of the checksum. When it gets added into the receiver's checksum, the result should be 0. If it's not, some error in data transmission occurred.

An assertion can be written to watch the bus lines to see if the sender is correctly sending its data and checksum. To handle this design situation, sequences can be written to include local variables on which calculations can be performed as the sequence progresses. If the sequence ends up having several assertion threads active, each thread has its own copies of these local variables.

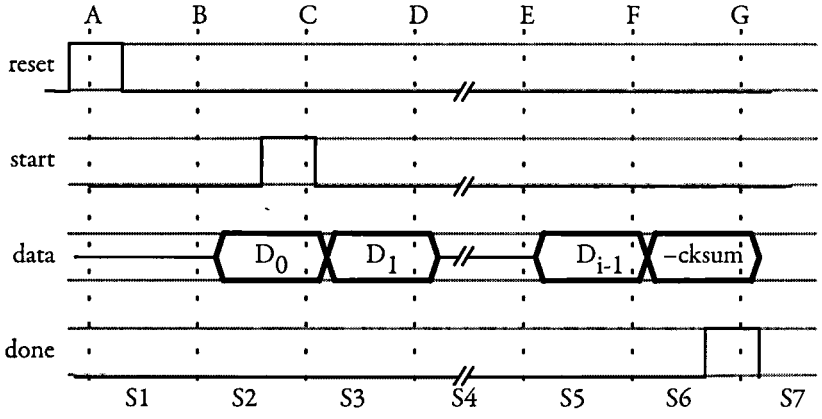

Figure 9.10 — Timing Diagram for Checksum Calculator

Consider the following sequence to calculate and check the checksum sent:

```
1  property ckCheckSumCorrect;
2    byte   cSum;
3    @(negedge ck) disable iff (reset)
4      (start, cSum = data) |=> (~done, cSum += data) [*0:19]
5                               ##1 (done && ((cSum + data) == 0));
6  endproperty
```

There are several new features in this property. First, just about all systems include a reset signal that puts a system into a specific state. In terms of using assertion in such situations, an assertion thread should not be started during reset, and assertion threads that are already in-flight should be cancelled. Line 3, with the inclusion of "disable iff (reset)," shows how to specify this. Essentially this is saying that when reset is asserted, all assertion threads with this specification are ended.

Secondly, a local variable is declared as shown on line 2.

The last new construct here couples an expression with an assignment statement separated by a comma, such as on line 4: (start, cSum = data). The first element (start) is the condition that is being checked to start the sequence. If it's TRUE, and thus the sequence is starting, then the current value of data is loaded from the bus (called data in the timing diagram) into local variable cSum. Once the sequence starts, there are between 0 and 19 consecutive states when ~done is TRUE. At each of these, data from the bus is added to cSum, calculating the assertion's checksum. done is asserted by the sender when it puts the negative of the checksum on the lines (clock tick G in Figure 9.10). When the asser-