

Hierarchical Design Representation and Milkyway

Laurence Brevard

Staff Engineer

Milkyway R&D

Synopsys, Inc.



Items for Today's Talk



- Hierarchical Design Concepts
- Hierarchical Features of the Milkyway database
- Topics
 - The Cell / Instance Containment Hierarchy
 - Folding and Unfolding
 - Flattening
 - The Instance Master Concept
 - Wire / Wire Master
 - Hierarchy Traversal
 - Hierarchical Naming
 - Instance paths
 - Non-scalar signal / net handling
 - Physical Hierarchy Example

What is Hierarchy?



- Containment definition and examples
- Hierarchical Items that are not directly supported in Milkyway include:
 - structured signals (buses and bundles)
 - but tools have mechanisms for tracking this
 - types /classes
 - could be part of a particular design style
- Software analogies
 - subroutines
 - macros

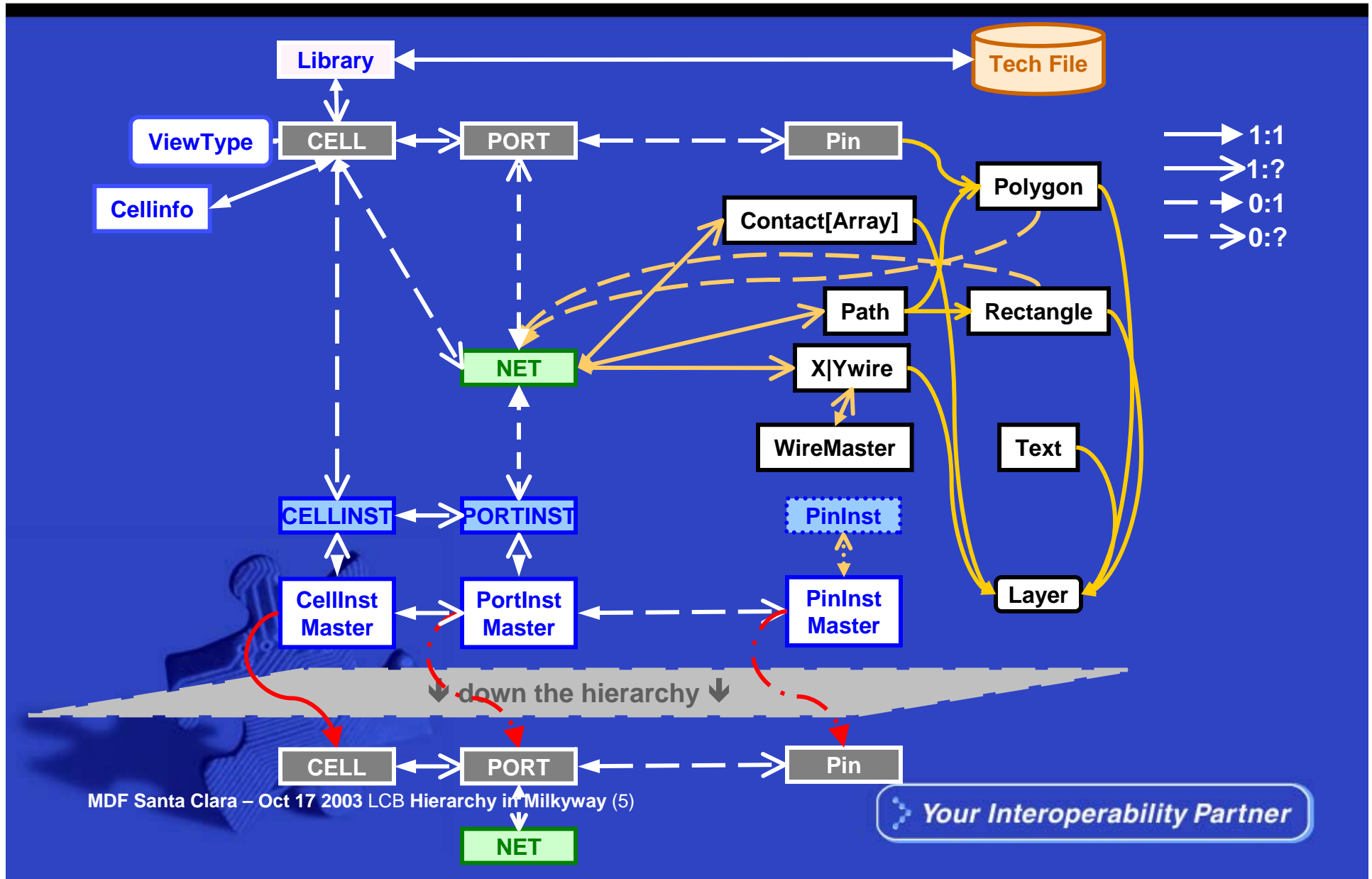
Why is Hierarchy used?

SYNOPSYS®

- to reuse designs or design fragments
- to manage complexity by factoring a design into smaller parts
- to confuse young engineers! 😊



Milkyway Objects Overview



Milkyway Hierarchy Objects



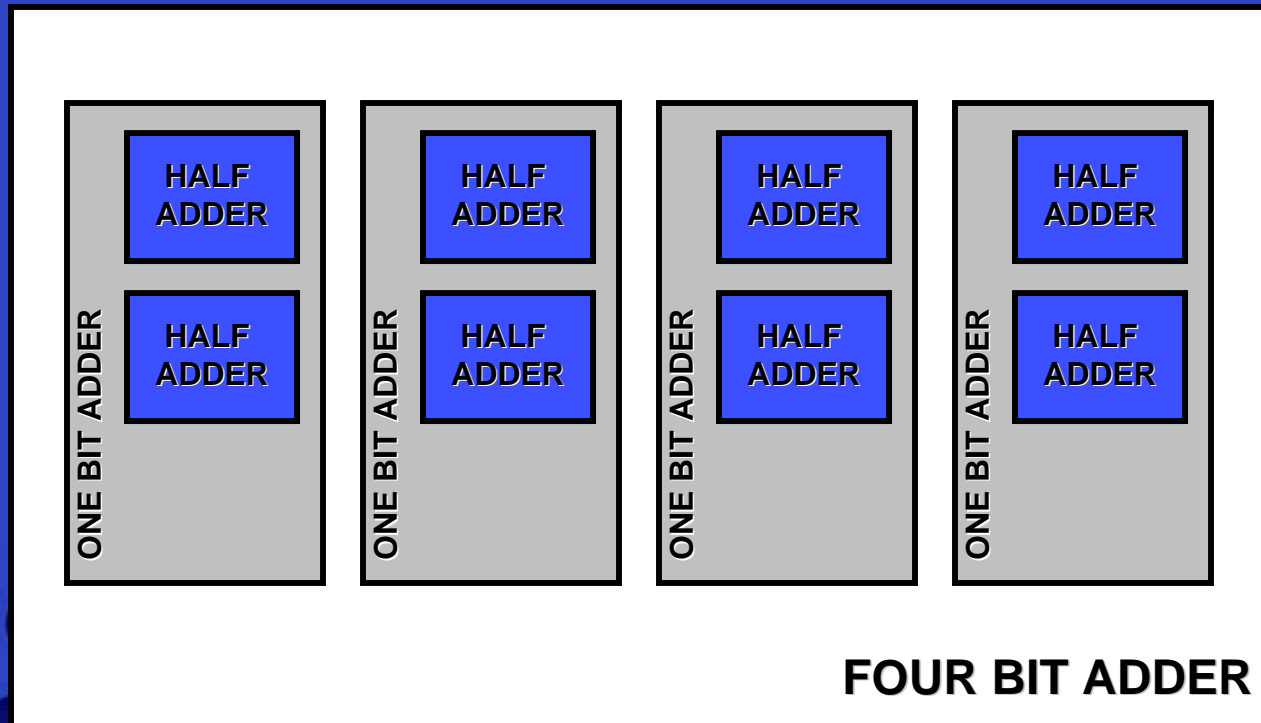
- Cell, Cell Instance, Cell Instance Master
- Port, Port Instance, Port Instance Master
- Pin, (Pin Inst), Pin Instance Master
- Not part of the containment hierarchy but definition and instance objects:
 - wire master, wire
 - contact definition /code, contact (i.e. via)

Next, Terminology Discussion for...



- Containment
- Folded representation
- Unfolding
- Expansion
- Flattening

Containment



- Milkyway provides a Containment Hierarchy
 - Cells can contain (references to) other cells.
 - A cell is a definition (ChdStd/IBMIDM “def box”)
 - A cell instance is a reference (ChdStd/IBMIDM “ref box”)
 - This is not a CLASS hierarchy such as done in an object oriented language – e.g. C++

- Used for:
 - Factoring a design into manageable pieces
 - Reuse
 - The same cell can be...
 - used in multiple places in a design
 - used in more than one design

- Milkyway provides a Folded Hierarchy Flattened by Tools
 - Folding == two instances can use the exact same cell as a definition
 - Unfolding == uniquification (== expansion*)
 - Flattening == create one cell with all leaf cells in it i.e., those with no further cell instances
- * The term expansion is sometimes used for unfolding and sometime for flattening!*

Adder Example: Hierarchy in Verilog



FULL ADDER

```
module full_add(in1, in2, cin, sum, cout);
  input in1, in2, cin;
  output sum, cout;
  wire w1, w2, w3;
  half_adder half_add_1(.in1(in1), .in2(in2),
                       .sum(w1), .cout(w2));
  half_adder half_add_2(.in1(w1), .in2(cin),
                       .sum(sum), .cout(w3));
  or02d1 or_1(.A1(w2), .A2(w3), .Z(cout));
endmodule
```

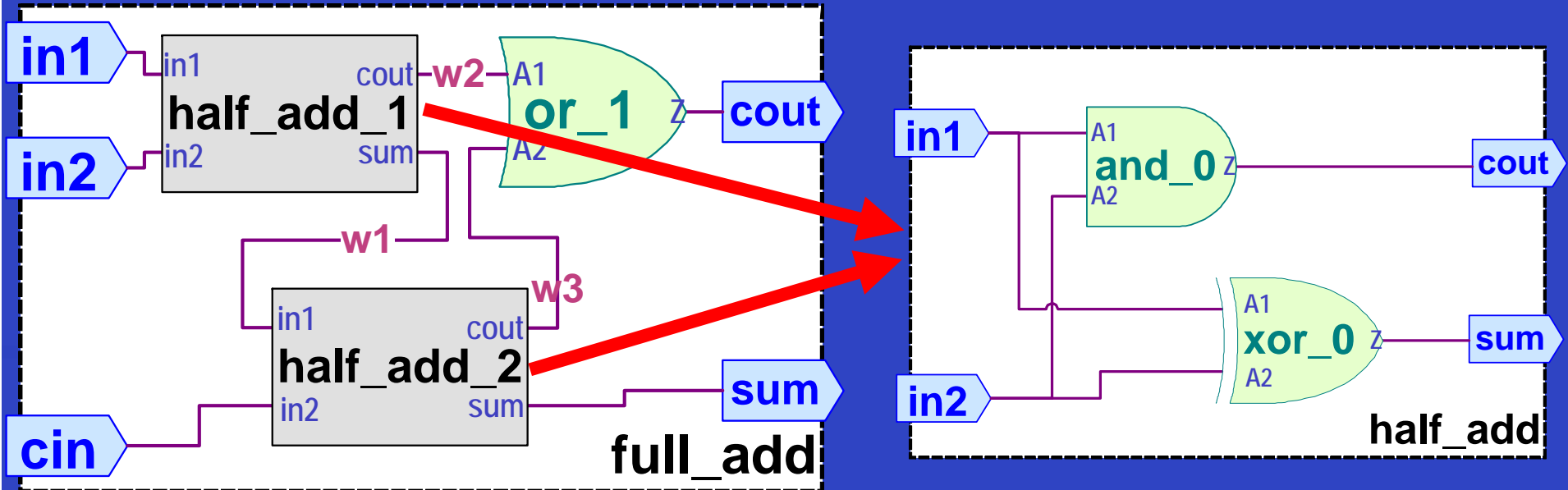
HALF ADDER

```
module half_add(in1, in2, sum, cout);
  input in1, in2;
  output sum, cout;
  and02d1 and_0(.A1(in1), .A2(in2), .Z(cout));
  xor02d1 xor_0(.A1(in1), .A2(in2), .Z(sum));
endmodule
```

Adder Hierarchy as Schematic...



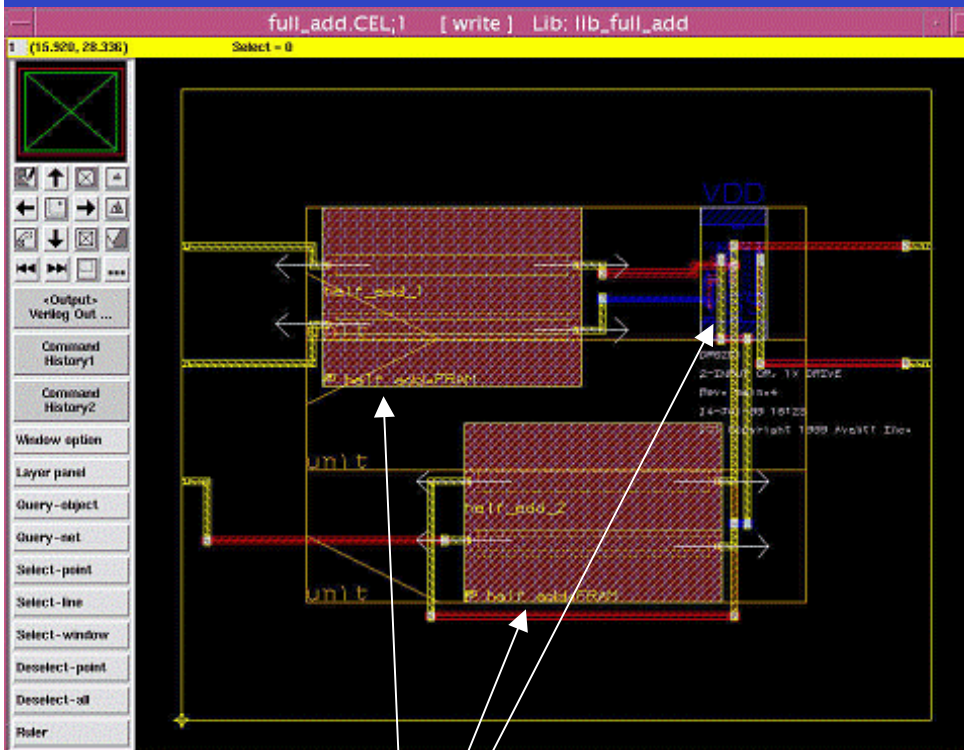
- We could diagram the previous Verilog example as follows...



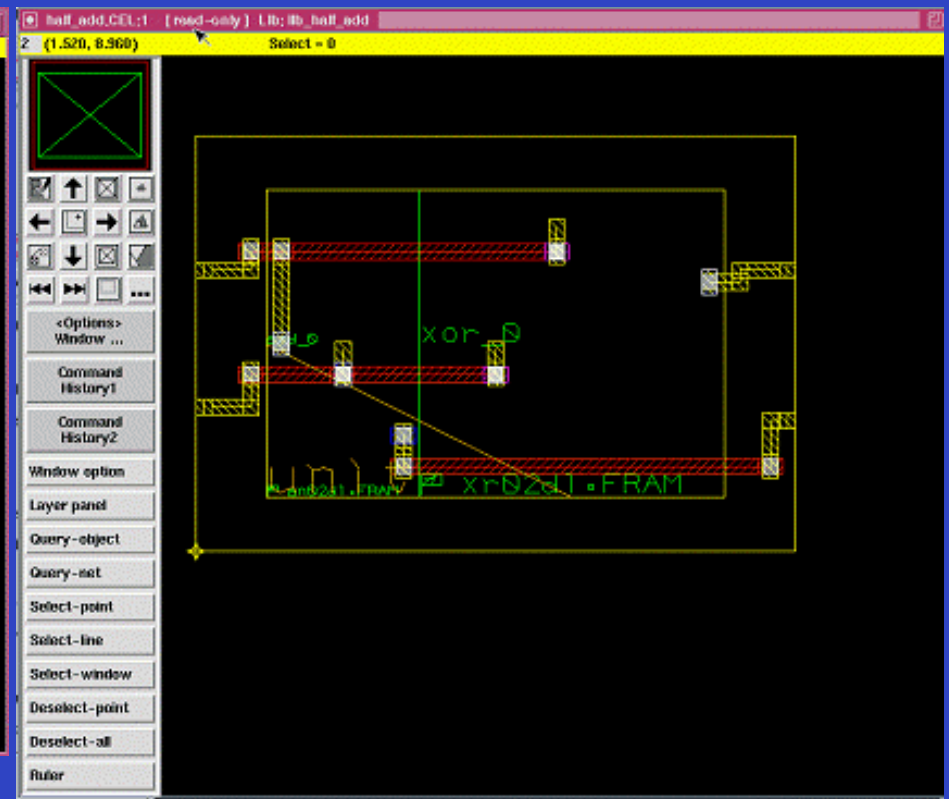
Adder Hierarchy as Layout...



Full Adder – Cell Definition



Half Adder – Cell Definition



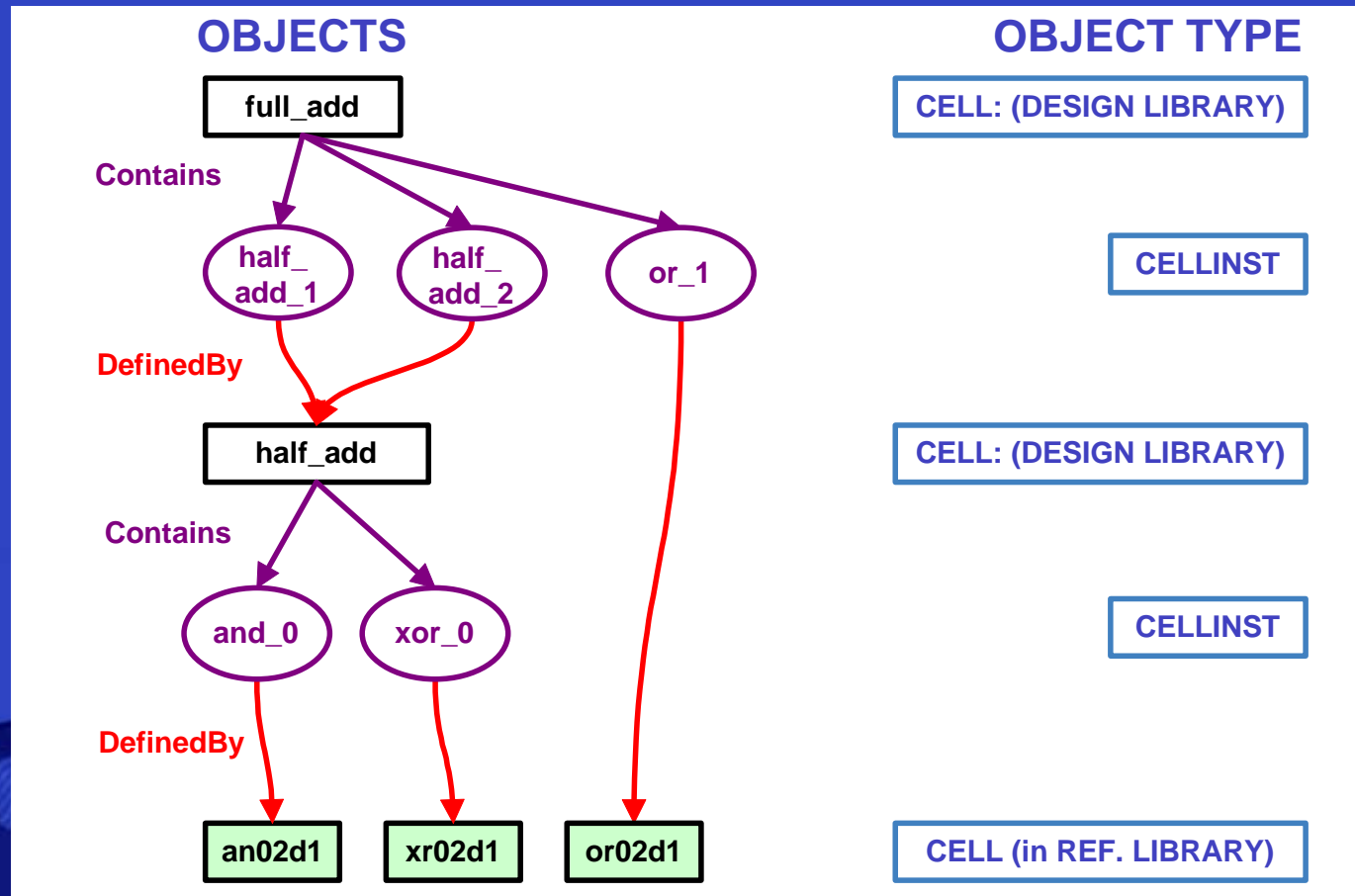
Cell Instances

✓ NOTE: Placing and routing this design did not stress Astro very much! 😊

Adder Hierarchy as a Graph...



- The hierarchical object relationship may be shown as a directed bi-partite graph:



- CellInstMaster
 - acts as a kind of *cache* of defining cell information
 - can avoid opening the defining cell
 - called an *instance header* in some other systems
- One CellInstMaster per uniquely referenced cell
 - Created when the first instance of a given cell is created
 - The set of all CellInstMasters in a given cell provides a “bill of materials” for that cell – showing what cell definitions are needed by it.
 - Each CellInstMaster points to all the cell instances that are using it – where-used information within this cell only

- Interface information for a given referenced cell definition is provided by the combination of...
 - One CellInstMaster,
plus...
 - PortInstMasters – one per port on the referenced cell
 - PinInstMasters – one per pin on the referenced cell

- Not part of the Cell / Cell Instance Hierarchy
- Wire information is factored into Wire Masters
 - Index
 - Layer
 - Width
 - Path Type (whether length includes half width or not)



- Individual Wires are either Horizontal or Vertical
 - References a wire master index
 - Has a start point and length
 - References a Net
 - Route type and region association for routing
 - Can override layer in wire master

So...

What do I mean by FOLDED?

Some kind of ORIGAMI ??? 😊

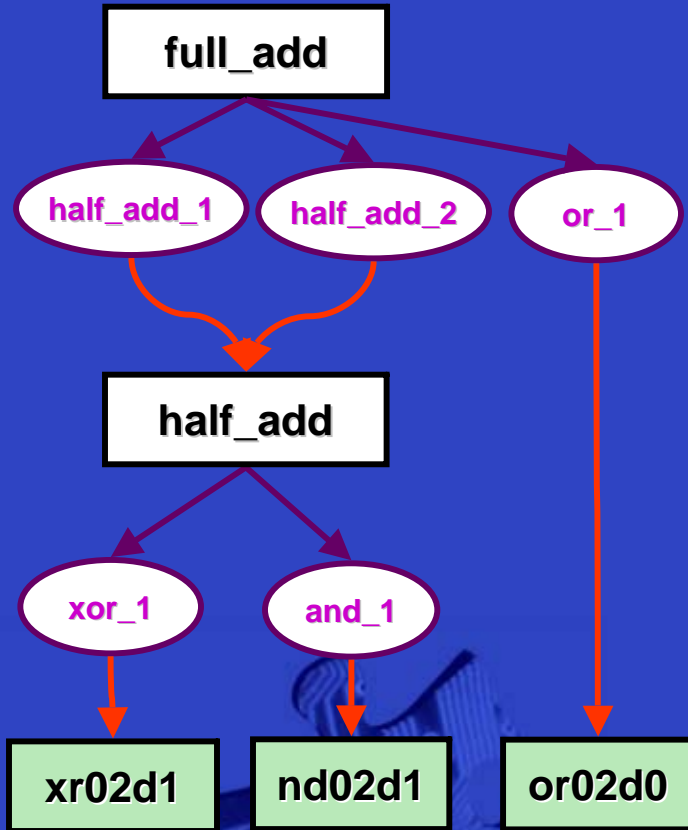
- FOLDED means that what is conceptually a containment tree is *folded* into a directed graph.
- See next slide...



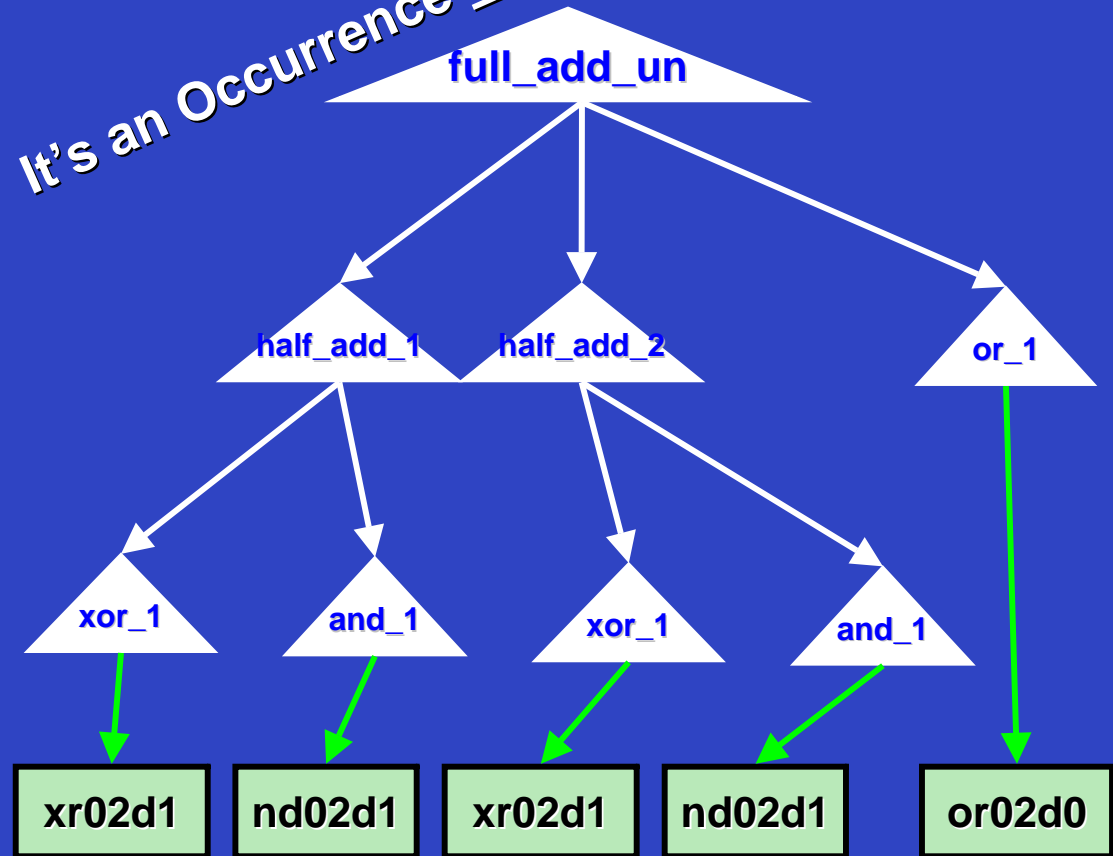
Folded vs. Unfolded Concept



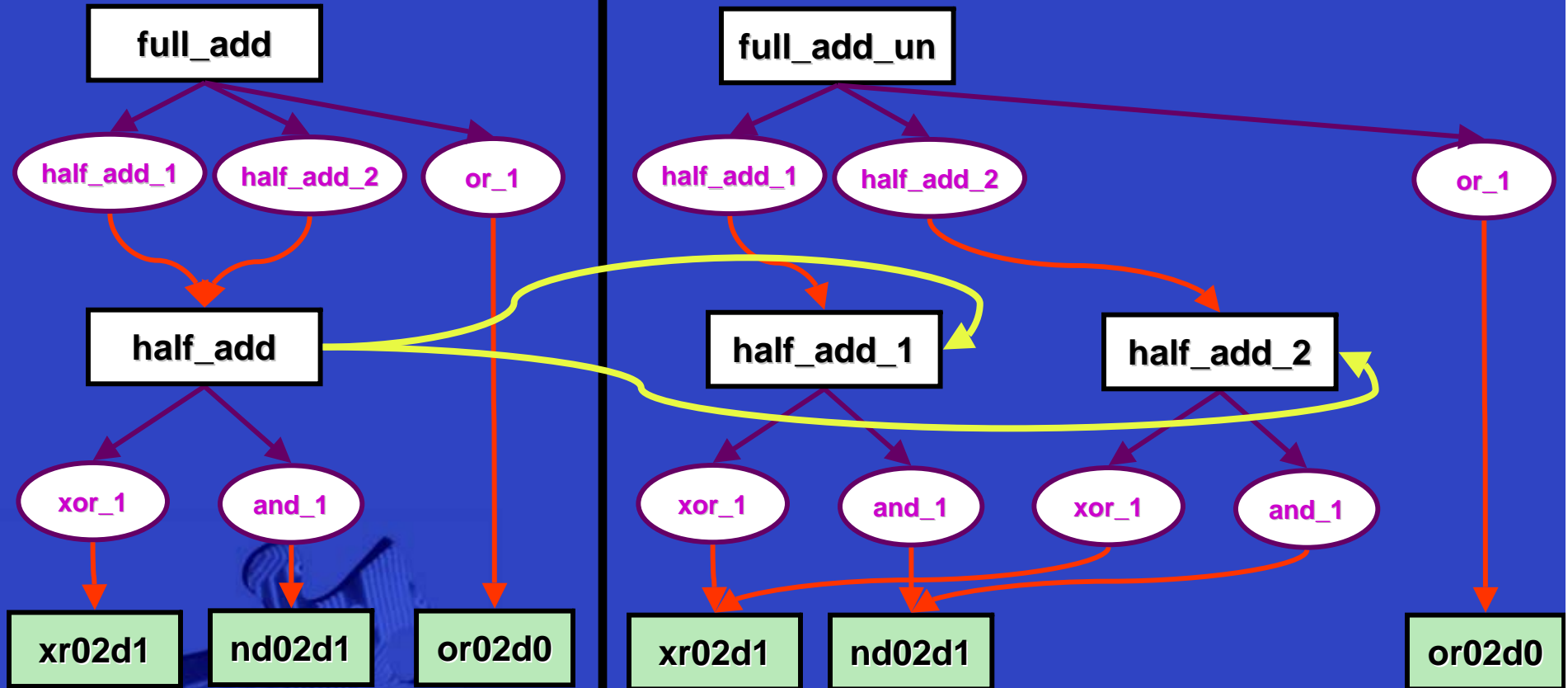
Folded => a Directed Graph



It's an Occurrence Tree!



Unfolding *Uniquifies* cell definitions



Unfolding versus Flattening



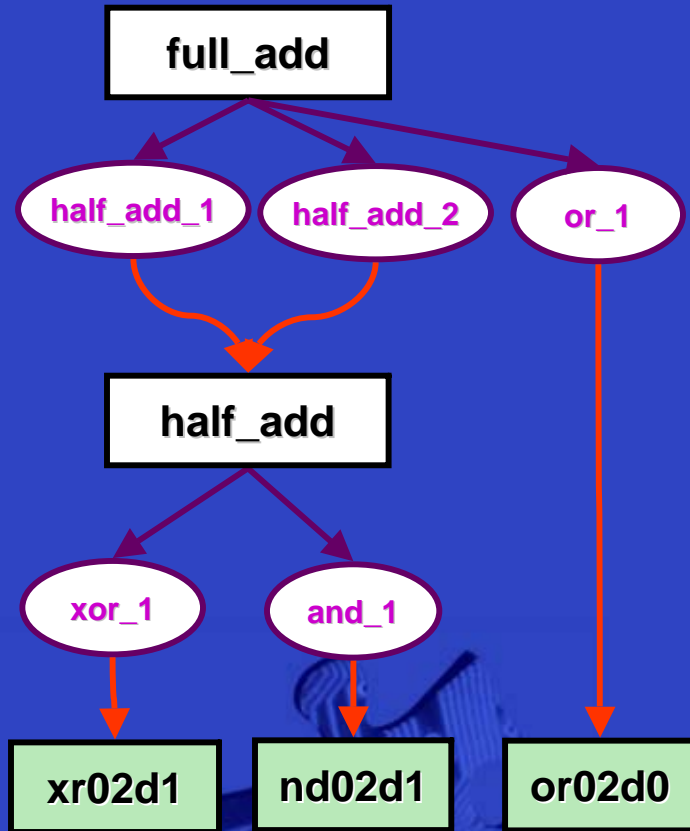
- If there are only two levels...
unfolding *is* flattening!
- Flattening means...
 - going all the way to the “bottom” of the design,
to “leaf cells” that contain no further instances, and
 - pulling the leaf cells up into the top cell.

NOTE: “leaf cell” depends on what is the “bottom” of the design
for *your* purposes

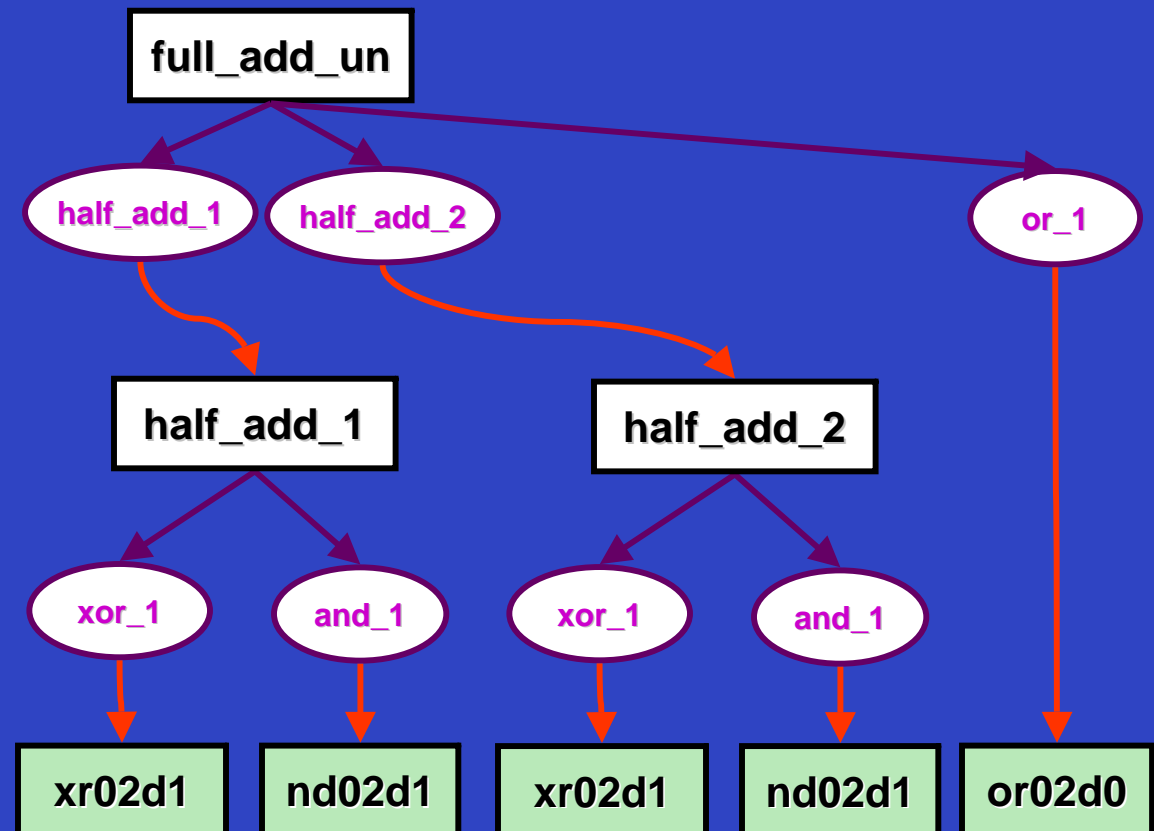
Unfolded still has intermediate levels



FOLDED...

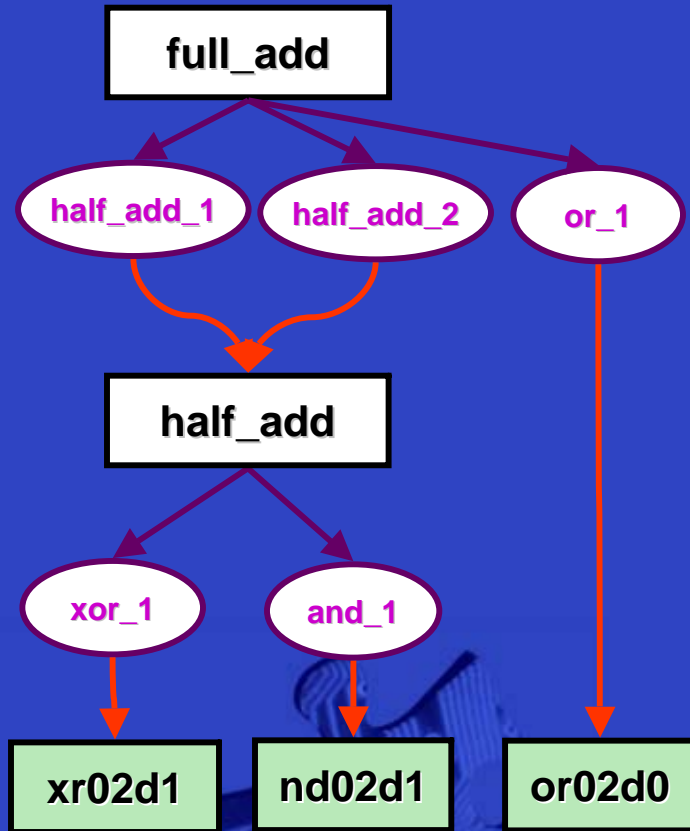


...UNFOLDED

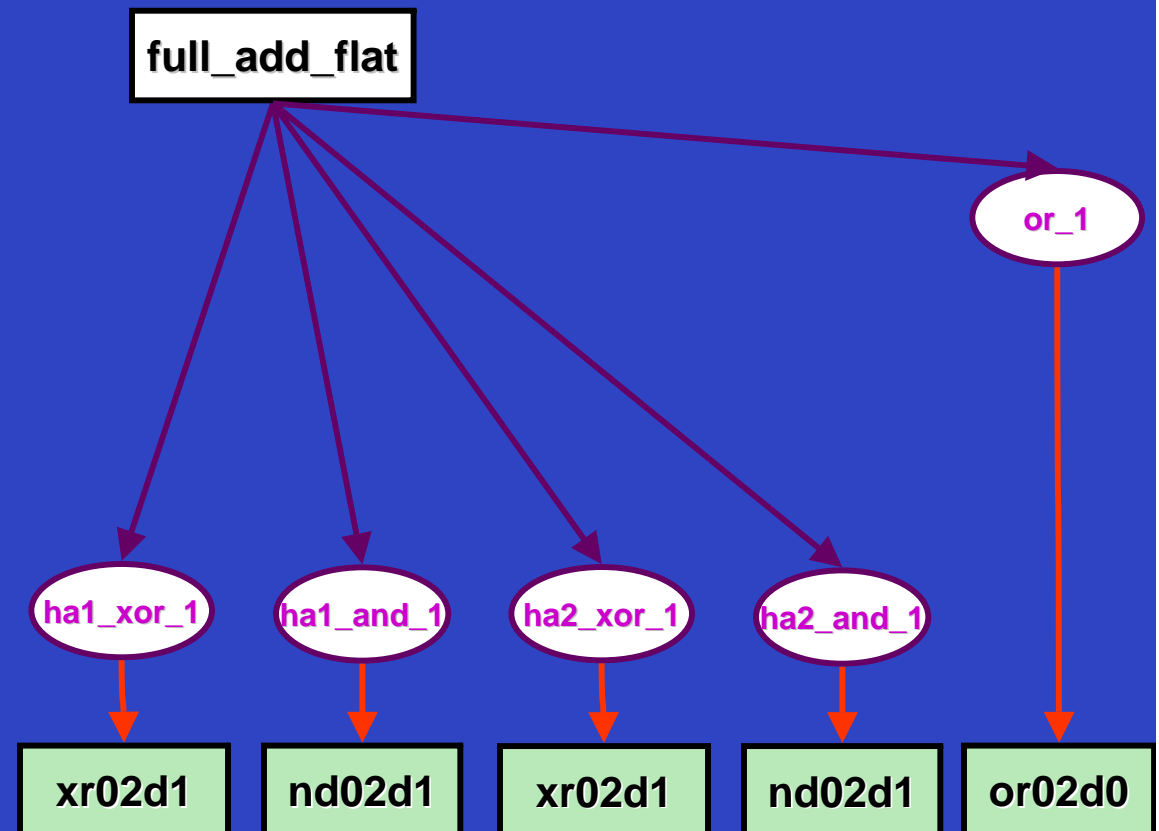


Flattened Design has only “leaf” cells **SYNOPSYS**[®]

FOLDED...



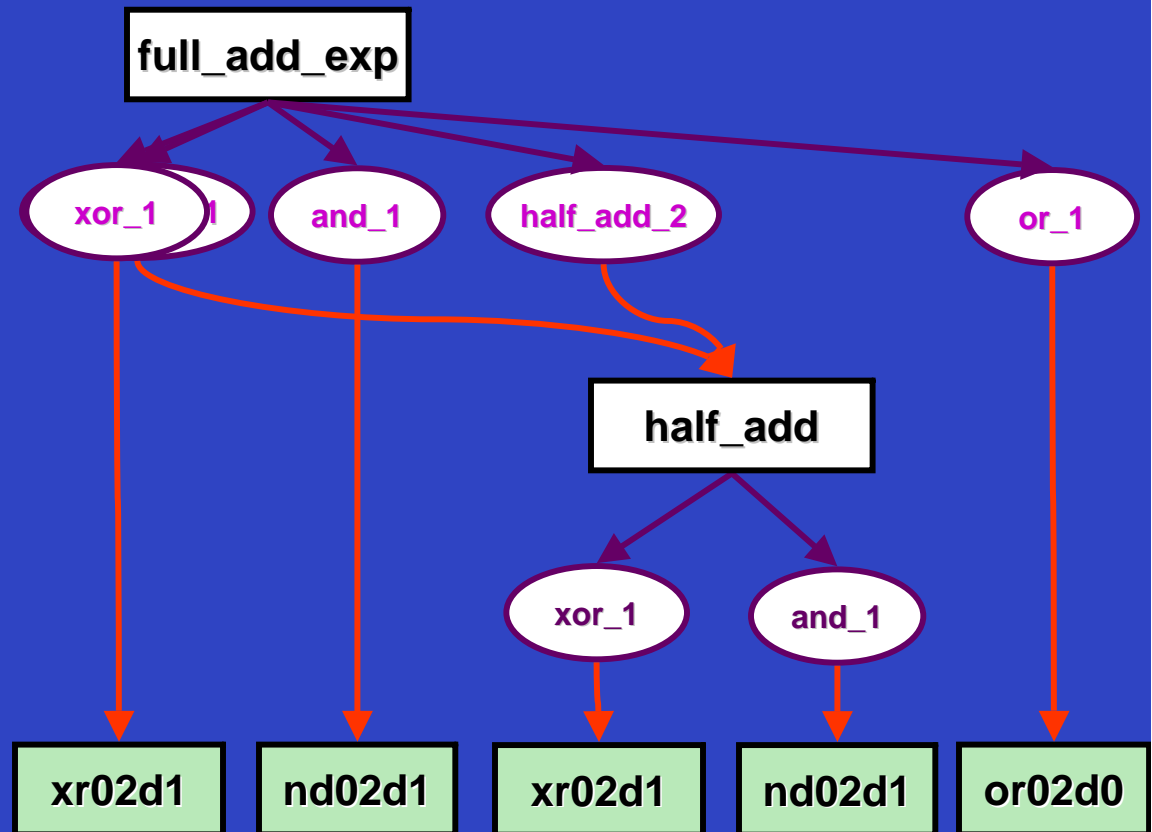
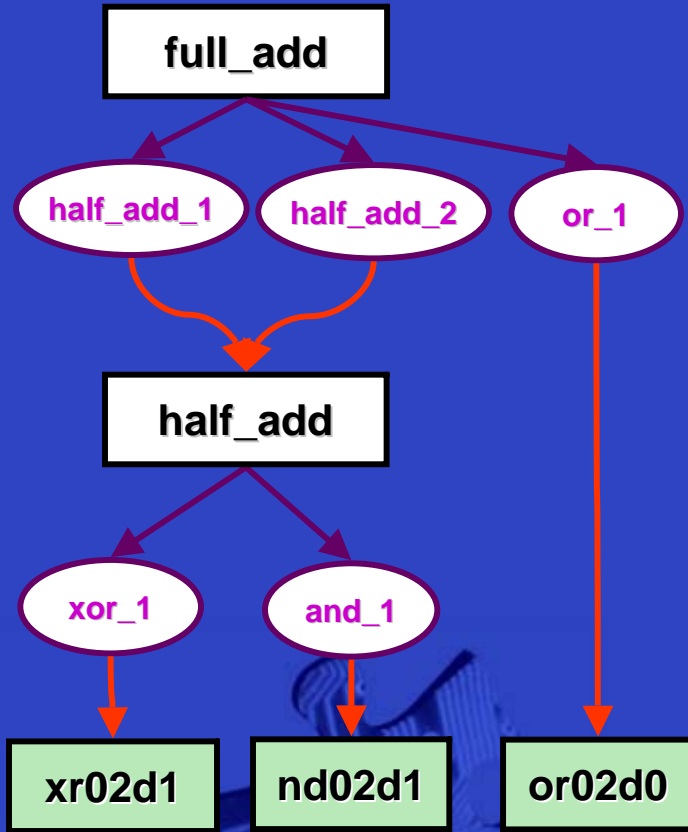
...FLATTENED...



Expanding One Level – Destructuring **SYNOPSYS**[®]

- You can also expand a cell one level – to just contain the cells directly below it
 - In this case, the cell instances (and all other contents) of the cell below are moved up into the upper cell in place of the single instance being expanded.
 - This is sometimes called “destructuring”.
 - This will not flatten the design unless the cell definitions of all instances being expanded have no further instances.

Expanding just one instance...



Synopsys has a **Virtual Flat Concept**



- A design is kept hierarchical but treated as expanded or flattened as needed.
- I wish I could say more.
- I really do...



Electrical Connectivity and Hierarchy **SYNOPSYS**[®]

Up to now we have just shown CONTAINMENT

Next, we will discuss...

CONNECTIVITY in a Milkyway design hierarchy



- **Netlist Electrical Connectivity in Milkyway** pretends there are functions, wired together with zero resistance and delay nodes (the Lumped Parameter Abstraction)
 - The Milkyway **Net** is an Equipotential Node in this abstraction
- Electrical Connectivity goes from this abstract concept through to physical implementation...
 - **Net** ← the most abstract
 - **Glink** – a global route (pin to pin straight lines)
 - **Path** – a detailed route or a polyline with width
 - **Wires and Contacts (Vias)** ← the least abstract

Beyond the basic net object...



NOTE...

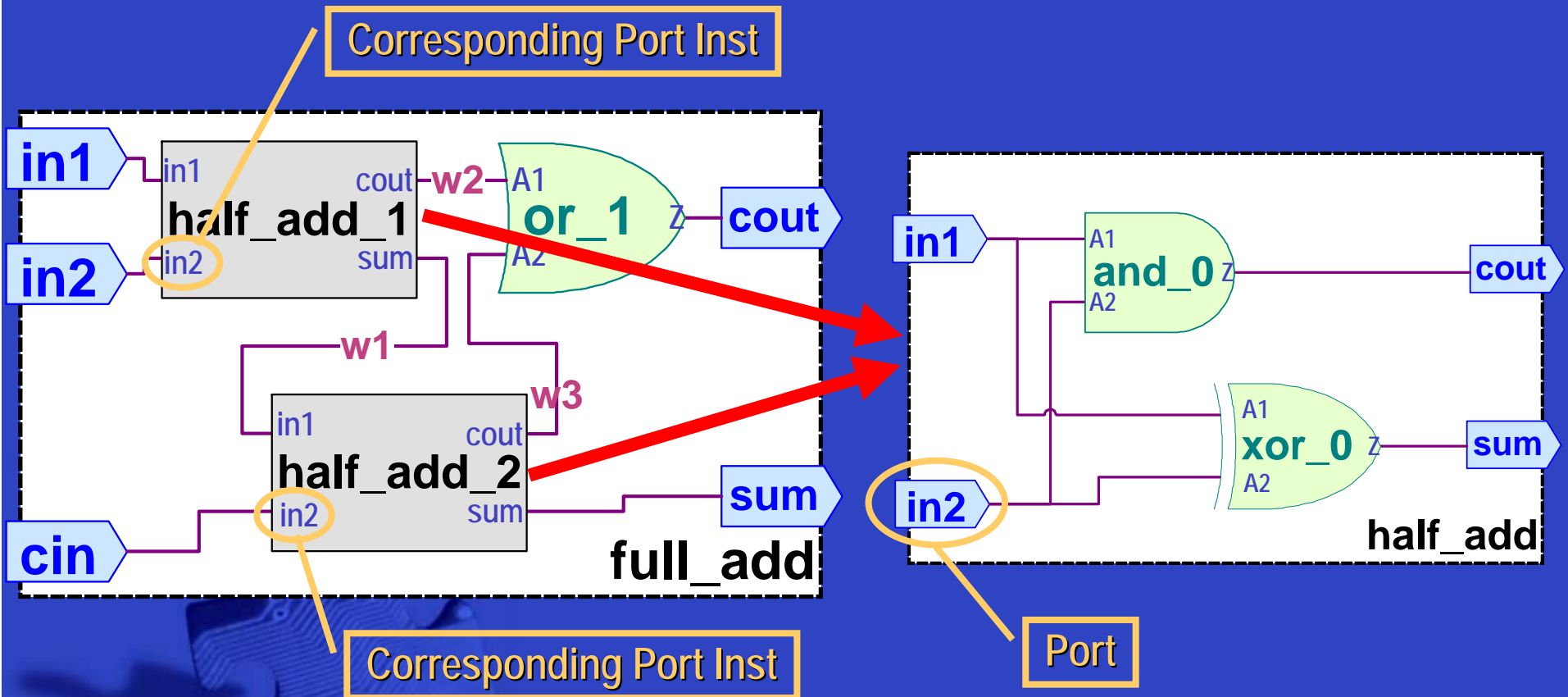
- While the Milkyway NET is a zero delay node in the abstraction...
- ... nets are ultimately implemented in layout from which various tools can extract extensive information about parasitics and interconnect delays.

But...

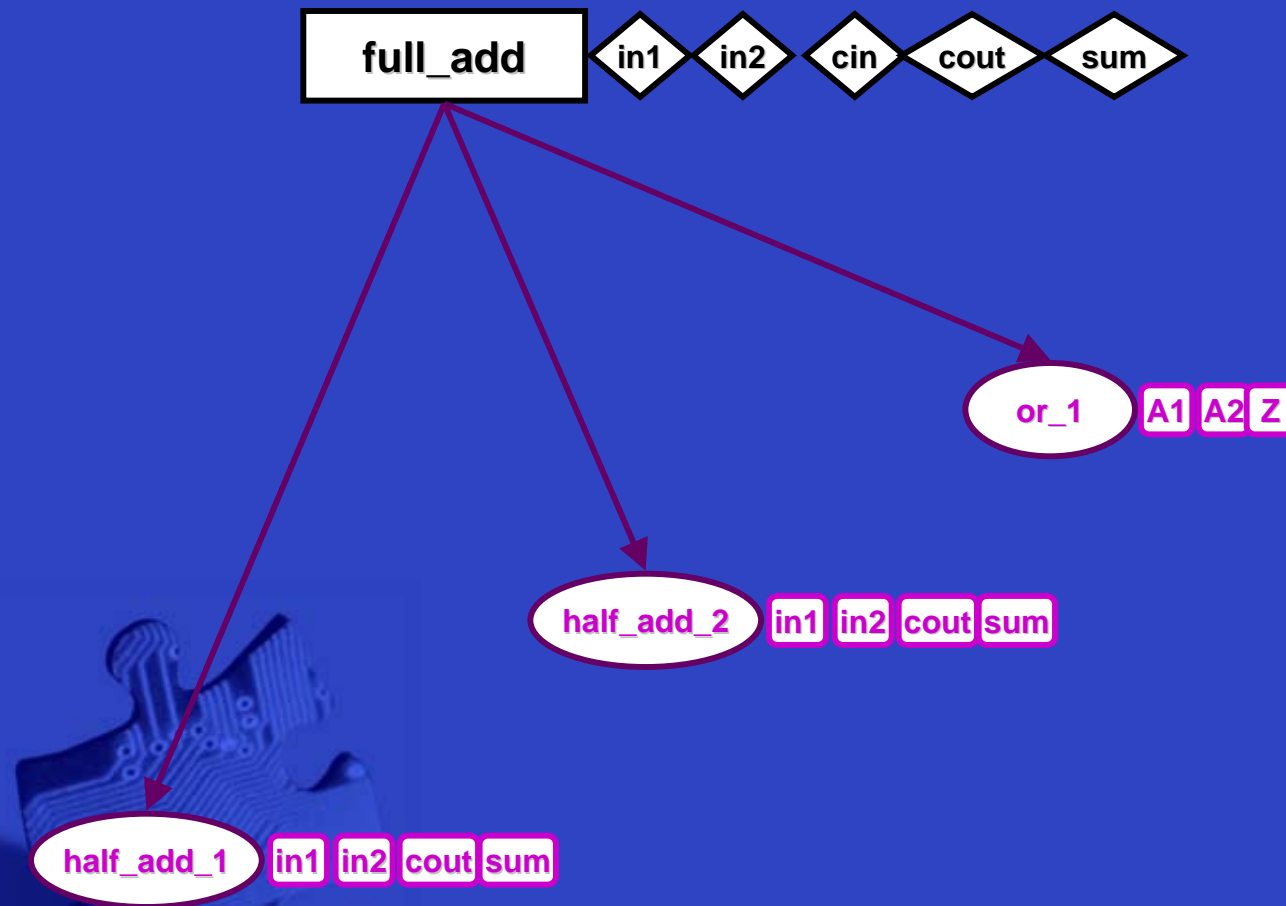
– We are not going to talk about that today.

- A Cell has a Port for any Net inside it which may be connected to from outside:
 - Akin to “formal parameters” to a subroutine
- Thus Cell Instances have Port Instances:
 - The “actual parameters” when calling the subroutine
- Nets connect Port Instances and Ports

- Using the previous adder example:



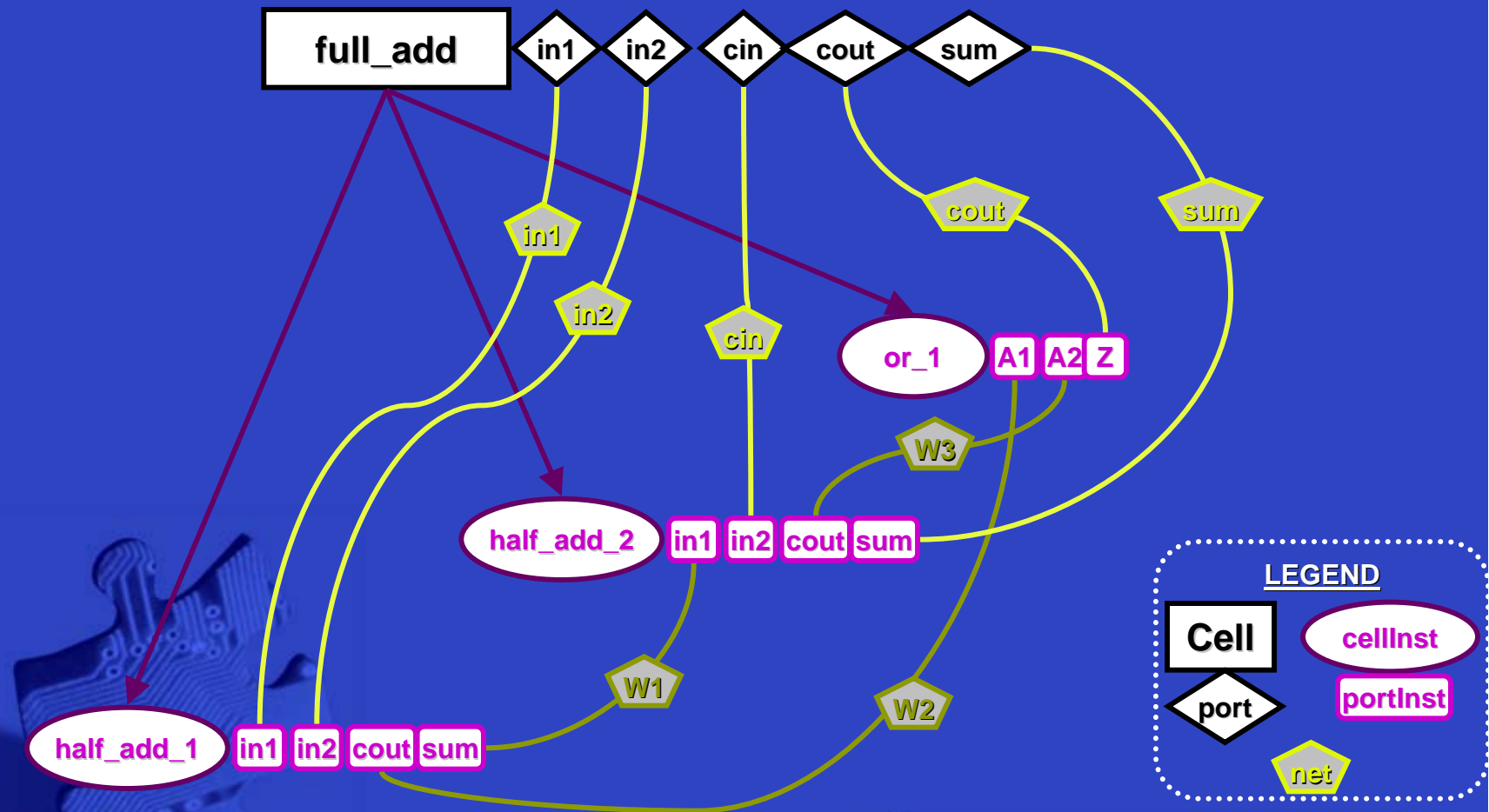
Ports in the Graph Notation



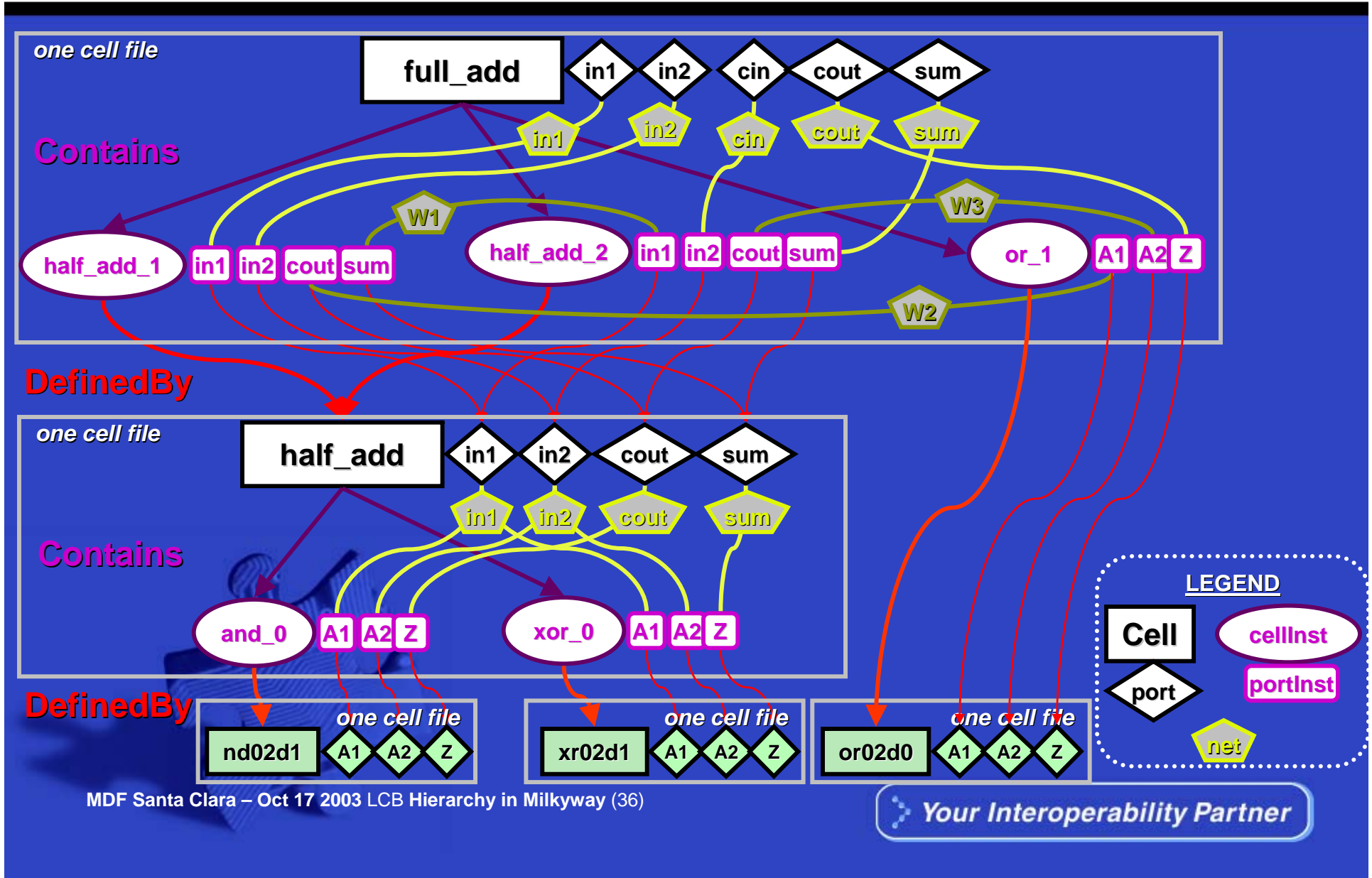
Ports and Nets in the Graph Notation



Begins to get rather messy – even without inst. masters!



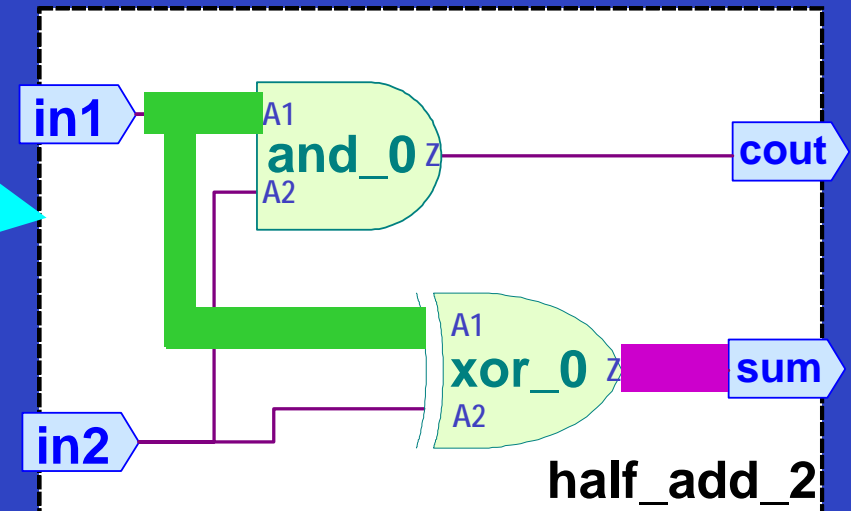
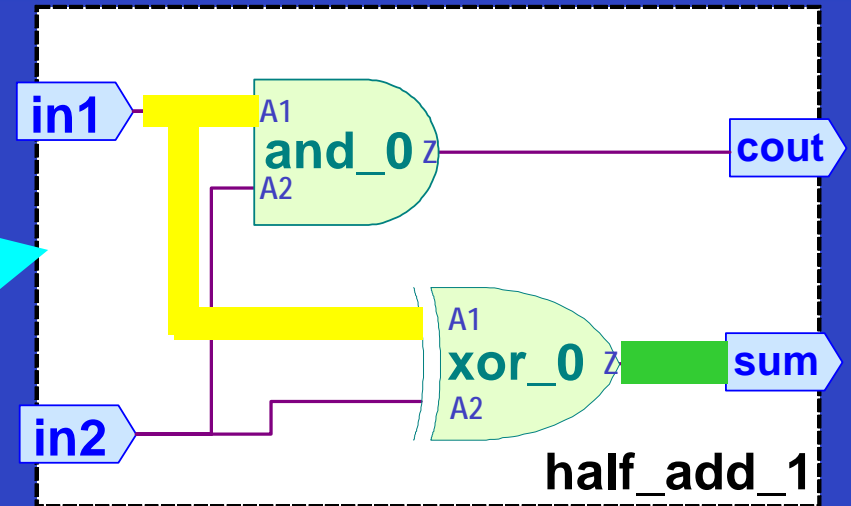
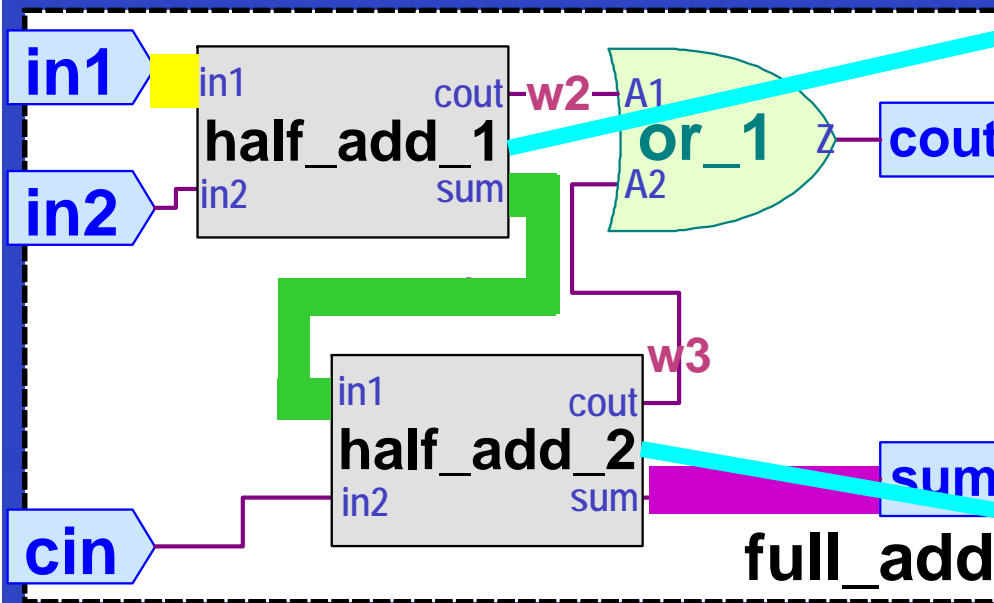
The full_add netlist as Graph...



Electrical Connectivity – Net



- Three Nets – shown in unfolded hierarchy



Hierarchical Milkyway Nets



- The Milkyway Net object is only ever in a single Cell
 - it cannot go between cells!
- Therefore a net in a design is *potentially* ☺ represented by several Milkyway Net objects.
- The first example design net in the previous slide was shown in **yellow**. It uses two Milkyway net objects in an unfolded Milkyway pair of cells.
 1. the Milkyway net (named 'in1') connected in Milkyway Cell 'full_add' to port 'in1' and to portInst 'in1' on cellInst 'half_add_1'
 2. the Milkyway net (also named 'in1') connected in Milkyway Cell 'half_add_1' to port 'in1' and to portInst 'A1' on cellInst 'and_0' and portInst 'A1' in cellInst 'xor_0'

Functions to Traverse Hierarchy

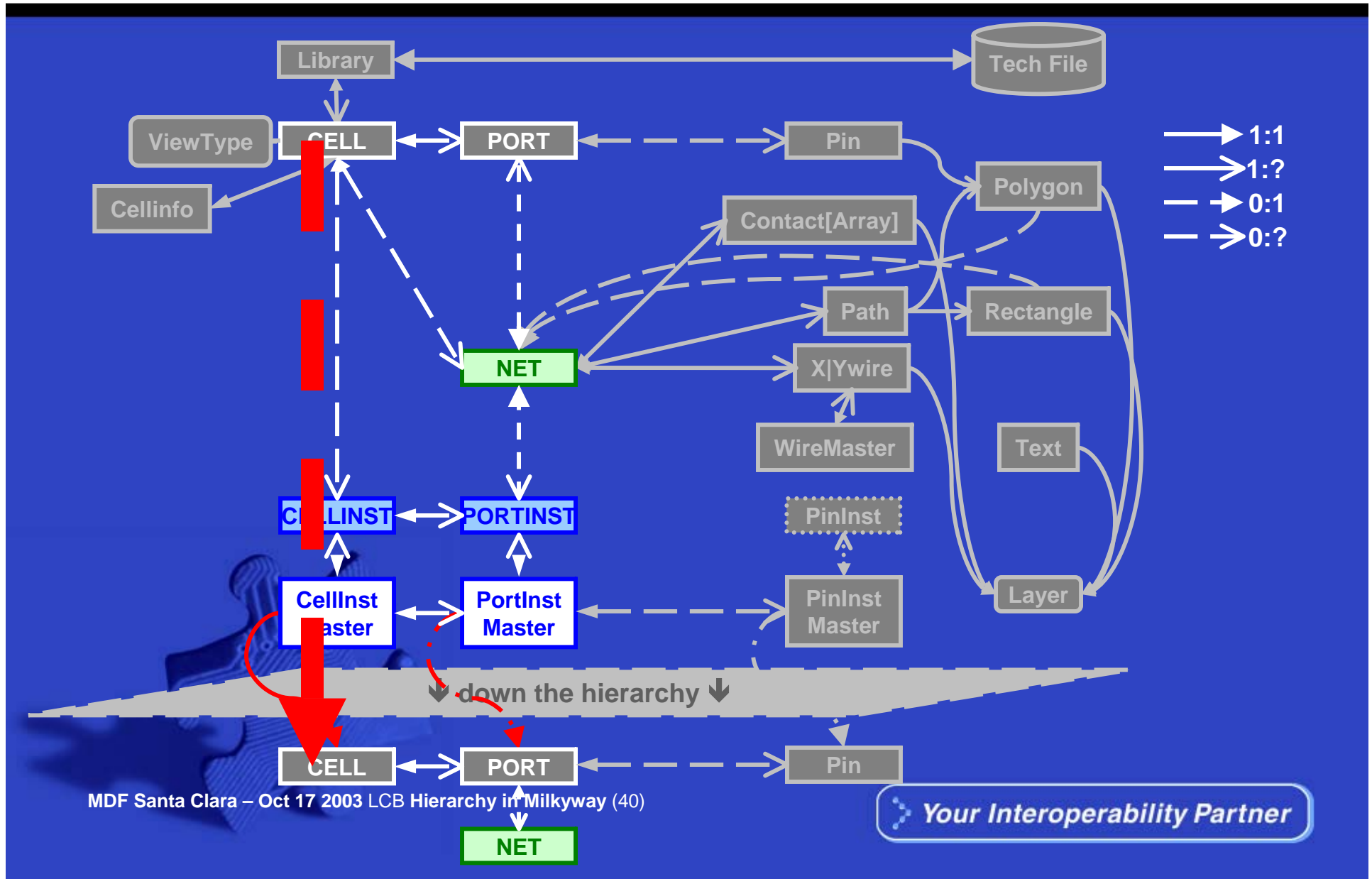


- Cell to Cell Instance
- Cell Instance to Defining Cell

- Port to Net
- Net to Port Instance
- Port Instance to Port on Defining Cell



Hierarchy Traversal – Cell to Cell



- Cell to Cell Instances

- Scheme

- ```
(db-foreach cellId '() "cellInst" nextInst
; operations on nextInst)
```

- C-API

- ```
MWXDb_Get_Objects_ByType(cellId, cellInstTypeCode,  
&numberOfInsts, &instIdArrayPointer)
```

- Specific Cell Instance to Defining Cell

- ```
MWXDb_Get_CellInst_cellInstMasterId(cellId,
cellInstId,
&cellInstMasterId);
```

- ```
MWXDb_Get_CellInstMaster_cellName(cellId,  
cellInstMasterId,  
&cellNameString);
```

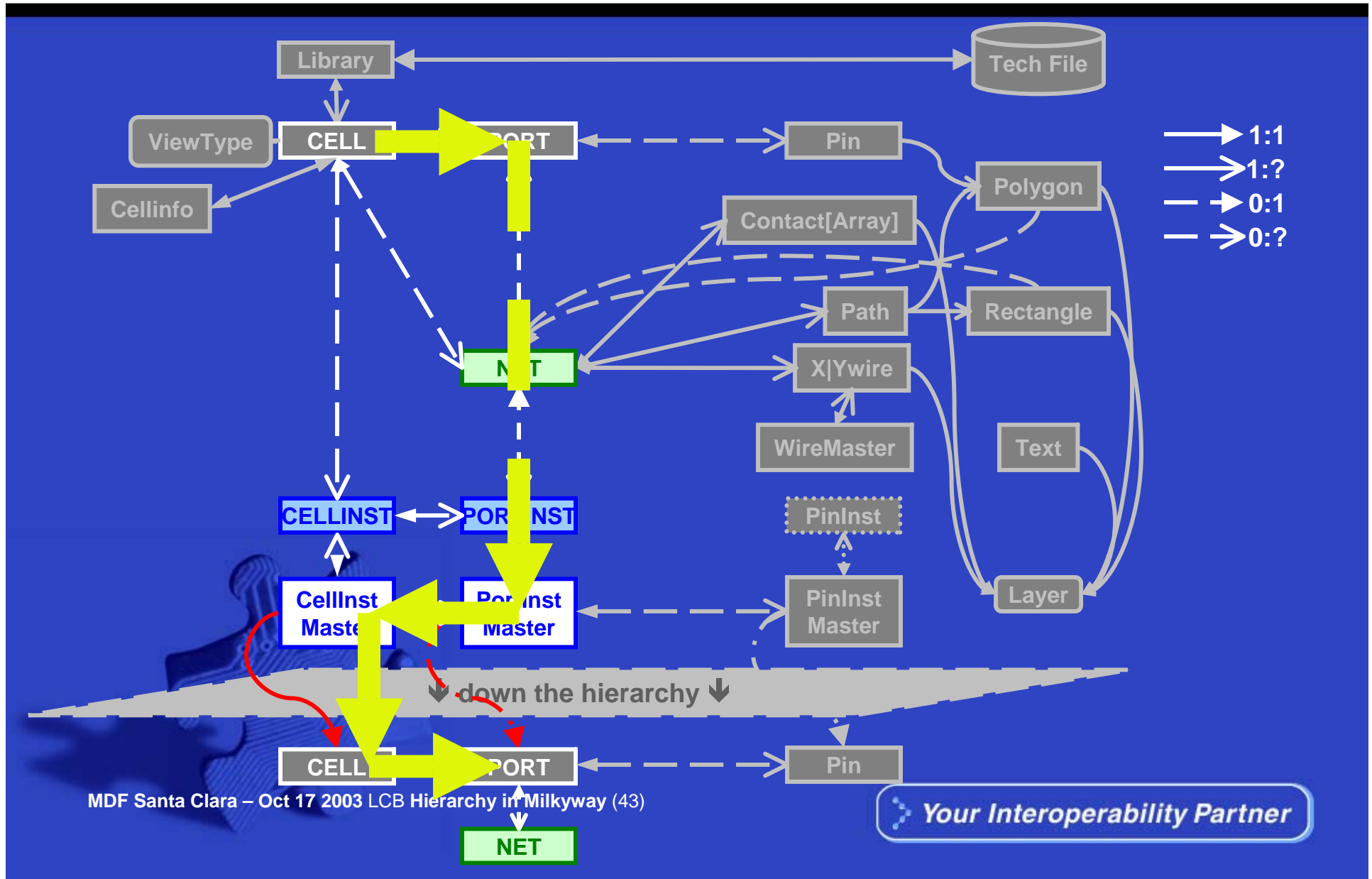
- ```
MWXDb_Open_Cell (cellNameString, accessMode,
&childCellId);
```

# In What Library is a Given Cell?



- The previous slide shows traversal from Cell Instance to the defining Cell – WHERE IS THAT CELL?
- The binding of Cell Instance to its defining Cell...
  - is really by name
  - to the first matching Cell name in a series of libraries
- **MWXDb\_Open\_Cell** opens the first matching cell
- If you need to know the actual library from which it is opened then two more functions are used:
- From the Cell, the Library Id of the specific library from which it is opened is available  
**MWXDb\_Get\_LibId\_ByCellId**
- **MWXDb\_Get\_LibName\_ById** can then get the name of that library

# Hierarchy Traversal – Port to Port



# Cell to Port to Net to PortInst [Master] **SYNOPSYS**<sup>®</sup>

- Cell to Port

```
MWXDb_Get_Port_By_Name(cellId, &portNameString,
 &portId)
```

- Port to Net

```
MWXDb_Get_Port_netId(cellId, portId,
 &netId)
```

- Net to PortInst

```
MWXDb_Get_NetPortInst(cellId, netId,
 &numberOfPortInsts,
 &portInstIdArrayPointer);
```

- PortInst to PortInstMaster

```
MWXDb_Get_PortInst_portInstMasterId(cellId,
 portInstId,
 &portInstMasterId);
```

(continued on next page ...)

- PortInstMaster to CellInstMaster and Port Name

```
MWXDb_Get_PortInstMaster_cellInstMasterId(cellId,
portInstMasterId,
&cellInstMasterId);
```

```
MWXDb_Get_PortInstMaster_name(cellId,
portInstMasterId,
&portNameString);
```

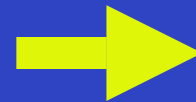
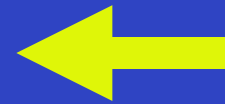
- CellInstMaster to Cell Name to Cell

```
MWXDb_Get_CellInstMaster_cellName(cellId,
cellInstMasterId,
&cellNameString);
```

```
MWXDb_Open_Cell(cellNameString, accessMode,
&childCellId);
```

- Cell to Port

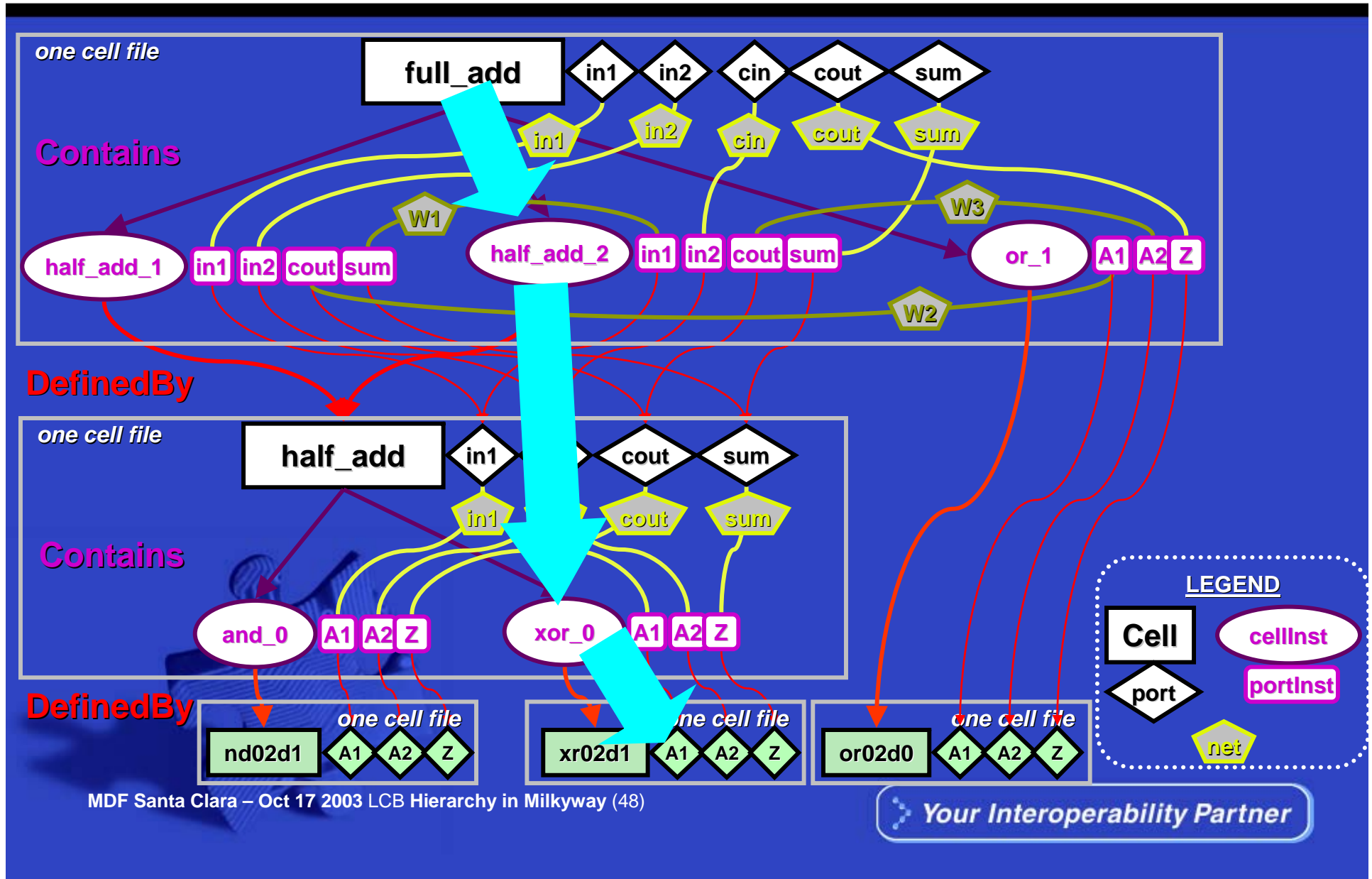
```
MWXDb_Get_Port_By_Name(childCellId,
&portNameString,
&childPortId);
```



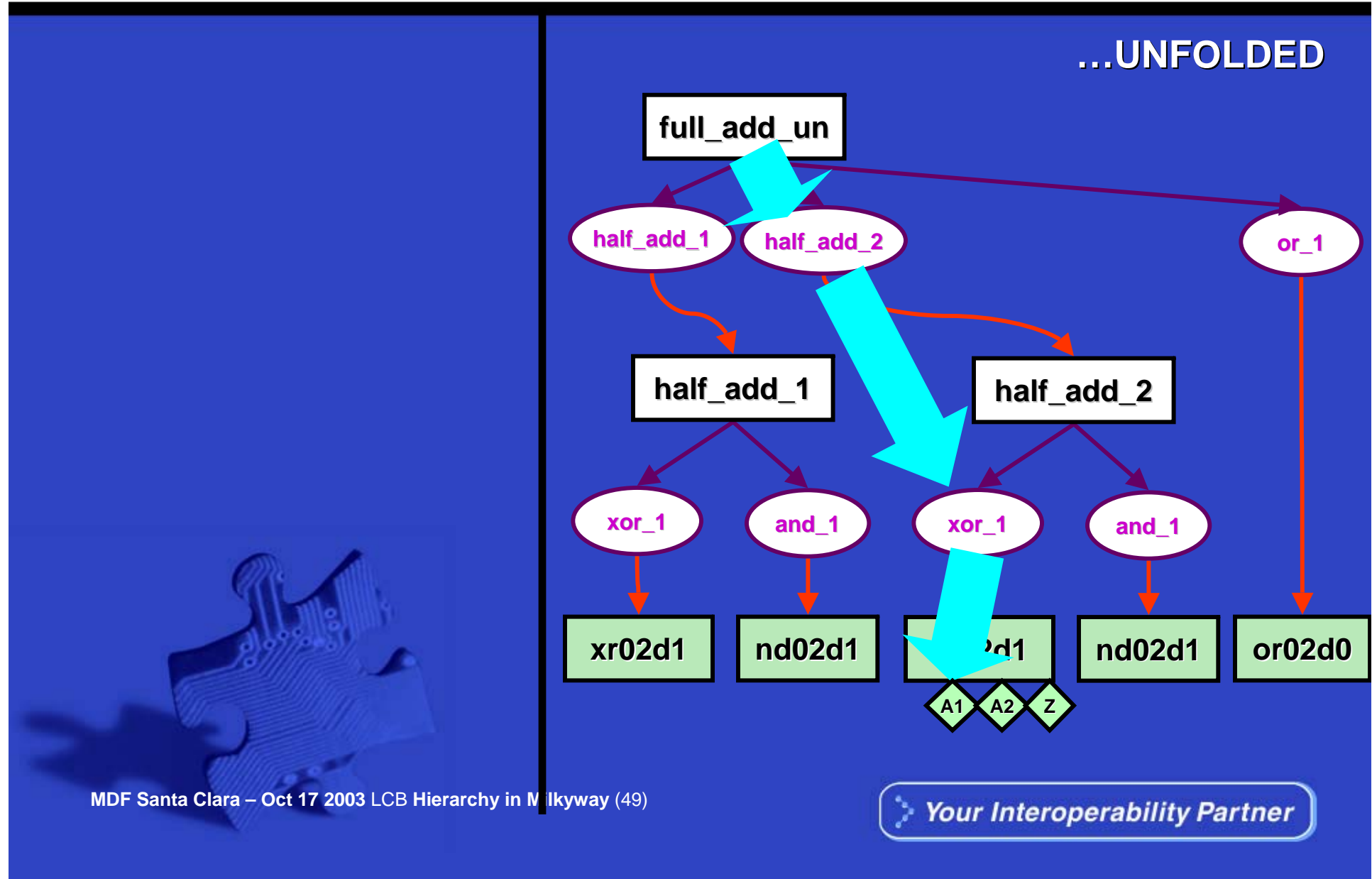
- Even in a folded representation...
- A *hierarchical pathname* can be constructed to allow reference to a specific *occurrence*
- Starting at the “top cell” a series of instance names can identify a specific path through an unfolded hierarchy.



# Instance path to port example...



# Unfolded provides unique path to leaf **SYNOPSYS**<sup>®</sup>



# Naming is also used to track Buses



- Milkyway ports and nets are “scalar” – one bit
- Consider the following Verilog with buses...

```
module ripple_carry_adder (cin , sum , cout , in1 , in2);
input cin ;
input [3:0] in1 ;
input [3:0] in2 ;
output [3:0] sum ;
output cout ;
```

```
full_adder full_adder_1 (.cout (w0) , .sum (sum[0]) , .cin (cin) ,
 .in2 (in2[0]) , .in1 (in1[0])) ;
full_adder full_adder_2 (.cout (w1) , .sum (sum[1]) , .cin (w0) ,
 .in2 (in2[1]) , .in1 (in1[1])) ;
full_adder full_adder_3 (.cout (w2) , .sum (sum[2]) , .cin (w1) ,
 .in2 (in2[2]) , .in1 (in1[2])) ;
full_adder full_adder_4 (.cout (cout) , .sum (sum[3]) , .cin (w2) ,
 .in2 (in2[3]) , .in1 (in1[3])) ;
endmodule
```

# Stored as scalars with these names... **SYNOPSYS**<sup>®</sup>

```
i nput \i n1[0] ;
i nput \i n1[1] ;
i nput \i n1[2] ;
i nput \i n1[3] ;
```

```
i nput \i n2[0] ;
i nput \i n2[1] ;
i nput \i n2[2] ;
i nput \i n2[3] ;
```

```
output \sum[0] ;
output \sum[1] ;
output \sum[2] ;
output \sum[3] ;
```

- Allows recreation of bus notation when output again as Verilog

# Hierarchy in Layout



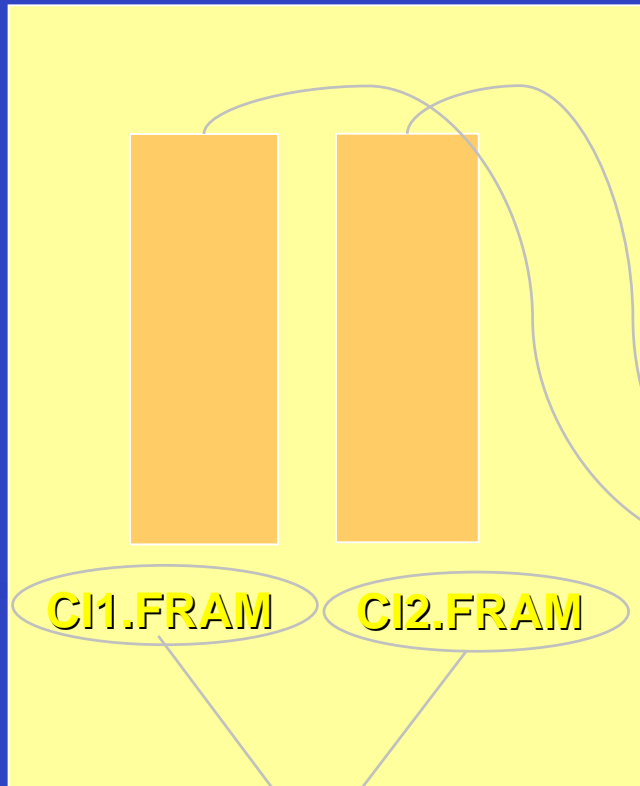
- Cell / Pins
- Cell Instance / Pin Instances
- Cell Instance View Type
  - A Cell can reference any view type for the Cell Instance
  - Astro uses the FRAM view for Cell Instances – this is an Astro specific abstraction of the cell and pins appropriate for Astro's Place and Route algorithms



# Milkyway Physical Cell Hierarchy



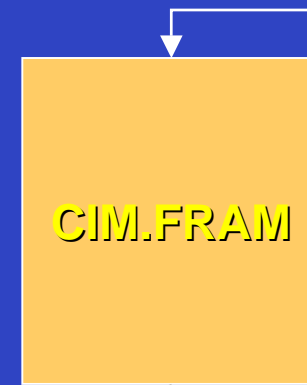
## TOP.CEL



The Cell Instances in the top cell reference the FRAM view of the defining Cell.

MDF Santa Clara – Oct 17 2003 LCB Hierarchy in Milkyway (53)

FRAM view represents only blockage, pins, and vias. Cell Instances can not be seen from this view



## CIM.CEL



Only CEL views show the lower level Cell Instances

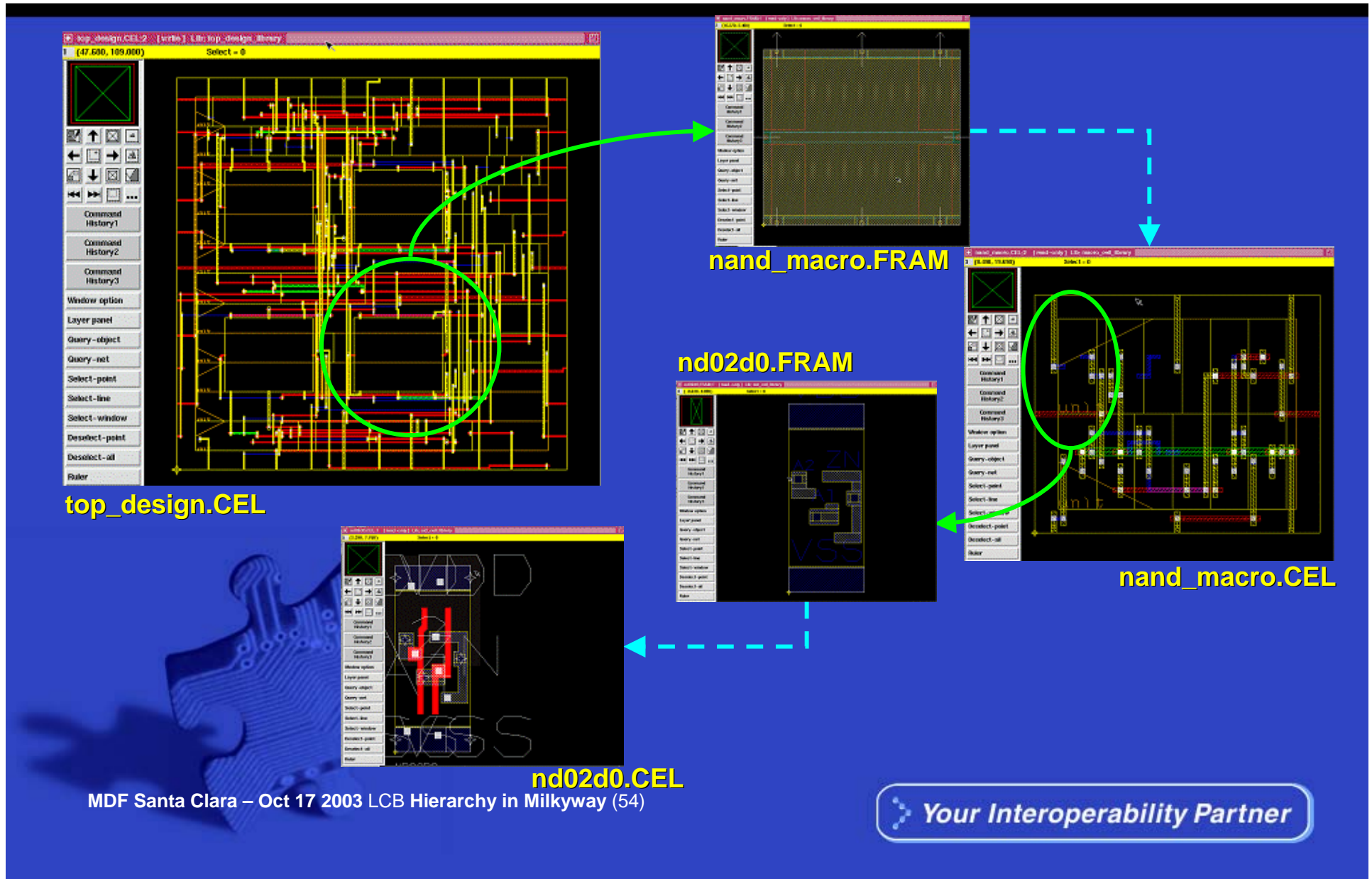


At the bottom of the hierarchy the cells have no cell Instances in the CEL view.

 Your Interoperability Partner

# Milkyway Physical Cell Hierarchy example design

SYNOPSYS®



# Logical versus Physical



- Tracking logical vs. physical hierarchy and connectivity is an important function as a design progresses from concept through completion.

*BUT...*

- This is not done by Milkyway
- It is done by tools that run on Milkyway

**The End**

**SYNOPSYS®**

**Thank you!**



MDF Santa Clara – Oct 17 2003 LCB Hierarchy in Milkyway (56)

 *Your Interoperability Partner*