```
always @(posedge CpuClk)
begin
  if (!Reset_) begin
    Prescaler[13:0] <= 14´h0;
    Timer[7:0] <= 8´h0;
    TimerRollOver <= 1´b0;
  end
  else
    if (Prescaler[13:0] == 14´d9999) begin
      Prescaler[13:0] <= 14´h0; // roll-back to zero
      if (Timer[7:0] == TermCount[7:0]) begin
        Timer[7:0] <= 8´h0; // start over
        TimerRollOver <= 1´b1; // trigger other logic
      end
    seel
        Timer[7:0] <= Timer[7:0] + 1;
    end
    else begin
      Prescaler[13:0] <= Prescaler[13:0] + 1;
      TimerRollOver <= 1´b0;
    end
end
```

FIGURE 10.10 Timer logic.

There are several scenarios that arise from clocking a flop input with unknown timing as shown in Fig. 10.11. First, there is a chance that the applied signal will be captured successfully if it happens to meet the flop's setup time specification. Second, there is a chance that the input data will be missed on the first clock cycle, because it occurs too late for the flop to detect it. If the data remains asserted through the next cycle, it will be properly captured at that time.

Finally, there is a questionable area between capturing and missing the signal. A flop is inherently an analog circuit, because it is built from transistors. Flops behave digitally when their timing specifications are adhered to. When timing violations occur, the flop circuit may behave in a nondigital manner and generate a marginal output that is somewhere between 1 and 0 before eventually settling to a valid logic state. It is not certain into which logic state the flop will settle, nor is it certain exactly how much time the flop will take to settle. This phenomenon is known as *metastability*. A flop is said to be metastable when a timing violation occurs and it takes some time for the output to stabilize. Metastability does not damage the flop, but it can cause significant trouble for a synchronous circuit that is designed with the assumption of predictable timing.

Clock domains can be reliably crossed when proper design techniques are used to accommodate likely timing violations. Some applications require that only control signals move between clock domains, and others must transport entire data paths. The simpler case of individual control signals is presented first and then used as a foundation for data paths.

Because it is impossible to avoid metastability when crossing clock domains, the phenomenon must be managed. A control signal ostensibly triggers activity in the logic that it drives, and this destination logic waits for the control signal to transition. Metastability does not prevent the signal from reaching its destination; it introduces uncertainty over exactly when the signal will stabilize in the destination clock domain. This uncertainty is dealt with by *synchronizing* the control signal using extra flops prior to treating the signal as a legal synchronous input. A two-stage synchronizing circuit is shown in Fig. 10.12. The synchronizing circuit takes advantage of the high probability that a metastable flop's output will achieve a stable digital state within a single clock cycle. If the first flop's output is stable, the second flop's output will transition perfectly and present a signal with

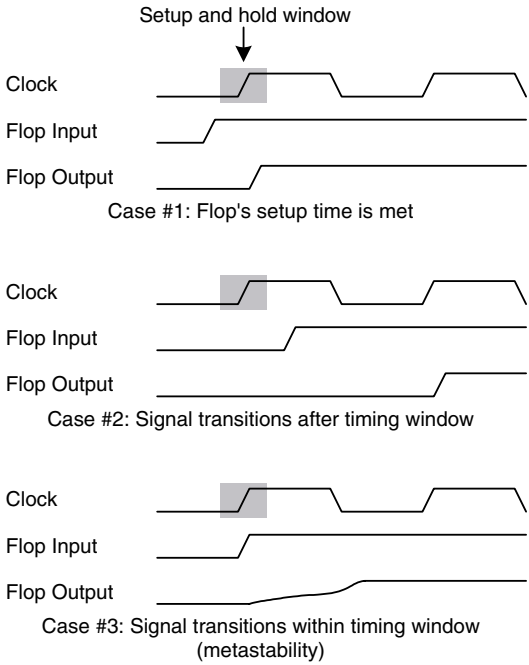

FIGURE 10.11 Flop operation with variable input timing.

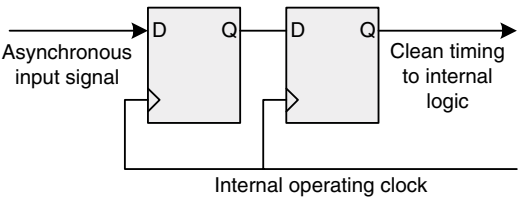

FIGURE 10.12 Synchronizing flip-flop scheme.

valid timing to the internal logic. If the first flop's output is not yet stable, chances are that it will be very close to a valid logic level, which generally should cause the second flop's output to transition cleanly. To deal with metastablility is to deal with probability. As the number of synchronizing flops is increased, the probability of a metastable state reaching the internal logic rapidly approaches zero. A general rule of thumb is that two flops reduce the probability of metastability in internal logic to practically zero.

An unavoidable downside of synchronizing flops is that they add latency to a transaction, because the internal logic will not detect an asynchronous signal's transition for two to three clock cycles after it actually transitions (when using a two-flop synchronizer). The best-case scenario of two cycles occurs when the input signal happens to transition early enough to meet the first flop's setup and hold timing. If these constraints are not met because the input signal transitions too late, the flop's