System-on-Chip (SoC) Yield Visualization Using Custom Heatmaps in JMP

Robert Murphy
Member of Technical Staff
Advanced Micro Devices, Inc.
Austin, Texas, USA
Robert.Murphy@amd.com

Jason Paquette
Product Development Engineer
Advanced Micro Devices, Inc.
Markham, Ontario, Canada
Jason.Paquette@amd.com

***Abstract*—In order to maximize use of design resources and minimize time to market, system-on-chip (SoC) designs employ a substantial amount of functional block reuse. This reuse provides the basis for comparisons of performance and yield at the block level, enabling focused debug and further optimization opportunities. This paper walks through the process of generating custom maps for use with JMP's Graph Builder platform. This technique offers a valuable approach for exploring spatially-driven signals with a highly interactive tool. Drive enhanced process improvement by unlocking the power of custom maps in JMP.**

***Index Terms*—Custom maps, Hyper-heatmap, Root-cause analysis, SoC, System-on-Chip, Yield.**

I. INTRODUCTION

Today's SoC is often a relatively large device built on a leading-edge semiconductor process technology that seeks out ever-increasing performance and power efficiency. It is made up of an array of both replicated and unique blocks that serve various chip functions, such as I/O, compute and graphics. These blocks may be represented according to their physical positions on the die as a map. By using this type of map as a custom map in JMP, an engineer or analyst may create powerful visualizations that enable faster debugging of process, test or yield issues. In turn, this supports faster ramp to yield and quality, and thus faster return on investment (ROI) for new product designs. By combining die-based heatmaps in Graph Builder according to their physical locations on a wafer, previously unnoticed effects become illuminated. This is due to the fact that many semiconductor manufacturing challenges have strong dependencies on the spatial location within a wafer, such as center or edge effects. Clear illustration of these positional dependencies narrows the list of potential root causes considerably and accelerates process improvement.

II. BENEFITS OF USING CUSTOM MAPS

Currently there are various tools at the disposal of an engineer or analyst to choose from to identify outlier responses. A few examples of these include histograms and box plots. Though valuable and highly utilized, these tools lack the ability to illustrate physical positioning for identifying patterns much more quickly. Understanding these patterns helps to reduce the scope of subsequent analysis and root-cause isolation, leading to faster turn-around times for debug and yield improvement.

Though this paper focuses on the semiconductor industry, the application of custom maps can find itself useful in many settings with spatial signals.

III. CREATING CUSTOM MAPS IN JMP

The generation of custom map shapes in JMP is performed by connecting the vertices of each shape, one-by-one. To accomplish this, JMP requires vertex coordinate information in the form of two definition files: a '-Name' file and an '-XY' file. The '-Name' file contains required numeric *Shape ID* and *Name* columns and the '-XY' file contains *Shape ID*, *Part ID*, *X* and *Y* coordinates of the vertices. The *Shape ID* column is what links the coordinates in the '-XY' file to the shape names in the '-Name' file.

If vertex coordinates are available, the '-Name' and '-XY' files can be generated in a few steps.

Summarize the coordinate file so each shape occupies a single row in the data table. Create a 'Shape ID' column in ascending order (starting from one) giving each shape a unique ID. Shape ID must be the first column in the table. Save the table as '*$$$*-Name.jmp' where *$$$* is a chosen

prefix. In this file, there should be a user-friendly name column to represent each shape. Within the column properties of this column, select Map Role and set it to 'Shape Name Definition' (see Figure 1). This is an important step as the values in this column will be used to link raw data for overlays.

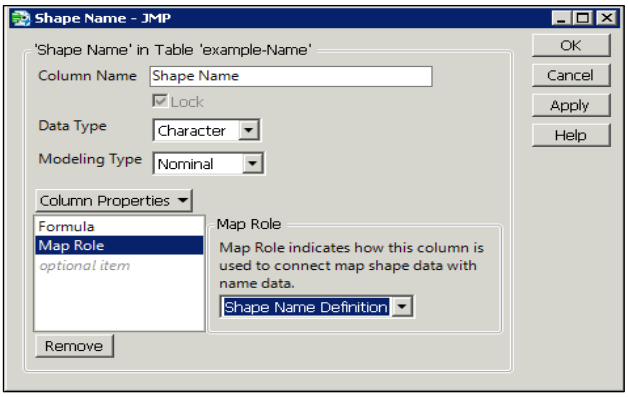

Fig. 1. Shape Name Definition Map Role Property Setup

With the original coordinate table still open, merge in the unique Shape ID column that was just created and sort the table in ascending order by Shape ID. Save the data table as '*$$$*-XY.jmp' where *$$$* must be the same prefix used for the '-Name' file.

This '-XY' file requires four mandatory columns, *Shape ID*, *X*, *Y*, and *Part ID*. In its current state, the file should have all of this information and might just need the columns to be renamed. However, it will be missing the *Part ID* column. This column essentially tells JMP if certain shapes have multiple, separate parts, such as the state of Hawaii which is comprised of multiple islands. Often it is the case where each shape has only a single part. In those cases, the Part ID column can be set as all ones. The *X* and *Y* columns contain the coordinates of each vertex in each shape. A subtle detail worth highlighting is that the order of appearance for each shape's vertices in the '-XY' file is important as it can lead to inaccurate representations of shapes, if misconfigured. Using the '-XY' mapping file, JMP traces shapes stepping from vertex to vertex (starting from the first row in the file to the last). A new trace starts with each new Shape ID or Part ID. Figure 2 shows an example of how the ordering of vertices makes a difference for tracing a simple rectangular shape.

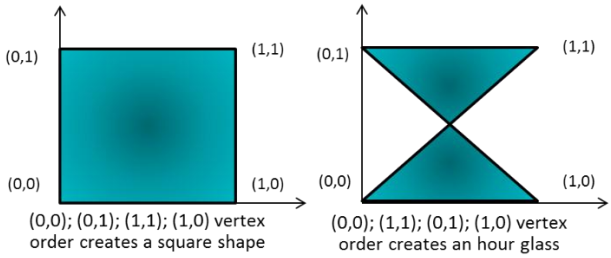

Fig. 2. Correct And Incorrect Vertex Ordering For Shapes

If the raw coordinate data is consistent across multiple type of custom maps, a JMP scripting language (JSL) script to create the '-XY' and '-Name' shape definition files would lend itself useful.
For more information on custom map generation check out the following resource: Help>Help Contents>Essential Graphing>Create Maps>Graph Builder>Custom Map Files.

One final point to note is that both '-Name' and '-XY' shape definition files must be saved within the same directory with the same prefix for mapping to function correctly.

IV. CUSTOM MAP USAGE

With the definition files configured, data can be imported into JMP. A final preparation step is needed to link the data with the custom map definition files before visualizing data in Graph Builder. There must be a column in the data that has values that identically match those found in the defined *Shape Name Definition* column in the '-Name' file, otherwise JMP will not be able to link the custom map.

To link the data table to the custom map, open the column properties of the column within the data table that stores the shape names. Within the Map Role property, make sure that the 'Shape Name Use' option is selected and select the '-Name' file as the 'Map name data table' with the appropriate Shape definition column chosen (see Figure 3). Note, in JMP11 these options do not have an interactive file dialog to search for the '-Name' file, so it will need to be copy and pasted or typed into the textbox (including the '.jmp' extension).

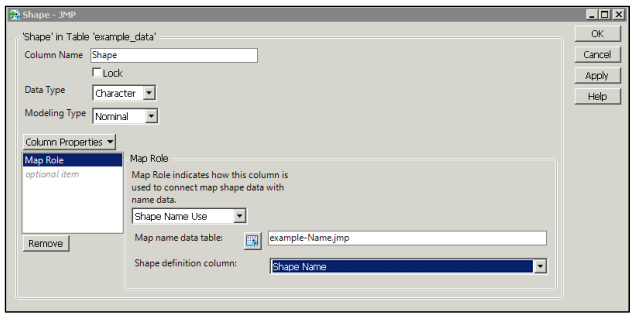

Fig. 3. Shape Name Use Map Role File Selection

JMP also supports placing the '-Name' and '-XY' files in a specific directory that the software checks automatically. However, in our application we use identical shape names with different mappings across different SoC products. So, in order to avoid name collisions, we utilize the file selection method. Please refer to JMP's Help to find the automatic directory location based on your computer's operating system.

With all of the prep work complete, open Graph Builder and drag-and-drop the linked shape column from the data table into the Map Shape drop-zone at the bottom-left corner of the window (see Figure 4). JMP then fills the display with all shape instances found in the data table that are also found in the shape definition files.

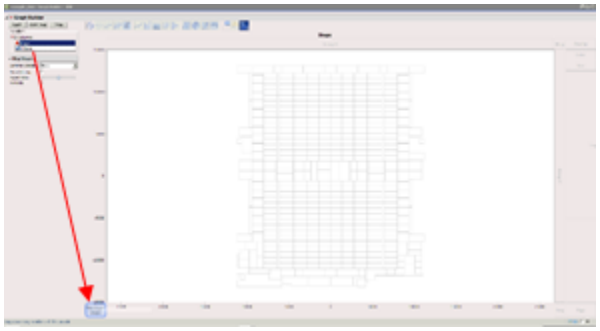

Fig. 4. Graph Builder Map Shape Drop Zone Location

There might be cases where certain shapes found in the data table are not recognized by the shape definition files, this could be due to names not being identical, different naming conventions, or they are simply not listed. These unrecognized shapes are displayed at the bottom of the Graph Builder window as 'Map shapes not found' (see Figure 5). Incorporating these missing shapes requires either modifying the data to match the shape definition files or to update the shape definition files themselves.

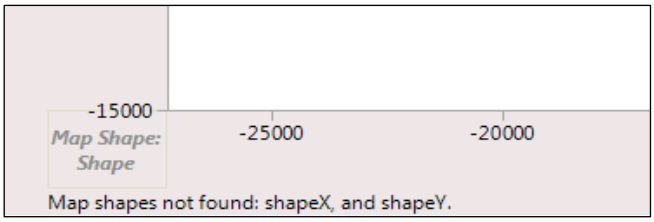

Fig. 5. Graph Builder Display When Shapes In Data Not Recognized

At this point data columns can be dragged-and-dropped onto the *Color* drop-zone which provides the appropriate color overlays based on the column data and modeling types.

The next section will go over a few detailed examples outlining different use-case scenarios and success stories.

V. EXAMPLES

A. Yield Data

Echoing the introduction, the benefit of having block reuse in SoC designs allows for a means of comparing performance and yield of individual blocks. It is very common to have test procedures such as Scan (flip-flops and common logic) and Built-In Self-Test (BIST, memory test and repair) that disposition on the block level. In the case of SoCs, these blocks are referred to as tiles. Figure 6 depicts the yield of a specific test for each tile by aggregating the results of a few thousand units. The tiles colored in red (and numbered in white) are low-yielding outlier tiles that warrant more attention. It will be shown in the next sub-section that there is another force that is driving these outliers that is not apparent when depicted in this view.

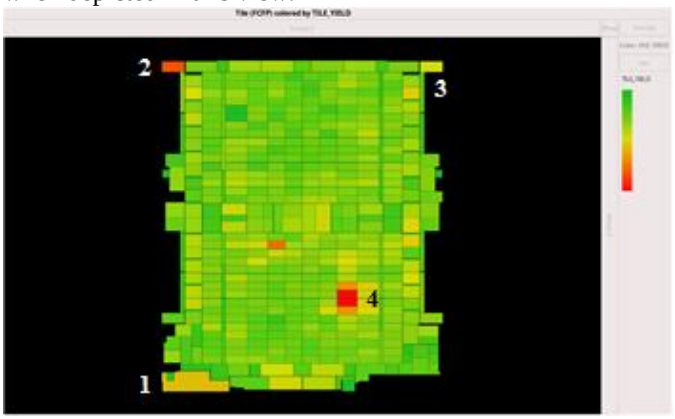

Fig. 6. SoC Tile Yield Heatmap With Outliers

B. Hyper-heatmaps

Continuing with the previous yield example, there is an obvious set of outlier tiles. However, it is not clear what might be wrong with them. Figure 7 depicts the hyper-heatmap that

nests each SoC tile view for each die location on the wafer, making use of the *Group X* and *Group Y* drop-zones. Die locations that are sparsely populated have low rates of tile failures. The numbered outliers from Figure 6 have been added to Figure 7 representing the die locations that drive those tile yields to be lower. For tiles *1*, *2*, and *3*, the primary reason is that they are corner tiles that are adjacent to the edge of the wafer for certain die X, Y locations. When immediately next to the wafer edge, these tiles are subjected to edge affects in wafer manufacturing processes that do not occur in other areas of the wafer. In order to more clearly view a hyper-heatmap with many objects, it may be useful to filter out the baseline or expected levels of the color variable. Figure 7 has been filtered in this way. Tile *4* yield fallout is focused primarily at the center of the wafer where there are also known variation effects due to the fabrication process. Manufacturing processes induce a variety of non-uniformities across the wafer, which are often stronger at the center or edge regions. This hyper-heatmap allowed for quick understanding of where the trouble areas were without spending resources to try to determine if the tiles themselves needed design optimization.

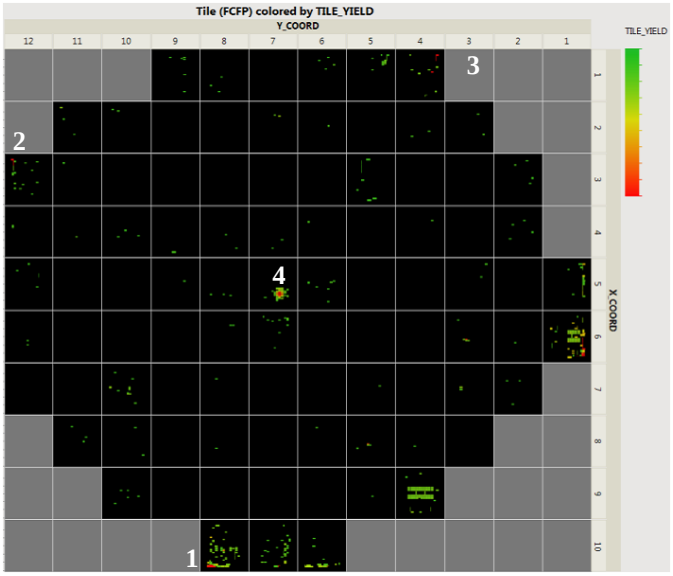

Fig. 7. Wafer-level Hyper-heatmap Highlighting Outlier Die Locations By Filtering Baseline Yield Rate

C. Parametric Data

Figure 8 is an example of how parametric data can be visualized on a wafer-level hyper-heatmap. Each tile shown contains an analog measurement structure whose value is used as a metric to correlate to device performance. The data table contains hundreds of wafers, and this particular view is showing the mean value for each tile's measurement at each die location on the wafer. Familiar remnants of the fabrication process are clearly evident by the concentric circles depicted by the overlay.

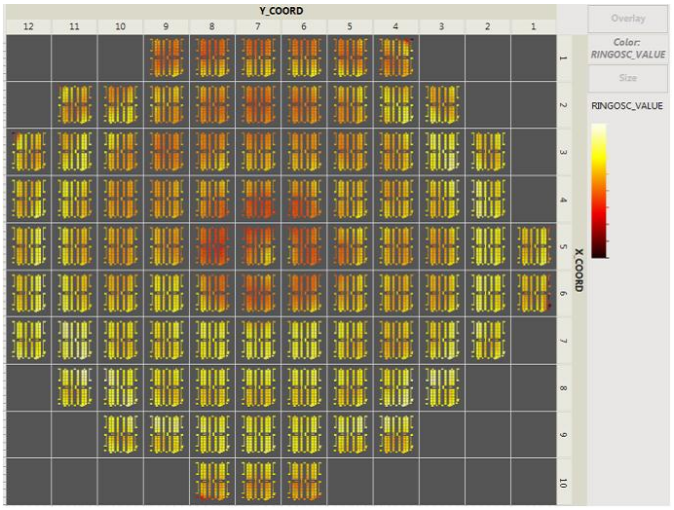

Fig. 8. Wafer-level Hyper-heatmap Of Analog Characterization Structure

VI. Success Stories

High volume product showed a noticeable outlier tile yielding worse than others (see Figure 9). In this case, red depicts lower yielding tiles and blue are yielding higher.

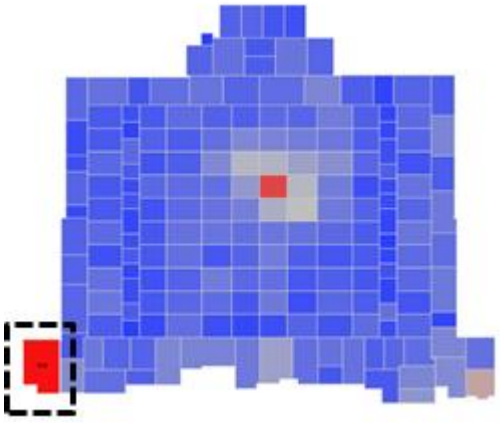

Fig.9. Initial SoC-level Yield With Outlier Tile Observation

Further analysis utilizing heatmaps led to the discovery that this was a systematic wafer-edge signature for the outlier tile on the lower left edge of the wafer where this tile was adjacent to the wafer edge (see Figure 10). This signal was brought to the foundry's attention, which led them to implement a

process improvement – optimized lithography adjustments at the extreme wafer edge. Overall wafer yield improved significantly due to this process improvement.

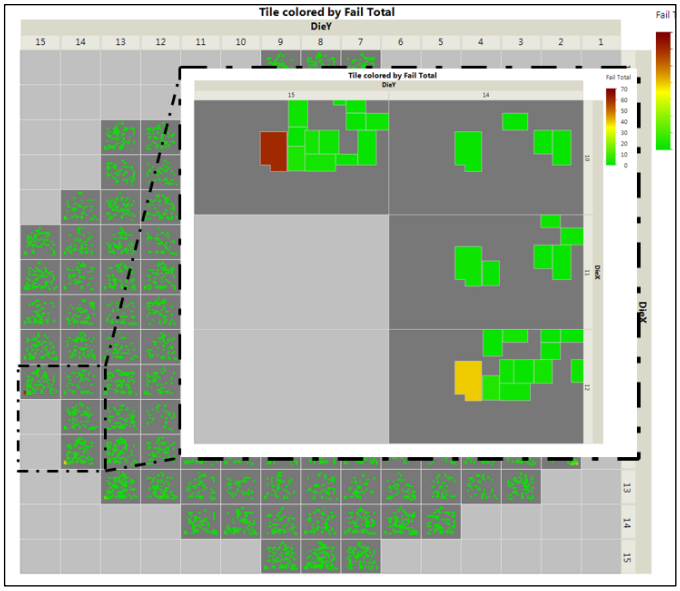

Fig. 10. Wafer-level Observation Of Outlier Tile At Lower Left Wafer Edge

Prior to the hyper-heatmap view, the initial investigation of this yield signal was focused on the tile itself, i.e. the wrong direction in this case. Once the wafer edge yield correlation was clear, the correct owner was able to immediately begin a continuous improvement effort.

VII. TIPS & TRICKS

A. Show Missing Shapes

Just below the column selection box within the Graph Builder platform, there is check-box titled, *Show Missing Shapes*. Selecting this populates the display with outlines of shapes that are in the definition files but not found in the current data table (see Figure 11). This is useful for spotting missing data or getting an overall picture of the map.

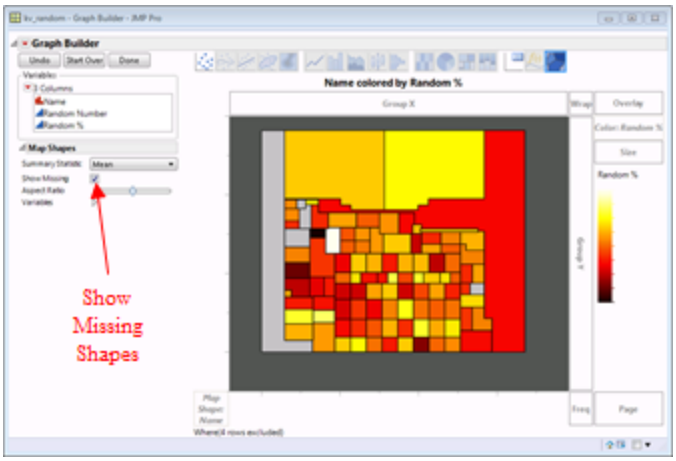

Fig. 11. Graph Builder Show Missing Shapes Checkbox

B. Enabling Data Label Overlays

Enabling the ability to hover over shapes or have labels displayed over-top of each shape is useful for quickly identifying patterns. Right-clicking on the column(s) of choice and selecting *Label/Unlabel* for the first time displays column data while hovering over shapes in Graph Builder. To statically display the labels on the shapes, select the rows in the data table of interest and select *Label/Unlabel*.

C. Automation with Filters and Column Switcher

When analyzing volume data, searching for subtle patterns can be an onerous task and minimizing the amount of mouse-clicks always helps. Making use of the *Filter Automation* and *Column Switcher* features (Graph Builder Red Arrow>Script) is an efficient way to scan through vast amounts of spatial data with different conditions.

D. Frame Paging with JMP12

The Graph Builder platform has a few noticeable upgrades in functionality with the recent JMP12 update, namely, the ability to split into multiple frames by using the *Page* drop-zone. Placing a data column into this drop-zone generates multiple instances of the Graph Builder frame according to the number of unique values found in that column. The report can be saved as an Interactive HTML document to be shared.

E. Frame Segmentation with Group and Wrap Drop-zones

As seen in the example of the wafer hyper-heatmap, frames can be segmented by using nested columns within the *Group X* and *Group Y* drop-zones. The wafer hyper-heatmap used die X and Y coordinates for segmentation, but any

columns can be used. Other examples frequently used are voltage test point, test temperature, and frequency test point. The *Wrap* drop-zone has similar functionality with more efficient space usage, but column nesting is not available. Right-clicking on the Group Variables allows the user to select the number of levels in view to effectively zoom into regions of interest (see Figure 12).

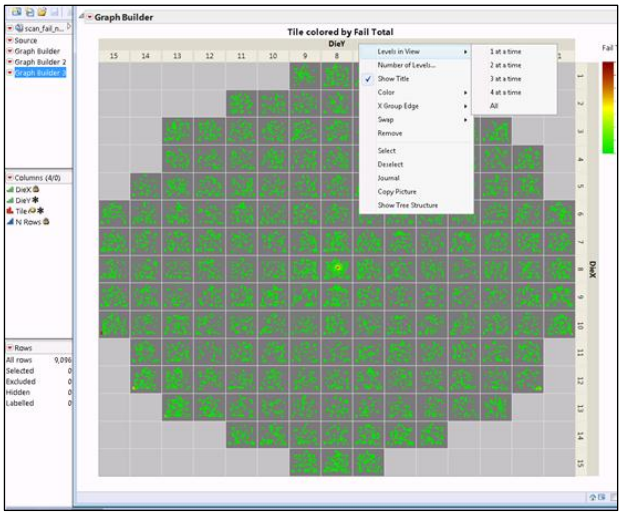

Fig. 12. Selecting Levels In View Option Within *Group* Variables Zones

F. Modifying Graph Builder Color Settings

When analyzing custom map overlay data, often a simple modification to the color settings can emphasize the appearance of outlier patterns significantly. The authors recommend using a dark background (Graph>Background Color) with a sequential color theme for single direction data (yield, speed, etc.) and a diverging color theme for two-sided data. Color themes can be accessed within the Gradient Settings by right-clicking the legend. By default Graph Builder adds gray border around each shape, one pixel in width, which might overshadow some of the smaller shapes. To remove this border right-click in the frame and select Customize>Map Shapes and change the line width to zero.

VIII. KNOWN ISSUES & CAUTIONS

A. Overlapping Shapes May Not Display Desired Color

This arises when there are shapes defined in the custom map that have overlapping regions. In Figure 13, the *acp* and *gnb* blocks share a common region, but only one color is used. This is something to be aware of when creating custom maps that contain any overlapping shapes.

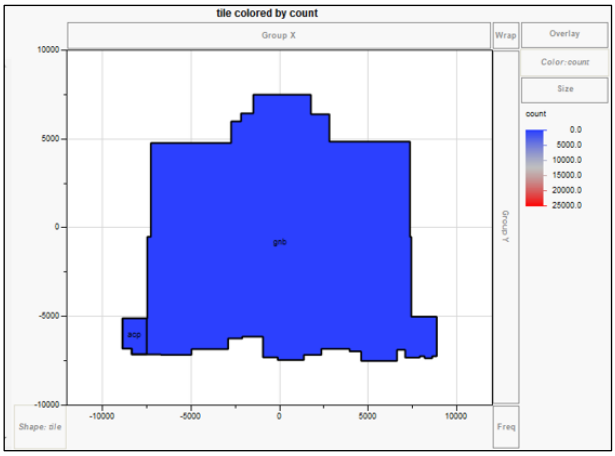

Fig. 13. Incorrect Color Overlay Of Two Overlapping Shapes

B. Dragging Label to Create Pointer Displays Unlabeled Row

Figure 14 shows how dragging a desired label (dotted black border) caused an extraneous label to appear (dotted red border) for an unlabeled row. This issue is fixed in JMP 12.

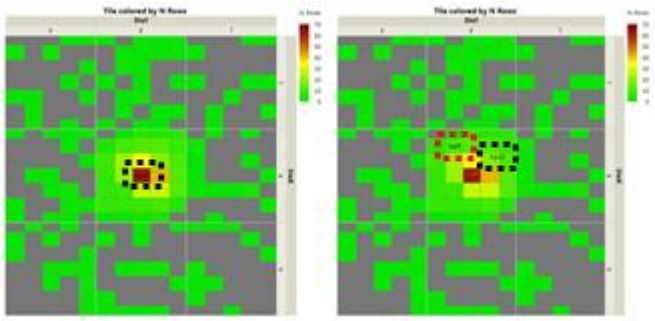

Fig. 14. Dragging Label (black) Causes Unlabeled Row To Appear (red).

C. Changing Levels in View Resets Frame Customizations

Figure 15 shows the progression of first setting the frame customizations for the default view, selecting four levels in view with one of the *Group* variables, and finally when moving back to the original view, the customizations originally applied are lost.

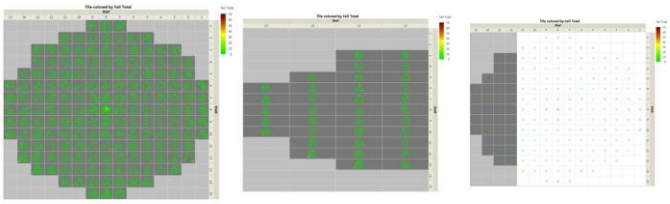

Fig. 15. Changing Level In View Feature Causes Loss Of Frame Customizations

D. Map Role Column Property and Summary Tables

If a column has a Map Role property applied, and the data table is summarized (with that column included), Graph Builder does not recognize the column property. It simply states, that there is no shape file found, even though it has been defined in the Map Role property. The workaround is to remove the Map Role property and add it again. Alternatively, if the file is saved, closed, and re-opened then the Map Role property will then function as expected.

E. Cannot Update Map Role Property

When using the Map Role column property, it was observed that once the property is set and a shape definition file has been applied, changing to select a different file will not update to the new file's settings. Also, the shape definition column cannot be updated. The workaround is to remove the Map Role property and add it again with the new file and column settings.

IX. ENHANCEMENT OPPORTUNITIES

Custom maps within the Graph Builder platform is a stand-out performer for analyzing patterns that correlate to spatial data with an exceptional interactive interface. As noted in this paper, this feature has enabled successful root-cause analysis leading to real financial impact. With extensive use of this functionality, its gain is evident and extending its usefulness with a few optimizations will only lead to better results.

Below are a few enhancement opportunities the authors have compiled based on their experiences.

A. More Options for Levels in View

Graph Builder offers the ability to segment the view with the use of *Group* drop-zones on both the *X* and *Y* axes. The right-click menu offers a few options for setting the number of levels in view, but it is fairly rigid (see Figure 16). Often more than four levels in view (but less than all) are desired but currently that option does not exist. This could be altered to be dynamic based on the number of values in the group column.

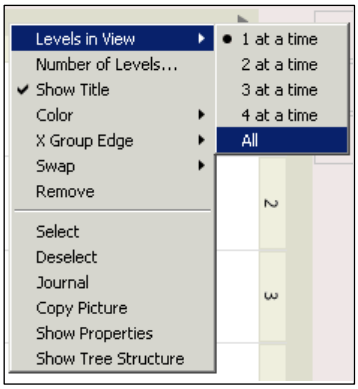

Fig. 16. Levels In View Options Limited To 1-4 Or All

B. Support Multi-column Paging

This is an incredibly useful feature for quickly scanning through multiple views with a simple scroll of the mouse. This feature could be further optimized if it could allow for more than one column to split the frames, similar to the '*By'* feature in other platforms. The workaround right now is to create a new column and concatenate the data from more than one column together to get different combinations, but multiple column paging would be a cleaner solution.

X. ACKNOWLEDGMENT

The authors would like to thank Paul Shook for his help in adding a special output format to AMD's floor plan software that aides considerably in converting to JMP's custom map format. Additionally, we'd like to thank fellow analysts at AMD for help in evaluating use of custom maps for SoC yield visualization in JMP.