Gent

Author: chaered@gmail.com

Version: 1.6

Date: 2010-03-18

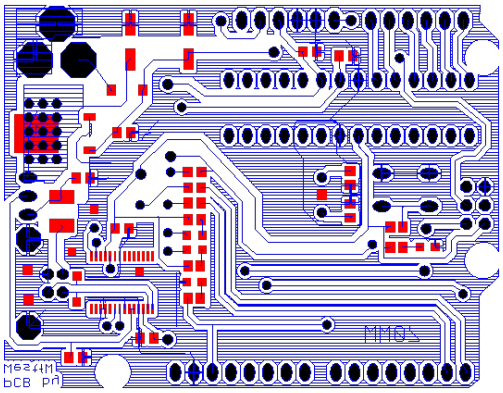

Table of Contents

1. Introduction

This document contains the development documentation for *gent*, an open-source program to visualize PCB (printed circuit-board) designs exported by a CAD application in RS-274X format, also known as the Gerber extended version of RS-274D.

The current Gent version is 0.1-7.

1.1. Acknowledgements

Thanks to user *FlyingElectron* at SourceForge for the original idea for doing this project in the first place, and for all the information and sample cases contributed during the project development.

2. Project

The *gent* project is being developed by me, as user *tinco* of SourceForge.net, and hosted under *http:://qtgent.sourceforge.net/*. It reuses some of the code developed for the *qtamaze* project by the same user.

3. Versioning

The sources at SourceForge are kept under CVS, and (with the exception of binary files) have a "$Revision" keyword expanded to their individual version. We're not using any branches at this point, so all file versions should be of the form **1.***x*.

We keep track of a tripartite overall product version number *major.minor-release (e.g "1.2-3")*, used to identify the downloadable binary installations. It is maintained on the last line of two files: *VERSION* and *RESEQNO*. All the places where the version numbers are used are derived from this. To change the numbers, update those two files, then run *gent/tools/upversion.sh*, then do "cvs commit".

The uploaded binaries are named e.g. *gent-1.2-3.msi*, but all releases for a version are also saved under CVS in *ububin* and *winbin* using the same name without the release number, e.g. *gent-1.2.msi*.

Uploading the web pages is semi-automated through the *web/upweb* script. Uploading the binaries to SourceForge is still a manual process.

4. Testing

To use the prototype, install and start it. Then use File>New to clear the canvas, File>Open to read and overlay some Gerber files on the canvas, and File>Export to store the result in a PNG.

To test the single-step view mode on a file *xxx.cmp*, try the following command:

```
gent.exe -do-view  -active -load xxx.cmp
```

then hit Ctrl-B (=single step) lots of times. The single-step mode has some rough edges being worked on.

5. Open issues

Open issues, which can be either bugs or feature requests, are sometimes also tracked in tickets in the *qtamaze* tracker system of SourceForge, as lines in the *gent*(1) man-page, in the *TODO* list in CVS, on the Wiki page *https://sourceforge.net/apps/mediawiki/qtgent/index.php?title=To_do*, and in “XXX” marked comments in the sources. This document is for a more detailed or comprehensive look at open issues.

5.1. General

The projects is still in the planning phase, the current code is just a prototype to test the RS-274X scanner. The menus, GUI lay-out, internal control flow etc. are not anywhere near usable.

5.2. Scanner

The most evolved part is the RS-274X scanner, class *GerberScan*. This takes a text file (stored internally as a *GerberData* object), and parses it to produce an image by calls to a given *QPainter*. The data object is designed so we can easily create a subview (used now to expand aperture macros, maybe

later for include files), and error methods have hooks so we could tie them into a text widget for highlighting an error fragment.

Ok	***Cmd***	***Function***	***Remarks***
*	AS	axis select	We currently ignore the A/B mapping.
*	FS	format statement	
*	MI	mirror image	
*	MO	mode of units	
*	OF	offset	
*	SF	scale factor	
-	IJ	image justify	
*	IN	image name	
*	IO	image offset	
*	IP	image polarity	
-	IR	image rotation	
*	PF	plotter film	Scanned but ignored.
*/-	AD	aperture description	Unimplemented operator '='. Not tested with multi-part macros.
*	AM	aperture macro	
-	KO	knockout	
*	LN	layer name	
*	LP	layer polarity	
*	SR	step and repeat	
-	IF	include file	Need to add an included-from reference in *GerberScan* so we can report context of location in error message.
*	N	sequence number	Parsed but ignored.
*/-	G	general	Unimplemented: G00 and G45.
*	D	plot	Exposure flags are ignored.
*	M	miscellaneous	

Some open scanner issues:

1. We don't support some commands at all yet, see table above.
2. We do properly scan some information, such as A/B mapping and exposure flags, and correctly

update the state information in *struct state_t*, but then ignore this state when actually drawing the picture.

3. The mapping of shape descriptions in *GerberScan::expand(...)* to *AperPart* objects is very partial, we only handle a few of the cases.
4. The *GerberScan::flash(...)* method, tasked with painting the aperture shapes onto a canvas, now only interprets a small subset of valid *AperPart* shapes, specifically non-rotated ovals and rectangles (both only without holes).and some polygons.

5.3. Export

1. We only offer export to PNG and GIF.
2. There is no way to adjust the scale, x/y flip, colors, image quality, etc.
3. In short, it's barely good enough for testing the scanner, let alone for actual use.
4. Not started on the real goal yet, which is 3D visual export. No clue even where to find the part lists etc. in a description.

5.4. Coding

So many holes... see the Wiki "To do" page.

The scanner stepper (classes *GerberView* and *GerberState*) is a construction zone right now, and not very useful yet. Need more work to figure out how to tie together the scanner, error traces, text field editing, location references, and the view actions. The state view GUI is also incomplete (current X-Y-I-J, etc.).

Note: When generating the MSI using WiX on WinXP, we get warnings like this:

```
gent.wxs(50) : warning LGHT1076 : ICE60: The file QtCoreDLL is not a Font,
and its version is not a companion file reference. It should have a language
specified in the Language column.
```

This is due to the Qt DLLs lacking a "language" property, not a problem in the build scripts. It may get fixed in some future Qt version.

5.5. Install and ports

WinXP MSI and Ubuntu/Debian package available now. Borrowed most of the logic for that from *qtamaze*.

5.6. Localization

Not localized yet, but most of the user-visible strings are inside a "*tr(...)*" call already.

6. Implementation

Some notes on the current design and implementation.

6.1. Limitations

We only draw lines with the D01 command if the aperture is defined as a non-holed circle. We may extend this with support for using an octagonal aperture as well. According to FE, that is pretty much the standard.

We don't cover all AM shapes. Specifically: no outline, moire, or thermal.

We don't cover all AD shapes. Specifically, we don't draw any shapes with holes.

We currently ignore AB-XY mapping, mirroring, and some other state settings when doing actual drawing.

6.2. Format

The input format is RS-274X, with two additions:

1. Lines starting with "#" (number sign) are skipped. They can't occur inside a data block, they are lexically treated like commands.
2. Lines starting with ":" (colon) are internal debugging switches. The set of commands and their syntax and semantics are constantly changing, it is just meant for internal development and testing. See Wiki page "Debug command" on *https://sourceforge.net/apps/mediawiki/qtgent/index.php?title=Debug_command*.

6.3. Aperture expansion

In principle we could do the interpretation of the aperture macro definition strings in one of three places: when we encounter the AM command, when we expand it in the AD command, and finally when we do the D command for the aperture reference. The first may be fastest, since we would not have to reinterpret the digits each time, but it would mean designing an intermediate expression tree to hold the operators and $-references. The last is slowest since it would mean deferring the expansion to repeat in every D reference. Doing it in AD seems the best compromise, also since it can ultimately define a single object shape for each aperture, and translate each D command to a linked instance of that shape. See classes *Aperture* and *AperPart*.

6.4. Picture

The drawing strategy is now as follows:

1. The scanner and the picture canvas point to a single *QPicture* object. The main *Gent* object initializes a painter on it.
2. The *main(...)* function reads a series of files, before entering the main event loop. Each file is read as a *GerberData* block, then passed to *GerberScan* for interpretation.

3. The scanner implements "D" commands by calling *GerberScan::flash(...)* (except for plain D01/D02 line commands), which draws shapes using the single painter.
4. The main loop calls *end()* on the painter, so the *QPicture* object is complete.
5. In *Canvas*, the overridden *paintEvent(...)* gets the picture's boundary information, and draws the (scaled, flipped, translated) picture onto the visible area after clearing the background.
6. The File>Export action uses the same *QPicture*, but paints it onto a *QImage* and then uses a *QImageWriter* to write it out to a file.

Update:

The first version had a single *Canvas* object, a single *QPicture* that the *GerberScan* would draw on, and when the canvas needed repainting it would draw the *QPicture* to the canvas. The problem was that this made it impossible to draw in between, or add a second scan, because a new painter *begin()* wipes a *QPicture* clean.

The second version used a *PicSet* class to encapsulate a series of *QPicture* instances, so painting could be done in increments. The *Canvas* class changed to paint from a *PicSet* instead. The problem was that there was no way to separate the drawing of the PCB layers, and all colors were hard-coded in the *GerberScan* class itself.

The third version prepped for multi-layer handling by wrapping the *PicSet* in a new *Plane* class to represent a board layer attributed with a color, name, layer type, and visibility toggle. Canvas was changed to indirect via the *Plane* instance. This still stuck to a single layer though.

The fourth version added class *Deck* to hold a series of *Plane* (PCB layer) objects, holding common attributes like scale and overall background color, and a companion GUI class *DeckView* providing a control panel for the layer stack. *Canvas* was changed again, to paint all visible planes in the deck.

Currently, I'm working on GUI actions to make *GerberScan* use the right plane in the deck, and change color composition of the picture.

Note: I wasted way too much time on *DeckView*, which is a relatively minor panel and of probably little interest to most user, due to running into memory management and interface issues when trying to use *QGridLayout* to represent a dynamically changing table. In retrospect, and if not trying to use this as a learning experience with Qt, I would and should have ditched the whole attempt earlier on and switched to a hardwired, pre-allocated or static, fixed-size matrix and grid.

6.5. 3D view

The 3D view port below the 2D canvas area is very experimental now. It uses the Qt4 OpenGL library. It tends to crash, colors are off, commands are undocumented, no visible controls, etc. No backdrop yet. Early stages.

7. References

For standard references:

1. http://en.wikipedia.org/wiki/G-code

2. http://www.linuxcnc.org/handbook/gcode/g-code.html

For component package dimensions:

1. http://www.panasonic.com/industrial/components/pdf/AOA0000CE1.pdf
2. http://www.digikey.com/
3. http://www.fairchildsemi.com/products/discrete/pdf/soic8_dim.pdf