

Property Checkers – A Technology Backgrounder.

The Properties Approach To Functional Verification.

Property Checking is a new approach to functional verification based upon formal methods. You describe things your design should always do, or things that your design should never do by writing a specification in fragments called Properties.

Having loaded your design and the required Properties into a Property Checker, such as Solidify from Averant, the tool will then consider every possible legal combination of inputs to your design to see whether the property holds true for all possible input vector sequences. The result of this is that Solidify will either confirm your design works and can never fail, or alternatively, provide an example of your design failing.

Solidify does not require either a vector set or a test bench to exercise the design – *and therefore there are no simulations to run*. The other advantage is that testbenches are generally written to exercise the behavior of the design in its expected mode of operation. In reality the input to a block often deviates from the designers initial expectations, and the design is then in untested territory.

Because Property Checkers consider all possible combinations of inputs to the design all possible modes of operation are tested, and hence corner case bugs the designer may not have thought to test for are exposed.

ADDER Example

So lets look at a practical example. We have two numbers 'a_in' and 'b_in' both of which are 32 bit buses. From this, we will then produce a single 33 bit output 'c_out'. Also, suppose that it is absolutely critical that this works correctly for all possible input combinations of a_in & b_in.

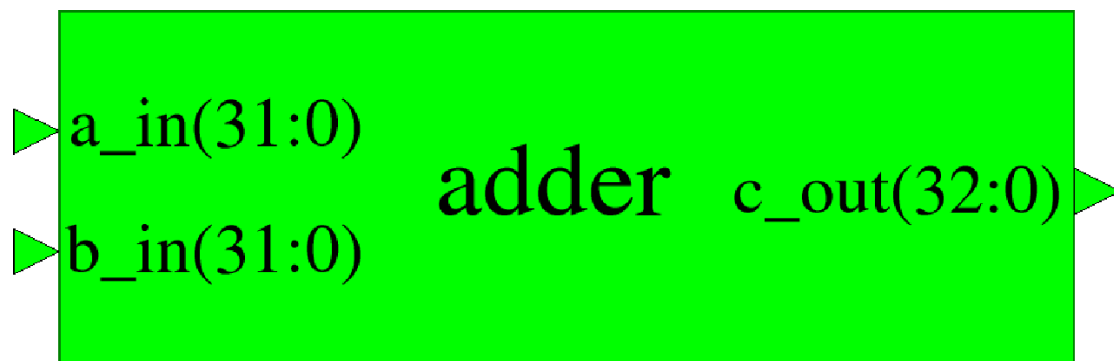


Figure 1: Adder

If we were to verify the adder using traditional simulation techniques we would sweep through the entire possible input range of 'a_in' and 'b_in' using counters and checking c_out after each addition. This requires approximately 70 lines of Test Bench.

What might the run time be for this process? Typically, a single computer will run 100,000 vectors per second. For a large task like this, we know one company that has created a server farm, where they link up to 5000 workstations together. So potentially, we could run at an impressive 500 million vectors per second.

The add instruction is going to require 2^{64} Vectors for a complete proof. Which gives us the following run time:

$2^{64}/5 \times 10^8$ seconds = 1,170 years!

Clearly, this is not going to be a practical solution even for our large server farm. How about using a Property Checking approach? The complete property to verify the add instruction is:

c_out == a_in+b_in;

This is certainly shorter than our 50 lines of Test Bench. What about run time?

- **We were able to exhaustively exercise the add instruction on a single 850MHZ Pentium III running Windows 2000 in approximately 2 seconds!**

Lets now look at an example with some stored internal memory states.

CPU Interface, Sequential Example.

This is a CPU interface from a Logic Analyser, and performs read and write applications on a CPU Bus. It also implements a time out releasing the bus if a device does not respond. We want to verify that the bus cannot lock up, and therefore that each request will get an 'ack' (acknowledge) or 'nack' (no-acknowledge)

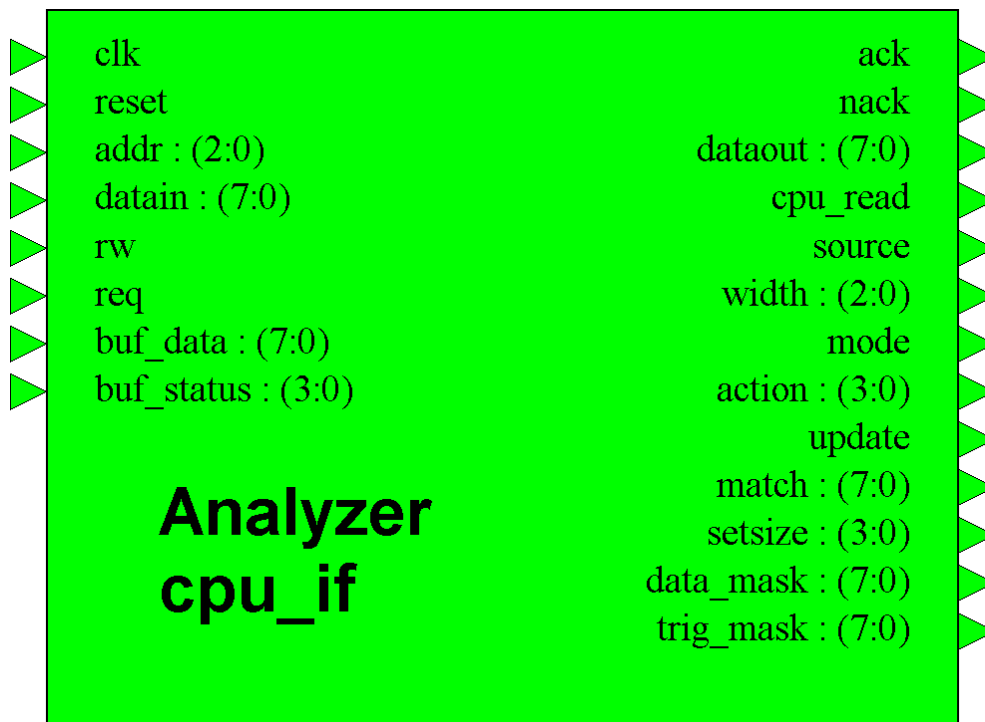


Figure 2: CPU Interface

Now the task here is not waiting for the vectors to complete, but determining what those vectors should be. We have many inputs here coupled with lots of internal state.

One of the beauties of using properties is that everything we want tested is not necessarily mentioned in the property itself. The more exhaustive the test, the more simple the property. The solution to this problem as a property is:

req && forever (!reset) => within5 (ack|nack) ;

This means that if we receive a req (request) and reset is held low, this implies that within 5 clock cycles, we will generate an ack (acknowledge) or nack (no-acknowledge). If there are any possible circumstances that can prevent this from happening, the tool will find it and report it to us. Any failures are reported in the form of a counter example. This is a short set of vectors, which if applied to the design will make it fail to behave as specified. You will notice that we deviate from a fully exhaustive search in only one way, in that we disallow the assertion of reset since we know that can prevent the design functioning. The above property takes less than a second of CPU time to verify on a Pentium III notebook.

Summary

Property Checking for Functional Verification offers an exciting new approach to verification, complimenting existing simulation methods. Its key advantage is that it can quickly and exhaustively verify blocks, which are difficult or

impossible using a simulation only approach. Property checkers also integrate well into existing verification environments, and do not disrupt existing design flows. Indeed, properties can be automatically converted to assertions and monitors for use during the simulation process.