LEF Comparison Tool

Billy Cody Hacker

Camden Penn

EE 584

LEF Files

- **Library Exchange Format (LEF)**
  - Human readable and editable
  - Plaintext document
- **LEF files are files used to store the positions of pins, obstructions, and properties of a circuit**
  - It only stores what is needed to place and route cells
  - Pins are the inputs and output pins from the respective cell
  - Obstructions list parts of metal layers that are used internally by the cell and cannot be used in routing

LEF File Hierarchy

- Cell
  - Cell Properties
  - Pins
    - Pin Properties
    - Port
      - » Layers
        - Coordinates of Layers
  - Obstructions
    - Layers
      - » Coordinates of Layer Obstructions

LEF Sample

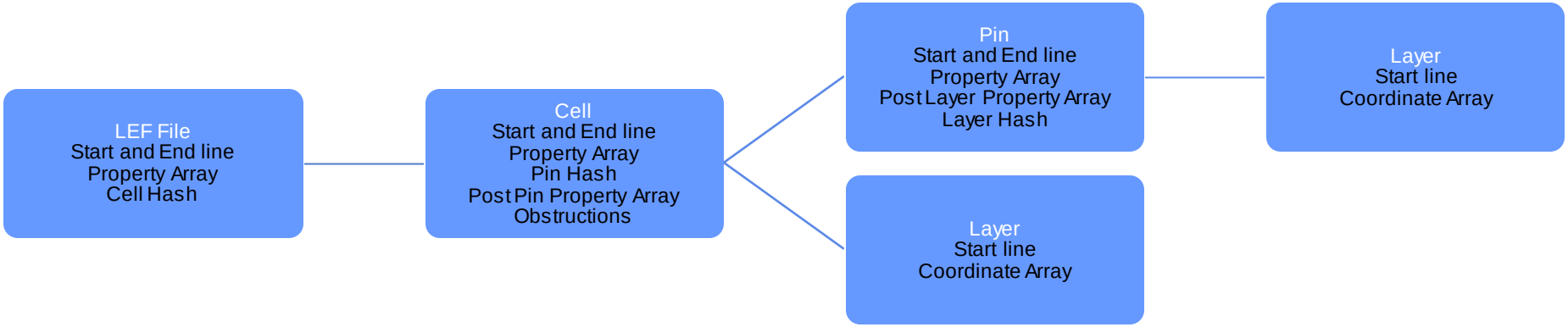
```
MACRO and2_16x
  CLASS CORE ;
  ORIGIN 0 0 ;
  FOREIGN and2_16x 0 0 ;
  SIZE 11.02 BY 2.32 ;
  SYMMETRY X Y ;
  SITE unit ;
  PIN A1
    DIRECTION INPUT ;
    USE SIGNAL ;
    ANTENNAMODEL OXIDE1 ;
      ANTENNAGATEAREA 0.64 LAYER M1 ;
    PORT
      LAYER M1 ;
        RECT 2.685 1.085 3.845 1.235 ;
      LAYER PC1 ;
        RECT 2.785 1.115 2.875 1.205 ;
        RECT 3.075 1.115 3.165 1.205 ;
        RECT 3.365 1.115 3.455 1.205 ;
        RECT 3.655 1.115 3.745 1.205 ;
    END
  END A1

  OBS
    LAYER M1 ;
      RECT 0.2 0.27 0.3 0.75 ;
      RECT 0.2 0.65 1.59 0.75 ;
      RECT 0.65 1.62 0.75 2.05 ;
      RECT 0.65 1.62 1.59 1.72 ;
      RECT 1.49 0.65 1.59 1.72 ;
      RECT 1.49 1.14 2.461 1.24 ;
  END
END and2_16x
```

LEF Comparison Tool

- LEF data is generated in a random order
  - Makes diffs between LEF files practically useless
  - Identical source files can generate very different LEF files
- LEF files can be sorted
  - No effect on data stored
  - Sorted LEF files can be diffed
- Secondary goal is to implement some basic error checks
  - Space before semicolon
  - Consistency with liberty files
    - Liberty files are an IEEE standard used to store PVT information and input/output characteristics

- **Starting Point**
- **First, a definition**
  - **In this presentation, 'Tier' refers to any one Cell, Pin, Layer, or Obstruction.**
- **Initial Ruby script provided**
  - Sorts within each Layer in the LEF file
  - Other tiers of data are left unsorted
- **Intended object-oriented structure**
  - Each Tier is described by a class.
  - These classes all have the following functions defined:
    - is_start_line
    - initialize
    - sort
    - print
  - Creating a new Tier calls initialize(file, index), which parses the file from the location noted in index until it recognizes an end delimiter. Attempting to initialize a Tier with a bad starting location raises an error.
  - A call to sort() on a Tier will call sort() for all child Tiers, and then sort the properties of the original Tier.

Class Structure



How to Use a Class Hierarchy

- Each Tier knows how it works
- Need a lower Tier's functionality? Ask it!

```
def sort()
  # TODO: Sort properties list [dangerous!]
  @pins.each_value{|pin| pin.sort()}
  if(!@obstructions.nil?)
    @obstructions.sort()
  end
end
```

Figure 1. ***The Cell tier's sort() function.***

Parsing

- Basic Structure
  - Store properties relating to current tier in array
  - check for the start line of the next tier
    - When a start line is found create a new instance of the appropriate class
    - Let the new instance parse that element
  - Errors are also checked at this stage
  - Return when the end line is found for the current object
    - Parent element continues parsing
- Shown is main parsing loop for a pin

```
252     while !(line=~/^\s*END #{Regexp.quote(@name)}/)
253       if (line=~/^\s*PORT/) && !found_port
254         @port_start_line = line
255         found_port = true
256         index.value += 1
257       elsif found_port && Layer::is_start_line(line)
258         new_layer = Layer.new(file, index, errors, debug_mode)
259         @layers[new_layer.name] = new_layer
260       elsif (line=~/END/) && found_port
261         @port_end_line = line
262         found_port = false
263         index.value += 1
264       else
265         if(@debug_mode)
266           puts (index.value + 1).to_s + ": found pin property " + line
267         end
268         if(line=~/\S;$/)
269           error_msg = (index.value + 1).to_s + "\n"
270           @errors[:line_ending_semicolons].push(error_msg)
271           line = line.gsub(/;$/, " ;")
272         end
273         # if (line=~/END/)
274         #   raise "Off the rails!"
275         # end
276
277         if line=~/^\s*PROPERTY/
278           @keywordProperties.push(line)
279         elsif !(line=~/^$/)
280           @properties.push(line)
281         end
282         index.value += 1
283       end
284       line = file[index.value]
285       line.chomp
286     end
287     @end_line = line
288     index.value += 1
289   end
```

Sorting and Printing

- The file is sorted as it is printed to ensure the correct order
- Basic structure
  - Start line is first printed
  - Sort and print properties relating to current tier
    - Some are printed after next tiers are printed
  - Call print for each instance from next tier in sorted order
  - Print end line
- Shown is the cell print method

```
186   def print(outFile)
187     outFile.print @start_line
188
189     @properties = @properties.sort_by { |e|
190       words = e.split
191       @@PropertyOrder.index(words[0].upcase)
192     }
193     @properties.each do |line|
194       outFile.print line
195     end
196     sortedPinKeys = @pins.keys.sort()
197     sortedPinKeys.each do |key|
198       @pins[key].print(outFile)
199     end
200     if(!@obstructions.nil?)
201       @obstructions.print(outFile)
202     end
203     @keywordProperties = @keywordProperties.sort
204     @keywordProperties.each do |line|
205       outFile.print line
206     end
207
208     outFile.print @end_line
209     outFile.print "\n"
210   end
```

Property Sorting

- Some properties require a specific order
  - We sort these according to the first word
  - The correct order was obtained from the LEF File Reference
  - Shown is the array that holds the correct order and the code that sorts the properties

```
221  @@PropertyOrder = [
222    "TAPERRULE", "DIRECTION", "USE", "NETEXPR", "SUPPLYSENSITIVITY",
223    "GROUNDSENSITIVITY", "SHAPE", "MUSTJOIN", "PROPERTY",
224    "ANTENNAPARTIALMETALAREA", "ANTENNAPARTIALMETALSIDEAREA",
225    "ANTENNAPARTIALCUTAREA", "ANTENNADIFFAREA", "ANTENNAMODEL",
226    "ANTENNAGATEAREA", "ANTENNAMAXAREACAR", "ANTENNAMAXSIDEAREACAR",
227    "ANTENNAMAXCUTCAR"
228  ]
```

```
296      @properties = @properties.sort_by { |e|
297        words = e.split
298        @@PropertyOrder.index(words[0].upcase)
299      }
```

Coordinate Sorting

- Special case of sorting
- Example

```
RECT 0.345 0.986 1.19 1.086 ;
RECT 1.09 0.44 1.19 1.63 ;
RECT 2.16 0.22 3.065 0.32 ;
RECT 2.965 0.22 3.065 0.6 ;
```

- Requires custom method
  - Sort first word alphabetically
  - Then sort number numerically
  - Method shown takes 2 keys and returns how they should be compared
    - Used in ruby sort method

```
376  def self.coordSort(a, b)
377    aspl = a.split()
378    bspl = b.split()
379    result = 0
380    #compare word part
381    if aspl[0] != bspl[0]
382      result = aspl[0] <=> bspl[0]
383    else
384      #compare each coordinate as a float
385      aspl.zip(bspl).each do |aword, bword|
386        af = aword.to_f()
387        bf = bword.to_f()
388        if af != bf
389          result = af <=> bf
390          break
391        end
392      end
393    end
394    return result
395  end
```

Error Checking

- Error checking is needed
  - There can be minor mistakes made while editing the files
  - Other mistakes can be made during cell creation in cad tools
  - The mistakes can be very difficult to detect via existing means
    - Tools may not give descriptive error messages
    - The files are very large so manual inspection is often impossible

Space Before Semicolon Check

- LEF files require a space before the semicolon
- The code on the right shows we handle this case
  - The space is added to the stored line for later use
  - The change is logged in an array for later printing

```
268         if(line=~/\S;$/)
269           error_msg = (index.value + 1).to_s + "\n"
270           @errors[:line_ending_semicolons].push(error_msg)
271           line = line.gsub(/;$/, " ;")
272         end
```

Origin and Foreign Check

- Values for both are expected to be "0 0"
  - Other values will raise a warning
  - Not corrected because other values aren't necessarily incorrect
- Shown on the upper right is code that adds to error array
  - This array will later be used to print errors
- Also shown is sample error file

```
154           if line=~/ORIGIN (\d+ \d+)/
155             if $1 != "0 0"
156               @errors[:strange_origin].push(@name + "\n")
157             end
158           end
159           if line=~/FOREIGN [\w\d_]+ (\d+ \d+)/
160             if $1 != "0 0"
161               @errors[:strange_foreign].push(@name + "\n")
162             end
163           end
```

```
 1
 2 Warning: The following cells have an unusual FOREIGN specified.
 3 ------------------------------------------------------------
 4 and2_2x
 5 ------------------------------------------------------------
 6
 7 Warning: The following cells have an unusual ORIGIN specified.
 8 ------------------------------------------------------------
 9 and2_4x
10 ------------------------------------------------------------
```

Demonstration