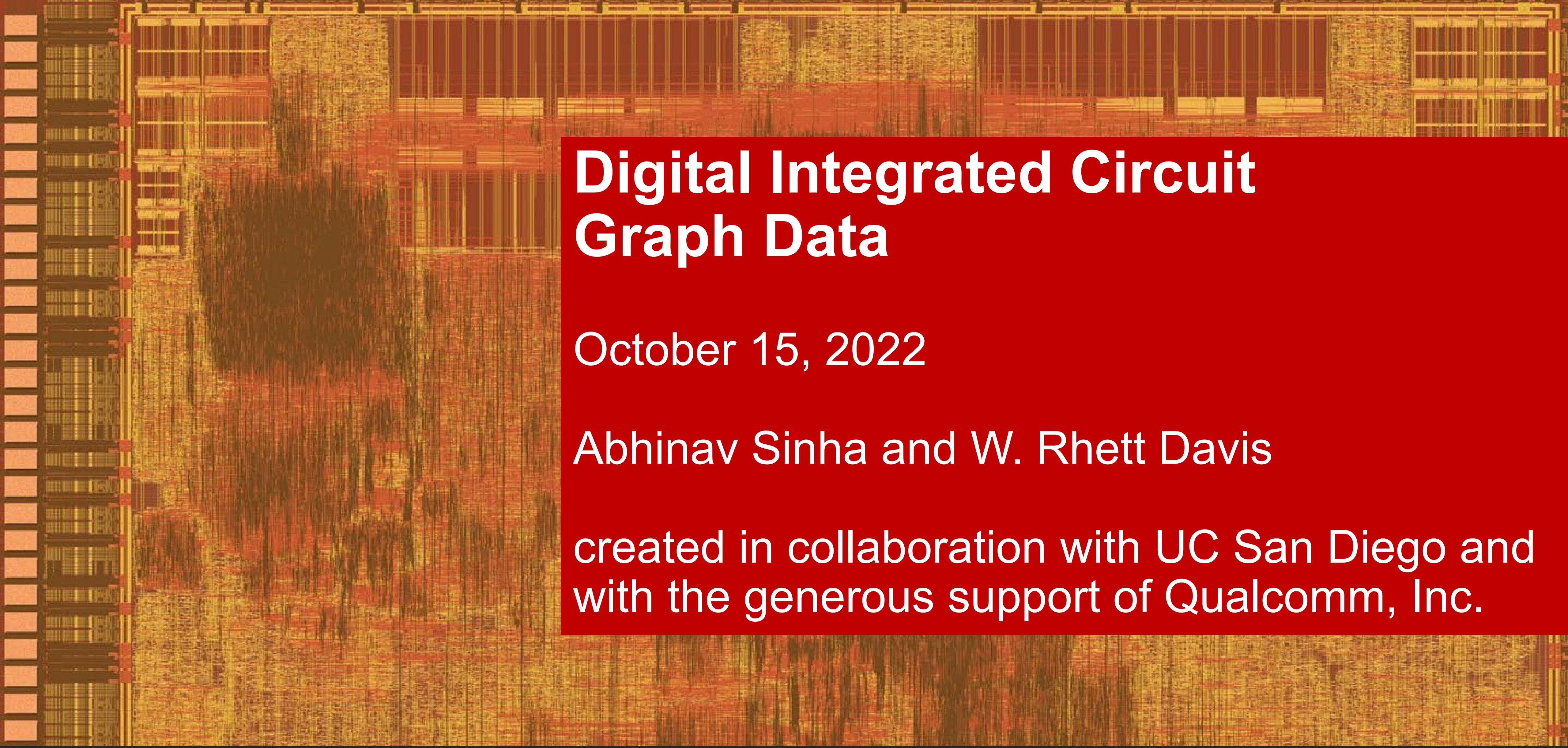


Digital Integrated Circuit Graph Data

October 15, 2022

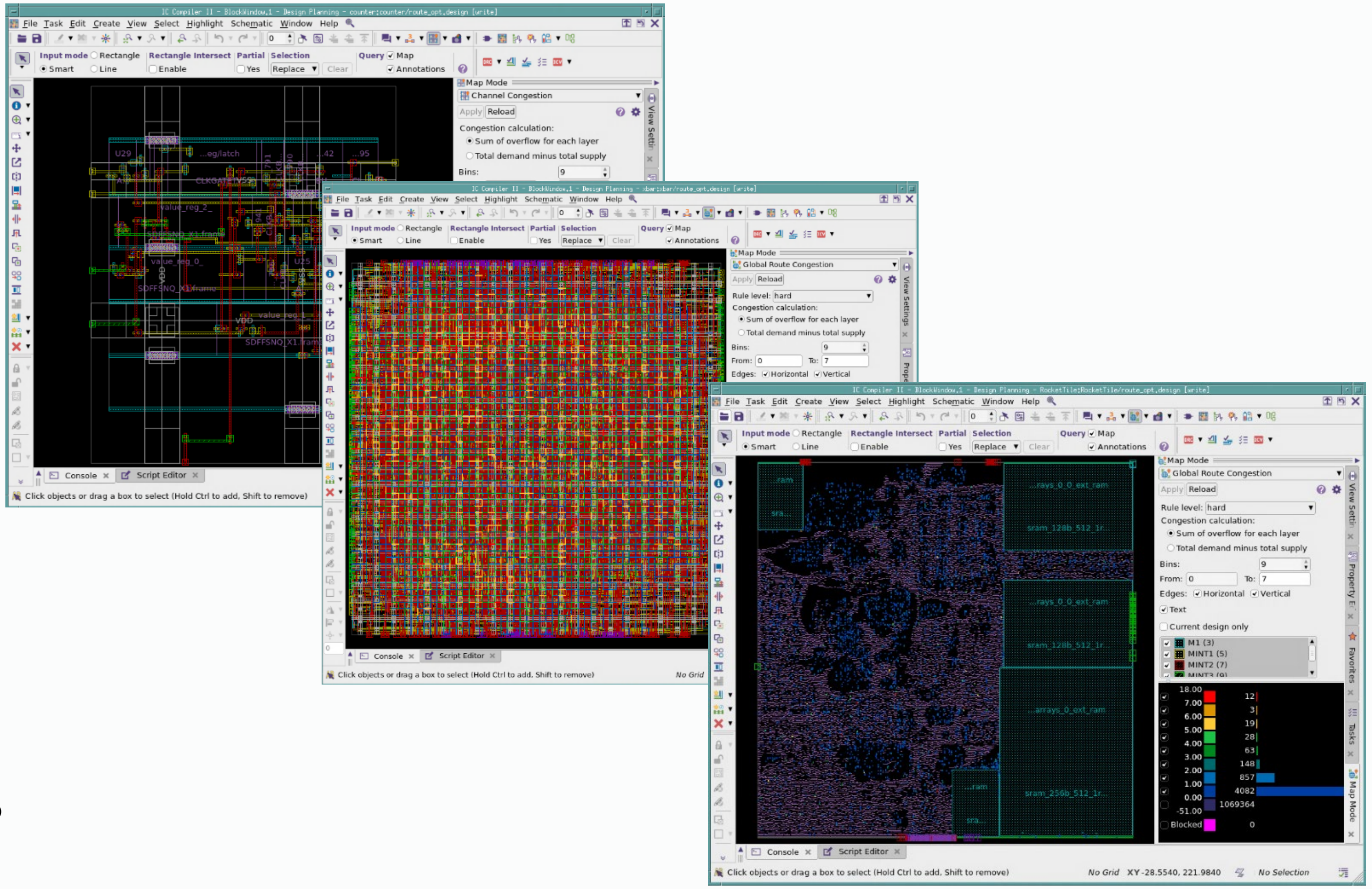
Abhinav Sinha and W. Rhett Davis

created in collaboration with UC San Diego and with the generous support of Qualcomm, Inc.

NC STATE UNIVERSITY Electrical & Computer Engineering

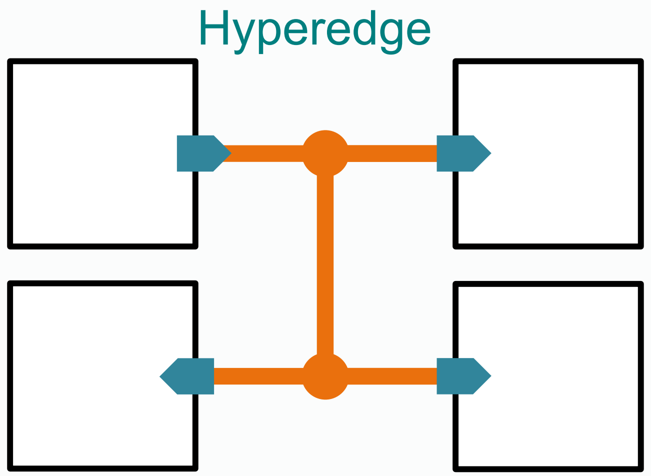
Example Designs

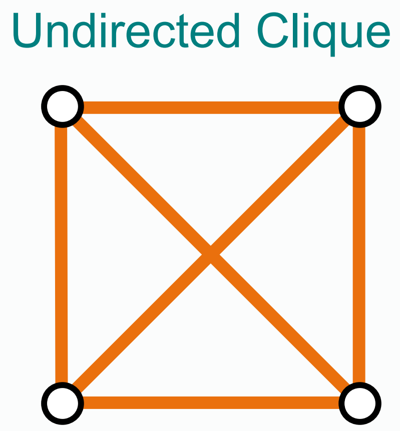
- counter
  - Simple starting example
  - 23 cells & 29 nets
- xbar
  - Highly congested routing
  - 3952 cells & 4484 nets
- RocketTile
  - Contains 5 memory macros
  - 183,560 cells & 84,494 nets



Hypergraph Representations

- How to represent hyperedges (*i.e.* nets) in a graph data-structure?
- Undirected clique method – Graph with all vertices fully connected
- Modified incidence matrix method – Sparse matrix with instance rows and net columns, value represents connected terminal index (or 0 if no connection)





Files provided for each design

- [design]_route_opt.def – Original design data
- [design]_clique.igpz – design as igraph Graph saved in “picklez” format
- **[design].json.gz – Instance and Net data**
- **[design]_connectivity.npz – Incidence matrix data as NumPy arrays**
- [design]_congestion.rpt – Routing congestion report
- [design]_congestion.npz – congestion as NumPy arrays
- [design]-congestion.png – Screen-shot of layout and congestion
- **Also find the cell library data in cells.json.gz and original library data in the lef subdirectory**

New

New

Supplemental files

- route_opt.check_routes – routing violations and resources usage per layer
- route_opt.report_utilization – ratio of required area to available area
- route_opt.report_clock_qor.structure – detailed clock-tree information
- route_opt.report_qor – Summary of system timing and other details

igraph Design Data (Clique Method)

```
>>> import igraph
>>> g = igraph.read('counter.igpz','picklez')
>>> len(g.vs)
23
>>> g.vs.attributes()
['instname', 'xloc', 'yloc', 'cell', 'orient']
>>> len(g.es)
56
>>> g.es.attributes()
['weight', 'netnames']
```

- read graph file
- get number of vertices
- get attributes for each vertex
- get number of edges
- get attributes for each edge

- Vertex Attributes
  - **instname** (string) – Instance name
  - **cell** (int) – Master library cell ID (array index)
  - **xloc, yloc** (int) – location of the instance
  - **orient** (int) – orientation of the instance

- Edge Attributes
  - **weight** (float) – edge weight, $4/[n^2-\mathrm{mod}(n,2)]$ (where n = hyperedge vertices)
  - **netnames** (string) – Net name(s) associated with this edge

Design Feature Data (Nets)

- Because multi-dimensional sparse arrays are not yet widely available, we cannot easily encode feature data in the incidence matrix
- Therefore, feature data for instances and nets are provided in [design].json.gz
- Contains a JSON object with the ‘nets’ member, which is an array of net objects
- Each net object has the following members:
  - name (str) – name of the net
  - id (int) – index of the net in the array

```
>>> with gzip.open('counter.json.gz','rb') as f:
...     design = json.loads(f.read().decode('utf-8'))
>>> design['nets'][3]
{'name': 'zero', 'id': 3}
```

Design Feature Data (Instances)

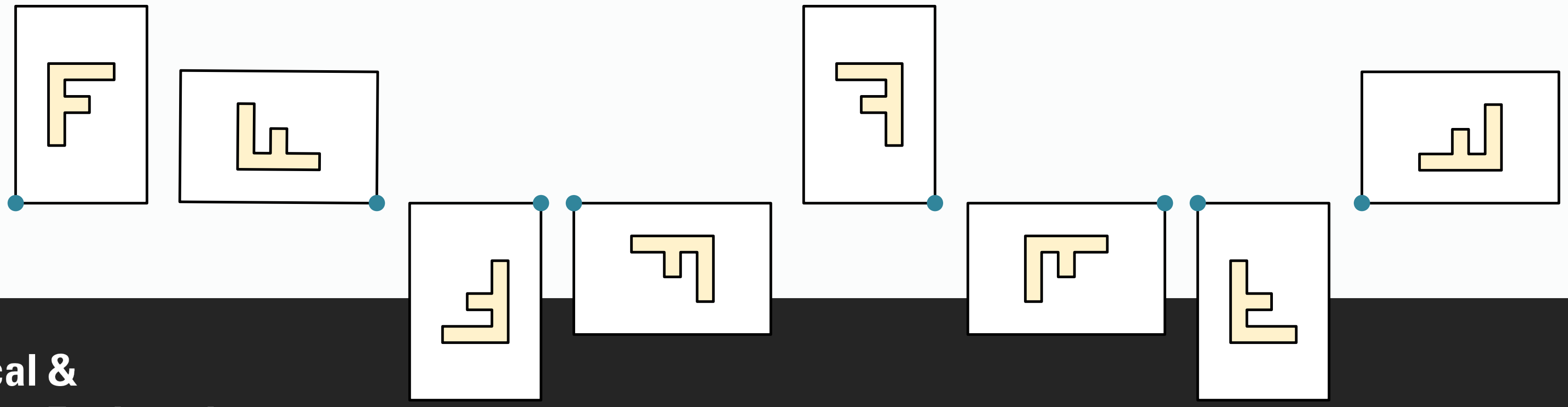
- The JSON object in [design].json.gz also contains the 'instances' member, which is an array of instance objects
- Each instance object has the following members:
  - name (str) – name of the instance
  - id (int) – index of the instance in the array
  - cell (int) – Master library cell ID (array index)
  - xloc, yloc (int) – location of the instance in ½ nm units (divide by 2000 to get microns)
  - orient (int) – orientation of the instance

```
>>> design['instances'][6]
{'name': 'U22', 'id': 6, 'xloc': 6784, 'yloc': 7680, 'cell': 7, 'orient': 6}
```

Orientations

- Orientation of each instance within a design is given by an integer as shown below (with OpenAccess and DEF orientations for comparison)
- Blue dot indicates the location (xloc & yloc) for each orientation
- Note that DEF always restricts the location to lower-left corner (unlike the Open Access location, which is used in our data-set)

This Dataset	0	1	2	3	4	5	6	7
Open Access	R0	R90	R180	R270	MY	MYR90	MX	MXR90
DEF	N	W	S	E	FN	FE	FS	FW



Cell Library Data

- Cell library data provided in cells.json.gz
- Contains an array of JSON objects with the following members
  - name (str) – name of the cell / module
  - id (int) – index of the cell in the array
  - width (int) – width of the cell in ½ nm units (divide by 2000 to get microns)
  - height (int) – height of the cell in ½ nm units (divide by 2000 to get microns)
  - terms – an array of terminal objects

```
>>> import json, gzip
>>> with gzip.open('cells.json.gz','rb') as f:
...     cells = json.loads(f.read().decode('utf-8'))
>>> cells[7]
{'name': 'AOI21_X1', 'id': 7, 'width': 768, 'height': 1536, 'terms': [{'name': 'A1', ...
```

Terminal Objects

- Terminal objects contain the following members
  - name (str) – name of the terminal
  - id (int) – integer index of terminal as it appears in the incidence matrix (same as array index plus 1)
  - dir (int) – direction of the terminal
    - 0 – input
    - 1 – output
    - 2 – inout
  - TBD – location of terminal

```
>>> cells[7]['terms'][2]
{'name': 'B', 'id': 3, 'dir': 0}
```

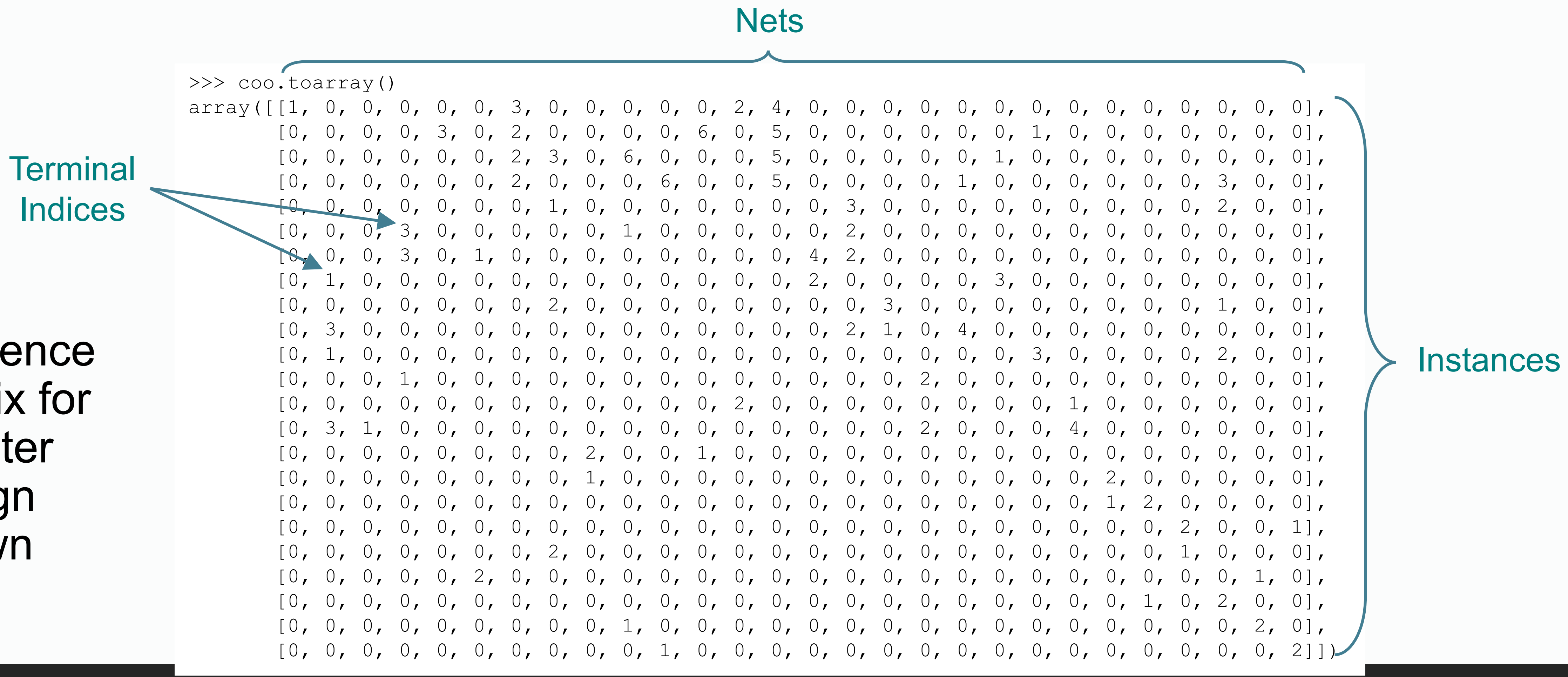
Incidence Matrix Data

- The file [design]_connectivity.npz contains NumPy arrays that can be used to define SpiPy sparse matrices in coordinate (COO) format
- COO matrices can be quickly constructed and easily converted into other matrices
- Each non-zero entry in the matrix gives the terminal index
- Remember to subtract 1 to get the index for each cell's terminal array
- Note that the matrix will contain duplicate indices if a net connects multiple terminals on one instance

```
>>> import numpy as np
>>> from scipy.sparse import coo_matrix
>>> conn=np.load('counter_connectivity.npz')
>>> coo = coo_matrix((conn['data'], (conn['row'], conn['col'])), shape=conn['shape'])
>>> coo.getrow(6).getcol(3).toarray()
array([[3]])
>>> cells[design['instances'][6]['cell']]['terms'][3-1]
{'name': 'B', 'id': 3, 'dir': 1}
```

Example Incidence Matrix

- Incidence matrix for counter design shown



```
>>> coo.toarray()
array([[1, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 3, 0, 2, 0, 0, 0, 0, 6, 0, 5, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 2, 3, 0, 6, 0, 0, 0, 5, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 6, 0, 0, 5, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 3, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0],
       [0, 0, 0, 3, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 2, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]])
```

NumPy Congestion Data

```
import numpy as np
data=np.load('counter_congestion.npz')
lyr=list(data['layerList']).index('M1')
ybl=data['yBoundaryList']
xbl=data['xBoundaryList']
i,j=getGRCIndex(xloc,yloc,xbl,ybl)
data['demand'][lyr][i][j]
data['capacity'][lyr][i][j]
```

- read NumPy file
- get the index for layer M1
- get the array of y-coordinates for GRC left-edge boundaries
- get the array of x-coordinates for GRC bottom-edge boundaries
- get the array indices for the GRC containing location (xloc,yloc)
- get the demand for layer M1 at location (xloc,yloc)
- get the capacity for layer M1 at location (xloc,yloc)

- Routers report congestion information for each Global Route Cell (GRC) using the following values
  - Demand – number of routing tracks needed in the GRC
  - Capacity – number of possible routing tracks in the GRC (this number does not vary with demand)
- GRCs are organized in a rectangular grid, but not all GRCs have the same dimensions.
  - We use the x|yBoundaryList arrays to keep track of the GRC boundaries. This helps convert locations to array indices.
  - All x,y dimensions are in ½ nm units (i.e. divide by 2000 to get microns)

getGRCIndex Function

```
def getGRCIndex(x,y,xbl,ybl):
    j=0
    for b in xbl[1:]:
        if x<b:
            break
        j+=1
    i=0
    for b in ybl[1:]:
        if y<b:
            break
        i+=1
    return i,j
```

- Routers report congestion information for each Global Route Cell (GRC).
- The getGRCIndex() function used decode GRC indices from x,y coordinates (shown on the previous slide) can be defined as shown on this slide.
- Note that the **demand** and **capacity** arrays are organized with x-coordinates increasing in rows (*i.e.* last array index) and y-coordinates increasing in columns (*i.e.* next-to-last array index). Therefore, in the reference data[‘demand’][lyr][i][j],
  i references the y-location,
  while j references the x-location.