

Combining Algebraic Constraints with Graph-based Intelligent Testbench Automation

by Mike Andrews, Verification Technologist, Mentor Graphics

ABSTRACT:

The Questa® inFact intelligent testbench automation tool is already proven to help verification teams dramatically accelerate the time it takes to reach their coverage goals. It does this by intelligently traversing a graph-based description of the test sequences and allowing the user to prioritize the input combinations required to meet the testbench coverage metrics while still delivering those sequences in a pseudo-random order to the device under test (DUT). The rule language, an extended Backus Naur Format (BNF) that is used to describe the graph structure, has recently been enhanced to add two powerful new features. Algebraic constraints can now be included to define relationships between the fields of the stimulus description (such as the fields of an OVM/UVM sequence item). Also, external testbench values can now be imported into the graph, allowing for the definition of relationships between Questa inFact-generated field values and externally selected values. The Questa inFact algorithms can now target cross combinations of fields that are under its control with fields that are outside of Questa inFact's control. This article describes these powerful new capabilities in more detail with some simple application examples.

INTRODUCTION:

The purpose of Questa inFact is to generate meaningful stimulus automatically and efficiently from a compact description of the scenario space of interest. Currently the most widespread automated stimulus generation methodology is constrained random, which generally comes hand in hand with coverage metrics defined in the testbench in order for the verification engineer(s) to track how well the random generation performed at hitting the important cases. As every constrained random user knows, there's a tradeoff to be made in how far to constrain the stimulus generation and how comprehensive the coverage metrics should be. Increasing the scope of the coverage metrics, especially when the constraint relationships are complex, tends to push the limits on what can be efficiently achieved by purely random generation. Limiting the scope of the verification metrics makes the verification process

more of a gamble, since beyond those metrics it can be very difficult to tell how effective the random generation has been.

Questa inFact has been helping verification teams to reach their desired coverage goals more predictably by defining the stimulus in a different way – specifically, using a rule based description that can be compiled into a graph. Powerful and efficient graph traversal algorithms can then more intelligently explore the stimulus space producing vectors that combine the benefits of random generation with the ability to prioritize specific coverage goals, including large cross combinations that can leave random generation stuck anywhere between 70-95% of the desired target.

Recently, the intelligence of the Questa inFact graph traversal algorithms has been significantly enhanced to allow algebraic constraints to be solved concurrently with the graph traversal process, thereby simplifying the stimulus definition process. This article describes how this powerful combination can be used to achieve more comprehensive verification goals in a more efficient and predictable way.

A DIFFERENT APPROACH TO CONSTRAINT SOLVING

When a constrained random solver generates a stimulus item its goal is to find a random valid solution to the user's stimulus description, and to continue this process repeatedly until the testbench halts and exits. The more complex the constraints are on what those valid solutions can be, the harder it is for a purely random engine to produce all the required combinations to thoroughly exercise the device under test. The primary issue is that, as complexity increases, so does the production of duplicate vectors. Many verification teams therefore can spend a lot of time analyzing the missed coverage and manually steering subsequent simulations to try to target the missing cover points – a process that can take weeks in some cases.

As mentioned before, the stimulus description for Questa inFact can now be a mix of rules (as compiled into graphs) and algebraic constraints. The power of Questa inFact

was originally in its ability to iterate through the graph description, randomly producing valid stimulus items while iterating through all combinations needed to meet the coverage goals. Where possible, the algorithms will attempt to meet more than one desired combination at a time from different cross coverage or individual stimulus field targets. As each defined coverage goal is achieved, Questa inFact will automatically revert to purely randomly generating the values for fields that are no longer involved in the remaining targets. So meeting a large total number of cover points, as defined by a number of different cover groups targeting different fields and field combinations, typically requires a number of vectors to be generated that is less than this total. This is in contrast to a pure random methodology, where the number of vectors needed to reach the desired coverage can be anything from 10x this total to effectively infinite.

Combining algebraic constraints with the traditional Questa inFact rule based description maintains this same ability to iterate efficiently through all the valid solutions. More specifically, Questa inFact is iterating through only the number of combinations that is needed to meet the goals of the various cover groups the verification engineer is targeting, since the total number of all valid solutions can be more than anyone has the time and resources to simulate.

THE NEED FOR CONTEXT DEPENDENT CONSTRAINTS

Questa inFact supports two different types of constraints, one of which is a global (or static) constraint that must always be satisfied, and the other is a context dependent (or dynamic) constraint that only needs to be satisfied for specific cases. When a dynamic constraint is declared, the graph structure is used to define the applicable context.

As an example, let's consider a stimulus generation application for controlling a robot. One field of the sequence item selects a general direction for the motion between the choices LEFT, RIGHT, FRONT or BACK. If the direction choice is FRONT, for example, then the resulting motion must be more in that direction than any other, but will

allow for some sideways component to the vector. Similarly, if the choice is either LEFT or RIGHT then the robot should move more sideways than forwards. There are another two fields to determine the new relative position, called xPos and yPos.. To ensure that the general direction choice is obeyed in the FRONT and BACK case, a constraint of 'Ypos > Xpos' is placed on these fields in that case. Where the direction is either LEFT or RIGHT there should therefore be the opposite constraint of 'Xpos > Ypos'. Hence the need for context dependent constraints to be specified in the stimulus description.

While the range of xPos and yPos are specified by 12-bit values, only discrete multiples of 64 and 128 can be used for these values respectively. Also, in the special case of direction 'FRONT' xPos must also be a multiple of 128.

These constraints on xPos and yPos can be implemented in SystemVerilog as shown in Figure 1.

```

constraint xPos_c {
    if (dir == BACK) {
        xPos % 64 == 0;
        xPos < yPos;
    } else if (dir == FRONT) {
        xPos % 128 == 0;
        xPos < yPos;
    } else if (dir == LEFT || dir == RIGHT) {
        xPos % 64 == 0;
    }
}

constraint yPos_c {
    yPos % 128 == 0;
    if (dir == LEFT || dir == RIGHT) {
        yPos < xPos;
    }
}

```

Figure 1. SystemVerilog constraints

Yet another field determines the speed, which is either SLOW or FAST. All directions except for BACK can have motion at both these speeds, while the BACK direction is limited to just SLOW.

DEFINING THE STIMULUS WITH RULES AND CONSTRAINTS

One option in the Questa inFact rule description would be to simply write a rule (otherwise known as a symbol) called 'sel_vals', that simply listed the order of selection of the fields of the item. Then the SystemVerilog constraints shown above could be added to the Questa inFact rule file, with one minor syntactic difference of a ';' terminator for the constraint block. Figure 2 shows an example segment of the Questa inFact rules with the sel_val rule and the constraint on xPos.

```
symbol sel_vals;
sel_vals = dir speed xPos yPos;

constraint xPos_c {
  if (dir == BACK) {
    xPos % 64 == 0;
    xPos < yPos;
  } else if (dir == FRONT) {
    xPos % 128 == 0;
    xPos < yPos;
  } else if (dir == LEFT || dir == RIGHT) {
    xPos % 64 == 0;
  }
};
```

Figure 2. Example Questa inFact rule segment

In the Questa inFact rule description we can also define relationships using the graph structure, where a branch in the graph defines the limitations for the BACK, FRONT and LEFT & RIGHT directions. I can also apply dynamic or context-dependent constraints on those branches such that those constraints will be obeyed only in the intended context. Figure 3 shows the amended rule for 'sel_vals' and the combination of dynamic and static constraints that would be needed.

Figure 3. Example branched graph rule

```
constraint xPos_back dynamic {xPos % 64 == 0; xPos < yPos;};
constraint xPos_front dynamic {xPos % 128 == 0; xPos < yPos;};
constraint xPos_side dynamic {xPos % 64 == 0; yPos <= xPos;};
constraint yPos_c {yPos % 128 == 0;};

symbol sel_vals;
sel_vals = (dir[BACK] speed[SLOW] xPos_back) |
           (dir[FRONT] speed xPos_front) |
           (dir[LEFT, RIGHT] speed xPos_side)
           xPos yPos;
```

In this example, the '|' choice operator is used to define the optional branches. The first branch limits the direction choice to BACK, limits the speed choice to SLOW, and then applies the dynamic constraint xPos_back. The next two branches similarly group the related field values and the algebraic constraints that must be applied to fields further down the graph path. The graph branches recombine before the selection of xPos and yPos. Figure 4 shows the graphical view of this rule graph.

This graphical view makes it much easier to visualize the stimulus description and has often helped the user to see errors that would have been much harder to discern using the text-only constraint description. The dynamic constraints appear in the graph as upside down trapeziums.

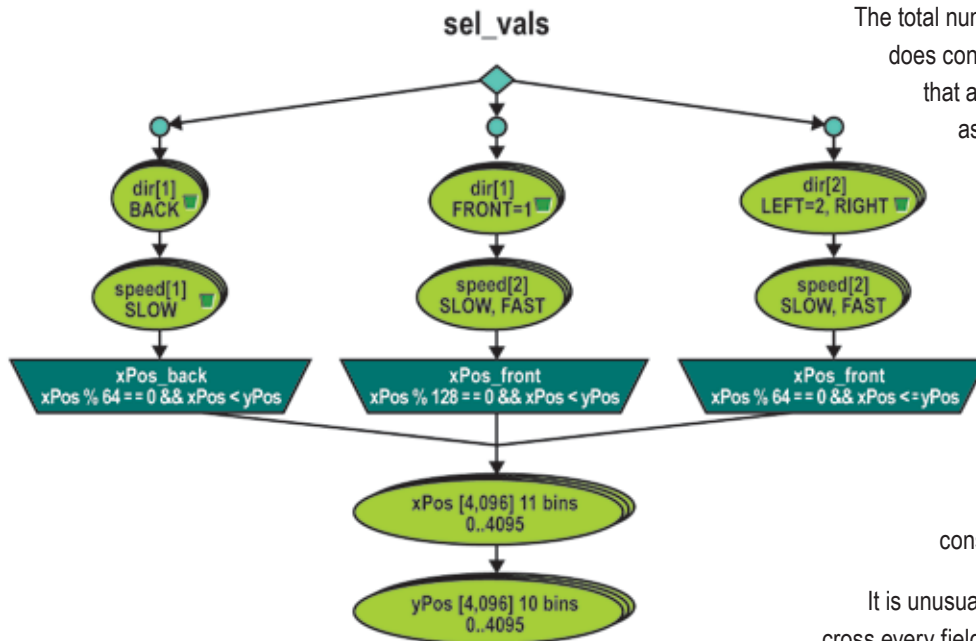
Note that in the graph view, there is an annotation on the xPos and yPos nodes stating how many bins are defined for these two fields. Given the large range of possible values, and the limits on which values are legal as defined by the constraints, it is obviously not practical to exercise all 4,096 values. A number of interesting bins are therefore defined for these two fields, and this information is used by the Questa inFact algorithms as they target the user's coverage goals. Figure 5 shows the definition of the bins for these fields.

The bins declaration follows the declaration of the domain of the field. The last bin uses a "*" to create an additional single bin which contains all remaining non-explicitly binned values. Bins can also be defined on a per coverage goal basis, so that different cross coverage goals that include the same variable can be binned differently.

COVERAGE GOALS AND THE STIMULUS SPACE

The sel_vals rule is one rule within a hierarchy of rules that define the full graph. A higher level rule call RobotCtrl includes an initialization step, a construct called a repeat,

some nodes that synchronize the graph execution with the testbench, and another field of the stimulus item called 'mode.' Figure 5 shows the top-level rule.



The total number of 3080 combinations does consider any bins defined that are global, i.e. are not only associated with a particular coverage target. It therefore reflects, in most cases, the number of cover points that would be reported for a cross cover group that included all the fields in the graph, prior to the definition of the exclusions due to constraints.

It is unusual of course to attempt to cross every field in the stimulus item, since in most cases that would be an impractical number of simulation vectors, even with sensible bins defined.

Figure 4. Example branched graph

```

meta_action xPos[unsigned 11:0];
bins xPos [64] [128] [192] [256] [320] [384] [448] [512] [1024] [2048] [*];
meta_action yPos[unsigned 11:0];
bins yPos [128] [256] [384] [512] [640] [768] [896] [1024] [2048] [*];

```

Figure 5. Defining bins in the rule description

The coverage metrics tend to combine cross coverage targets with individual field targets.

Figure 6 shows the top-level graph. The numbers annotated onto the graph show the size of the stimulus space defined by the elements of the graph, without considering the effect of the constraints.

```

RobotCtrl = init repeat {
  pre_fill
  mode sel_vals
  post_fill
};

```

Figure 6. Top-level RobotCtrl rule

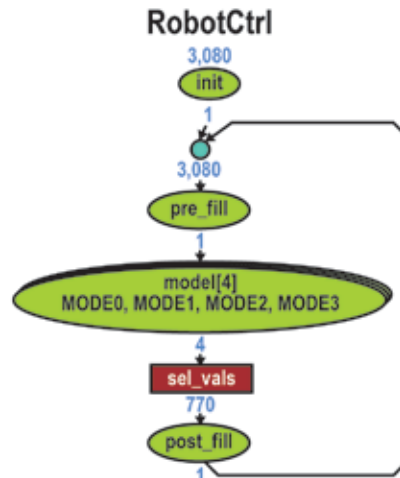


Figure 7. Top-level RobotCtrl graph with size annotation

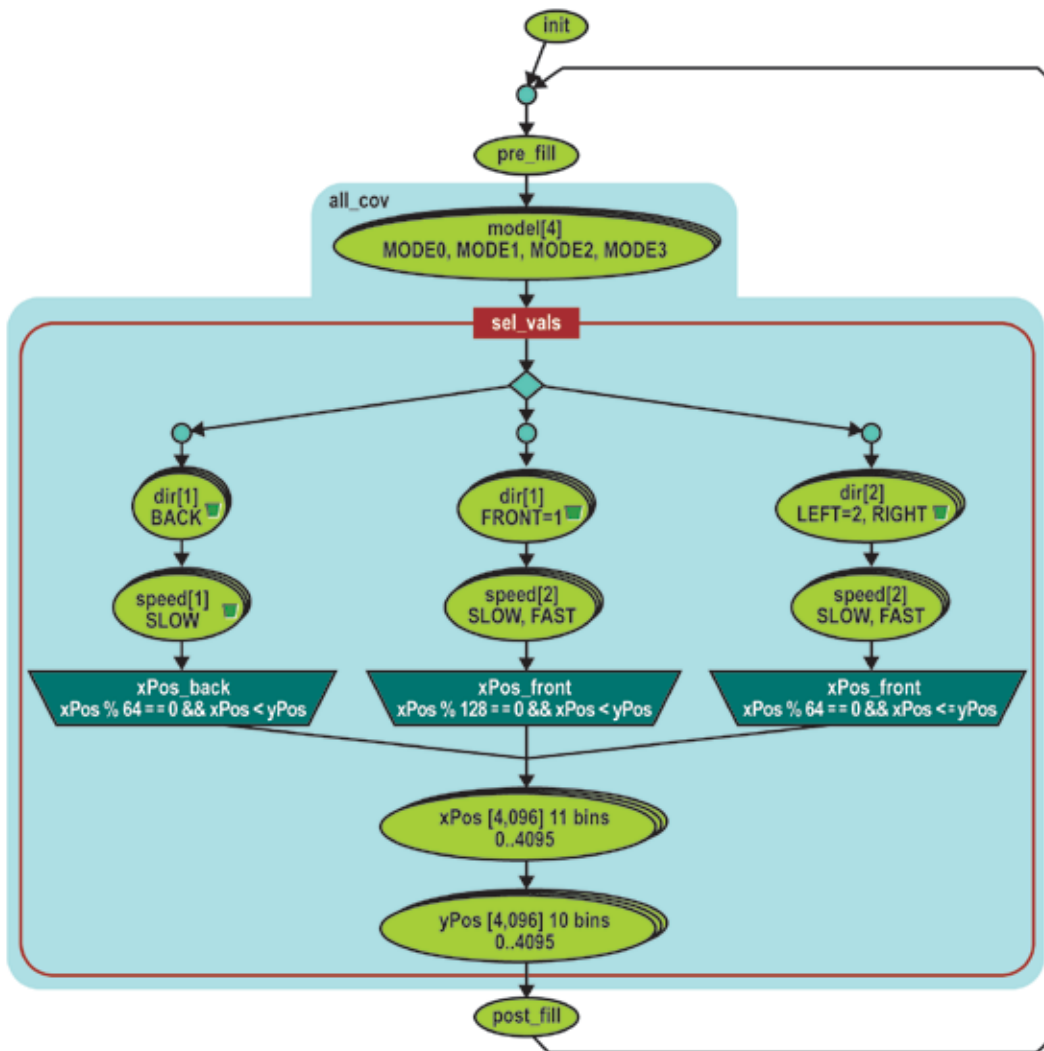


Figure 8. Definition of the coverage strategy

In Questa inFact this is achieved by overlaying a user-defined coverage strategy onto the graph. The coverage strategy mirrors the coverage goals by specifying which fields require cross coverage and which are targeted for single-value coverage.

With our robot control verification project, we will assume that the cross of all fields is the goal, so our coverage strategy contains that one goal. A graphical editor can be used to define the region of interest, which in the fully expanded graph goes from the 'mode' node down to the yPos node at the bottom. That coverage goal is called a path coverage goal in Questa inFact terminology and is given a name for reporting purposes. Figure 8 shows this goal reflected in the graphical editor.

Any of the fields can be excluded by declaring it a 'don't care' for this particular goal. An important benefit of Questa inFact's ability to comprehend the graph structure and the constraints simultaneously is that it can report the total number of valid combinations.

Our example has an additional constraint on the mode field that limits it to just the last three options, so this will also be considered in producing the total valid combinations. Figure 9 shows the result of this calculation that can be performed independently of simulation at the same time as the coverage goals are defined. This is expressed as a 'Path Count' in the Questa inFact tool.

Path Coverage	Count	Status
all_cov	1053	COMPLETE

Figure 9. Path Count Result for the Cross Coverage Goal

While we have 3520 possible combinations of these field values as binned, the graph structure reduces this number to 3080, since it reflects the relationship between dir and speed, and then the constraints further reduce the legal set to just 1053. It can be difficult to get to this final number in the presence of many complex relational constraints, and to have the cover group accurately reflect this, but Questa inFact can calculate this statically very efficiently. By adding just 3 bins to mode to reflect the constraint on that field, the cover group would contain 2640 cover points in the cross. Without specifying all the remaining illegal combinations in the cover group definition the total possible coverage that we can get is $1053/2640 = 39.886\%$.

COMPARING THE RESULTS

As expected, when the testbench is run with the Questa inFact graph, all 1053 legal combinations are created in exactly that number of generated items. The coverage that is reported for the cross is 39.8%. If we define this as the coverage target for a constrained random run a comparison can be made.

As we would expect from a constrained random generation, the progress towards the coverage goals tails off as we get closer to the goal, with significant redundancy in the vectors produced. After 120,000 items, the constrained random generation has hit 1050 of the 1053 legal combinations. It takes another 20,000 generated items to raise this to 1051. A little over another 60,000 items puts us at 1052, and the final missing combination is achieved after a total of 271,700 items. The effective improvement in the number of vectors it takes to achieve the coverage goal using Questa inFact is therefore 258x versus the constrained random equivalent. If each vector took a minute

to simulate (not an unreasonable estimate) that would mean 17.5 hours of simulation with Questa inFact, and 4,528 hours (or almost 27 weeks) with constrained random. Of course, in a real verification project, other techniques would be used to get to coverage faster, such as using

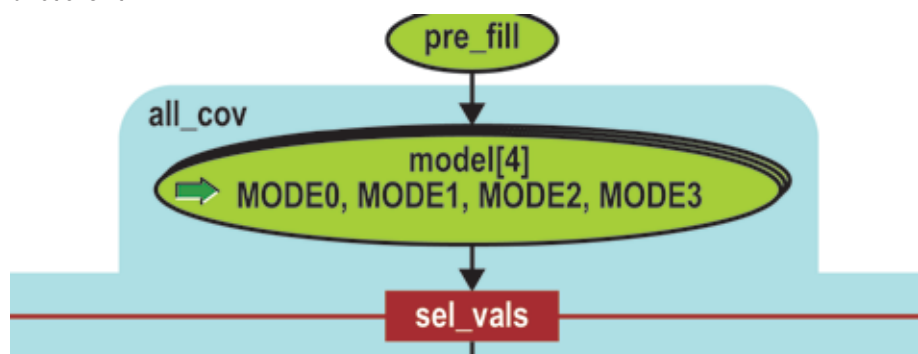
multiple parallel simulations, adding directed tests, or writing further constraints to steer the random generation closer to the missing coverage. A combination of these techniques would probably be used (rather than waiting for three months for the last three valid vectors).

WHAT IF I CAN'T CONTROL ALL FIELDS FROM A GRAPH?

A little more complex case is when one or more fields can't be selected by the graph, but is a product of some testbench state. In this case, we would use the new import feature in inFact to bring awareness of this field value to the inFact algorithms. With a slight modification (two lines of code affected) to the example Robot control testbench, we can move the mode field out of inFact's control and instead randomize it before we call the inFact graph to select the rest of the fields. When a variable is declared as an import in the graph, then its value is read from the testbench when the algorithms traverse through that node in the graph. Figure 10 shows the 'mode' field expressed as an import – denoted by the arrow on the left side of the ellipse.

Note that the domain of the variable is still expressed in the rules, allowing the inFact algorithms to target all the desired values and cross combinations.

Figure 10. Example of an imported variable



In this case, during simulation, a different aspect of the intelligence in Questa inFact's algorithms is exercised, that is the ability to react to the testbench state when producing a new stimulus item. If the value of mode is imported into the graph, and the coverage strategy is still to produce the cross of all fields, including mode, then Questa inFact can still far outperform constrained random, taking only 1340 items to produce all the 1053 valid items.

DO I LOSE ANYTHING WITH THIS APPROACH?

Over the years I have been working with verification teams and verification technologists I have often heard this question. After all, random generation produces more vectors than just the ones needed to meet the coverage metrics defined by the user. The generally accepted opinion is that more vectors simulated is equivalent to more verification. The promise of achieving the targeted coverage in anything from a 10th to a 1000th of the previously required vectors can therefore be a concern. The problem is there is no way to discern if those extra, mostly redundant vectors, that were generated before were actually exercising the DUT in a different or useful way. There are two ways to respond to this concern.

The first way requires virtually no extra effort on behalf of the verification team, and is to use Questa inFact to prioritize the needed vectors for coverage, and then continue to run in a purely random mode for as long as time and resources allow. A variant of this approach would be to have Questa inFact target the desired coverage more than once, taking advantage of the fact that it will produce different vectors each time, in a totally different order. This means that the combinations that the verification engineers thought were of interest get exercised in different contexts.

A second approach assumes that the original verification metrics were not as comprehensive as they could be. Expanding the scope of these metrics does of course require some effort, but this effort clearly pays off in the confidence level that can be achieved, and as proven in some cases, in the increased number of bugs that are found earlier in the verification process.

SUMMARY

As the description of the stimulus to the DUT becomes more complex, with complex constraint relationships needing to be defined, reliance on randomly generated stimulus to achieve comprehensive coverage metrics becomes a less predictable and more labor intensive process. With the addition of algebraic constraints to the Questa inFact rule based stimulus description a more intelligent approach can be taken, that can be tremendously effective in saving time and resources and is now much easier to implement.