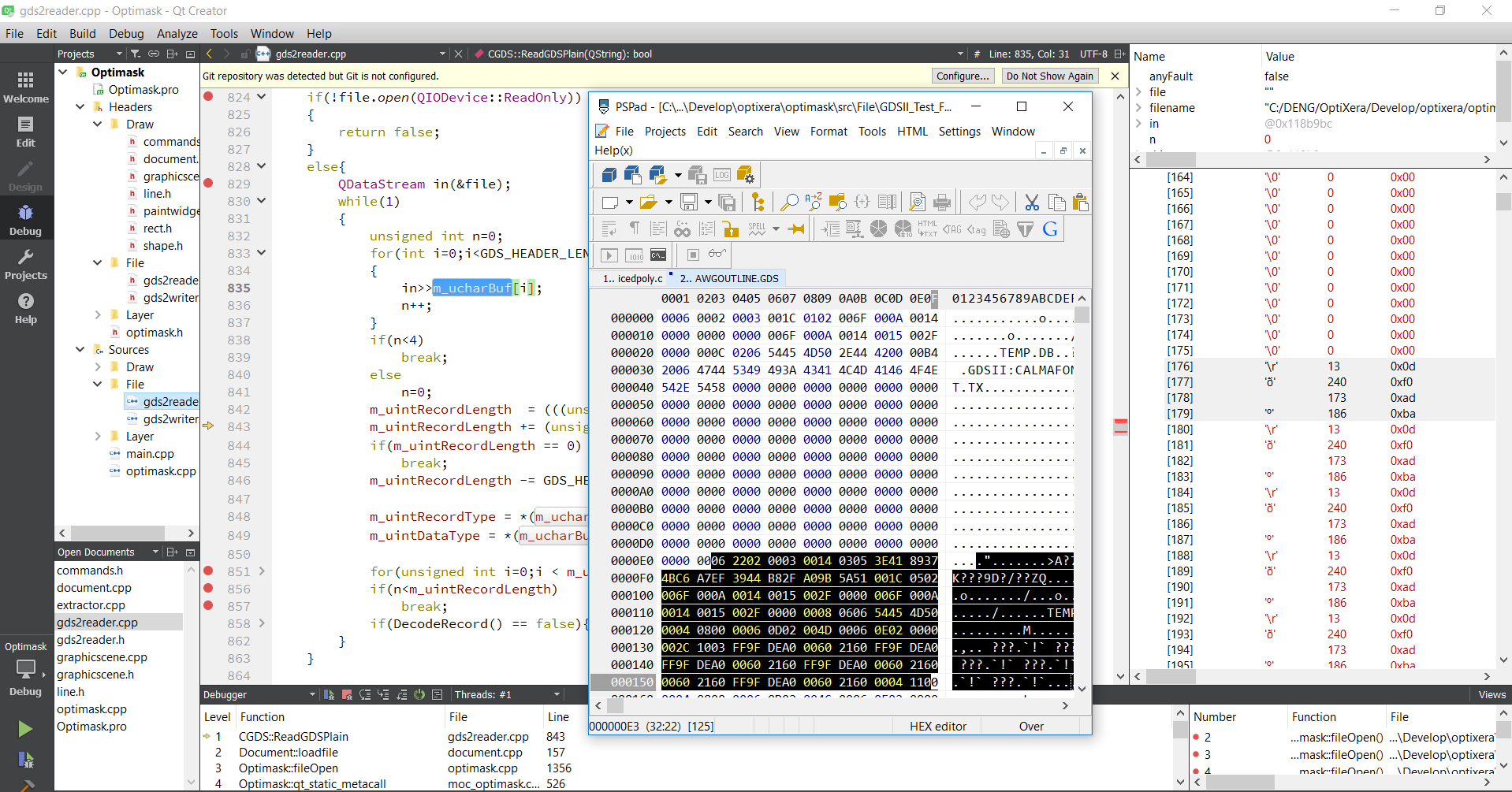
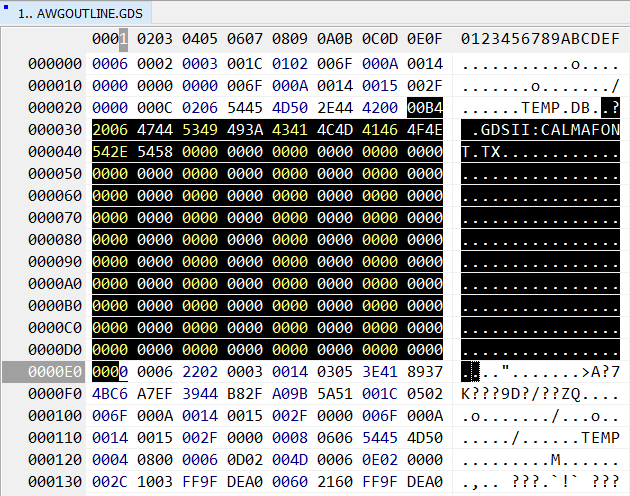
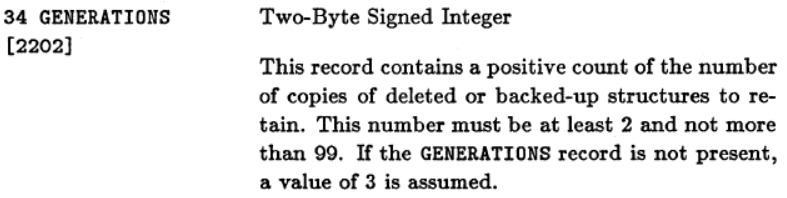
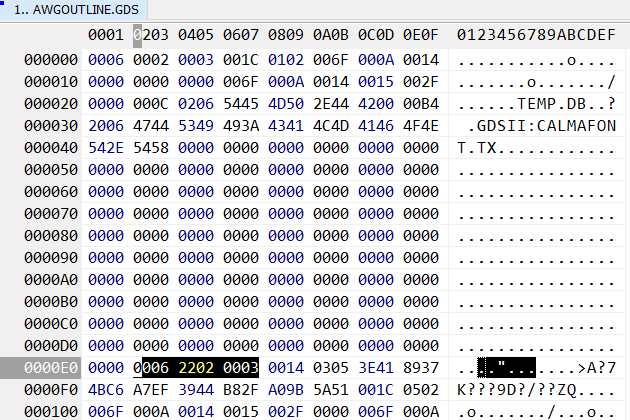
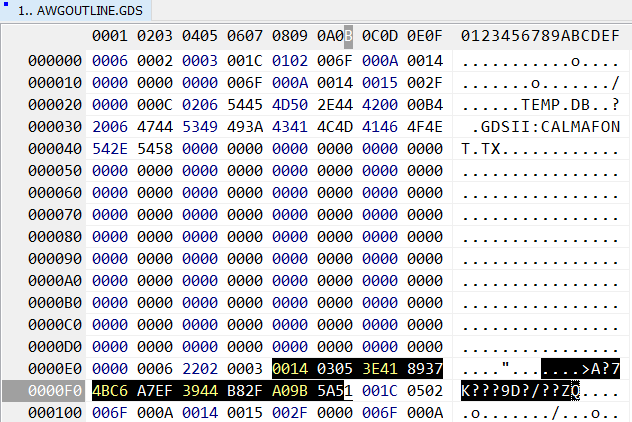
调试记录

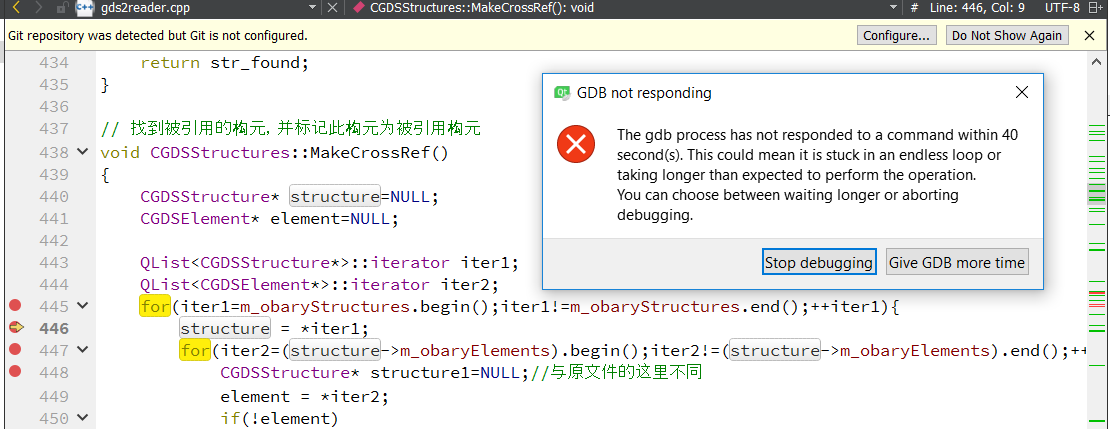
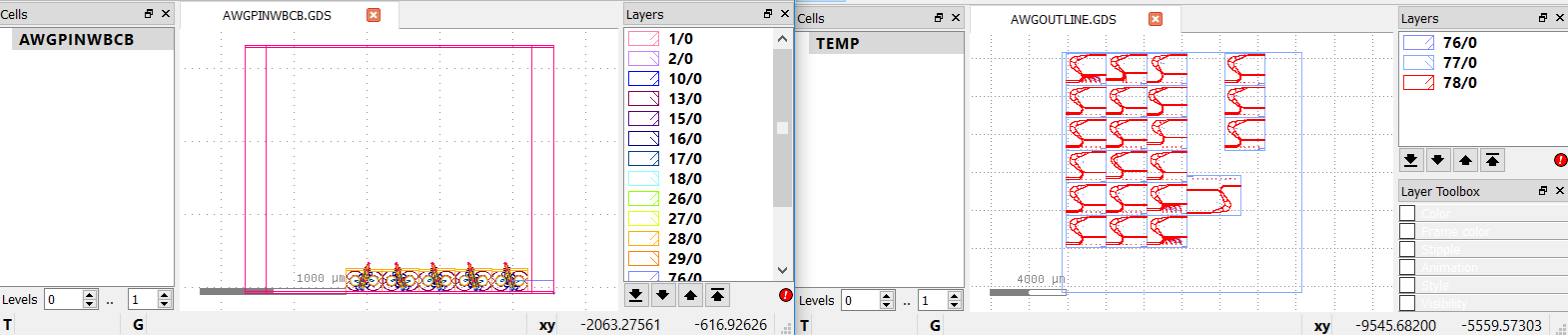
**GDSII文件读取死循环调试20170422**

其实可以直接用断点调试程序debug，查找死循环的位置。  
见下图，打开AWGOUTLINE.GDS时，此文件比其它文件不一样处是，开始处有很大一块全0的区域。你的程序读完全0区域之后，马上出错了。  
在gds2reader.cpp的835行m\_ucharBuf变量读出来的[176][177][178][179]四个字节之后一直死循环重复(见图右方观察窗内选中的四行）。对应的GDSII二进制文档打开之后却应该是  06 2202 0003 等等如选中所示。  
（在设置断点时，如果想查看某变量，就选中该变量，点击右键，选择“Add Expression Evaluator"即可添加查看）  
请用我上述方法设置断点，Debug-->Start Debugging来解决这个死循环问题）  


华丽C:\Users\Deng\AppData\Local\Temp\}1QQZE8R6JU3JNL5~{]RQ[0.png联(1816023692)  11:28:24 PM  
还有，gds2reader.cpp的830行 while(1) ｛｝  
不建议使用永真的while(1)，如果break设置不好，可能就漏网之鱼变死循环了。  
建议使用读取    ”剩余的字节长度  ” 为循环条件，这样循环一定会抵达一个确定的结束条件。

见 \optixera\optimask\dev\GDSII格式研究.docx 第5页，同理应用于AWGOUTLINE.GDS如下。选中的区域为  
【FONTS】 Record（字体）  
00B4 = 180个比特 （16进制00B4 = 10进制 180。所有字节如选中所示）；  
2006 = FONTS （此记录型号本身占掉2-byte Signed Integer）；  
接下来的174比特（180-2-2=174）（348个1-byte String）是Fonts信息；  
而该区域大部分字节为00。  
  
然后继续到下一个区域，应该是0006 2202 0003，如下。  
0006表示6个字节；2202查表发现是Generations，表示备份数（查GDS\_II\_Stream\_Format\_Manual\_6.0\_Feb87.pdf第4-14页）；0003是缺省值。  
  
Generations这一段在GDSII格式研究.docx文档里面是没有的，可能你就是因为没有考虑这个条件，死循环在此了。  
之后的区段走入正常，下面是【UNITS】 Record（长度单位），见 \optixera\optimask\dev\GDSII格式研究.docx 第6页；  
  
之后是 【BGNSTR】 Record（Begin of Structure，结构起始）：001C0502006F000A00140015002F0000006F000A00140015002F00000008  
一切走入正常。。。  
  
   
华丽C:\Users\Deng\AppData\Local\Temp\}1QQZE8R6JU3JNL5~{]RQ[0.png联(1816023692)  10:46:45 AM  
关于   读取    ”剩余的字节长度  ” 为循环条件，以AWGOUTLINE.GDS为例，  
PSPad打开文件AWGOUTLINE.GDS可以看到文件结尾地址为  0x 3247 0E0F  = decimal 843517455，这个即为文件总字节数。  
你应该有办法在读取开始时查到（或者算到）这个总字节数，设置它为计数器（即循环条件）。  
之后每次读取一段之后（读取不成功处，可以报错后前行），该计数器减掉已读取的长度。当文件读到结尾时，计数器会变为0，此时循环中断即可。  
这样循环一定会抵达一个确定的结束条件。

华丽C:\Users\Deng\AppData\Local\Temp\}1QQZE8R6JU3JNL5~{]RQ[0.png联(1816023692)  10:59:57 AM  
关于 " Qt里面的调试不怎么会用，我尝试用QDebug调试吧。"  
其实我也是初次使用Qt Debug，觉得并不难。（我不知道QDebug，说不定比Qt Debug更复杂。）  
程序调试基本思路是一致的：即顺着代码流程走，在关键处设置断点，显示查看关键变量值及其变化。  
作为常识，我认为程序代码的可读性，可移植性，可维护性等非常重要。好的程序一定要让他人很容易理解，你去读别人的代码也希望很容易理解。希望写代码时一直有这个原则，代码才会高质量。  
上次说过，写代码和写文章是类似的--希望思路清楚，语句明确，言简意赅。

关于   读取    ”剩余的字节长度  ” 为循环条件，以AWGOUTLINE.GDS为例，  
PSPad打开文件AWGOUTLINE.GDS可以看到文件结尾地址为  0x 3247 0E0F  = decimal 843517455，这个即为文件总字节数。  
你应该有办法在读取开始时查到（或者算到）这个总字节数，设置它为计数器（即循环条件）。  
之后每次读取一段之后（读取不成功处，可以报错后前行），该计数器减掉已读取的长度。当文件读到结尾时，计数器会变为0，此时循环中断即可。  
这样循环一定会抵达一个确定的结束条件。  
华丽C:\Users\Deng\AppData\Local\Temp\}1QQZE8R6JU3JNL5~{]RQ[0.png联(1816023692)  10:59:57 AM  
关于 " Qt里面的调试不怎么会用，我尝试用QDebug调试吧。"  
其实我也是初次使用Qt Debug，觉得并不难。（我不知道QDebug，说不定比Qt Debug更复杂。）  
程序调试基本思路是一致的：即顺着代码流程走，在关键处设置断点，显示查看关键变量值及其变化。  
作为常识，我认为程序代码的可读性，可移植性，可维护性等非常重要。好的程序一定要让他人很容易理解，你去读别人的代码也希望很容易理解。希望写代码时一直有这个原则，代码才会高质量。  
上次说过，写代码和写文章是类似的--希望思路清楚，语句明确，言简意赅。  
华丽C:\Users\Deng\AppData\Local\Temp\}1QQZE8R6JU3JNL5~{]RQ[0.png联(1816023692)  12:00:37 PM  
@潘雄 我之前的死循环位置并没有检测准确，才发现你的m\_ucharBuf对每一区段是重新读取的。但是调试方法和原则依然是有效的。只要你能够显示输出读取过程中的文件字节内容，字节地址，和当前时间，查看该输出应该就可以定位出问题的位置了。仍然建议你显示当前字节的地址，以及使用读取    ”剩余的字节长度  ”计数器 为循环条件。  
华丽C:\Users\Deng\AppData\Local\Temp\}1QQZE8R6JU3JNL5~{]RQ[0.png联(1816023692)  1:11:17 PM  
@潘雄 应该找到问题的位置了。GDSII文件AWGOUTLINE.GDS读取实际上是完成了的，但是从bool CGDS::ReadGDSPlain(QString filename) 进入 MakeCrossRef()  （第869行）后在双重循环开始时出错。  
  
AWGOUTLINE.GDS文件没有构元分级，只有一级（FLAT），这是要注意的。  
华丽C:\Users\Deng\AppData\Local\Temp\}1QQZE8R6JU3JNL5~{]RQ[0.png联(1816023692)  1:25:23 PM  
@潘雄 @朱杏子 目前程序对于然后FLAT的构元结构都会出错。就是MakeCrossRef()没有处理好特殊情况。  
我随便同理类推测试AWGPINWBCB.GDS，果然也进入死循环。  
注意对于程序和算法，特殊情况和边界条件非常重要。请时刻注意。  
并不是很建议重写所有代码，要记住代码是要经历过很多次调试的（包括准确性和速度效率）。如果有现成的尽量先借鉴着，等我们框架好了之后再全部重新写。  
华丽C:\Users\Deng\AppData\Local\Temp\}1QQZE8R6JU3JNL5~{]RQ[0.png联(1816023692)  1:28:17 PM  
程序对于任何FLAT（没有构元分级）的构元结构都会出错。就是MakeCrossRef()没有处理好特殊情况。