7.7 Clocks

(Original section[1])

Before we discuss memory elements and sequential logic, it is useful to discuss briefly the topic of clocks. This short section introduces the topic and is similar to the discussion found in COD Section 4.2 (Logic design conventions). More details on clocking and timing methodologies are presented in COD Section A.11 (Timing methodologies).

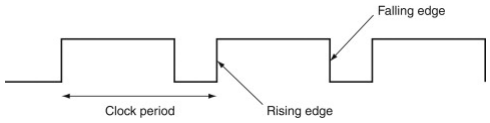
Clocks are needed in sequential logic to decide when an element that contains state should be updated. A clock is simply a free-running signal with a fixed cycle time; the clock frequency is simply the inverse of the cycle time. As shown in the figure below, the clock cycle time or clock period is divided into two portions: when the clock is high and when the clock is low. In this text, we use only *edge-triggered clocking*. This means that all state changes occur on a clock edge. We use an edge-triggered methodology because it is simpler to explain. Depending on the technology, it may or may not be the best choice for a *clocking methodology*.

Edge-triggered clocking: A clocking scheme in which all state changes occur on a clock edge.

Clocking methodology: The approach used to determine when data are valid and stable relative to the clock.

Figure 7.7.1: A clock signal oscillates between high and low values (COD Figure A.7.1).

The clock period is the time for one full cycle. In an edge-triggered design, either the rising or falling edge of the clock is active and causes state to be changed.



In an edge-triggered methodology, either the rising edge or the falling edge of the clock is *active* and causes state changes to occur. As we will see in the next section, the *state elements* in an edge-triggered design are implemented so that the contents of the state elements only change on the active clock edge. The choice of which edge is active is influenced by the implementation technology and does not affect the concepts involved in designing the logic.

State element: A memory element.

The clock edge acts as a sampling signal, causing the value of the data input to a state element to be sampled and stored in the state element. Using an edge trigger means that the sampling process is essentially instantaneous, eliminating problems that could occur if signals were sampled at slightly different times.

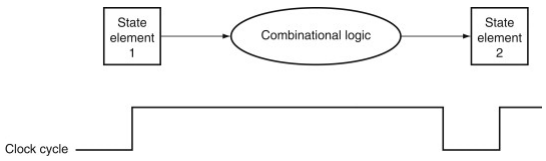
The major constraint in a clocked system, also called a *synchronous system*, is that the signals that are written into state elements must be valid when the active clock edge occurs. A signal is valid if it is stable (i.e., not changing), and the value will not change again until the inputs change. Since combinational circuits cannot have feedback, if the inputs to a combinational logic unit are not changed, the outputs will eventually become valid.

Synchronous system: A memory system that employs clocks and where data signals are read only when the clock indicates that the signal values are stable.

The figure below shows the relationship among the state elements and the combinational logic blocks in a synchronous, sequential logic design. The state elements, whose outputs change only after the clock edge, provide valid inputs to the combinational logic block. To ensure that the values written into the state elements on the active clock edge are valid, the clock must have a long enough period so that all the signals in the combinational logic block stabilize, and then the clock edge samples those values for storage in the state elements. This constraint sets a lower bound on the length of the clock period, which must be long enough for all state element inputs to be valid.

Figure 7.7.2: The inputs to a combinational logic block come from a state element, and the outputs are written into a state element (COD Figure A.7.2).

The clock edge determines when the contents of the state elements are updated.



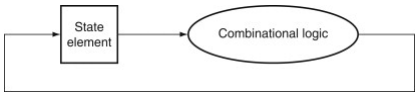
In the rest of this appendix, as well as in COD Chapter 4 (The Processor), we usually omit the clock signal, since we are assuming that all state elements are updated on the same clock edge. Some state elements will be written on every clock edge, while others will be written only under certain conditions (such as a register being updated). In such cases, we will have an explicit write signal for that state element. The write signal must still be gated with the clock so that the update occurs only on the clock edge if the write signal is active. We will see how this is done and used in the next section.

One other advantage of an edge-triggered methodology is that it is possible to have a state element that is used as both an input and output to the same combinational logic block, as shown in the figure below. In practice, care must be taken to prevent races in such situations and to ensure that the clock period is long enough; this topic is discussed further in COD Section A.11 (Timing methodologies).

Figure 7.7.3: An edge-triggered methodology allows a state element to be read and written in the same clock cycle without creating a race that could

lead to undetermined data values (COD Figure A.7.3).

Of course, the clock cycle must still be long enough so that the input values are stable when the active clock edge occurs.



Now that we have discussed how clocking is used to update state elements, we can discuss how to construct the state elements.

Elaboration

Occasionally, designers find it useful to have a small number of state elements that change on the opposite clock edge from the majority of the state elements. Doing so requires extreme care, because such an approach has effects on both the inputs and the outputs of the state element. Why then would designers ever do this? Consider the case where the amount of combinational logic before and after a state element is small enough so that each could operate in one-half clock cycle, rather than the more usual full clock cycle. Then the state element can be written on the clock edge corresponding to a half clock cycle, since the inputs and outputs will both be usable after one-half clock cycle. One common place where this technique is used is in register files, where simply reading or writing the register file can often be done in half the normal clock cycle. COD Chapter 4 (The Processor) makes use of this idea to reduce the pipelining overhead.

Register file: A state element that consists of a set of registers that can be read and written by supplying a register number to be accessed.

(*1) This section is in original form.

Provide feedback on this section