A Proposal on Symbol Pin Mapping

for KiCad v6 or beyond

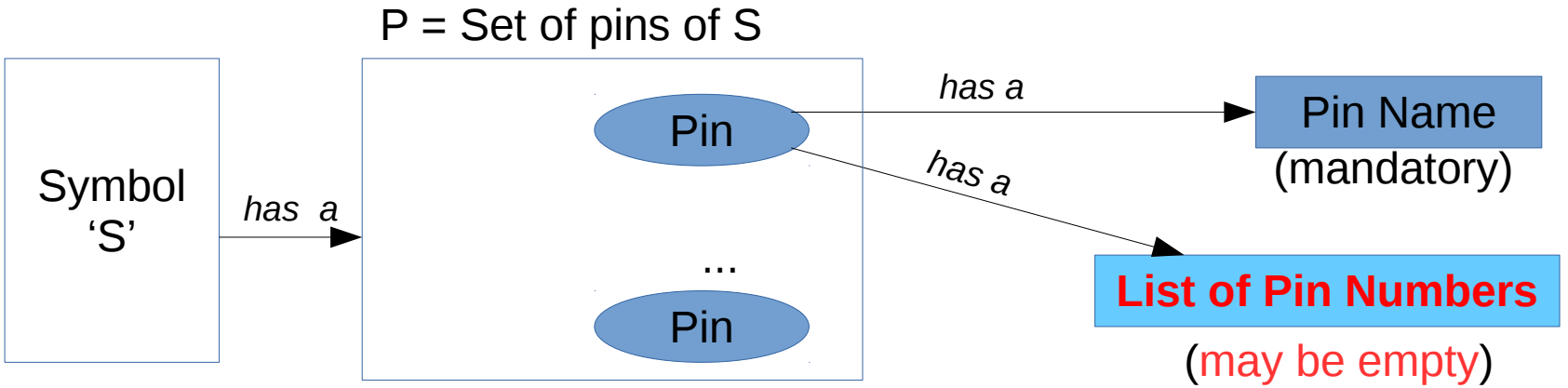
Ajith N.
@EL84 [KiCad Forum]
ajith@zoidlabs.com
27 Aug 2019

Summary of Key Points
(as related to the Data Model)

- Proposal to modify / extend / replace physical “pin number” with “list of pin numbers”
  - Backward compatible: single pin = list of one item
  - Other benefits are discussed
- To elevate pin mappings into ‘proper’ entities
  - As assets living in libraries
  - (This is already in developers’ plans in some form, AFAIK)

In the following slides, highlights are marked in red.

Symbols, Pins, Pin Identification

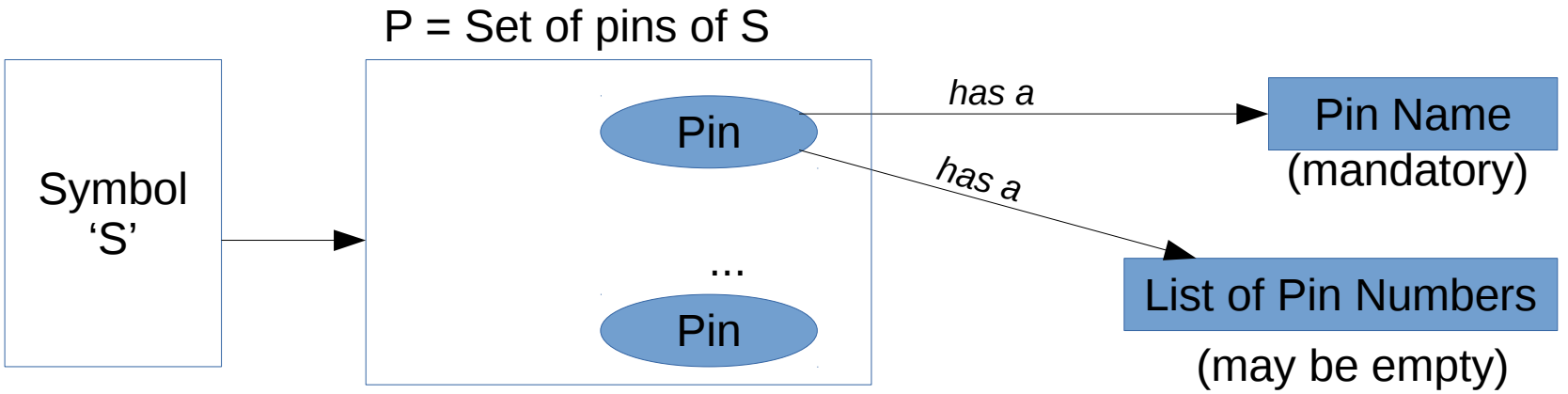


A symbol S has a set of pins ("logical pins") P
Each pin:
- has a pin name (a.k.a logical pin name)
- has a **list of pin (physical pin/pad) numbers** – (could be an empty list)
- has many other attributes (for rendering, electrical class etc) – ignored for now

Rationale #1: Associating a logical pin with a **list** of physical pin numbers (as opposed to exactly one pin), reduces schematic clutter.
(Further, benefits of an empty list are developed later on)

(We have ignored for now the other attributes of a symbol; such as the structure of units, footprint information, BoM attributes etc, for simplicity)

Constraints and Non-Constraints



P may be empty – i.e., a symbol which has no pins is also allowed.

A pin **must** have a Pin Name (name is mandatory)

Pin names do not need to be unique within a Symbol

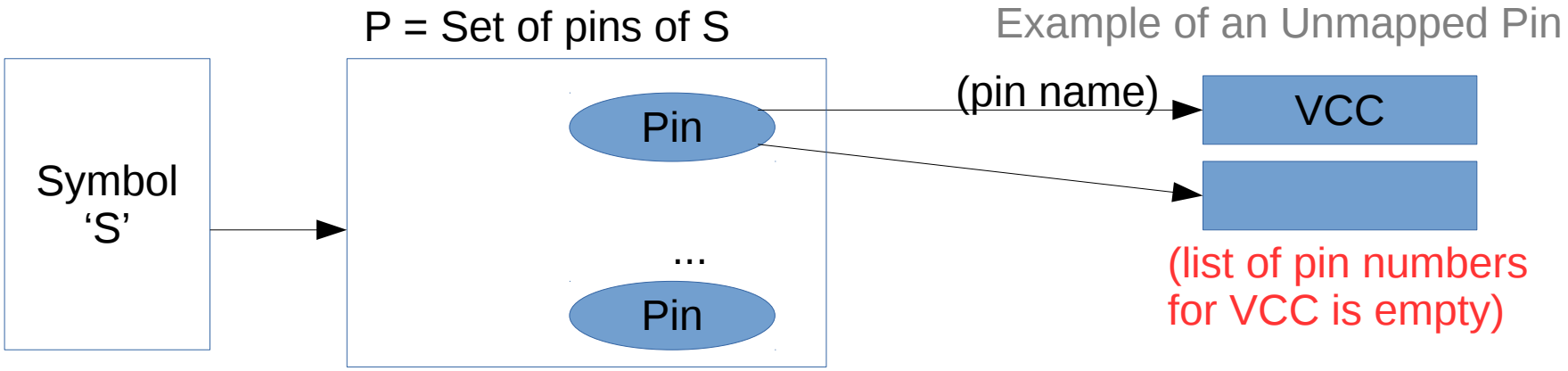
E.g. We may have two pins named GND.

Example of use: One will map to a large thermal pin, the other to a normal pin.
To specify thermally optimal layout, schematic designer can connecting them to different nets in the circuit, and give these nets different net class attributes. Also useful in multi-Unit cases.

Pin numbers **must** be unique within S. For example:

Pin VCC : `name = "VCC", pin-numbers = [2, 16]` … is in conflict with
Pin GND : `name = "GND", pin-numbers = [2, 10]` (as pin number 2 appears in both).

Unmapped Pins



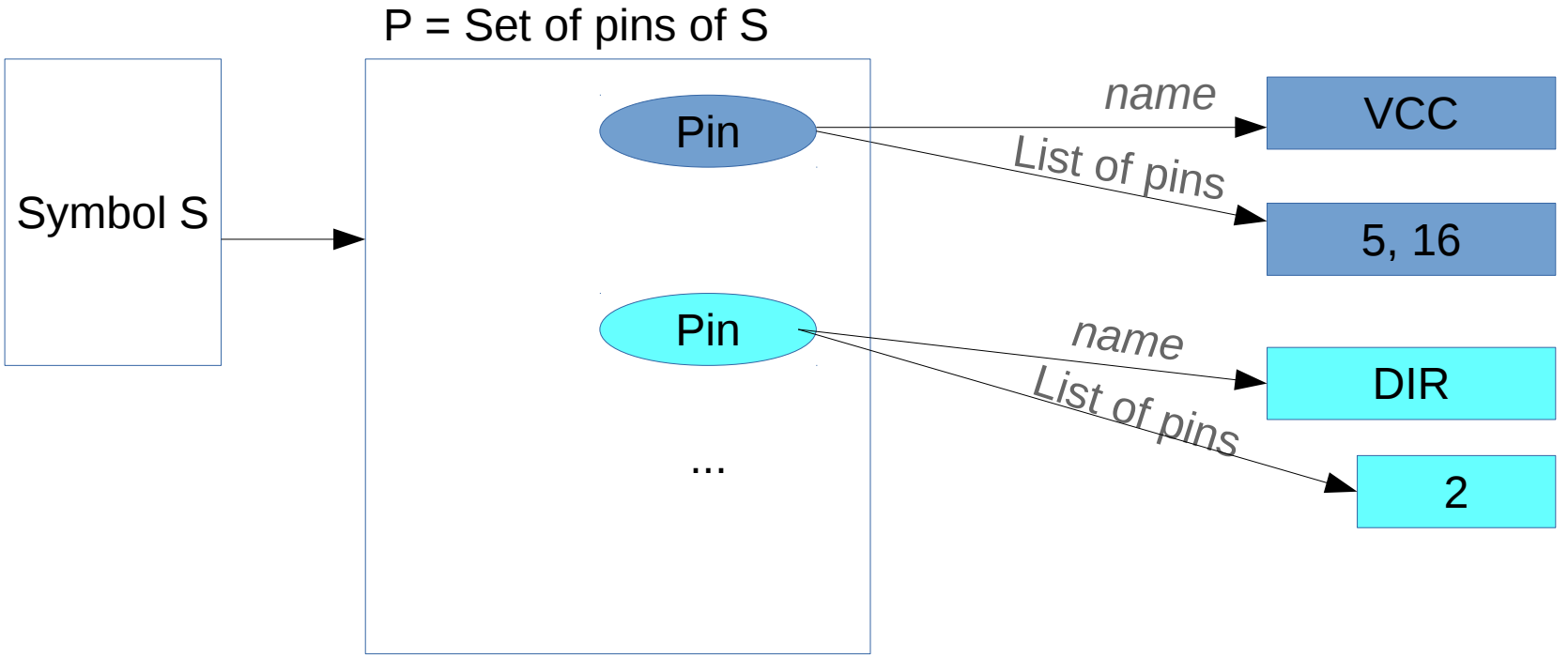
An <u>unmapped pin</u> is a pin without a pin number – i.e. its list of pin numbers is empty.

Unmapped pins are/were a "problem" because:
- They must generate ERC warnings/errors
- They will be unconnectable in Layout, and must generate connectivity errors

But, unmapped pins are **<u>worth tolerating</u>**... because:
- Circuit creation and schematic capture, simulation, part selection, etc are not hampered by the presence of unmapped pins in symbols.
- Pin Mapping is footprint dependent
- (Other use-cases supporting this notion will be developed in later slides)

Btw: in KiCad v5, setting pin number to "~" effectively makes the pin unmapped (but not much more can be done with it)

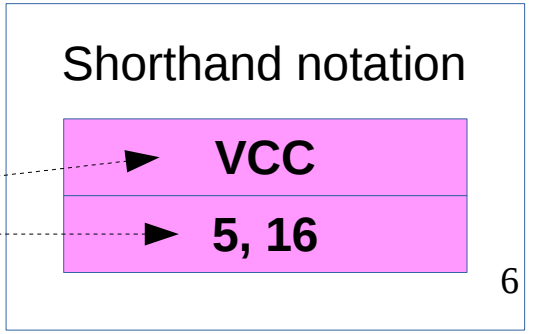
“Pin Mapping” as an entity on its own



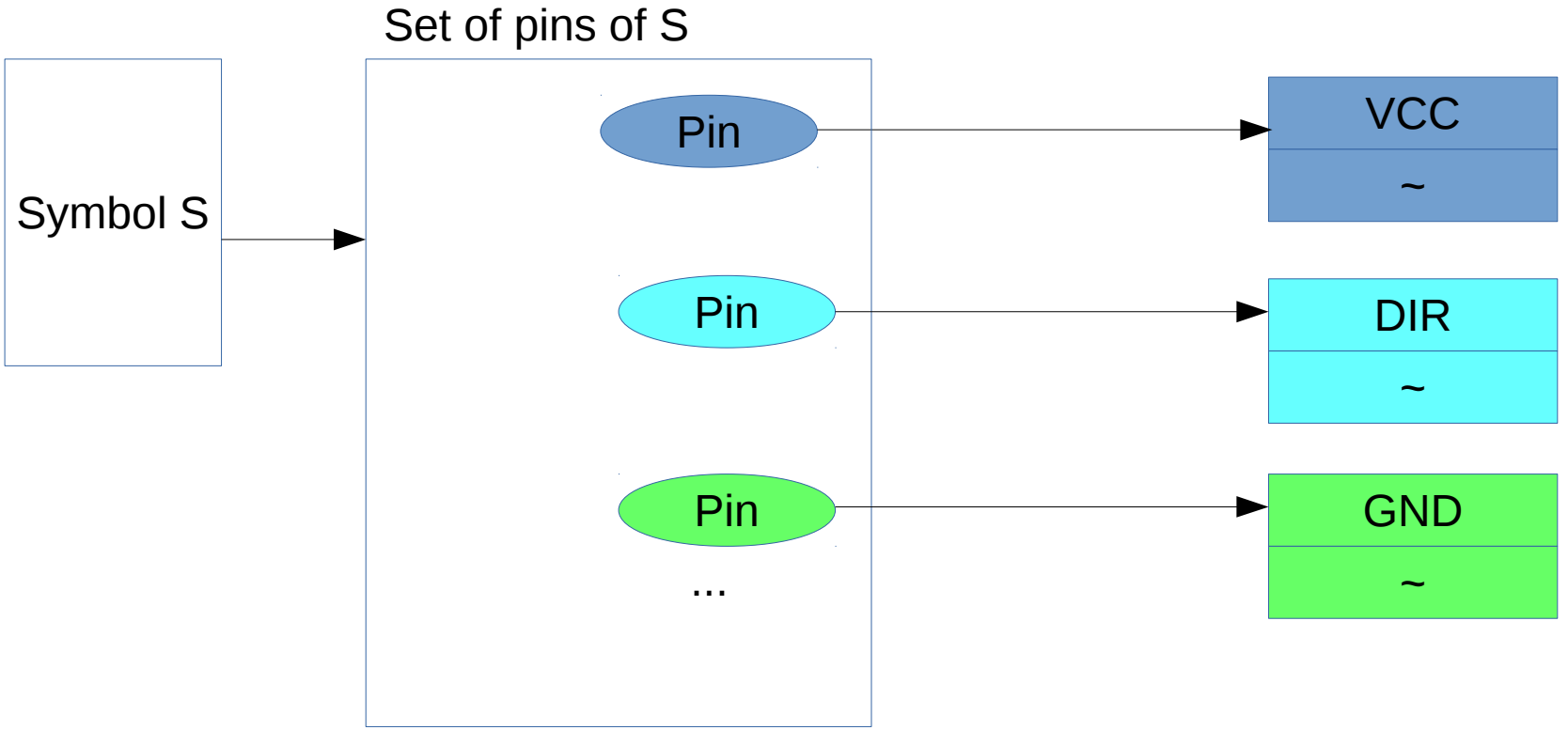
A pin mapping associates a pin name to a list of pin numbers.

e.g. VCC → pins 5, 16
DIR → pin 2
etc.

(Used hereafter)



Symbol with Unmapped pins



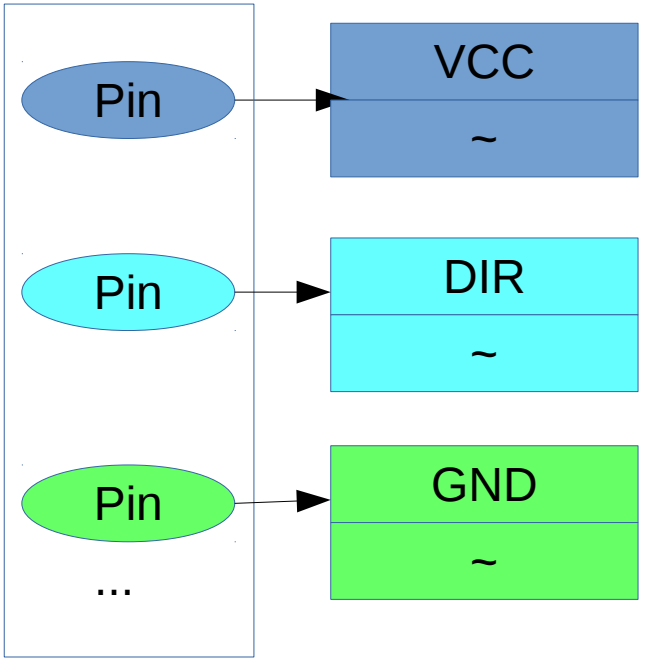
An unmapped symbol is one with one or more unmapped pins.
(Could be called a generic symbol, an abstract symbol or more precisely, a symbol with incomplete pin mappings)
(In contrast, a pin-committed symbol is one in which all pins are mapped).

Of what use, these unmapped symbols ???

Good (good enough) for -

Schematic work & Simulation

Allows early stage design tasks to proceed (Schematic review, refactoring, simulation, part/footprint selection etc)



What else?

#1. Unmapped symbols increase **symbol availability**.

E.g. a generic NMOS FET Symbol allows a new KiCad user to get started with schematics immediately.

(If an Atomic Part is already available and identified, symbol availability is not a matter of concern. Atomic part use-cases should be supported well without any issue. Beyond that, this discussion has very little impact on Atomic Part use-cases).

For example (details in the next slide): to use NMOS FET FDG311N in a schematic,
There is currently no library symbol that can be readily dropped in.
At the schematic stage, not having a symbol is a barrier.
Lack of availability of a suitable symbol is worth addressing at the root cause level.

Symbols Scarcity – Example

Claim: Unmapped symbols alleviate the issue of **symbol unavailability / scarcity**.

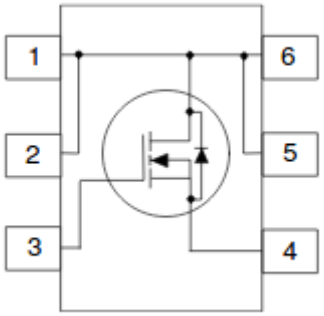
Example

Consider Joe, a KiCad user who is not yet proficient in creating library assets.

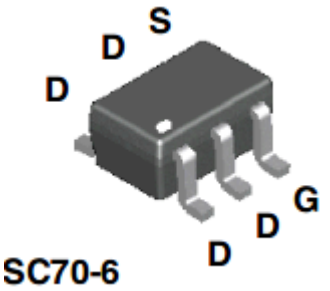
Joe wants to use an NMOS FET FDG311N (Fairchild) in a design because it is a good fit for his needs.

Currently there exists no Atomic Part for any FDG311N device.

FDG311N comes with only one footprint.



(symbol graphic exactly as in datasheet)



Symbols Scarcity – Example

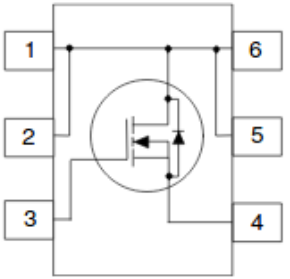
(Claim: Unmapped symbols increase **symbol availability)**.

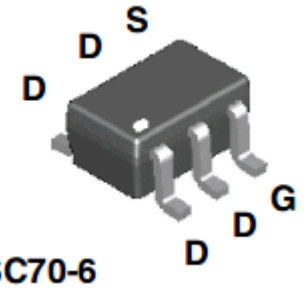
Pin mappings for this footprint are:
FDG311N: { D → [1,2,5,6] ; G → [3]; S → 4 }

No suitable symbols are available. Why?
*(Hint: **Not** because library creators were lazy or lax)*

There are at least 8 NMOS symbols in the “Device” library.

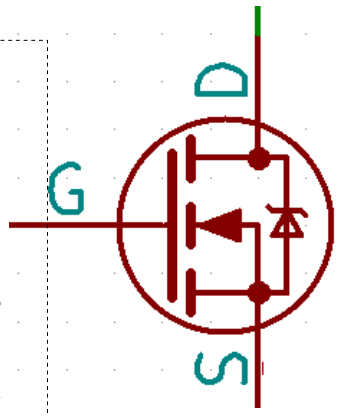
But none are suitable for FDG311N. Why?

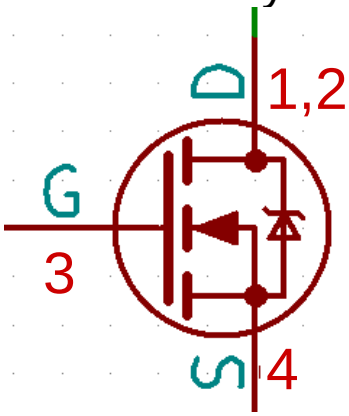




SC70-6

A desired unmapped (generic) symbol, good enough for schematic work





A highly desirable pin-committed symbol, layout-ready

*Shows how associating a **list of pins** with a single pin name allows re-use of the symbol graphic and reduces visual clutter*

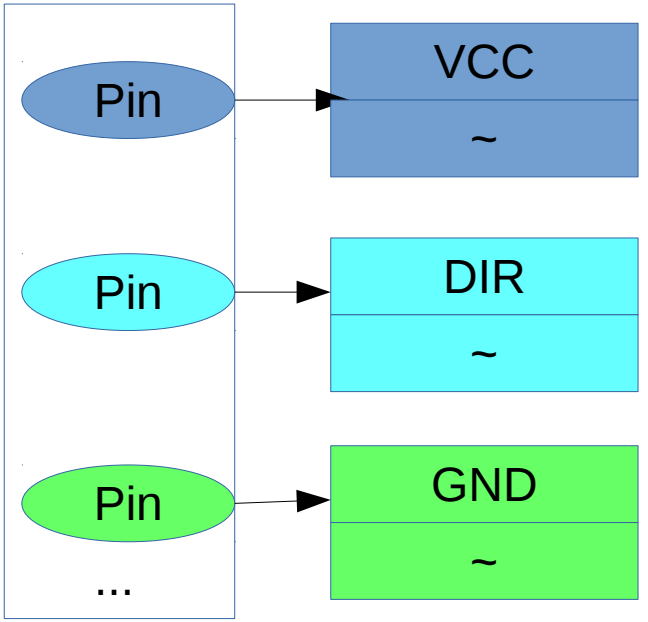
(Note: these hypothetical symbols are **not to be found** in available libraries;
even as the beautiful graphic symbols shown above are very much from KiCad libraries!)

What use, Unmapped Symbols?

By **separating pin-mapping from other aspects of a symbol :**
#2. We can increase **symbol re-use** (from libraries).

Currently, due to lack of such re-usability, there are at least 8 NMOS symbols in the “Device” library (but none are suitable for FDG311N).

Also there are over 20 NMOS symbols in the Digikey Library (again, none suitable for FDG311N).



Other similar instances are documented in the KiCad User Forum (see References).

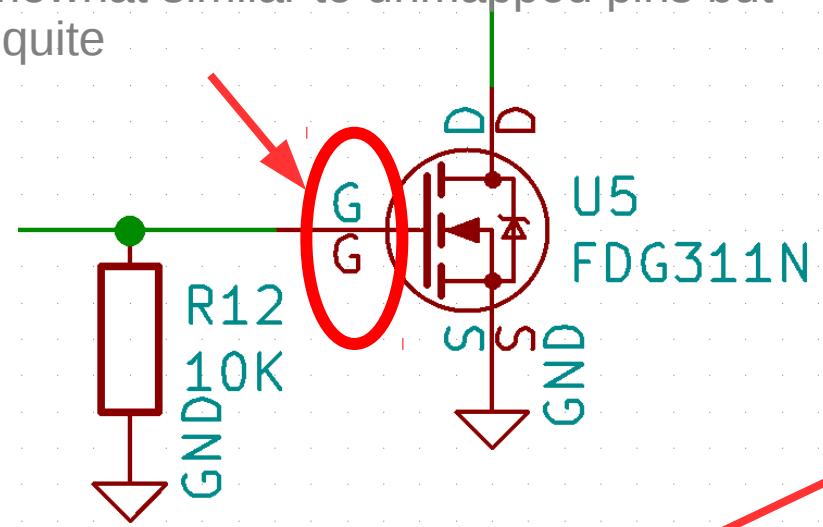
Root cause of the issue is – tight coupling of the implicit pin-mapping which is baked into the symbol.

Conversely, implicit pin-mapping reduces the suitability of those library symbols for re-use. They are not really as generic/re-usable as they would otherwise be.

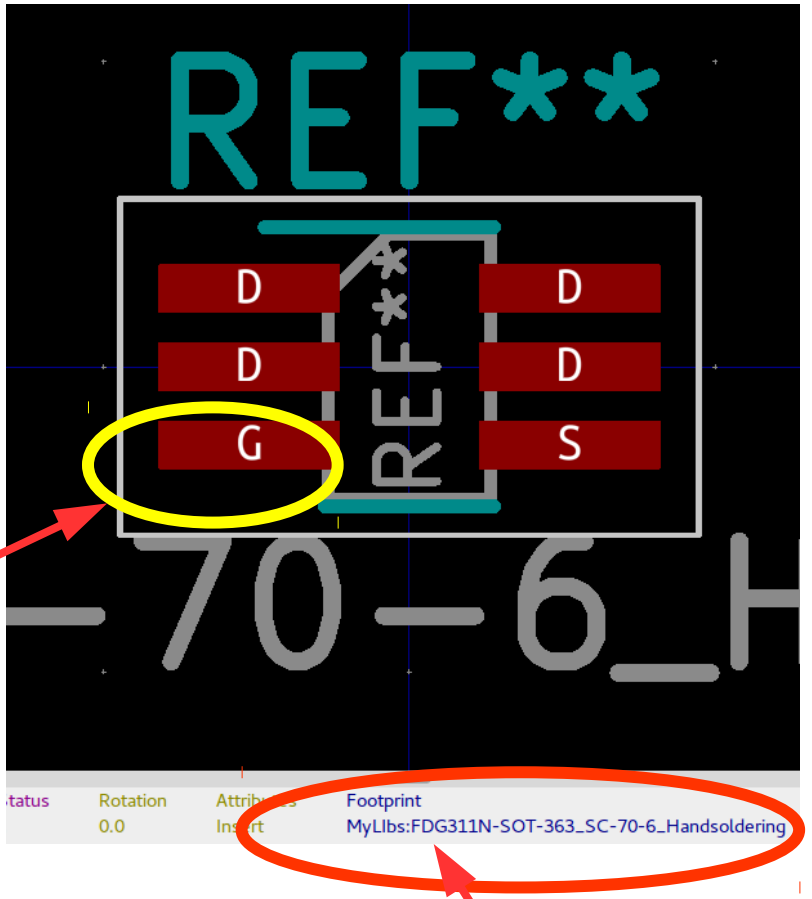
It is also highly desirable to have fewer schematic symbols in the library.
That would keep libraries small and symbols highly utilized, making library effort well spent.

What if… we store the mapping in the footprint?

Say, in the symbol, we **map pin name 'G'** to pin **number 'G'**
(KiCad accepts non-numeric pin numbers)
Somewhat similar to unmapped pins but not quite



In footprint, use a non-standard pin numbering



Oh No! It works, but is a poor solution.

Symbol and footprint were easily customized by copying from library and changing field values.

But non-standard pinning is harmful, will confuse assembly and troubleshooting etc.

Custom Footprint (pin numbering is customized)

With v6 Symbol File Syntax

A logical pin, mapped to a single physical pin (identified by "NUMBER")

```
(pin ...
   (at ...)
   (length  ...)
   (name "NAME" ...)
   (number "NUMBER" ... )
  )
```

A logical pin, mapped to a list-of-pin-numbers

```
(pin ...
   (at ...)
   (length  ...)
   (name "NAME" ...)
   ((number "NUMBER1" ... )
   (number "NUMBER2" ... )
   ...)
  )
```

Proposed

An unmapped logical pin

```
(pin ...
   (at ...)
   (length  ...)
   (name "NAME" ...)
   ()
  )
```

Conclusion: v6 Symbol syntax is already equipped to handle these cases:

- *a list of pin numbers,*
- *a single pin number,*
- *an unmapped pin.*

Or a more regular syntax: (pin ... (pads (number…) (number…))) could be used. Symbol format for v6 handles lists naturally thanks to S-expr's roots in LISP.

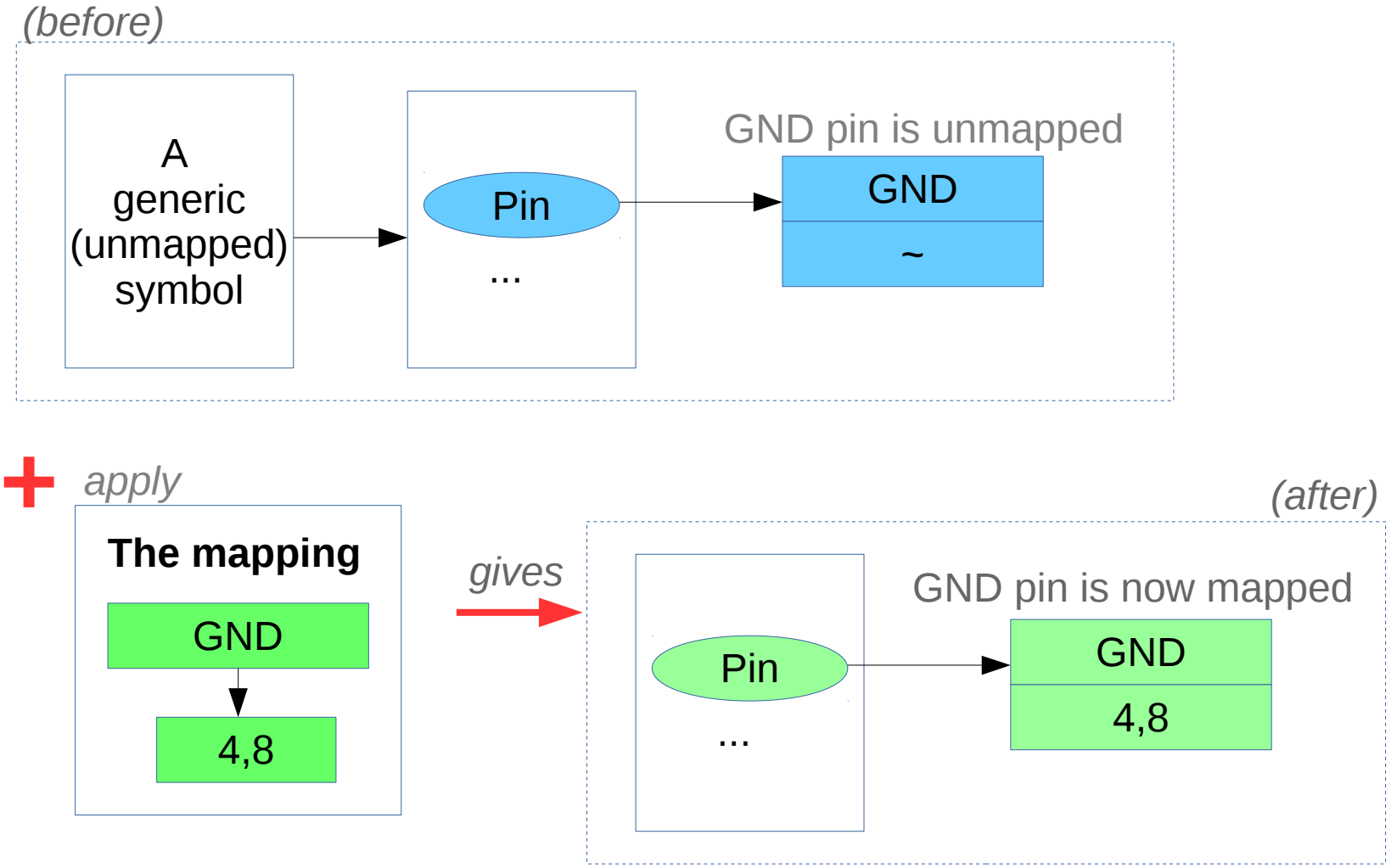
Interim conclusions

- Pin mapping, from pin name, into a *list of physical pins* is good
- It is expressible in v6 symbol file format
- Unmapped generic symbols are useful; they fill a gap
- Generic (standard) footprints are good
- Having pin mapping as not necessarily implicit in the symbol, will be good
- Having pin mapping implicit in footprints is harmful, not recommended
  - Some KiCad users are tempted to take this path; KiCad can help by providing the alternative.
- It's better to use library symbols and library footprints unmodified (if that is possible); especially the graphics, footprint geometry and the pinning.
- Atomic parts are good but scarce
  - N pin-committed symbols with M possible pin mappings makes a large space of size M x N which is sparsely populated or conversely, is pointless to populate densely
- The wait for atomic parts can be a long wait… (Economics)

… moving on to suggested mechanisms →

Applying a mapping to a pin

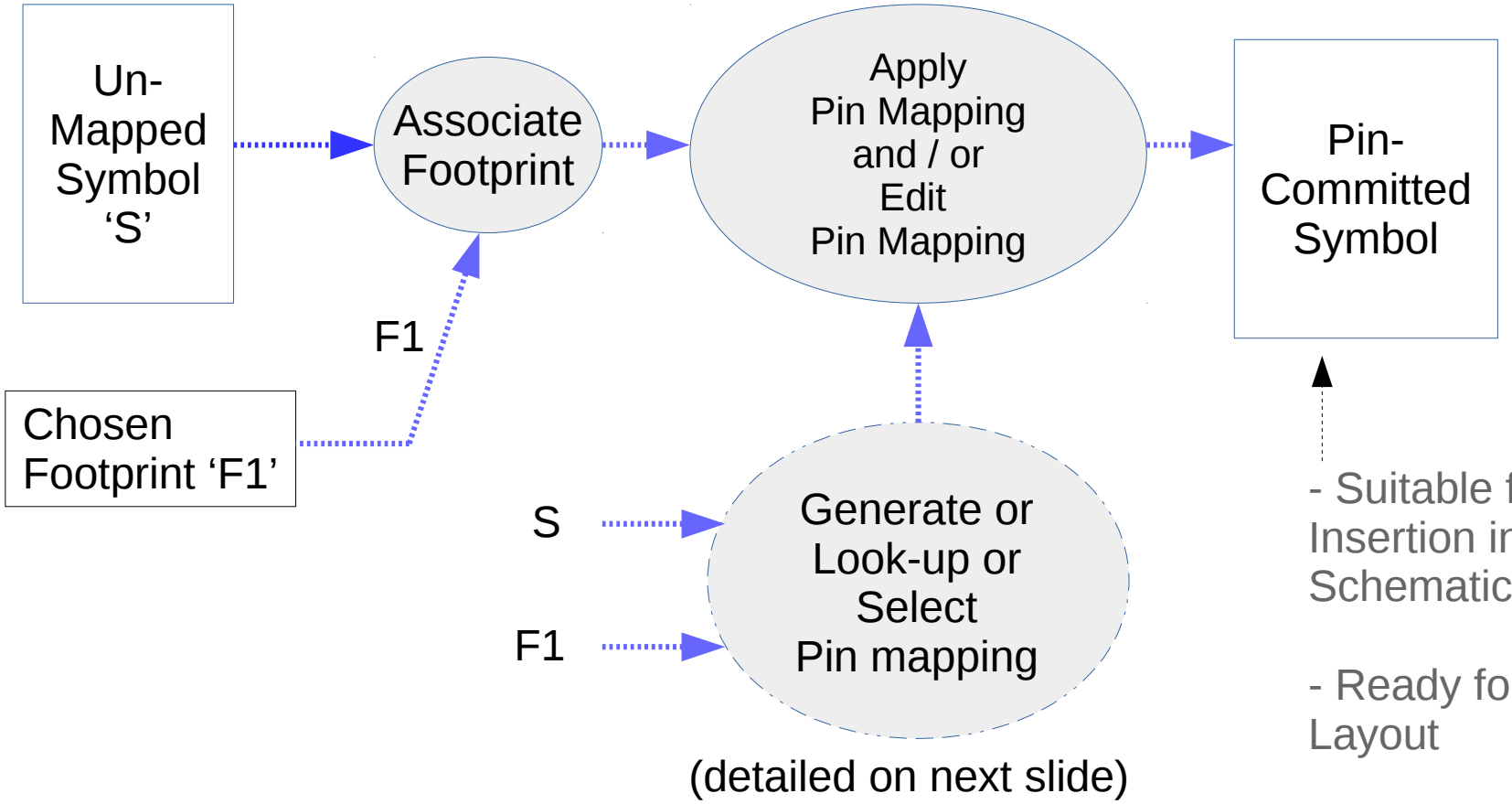
(taking the GND pin for example)



Unmapped generic symbol + Pin Mappings → Pin-committed symbol

Note: Symbol Graphic can be re-used as-is. Also logical pin's electrical class, etc

Applying pin-mappings to a symbol



- Suitable for Insertion into Schematics

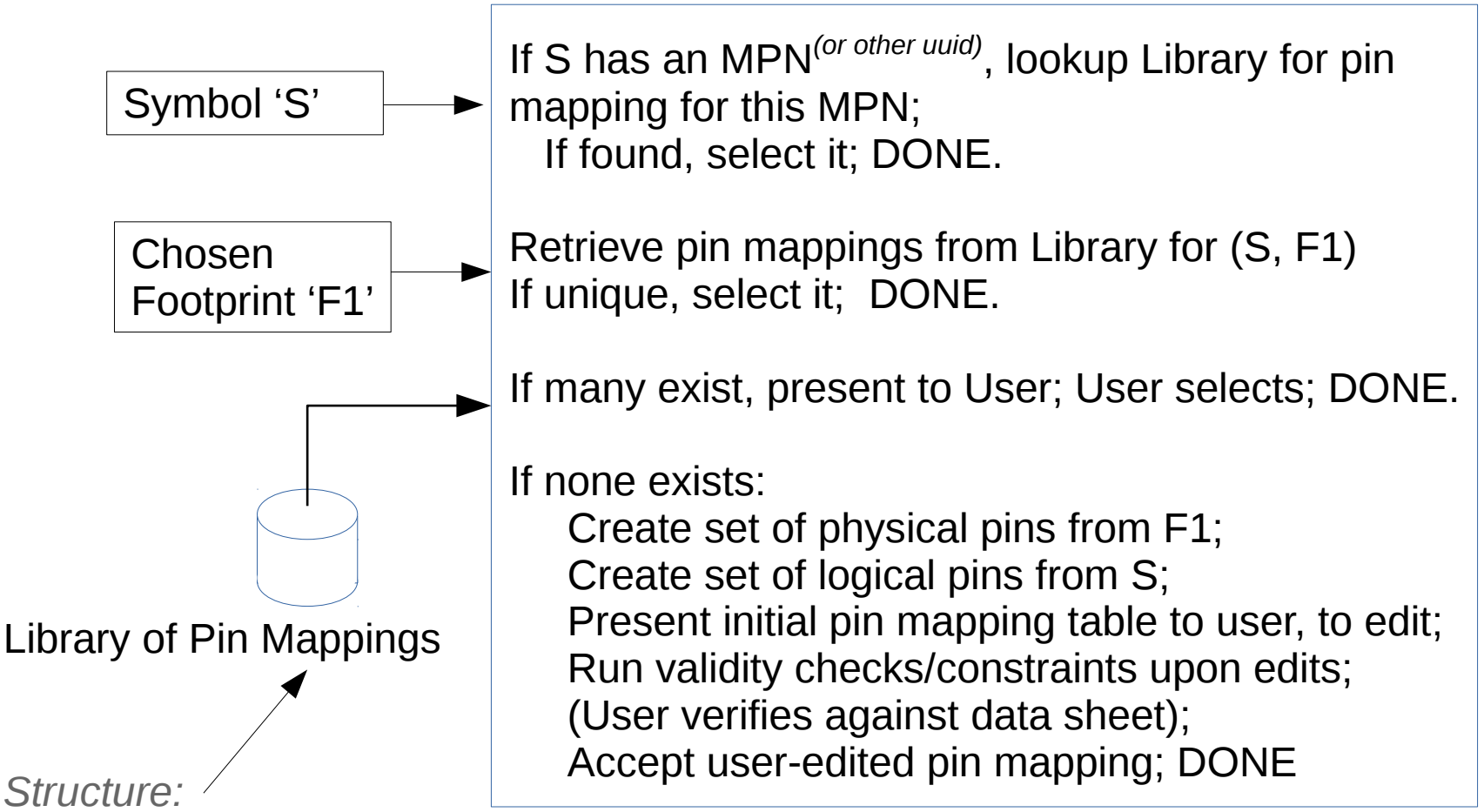
- Ready for Layout

Could be saved as a library asset or even as an Atomic Part

Legend: ⇢ *Data flow*

Note: Pin mappings are, by definition, footprint dependent.

Generating / Selecting a Pin-Mapping



Structure:
Non-uniquely indexed by (S,F1) and uniquely indexed by MPN (optional) or uuid

Note: Editing of the mapping in an already mapped symbol can be similarly supported.

Pin mapping as an asset

Creation from scratch: A default pin map can be generated as a template to seed the editing process; the flow could be as follows.

- GUI presents a mapping table, with pin numbers (generated from Footprint) on column #1 and logical pin name choices on column #2; one row per pin number. User selects the corresponding logical pin name on column #2 (e.g. aided by a list containing all candidate pin names) and associates the two.
- Save edited mapping to a library, with MPN (or a uuid) as unique key (for supporting Atomic Parts), and/or:
- Save edited mapping to a library, with (Symbol, Footprint) as non-unique key for later retrieval.

Applying a pin mapping is extending a “base” symbol

(Caution: based on my very limited understanding of the planned v6 Symbol format!)

- For a mapped (or unmapped) logical pin in any symbol S, applying a pin mapping amounts to the over-riding (or first-time setting) of a property of that pin within S.
- Such property change in S amounts to “extending” S to become S1 where S1 is the “extended symbol” derived from S.
- It then seems logical that an unmapped symbol is a candidate “base symbol” in being “purer” or “more abstract” which can be “extended” to produce concrete symbols which are instantiable in design.
- Certainly, there are other forms of extending a base symbol to include BoM information etc, but applying/modifying pin mappings is already part of the possibilities envisioned for v6.
- It is possible to view Atomic Parts as fully extended symbols which are flagged as final, that is, are not permitted to be any further extended.

Supporting Atomic Parts

For Atomic Part use-case, the expectation is that the libraries used are ***trusted*** to a high level. Verification by user could then be skipped without adverse impact. This can be achieved if these symbols (and properties) are immutable, once created.

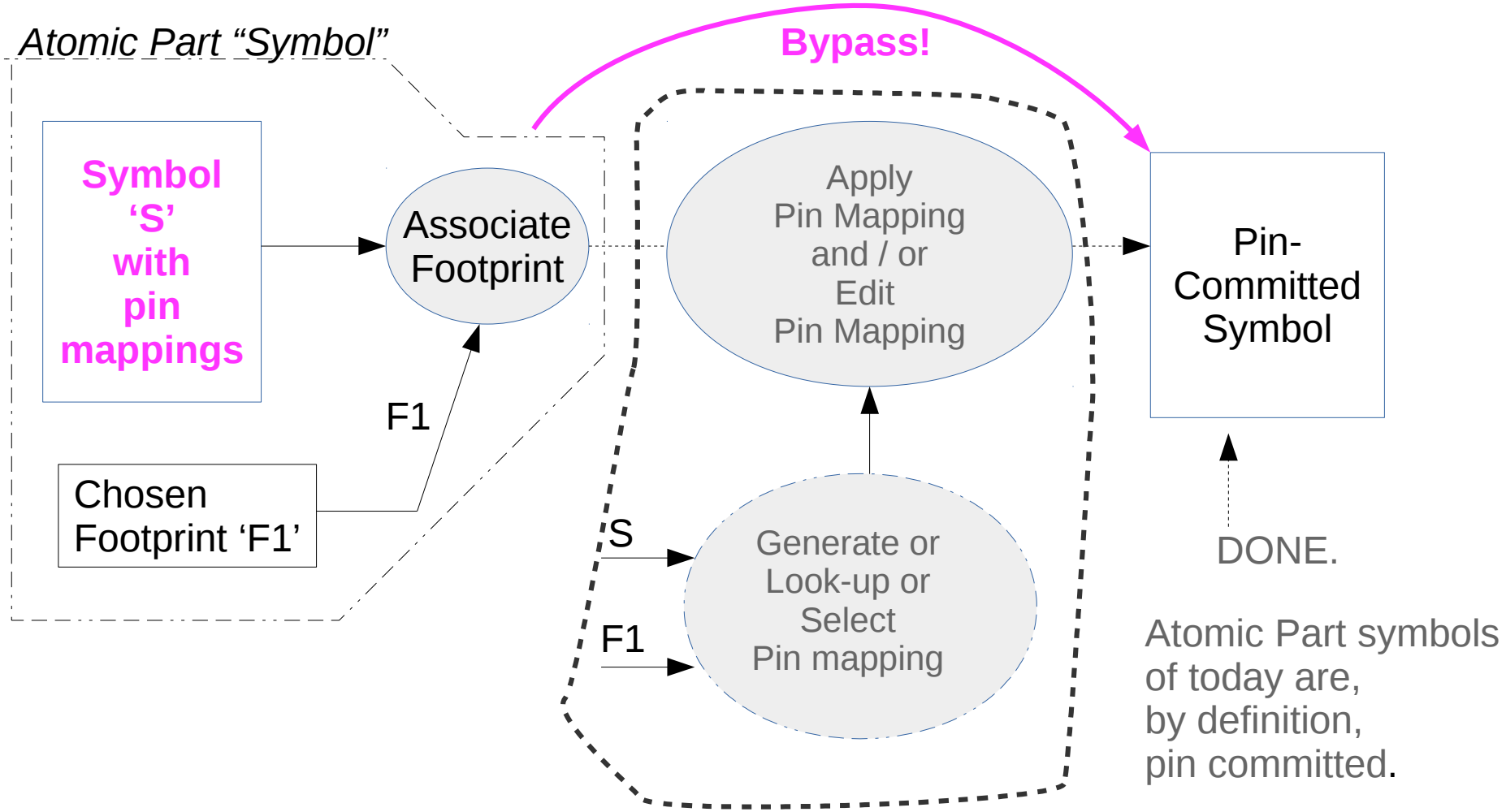
Atomic Part Symbols can be treated as Extended Symbols which are closed for further extension. However, supporting minor variants of footprints such as a hand-soldering variant, is acceptable and desirable. The primary attraction of Atomic Parts is that it locks in the type of footprint, but not the exact variant. On closer examination, it is really the pin-mapping (which is indeed footprint dependent) that makes Atomic Parts attractive. (BoM attributes comes second).

Since an MPN will be available when Atomic Parts are used, Atomic Parts could be instantiated through a process of “extension” (i.e., applying property changes), in terms of applying a pre-defined and immutable pin mapping.

If that route is taken, then Atomic Parts can be handled well and in a unified manner by maintaining its pin mappings uniquely indexed by MPN (or other uuid).

Backward compatibility

for Atomic Parts of today



(This shows "bypass path" modifying the data flow shown in page 16)

Problems addressed & some References

- Handling of multiple physical pins per logical pin:
  - Solves the ERC issue in: https://bugs.launchpad.net/kicad/+bug/1469525
  - Addresses the “oddball problem” in:

    https://forum.kicad.info/t/erc-with-symbol-providing-multiple-ground-outputs/13268
- Issue of Symbol proliferation (One-symbol-variant-per-footprint), eg:

https://forum.kicad.info/t/relating-pins-and-pads-kicad-pitfalls-for-newcomers/6186

- Related references

https://forum.kicad.info/t/some-thoughts-on-the-underlying-data-model-symbols/18502/

https://forum.kicad.info/t/idea-for-pin-mapping-hard-wired-items-ie-mosfets/18586

(Recent threads on this topic)

https://forum.kicad.info/t/chips-with-distinct-pin-numbers-per-footprint-type/3584

(Pin mapping issues)

https://forum.kicad.info/t/notion-of-part-in-kicad/4057 (Makes the case for separate pin-mapping)

Thanks & Acknowledgements

- To KiCad Developers, for KiCad
- To KiCad Librarians & various asset creators
- To KiCad User Forum members, for discussions & sharing