Debugging Functional Coverage Models Get The Most Out of Your Cover Crosses

Mennatallah Amer, Amr Hany
Mentor, A Siemens Business
78 El Nozha St., Heliopolis, Cairo. +202 24141306
mennatallah_amer@mentor.com, amr_hany@mentor.com

***Abstract*- Functional coverage models have grown in complexity to account for the increasing demands of designs today. Traditional and even advanced analysis techniques have yet to evolve to provide the verification engineer with actionable insights on how to debug their functional coverage model. In this paper, we generalize advanced hole analysis techniques to be able to get the most out of cover groups containing multiple cover crosses. Applying hole analysis on each cover cross independently can lead to misleading results and is sometimes prohibitive due to the sheer number of crosses. Additionally, we introduce a metric, hole effect, that is proportional to the coverage gains that would result upon resolving the highlighted hole. We evaluate our approach on a real processor's data processing unit to validate its applicability and usefulness for debugging complex functional coverage models.**

I. Introduction

Functional coverage paired with constrained random testing is a critical, invaluable methodology in a verification engineer's tool-set. The verification engineer has to capture the correct, important design aspects within the coverage model. Since this is an iterative process due to the increasing complexity of the design under verification and the often ambiguous, evolving functional specifications, the coverage model itself can end up containing bugs. That requires the engineer to go through a time-consuming debugging process. Conventionally, coverage model debugging is performed by the manual inspection of the coverage reports generated by EDA coverage tools. The end goal is to extract actionable insights that can aid in adjusting the coverage model, or perhaps adjusting the constraints in the random test generator to reach coverage holes. Those conclusions are incredibly hard to extract from the influx of information available in the traditional coverage reports. Applying advanced analysis techniques to get the most out the available information would save verification engineers precious, critical time and aid in reaching faster coverage closure.

In this paper, we adopt and generalize the hole analysis technique proposed in [1], reaping the benefits of concise representation of the vast information present in complex functional coverage models. Our focus is on covergroups with a number of cover crosses. A Covergroup is composed of a set of coverpoints and an optional set of cover crosses modeling the combination of the various coverpoints. Verification engineers use cross coverage to define expected valid combination of signals or variables in the design under verification. To help with the analysis of complex designs, verification engineers leverage SystemVerilog syntax to perform semantic grouping of valid expected combinations. That in terms helps with the analysis as well as trimming down the number of bins in the coverage model. On the other hand, that can result in a large number of possibly related cover crosses which poses an additional burden of having to analyze them all in order to debug/understand any issue in the coverage model. It is common to find multiple crosses sharing common coverpoints as shown in the examples in this paper. Most probably the coverage model captures the combinations of a group of common coverpoints with one or two unique coverpoints. In other cases, multiple crosses share exactly the same coverpoints where the differentiating metric for each cross can be one of the following:

1) *Some combinations are marked illegal/ignored in one cross*
2) *The coverpoint itself is used to bank a part of a bus/memory using enable signal (ex. read/write signal)*
3) *The bins of each cross are grouped in a different way*

In these cases, analyzing the holes of each cross individually may be tricky as the ignored or illegal bins can mislead the results since similar combinations are excluded from coverage calculations. Multi-cross hole analysis on such overlapping crosses gives more precise results.

Applying hole analysis techniques on multiple crosses sharing common coverpoints can identify larger holes and help in guiding further analysis of selected crosses. The combined analysis leads to more accurate results on the selected coverpoints and is much more efficient than analyzing each cross independently. Additionally, we define a new metric, hole effect, that gives the user insight into the optimistic ripple effect on the coverage if the identified hole is covered.

Debugging cover crosses is a tedious task due to the number of possible bins. The papers described in [1, 2, 3], tackle this problem. The work in [2] described coverage views and how defining such views allows users to focus on certain aspects of coverage data. In [3], the authors describe the technique of automatic hole detection. Different algorithms like partitioning, projection, and aggregation are described. This work is concluded in [1] by defining automatic, interactive coverage analysis, quasi-hole analysis for lightly covered areas, and coverage query which is a mix of automatic and interactive analysis. These methodologies enable the extraction of useful salient holes that can be the seed for other coverage driven tools in the verification tool chain. The approaches described in the literature so far are applied on a single cover cross at a time and do not consider searching for larger holes by exploring multiple crosses sharing common properties. In case of many holes in the coverage model, the verification engineers rely on their semantic knowledge of the coverage model, system constraints and illegal combinations in order to begin their debugging effort.

There are other techniques in the literature aiming at discovering holes in the coverage model generally and not just in the functional coverage model. The authors of [6], use Inductive language programming for test directive generation, clustering is used to group holes according to the coverage model. A code coverage variant for hole analysis is introduced in [7]. It leverages Substring analysis of source code elements mapped to certain functionalities.

There are several tools that can leverage the output of advanced hole analysis to generate new tests or formally prove reachability. The technique used by the authors in [4] extracts the holes in the cross coverage model together with a set of constraints extracted by data-mining techniques on the simulation trace in order to feed back the data into the intelligent test generator. Each of the extracted holes is passed to the formal engine in order to verify its reachability. The author in [5] built a feedback link between the coverage data and random stimuli in order to avoid redundant stimuli and accelerate coverage closure. The author implemented an API to extract holes but didn't add any ranking to the obtained holes. The work in this paper adds sorting to coverage holes according to the coverage increase resulting from fixing the holes. By starting with resolving the larger holes, the coverage closure is accelerated since the design's bigger problems are targeted first.

The rest of paper is organized as follows, a review of single cross hole analysis is presented in section II. The technique is outlined on an illustrative example that will be used in the following section as well. The novel multi-cross hole analysis technique is described in section III, together with the associated metric hole effect that can pinpoint covergroup specific holes. Section IV presents the application of our proposed technique on a real processor covergroup, highlighting the various insights that the user can extract. Section V contains the conclusion, whilst section VI describes the possible extensions

II. HOLE ANALYSIS

In the reference [2], the authors identified various techniques to view cover cross data. One of the most useful operations they identified is projection, where you can decrease the number of bins you are currently analyzing by simply projecting the results onto certain coverpoints. That means that a combination of the selected coverpoints map to more than one bin combination in the actual cross. Those number of bins could be either covered or uncovered. Hence a metric is defined, the density, as the ratio of the covered to the total bins available in the selected combination. This enables the users to only focus on a subset of the results instead of going through the complete cross space.

To better understand the benefits of Hole Analysis technique let us consider the following simple example of a data processing unit coverage model. The model consists of seven coverpoints representing the different attributes of data transaction as illustrated in Table 1. The number of bins can be deduced from the values of each coverpoint. The number of bins starts to be a problem when the Cartesian product of coverpoints is evaluated in the coverage crosses. The cross coverage of this coverage model is represented by the following coverage crosses in Table 2. Note that some of the cross bins contain illegal combinations resulting in a smaller total bin size.

TABLE 1
DATA TRANSACTION COVERAGE MODEL

Attribute	Values
Burst	single, incr, wrap4, incr4, wrap8, incr8, wrap16, incr16
Access	unlocked, locked
RW	Read, Write
Size	4, 8, 16, 32, 64
Prot	opcode, data, user, private
Resp	OK, Error
Secure	Yes, No

TABLE 2
COVERAGE CROSSES

Cross	Coverpoints	Hits	Bins	Coverage
cross_1	cvp_burst, cvp_secure, cvp_rw, cvp_access	16	32	50.00%
cross_2	cvp_burst, cvp_rw, cvp_size	60	60	100.00%
cross_3	cvp_burst, cvp_rw, cvp_access	24	32	75.00%
cross_4	cvp_burst, cvp_rw, cvp_prot	48	64	75.00%
cross_5	cvp_rw, cvp_prot, cvp_resp	14	16	87.50%
cross_6	cvp_burst, cvp_rw, cvp_access, cvp_resp	48	64	75.00%
cross_7	cvp_burst, cvp_prot, cvp_resp	56	64	87.50%

Simulating the model shows a covergroup coverage of 89%. All coverpoints that represent the model's attributes score 100%, so the problem lies within cover crosses as some combinations are not hit. Analyzing all combinations of each cross is a tedious task even for this simple example. Typical coverage reports lists all combinations annotated with coverage score. For this simple example, the total number of combinations is 357. The Hole Analysis technique [1] can help manage and debug any single cross in the coverage model. Hole analysis enables the user to gain more insight by outlining salient holes. In the following subsections, we will outline the results of the single hole analysis techniques applied on cross_3.

A. Single Cross Projection

By Projecting the 32 combinations of cross_3 onto a subset of the coverpoints namely cvp_rw and cvp_access, it ends up with only 4 combinations that are used for illustration in Table 3.

With every projection in this case, 8 bins are actually mapped to a single combination and hence [3] defines a new metric, density, that would capture the coverage of the bins having that combination. By creating that view, the user can automatically infer that the problem is when cvp_rw is "Write" and hence we can cut down the search space by half from the first glance.

B. Single Cross Hole Detection

Automatic hole detection can further be used in order to detect larger contiguous holes promptly. Applying projected hole analysis followed by an aggregation yields all the 12 missed bins in one descriptive line:

<cvp_burst, cvp_rw, cvp_access>=<{incr, incr4, incr8, incr16}, Write, *>.

The projected hole analysis inferred that for all combinations of "incr*" and "Write" all the bins are missed. Then doing a recursive aggregation for holes that have a Manhattan distance less than one yields a single large hole representing all the missed combinations.

To analyze the rest of the covergroup, single hole analysis has to be applied on each individual cross. In the above example, if the user selected cross_1, which has the least coverage as the seed for the first analysis steps, the resulting hole would be:

<cvp_burst, cvp_secure, cvp_rw, cvp_access>=<*,No,*,*>

The user can start debugging the issue in cvp_secure first. It will be shown later that this should not be the primary focus of the verification engineer for two reasons:

1) *Solving that hole in that cross would not result in the highest covergroup coverage gain.*
2) *cvp_secure is only present in cross_1 and hence guiding the focus of the user to it is not ideal.*

III. MULTI-CROSS HOLE ANALYSIS

Single hole analysis is a powerful technique, but even in the simple example it can be inefficient, tedious and time consuming. In the simple example, Fig. 1 shows the number of crosses that each coverpoint is contributing to. We can see that five of the coverpoints are contributing to at least three crosses and two coverpoints are contributing to six out of the seven crosses. A visualization is shown in Fig. 2 that presents the relation between the coverpoints and crosses. In order to capture the overlap between the various crosses in a better way, each of the cross pairs can be examined to see the size of the overlap. In this example, there are 7 crosses which form 21 possible combinations. Plotting the degree of overlap with the number of cover cross pairs is shown in Fig. 3.

TABLE 3
SINGLE-CROSS HOLE ANALYSIS

cvp_rw	cvp_access	hits	bins	Density
Write	locked	4	8	50%
Write	unlocked	4	8	50%
Read	locked	8	8	100%
Read	unlocked	8	8	100%

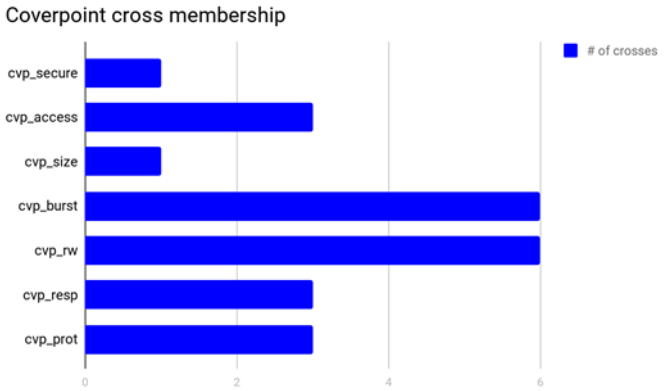

Figure 1. Coverpoints contributing to number of crosses

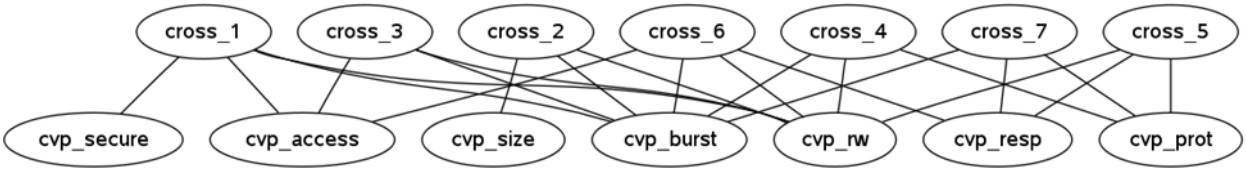

Figure 2. Relation between coverpoints and crosses

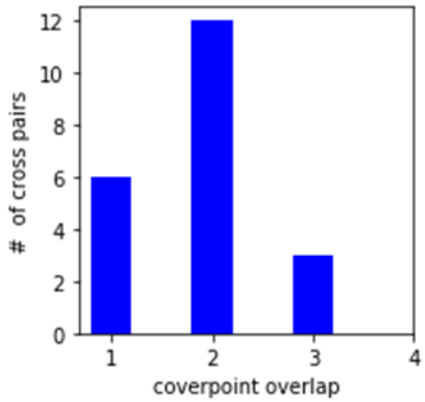

Figure 3. Degree of overlap of cross pairs

It is highlighted that there are three cross pairs that have three coverpoints in common (*cross_1*, *cross_3*), (*cross_1*, *cross_6*) and (*cross_3*, *cross_6*). Hence taking a global view and generalizing the approach for multiple crosses can save invaluable debugging time.

A. Multi-Cross Projection

Coverpoints appearing in more than one cross are intuitively important coverpoints and hence projecting the coverage of the crosses on those coverpoints give a top level summary of the likely common problems in the crosses space. From Fig. 2, we can see that cvp_burst and cvp_rw occur in six out of the seven crosses. In fact the two coverpoints occur together in five out of the seven crosses: *cross_1*, *cross_2*, *cross_3*, *cross_4*, and *cross_6*. Collectively projecting the crosses on both coverpoints would yield the snippet in Table 4.

TABLE 4
MULTI-CROSS HOLE ANALYSIS

cvp_rw	cvp_burst	hits	bins	density
Write	incr	0	10	0.00%
Read	incr	15	15	100.00%
Write	single	17	19	89.47%
Read	single	17	19	89.47%
Write	incr4	0	10	0.00%

By simply sorting by the density you can see that the problem is in <Write, incr*> for all five crosses.

B. Multi-Cross Hole Detection

Generalizing the automatic hole detection techniques on the combined crosses is also possible. However simply using the number of missed bins as an indicator of the hole size is not enough. The effect of the hole size on the cross coverage depends on the total bins of that cross. Consider a hole of size 10 in two crosses with 20 and 100 total bins respectively. The effect of the hole on the cross coverage of the first small cross is 50%, whilst that of the larger cross is only 10%. Additionally SystemVerilog covergroup coverage is a weighted average of the constituents of the covergroup and hence various cover crosses can have higher impact on the covergroup coverage.

For each hole, the expected coverage increase in a cross can be computed as the number of missed bins in that hole over the total number of bins in the cross, as shown in equation (1). The hole effect of the analyzed crosses is then computed as the weighted average of the coverage increase in each individual cross in the set as depicted in equation (2).

$$covInc_c = \frac{miss_c}{bin_c} \qquad (1)$$

$$HoleEffect_C = \frac{\sum_{c \epsilon C} w_c \cdot covInc_c}{\sum_{c \epsilon C} w_c} \qquad (2)$$

Here C is used to denote the set of crosses being analyzed, c is used to denote the individual crosses. $miss_c$,$bins_c$, w_c are used to denote the number of missed bins in the hole for that cross, the total number bins of the cross and the cross weight respectively.

That score can be computed for each of the extracted holes, providing the user with the ability to rank the holes based on largest effect on the cover group coverage. The assumption is that if the verification engineer is able to fully target the identified hole, the average coverage of the selected crosses will increase by the hole effect. This is a really optimistic assumption since a large hole is usually broken up into smaller holes upon increasing the test stimuli. Note that the hole effect is meant to compare the results of the holes in the concurrently analyzed crosses. To generalize that to all the covergroup reports, the denominator of the equation (2) should be equal to the total weight of the covergroup constituents.

Automatic Hole Analysis can either be applied on the coverpoints that are present in all or some of the analyzed crosses. Applying hole analysis on the coverpoints that are present in all of the crosses is quiet optimistic and you might not be able to view all if any of the holes. Allowing coverpoints that are contributing in some of the crosses gives more granularity and can capture more holes. The only caveat is that due to the exponential nature of the hole analysis algorithm in the number of coverpoints, there is an upper limit to the number of coverpoints that can be used in the combined analysis. This can be achieved by performing projection on the crosses on the target coverpoints. We can apply both upward and/or downward projection as we will describe in the next paragraphs.

In the simple example, multi-cross hole detection can be applied on all coverpoints and crosses resulting in the results shown in Table 5. Since we have only 7 coverpoints we can afford to analyze all the cover cross bins relative to it. That would mean that we need to perform upward projection on all the cross bins to the seven coverpoints. This operation is performed by setting the missing coverpoints to value "*". For example a bin in cross_3 (<cvp_rw, cvp_burst, cvp_access>) having the value of <Write, incr, locked> would be mapped to the entry of <cvp_burst, cvp_rw, cvp_secure, cvp_prot, cvp_resp, cvp_access, cvp_size>=<incr, Write,*,*,*,locked,*>. That simple operation enables us to detect all the holes in the seven crosses with one hole analysis run. Hence, by analyzing multiple crosses together we can avoid missing holes that might be masked by illegal combination of a single cross and accelerate the coverage analysis process.

More generally applying multi-cross hole detection on a subset of the coverpoints can also shed some light into the problematic combinations in a computationally feasible time. To apply this technique on all the crosses in the simple example using the five coverpoints <cvp_access, cvp_burst, cvp_rw, cvp_secure, cvp_resp>, we need to both project up and down each of the crosses depending on its nature. *cross_1*, which does not have cvp_resp, would have all its leaf level combinations equivalent to having all values of cvp_resp "*". In contrast cross_2, which has cvp_size that

is not present in selected coverpoints, then the projection of the remaining contributing coverpoints cvp_burst and cvp_rw would be present as leaf level combinations of that cross with the remaining coverpoints set to "*". Applying combined hole analysis for that case would result in the results shown in Table 6. In Table 6, all holes that are present in all crosses are projected on only five coverpoints. This projection captures the top two holes in Table 5 and missing the last one due to the coverpoint selection.

In more complex examples, the algorithm may flag different orthogonal holes. This is where the hole effect would differ than the number of missed bins. The following section contains the results of a real processor.

Another degree of flexibility and abstraction can be added through detecting quasi, non-pure, holes. Non-pure holes point to areas that are lightly covered whilst pure holes pure to areas which have zero coverage. Working on a subset of coverpoints more likely detects quasi holes as it is hard to capture pure holes due to the loss of information associated with projecting onto lower dimensions.

IV. USAGE EXAMPLES

In order to show the real power of the multi-hole analysis technique, a covergroup from a real ALU design is used. The selected covergroup contained a total of 202 crosses, 138 coverpoints and a total of 45,991 bins of which only 30.78% are hit, leaving almost 70% of the bins missed. Like the simple example, all the coverpoints have a 100% coverage and hence the problem is in the cover crosses. To characterize the degree of overlap between the crosses, the number of cross pairs is displayed in Fig. 4. It is shown that there are 229 cross pairs sharing six coverpoints which shows a large degree of overlap.

The verification engineer can select the overlapping crosses to start analysis. There are 22 overlapping crosses that share up to six coverpoints. The selected crosses top holes are shown in Table 7.

Most of the crosses have low coverage which indicates a common problem across these related crosses, the average coverage of 22 crosses is 13.4%. Those crosses contribute to approximately 10% of the covergroup coverage. So instead of analyzing a single cross at a time, this approach can identify the biggest common hole in a single step incurring the cost of a single hole analysis run and reducing the amount of data that the verification engineer should amalgamate manually. As evident in Table 7, each cross has varying number of total bins, supporting the notion of adopting the hole effect metric rather than the number of missed bins in the hole.

Analyzing the crosses on five contributing coverpoints, four of which are present in all 22 crosses and *hstr* is present in 21 of the crosses. The selection of coverpoints on which the analysis is performed could be based on its frequency among the crosses set, or guided through the semantic meaning of the coverpoints. The results of the top five holes extracted by the multi-hole analysis is presented in Table 8.

TABLE 5
MULTI-CROSS HOLE DETECTION-ALL COVERPOINTS (CVP_ACCESS, CVP_SIZE ARE ALL * AND REMOVED FOR COMPACTION)

cvp_burst	cvp_rw	cvp_secure	cvp_prot	cvp_resp	miss	Hole Effect
incr*	Write	*	*	*	40	10.71%
*	*	No	*	*	16	7.14%
*	*	*	private	error	10	3.57%

TABLE 6
MULTI-CROSS HOLE ANALYSIS-SELECTED COVERPOINTS

cvp_burst	cvp_rw	cvp_secure	cvp_access	cvp_resp	Missed	Hole Effect
incr*	Write	No	*	*	40	10.71%
*	*	No	*	*	16	7.14%

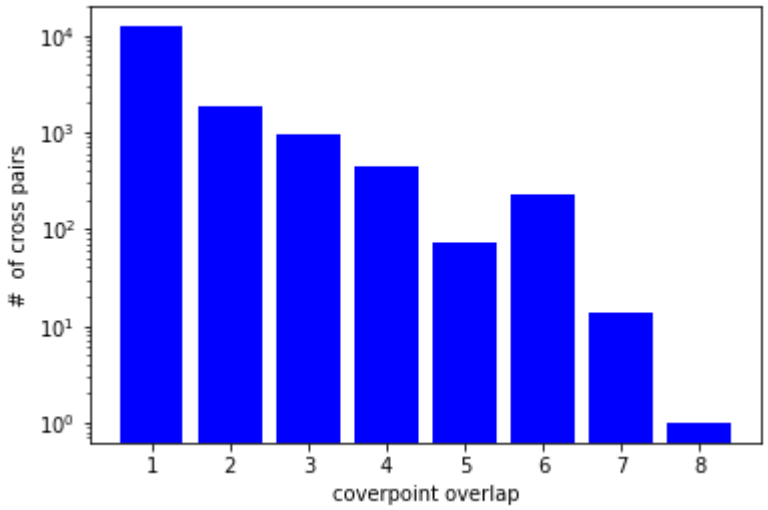

Figure 4. Degree of overlap of cross pairs in ALU example

TABLE 7
OVERLAPPING CROSSES

Crosses	Coverpoints	Hits	Bins	Coverage
mcr	opcode, priv, debug, reg, en, tpm, hstr, excp, tpmcr	16	480	3.3%
tpm	opcode, priv, debug, reg, en, tpm, hstr, excp	158	2856	5.5%
ttlb	opcode, priv, debug, reg, hstr, excp, ttlb	18	312	5.7%
tid2	opcode, priv, debug, reg, hstr, excp, tid2	26	224	11.6%
tvm	opcode, priv, debug, reg, hstr, excp, tvm	166	1152	14.4%

TABLE 8
MULTI-CROSS HOLE ANALYSIS

hstr	reg	debug	excp	priv	Missed	Hole Effect
*	allis, asidis, vais	1	*	*	20671	34.19%
1	*	1	*	*	12878	24.94%
1	*	*	*	krnl, priv	7539	14.87%
*	*	1	*	n_user, mon	7905	14.68%
*	allis, asidis, vais	*	*	mon	7302	12.9%

As illustrated in the table, the common coverpoints have some missing combinations. The holes identified by these combinations affect each cross within the analyzed ones. The holes are ranked according to hole effect, the top hole identified by the algorithm occur due to a missing combination between "*debug* == 1" && "*reg* == allis || asidis || vais". Resolving this hole leads to average coverage increase of 34.19% over the analyzed crosses. We can also note that the 3rd top hole is ranked higher than 4th hole, even though it has a smaller number of missed bins.

The proposed technique applied to these crosses reduces the number of missed bins into 60 distinct buckets. For an even higher level view, each hole can be allowed to contain some covered items. This can be achieved through applying the quasi hole detection of lightly covered areas. The user needs to vary a threshold that would allow for some impurity in the extracted holes. Setting the threshold is particularly useful since the projection is lousy. By setting the threshold to 10%, 25 holes are generated instead, which are outlined in Table 9. It can be inferred that the second top hole in Table 8 got absorbed into the top hole of Table 9, which has a hole effect of 47.12%. This top hole identifies "*hstr*==1" as the most problematic bin in all of the crosses. Similarly, the top hole in Table 8 got absorbed into the large 2nd top hole in Table 9.

TABLE 9
MULTI-CROSS QUASI HOLE ANALYSIS

hstr	reg	debug	excp	priv	Missed	Total	Density	Hole Effect
1	*	*	*	*	25260	25994	2.82%	47.12%
*	allis, asidis, vais, biall,....	*	*	*	34092	36092	5.38%	36.06%
*	scr, sder, nsacs, mvbar,...	1	*	*	7677	7809	1.69%	28.15%
*	*	*	*	ns_mon	7786	7830	0.56%	13.99%
*	*	*	data_abort,hvc	*	4642	4689	1.39%	10.87%

V. CONCLUSION

Applying multi-cross hole analysis on real coverage models showed a great value by quickly and directly pointing to the problematic combinations of significant holes among the selected crosses. By selecting crosses that share semantic meaning or simply sharing coverpoints, salient problems can be uncovered directly saving the time and effort of analyzing the complete coverage data of each of the crosses independently. Our approach provides metrics such as the number of missed bins and hole effect to direct the verification engineer to which issue to debug first. The approach can equally work on coverpoints shared by all crosses or a subset of the crosses which enables the user to control the granularity of his holes. Quasi hole analysis also comes into play to capture lightly covered areas which might have resulted from the downward projection of a cross on the target coverpoints. The combination of user guided and automatic analysis ensures the most productive use of our precious man hours.

VI. FUTURE WORK

The starting point of our proposed algorithm is a user-selected subset of cover crosses and target coverpoints. This step can be automatically learned from the coverage data model through applying clustering algorithms. That would guide the user to the top holes without having to wait for the human-driven factor. This should bridge the gap between the complex covergroups and our analysis engine.

REFERENCES

[1] H. Azatchi, L. Fournier, E. Marcus, S. Ur, A. Ziv and K. Zohar, "Advanced Analysis Techniques for Cross-Product Coverage," in IEEE Transactions on Computers, vol. 55, no. 11, pp. 1367-1379, Nov. 2006.

[2] S. Asaf, E. Marcus and A. Ziv, "Defining coverage views to improve functional coverage analysis," Proceedings. 41st Design Automation Conference, 2004, San Diego, CA, USA, 2004, pp. 41-44.

[3] O. Lachish, E. Marcus, S. Ur and A. Ziv, "Hole analysis for functional coverage data," Proceedings 2002 Design Automation Conference (IEEE Cat. No.02CH37324), 2002, pp. 807-812.

[4] E. E. Mandouh and A. G. Wassal, "Guiding intelligent testbench automation using data mining and formal methods," 2015 10th International Design & Test Symposium (IDT), Amman, 2015, pp. 60-65.

[5] A. Yehia, "Faster coverage closure: Runtime guidance of Constrained Random stimuli by collected," 2013 Saudi International Electronics, Communications and Photonics Conference, Fira, 2013, pp. 1-6.

[6] H. w. Hsueh and K. Eder, "Test Directive Generation for Functional Coverage Closure Using Inductive Logic Programming," 2006 IEEE International High Level Design Validation and Test Workshop, Monterey, CA, 2006, pp. 11-18.

[7] Y. Adler et al., "Code coverage analysis in practice for large systems," 2011 33rd International Conference on Software Engineering (ICSE), Honolulu, HI, 2011, pp. 736-745.