EDA Scripting Unleashed: Real-Life Examples Using oaScript and oaxPop

James D. Masters
Intel Corp
Si2CON
October 6, 2015
Agenda

• oaScript overview
• oaxPop overview
• Intel’s experience with oaScript and oaxPop
  – Framework bundle of required packages
  – Performance observations
  – Density Calculator application
  – Methodology Checker application
• oaScript/oaxPop Roadmap
• Summary
oaScript overview

- Standalone direct interface to OpenAccess (OA) using Perl, Python, Ruby, or Tcl
  - Enables rapid development of powerful OA-based software
  - Performance and memory usage is mostly comparable to that of a C++ application (a few exceptions will be mentioned later)
- Matches OA C++ API very closely
  - Existing C++ API documentation can be referenced
  - Auto conversion of types between C++ and scripting language
- Includes convenience functions to reduce code and improve productivity
- Initiated in 2009, and refined over the past 6 years in Si2’s oaScript working group (code base is stable)
oaScript interaction with OA API (via SWIG)

- Uses the Simplified Wrapper and Interface Generator (SWIG) tool to expose C++ APIs to scripting languages
  - Common interface through SWIG ensures cross-language consistency and reuse
  - All languages interface OA through the official OA API
oaxPop overview

- Provides high-speed polygon manipulation capabilities in OA-based applications
  - Leverages the open-source Boost* **Boost.Polygon** high-speed polygon manipulation library contributed by Intel in 2008
  - Works directly with OA object types (oaShape, oaBox, oaPointArray, etc.)

- Work started with a proof-of-concept sample in 2011 and created a formal Si2 oaxPop WG in 2012
  - Added Unified Layer Model (ULM) high-level shape relationship operations
  - Added scripting interface (leveraging oaScript work) to enable in Perl, Python, Ruby, and Tcl

* Other names and brands may be claimed as the property of others.
**oaxPop interaction with OA API and Boost**

Diagram showing the interaction between **OA API**, **oalex**, **OpenAccess Extensions**, and **Boost.Polygon**. The diagram illustrates the flow of data and software components:

- **OA API** interacts with the **C++ Application**.
- The **C++ Application** imports **oalex** which interfaces with **oalex** script interface and **Boost.Polygon**.
- **OpenAccess Extensions** include languages like Perl, Python, Ruby, and Tcl, which interact with **oalex**.

**Boost.Polygon** is described as the core polygon operation engine.
oaxPop has more than boolean operations...

- Standard operators using +, |, *, &, -, ^
- The Unified Layer Model (ULM) operations are also included
  - Industry-standard description of relationships between polygons
  - Provides a rich set of functionality

- Area selection operators to find polygons meeting a criteria using <, <=, ==, >, >=
- Resize operations to resize all edges of a polygon (all types) or orthogonal directional-based resize: north, south, east, west (only on 90-degree shapes)
oaxPop performance

- oaxPop performance compared against other polygon manipulation libraries/software
  - Boost.Polygon outperforms other open-source libraries according to results published on the Boost.Polygon website
  - oaxPop FigSet90 performance is competitive with other professional polygon manipulation libraries/tools (see table)

- 50M shapes on “A”
- 50M shapes on “B”
- All flat – no hierarchy
- GDSII size was 6GB
- Only used single thread

<table>
<thead>
<tr>
<th>Tool</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>55s</td>
<td>47s</td>
<td>58s</td>
<td>45s</td>
</tr>
<tr>
<td></td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
<td>1x</td>
</tr>
<tr>
<td>oaxPop</td>
<td>29s</td>
<td>58s</td>
<td>93s</td>
<td>57s</td>
</tr>
<tr>
<td></td>
<td>0.53x</td>
<td>1.23x</td>
<td>1.60x</td>
<td>1.27x</td>
</tr>
<tr>
<td>Data1</td>
<td>35s</td>
<td>61s</td>
<td>113s</td>
<td>80s</td>
</tr>
<tr>
<td></td>
<td>0.63x</td>
<td>1.30x</td>
<td>1.95x</td>
<td>1.78x</td>
</tr>
<tr>
<td>Data2</td>
<td>109s</td>
<td>61s</td>
<td>204s</td>
<td>82s</td>
</tr>
<tr>
<td></td>
<td>1.98x</td>
<td>1.30x</td>
<td>3.52x</td>
<td>1.82x</td>
</tr>
</tbody>
</table>

Data1: 29s 0.53x 58s 1.23x 93s 1.60x 57s 1.27x
Data2: 35s 0.63x 61s 1.30x 113s 1.95x 80s 1.78x
Data3: 109s 1.98x 61s 1.30x 204s 3.52x 82s 1.82x

Note: Data* time/factor data are scrambled (sorted)
# oaxPop performance (basic + ULM ops)

<table>
<thead>
<tr>
<th># shapes</th>
<th>and</th>
<th>or</th>
<th>xor</th>
<th>sub</th>
<th>avoi</th>
<th>bO</th>
<th>encl</th>
<th>insi</th>
<th>inte</th>
<th>outs</th>
<th>over</th>
<th>stra</th>
<th>touc</th>
<th>butt</th>
<th>bOC</th>
<th>bOO</th>
<th>coin</th>
<th>cO</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.8</td>
<td>0.4</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>0.2</td>
<td>1.7</td>
<td>3.9</td>
<td>1.9</td>
<td>2.2</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>30000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.3</td>
<td>0.5</td>
<td>0.3</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>8.8</td>
<td>17.4</td>
<td>8.6</td>
<td>9.9</td>
<td>9.3</td>
<td></td>
</tr>
<tr>
<td>40000</td>
<td>0.1</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>0.4</td>
<td>0.7</td>
<td>0.4</td>
<td>0.1</td>
<td>0.3</td>
<td>0.4</td>
<td>0.4</td>
<td>0.5</td>
<td>25.6</td>
<td>52.9</td>
<td>26.6</td>
<td>27</td>
<td>26.3</td>
<td></td>
</tr>
<tr>
<td>50000</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.9</td>
<td>0.6</td>
<td>0.2</td>
<td>0.4</td>
<td>0.5</td>
<td>0.5</td>
<td>0.6</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>1</td>
<td>1.8</td>
<td>1.3</td>
<td>0.5</td>
<td>0.9</td>
<td>1.1</td>
<td>1.6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200000</td>
<td>0.4</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>2.4</td>
<td>3.8</td>
<td>3.1</td>
<td>1.3</td>
<td>2</td>
<td>2.5</td>
<td>2.3</td>
<td>4.5</td>
<td>2.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400000</td>
<td>0.9</td>
<td>0.8</td>
<td>0.9</td>
<td>0.8</td>
<td>5.9</td>
<td>7.7</td>
<td>7.7</td>
<td>4.1</td>
<td>4.3</td>
<td>6.7</td>
<td>5.4</td>
<td>13.2</td>
<td>5.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>800000</td>
<td>1.9</td>
<td>1.4</td>
<td>1.5</td>
<td>1.4</td>
<td>13.6</td>
<td>16.5</td>
<td>21.8</td>
<td>13.5</td>
<td>9</td>
<td>13.8</td>
<td>13.3</td>
<td>38</td>
<td>12.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*(randomized shape sizes and locations; time is in seconds)*

## Operations Key

- **and** = &
- **or** = |
- **xor** = ^
- **sub** = -
- **avoi** = avoiding
- **bO** = buttOnly
- **encl** = enclosing
- **insi** = inside
- **inte** = interacting
- **outs** = outside
- **over** = overlapping
- **stra** = straddling
- **touc** = touching
- **butt** = butting
- **bOC** = buttingOrCoincident
- **bOO** = buttingOrOverlapping
- **coin** = coincident
- **cO** = coincidentOnly
Intel’s experience with oaScript and oaxPop

- Deployed oaScript in small production flows since 2011
- Deployed both oaScript and oaxPop in larger production flows including deployment to design groups starting in 2014
  - All oaScript languages used to some extent throughout Intel, but Python has recently been gaining popularity
- Areas of opportunities for oaScript/oaxPop:
  - Intel-specific flow needs that may not be immediately found in or are outside the scope of EDA tool features
  - QA flows to check large amounts of OA-based data very quickly
  - OA data mining flows to get quick statistical answers
Bringing it all together: Intel’s “framework”

- Intel defines a “framework” for required components
  - Developers don't need to compile and install separate modules
  - Users receive a self-contained tested package
- Other modules are available for non-OA features
- Automation can be run standalone or from within a layout editor (using inter-process communication hooks)
- Access to the framework occurs through a “wrapper” which sets the needed environment (like an EDA tool)
Extending OpenAccess

• Intel added high-level functions directly in Python OA classes for common sequence of steps
  – Overrode Python object “dictionary” to give a feel of functions being directly in the object (instead of disconnected functions)
  – Use a convention of prefixing extended functions with “x_” to clearly separate OA functions from extended functions

• Extensions are built-into the framework to allow for easy access during development
Density Calculator: Quick in-editor solution for real-time density feedback

- All mask layers with full hierarchy depth supported
- Derived layer calculations available (not shown here)

Performance on medium-size block
- Hierarchy depth: 4
- Number of queried layers: 19
- Shape count: ~277k
- Runtime: ~1 second*

* With C++ helper class (details on next slide)
oaShapeQuery performance issues

```python
class MyShapeQuery(oa.oaShapeQuery):

    def __init__(self):
        oa.oaShapeQuery.__init__(self)
        self.figset = oaxPop.FigSet90()

    def queryShape(self, shape):
        if isinstance(shape, oa.oaRect) or
           isinstance(shape, oa.oaPolygon) or
           isinstance(shape, oa.oaPath) or
           isinstance(shape, oa.oaPathSeg):
            occ_shape = self.getOccShape(shape)
            hier_path = occ_shape.getHierPath()
            self.figset.append(shape, hier_path.getTransform())
```

Long runtimes using
oaScript-based
shape query
(function call overhead)

Biggest Time
Consumer
Created custom C++ based shape query

- Created small C++ class to perform shape query operations
  - Avoided python function call overhead for each shape
  - Abstracted inner-workings of oaShapeQuery to allow script author to only make high-level function calls
  - Added “caching” mechanism to avoid re-query in areas that were already visited
- Observed **400x~2000x improvement** in runtime versus the pure-python shape query

```python
squery = design.x_initShapeQuery()
squery.setRegion(bbox)
lpps = ("m1", "drawing"), ("m1", "fill")
polygons = squery.group(lpps).figset() ➔ Queried FigSet90
```
Methodology Checker: Quick check of design methodology compliance
- All mask layers with full hierarchy depth supported
- Many other Intel-specific checks exist (not shown here)

Use of oaxPop essential
- **DesignGrid**: err = wires - grid
- **StdCellRowGrid**: synthesize row grid; err = stdcell_prb - row_grid
- **LabelFloating**: 2x2 DBU label squares; err = lbls - lbls.touching(shps)

Error Navigator started in layout editor
oaScript/oaxPop scripting limitations

• Need to save design between layout editor and OA script
  – Mitigation can be achieved through auto-detection of unsaved edits which are saved and re-read in the OA script application

• Careful management of OA_PLUGIN_PATH and LD_LIBRARY_PATH for pcell and lib.defs plug-ins
  – Setup wrapper code to detect environment and set accordingly
  – There are still many “gotcha” scenarios to address

• No native DRC-checking features in oaxPop
  – No edge/vertex distance operations
  – Some workarounds exist, but require several layer operations and/or iterating over shapes in a layer set
Roadmap

2015 Si2CON 1.3 Release (October)
• Invalid shape detection (inverted box)
• Updates to documentation

2016 DAC Release (May 2014)
• Contribute shape query improvements

Beyond:
• Multi-threading and other performance improvements
• Vertex and edge-based operations
Summary

• oaScript and oaxPop are stable extensions available from Si2 (have had several years of development)
• Powerful OA-based applications can be written very quickly using oaScript and oaxPop
• Intel has successfully used oaScript and oaxPop in production applications
  – Applications are available either standalone or from an integration in a layout editor
  – Excellent performance observed in applications using oaScript and oaxPop (some adjustments required in certain cases)
References

**oaScript:**
- [https://www.si2.org/openeda.si2.org/projects/oascript/](https://www.si2.org/openeda.si2.org/projects/oascript/)

**oaxPop:**
- [https://www.si2.org/openeda.si2.org/projects/oaxpop/](https://www.si2.org/openeda.si2.org/projects/oaxpop/)