How pcb−gcode Works

EAGLE has it's own programming language that is similar to the C programming language. It has several enhancements that make it more suitable to writing EAGLE ULPs, such as better text handling and dialog creation.

When a ULP is run from EAGLE using `run pcb-gcode`, the ULP can return something back to EAGLE. It can return a number to let EAGLE know if the command executed okay, or it can return a string of text. If the ULP returns a string, EAGLE will run any commands in that string, just as if they had been typed into the command line in EAGLE. This enables pcb−gcode to control EAGLE and to set various settings and create objects needed to make the NC files. pcb−gcode also uses this ability to re−run itself after the commands have run. It could be thought of as, Do these things, when you're through, call me back. When you run pcb−gcode, the ULP actually runs itself several times. Each of these runs is called a ***phase***, and creates a different part of the NC files. pcb−gcode passes a number back and forth[13] to keep up with where it is in the process. The phases are as follows:

1. Generate ***outlines*** for the top layer.
2. Write the top outline commands to an NC file.
3. Generate the top ***fill*** lines.
4. Write the top fill lines to an NC file.
5. Generate outlines for the bottom layer.
6. Write the bottom outline commands to an NC file.
7. Generate the bottom fill lines.
8. Write the bottom fill lines to an NC file.
9. Generate the top ***drill*** commands and write to an NC file.
10. Generate the bottom drill commands and write to an NC file.
11. Generate the ***mill***ing commands and write to an NC file.

When the outlines are generated[3], EAGLE is told to set the amount of isolation and a few other options, then a rectangular ***polygon*** is created on the appropriate ***layer*** (top or bottom) using the special signal name `_OUTLINES_`. When a `ratsnest` command is executed, EAGLE changes the solid rectangle into a bunch of pieces that fill all the spaces around the tracks[15], leaving `isolate` space between them. You could think of this like coloring in a picture in a coloring book, but leaving a space around each line. Now what we have is a polygon with an outline exactly where we want to move our cutting tool on the board. The next phase finds the signal name `_OUTLINES_`, and writes each line in the polygon to an NC file[4], adding all the tool up, tool down, etc. code. Phases 1, 2 and 3, 4 are repeated[5] with an increasing isolate parameter. That is how multiple passes around the tracks are created. The isolate parameter begins at the default isolation[6] settings, and increases by the step size[7] until it is greater than the maximum isolation setting[8].

Fill files are created using generate_outlines and write_outlines in the same fashion as the outlines. *I don't believe anyone uses the fill files*.

The NC files for drilling holes are created[11] by gathering all the holes from the board, all the vias from the signals, all the pads from the package contacts, and all the holes from the packages into one place[12]. They are sorted according to the drill size, and the drill files are created[9].

Milling NC files are created[10] by looking at all the wires and arcs on the milling layer on the board, and creating an NC file for them.

Those four tasks, ***outline***, ***fill***, ***drill***, ***mill*** generate all the NC files. The NC code is actually created in several other functions. Outline, fill and mill use the `device_init`, `device_draw`, and `device_end` functions to write the spindle commands, movement commands, and end program commands. In turn, `device_init`, `device_draw`, `device_end` and drill all use functions in the `pcb-file-utils.h` file to write the code to the file. The functions in `pcb-file-utils.h` can, in turn, use functions in user−gcode.h to generate the user's custom NC code.

All the NC command codes are defined in `gcode-defaults.h`. When you select a profile in the setup[14] program, the profile file (such as `emc.pp`) is copied to `gcode-defaults.h`. The NC commands for user g−code are defined in `user-gcode.h`. The `user-gcode.h` file is for advanced users who want to insert their own code at particular steps in the NC files. This could be used to orchestrate using a tool changer, loading pcb stock, etc. There is a separate document on setting up `user-gcode.h`.

The settings in `pcb-defaults.h` define the isolation, step, and maximum isolation parameters, as well as which files (outline, drill, etc.) should be created, and a couple of other options pertaining to the board itself. All these settings are set in the `Generation Options` tab of the setup program.

Settings that pertain to an individual machine are defined in `pcb-machine.h`. This includes Z up position, tool change position, feed rates, etc. All these settings are set in the `Machine` tab of the setup program.

Settings in the `pcb-gcode-options.h` file control various comments inserted in the NC files, as well as the default extensions to use for the NC files.

See Also

The Yahoo! pcb−gcode group.

Footnotes

[1] http://en.wikipedia.org/wiki/C_programming_language

[2] http://en.wikipedia.org/wiki/String_%28computer_science%29

[3] Phases 1 and 5, using the generate_outlines function.

[4] Phases 2 and 6, using the write_outlines function.

[5] Using the next_phase function.

[6] In pcb−gcode−setup, Generation Options tab, Isolation group, Default.

[7] In pcb−gcode−setup, Generation Options tab, Isolation group, Step size.

[8] In pcb−gcode−setup, Generation Options tab, Isolation group, Maximum.

[9] The drill function does most of the work, using add_hole and the stack routines from pcb−gcode−stack.h.

[10] In the output_mill_code function.

[11] In the drill function.

[12] The place is a stack implemented in pcb−gcode−stack.h

[13] In the variable g_phase.

[14] Using `run pcb-gcode --setup` from the command line in EAGLE.

[15] This is called a ***pour***, it's as if you poured copper onto the board to fill the spaces.