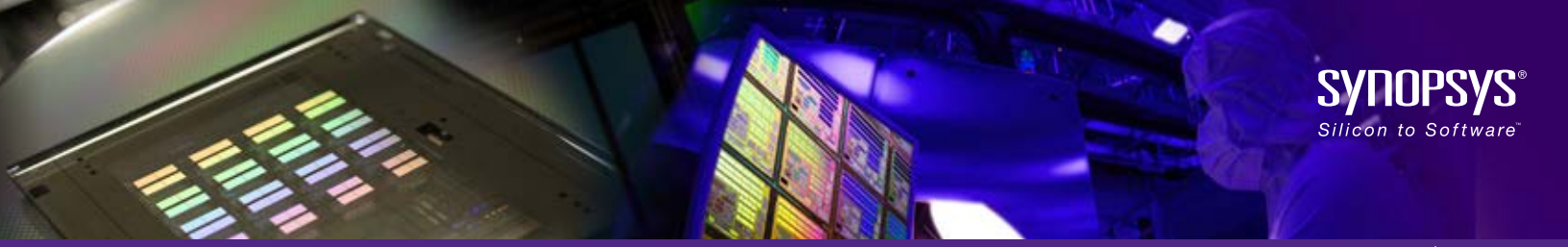

Newsletter for data prep engineers

Issue 6 | Fall 2015

CATS news

Dear CATS User:

Mask manufacturing at technology nodes below 20nm presents new challenges for mask lithography and verification. Release K-2015.09 of CATS breaks new ground in addressing these challenges with comprehensive support for multi beam mask writers. In addition, this release delivers higher fracture performance and new capabilities to meet the throughput requirements as mask data sets get larger and more complex. CATS is the most widely used mask data prep solution and this is dependent on our customers success in using the product and on our ability to meet your needs. We look forward to your feedback on this newsletter and working with you in ensuring that CATS addresses all your requirements for mask data prep. Thank you very much for your continued support of CATS.

Anjaneya Thakar
CATS, Product Marketing Manager

An Abbreviated CPEC User Guide: Simple Steps to Improve the Dose Correction

Alex Zepka, CATS CAE

CATS CPEC allows for the correction of e-beam proximity effects due to e-beam scattering. This correction is done via modulations on the exposed dose and can be applied to any lithographic process that involves the exposure of a material stack by an electron beam, e.g., mask manufacturing as well as direct-write to wafer.

The setup for CPEC can be a bit overwhelming as some settings are interdependent. This article attempts to provide some guidance about typical cases and what types of parameters are most likely to require finer adjustments in order to improve the correction. It is not intended to be a complete manual (that is provided in the CATS documentation) and it does not cover all CPEC settings. Instead, it is meant to help the user in deciding which settings to use in most cases, and which settings are more likely to require fine-tuning in order to improve the quality of the correction or turnaround time.

Minimum Settings ("Pipe-cleaning" Mode)

When using CPEC for the first time, the user may merely be trying to obtain a simple correction to visually verify that CPEC is indeed assigning doses. In this "newbie" scenario, the default configuration can be turned on by using:

```
Command: CPEC YES                                                    (1)
```

A simple upgrade on this correction would be to turn on subfracturing. In this case, add:

```
Command: CPEC SUBFRACTURING MANHATTAN                                (2)
```

Continued on page 2

In This Issue

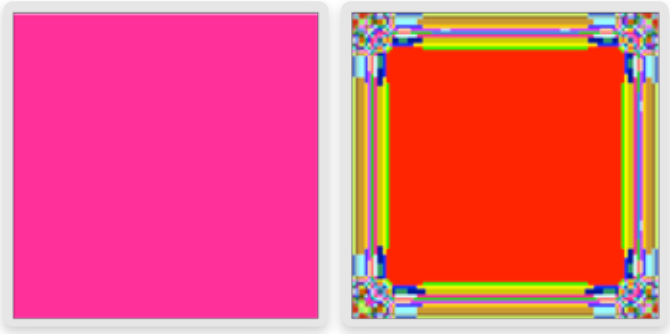

Figure 1. Results for "CPEC YES" (left) and adding "CPEC SUBFRACTURING MANHATTAN" (right).

Figure 1 displays the output of these two settings for the case of a square with dimensions of 100x100 μm^2. The result on the left corresponds to (1) above, while the figure on the right shows the result if subfracturing Manhattan is added as in (2). The Draw Option is Color_By > Datatype. As expected in the left of this figure, CPEC contains a single dose (CFA=17). Here, the settings of HEIGHT and WIDTH were such that the edges of the input and output shapes are identical.

On the other hand, the result on the right shows how subfracturing adds an increasingly large number of shapes closer to the square corners, improving the overall correction. Also, as it will be shown later in this article, using SUBFRACTURING NONE may be enough if the shapes to be corrected are small (relative to the range of the proximity effects).

Now that we know how to get a result with CPEC, we need to customize the settings. Start by displaying the default settings. This can be done by typing "CPEC ?" in the CATS Command line:

```
Command: CPEC ?
CPEC is enabled
CPEC FILE VOID
CPEC NUMBER_OF_DOSES 256
CPEC SUBFRACTURING none
CPEC SUBFRACTURING_MIN_DIMENSION 0
CPEC SUBFRACTURING_GRID 0
CPEC SR_RESOLUTION 0
CPEC LR_RESOLUTION 0
CPEC NUMBER_OF_ITERATIONS 4
CPEC CORRECTION_MODE edgeplacement
CPEC CORRECTION_QUALITY fast
CPEC TARGET_LEVELS 1
CPEC PSF_BEAM_WIDTH 0
CPEC PSF_TYPE gaussian
CPEC PSF_POINTWISE_FILE VOID
CPEC PSF_SEPARATION_RADIUS 0.1
CPEC PSF_VALUES_RADIAL 0.05 10
CPEC PSF_VALUES_WEIGHT 1 0.5
CPEC PSF_UNITS um
CPEC DOSE_TABLE_FILE VOID
CPEC WRITE VOID
CPEC DOSE_VALUES VOID
```

Settings that have interdependencies are shown in the same color above. A best practice is to address them in groups, starting with the setting of the dose table.

Setting up the Dose Table

The default settings for the doses that are available in CPEC when applying the correction are estimated based on the assumption that the layout area coverage varies within its maximum range: from 0.0 (empty) to 1.0 (fully covered). In a general layout, that is not always the case, and the number of doses that are actually used in the correction is smaller than what is available. Therefore, it is sometimes necessary to manually reset doses in order to maximize the number of dose tags used in the correction.

There are two ways to manually reset the default dose table:

1. By setting the number of doses, and the minimum and maximum dose values. In this (most common) case, the user will first pick the **number of doses** that will be available for the correction. This value is typically tied to the writer tool (that the output is slated for) and it is set by default to 256. If the output is not scheduled for exposure (such as for simulation purposes), it is often the case that the default setting is appropriate. The **minimum and maximum dose** values are set using the CPEC DOSE_VALUES *<min> <max>* command. From these three settings, the entire dose table is constructed, going from minimum to maximum and with constant dose steps of (max-min)/(Number_of_Doses-1).
2. By inputting the doses from a dose table file. If the user already has a list of doses, these can be directly used in the correction via "CPEC DOSE_TABLE_FILE <file>". This list is a simple ASCII file with each dose corresponding to a row in the format:

```
Dose_Tag    Relative_Dose_Value
```

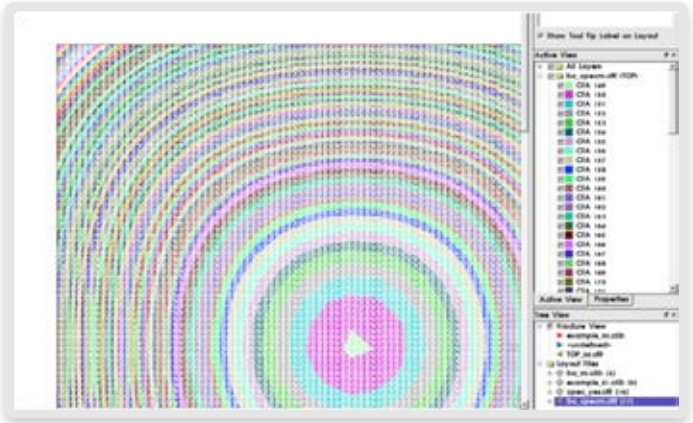

Figure 2. Results using default dose table.

If the dose settings are left untouched, the automatic setting of the dose table might result in a very narrow range of dose tags being used in the corrected layout, particularly if the layout area coverage does not vary much. For instance, in the example shown in Figure 2, by using the default dose settings, only CFA tags between 149 and 225 end up being actually used. This example consists of an array of identical shapes with sizes and pitches of the order of 100 nm. CFA tags below 149 are not present because there are no large shapes in this layout that would get assigned lower doses while values above 225 would correspond to small *isolated* features, which are also not present.

In order to customize the dose table and maximize the number of dose tags used in the correction, it is necessary to manually set the minimum and maximum dose values by using the CPEC DOSE_VALUES setting in a subsequent CPEC run. The minimum and maximum dose values are determined from the smallest and highest dose tags (respectively) displayed in the first run. In this example, tag 149 corresponded to dose=1.188 and tag 225 corresponded to dose=1.412 (both values are obtained from the *.dose file created in the preceding run). Therefore, by using:

```
Command: CPEC DOSE_VALUES 1.188 1.412
```

a new outcome that uses a larger range of CFA tags is generated (from CFA=2 up to CFA=254):

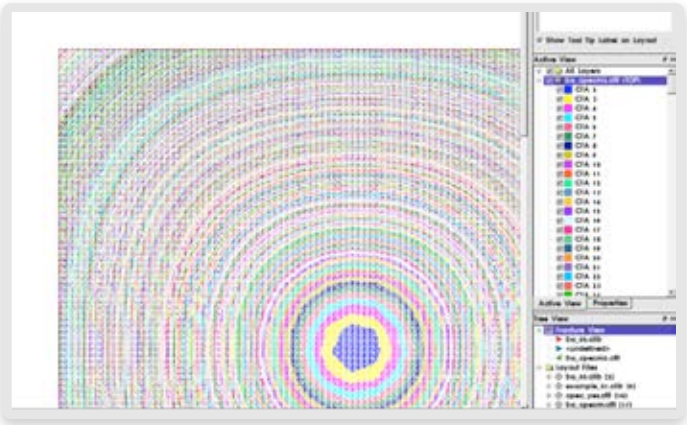

Figure 3. Results obtained by tweaking dose table to expand the number of doses actually assigned.

In order to optimize the correction, it is important to maximize the number of doses actually used in the correction.

When Should I Use Subfracturing?

When deciding on SUBFRACTURING settings, it is important to compare two dimensions:

a. The size of the largest fractured shapes in the layout

b. The range of the proximity effects. Most often, the energy of the e-beam in the writer tool will be of the order of 50 keV. In that case, the range of the proximity effects that can be corrected with CPEC will be around 10 µm.

When (a) above is comparable or larger than (b), the level of backscatter within these "large" shapes is enough to warrant subfracturing. In Figure 4 below, a simple example containing 10 long narrow lines (1000 µm x 20 nm) demonstrates why subfracturing is required in order to properly correct for proximity effects with a range of the order of 10 µm. As the lines are very long, the backscatter changes in a scale ~1 µm. If SUBFRACTURING NONE is used, a single dose would be assigned to correct for the backscatter (top result in Figure 4). That clearly is not sufficient and subfracturing should be allowed to break the lines into segments of ~1 µm long as in the lower result in Figure 4.

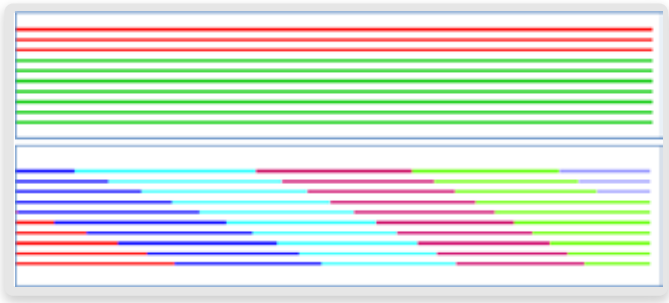

Figure 4. Top: CPEC correction using SUBFRACTURING NONE. Bottom: CPEC correction using SUBFRACTURING MANHATTAN. In the top result, each of these lines (20 nm width, 1000 µm long) is assigned a single dose by CPEC, while in the bottom result, CPEC will break the lines into pieces of (nearly) constant backscatter before assigning doses to each piece. The window here is only 7.4 x 1.2 µm, displaying only the line ends.

On the other hand, if all shapes in the layout all small compared to the range of the proximity effects, it is safe to use SUBFRACTURING NONE. The fracture will be thereby faster with little or no impact to the fracture runtime. Figure 5 shows an example where all shapes are "small" enough (<0.2 µm) so that simply assigning a dose to the fractured shapes (SUBFRACTURING NONE) is enough to properly dose-correct the layout.

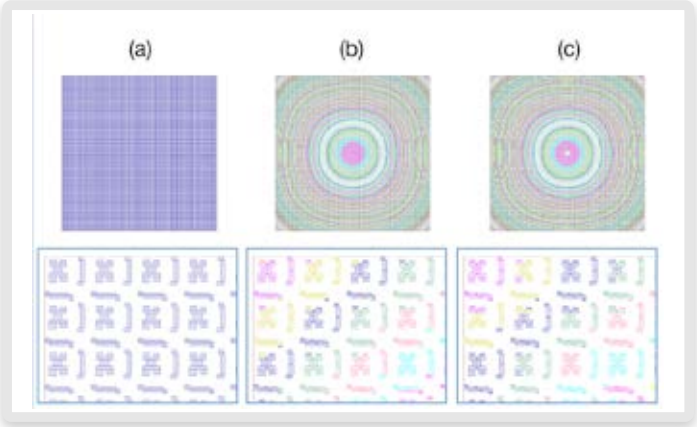

Figure 5. (a) uncorrected fracture; (b) CPEC correction using Manhattan subfracturing; (c) CPEC correction using no subfracturing (LR_RESOLUTION=0.2 µm). The bottom row shows the zoomed-in upper-left corner of the results in the top row. In this example, it is not necessary to turn on subfracturing when using CPEC.LR_RESOLUTION

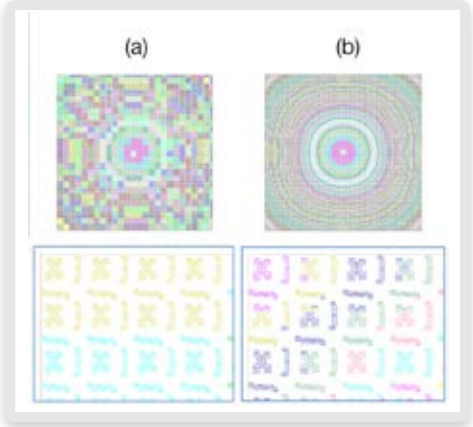

Figure 6. (a) CPEC correction using default settings (SUBFRACTURING NONE); (b) CPEC correction using no subfracturing but modifying LR_RESOLUTION to 0.2 µm. The zoomed-in details are as in Figure 5, PSF Settings and Separation Radius

While the size of the fractured shapes provides enough information to decide whether to enable subfracturing, it may be necessary to modify the setting of LR_RESOLUTION when using SUBFRACTURING NONE. The value of LR_RESOLUTION is tied to an internal grid that CPEC uses for the calculation of the long-range PEC. Normally, this setting is derived from the long-range component of the PSF, but it might need to be refined if the user wants a smoother dose gradient through the corrected layout. The outcome of using the default setting of LR_RESOLUTION and SUBFRACTURING NONE for the example of Figure 5 is shown on the left in Figure 6. If the user sets LR_RESOLUTION to 0.2 µm, the blocky appearance of the correction is smoothed out and the result on the right is nearly identical to what one obtains from SUBFRACTURING MANHATTAN (as shown in Figure 5).

The value of LR_RESOLUTION has an effect only when SUBFRACTURING NONE is used. If the user switches to SUBFRACTURING MANHATTAN, this property has no effect in the correction outcome; instead the setting of SUBFRACTURING_MIN_DIMENSION allows for a refinement of the correction (the smaller this setting, the smaller the resulting subfractured shapes).

In summary, take note of the following association when attempting to improve the CPEC correction:

- LR_RESOLUTION impacts the correction when using SUBFRACTURING NONE
- SUBFRACTURING_MIN_DIMENSION affects SUBFRACTURING MANHATTAN

The actual PSF model must be available prior to actually applying the correction in CPEC. If the user does not have a PSF model, CPEC will load in a default model based on a 50 kV e-beam and typical resist and material stack settings. The PSF settings are:

```
CPEC PSF_BEAM_WIDTH 0
CPEC PSF_TYPE gaussian
CPEC PSF_POINTWISE_FILE VOID
CPEC PSF_SEPARATION_RADIUS 0.1
CPEC PSF_VALUES_RADIAL 0.05 10
CPEC PSF_VALUES_WEIGHT 1 0.5
CPEC PSF_UNITS um
```

Normally, the beam width is included in the PSF model and should be left set to “0”. The most common representation of the PSF models is a sum of Gaussians, typically 2 or 3 components. These Gaussians are defined by the PSF_VALUES_RADIAL and PSF_VALUES_WEIGHT settings; the first setting contains the Gaussian “sigma” values (related to their respective ranges) while the second contains the relative weights. It is important to note that the PSF_VALUES_RADIAL are larger than the usual definition of a Gaussian sigma by a factor of sqrt(2). In other words, if for instance the PSF model is:

$$PSF(r) = \eta_1 e^{-(r^2/2\sigma_1^2)} + \eta_2 e^{-(r^2/2\sigma_2^2)}$$, with σ_1=0.05 µm and σ_2=10 µm

the values of PSF_VALUE_RADIAL should be set as:

```
CPEC PSF_VALUES_RADIAL 0.071 14.1
```

One other item to be considered when entering the PSF settings is the PSF_SEPARATION_RADIUS value. This setting will determine which PSF components will be used in the correction. For instance, the settings:

```
CPEC PSF_SEPARATION_RADIUS 0.1
CPEC PSF_VALUES_RADIAL 0.05 10
```

will result in a CPEC correction for the "long-range" component (10 µm) of the PSF, with the "short-range" component (range ~ 50nm) being ignored. Be careful that even though the short-range component is not corrected in CPEC, it is still important to input the correct relative weights (from the PSF model) in the PSF_VALUES_WEIGHT settings. Although it is common to renormalize these values so that the first Gaussian weight is set to "1", this is not a requirement. For instance, the settings:

```
CPEC PSF_VALUES_WEIGHT 0.1 0.05
```

and

```
CPEC PSF_VALUES_WEIGHT 1 0.5
```

are equivalent as CPEC renormalizes these values before applying the correction anyway.

Correction_Mode and Target_Levels

In a majority of cases, CPEC is used to achieve CD fidelity by making sure the edges of the corrected shapes print on target. If that is the user's goal, the appropriate selection is (default in CPEC):

```
CPEC CORRECTION_MODE EDGEPLACEMENT
CPEC TARGET_LEVELS 1
```

On the other hand, if the main goal is to have specific shapes to achieve a user-defined dose level. For example, if a user wanted to define 1.2 times the nominal dose, one would use instead:

```
CPEC CORRECTION_MODE TARGETLEVEL
CPEC TARGET_LEVELS 1.2
```

TARGET_LEVELS can take a list of values with each one corresponding to a datatype tag, starting with datatype=0. For instance, if there are three datatypes in the layout corresponding to:

- datatype=0 must achieve 0.75x the nominal dose
- datatype=1 must achieve 1.0x the nominal dose
- datatype=2 must achieve 1.5x the nominal dose

The correct setting is (as expected):

```
CPEC TARGET_LEVELS 0.75 1.0 1.5
```

Finally, the two correction modes can be mixed in the same run by toggling the sign of the value in TARGET_LEVELS. For instance, assume datatype 3 is added to the correction example above and edgeplacement is used as the correction mode for this extra datatype. This can all be achieved in the same CPEC command by doing either or the following commands (they are equivalent):

```
CPEC CORRECTION_MODE TARGETLEVEL
CPEC TARGET_LEVELS 0.75 1.0 1.5 -1.0
```

Or

```
CPEC CORRECTION_MODE EDGEPLACEMENT
CPEC TARGET_LEVELS -0.75 -1.0 -1.5 1.0
```

The sections above provide some guidance on how CPEC is run in the vast majority of cases. More details about the CPEC commands are provided in the CATS documentation (CATS Command Manual) and by contacting the CATS Customer Support: cats@synopsys.com. ■

Achieving Faster Layout Coverage Calculations with the DENSITY Command

Yeghia Dolbakyan, CATS R&D

There are often questions asked by chip vendors during mask preparation process, such as what is the area, how many shapes will be exposed on the mask, or what is the percentage of the area covered by geometries?

DENSITY Command

Recently, a new command called DENSITY was added to CATS®. This command is used to perform fast area (coverage) calculations on the specified layout file, and supports all formats accepted by the application. Unlike the AREA command, the DENSITY command works much faster in most cases, but is not meant to be a replacement for AREA, since results calculated by AREA are more precise. Similarly to AREA, DENSITY does not remove overlaps, and as such, the coverage value can exceed 100%.

The simplest run can be performed with the following command:

```
Command: Input <Input File>
Command: Density

Coverage: 82.102%
Processing time: 0sec
```

The output of the DENSITY command is the percentage coverage and the time spent on the calculation.

It is possible to specify the file space (PRIMARY, SECONDARY, or OUTPUT) on which to apply the DENSITY command. For example,

specifying SECONDARY as an argument will result in the calculation of the coverage of the secondary file:

```
Command: DENSITY SECONDARY
```

DENSITY can also be used to output an image, which gives the user an overall view of the dense areas on the particular layers. The IMAGE command splits the layout into a grid of pixels, where the pixel size is defined by the <*width*> and <*height*> parameters specified on the command line. The *density* value of a pixel is the percentage of the area in that pixel that is covered by geometries touching the pixel. The user must specify a filename to generate a PNG file, which reflects the coverage as a bitmap. The image in Figure 1 is generated by the following command:

```
Command: DENSITY IMAGE 10, 10 /out/test10.png
```

Here, pixel size is set to 10x10 microns so each pixel in the test10.png output file corresponds to an area of 10x10 microns of the layout file.

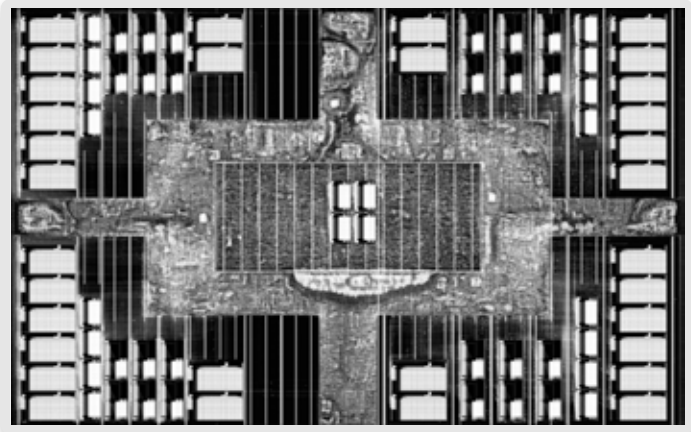

Figure 1. PNG Output Example
Portions of the Leon3 VHDL Model have been reproduced by Synopsys, Inc. with the permission of Aeroflex Gaisler AB. COPYRIGHT © 2010-2012, Aeroflex Gaisler. All Rights Reserved.

Quality of Results Improvements

As is often the case, there was an initial trade-off of coverage accuracy (deviation of the results between AREA and DENSITY commands) for the tremendous performance benefits of DENSITY. However, several enhancement options added to DENSITY have minimized the coverage accuracy differences between the two commands to the extent that there is no longer any practical difference. To achieve this, for jobdecks, a density image is calculated for each pattern file and a "blending" operation is used to overlay it on top of the output file representing the density image for the entire jobdeck. The pixel size of the density image of the pattern file is half the size of the specified pixel size. The user can control this ratio using the *RATIO_EXPONENT* <*value*> keyword, where <*value*> is an integer number representing the exponent of the power of 2.

```
Command: Density IMAGE 100,100 example.png
RATIO_EXPONENT 5
```

Here, the pattern file's density image will be created with a pixel size that is 32 (2^5) times smaller than the final image's pixel size. So, the larger the specified exponent value, the better the precision. However, the computation time and memory footprint will increase to some degree.

By default, the DENSITY command adjusts the pixel size of the output image in case the pattern width/height is not a multiple of the grid dimension. This adjustment impacts the accuracy and makes it difficult to compare the AREA and DENSITY results. To prevent this adjustment, the FIXED keyword can be specified. In such cases, the pixel size will not be adjusted. However, there might be a small blank area on the right and top side.

```
Command: Density IMAGE 100,100 example.png FIXED
```

Achieving Better Performance

Performance of the density calculation depends on multiple factors: storage speed, network connectivity, and whether it is a hot/cold file (meaning, whether the file was previously accessed).

The DENSITY command shows very good performance with MEBES files. There are plans to improve the performance for CFLT, CREF, and OASIS.MASK formats as well. But even with the current implementation, the user can achieve better turnaround time (TAT) using multithreading (MT) or distributed processing (DP). A hybrid approach can also be used (DP+MT).

To enable MT mode, the user can specify the *THREADS* (<*number*>|AUTO) keyword. The number tells the application how many threads will be used for calculation, while AUTO will use all available cores of a system. If the *THREADS* option is not specified, the number of threads specified by the native THREADS command will be used instead. Consider:

```
Command: Density THREADS 4 IMAGE 100,100 ./
example.png
```

In the previous example, using the DENSITY command on CFLT file with four threads improves the performance almost two times, compared to when that file was not previously accessed. Executing the same command a second time shows almost 4x improvement in TAT.

To enable distributed processing, a DP hosts file must be specified. It is the same file used for distributed fracturing. An example of this type of file is shown below:

```
cats-lnx3 1 ./tmp
cats-lnx4 2 ./tmp
cats-lnx5 3 ./tmp
cats-lnx6 4 ./tmp
```

To use DENSITY in DP mode, the DP keyword must be added, as illustrated in the following example:

```
Command: Distribute HOSTS ./hostfile
Command: Density DP IMAGE 100,100 ./example.png
```

To achieve better scalability, it is advised to avoid duplicate host name entries in the hosts file, and instead use MT to better utilize a particular host by adding the THREADS option to the command line. For example:

```
Command: Density DP IMAGE 100,100 ./example.png
THREADS AUTO
```

There is special handling implemented for OASIS.MASK files used in a distributed DENSITY calculation. Correspondingly, jobdeck files referencing OASIS.MASK files and multifile OASIS.MASK files can also benefit from it. With these formats, the application distributes portions of an OASIS.MASK file among workers. Each DP worker calculates its part of the DENSITY job (whether it is coverage or image) and when that part is ready, the DP Master stitches the result into the final output. The user can control the size of the task (by default, it is set to 10MB) using the TASKSIZE option.

In case a jobdeck references other format files (such as MEBES), the entire pattern file will be processed by a DP worker. Situations can arise in which an OASIS.MASK file size is smaller than the specified task size – in that case, the file will also be processed by the DP worker as a single piece. The following example shows that the task size is set to 20MB:

```
Command: Distribute HOSTS ./hostfile
Command: Density DP IMAGE 100, 100 example.png
TASKSIZE 20
```

Additional Options

It is quite possible that the jobdeck files can reference patterns that are missing. To account for this, the application checks the corresponding option specified by the JD MISSING command during the DENSITY calculation. The possible options for this command include ABORT, WARN, IGNORE, and VOID.

When the ABORT option is specified, the density calculation stops in case of missing patterns. Otherwise, with the WARN or IGNORE options, the name of the missing pattern appears, and the calculation continues to completion. For example:

```
Command: Jd MISSING ABORT
Command: Density IMAGE 100,100 ./example.png

Output message: DENSITY aborted. Pattern
<pattern name> file does not exist, use JD
MISSING IGNORE if you want to continue
```

There are other options provided by the DENSITY command, such as those that generate a CSV file, instead of an image, or those that calculate only a portion of layout file. All these options are documented in more details in the CATS Command Manual.

Conclusion

The DENSITY command allows users to calculate the coverage of layout files both in single-host and distributed environments, utilizing all CPU cores. Any file type supported by the application can be used for calculations. While this represents a major performance improvement over the AREA command, even further improvements are planned for the near future. ■

Control Job Prioritization and Maximize Utilization of CATS Distributed Processing Licenses

Gary Meyers, CATS CAE

For some time, CATS® has had the ability to dynamically adjust resources during a distributed fracture. The feature was released in CATS Version E-2011.03-SP4. This article is both a reminder and a short tutorial of that capability.

The typical usage occurs when a fracturing cluster is fully loaded and a job with a higher priority arrives. A job already executing can have either more resources added to speed up the fracture and free up the resources sooner, or can have resources removed to be utilized by the higher priority job. Either of these scenarios allows the running jobs to complete without having to kill a lower priority job to free up resources for the higher priority job. Killing an existing job loses all work that has been completed by the job until that point.

Syntax and Usage

The key idea is to enable the DISTRIBUTE UPDATE_HOST command as your default. The command syntax is DISTRIBUTE UPDATE_HOST <1/0> with the '1' value enabling it. There are no downsides to enabling the command by default. The command must be enabled at the start of a distributed fracture if you want the flexibility to add or remove fracture clients or, in the case of a VSB12 fracture, adding or removing stitchers from an active fracture.

When the UPDATE_HOST parameter has been enabled, an additional directory called 'host' is created in the associated Job* directory.

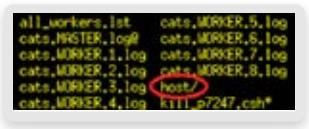

Figure 1. Host Directory

Once the job starts running, the Master process monitors the 'host' directory for the presence of a file called 'host.update'. The contents of the host.update file are the same as that of the original file but with either additional entries or fewer entries depending on the goal to 'throttle up' or 'throttle down' the distributed fracture. Shortly after the host.update file is copied into the host directory, the Master will read it. If the read is successful, the filename is appended with a '.used. <time stamp> suffix.

Example: **host.update.used.10:55:54**

If there is a problem reading the host.update file, it will have an 'err. <time stamp>' suffix appended to it. Typically, this happens when there is a syntax issue within the update.host file.

Example: **host.update.err.11:33:45**

If this occurs, review the host.update file for the syntax error. After the host.update file has been corrected, copy the updated host.update to the /Job*/host/ directory again and monitor the host directory to verify it has been read successfully.

Shortly after the host.update file has been successfully read, you should be able to observe the resource changes via the CATS dpmanager utility. In the following image, the number of workers is indicated by the number of INPROGRESS tasks in the Status column.

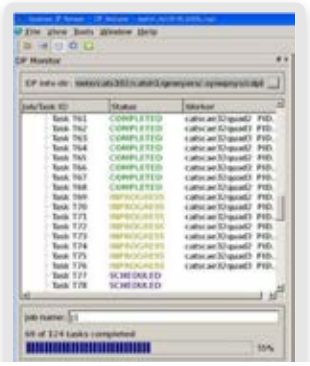

Figure 2. Status column

You can also view the status of workers via the master log file in the dpmanager. In the image below, the DP job started with eight workers. Four workers were removed and later, the four workers were added back in.

Figure 3. Gantt Chart of workers.

You can also use the Linux 'top' command or the resource monitor of your choice to monitor the total number of active workers.

For any additional questions please contact CATS support at 650-584-2600 or via email at cats@synopsys.com.

Qualify Fractured Output by Rendering, Finding and Extracting Small Figures (Slivers)

John Valadez, CATS CAE

Today's aggressively OPCed designs can lead to issues for VSB (Variable Shaped Beam) mask writing tools. Small figures known as slivers are created when the design is converted to the writer tool format. After data conversion, it is important to know not only where these slivers are located, but also to be able to gather statistics on them in order to verify, analyze, and qualify the converted output file. Whether the fractured output is checked visually, or through in-house automation, CATS® provides users with three different methods to be used in any flow:

1. Distinguishing slivers from other figures by drawing or rendering in contrasting colors
2. Finding slivers, with step-through functionality, as well as exporting figure information to common text formats like XML and CSV
3. Extracting slivers to output formats GDSII or OASIS
4. Setting Filtering Options

The first thing the user must do is set up the proper layout options. This can be accomplished using the CATS graphical interface, or through the command line by setting the appropriate TCL values. Using the graphical interface to do this is very simple. Choose **Tools > Options** on the main CATS menu, or use the **Alt+TO** hotkeys. After the Options dialog box appears, click on **Layout** on the left side (see Figure 1).

Figure 1. Defining Layout Filtering Options

The **Small shapes** checkboxes activate this functionality. The size (in nanometers) of the small shapes is entered in each of the respective textboxes next to each checkbox. This value represents the smallest of either the height or width of the shape. In Figure 1, each value is set accordingly:

- Find shapes smaller than 80nm
- Extract shapes smaller than 80nm
- Draw shapes smaller than 80nm

Note the drawing functionality allows the user to choose a color, fill pattern, and line width. The palette choices are the same as they are for displaying input and output files.

If TCL is used, the user can set the following variables at the command line to match the settings in Figure 1.

```
Command: tcl "default findFilter on"
Command: tcl "default findFilterSize 80"
Command: tcl "default extractFilter on"
Command: tcl "default extractFilterSize 80"
Command: tcl "default renderingFilter on"
Command: tcl "default renderingFilterSize 80"
Command: tcl "default renderingFilterFillColor
     #ff0000"
Command: tcl "default renderingFilterFillPattern
     fill50"
Command: tcl "default renderingFilterLineWidth 3"
Command: tcl "default renderingFilterOutlineColor
     #ff0000"
```

Drawing or Rendering Small Shapes

With the settings in place for rendering, in this case smaller than 80 nanometers, the user will be able see all small shapes drawn with the chosen color and fill. Figure 2 shows a fractured OPC design where sliver rendering is not activated, while Figure 3 shows the exact same window with slivers rendered in red.

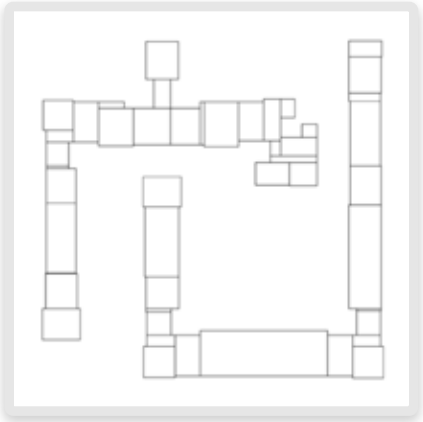

Figure 2. Small Shape Rendering Off

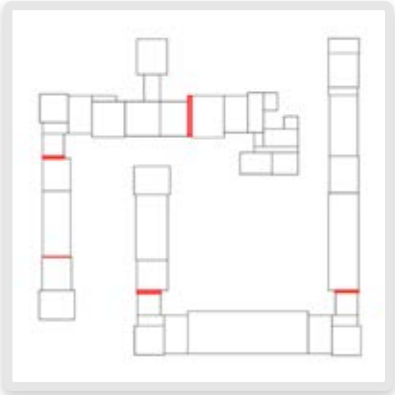

Figure 3. Small Shape Rendering On

The small shape rendering method is best suited for users who want to perform quick visual inspections or for saving screen captures of critical areas.

Finding Small Shapes

Using the same fractured pattern in Figure 2, a user can explore and step through each small shape with the Find utility. With the settings activated and in place for finding small shapes, the Find utility provides the user with specialized functionality just for shapes. This includes Circles, Polygons, XTraps, YTraps, Rectangles, and Paths, and because the shapes are filtered, only the small shapes are reported.

Before running the Find utility, the user must enable the Find Information view. This is done by choosing **View > Views > Find Information** (see Figure 4) from the main CATS menu.

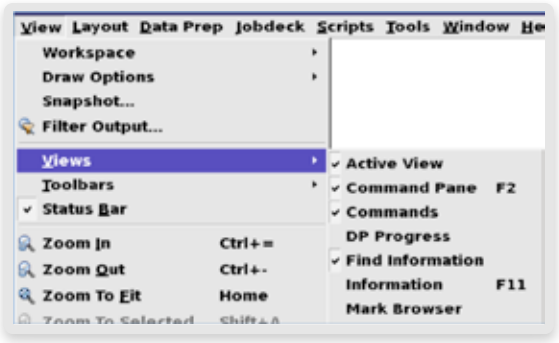

Figure 4. Displaying the Find Information View

To run the Find utility itself, choose **Edit > Find > Settings** from the main CATS menu, or use the **Ctrl+F** hotkey. The **Find Settings** dialog appears. Simply select **Shapes** as shown in Figure 5 below.

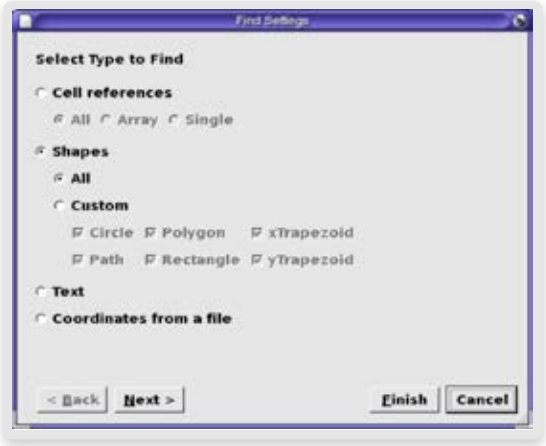

Figure 5. Find Settings

Clicking the **Next** button once will allow for layer selection. Clicking the **Next** button twice will allow for a region to be specified. The latter is highly recommended because the input file might contain a very large number of slivers. Alternatively, a **Find limit** can be specified by choosing **Tools > Options > General**, and specifying it under the **Maximum number of items to find** option, or by setting the following TCL variable:

```
Command: tcl "default find_limit <# value>"
```

The final step is to click the **Finish** button in the lower right. Afterward, the **Find Information** view contains the small shape information in a browsable table format (Figure 6). Each row contains the shape's location as well as other information. When a row is clicked, the window displays a zoomed in view of the small shape.

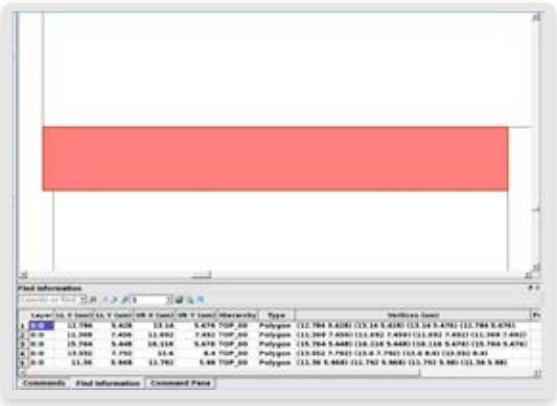

Figure 6. Find Information View

By clicking the small disk icon, the user can save the information presented in the table to a separate Text, XML, or CSV file.

For more information on the Find utility, refer to the CATS User Guide.

Extracting Small Shapes

Finally, the user can extract the small shapes into a GDSII or OASIS file simply by using the Layout Export utility. Exporting can be accomplished by using the more common method of choosing **File > Export** from the main CATS menu, or by using the **Shift+X** hotkey. Another method is by drawing a highlight around an area of interest, then using the right button mouse and clicking on the selected highlight in the Active View. After calling the Export utility, the **Layout Export** dialog appears (Figure 7).

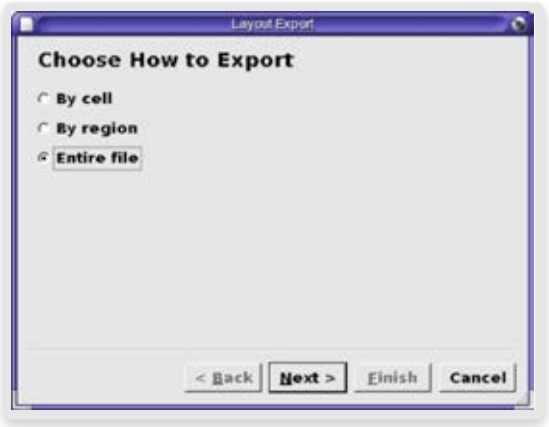

Figure 7. Defining the xxport method

Because the Small shape filtering is activated, the output library file contains only the small shapes that meet the criteria. Again, these are shapes smaller than 80nm as specified earlier. In Figure 8, the output is GDSII format and the output file name is TOP_00_slivers.gds.

Figure 8. Exporting the file

After clicking **Finish**, the output file is created, and the user will now have a file that contains only small shapes (Figure 9).

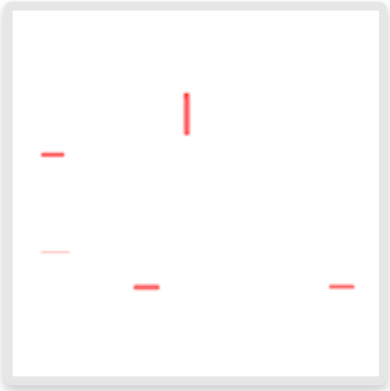

Figure 9. Small shapes in output

This file can be used to overlay with the original fractured output file for other graphical checks, or both files can be input into the application to perform multilayer operations using MRC. For example, MRC can isolate CDs and determine whether a sliver is present within the CD region. Here is a brief example CINC file with the basic steps necessary to do this.

```
Format FLAT
Function BSPACE
Input ./original.pattern
Switch
Input ./patternslivers.gds
Switch
! MRC Parameters
MRC INTERNAL 1 ON
MRC INTERNAL 1 INPUT P:0
MRC INTERNAL 1 LAYER 100
! Define CD Region Here
MRC INTERNAL 1 LONGEDGE > 1
MRC INTERNAL 1 LONGEDGE_TO_EDGE = 0.283 0.285
! Select Slivers Interacting with CD Region
MRC SELECT INTERACT 2 ON
MRC SELECT INTERACT 2 INPUT S:0 O:100
MRC SELECT INTERACT 2 LAYER 101
Output $TED/interacting_slivers.cflt
```

The flow is illustrated in Figure 10 below.

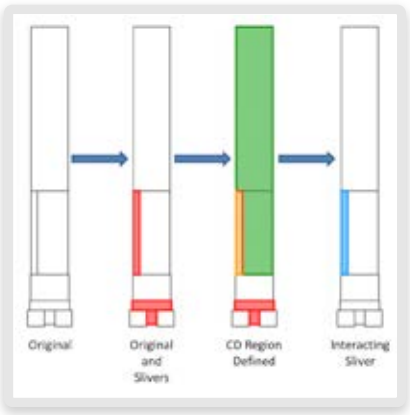

Figure 10. MRC Flow

Summary

Mask quality is extremely important, and slivers can have a significant impact. With advancing technology nodes, this becomes increasingly relevant. CATS users have the tools at their disposal to perform advanced mask checks and to fully analyze the effects these small shapes have in determining quality of a fractured design. ■

The Rise of OASIS.MASK:

John Burns, CATS CAE

OASIS.MASK is gradually replacing MODE5 as a convenient interchange format between companies as well as between tools. Synopsys, in conjunction with other companies, wants to ensure strict adherence to the format unlike what happened with GDSII. OASISMASK_VALIDATOR was created for this purpose.

MODE5 Still a Popular Choice

Before discussing OASIS.MASK, it should be noted that MODE5 usage is still very common among many CATS customers, and there are machines that consume MODE5 for both mask writing and inspection. Meanwhile, OASIS.MASK has been used in some inspection systems for the past few years, and is finally breaking through into the data path for certain mask writers.

That being said, MODE5, with its more limited format that restricts stripe and segment size, can lead to comparatively large output files. If precision demands a nanometer or smaller resolution, generating these fixed stripes costs time and leads to trouble in transferring, verifying, viewing, and ultimately using such files to expose masks.

The user should compare the data produced as MODE5 and OASIS.MASK to see the significant reduction in file sizes in the case of OASIS.MASK.

In order to generate OASIS.MASK output files in the application, the commands would resemble:

```
Format FLAT
Direct_Output OASIS
Resolution .0005
Height 500
Width 500
Output $TED/TEST_FILE_WITH_WHATEVER_NAME.oas
```

The user may or may not need such a fine resolution, but it is important to understand that as the precision becomes smaller and smaller, the difficulty in outputting MODE5 not only increases, but also, can become impossible.

For example, an error similar to the following might occur:

```
Error: Cannot write 19725 segments to Mode5
      (limit 10000)
```

Regardless of the input data and the required resolution, users should see file size savings when they generate OASIS.MASK output. Here are some typical examples:

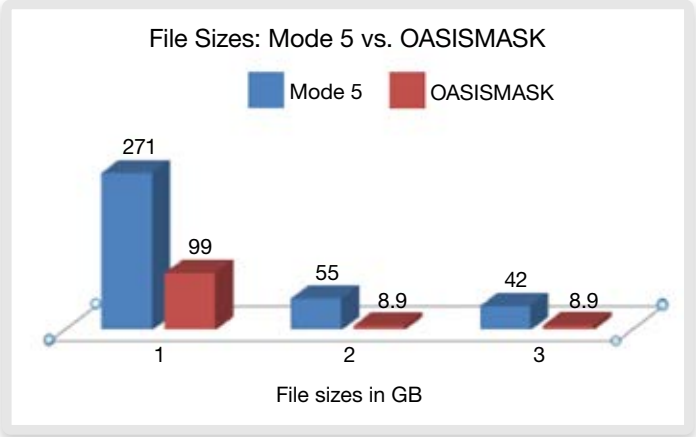

Figure 1. MODE5 versus OASISMASK sizes (using 0.5 nm resolution)

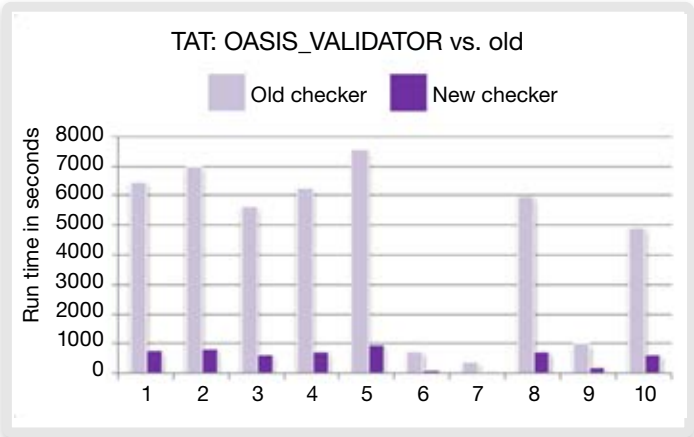

Figure 2. OASISMASK_VALIDATOR running with all threads on a 12 CPU box versus CHECKP44OAS

While these smaller output file sizes offer huge benefits when transferring files, especially across slower networks, there are additional benefits for improved performance in viewing and processing the data. That is because the nature of the OASIS.MASK format allows the application to input and draw the data without requiring a cache file.

For example, in the past, some users might have used ETEC's proprietary CHECKPAT executable on MODE5 files. Tests run with CHECKPAT on the smallest file above took approximately an hour to complete. Running OASISMASK_VALIDATOR takes about a quarter of the time, which is a significant savings.

However, when OASISMASK_VALIDATOR is run at full speed utilizing all the 8 CPUs the box has, the time then dropped to under **three** minutes.

The Need for Speed

Adding a threaded capability to this new checker was seen as an important capability. Previously, the application had an in-house checker (CHECKP44OAS) that would use only a single CPU. The goal here was to create a checker that was more robust, and one that checked far more than its in-house predecessor, while making it run as quickly as possible.

The formatting of OASIS.MASK files, which in nearly all cases involves two levels of hierarchy and has cells arranged like a checkerboard with clear indices into those cells, was ideal for speeding up.

The input file sizes ranged from 6.8 GB to 128 GB. All OASIS.MASK files had compression enabled. More importantly, the longest run took just over 15 minutes. OASISMASK_VALIDATOR was able to run all 10 test cases in the time that CHECKP44OAS took to run the largest case.

Why Run the OASISMASK_VALIDATOR Tool?

It is always a good idea to run OASISMASK_VALIDATOR on files that have been FTPed or copied over to ensure the integrity of the file and to check the adherence to the OASIS.MASK standard. The time spent running the validator prior to viewing or refracturing of these files can save potential trouble down the line. Viewing and Fracturing applications check for valid OASIS.MASK when they read the input, but the checking will not be as thorough and the error reporting will not be as helpful as what you would get by running OASISMASK_VALIDATOR.

Some customers using MODE5 files as an interchange format, sending them between different groups or companies, are already performing a checksum on the files before and after. OASISMASK_VALIDATOR performs a simple checksum in addition to the extensive checks.

Finally, the importance of keeping the OASIS.MASK standard precise and uncompromised has been a major concern voiced by numerous customers. Over time, the GDSII format developed a great degree of variation in how its specification was interpreted and implemented. Those variations end up costing time for interpretation, clarification, and significant amount money, when bad masks are made.

Conclusion

Using OASISMASK_VALIDATOR ensures that the OASIS.MASK files adhere to the standard and avoid variations of the format. OASISMASK_VALIDATOR ships free as part of the CATS package starting with CATS Version J-2014.06-SP4.

Using SQL to Query CATS MRCDB Files

John Valadez, CATS CAE

During an MRC process, if OUTPUT VERBOSE is set to TRUE, CATS® will create and store errors in an MRCDB file. This file is actually an SQLite database. Other Synopsys products, such as PLRC and ICV, also write and store data in a database. SQLite databases are used for many applications, such as specialized apps on smartphones, internet store fronts, and web browsers. Starting in CATS Version J-2014.06-SP5, the application will include the SQLite TCL library with each installation to allow access to the MRCDB files in order to run custom queries and extract desired information. The functionality the library provides is accessible only via a TCL script or macro. This article explains the basic steps necessary to accomplish this.

MRC Example to Create the MRCDB file

The first step to creating an MRCDB file is to run MRC. This example uses the grp7v6.gds input file, which is shipped with CATS® and located in the "demo" subdirectory where the application is installed (represented by the $CATS_HOME environment variable). The sequence of commands for this example is:

```
Clear
Format FLAT
Compact NO
Resolution 0.001
Height 100
Width 100
Input $CATS_HOME/demo/grp7v6.gds
Structure vert_cell
Layers -
Limits (INPUT 15,15) (INPUT 245,302.1)
! MRC Parameters
MRC INTERNAL 1 ON
MRC INTERNAL 1 LAYER 1
MRC INTERNAL 1 LONGEDGE > 2, < 10
MRC INTERNAL 1 LONGEDGE_TO_EDGE < 1
MRC OUTPUT VERBOSE TRUE
MRC OUTPUT MAX_ERRORS ALL
Output ./example1_out.cflt
DO
```

After executing the commands, the following output appears in the Commands pane:

```
Output size: 230,287.1
100.00% at 10:10 00.01e 00.00r 3387f 22.8kb 67%c
Processing:1 seconds Rectangles:69 Trapezoids:0
Total:616
Array Rects:547 Array Traps:0 Grand Total:3422
Saved: 67%
total MRC violations: 3414
```

As a result, the example1_out.MRC_ERRORS.db MRCDB file will be created. This file will be used as input for the following example TCL script.

Example Script to Load SQLite TCL Library, Execute Basic SQL Statements, and Write to Report File

This TCL script will query the previously created MRCDB file for four values:

1. The total number of violations found
2. The resolution used to create the MRCDB file
3. The input file used to create the MRCDB file
4. The Top 10 minimum widths of the violations found and their totals

The script will also print out each of the values to an example text report file. The full script is given below. Simply copy and paste it into a new TCL file called "create_report.tcl".

```
##########################################
## Demo script to query MRCDB file for the top
10 minimum ##
## violation widths and their total counts, and
write out ##
## everything to an example MRC report text file   ##
##########################################

# Load the sqlite3 TCL package
package require sqlite3

# Open MRCDB database
sqlite3 mymrcdb ./example1_out.MRC_ERRORS.db

# Query MRCDB for total overall count
set total_vio_count [mymrcdb eval {SELECT
flatcount FROM info}]

# Query MRCDB for resolution and input file used
set dbu [mymrcdb eval {SELECT DBUPerMicron FROM info}]
set primary_input [mymrcdb eval {SELECT
PrimInputFile FROM info}]

# Query top 10 smallest violation widths and
location counts
set mydata [mymrcdb eval {SELECT distance,
sum(LocationCount) \
FROM violations where layer=1 AND distance in
(SELECT DISTINCT \
distance FROM violations ORDER BY distance ASC
LIMIT 10) group by distance}]

# Open report file for writing and print header
set outfile [open myreport.txt w]
```

```
puts $outfile "MRC Error Report"
puts $outfile "----------------\n"
puts $outfile "Input File: $primary_input"
puts $outfile "Total Violations: $total_vio_count\n"
puts $outfile "Top 10 Smallest Violations"
puts $outfile "--------------------------"
puts $outfile "Rank\tWidth\tCount"

# Initialize rank number
set vcount 1

# Loop through violations and counts and print
to report
foreach {width count} $mydata {
 puts $outfile "$vcount\t[expr {$width * $dbu}]\
t$count"
 incr vcount
}

# Close report and MRCDB file
close $outfile
mymrcdb close

##########################################
## End Script                           ##
##########################################
```

Executing the TCL script

The demo TCL script can be executed at the command line or by using the CATS graphical interface. For example, after running MRC using the commands that created the MRCDB file, a user could enter the following at the command line:

```
tcl source create_report.tcl
```

Alternatively, a user could choose **Scripts > Run** from the main menu and select the “create_report.tcl” file as shown in Figure 1.

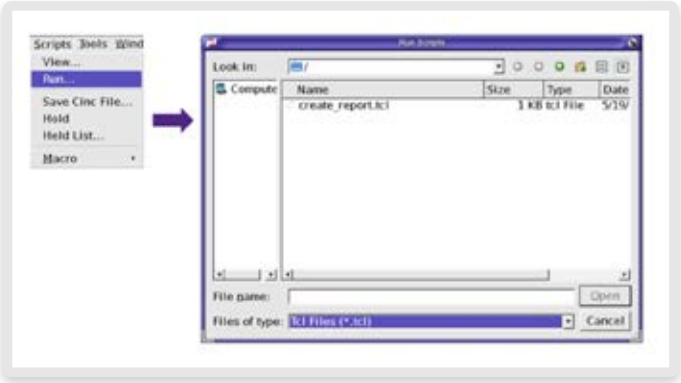

Figure 1. Running Scripts

For more information on running TCL scripts or macros, refer to the CATS User Guide.

After execution, a text file named myreport.txt will be created with the contents as shown:

```
MRC Error Report
----------------

Input File: grp7v6.gds
Total Violations: 3414

Top 10 Smallest Violations
--------------------------
Rank  Width  Count
1     0.14   708
2     0.15   676
3     0.16   647
4     0.17   64
5     0.18   591
6     0.2    71
7     0.24   63
8     0.28   39
9     0.29   16
10    0.3    250
```

Summary

By providing the SQLite TCL bindings, users can create custom reports and histograms, and perform other forms of statistical processing from the violations found simply by running queries to extract information. Users can even create their own databases to store information of their choosing, such as fracture settings and processing time. Future newsletter articles will cover more advanced topics related to SQLIte, TCL, and CATS®.

More information on SQLite and the TCL library can be found at https://www.sqlite.org/tclsqlite.html. ■

How to Read a Barcode Pattern in a Jobdeck Using CATS® and ZBar

Seurien Chou, CATS CAE

In mask production environments, CATS users work with many layers on a daily basis. Although users typically have a system to match the layer data with its corresponding paperwork, this system may not always be one hundred percent foolproof. Errors could occur when users have to manually process a layer (often using old data and old paperwork), or must deal with multiple similar data sets. Layers can become switched and a bad mask can be produced. The best practice to prevent such mix-ups is to match the barcode text in the jobdeck to the paperwork to ensure the right processing is being done to the right data. This reading of the barcode is typically a manual operation, and by definition, subject to human error.

This article describes a method of scripting the reading of a pattern barcode using CATS® and an open source barcode reader, ZBar. This would allow a user to automatically capture the barcode text in many popular types of bar codes such as Code 39 and QR Code and then compare this text to their work order. This could be fully automated and act as a last line of defense for ensuring that the work being done (such as with Booleans, sizing, and so forth) is correctly applied to the appropriate layer.

Figure 1. Example of QR code barcode

The idea behind the jobdeck barcode reader script is simple. The user specifies the location of the barcode inside the jobdeck using the coordinate system of the jobdeck. The application then reads the jobdeck at that location and generates temporary files in CATS flat format for all common orientations and for each of these in both negative and positive tones. The application then converts each of these files to a common image format such as PNG to be later read by a third-party barcode reader, such as ZBar. Each barcode image file is then read by the barcode reader. If a unique text is read, then that text is returned. If two or more different texts are read, or if none can be read, then an error is returned.

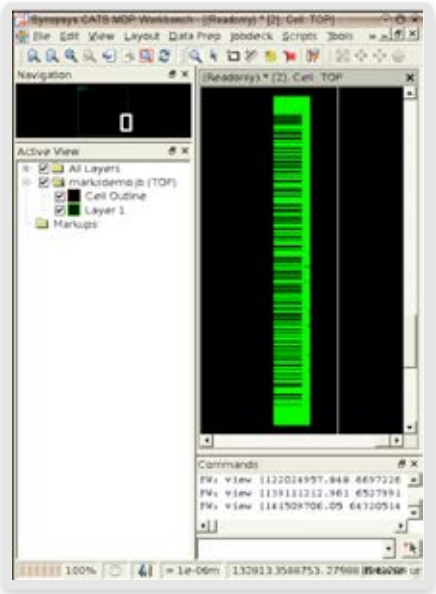

Figure 2. Example of CODE-39 barcode

Although the application does not support native reading of barcodes, the process is relatively simple and can be easily scripted after all the prerequisites are met:

1. The first requirement is to install a third party barcode reader. This example uses ZBar from Source forge (http://zbar.sourceforge.net/index.html) Ensure all dependencies for ZBar are also installed. If the user cannot find a pre-built binary for the current OS, it might have to be built from source code. ZBar does not need to be built with QT, x, pthread, video, python, or gtk. The recommended configuration line is shown in the following figure:

```
[15:27][~] $ ./configure --prefix=/tmp/zbar_0.10 --with-qt=no --with-x=no --enable-pthread=no --enab
le-video=no --with-python=no --with-gtk=no
```

Figure 3. ZBar ./configure arguments

It is important to use the latest version of ImageMagick, which enables ZBar to read image files. Many Linux systems come preinstalled with it, but make certain the version is 6.0 or higher. Version 6.2.8 has been proven to be successful. To find out the version of ImageMagick, use the "convert –version" command.

```
cats G-2012.09-SP18 > convert -version
Version: ImageMagick 6.2.8 10/06/10 Q16 file:/usr/share/ImageMagick-6.2.8/doc/index.html
Copyright: Copyright (C) 1999-2006 ImageMagick Studio LLC
```

Figure 4. ImageMagick version

The only binary used by the example script below is zbarimg, which is part of ZBar. (Note that ZBar is licensed under the GNU LGPL 2.1 to enable development of both open source and commercial projects).

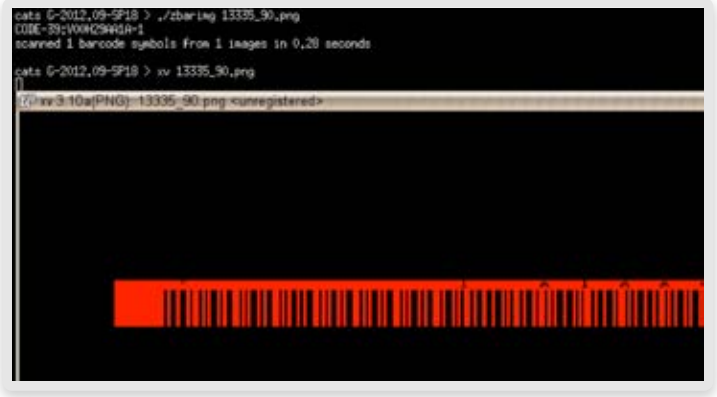

Figure 5. Results of zbarimg

2. The second requirement is to have CATS Version G-2012.09 or higher installed. Please ensure the required Synopsys licenses to read the desired type of jobdeck and pattern files are available. Any supported jobdeck format and pattern file can be read as input.
3. The final requirement is to have X virtual frame buffer, Xvfb, on the system. Xvfb comes preinstalled on many systems. However, if it is not, the user may have to work with their system support group to get it installed. Xvfb implements the X11 display server protocol, which performs all graphical operations in memory without displaying them on the screen. This is required by the application to ensure the captured barcode is of high enough fidelity to be read by ZBar. In effect, a large screen is defined so each identifying feature of the barcode (such as a skinny bar or a pixel) is distinguishable in a single screen capture. The example script defines a virtual screen that is 2560x2048 pixels.

The following example shows a script that uses the application and ZBar to read a barcode from a jobdeck. Before running the script, the binaries for CATS®, Xvfb, and zbarimg must be installed and defined in the user's PATH environmental variable. This script is shown as an example but is not included with the CATS binaries. Please contact CATS support at cats@synopsys.com if you would like a copy.

```
Usage:
read_barcode.pl [jobdeck|file] [-l layer]
[Location of barcode capture four point window
in nanometers]
```

Consider the following two example runs. The first one uses a MEBES jobdeck as input, whereas the second one uses a MEBES pattern file as input.

Example 1: MEBES Jobdeck Input

Notice that the layer flag is set to 1, and the coordinates of the barcode inside the jobdeck are given in terms of the MEBES jobdeck coordinate space (where the origin is at the lower left). The last line of the output shows the text of the barcode. In this case, it is "V00H29AA1A-1". The string also contains the type of barcode that has been scanned. In this case, it is "CODE-39." Notice the script reporting back scanning lines. These are CATS fractures to different orientations, tones, and mirroring options. This allows checking for barcodes in all possible configurations automatically.

```
cats G-2012.09-SP18 > ./read_barcode.pl
marksdemo.jb -l 1 140844210.268 65308317.6256
150217599.011 15785580.4352

STATUS: Input file name: marksdemo.jb

STATUS: Coordinates llx,lly,urx,ury: 140844.210268
,65308.3176256,150217.599011,15785.5804352

STATUS: Starting child Xvfb on :51

STATUS: Running CATS on 13335_runme.cincp.
Please wait ...

rxusage: peakvsz=968.90MB peakrss=173.87MB
utime=14s stime=1s elapsed=21s cnt=218
FreeFontPath: FPE "unix/:7100" refcount is 2,
should be 1; fixing.

STATUS: Running zbarimg to scan png files. Please
wait ...

STATUS: Scanning 13335_main.png
 ZBARIMG: CODE-39:V00H29AA1A-1
STATUS: Scanning 13335_mainr.png
 ZBARIMG: Not scanned.
STATUS: Scanning 13335_90.png
 ZBARIMG: CODE-39:V00H29AA1A-1
STATUS: Scanning 13335_90r.png
 ZBARIMG: Not scanned.
STATUS: Scanning 13335_180.png
 ZBARIMG: CODE-39:V00H29AA1A-1
STATUS: Scanning 13335_180r.png
 ZBARIMG: Not scanned.
STATUS: Scanning 13335_270.png
 ZBARIMG: CODE-39:V00H29AA1A-1
STATUS: Scanning 13335_270r.png
 ZBARIMG: Not scanned.
STATUS: Scanning 13335_mirror0.png
 ZBARIMG: CODE-39:V00H29AA1A-1
STATUS: Scanning 13335_mirror0r.png
 ZBARIMG: Not scanned.
STATUS: Scanning 13335_mirror90.png
 ZBARIMG: CODE-39:V00H29AA1A-1
STATUS: Scanning 13335_mirror90r.png
 ZBARIMG: Not scanned.

BARCODE_TYPE: CODE-39

BARCODE_TEXT: V00H29AA1A-1
```

Note that when describing the syntax in more general terms, the variables are written in italics. For example:

```
Command: ALIGN_COORDINATES SECONDARY [x1, y1]
```

Example 2: Single MEBES Pattern File Input

Notice the coordinates are specified in pattern file coordinates.

```
cats G-2012.09-SP18 > ./read_barcode.pl V00H29A01.
AW 139696727.042 114053695.613 143188923.616
110488912.968

STATUS: Input file name: V00H29A01.AW

STATUS: Coordinates llx,lly,urx,ury: 139696.727042
,114053.695613,143188.923616,110488.912968

STATUS: Starting child Xvfb on :51

STATUS: Running CATS on 13000_runme.cincp.
Please wait ...

rxusage: peakvsz=954.70MB peakrss=170.16MB
utime=13s stime=2s elapsed=22s cnt=219
FreeFontPath: FPE "unix/:7100" refcount is 2,
should be 1; fixing.
```

```
STATUS: Running zbarimg to scan png files. Please
wait ...

 STATUS: Scanning 13000_main.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_mainr.png
  ZBARIMG: QR-Code:V00H29AA1A-1
 STATUS: Scanning 13000_90.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_90r.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_180.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_180r.png
  ZBARIMG: QR-Code:V00H29AA1A-1
 STATUS: Scanning 13000_270.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_270r.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_mirror0.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_mirror0r.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_mirror90.png
  ZBARIMG: Not scanned.
 STATUS: Scanning 13000_mirror90r.png
  ZBARIMG: Not scanned.

  BARCODE_TYPE: QR-Code

  BARCODE_TEXT: V00H29AA1A-1
```

This article described a method for automatically reading a barcode inside of a jobdeck. This can reduce human errors in the mask making flow.

This methodology can be coupled with CATS pattern matching capabilities to automatically search for barcodes within a pattern or a jobdeck, thus removing the need to specify the location in the flow suggested above. This extension would allow a user to specify only a jobdeck and layer, and the application would be able to return the corresponding barcode texts. ■

For questions about CATS please send email to cats@synopsys.com.

SYNOPSYS®
Silicon to Software™

Synopsys, Inc. • 690 East Middlefield Road • Mountain View, CA 94043 • www.synopsys.com

©2015 Synopsys, Inc. All rights reserved. Synopsys is a trademark of Synopsys, Inc. in the United States and other countries. A list of Synopsys trademarks is available at http://www.synopsys.com/copyright.html. All other names mentioned herein are trademarks or registered trademarks of their respective owners.
09/15.CE.CS6102.