Learn the OpenAccess API
Using Python

©Silicon Integration Initiative
Initial Contribution By

James Masters
Intel - 2013
Updates & Additions
by Silicon Integration Initiative - 2020
Section 7 - oaShape

- All shapes are placed directly in oaBlock (not oaDesign)
- Same API calls in creating shapes for layout, schematic, and symbol designs
  - Focus in this training will be on layout design
- Be sure to view the “inherited members” in online documentation as shapes have many inherited methods
Getting `oaLayer` and `oaPurpose`

- All `oaShape` objects have an `oaLayer` and `oaPurpose` defined
  - Both layer and purpose number are required to create a shape
  - Layer and purpose numbers are mapped to names through the `oaTech`
  - May be a good idea to look up the layer and/or purpose numbers prior to usage once and then use many times in code (as opposed to looking up constantly)
    - Consider using a hash to pre-load all layers – the code below can be extensive if repeated for many layers

```python
poly = oa.oaLayer.find(tech, "poly")
if poly:
    poly_num = poly.getNumber()
else:
    print "ERROR: cannot find layer 'poly' in tech!"
    sys.exit(1)

# Can use poly_num below anywhere the poly layer number
# is needed...
```
DBU/UU Conversion

• All shape coordinates must be in Database Units (DBU)
• The oaTech class has multiple functions to help in converting between user units (UU), like microns or your cursor coordinates, to DBU:
  – uuToDBU: Convert from user units (e.g. microns) to DBU
  – dbuToUU: Convert from DBU to user units (e.g. microns)
  – Both require a view type (e.g. maskLayout) to convert

```python
mlay = oa.oaViewType.get("maskLayout")
tech.uuToDBU(mlay, 0.250) #=> 250
tech.uuToDBU(mlay, 1.5) #=> 1500
```

• There are convenience methods in oaScript which will be shown later in this section to help in dealing with conversion to DBU
oaRect

- Rectangle dimensions come from the given oaBox

```python
# Get layer and purpose numbers
ml = oa.oaLayer.find(tech, "m1").getNumber()
drw = oa.oaPurpose.find(tech, "drawing").getNumber()

# Create 1.0um rectangle centered at (1.0, 2.0) [um]
# Placed on m1/drawing.
    width_2 = tech.uuToDBU(mlay, 0.5)
    ctr_pt = [tech.uuToDBU(mlay, 1.0), tech.uuToDBU(mlay, 2.0)]
    box = oa.oaBox(ctr_pt, width_2)
    rect = oa.oaRect.create(blk, ml, drw, box)
```
oaPath

• A path is two or more points with a width and optional begin/end styles (shown) and extension distances (not shown)

```python
# Get layer and purpose numbers
ml = oa.oaLayer.find(tech, "ml").getNumber()
drw = oa.oaPurpose.find(tech, "drawing").getNumber()
wid = tech.uuToDBU(mlay, 0.5)
pts = [ [tech.uuToDBU(mlay, 1.00), tech.uuToDBU(mlay, 1.00)],
       [tech.uuToDBU(mlay, 1.00), tech.uuToDBU(mlay, 1.65)],
       [tech.uuToDBU(mlay, 2.32), tech.uuToDBU(mlay, 1.65)] ]
path = oa.oaPath(blk, ml, drw, wid, pts, "extend")
```
oaText

• Most advanced design methodologies attach shapes to nets to establish connectivity; however, text labels are sometimes used in final verification.

• A text label is created using `oaText` which has the following attributes used most commonly:
  – Layer/purpose number
  – Text string of the text label
  – Origin (point) and text alignment
  – Font (nearly always “stick”)
  – Height (to control size of text)

```python
mlay = oa.oaViewType.get("maskLayout")
m1 = oa.oaLayer.find(tech, "m1").getNumber()
drw = oa.oaPurpose.find(tech, "drawing").getNumber()

text = oa.oaText.create(rblk, m1, drw, "A", [0,0],
oa.oacCenterCenterTextAlign, oa.oacR0,
oa.oacStickFont, tech.uuToDBU(mlay, 0.5))
```
Lab 7.1: Create Resistor Layout

• Goal - Become familiar with creating various oaShape objects and converting microns to DBU
• Write a script to:
  • Create resistor layout as shown on the next slide
    – Create “res” cell in “w” mode to overwrite any previous work (you will likely make several iterations before the code is perfected)
    – Try to get as far as you can; this lab is intended to be longer with a bigger emphasis on problem solving
      • Copy final library from the lab solutions if you run out of time
      – You will need to use the oaTech dbuToUU() function to convert from microns to the actual DBU in the design
    – **Don’t forget to save the design at the end of your code!**

Compare your script to labs/7.1/makeResistor.py
Resistor Design Rules (Lab 7.1)

- Resistor device origin (0,0) is in the middle of the bottom contact as shown by the red cross “+” below.
- Contact size is 0.10um square.
- Poly width is 0.25um.
- Poly overlap of contact 0.125um.
- Metal 1 overlap of contact 0.08um.
- Resistor ID for this resistor should cover the entire poly region between the two contact edges and extend for 2.00um as shown in the diagram.
- Terminals “A” (bottom) and “B” (top) should be texted as such with a text label touching the edge or inside of the metal landing pad.
- Note: Diagram is not drawn to scale.

The diagram illustrates the dimensions and layout of the resistor, including the contact size, poly and metal overlaps, and the location of terminals A and B.
oaScript DBU/UU Conversion Features

• OA Script comes with convenience classes to allow you to use user units (UU) instead of DBU
  – NOTE: these convenience methods have limited documentation and support (use with caution)
  – The tech object and view type are needed when constructing the oasUU* class objects (  
    o By itself, it doesn’t know the DBU/UU ratio)
    o These are put at the end of the constructor

• UU Classes
  – oasUUBox
  – oasUUBoxArray
  – oasUUPoint
  – oasUUPointArray
  – oasUUSegment
  – oasUUTransform
  – oasUUVector

• Example:
  mlay = oa.oaViewType.get("maskLayout")
  uubox = oa.oasUUBox(0, 0, 3.195, 9.264, tech, mlay)
  uubox.upperRight().y() #=> 9.264
oaShapes may be created with user units by using the `createUU` constructor.

- Tech and view type are not needed for `createUU` constructors since the tech and view type can be found from the block object.

- Coordinate points will be converted to DBU internally using the tech from the design where the shape is placed.

```python
# Get layer and purpose numbers
ml = oa.oaLayer.find(tech, "m1").getNumber()
drw = oa.oaPurpose.find(tech, "drawing").getNumber()

poly = oa.oaPolygon.createUU(block, ml, drw,

    \([-1.1, -2.4], [1.06, 2.54], [3.12, -4.88]\])
```
Most wires are now created using oaPathSeg objects
- An oaPathSeg has exactly two points and an oaSegStyle
- A series of oaPathSeg objects connected to each other creates a route

oaSegStyle contains the following attributes:
- Width, begin/end style, extension(s) of begin/end style
- Begin/end styles: truncate, extend, variable, chamfer, custom
- Recommend “variable” to extend to the next oaPathSeg to ensure smooth edges (see colors below for ss1 and ss2)
- Note – no oasUUSSegStyle yet (missing)

```python
ss1 = oa.oaSegStyle(200, 'trtruncate', 'variable', 0, 225)
ps1 = oa.oaPathSeg.createUU(block, m1, drw, [0.00, 0.00], [2.34, 0.00], ss1)
ss2 = oa.oaSegStyle(500, 'variable', 'trtruncate', 100, 0)
ps1 = oa.oaPathSeg.createUU(block, m1, drw, [2.34, 0.00], [2.34, 1.55], ss2)
```
Lab 7.2: UU Familiarization

- Goal - Familiarize yourself with the UU features

- Write a script to:
  - Use the UU helper features to draw a few shapes in a new design including: oaRect, oaPolygon, and oaPathSeg
    
    Place two path segments orthogonally connected to each other

    compare your script to labs/7.2/uushapes.py
Final Word on UU Conversion

- Constantly converting UU to DBU (and back) can be a pain
- The oaScript built-in UU to DBU auto-conversion functions are still beta and do not fully cover the API yet
- For the remainder of this training, we will use our own UU to DBU conversion functions

```
tech = oa.oaTech.open(lib)
mlay = oa.oaViewType.get("maskLayout")

def cd2dbu(coord):
    """Map a single coord (in UU) to DBU""
    return(tech.uuToDBU(mlay, coord))

def pt2dbu(point):
    """Map a single point as python list (in UU) to DBU""
    return([cd2dbu(point[0]),
            cd2dbu(point[1])])

def pts2dbu(points):
    """Map a python list of points (in UU) to DBU""
    return( [pt2dbu(point) for point in points] )
```
def dbu2cd(coord):
    """Map a single coord (in DBU) to UU""
    return(tech.dbuToUU(mlay, coord))

def dbu2pt(point):
    """Map a single point as python list (in DBU) to UU""
    return([dbu2cd(point[0]), dbu2cd(point[1])])

def dbu2pts(points):
    """Map a python list of points (in DBU) to UU""
    return([[dbu2pt(point) for point in points] )
Iterating through Shapes

- You can query objects within an oaBlock using the get*() functions
  - Review the list in your OA API documentation now
- Iterating shapes using oaBlock::getShapes()
  - Will retrieve all shape types on all layers
  - Can be inefficient if you are only looking for a shapes on a given layer
    - In this case use oaBlock::getLayerHeaders::getShapes() instead
  - You will need to determine the shape type before doing much with it
    - Is it a rectangle, path, polygon, etc?
      - **Option 1:** Find the shape’s type by checking the class in your language
        ```python
        for shape in block.getShapes():
            if shape._class == oa.oaRect:
                # do oaRect stuff in here...
        ```
      - **Option 2:** Find the shape’s type by calling OA’s shape.getType() method
        ```python
        for shape in block.getShapes():
            if int(shape.getType()) == oa.oacTextType:
                # do oaText stuff in here...
        ```
Iterating through LPP Headers

• Using `oaBlock::getLayerHeaders()` is an efficient way to quickly loop through shapes on a given layer without having to look at shapes on other layers
  – `oaLPPHeader` has `getLayer()` and `getPurpose()` methods

```cpp
for lpwh in block.getLPPHeaders():
    if (some LPP filter):
        for shape in lpwh.getShapes():
            # only iterates shapes on the selected LPP
```

• Can also get quick shape counts
• In the example below we’d want to make sure that `getLayer()` and `getPurpose()` return an object
  – if the tech doesn’t have them defined, these will return NULL (None in Python)

```cpp
for lpwh in block.getLPPHeaders():
    layer = lpwh.getLayer()
    purpose = lpwh.getPurpose()
    shapes = lpwh.getShapes()
    print "%d shapes on %s/%s" %(shapes.getCount(), layer.getName(), purpose.getName())
```
Lab 7.3: Iterate Through Shapes and LPP Headers

• Goal - Become familiar with iterating shapes in an oaBlock and understand how oaLPPHeaders can help efficiency

• Write a script to:
  – Open the “res” design from Lab 7.1 using read access
    des = oa.oaDesign.open("mylib","res","layout","r")
    blk = des.getTopBlock()
  – Iterate through the shapes in the block and print something out about the shapes
    blk.getShapes()
    Hint – you will first need to determine the shape type and then decide what to print about the shape
  – Use the oaLPPHeader to filter shape iteration down to the m1/drawing LPP and print the layer count
    blk.getLPPHeaders()

Compare your script to labs/7.3/get_shapes.py
Section 7 Summary

• Examined User Units (UU) and Database Units (DBU) to understand their use and application
• Examined layers and purposes more closely
• Learned techniques for iterating over objects within a block
• Discovered the oaLPPPHeader and how it improves your application’s efficiency
Silicon Integration Initiative

www.si2.org

For details contact Marshall Tiner
Director of Production Standards

mtiner@si2.org