tion runs as a result of clock G, the value being sent on the data bus is the negative of the checksum as calculated by the sender, and the value cSum is the checksum calculated by the assertion. The sequence (line 5) checks to see that when done is TRUE, that the sum of the sender's and assertion's checksums should be 0.

As a separate example, consider checking a pipelined multiply unit. The unit reads its two inputs at each active clock edge and a produces a result based solely on those two values three clocks later. The issue is how to write an assertion to check the result. The functional unit and its property would be:

```
1   module pipeMult (
2     input bit [9:0]   inA, inB,
3     input bit         ck, reset,
4     output bit [9:0] result);
5
6     bit [9:0] stage1out, stage2out;
7
8     always_ff @(posedge ck) begin
9       stage1out <= inA *  inB;
10      stage2out <= stage1out;
11      result <= stage2out;
12    end
13
14    property pipelinedFunction(inA, inB);
15      bit   [9:0]   A, B;
16      @(posedge ck) disable iff (reset)
17        (1, A = inA, B = inB) ##3 (result == A * B);
18    endproperty
19  ...
20  endmodule: pipeMult
```

The 3-stage pipelined multiplier's functionality is shown on lines 8-12 above. It's modeled as the multiply taking place during the first stage while the two inputs are still valid. (line 9). Lines 10-11 model this value being passed down the stages until it's finally on the output (result).

This property's sequence to check this uses local variables (A and B, on line 15) to remember the inputs when the assertion thread first starts. The sequence, specified on line 17, always starts because the 1 is a logic TRUE. The sequence then waits three clock ticks and compares the result output of module pipeMult with this assertion's internally calculated product. The sequence starts up a new assertion thread at every positive clock edge and thus there will be three threads in flight in steady state. Note that there will be a separate copy of local variables for each of the assertion threads.

## 9.5 Sampled Value Functions

Sampled value functions provide direct access to the values sampled during the preponed region of the simulation kernel.