CvPcb

June 13, 2018

Contents

参考手册

* 贡献者 *

Jean-Pierre Charras, Fabrizio Tappero, Wayne Stambaugh.

翻译

Liu HanCheng <buaa_cnlhc@buaa.edu.cn>, 2018. **反馈**

请将所有的 Bug, 建议以及新版本重定向于此:

- 关于 KiCad 文档: https://github.com/KiCad/kicad-doc/issues
- 关于 KiCad 软件: https://bugs.launchpad.net/kicad
- 关于 KiCad 软件国际化: https://github.com/KiCad/kicad-i18n/issues

发行日期及软件版本

发布于 2015-05-22.

CvPcb 导论

CvPcb 能够为原理图中的元器件与进行 PCB 布局时的封装分配关联。二者的关联关系将被添加入由原理图创建程序 Eeschema 创建的网络列表文件中。

仅当在元器件的封装字段初始化后, 由 Eeschema 生成的网络列表文件才会包含元器件 PCB 封装与原理图端口的关联关系。

这种情况下，封装和原理图之间的关联是用户在编辑原理图时，通过设置元件的封装字段创建的。此外封装也可能被预定义于原理图符号库中, 在用户从库中加载这类元器件时, 其封装会被自动设置。

CvPcb 提供了在创建原理图的过程中为元器件分配 PCB 封装的简便方法。它拥有封装列表过滤, 封装预览以及 3D 模型预览功能. 这些功能旨在提高分配封装时的准确率。

用户可以手动为元器件分配对应的封装。通过创建.equ 文件, 也可以实现封装的自动分配。.equ 文件包含了元器件和其对应封装的相关信息

我们认为使用这种交互式的封装分配方法, 比起直接在绘制原理图的时候进行封装分配, 更加简单, 并且拥有更高的正确率。

使用 CvPcb, 你可以看到所有可能可用的封装列表。此外, 你还能在窗口中看见不同封装的真实几何外形, 这可以帮助你为原理图中的元器件选择正确的封装。

CvPcb **只能通过 Eeschema 启动**, 其入口位于 Eeschema 的顶部工具栏处。无论 Eeschema 是通过 Kicad 的项目管理器启动, 还是作为独立组件单独启动, 都可以通过其顶部工具栏按钮访问 CvPcb。

从通过 Kicad 项目管理器启动的 Eeschema 访问 CvPcb 通常来说是更好的选择, 这是因为:

- CvPcb 需要读取项目配置文件, 以确定哪些封装库需要被加载
- 当项目文件和打开的原理图文件处于同一目录时, Cvpcb 可以初始化元器件的封装设置字段。

从通过 Kicad 项目管理器启动的 Eeschema 访问 CvPcb, 可以确保上述要求自动得到满足。

Warning

尽管用户确实 能够从单独启动的 Eeschema 中访问 Cvpcb, 但是请注意, 单独打开的原理图文件由于可能缺失相关的项目文件, 进而导致缺失相关的库文件, 这会使 Cvpcb 的工作出现问题。如果在单独打开的原理图文件所在的目录中, 不包括其他 fp-lib-table 文件, 那么工程封装库列表也是不可用的

CvPcb 特性

手动或自动关联

Cvpcb 同时支持交互式的手动封装分配和通过.equ 文件进行的自动封装分配。

启动 Cvpcb

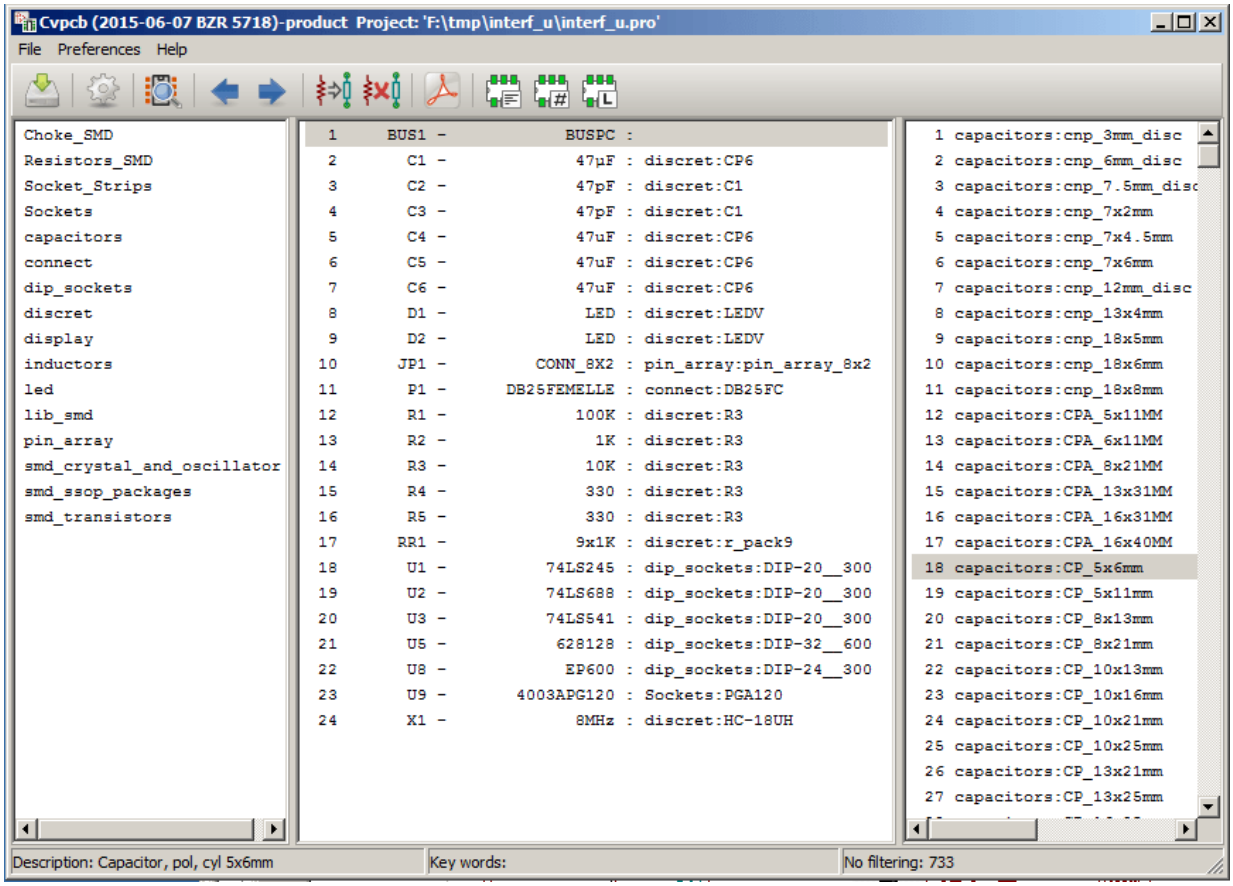
Cvpcb 仅能从原理图绘制程序 Eeschema 中启动

Eeschmea 会自动的将一些信息 (例如当前原理图中元器件列表和可用的封装), 传递给 CvPcb。在调用 CvPcb 之前, 用户唯一需要做的工作就是为原理图中的各个元件编号。

CvPcb 命令

主界面

下面的图片展示了 CvPcb 的主界面



在主界面中，左窗格包括了所有与项目关联的可用的封装库文件名列表; 中间窗格包括了从网络文件中读入的所有元器件列表; 右窗格包括了所有从与项目关联的封装库中加载的可用的封装列表。如果没有加载网络文件, 那么元器件列表将是空的; 类似的, 如果没有找到可用的封装库, 那么位于右侧的封装列表也将是空的。

主界面工具栏

顶部的工具栏提供了下列命令的快速访问接口：

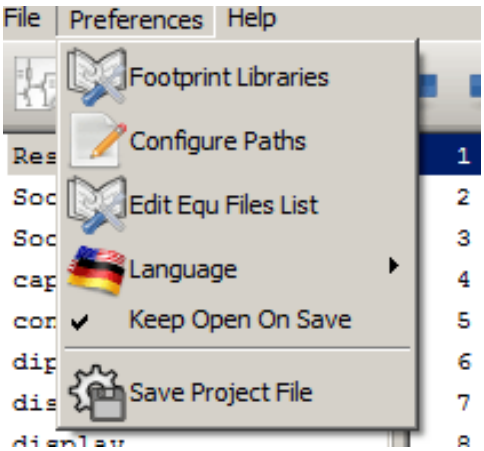
	将当前的封装关联转移到 Eeschema 中。
	启动 CvPcb 配置菜单。
	查看选中封装的几何外形。
	自动选中列表中上一个未分配封装的元器件。
	自动选中列表中下一个未分配封装的元器件。
	使用.equ 文件自动为元器件分配封装。
	删除所有的封装关联。
	使用默认的 pdf 阅读器打开选中封装的 pdf 文档。
	改变元器件过滤器的状态。
	改变引脚数目过滤器的状态。
	改变封装库过滤器的状态。

主界面键盘快捷方式

下面的表格列出了 CvPcb 主窗口中的键盘快捷方式。

右箭头 / Tab	使当前焦点右侧窗体部分获得焦点。在当前焦点为最右窗体时，则最左窗体部分获得焦点。.
左箭头	使当前焦点左侧窗体部分获得焦点在当前焦点为最左窗体时，则最右窗体部分获得焦点。
上箭头	选择当前选中列表中的上一项
下箭头	选择当前选中列表中的下一项
Page Up	选择当前选中列表中一整页之前的项。
Page Down	选择当前选中列表中一整页之后的项。
Home	选择当前选中列表中的第一项。
End	选择当前选中列表中的最后一项

CvPcb 配置



CvPcb 可以在保存封装关联关系后自动关闭或手动关闭。

点击菜单中的“设置” — “封装库”将会打开封装库配置对话框。

根据 CvPcb 版本得不同，存在两种封装库管理方式:

- 传统方式是使用.mod 文件进行管理, 用户可以看到封装库文件列表。
- 新的管理方式使用“Pretty”格式。这种管理方式将会使用一个文件夹列表, 每个文件夹 (文件夹名为 *.pretty) 就是一个库。新的管理方式允许用户使用来自 gEDA/gPCB 中的库以及 Eagle 软件中 xml 格式的库。

封装库管理

重要提示:

本节内容只与 *2013* 年 *12* 月之后发行的 *KiCad* 相关

封装库表

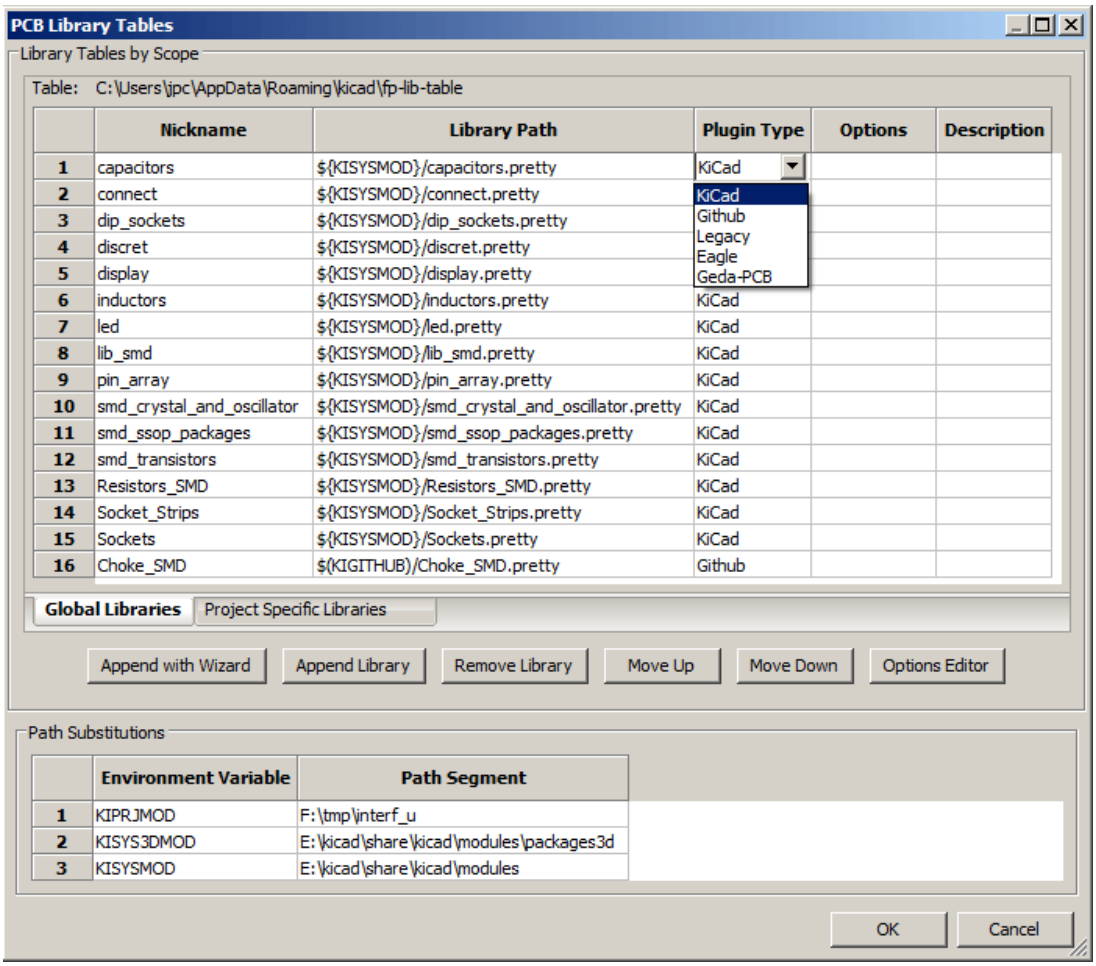
从 2013 年 12 约以后，Pcbnew 和 CvPcb 使用了新的基于 **封装库列表**的库管理工具。新的管理工具允许用户通过以下方式 **直接使用封装库**:

- KiCad 传统封装库文件 (.mod 文件)
- KiCad 新式 *.pretty* 封装库 (在用户的本地磁盘中，拥有.pretty 扩展名并包含了.kicad_mod 文件的文件夹)
- KiCad’s 新式 *.pretty* 封装库 (托管在官方或第三方维护的 Github 仓库中)
- GEDA 封装库 (包含了.fp 文件的文件夹)
- Eagle 封装库

Note

- **用户仅能改写位于本地磁盘上的** Kicad *.pretty* **封装库 (以及这些文件夹中包括的**.kicad_mod **文件)**,
- **其余的格式都是只读的**。

下面的图片展示了封装库列表编辑对话框, 该对话框可以通过菜单栏中的“设置”- “封装库”打开。



封装库列表的作用是为 Kicad 支持的封装库分配一个别名。* 可利用该别名, 而不是之前基于封装库路径查找顺序的方法, 进行封装的查找.

这项功能能够使 Cvpcb 控制不同封装库的加载，从而使得访问那些位于不同封装库但是拥有相同名称的封装可以被正确访问。此外，这项功能还能使 Kicad 访问来自其他 PCB 编辑器，例如 Eagle 和 GEDA，的封装库

全局封装库列表

全局封装库列表包括了那些在任何项目中都可访问的封装库。该表格的配置存储在用户主目录下的 fp-lib-table 文件中。用户主目录的具体位置由用户使用的操作系统决定。

项目封装库列表

项目封装库列表包括了仅在当前打开项目内能访问的封装库。项目封装库列表仅能在项目的网络列表文件被加载时才能编辑。如果当前没有打开任何项目，或是在打开的项目目录中没有封装库列表文件，那么系统将创建一个新的可供编辑的表格文件。

初始设置

首次运行 Pcbnew 或 Cvpcb 时，如果在用户主目录下无法找到全局封装库列表文件 * fp-lib-table*，那么 Pcbnew 或 CvPcb 将尝试将存储在 Kicad 模板文件夹中的默认封装库列表文件复制到用户主目录中。

如果默认的 fp-lib-table 文件无法被找到，那么将会在用户主目录下创建一个新的封装库列表文件。这种情况下，用户可以从别处复制 fp-lib-table 文件，或是手动进行封装库配置。

默认的封装库列表将作为 kicad 的一部分而被安装，其中包括了许多标准的封装.

显然，用户 **首先**需要根据实际需求，修改该列表 (添加/移除项目)。

(加载过多的封装库会耗费许多时间)

添加列表项目

如果要使用一个封装库，它必须先被添加到全局封装库列表或工程封装库列表中。仅当用户当前拥有一个网络列表文件时，工程封装库列表才是可用的。

封装库列表中的项目别名不可重复

“别名”字段可以由用户自行决定，不必和封装库的路径/封装库文件名等相关。别名中不可以出现冒号: 。列表中的每一项需要有一个可用的路径。根据封装库类型的不同，路径的具体表现形式可能不同。“路径”字段中的内容可以是绝对路径，相对路径，或者是环境变量 (下文中会进一步讨论)

为了正确读取封装库，列表中每一项的“插件类型”字段必须被正确选择。目前 KiCad 支持的类型包括 KiCad legacy, KiCad Pretty, Eagle, 和 GEDA 封装库。

列表中的“描述”字段，用于为项添加额外的备注信息。列表中的“设置”字段在当前版本尚未使用，修改该字段没有任何效果

- 注意，用户无法在同一个封装列表中为两项分配相同的别名。但是可以在全局封装列表和工程封装列表中使用相同的别名。
- 如果重名发生，工程封装列表中的名称将优先被使用。定义在工程封装列表中的项目，将会被写入当前网络列表所在目录下的 fp-lib-table 文件中。

环境变量替换

环境变量替换是封装库列表的强大功能之一。它将允许用户使用环境变量定义自定义的封装路径。使用环境变量替换，需要在封装库列表的“路径”字段中，遵守以下语法: +${ENV_VAR_NAME}

运行时，KiCad 默认定义 **两个环境变量**:

- `KIPRJMOD` 环境变量。该变量指向当前项目的目录，不可被改变。
- `KISYSMOD` 环境变量。该变量指向默认随 KiCad 安装的默认封装库目录。

用户可以通过在菜单栏中的“设置”- “配置目录”中重载 `KISYSMOD` 的值，因此用户可以使用自定义的封装库替换 Kicad 的默认封装库。

如果当前项目的网络列表文件已被加载, CvPcb 会将 `KIPRJMOD` 的值设置为网络列表文件的目录 (即项目目录)。

在加载一个 board file 时，Pcbnew 也会设置该环境变量。

该环境变量允许用户在不知道项目绝对目录的情况下，将封装库存储于项目目录下。

使用 Github 插件

Github 插件提供了只读访问那些包含 Kicad pretty 封装库文件的 Github 仓库的接口。该插件也提供了 COW(“Copy On Write”) 功能。该功能是可选项，它将允许用户编辑从 Github 仓库读取的封装库，并且将它们保存在本地。因而， “Github” 插件用于 * 只读访问托管于 https://github.com/ 的远程 pretty 封装库 * 如果要向封装库列表中添加 Github 项，其 ``路径” 字段需要被设置为一个合法的 Github 地址。

例如:

https://github.com/liftoff-sr/pretty_footprints

或

https://github.com/KiCad

典型的 Github URL 为以下形式:

https://github.com/user_name/repo_name

“插件类型’字段必须被设置为‘ ``*Github*’。如果要使用 COW 功能, 必须向设置字段内添加 **allow_pretty_writing_to_this_dir** 的选项. 该选项的值用于设置对 Github 上封装库的修改副本的存储路径. 存储在该目录中的封装将和 Github 上的只读部分共同构成封装库. 如果没有声明该选项, 那么 Github 封装库就完全是只读的. 如果声明了该选项, 那么对该 ``混合’ ’ 封装库的修改, 将存储到本地的.pretty 文件夹中. 注意该 ``混合” 封装库中位于 Github 中的那一部分始终是只读的, 这意味着你无法直接删除或修改特定 Github 仓库中的内容. 进一步的讨论认为该混合仓库仍然属于 “Github” 类型, 只是它包括了本地的读/写部分及远程的只读部分.

下面的表格展示了没有 **allow_pretty_writing_to_this_dir** 设置的封装库列表项:

别名	库路径	插件类型	设置	描述
github	https://github.com/liftoff-sr/-pretty_footprints	Github		Liftoff’ s GitHub Foot-prints

下面的表格展示了开启 COW 功能的封装库列表项。注意环境变量 ${HOME} 仅做举例用。github.pretty 目录应该位于 ${HOME}/pretty/ 。如果用户使用了 **allow_pretty_writing_to_this_dir** 选项，则应该提前手动创建上述目录并以.pretty 结尾。

别名	库路径	插件类型	设置	描述
github	https://github.com/liftoff-sr/-pretty_footprints	Github	allow_pretty_writing_to_this_dir= ${HOME}/pretty/github.pretty	Liftoff’ s GitHub Foot-prints

对于设置了 **allow_pretty_writing_to_this_dir** 的列表项，将首先加载位于本地的封装库。一旦用户使用封装编辑器对封装进行修改并保存在了 COW 本地文件夹中，那么 Github 仓库中, 和用户修改并保存过的的封装同名的任何封装的更新将不会被看见。

始终应该对每一个 Github 封装库使用不同的本地.pretty 目录。请不要试图通过多次引用同一目录的方式来结合两个不同的 Github 封装库。

也不要在不同的封装库列表项中使用相同的 COW 目录。这将带来不少问题。

设置字段中，**allow_pretty_writing_to_this_dir** 的设置值将会通过使用 ${} 表示的环境变量来创建路径，就和在库路径字段中使用环境变量一样。

那 COW 的目的究竟是什么？其实它是为了更好的促进封装库的分享。

如果用户周期性的将通过 COW 修改的 pretty 元件库反馈到 Github 仓库的维护者处，那么用可以帮助更新 Github 中的封装库。用户可以仅将 COW 文件夹中的 *.kicad_mod 文件发送给仓库的维护者。当用户得知他们的修改被同步到了 Github 仓库时，他们就可以删除本地的 COW 文件，然后使用 Github 插件提供的只读访问功能。用户应尽量向位于https://github.com 的主仓库多多提交，让自己的 COW 文件尽可能的小

使用模式

封装库既可以在全局定义，也可以在当前工程的范围内定义。全局定义的封装库将被存储在用户主目录下的 fp-lib-table 文件中，它们将可以在所有的项目中使用。

全局封装库始终可以访问，即使当前没有加载项目。

工程封装库则仅能在当前打开的网络列表文件中使用。

工程封装库列表存储在项目文件夹内的 fp-lib-table 文件中。你可以自由选择在哪个列表中定义封装库。

两种方法各有优劣。你可以在全局封装库列表中定义所有你可能会用到的封装，这样可以实现随用随取。这样做的缺点是你必须在非常多的封装中寻找你所需要的封装。你也可以根据项目的需求定义你的封装库，

这样做的好处在于你仅需要定义你在某个项目中需要用到的封装，这将大大减小搜索的复杂度。

这样做的缺点是，每新建一个项目，你都得重新手动定义每一个该项目中需要用到的封装。你也可以同时在全局范围和工程范围内定义你的封装库。

一种使用模式是，将所有常用的封装库定义在全局封装库列表中，将一些只在特定项目中使用的封装库定义在工程封装库列表中。总而言之，用户完全可以自主决定封装库的管理方式。

使用封装库列表添加向导

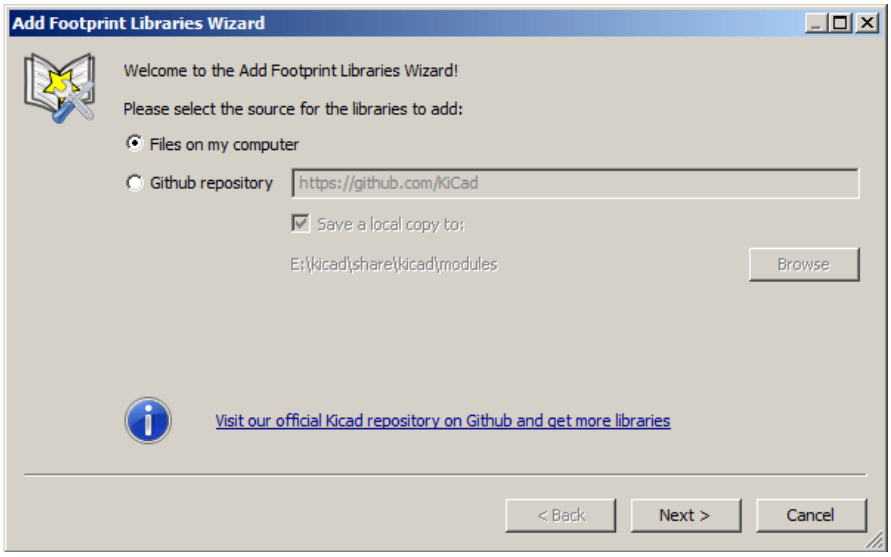
添加向导用于向封装库列表中添加封装库。可以在 *封装库列表编辑对话框*中打开添加向导。

待添加的封装库可以是任何 Kicad 支持的类型。

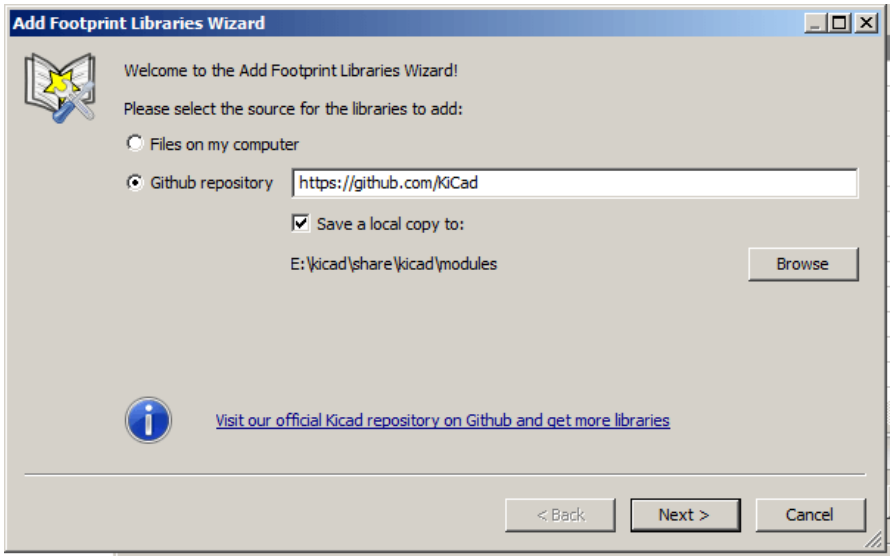
它可以是本地的封装库或是 Github 上的封装库。

当库位于 Github 仓库中时，它们可以被添加为远程仓库，也可以 **将它们下载到本地作为 本地仓库添加**。

此处本地封装库默认被选中。

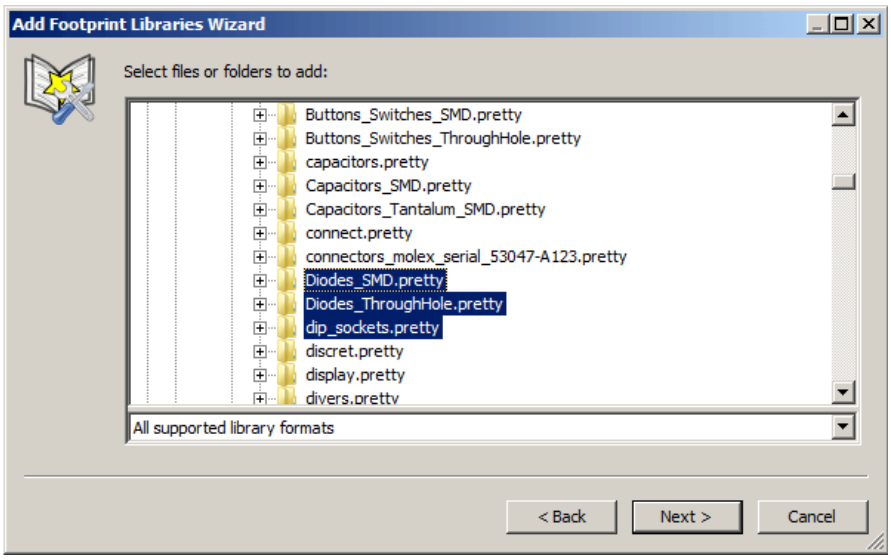


此处远程封装库选项被选中。

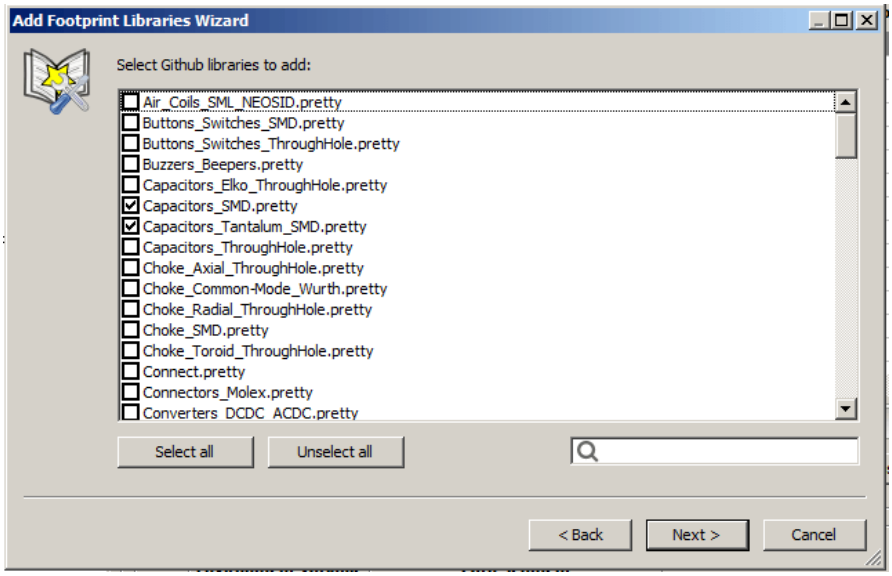


根据选择的不同，将会显示这些页面中的一个，用于让用户选择要添加的封装库列表:

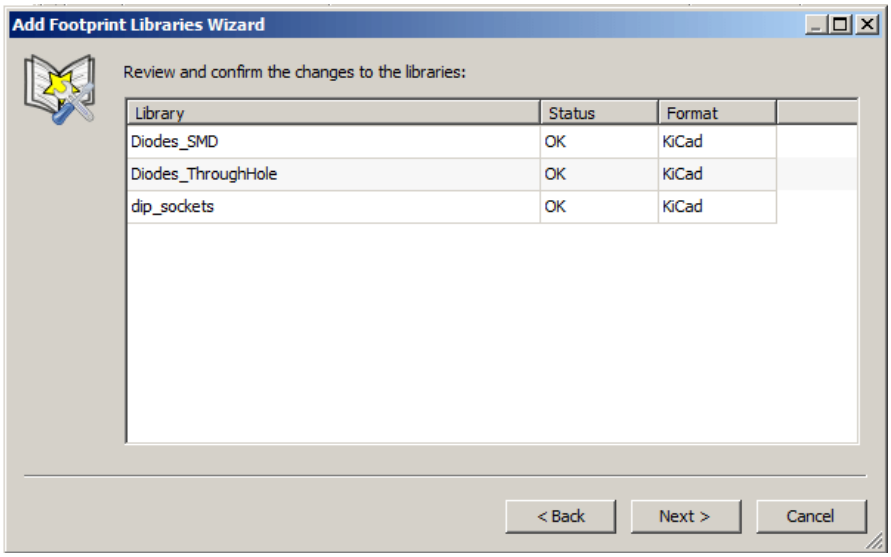
此处，本地选项被选中.



此处, 远程仓库选项被选中.



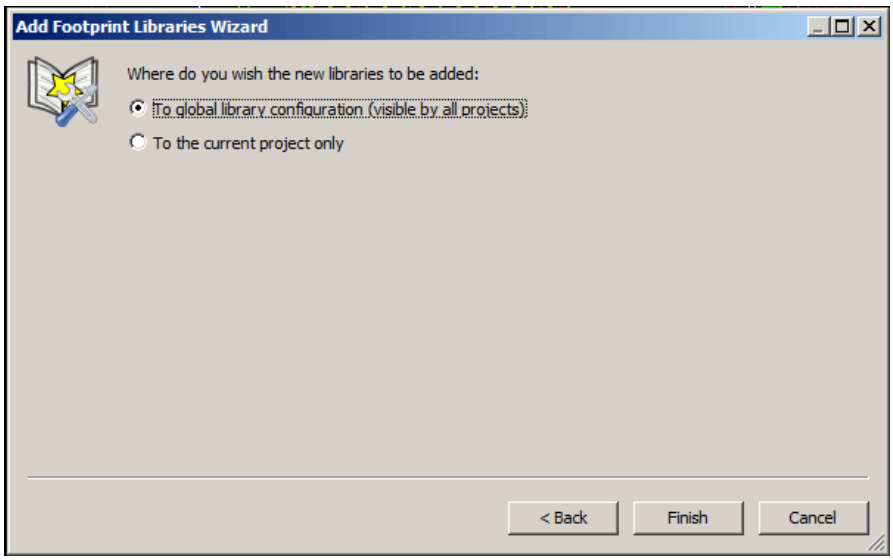
当一系列封装库被选中后, 下一个页面将验证选择:



如果一些选中的库不正确 (不支持, 不是封装库,…), 它们将被标记为“不可用”

最后一个页面是选择需要导出的封装库列表.

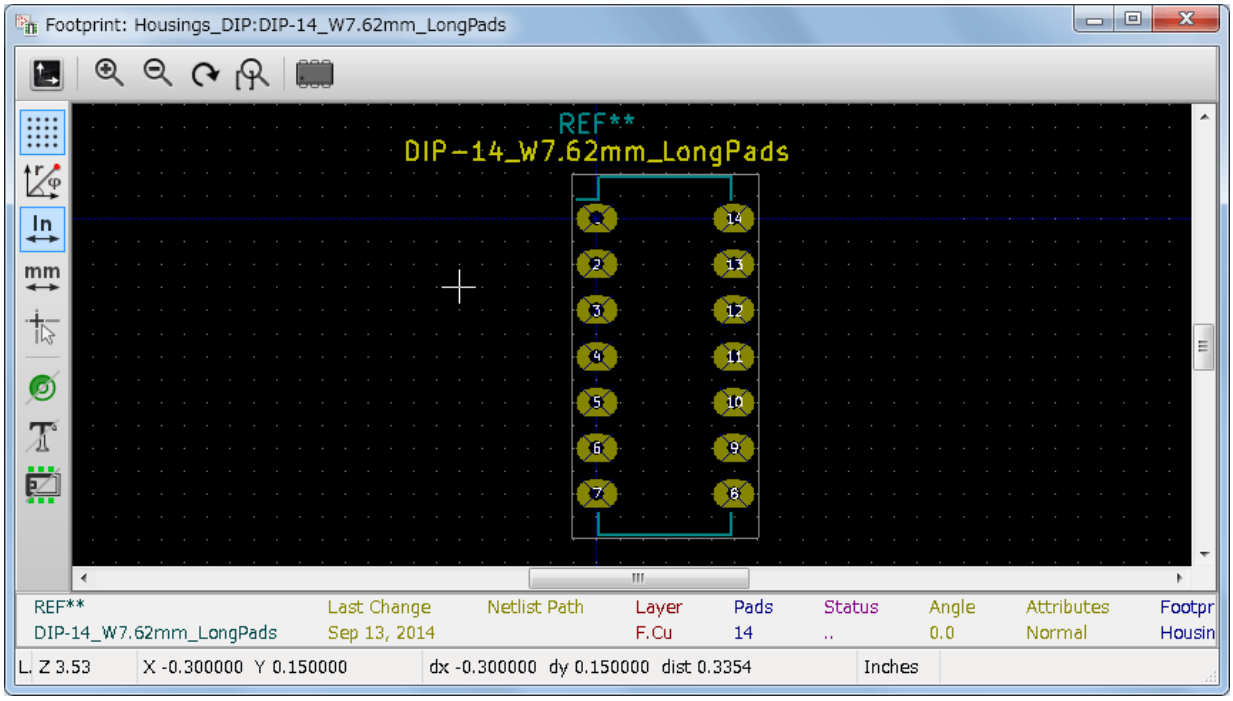
- 全局库配置
- 仅当前工程



查看当前封装

预览封装命令

预览封装命令会将当前选中的封装显示在 _ 封装 _ 窗口中. 如果封装已经被分配了一个 3D 模型, 那么三维模型也会在该窗口中显示。下面的图片展示了封装查看器窗口。



状态栏信息

状态栏位于 CvPcb 主窗口的底部，它为用户提供了许多有用的信息。下面的表格列出了状态栏中不同窗格的定义。

左	元器件统计: 总数量，未分配封装的数量
中	选中元件的过滤列表
右	过滤器状态以及可用封装的数量

键盘快捷键

F1	放大
F2	缩小
F3	刷新显示
F4	将光标移动到窗体中心
Home	是封装大小自适应于当前窗体
空格	设置相对于当前光标的相对坐标
右箭头	将光标向右移动一个网格单位
左箭头	将光标向左移动一个网格单位

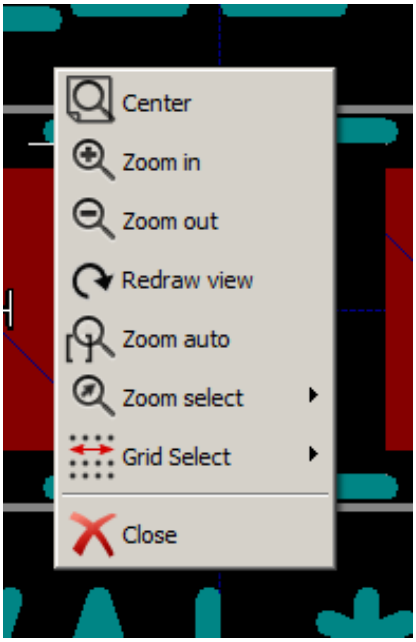
上箭头	将光标向上移动一个网格单位
下箭头	将光标向下移动一个网格单位

鼠标操作

滚轮	在当前游标位置放大或缩小
Ctrl + 滚轮	水平方向平移
Shift + 滚轮	垂直方向平移
鼠标右键	打开右键菜单

右键菜单

通过点击鼠标右键打开右键菜单



缩放选择	选择当前缩放的倍数
网格选择	选择网格大小

水平菜单

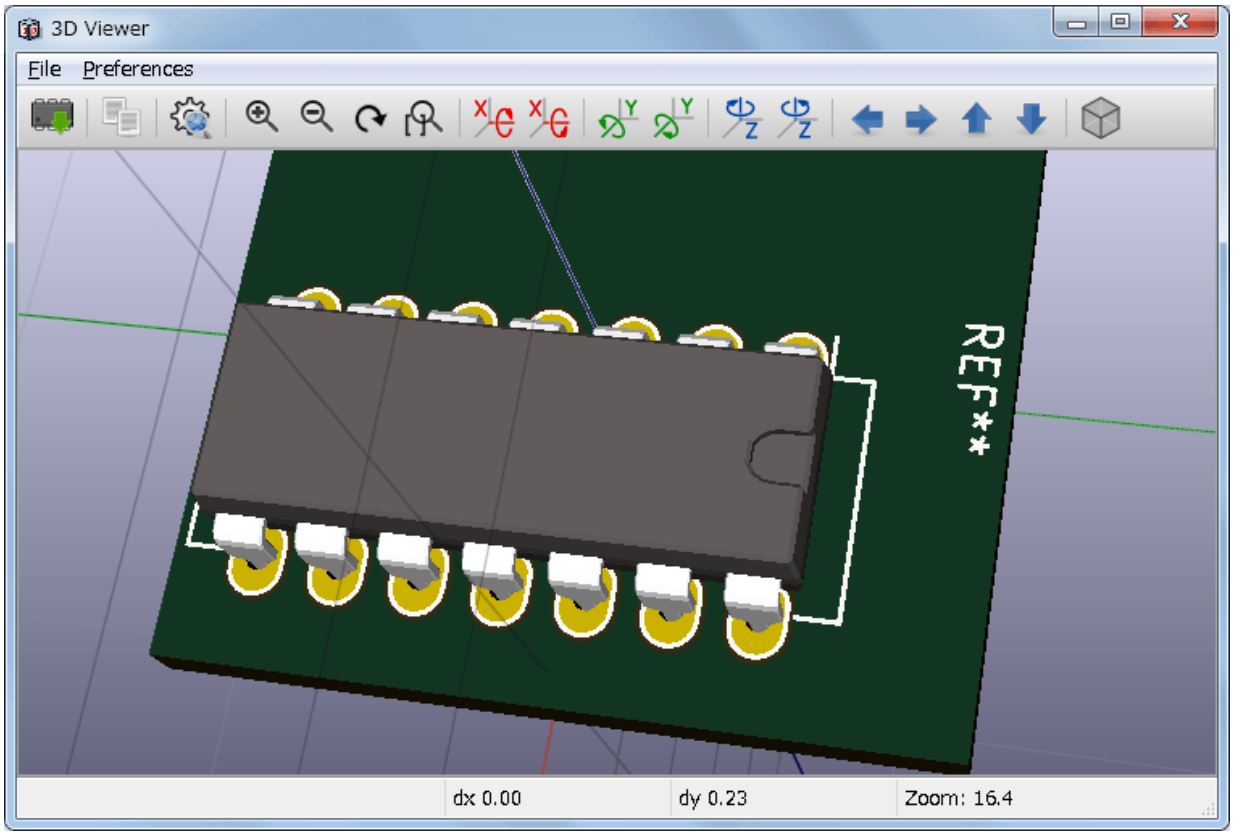
	打开显示选项对话框
	放大
	缩小
	重新绘制

	自适应显示
	打开 3D 模型浏览器

垂直工具栏

	显示或隐藏网格
	显示极坐标或矩形表示法的坐标
	以英寸为单位显示坐标
	以毫米为单位显示坐标
	切换指示器风格
	切换焊盘的显示方式。
	切换文本的显示方式
	切换边的显示方式

查看当前 3D 模型



鼠标操作

鼠标滚轮	在光标处放大或缩小
Ctrl+ 鼠标滚轮	水平方向平移
Shift + 鼠标滚轮	垂直方向平移

水平菜单

	重新加载 3D 模型
	将 3D 模型的图片复制到剪贴板
	3D 查看器选项
	放大
	缩小
	重新绘制
	自适应显示
	沿 X 轴反向旋转
	沿 X 轴正向旋转
	沿 Y 轴反向旋转
	沿 Y 轴正向旋转
	沿 Z 轴反向旋转
	沿 Z 轴正向旋转
	向左平移
	向右平移
	向上平移
	向下平移
	打开和关闭正交投影模式

使用 CvPcb 向元器件分配封装

手动分配封装

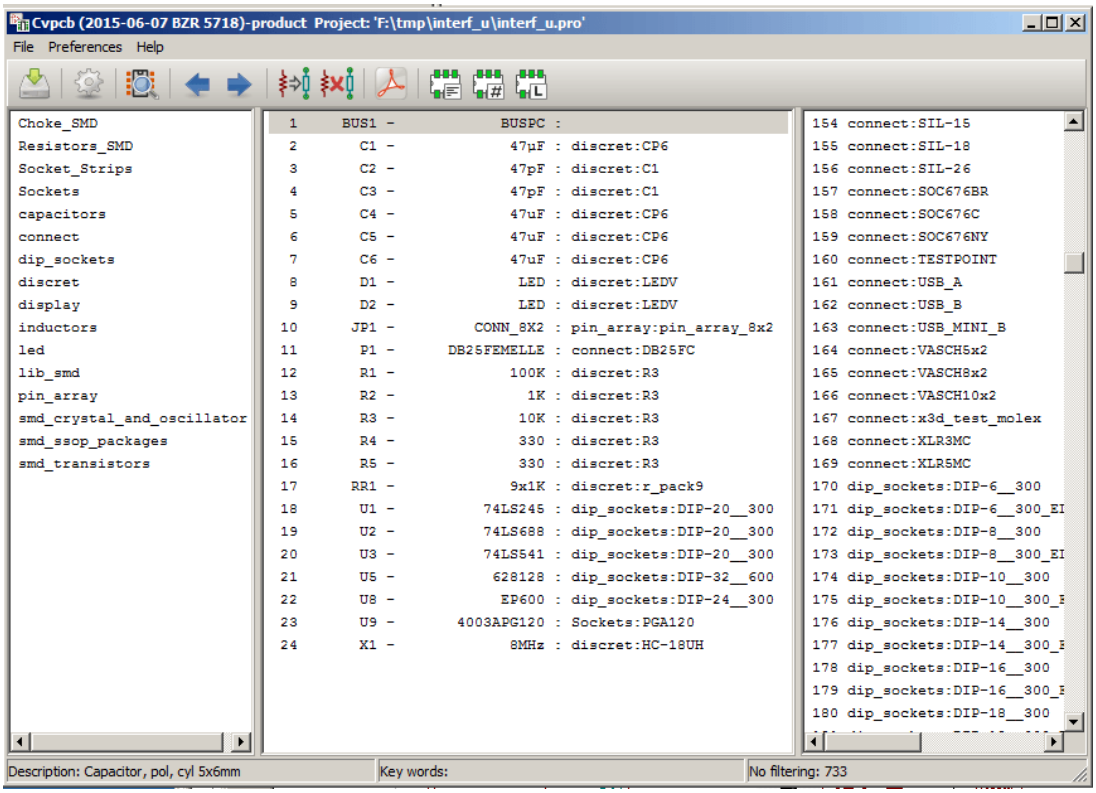
如果要手动分配封装, 首先在需要位于窗口中部的元器件窗格中选择一个元器件. 然后在右侧的封装窗格中, 鼠标左键双击想要分配的封装. 然后该封装就会被分配给选择的元器件. 此外, 分配完成后, 下一个未分配封装的元器件将会被自动选中. 改变元器件的封装时, 操作类似.

过滤封装列表

如果高亮选中某元器件/封装库时, 有一个或多个过滤选项已经打开, 那么, 右侧的封装窗格将自动显示过滤后的封装列表.

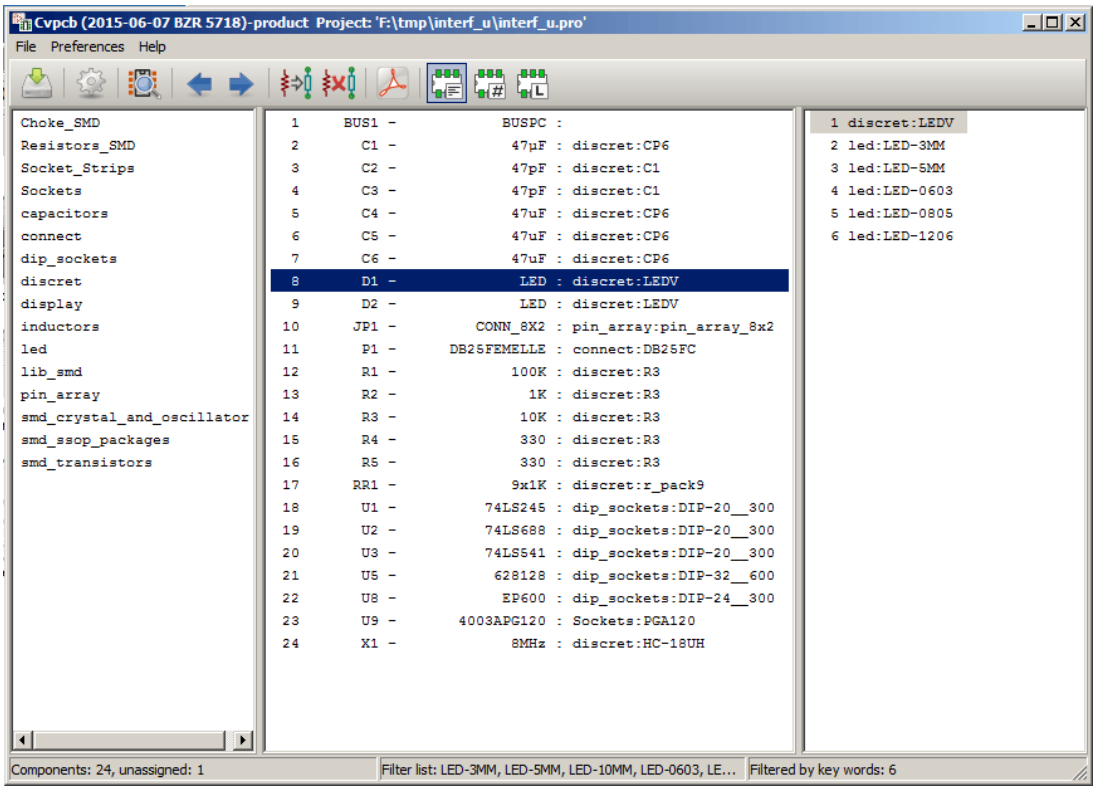

点击这些按钮将打开或关闭过滤器. 当所有的过滤器都处于关闭状态时, 将会在右侧显示所有可用的封装.

关闭过滤器:

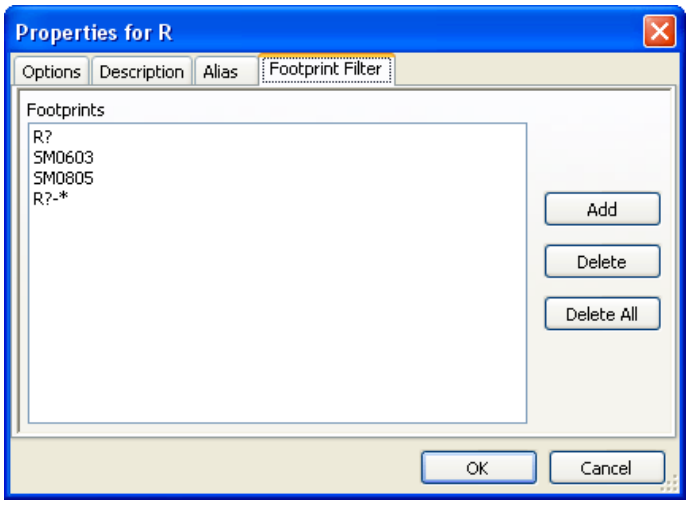


根据应用于选定元器件的过滤器对封装列表进行过滤. 选定元器件中启用的过滤器显示于主窗口底部的状态栏中部.

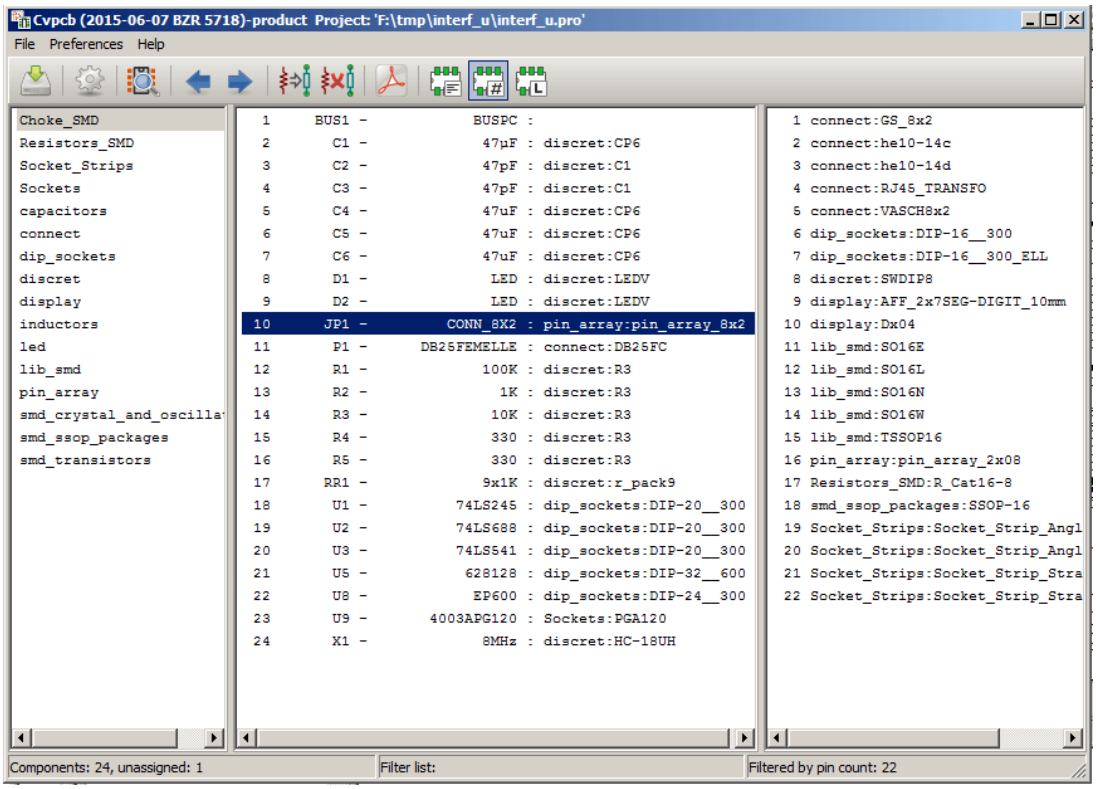
根据应用于选定元器件的过滤器对封装列表进行过滤



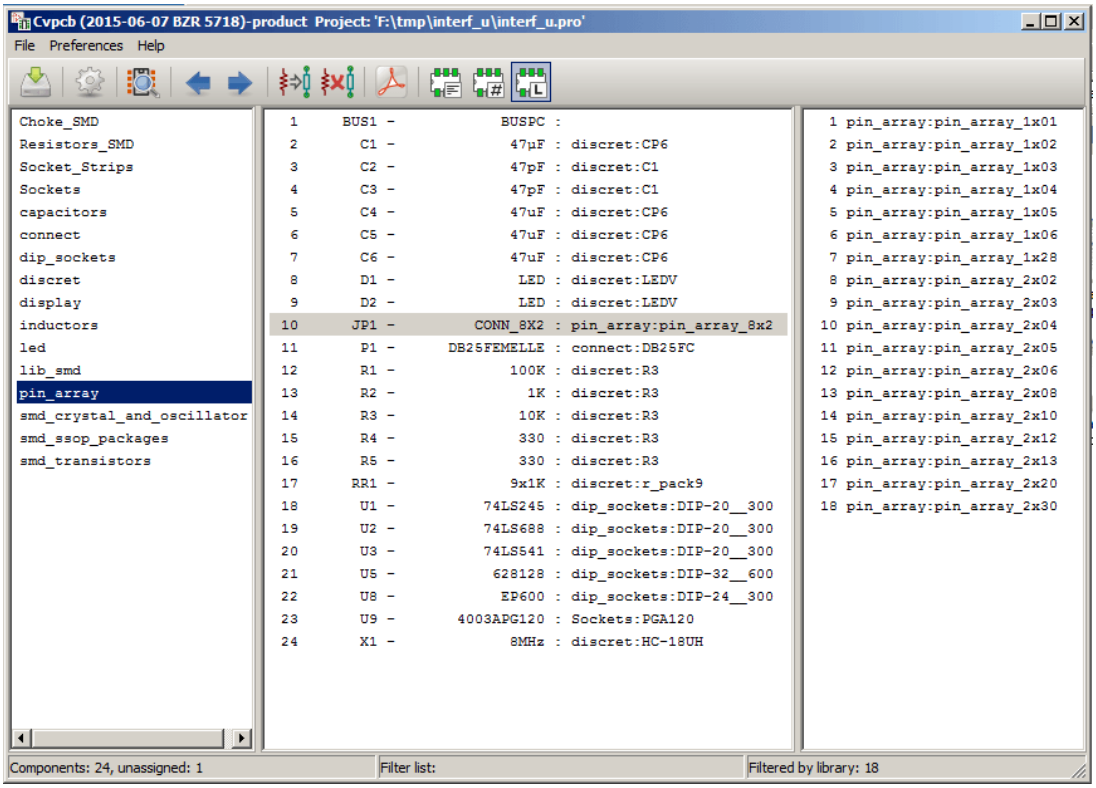
在 Eeschema 的元器件库编辑器中, 用户可以在元器件属性对话框的 ``封装'' 选项卡中设置元器件的可用封装列表. 元器件属性对话框如下图所示:



根据选中的元器件的引脚数目进行过滤:

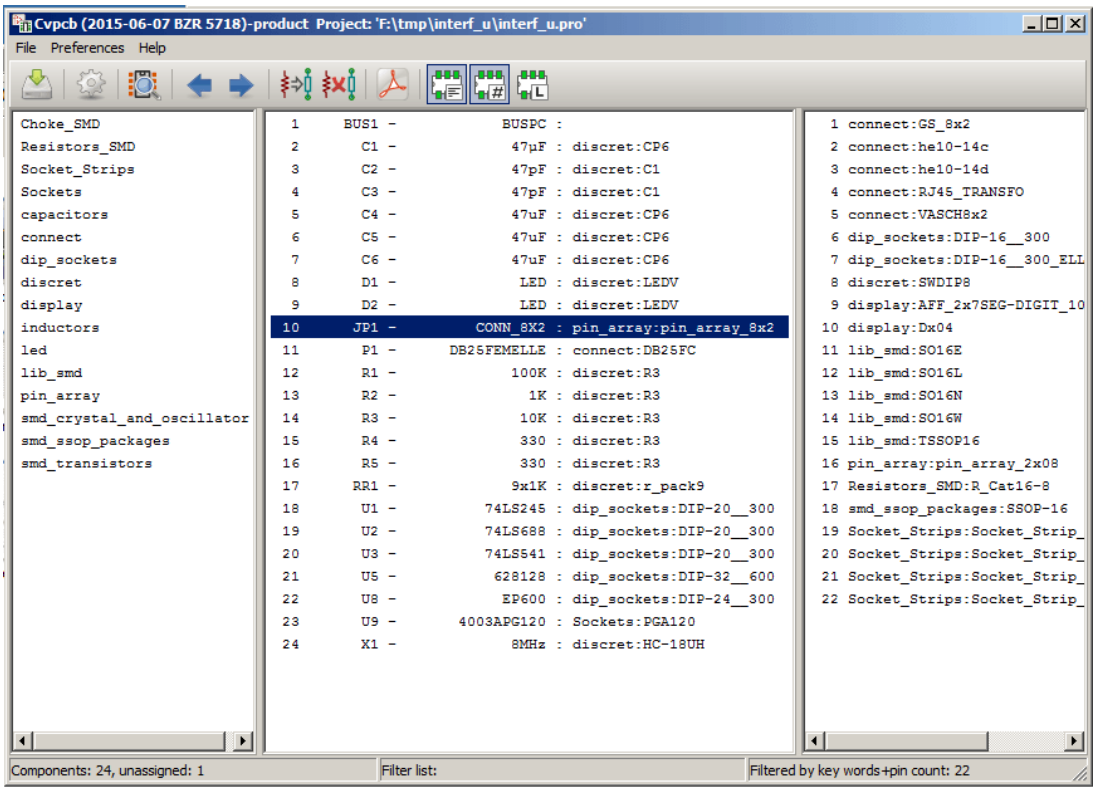


根据选择的封装库进行过滤



不同的过滤器可以叠加作用, 实现复杂的过滤需求. 这可以帮助减少右侧窗格中的封装数目, 方便查询.

根据选定元器件的引脚和元器件过滤器进行过滤:


Cvpcb (2015-06-07 BZR 5718)-product Project: 'F:\tmp\interf_u\interf_u.pro'
File Preferences Help
Choke_SMD
Resistors_SMD
Socket_Strips
Sockets
capacitors
connect
dip_sockets
discret
display
inductors
led
lib_smd
pin_array
smd_crystal_and_oscillator
smd_ssop_packages
smd_transistors
1 BUS1 - BUSPC :
2 C1 - 47µF : discret:CP6
3 C2 - 47pF : discret:C1
4 C3 - 47pF : discret:C1
5 C4 - 47uF : discret:CP6
6 C5 - 47uF : discret:CP6
7 C6 - 47uF : discret:CP6
8 D1 - LED : discret:LEDV
9 D2 - LED : discret:LEDV
10 JP1 - CONN_8X2 : pin_array:pin_array_8x2
11 P1 - DB25FEMELLE : connect:DB25FC
12 R1 - 100K : discret:R3
13 R2 - 1K : discret:R3
14 R3 - 10K : discret:R3
15 R4 - 330 : discret:R3
16 R5 - 330 : discret:R3
17 RR1 - 9x1K : discret:r_pack9
18 U1 - 74LS245 : dip_sockets:DIP-20__300
19 U2 - 74LS688 : dip_sockets:DIP-20__300
20 U3 - 74LS541 : dip_sockets:DIP-20__300
21 U5 - 628128 : dip_sockets:DIP-32__600
22 U8 - EP600 : dip_sockets:DIP-24__300
23 U9 - 4003APG120 : Sockets:PGA120
24 X1 - 8MHz : discret:HC-18UH
1 connect:GS_8x2
2 connect:he10-14c
3 connect:he10-14d
4 connect:RJ45_TRANSFO
5 connect:VASCH8x2
6 dip_sockets:DIP-16__300
7 dip_sockets:DIP-16__300_ELL
8 discret:SWDIP8
9 display:AFF_2x7SEG-DIGIT_10
10 display:Dx04
11 lib_smd:SO16E
12 lib_smd:SO16L
13 lib_smd:SO16N
14 lib_smd:SO16W
15 lib_smd:TSSOP16
16 pin_array:pin_array_2x08
17 Resistors_SMD:R_Cat16-8
18 smd_ssop_packages:SSOP-16
19 Socket_Strips:Socket_Strip_
20 Socket_Strips:Socket_Strip_
21 Socket_Strips:Socket_Strip_
22 Socket_Strips:Socket_Strip_
Components: 24, unassigned: 1
Filter list:
Filtered by key words+pin count: 22

自动关联

Equivalence 文件

Equivalence 文件可以帮助用户自动为元器件分配封装.

它会根据元器件的名称属性 (*value field*) 列出与之对应的封装. Equivalence 文件的文件扩展名为 **.equ**

Equivalence 文件格式

equ 文件中每一行对应一个元器件. 每行的格式如下: ‘元器件值’ ‘封装名’

每个名称都应该被单引号括起, 不同的封装名应该由一个或者多个空格隔开.

例子:

如果 U3 是 14011, 它的封装是 14DIP300, 那么其对应行应写为:

‘14011’ ‘14DIP300’

开头的行为注释行.

Equivalence 文件示例:

```
#Integrierte Schaltkreise (SMD):
'74LV14' 'SO14E'
'74HCT541M' 'SO20L'
'EL7242C' 'SO8E'
'DS1302N' 'SO8E'
'XRC3064' 'VQFP44'
'LM324N' 'SO14E'
'LT3430' 'SSOP17'
'LM358' 'SO8E'
'LTC1878' 'MSOP8'
'24LC512I/SM' 'SO8E'
'LM2903M' 'SO8E'
'LT1129_SO8' 'SO8E'
'LT1129CS8-3.3' 'SO8E'
'LT1129CS8' 'SO8E'
'LM358M' 'SO8E'
'TL7702BID' 'SO8E'
'TL7702BCD' 'SO8E'
'U2270B' 'SO16E'
#Xilinx
'XC3S400PQ208' 'PQFP208'
'XCR3128-VQ100' 'VQFP100'
'XCF08P' 'BGA48'

#upro
'MCF5213-LQFP100' 'VQFP100'
```

```
#Spannungsregler
'LP2985LV' 'SOT23-5'
```

自动为元器件分配封装

点击位于顶部工具栏的自动分配按钮以解析 Equivalence 文件.

所有在*.equ* 文件中能找到相关记录的元器件, 将会被自动分配封装。