Kicad Board Refining

Proposal Specification

SNH 20180822

Introduction

This document outlines the features and requirements for implementing board-level refinement in KiCad version 6.

Definitions

A **primitive** is a basic PCB/Schematic item, such as a wire, track segment, via, component, etc.

A **track** is a set of segments that define a copper connection between two pads or vias on a board.

An **unique ID** is a project-wide unique numerical identifier for each item.

A **generating item** is a primitive that is used to calculate a refinement.

A **generated item** is a primitive that is the result of a refinement on the generating items.

Data model

This proposal is about post-placement changes to a board items.

- Each primitive in the board model has an unique identifier (not an equivalent of Kicad's already existing timestamp, as timestamps are not guaranteed to be unique).
- Refinements are adjustments that are made after a primitive is placed on the board.
- The generated items are stored in the pcb file
- The generating items are stored in the pcb file
- The generated items and generating items belong to the same net.
- The generated items are always shown on the board and used in DRC and plotting output.
- The generating items are not shown on the board unless their id is also listed in the generated items list.

- Neither source nor generated items may include a refinement id as one of their elements

File format

Refinement element requirements

Refinement routines will require modification to existing elements in the s-expr board file. Specifically, graphical items including polygons, arcs, circles and bezier curves shall be permitted on the copper layers and optionally assigned a net.

Refinement definition

Each item in the s-expr file contains a property with an unique 64-bit unsigned integer, for example:

```
(refinement (id 12345678)
      (teardrop | lengthtuning | via_fill
            (parameters param1=x param2=y … param_last=z)
      )
      (source_members id_1 id_2 … id_last )
      (generated_members id_1 id_2 … id_last )
)
```

Refinement elements

1. `teardrop` or `lengthtuning` or `via_fill` -- Each refinement can only be a single operation.
    a. Note that additional refinements would be extended here. These could include differential pair handling, length matching (skew) and others.
    b. All refinements take input (source) and output (generated) elements from the board.
2. `source_members` -- The list of ids given elsewhere in the kicad_pcb file that denote elements used in the generation of the refinement.
    a. In the case of **teardrops**, these elements are duplicated to `generated_members` and therefore remain visible and editable.
    b. In the case of **length tuning**, these elements are removed from the displayed board. Elements that are not used in the length tuning will be added to `generated_members`.
    c. In the case of **via fills**, this is a single zone that exists on at least one copper layer and a single via that exists on at least one of the source layers.

3. `generated_members` -- The list of ids given elsewhere in the kicad_pcb file that denote elements created by the refinement operation. Generated members and source members may include the same id where the primitive is both used in the calculation and to be displayed in the result.
    a. In the case of teardrops, these are polygons and the source members used to calculate them.
    b. In the case of length tuning these are track segments.
    c. In the case of via fills, these are vias and the source zone.

Teardrop definitions

Teardrops are represented by a sub-entity in the s-expr file, e.g.:

```
(teardrop
      (parameters
            style=curved|straight|keyhole
            drc=follow|ignore
            shrink=yes|no
            via=yes|no
            via_width=float percent
            via_height=float percent
            smd=yes|no
            smd_width=float percent
            smd_height=float percent
            track=yes|no
            track_width=float percent
            junction=yes|no
            junction_width=float percent
            junction_height=float percent )
)
```

Parameters

1. `style` -- This parameter is either straight or curved.
    a. A straight teardrop will draw a straight line from the start to the end point as defined by the width/height percentages
    b. A curved teardrop uses a spline to match the slope at the beginning and end of the teardrop
    c. A keyhole teardrop places a second circular copper element at the junction between the landing pad and trace
2. `drc` -- This controls whether the teardrop is created if doing so would violate the DRC. Note that this does not apply to zones, only tracks, pads and vias.

3. `shrink` -- This controls whether we allow the size of the teardrop to adjust downward to fit in the given space.
    a. If shrink is “yes”, the width and height values may be adjusted down so that they do not conflict with other items in the board. Both width and height will be adjusted at the same time.
    b. If shrink is “no” and the teardrop would conflict with other items in the board, it will not be generated if the drc parameter above is set to **follow**.
4. `via` -- Generate a teardrop where a track connects to a via or plated through-hole (PTH)
5. `via_width` -- Width of the teardrop as measured from the tangent of the via copper on the track’s layer to the point at which the teardrop meets the track copper. Width given in multiple of the via diameter. Via diameter is defined as the diameter of the enclosing circle.
6. `via_height` -- Height of the teardrop from as measured across the full teardrop perpendicular to the track. Height given in multiple of the via diameter.
7. `smd` -- Generate a teardrop where a track connects to an SMD pad.
8. `smd_width` -- Width of the teardrop as measured from the pad tangent to the point at which the teardrop meets the track copper. Width given in multiple of the track width.
9. `smd_height` -- Height of the teardrop from as measured across the full teardrop perpendicular to the track. Height given in multiple of the track width.
10. `track` -- Generate a teardrop connector between two track elements of differing widths.
11. `track_width` -- Width of the teardrop along the thinner track measured as a multiple of the thicker track width.
12. `junction` -- Generate a teardrop connector between two track elements that meet at 90° angles.
13. `junction_width` -- Width of the teardrop along the continuing track measured as a multiple of the continuing track width.
14. `junction_height` -- Height of the teardrop along the terminating track measured as a multiple of the continuing track width. Height measured from the edge of the continuing track copper to the point at which the teardrop contacts the terminating track copper.

Implementation notes

Teardrops should be generated as polygons (not filled zones)

Dragging a track that has teardrops in the dragged segment should remove the teardrops while dragging and replace them once the drag has completed.

A teardrop may at no time be wider than the perpendicular projection of the copper to which it connects.

If new track segments are created between the first two non-source track elements by a drag operation, then the refinement operation will include them in the source and destination member structures.

Dragging or drawing a track against a teardrop shall cause a collision and the associated action depending on the router mode.

Length tuning definitions

Length tuning parameters are represented as a sub-entity of the refinement element as follows

```
(length_tuning
      (parameters
            drc=follow|ignore
            target_length=integer nm
            minimum_length=integer nm
            maximum_length=integer nm
            maximum_amplitude=integer nm
            variable_height=yes|no
            spacing=integer nm
            style=arc|45
            miter_radius=integer percent
            pad_length=yes|no
      )
)
```

Parameters

1. `drc` -- This sets whether drc rules be followed when generating the new track.
    a. If set to `follow` and the generated track cannot fit with any parameter combination, the tuned track will not be generated.
    b. If set to `ignore`, the tuned track will be generated with the fewest possible DRC violations.
2. `target_length` -- Length in nm for the ideal length of the new track.
3. `minimum_length` -- Smallest acceptable length of the new track
4. `maximum_length` -- Longest acceptable length of the new track
5. `maximum_amplitude` -- Maximum amplitude in nm measured from the center line of the generating track along the axis perpendicular to the generating track to the farthest offset of the generated track.
6. `variable_height` -- If yes, allow the length tuning to select smaller track meander heights for some of the meanders to meet length or DRC requirements.
7. `spacing` -- Distance in nm from between the centerlines of the first two perpendicular generated tracks.

8. `style` -- Determines whether the meander head is generated as an arc or a chamfered track with 45° joining segments
9. `miter_radius` -- The distance from along the generating track axis from the centerline of the perpendicular segment to the first parallel segment of the generated track meander.
10. `pad_length` -- If yes, the track length calculation takes into account the internal track width of the connecting pads (if given by the footprint).

Implementation notes

Dragging a length-tuned refinement element should update the refinement as it is dragged. If this proves to be not performant, then we should have an optional selection to update refinement only when dragging stops or only when explicitly requested.

Dragging a track may modify the source and destination elements of the refinement. The additional source elements shall only include elements that are contained between the first two, previously non-contained elements.

Dragging while in shove mode should keep the original refinement to the extent possible while adding a removing meanders to maintain target length. Meanders should only be added/remove in the section of track being dragged.

Dragging while in walk around mode should attempt to create/remove meanders along the entire length to satisfy length requirements.

Dragging a refinement that is properly tuned shall be prevented if the action of dragging it will require it to de-tune. This limit shall be optional. If the action is prevented, there will be a textual indicator of why the action is disallowed and the limiting elements will be highlighted.

Dragging a track that is not properly tuned shall be allowed if the new position improves on the original tuning. This limit shall be optional.

While dragging a length tuning refinement, there shall be a graphical bar indicator showing target length, current length and allowable deviation limits. This graphical bar shall be updated as the user adjusts the refinement.

Via Fill definitions

Via fill parameters are represented as a sub-entity of the refinement element as follows

```
(via_fill
      (parameters
            drc=follow|ignore
```

```
            x_spacing=integer nm
            y_spacing=integer nm
            clearance=integer nm
            stagger=none|horizontal|vertical
            offset_pattern=(
                  off_2=integer nm
                  off_3=integer nm
                   …
                  off_n=integer nm )
            override_netclass=yes|no
      )
)
```

Parameters

1. `drc` -- This sets whether drc rules be followed when generating the vias.
    a. If set to `follow` and the generated via cannot fit in its assigned locate, the via will not be generated.
    b. If set to `ignore`, the vias will be generated in regardless of the intersections with other elements on the board.
2. `x_spacing` -- This sets the center-to-center x-distance in nanometers of the via fill.
3. `y_spacing` -- This sets the center-to-center y-distance in nanometers of the via fill.
4. `clearance` -- This sets a minimum required radial clearance in nanometers between the via copper and other board elements. This does not override netclass requirements by default.
5. `stagger` -- This allows either the rows or columns of the via fill to be staggered by an offset pattern.
6. `offset_pattern` -- This sets sequential offsets in nanometers for either the rows or columns of the fill. The pattern starts with the offset between the first and second row or column. It then takes (optionally) the offset between the second and third row or column. The pattern will repeat with the n+1th row or column being placed at offset=0 from the zero-th row or column.
7. `override_netclass` -- If yes, then the vias' clearance is set to the value in `clearance`. If no, then the vias' clearance is set to the maximum of `clearance` and the appropriate netclass clearance value.

Implementation notes

The fill is tied to one zone at a time. Editing the via fill is done using a button in the zone properties dialog that brings up the via fill dialog.