15	IU	:
36	15	IU	:
36	15	IC	:
36	16	IU	:
37	16	IU	:
37	16	IC	:
37	17	IU	:
38	17	IU	:
38	17	IC	:
38	18	IU	:
39	18	IU	:
39	18	IC	:
39	19	IU	:
40	19	IU	:
40	19	IC	:
40	20	IU	:
41	20	IU	:
41	20	IC	:
41	2	IU	:
42	2	IU	:
42	2	IC	:
42	3	IU	:
43	3	IU	:
43	3	IC	:
43	2	IU	:



Appendix B. LOG/iC Simulation: Clock State Machine (continued)

PIPELINED CLOCKING SYSTEM OD20G10 3/7/90



Appendix B. LOG/iC Simulation: Clock State Machine (continued)

PIPELINED CLOCKING SYSTEM OD20G10 3/7/90

Event	State	RUN	NPL	INTR	IEN	WAIT	WEN	RESET	CLK_1A	CLK_1B	CLK_2A	CLK_2B	CLK_3A	CLK_3B	CLK_A	CLK_B
#	#	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1
59	8 IU															
60	8 IU															
60	8 IC															
60	9 IU															
61	9 IU															
61	9 IC															
61	8 IU															
62	8 IU															
62	8 IC															
62	14 I															
63	14 I															
63	14 IC															
63	13 I															
64	13 I															
64	13 IC															
64	14 I															
65	14 I															
65	14 IC															
65	13 I															
66	13 I															
66	13 IC															
66	14 I															
67	14 I															
67	14 IC															
67	8 IU															
68	8 IU															
68	8 IC															
68	9 IU															
69	9 IU															
69	9 IC															
69	8 IU															
70	8 IU															
70	8 IC															
70	9 IU															
71	9 IU															
71	9 IC															
71	1 IU															
72	1 IU															
72	1 IC															
72	1 IU															
73	1 IU															
73	1 IC															

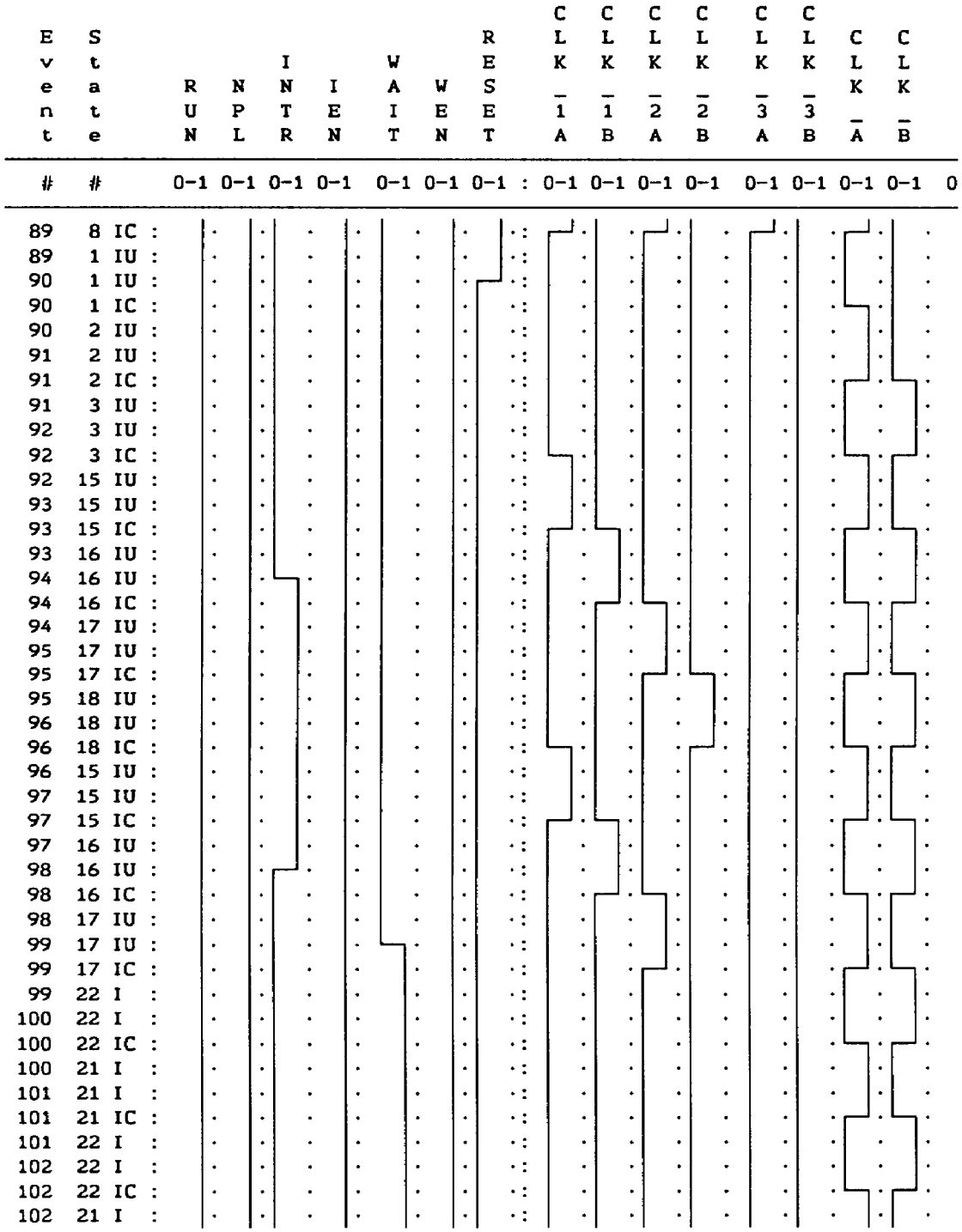
Appendix B. LOG/iC Simulation: Clock State Machine (continued)

PIPELINED CLOCKING SYSTEM OD20G10 3/7/90

Event #	State #	RUN	NPL	INTR	IEN	WAIT	WEN	RES	CLK 1A	CLK 1B	CLK 2A	CLK 2B	CLK 3A	CLK 3B	CLK A	CLK B
		0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1
73	1 IU															
74	1 IU															
74	1 IC															
74	2 IU															
75	2 IU															
75	2 IC															
75	3 IU															
76	3 IU															
76	3 IC															
76	4 IU															
77	4 IU															
77	4 IC															
77	5 IU															
78	5 IU															
78	5 IC															
78	6 IU															
79	6 IU															
79	6 IC															
79	7 IU															
80	7 IU															
80	7 IC															
80	4 IU															
81	4 IU															
81	4 IC															
81	5 IU															
82	5 IU															
82	5 IC															
82	6 IU															
83	6 IU															
83	6 IC															
83	7 IU															
84	7 IU															
84	7 IC															
84	8 IU															
85	8 IU															
85	8 IC															
85	9 IU															
86	9 IU															
86	9 IC															
86	8 IU															
87	8 IU															
87	8 IC															
87	9 IU															
88	9 IU															
88	9 IC															
88	8 IU															
89	8 IU															

Appendix B. LOG/iC Simulation: Clock State Machine (continued)

PIPELINED CLOCKING SYSTEM 0D20G10 3/7/90



Appendix B. LOG/iC Simulation: Clock State Machine (continued)

PIPELINED CLOCKING SYSTEM OD20G10 3/7/90

Event	State	RUN	NPL	INTR	IEN	WAIT	WEN	RESET	CLK 1A	CLK 1B	CLK 2A	CLK 2B	CLK 3A	CLK 3B	CLK A	CLK B
#	#	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1	0-1
103	21 I															
103	21 IC															
103	22 I															
104	22 I															
104	22 IC															
104	17 IU															
105	17 IU															
105	17 IC															
105	18 IU															
106	18 IU															
106	18 IC															
106	19 IU															
107	19 IU															
107	19 IC															
107	20 IU															
108	20 IU															
108	20 IC															
108	15 IU															
109	15 IU															
109	15 IC															
109	16 IU															
110	16 IU															
110	16 IC															
110	17 IU															
111	17 IU															
111	17 IC															
111	18 IU															
112	18 IU															
112	18 IC															
112	19 IU															
113	19 IU															
113	19 IC															
113	20 IU															
114	20 IU															
114	20 IC															
114	2 IU															
115	2 IU															
115	2 IC															
115	3 IU															
116	3 IU															
116	3 IC															
116	2 IU															

LOG/iC is a trademark of Isdata Corporation.