Reverse Engineering the Gigabyte G1 GTX 980 Ti LED System and OC Guru II

hoff.industries

July 24, 2020; Revised March 21, 2024

1 Acknowledgements

My reverse engineering of the OC Guru II software could not have been done without essential groundwork done by Adam Honse from the OpenRGB project. He has previously reverse engineered the newer version of Gigabyte's control software, Aorus Engine and his work from that has provided invaluable help during my attempts at OC Guru II.

2 Definitions

- *OC Guru II* – Gigabyte's software control centre for their graphics cards up to and including Maxwell (900 series).
- *De-compiler* – A program which takes a compiled piece of software (for example an exe or dll) and attempts to convert it back into source code for analysis.
- *API* – Application Programming Interface; a piece of software that provides a predefined library (functions, definitions, variables, etc.) for interacting with other pieces of software or hardware.
- *DLL* – Dynamic Link Library; A module consisting of functions to be used by other applications. DLLs are given the .dll extension.
- *Snooping* – The process of listening to a particular application and logging what functions, modules or libraries it calls.
- *Breakpoint* – A trigger point during the runtime of a program where execution is halted for analysis of the current state of the program, looking at its memory for example.
- *Call* – When an executable attempts to use a function from an API or DLL, this is known as a "call" to that function.
- I^2C – A serial communication bus commonly used in graphics card LED control systems.

3 Prior Observations on how the LED System Works

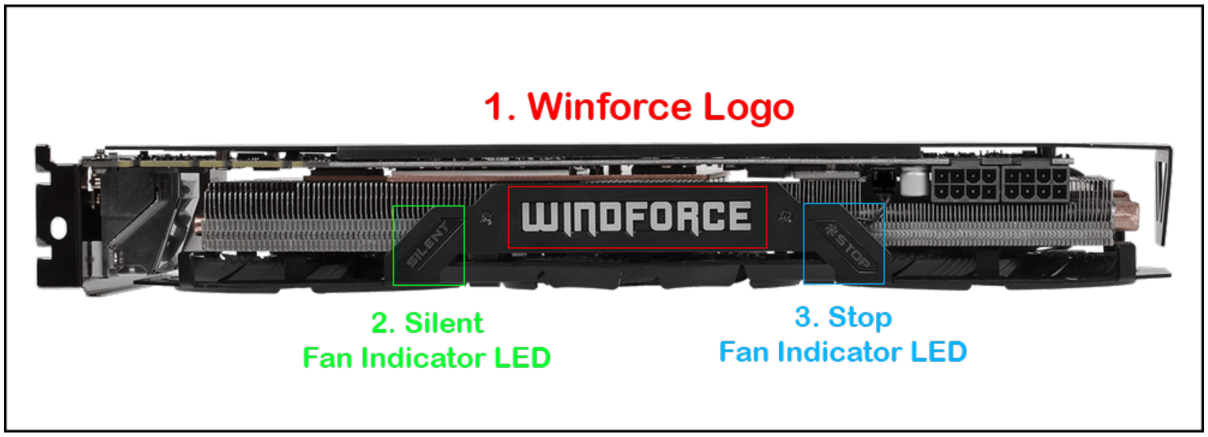


Figure 1 - G1 LED Configuration

Before doing any reverse engineering at all, there are a few behaviours shown by the LED control system that are worth noting. These are:

1. The LED settings are not persistent; after the computer is shut down or power is removed, the LEDs will reset to the default white. Thus it can be deduced that the LED colour must be set by a software program every session.
2. The default values for the LED when the computer is turned on appear to be the colour White, the mode Auto and the brightness 100%.
3. The LED system does not appear to be 16.8m RGB. OC Guru II is able to set 16.8m colours on cards that support it (the Xtreme and Waterforce versions of the 980 Ti are examples). Since OC Guru II has the capacity to set 16.8m colours, but it is unavailable on the G1, most likely the G1 does not support any more colours than the 7 available for the user to choose.
4. OC Guru II is 32-bit software. This is important for the reverse engineering process as our reverse engineering tools will have to support 32-bit applications.

4 Reverse Engineering Tools

In order to find out how LED setting works, we are going to need some tools to aid our reverse engineering efforts. These tools and what I used them for are:

1. API Monitor – Used to spy on internal and external calls to DLLs that OC Guru makes in order to find out what system services it uses. Also good for viewing and editing memory.
2. Ollydbg – Great program for stepping through a program's execution, viewing function calls and setting breakpoints.
3. Ghidra – Well known as the reverse engineering tool made by the NSA. Used this for de-compiling the DLL and executable files associated with OC Guru II.
4. Snowman – Another de-compilation tool. As de-compilation programs don't always provide the most accurate results, comparing two different ones can aid in identifying the full story on how they work.
5. Linux – the GNU/Linux operating system provides a wide library of tools for accessing and viewing hardware, and one of the goals of this project was to get LED control support on Linux.

5 Getting an Overview of OC Guru II

OC Guru II exposes 4 main options to the user for controlling the LEDs on the G1:

1. *Style* – Could also be referred to as *Effect*. Styles/Effects include 'None', 'Breathing', 'Flashing' and 'Double Flashing'.

2. *Brightness/Speed* – Brightness applies for Style None and Speed applies for Breathing, Flashing and Double Flashing. You cannot select Brightness for modes other than None, nor can you specify Speed for the mode None.

3. *Mode* – 'Auto,' 'On' and 'Off.' Controls the current state of the LED. When the mode is set to Auto the Windforce logo switches off when the fans are not spinning, and the Silent/Stop LEDs flanking it light up instead. It switches on when the fans start spinning. If the LED is set to On, the Windforce Logo is lit up all the time and the Silent/Stop are enabled when the fans are not spinning. Finally, if the mode is set to Off then all LEDs will be disabled completely.

4. *Colour* – You can select up to 7 colours, and 1 'cycle' mode. The colours available are: Blue, Red, Green, Yellow, Magenta, Cyan and White. Colour cycle goes through all of them in order and changes every few seconds.

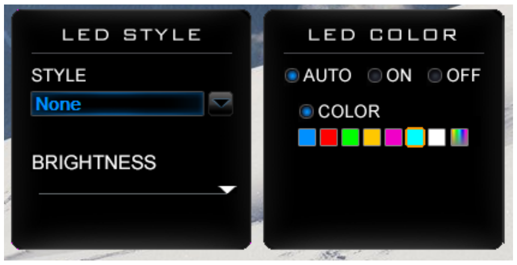
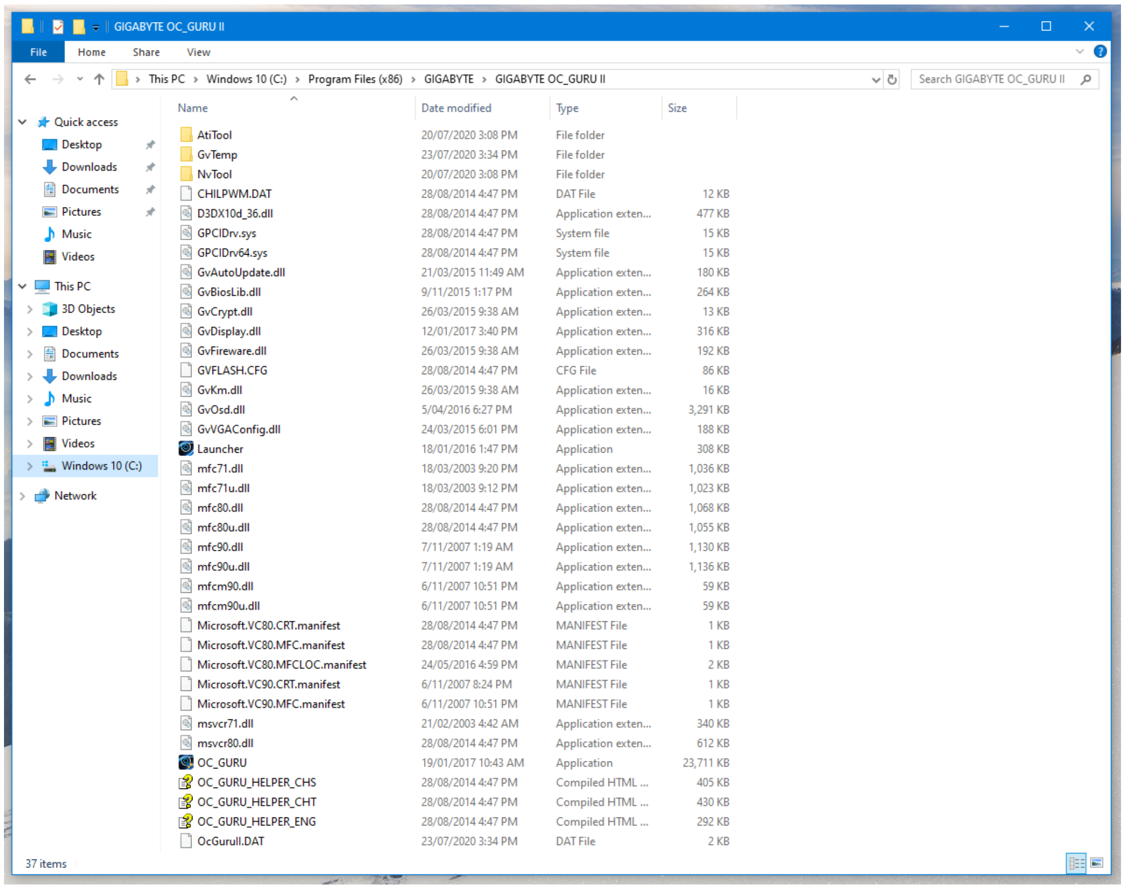


Figure 2 - OC Guru II LED Settings Menu

One of the most important things to understand when reverse engineering an application is how that application is laid out and what it's made up of. So, to get a brief overview, have a spy at the application's system folder.



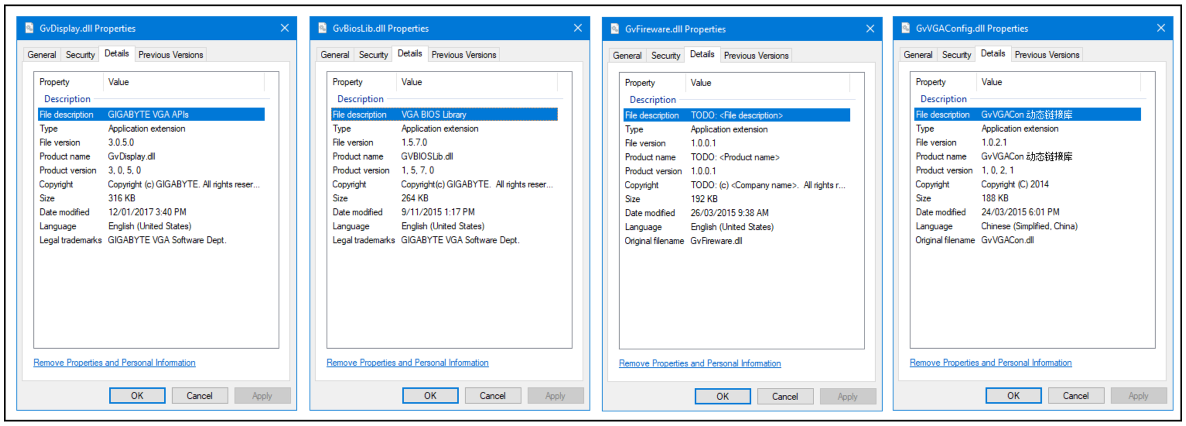
Name	Date modified	Type	Size
AtiTool	20/07/2020 3:08 PM	File folder	
GvTemp	23/07/2020 3:34 PM	File folder	
NvTool	20/07/2020 3:08 PM	File folder	
CHILPWM.DAT	28/08/2014 4:47 PM	DAT File	12 KB
D3DX10d_36.dll	28/08/2014 4:47 PM	Application exten...	477 KB
GPCIDrv.sys	28/08/2014 4:47 PM	System file	15 KB
GPCIDrv64.sys	28/08/2014 4:47 PM	System file	15 KB
GvAutoUpdate.dll	21/03/2015 11:49 AM	Application exten...	180 KB
GvBiosLib.dll	9/11/2015 1:17 PM	Application exten...	264 KB
GvCrypt.dll	26/03/2015 9:38 AM	Application exten...	13 KB
GvDisplay.dll	12/01/2017 3:40 PM	Application exten...	316 KB
GvFireware.dll	26/03/2015 9:38 AM	Application exten...	192 KB
GVFLASH.CFG	28/08/2014 4:47 PM	CFG File	86 KB
GvKm.dll	26/03/2015 9:38 AM	Application exten...	16 KB
GvOsd.dll	5/04/2016 6:27 PM	Application exten...	3,291 KB
GvVGAConfig.dll	24/03/2015 6:01 PM	Application exten...	188 KB
Launcher	18/01/2016 1:47 PM	Application	308 KB
mfc71.dll	18/03/2003 9:20 PM	Application exten...	1,036 KB
mfc71u.dll	18/03/2003 9:12 PM	Application exten...	1,023 KB
mfc80.dll	28/08/2014 4:47 PM	Application exten...	1,068 KB
mfc80u.dll	28/08/2014 4:47 PM	Application exten...	1,055 KB
mfc90.dll	7/11/2007 1:19 AM	Application exten...	1,130 KB
mfc90u.dll	7/11/2007 1:19 AM	Application exten...	1,136 KB
mfcm90.dll	6/11/2007 10:51 PM	Application exten...	59 KB
mfcm90u.dll	6/11/2007 10:51 PM	Application exten...	59 KB
Microsoft.VC80.CRT.manifest	28/08/2014 4:47 PM	MANIFEST File	1 KB
Microsoft.VC80.MFC.manifest	28/08/2014 4:47 PM	MANIFEST File	1 KB
Microsoft.VC80.MFCLOC.manifest	24/05/2016 4:59 PM	MANIFEST File	2 KB
Microsoft.VC90.CRT.manifest	6/11/2007 8:24 PM	MANIFEST File	1 KB
Microsoft.VC90.MFC.manifest	6/11/2007 10:51 PM	MANIFEST File	1 KB
msvcr71.dll	21/02/2003 4:42 AM	Application exten...	340 KB
msvcr80.dll	28/08/2014 4:47 PM	Application exten...	612 KB
OC_GURU	19/01/2017 10:43 AM	Application	23,711 KB
OC_GURU_HELPER_CHS	28/08/2014 4:47 PM	Compiled HTML ...	405 KB
OC_GURU_HELPER_CHT	28/08/2014 4:47 PM	Compiled HTML ...	430 KB
OC_GURU_HELPER_ENG	28/08/2014 4:47 PM	Compiled HTML ...	292 KB
OcGuruII.DAT	23/07/2020 3:34 PM	DAT File	2 KB

Figure 3 - The system folder for the latest and final version of OC Guru II (version 2.08).

Looking at all the files associated with OC Guru II, we can both select and rule out possibilities for what might be used in selecting the LED colours. Consider the release date of the GTX 980 Ti – the 2nd of June, 2015 (02/06/2015). Anything that has a last modified date outside a reasonable period before 02/06/2015 can be ruled out as a possibility, as it would be impossible to implement support for a card in say 2013 that releases in 2015 (Revision 21/3/24: unless they re-use the same LED controller from an earlier model). All the DLLs prefixed by 'Gv' seem to meet this criteria, and so now we'll have a look at the names to see if anything is promising. Files like 'GvAutoUpdate' probably aren't associated with changing the LEDs, so we can rule that out. Reasonable possibilities seem to be:

1. GvDisplay.dll
2. GvBiosLib.dll
3. GvFireware.dll
4. GvVGAConfig.dll

Now we can look further into these DLLs without having to do any reverse engineering at all. By simply going to the "Details" section of the properties window, we can see some interesting things:



GvDisplay.dll Properties

Property	Value
Description	
File description	GIGABYTE VGA APIs
Type	Application extension
File version	3.0.5.0
Product name	GvDisplay.dll
Product version	3, 0, 5, 0
Copyright	Copyright (c) GIGABYTE. All rights reser...
Size	316 KB
Date modified	12/01/2017 3:40 PM
Language	English (United States)
Legal trademarks	GIGABYTE VGA Software Dept.

GvBiosLib.dll Properties

Property	Value
Description	
File description	VGA BIOS Library
Type	Application extension
File version	1.5.7.0
Product name	GVBIOSLib.dll
Product version	1, 5, 7, 0
Copyright	Copyright(c) GIGABYTE. All rights reser...
Size	264 KB
Date modified	9/11/2015 1:17 PM
Language	English (United States)
Legal trademarks	GIGABYTE VGA Software Dept.

GvFireware.dll Properties

Property	Value
Description	
File description	TODO: <File description>
Type	Application extension
File version	1.0.0.1
Product name	TODO: <Product name>
Product version	1.0.0.1
Copyright	TODO: (c) <Company name>. All rights r...
Size	192 KB
Date modified	26/03/2015 9:38 AM
Language	English (United States)
Original filename	GvFireware.dll

GvVGAConfig.dll Properties

Property	Value
Description	
File description	GvVGACon 动态链接库
Type	Application extension
File version	1.0.2.1
Product name	GvVGACon 动态链接库
Product version	1, 0, 2, 1
Copyright	Copyright (C) 2014
Size	188 KB
Date modified	24/03/2015 6:01 PM
Language	Chinese (Simplified, China)
Original filename	GvVGACon.dll

Figure 4 – DLL Properties, *Details* tab.

"GvDisplay.dll" – Gigabyte VGA APIs huh? Interesting. If you're wondering, the Chinese in the "GvVGA-Config.dll" description translates to "Dynamic link library," which isn't particularly useful.

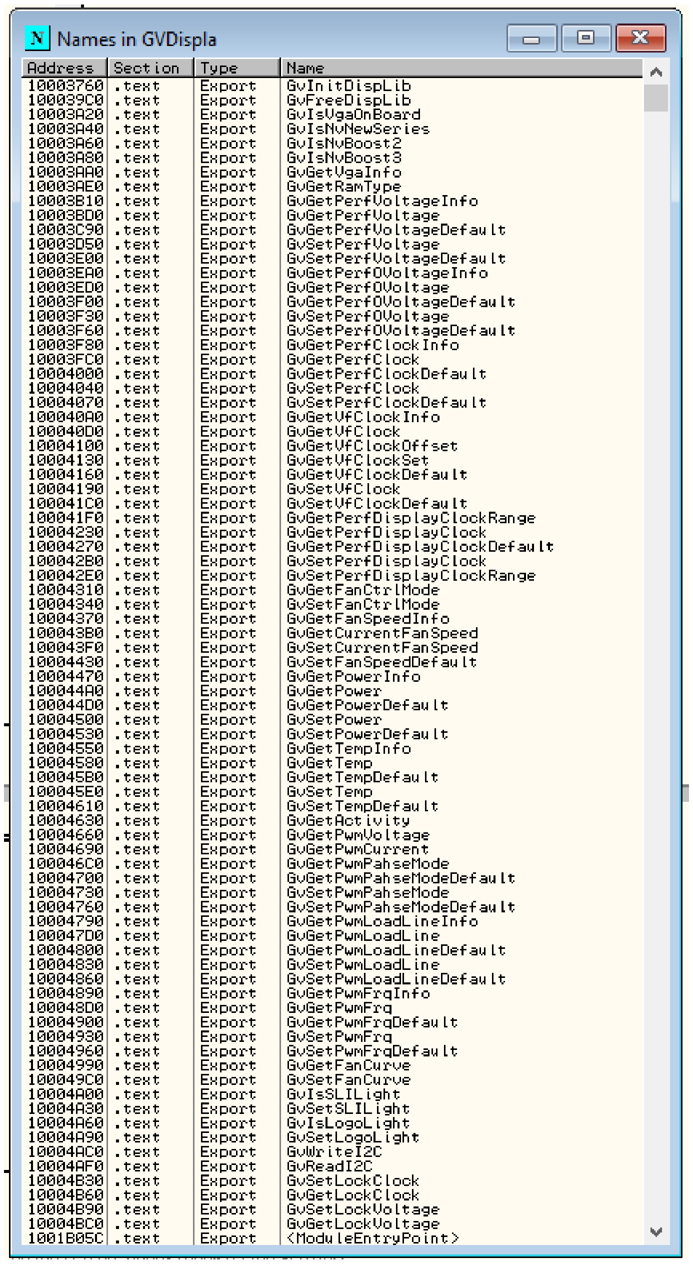
6 Investigating the Libraries Further

Now that we have a list of promising sounding libraries, it's time to investigate them in more depth. To do this, ollydbg provides a really nice way to view all the functions present in a DLL and the ability to set a breakpoint when they're called. It's also worth noting that since OC Guru II runs as administrator, to snoop on it we need to run ollydbg as administrator as well.

To use ollydbg effectively, we are going to need to set up its window so that relevant information can be displayed. In particular, enabling the call stack, memory map and executable modules will be useful for this particular session. Once we've attached or loaded OC Guru II with ollydbg, we can view the functions that are present in the DLLs we've flagged as interesting. To do this, right click on its name in the "Executable modules" window and select "View names," and click on the "Type" header to sort by type. We are only interested in the functions each DLL exports, as these will be the ones it provides to OC Guru II that enable it to access our graphics card.

Right off the bat, we can eliminate GvVGAConfig.dll and GvBIOSLib.dll as they provide no functions that appear to be useful in setting LEDs. GvVGAConfig.dll exposes only 2 named functions, "GvCfgGetPerformances" and "GvCfgSetPerformances," both of which appear to have little relevance to LED control. GvBIOSLib.dll on the other hand exposes more functions, but they all are related to loading, flashing, reading and saving BIOS ROM files and as we know from the behaviour we've seen the LEDs exhibit, they reset on power off so the LED settings are definitely not set in the vBIOS.

GvFireware.dll contains some possibilities: "GvConnectBrd", "GvGetBoards" and "GvSendCommand", but then GvDisplay.dll is where things really start to get interesting. True to its 'API' description, GvDisplay.dll exposes a ton of functions for use with the graphics card.



Address	Section	Type	Name
10003760	.text	Export	GvInitDispLib
100039C0	.text	Export	GvFreeDispLib
10003A20	.text	Export	GvIsVgaOnBoard
10003A40	.text	Export	GvIsNvNewSeries
10003A60	.text	Export	GvIsNvBoost2
10003A80	.text	Export	GvIsNvBoost3
10003AA0	.text	Export	GvGetVgaInfo
10003AE0	.text	Export	GvGetRamType
10003B10	.text	Export	GvGetPerfVoltageInfo
10003BD0	.text	Export	GvGetPerfVoltage
10003C90	.text	Export	GvGetPerfVoltageDefault
10003D50	.text	Export	GvSetPerfVoltage
10003E00	.text	Export	GvSetPerfVoltageDefault
10003EA0	.text	Export	GvGetPerfOVoltageInfo
10003ED0	.text	Export	GvGetPerfOVoltage
10003F00	.text	Export	GvGetPerfOVoltageDefault
10003F30	.text	Export	GvSetPerfOVoltage
10003F60	.text	Export	GvSetPerfOVoltageDefault
10003F80	.text	Export	GvGetPerfClockInfo
10003FC0	.text	Export	GvGetPerfClock
10004000	.text	Export	GvGetPerfClockDefault
10004040	.text	Export	GvSetPerfClock
10004070	.text	Export	GvSetPerfClockDefault
100040A0	.text	Export	GvGetVfClockInfo
100040D0	.text	Export	GvGetVfClock
10004100	.text	Export	GvGetVfClockOffset
10004130	.text	Export	GvGetVfClockSet
10004160	.text	Export	GvGetVfClockDefault
10004190	.text	Export	GvSetVfClock
100041C0	.text	Export	GvSetVfClockDefault
100041F0	.text	Export	GvGetPerfDisplayClockRange
10004230	.text	Export	GvGetPerfDisplayClock
10004270	.text	Export	GvGetPerfDisplayClockDefault
100042B0	.text	Export	GvSetPerfDisplayClock
100042E0	.text	Export	GvSetPerfDisplayClockRange
10004310	.text	Export	GvGetFanCtrlMode
10004340	.text	Export	GvSetFanCtrlMode
10004370	.text	Export	GvGetFanSpeedInfo
100043B0	.text	Export	GvGetCurrentFanSpeed
100043F0	.text	Export	GvSetCurrentFanSpeed
10004430	.text	Export	GvSetFanSpeedDefault
10004470	.text	Export	GvGetPowerInfo
100044A0	.text	Export	GvGetPower
100044D0	.text	Export	GvGetPowerDefault
10004500	.text	Export	GvSetPower
10004530	.text	Export	GvSetPowerDefault
10004550	.text	Export	GvGetTempInfo
10004580	.text	Export	GvGetTemp
100045B0	.text	Export	GvGetTempDefault
100045E0	.text	Export	GvSetTemp
10004610	.text	Export	GvSetTempDefault
10004630	.text	Export	GvGetActivity
10004660	.text	Export	GvGetPwmVoltage
10004690	.text	Export	GvGetPwmCurrent
100046C0	.text	Export	GvGetPwmPahseMode
10004700	.text	Export	GvGetPwmPahseModeDefault
10004730	.text	Export	GvSetPwmPahseMode
10004760	.text	Export	GvSetPwmPahseModeDefault
10004790	.text	Export	GvGetPwmLoadLineInfo
100047D0	.text	Export	GvGetPwmLoadLine
10004800	.text	Export	GvGetPwmLoadLineDefault
10004830	.text	Export	GvSetPwmLoadLine
10004860	.text	Export	GvSetPwmLoadLineDefault
10004890	.text	Export	GvGetPwmFrqInfo
100048D0	.text	Export	GvGetPwmFrq
10004900	.text	Export	GvGetPwmFrqDefault
10004930	.text	Export	GvSetPwmFrq
10004960	.text	Export	GvSetPwmFrqDefault
10004990	.text	Export	GvGetFanCurve
100049C0	.text	Export	GvSetFanCurve
10004A00	.text	Export	GvIsSLILight
10004A30	.text	Export	GvSetSLILight
10004A60	.text	Export	GvIsLogoLight
10004A90	.text	Export	GvSetLogoLight
10004AC0	.text	Export	GvWriteI2C
10004AF0	.text	Export	GvReadI2C
10004B30	.text	Export	GvSetLockClock
10004B60	.text	Export	GvGetLockClock
10004B90	.text	Export	GvSetLockVoltage
10004BC0	.text	Export	GvGetLockVoltage
1001B05C	.text	Export	<ModuleEntryPoint>

Figure 5 - Functions exported by GvDisplay.dll

Looking at some of the functions it exports, the ones of interest to us appear to be:

- GvSetSLILight
- GvSetLogoLight

We will find out soon using our next tool, API Monitor if these functions are actually used in setting the LEDs.

7 Identifying How the Card Communicates

The next thing we have to do is find out how the software communicates with the hardware. This is where we will use our next tool, API Monitor. Just like with ollydbg, API Monitor needs to be run as administrator, and also be sure to run the 32-bit version since OC Guru II is 32-bit.

Just to get a good overview of how the program works in general, I enabled all the options in the API Filter “Capture” tab. In addition to default Windows stuff, we should look at the DLLs that we’ve narrowed down. In the “External DLL” tab of the API Filter section (tabs are found at the bottom of this section), we can select “Add External DLL” and add GvDisplay.dll and GvFireware.dll.

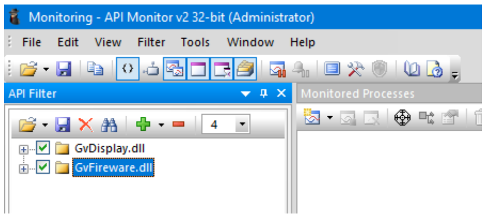
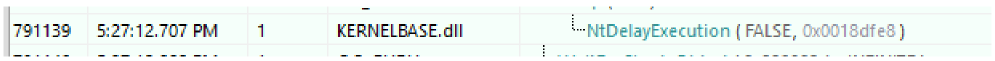

Monitoring - API Monitor v2 32-bit (Administrator)
File Edit View Filter Tools Window Help
API Filter
Monitored Processes
4
GvDisplay.dll
GvFireware.dll

Figure 6 - External DLLs tab in API Monitor with relevant DLLs enabled.

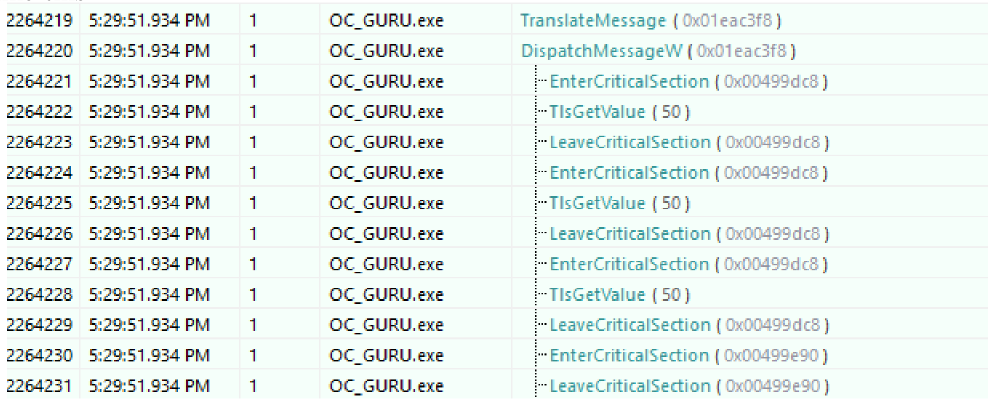
In the calls section we should make sure autoscroll is enabled. Our goal is now to identify 2 things: what call happens as soon as we press the LED change button and what call happens at the exact moment the LED changes colours. Starting to monitor OC Guru II, it constantly calls things in the background so filtering the right calls might be tricky. One way around this is simply just to spam click the colour button, then disable autoscroll and we should be able to set a breakpoint at the end. Spamming the blue colour button, the last thing that seems to happen is:



```
791139   5:27:12.707 PM   1   KERNELBASE.dll   NtDelayExecution ( FALSE, 0x0018dfe8 )
```

Figure 7 - The last function called while we spam click the colour change button.

And now we know where to set our first breakpoint. To break right after our LED change, we simply set a breakpoint before the last call that happens during the LED change. After we have this set, simply re-enable autoscroll and change the colours again and we have the calls we need. To verify this was actually the endpoint of the LED change, one can simply set a different colour and verify that before this breakpoint happened, the colour did in fact change. In this case, it’s actually pretty obvious where the LED change starts. As I mentioned before, OC Guru II constantly calls a specific set of things in the background, which look a bit like this:



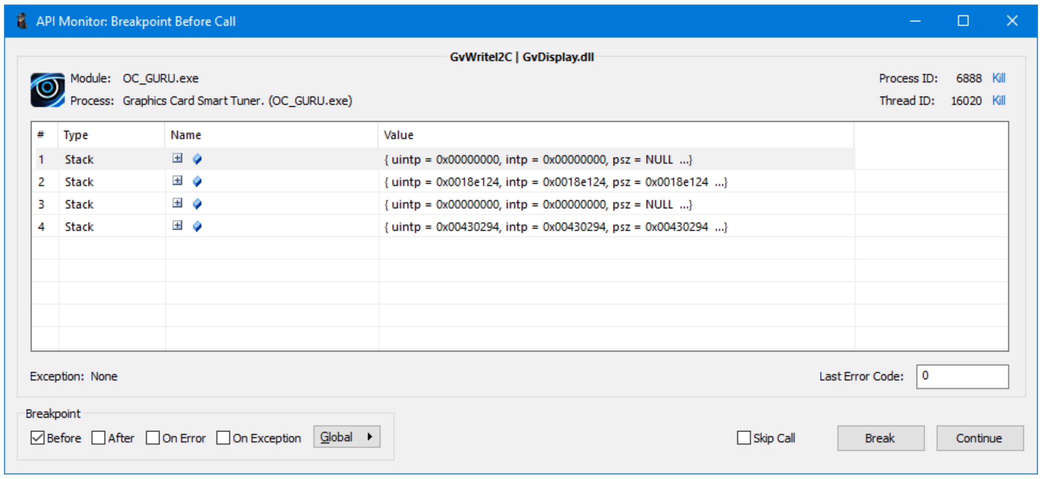
```
2264219  5:29:51.934 PM  1  OC_GURU.exe  TranslateMessage ( 0x01eac3f8 )
2264220  5:29:51.934 PM  1  OC_GURU.exe  DispatchMessageW ( 0x01eac3f8 )
2264221  5:29:51.934 PM  1  OC_GURU.exe    EnterCriticalSection ( 0x00499dc8 )
2264222  5:29:51.934 PM  1  OC_GURU.exe    TlsGetValue ( 50 )
2264223  5:29:51.934 PM  1  OC_GURU.exe    LeaveCriticalSection ( 0x00499dc8 )
2264224  5:29:51.934 PM  1  OC_GURU.exe    EnterCriticalSection ( 0x00499dc8 )
2264225  5:29:51.934 PM  1  OC_GURU.exe    TlsGetValue ( 50 )
2264226  5:29:51.934 PM  1  OC_GURU.exe    LeaveCriticalSection ( 0x00499dc8 )
2264227  5:29:51.934 PM  1  OC_GURU.exe    EnterCriticalSection ( 0x00499dc8 )
2264228  5:29:51.934 PM  1  OC_GURU.exe    TlsGetValue ( 50 )
2264229  5:29:51.934 PM  1  OC_GURU.exe    LeaveCriticalSection ( 0x00499dc8 )
2264230  5:29:51.934 PM  1  OC_GURU.exe    EnterCriticalSection ( 0x00499e90 )
2264231  5:29:51.934 PM  1  OC_GURU.exe    LeaveCriticalSection ( 0x00499e90 )
```

Figure 8 - Examples of the constant calls OC Guru makes in the background that can be used to differentiate from LED changing calls.

So we can determine what initiates the LED change simply by determining where the first different function call occurs that differs from this, and it turns out to be a call to a function named *GvWriteI2C*. This is our first major breakthrough. Thanks to Gigabyte giving their API functions reasonably descriptive names, it leads us to some conclusions that will be useful for communicating with the hardware ourselves:

1. LED setting uses I^2C. Linux has native support for I^2C, so:
2. We can detect and dump information about I^2C devices in Linux
3. We can write our own values to I^2C addresses of our card in Linux

Looking through the rest of our function calls, we can see that actually GvSetLogoLight was a red herring; it doesn't appear anywhere in here! The only function from our Gv DLLs called is GvWriteI2C, so I set a breakpoint on GvWriteI2C and looked at what happens:



GvWriteI2C | GvDisplay.dll

Module: OC_GURU.exe
Process: Graphics Card Smart Tuner. (OC_GURU.exe)
Process ID: 6888 Kill
Thread ID: 16020 Kill

#	Type	Name	Value
1	Stack		{ uintp = 0x00000000, intp = 0x00000000, psz = NULL ...}
2	Stack		{ uintp = 0x0018e124, intp = 0x0018e124, psz = 0x0018e124 ...}
3	Stack		{ uintp = 0x00000000, intp = 0x00000000, psz = NULL ...}
4	Stack		{ uintp = 0x00430294, intp = 0x00430294, psz = 0x00430294 ...}

Exception: None
Last Error Code: 0

Breakpoint
[✓ Before] After On Error On Exception

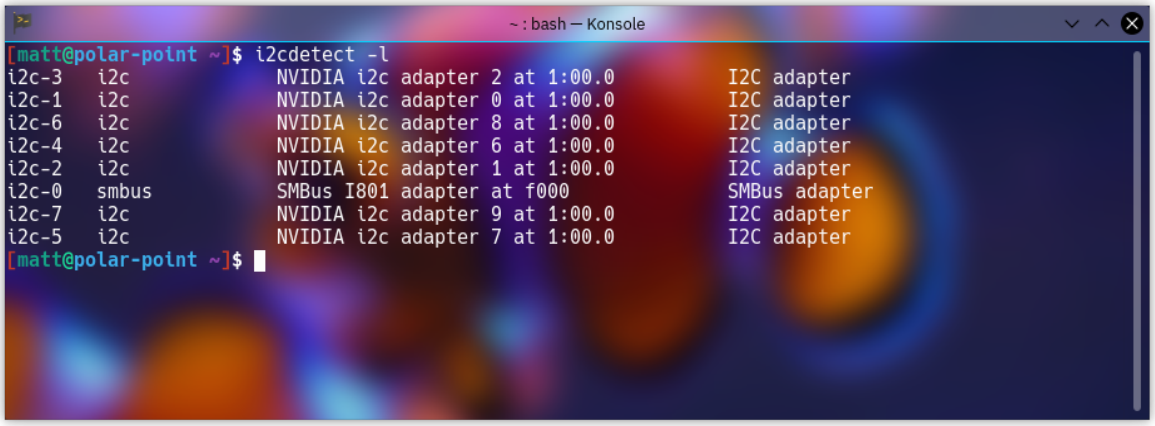
Figure 9 - API Monitor breakpoint information dialogue - *GvWriteI2C* call #1.

- GvWriteI2C is called 6 times when setting a static colour. We can conclude this because we get 6 breakpoints before OC Guru II continues normally.
- The LED change happens after the 2^{nd} call. We can see this as our LED on our GPU changes after we hit "Continue" on the breakpoint window after the 2nd breakpoint is triggered.

We clearly need to investigate this function more. We can now conclude that the GvWriteI2C function from the GvDisplay.dll library is the thing that actually changes our LED colours. We've also found out that the card communicates with the software over I^2C. In addition to this, the nvapi.dll is also called so NVAPI may be involved in setting LEDs as well.

8 Investigating the I^{2}C buses of the G1

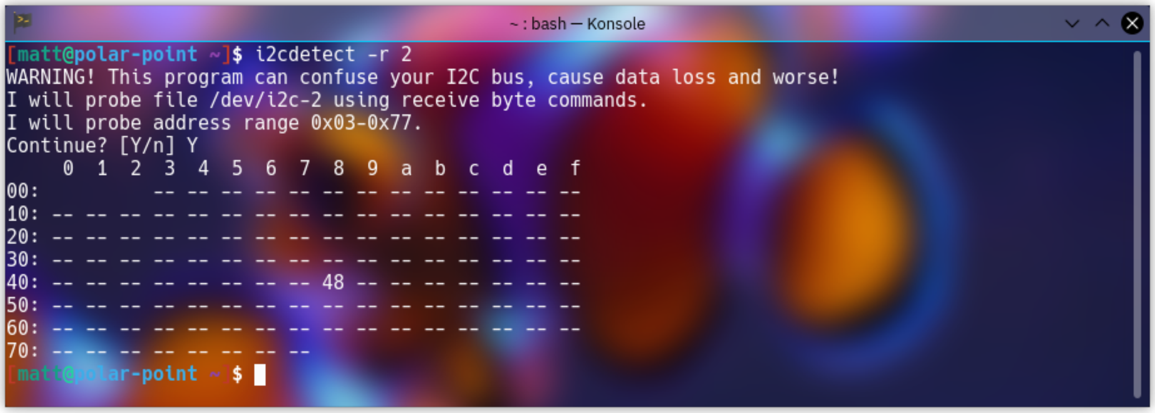
Back in Linux, a certain set of tools are required to interface with I^{2}C devices. There is only a single package that needs installing, on Arch and Ubuntu (and possibly others) it is simply 'i2c-tools'. Once this is installed, we can run i2cdetect to view our current available I^{2}C devices. After loading the i2c driver with sudo modprobe i2c-dev, we can query the current i2c devices:



```
[matt@polar-point ~]$ i2cdetect -l
i2c-3   i2c             NVIDIA i2c adapter 2 at 1:00.0          I2C adapter
i2c-1   i2c             NVIDIA i2c adapter 0 at 1:00.0          I2C adapter
i2c-6   i2c             NVIDIA i2c adapter 8 at 1:00.0          I2C adapter
i2c-4   i2c             NVIDIA i2c adapter 6 at 1:00.0          I2C adapter
i2c-2   i2c             NVIDIA i2c adapter 1 at 1:00.0          I2C adapter
i2c-0   smbus           SMBus I801 adapter at f000              SMBus adapter
i2c-7   i2c             NVIDIA i2c adapter 9 at 1:00.0          I2C adapter
i2c-5   i2c             NVIDIA i2c adapter 7 at 1:00.0          I2C adapter
[matt@polar-point ~]$
```

Figure 10 - List of I^{2}C adapters exposed to Linux

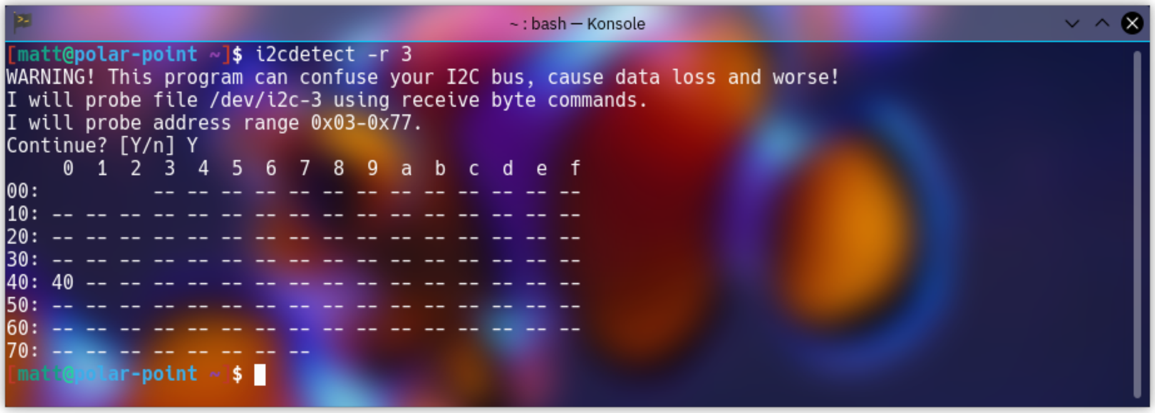
And we get a bunch of hits for NVIDIA I^{2}C adapters. Great! Now we can investigate each of these I^{2}C adapters further using the -r option. "NVIDIA i2c adapter 0" gives an empty table upon probing. Doesn't seem like this is the one I'm looking for. i2c-2 reads as "NVIDIA i2c adapter 1". Turns out I get one hit for an address at 0x48:



```
[matt@polar-point ~]$ i2cdetect -r 2
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2 using receive byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] Y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
[matt@polar-point ~]$
```

Figure 11 - Addresses found on the scan of I^{2}C bus 2: 0x48.

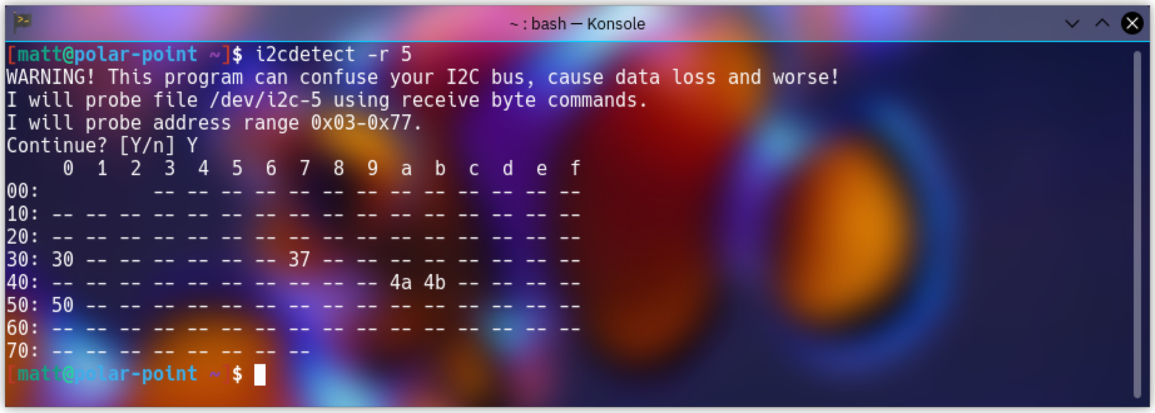
Having a look at "NVIDIA i2c adapter 2" on i2c-3 we get a hit for 0x40:



```
[matt@polar-point ~]$ i2cdetect -r 3
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-3 using receive byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] Y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
[matt@polar-point ~]$
```

Figure 12 - Addresses found on the scan of I^{2}C bus 3: 0x40.

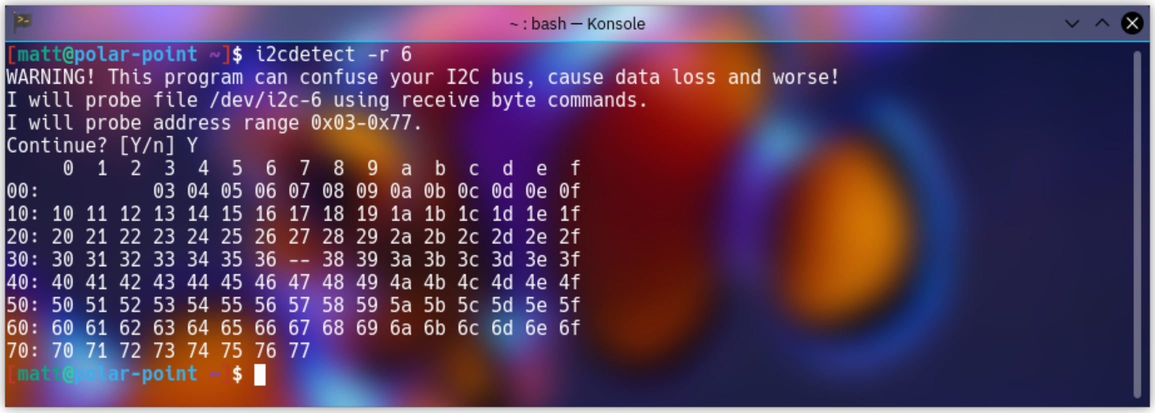
We get another empty table on i2c-4, “NVIDIA i2c adapter 6”. On i2c-5 “NVIDIA i2c adapter 7”:



```
[matt@polar-point ~]$ i2cdetect -r 5
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-5 using receive byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] Y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: 30 -- -- -- -- -- -- 37 -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- 4a 4b -- -- -- --
50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
[matt@polar-point ~]$
```

Figure 13 - Addresses found on the scan of I^{2}C bus 5.

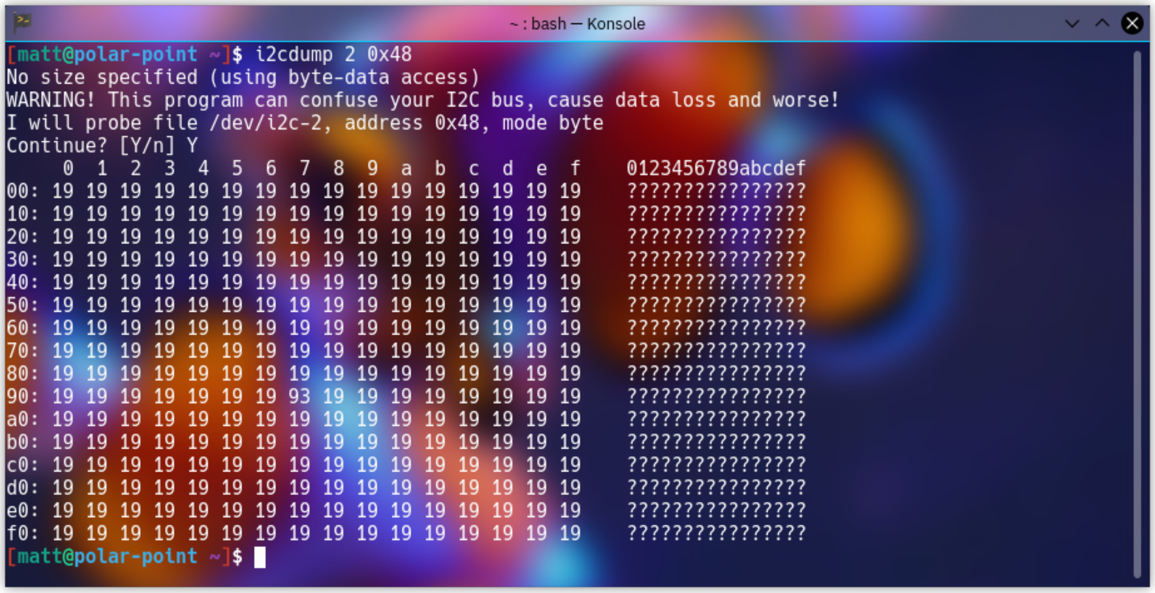
i2c-6, “NVIDIA i2c adapter 8”:



```
[matt@polar-point ~]$ i2cdetect -r 6
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-6 using receive byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] Y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10: 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
20: 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
30: 30 31 32 33 34 35 36 -- 38 39 3a 3b 3c 3d 3e 3f
40: 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f
50: 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
60: 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
70: 70 71 72 73 74 75 76 77
[matt@polar-point ~]$
```

Figure 14 - Addresses found on the scan of I^{2}C bus 6.

And finally “NVIDIA i2c adapter 9” on i2c-7 reads another empty table. The i2c-tools package also includes a utility called i2cdump. My first move when investigating these addresses I’ve found was to try and run i2cdump on them to see if there's any useful information returned. I decided to dump them in ascending order, so firstly I looked at "NVIDIA i2c adapter 1", 0x48 on i2c-2. Running the dump command, I get:

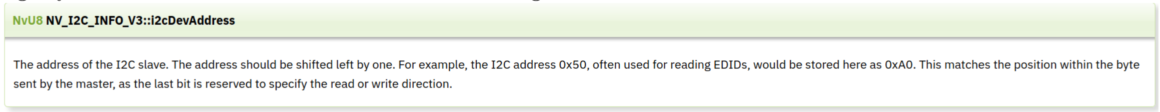


```
[matt@polar-point ~]$ i2cdump 2 0x48
No size specified (using byte-data access)
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2, address 0x48, mode byte
Continue? [Y/n] Y
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
10: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
20: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
30: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
40: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
50: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
60: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
70: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
80: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
90: 19 19 19 19 19 19 19 93 19 19 19 19 19 19 19 19    ????????????????
a0: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
b0: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
c0: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
d0: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
e0: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
f0: 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19    ????????????????
[matt@polar-point ~]$
```

Figure 15 - Dump of I^{2}C adapter 2, address 0x48. Not really any useful or telling information.

And Bingo! When I dumped this data, the LEDs on the graphics card turned off. To me, this indicates I may have found the address the LED controller uses to accept communications right off the bat, as I was able to interact with it in a roundabout sort of way. Turning my PC fully off and back on so that the LEDs reset back to default, and dumping the other single address I found, 0x40 on i2c-3 has no affect on the LEDs. I will proceed assuming 0x48 on i2c-2 is the correct address for the LED controller.

According to NVIDIA's official documentation for NVAPI, the address of the I^{2}C slave will be shifted bitwise by one to the left. Shifting the Hexadecimal value 48 by 1 to the left is 90, so theoretically NVAPI or Gigabyte's GvWriteI2C function will be writing to the address 90.



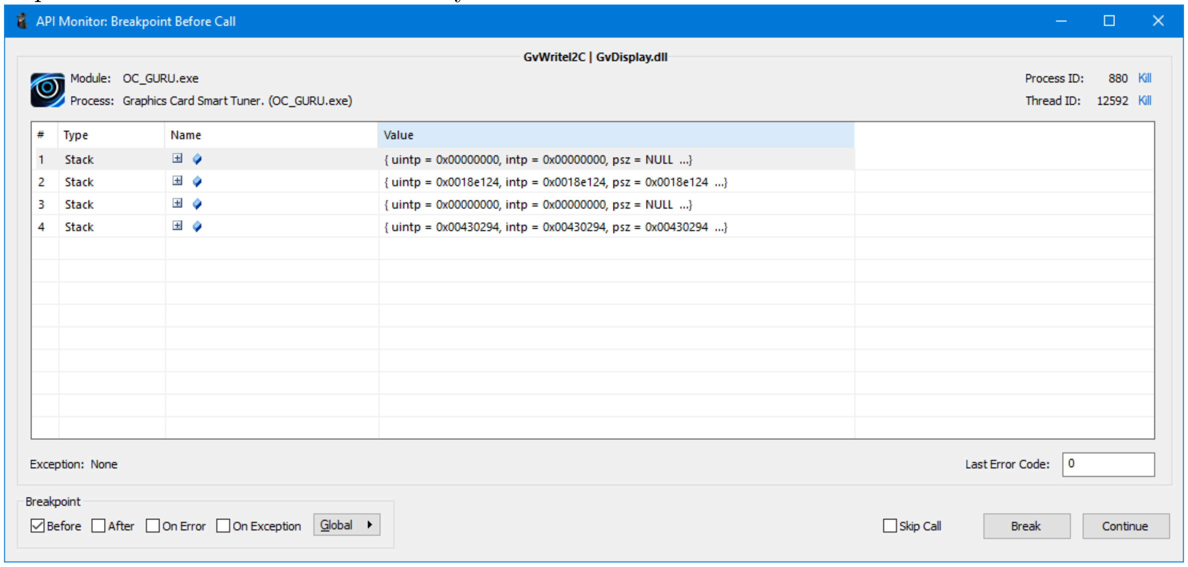
NvU8 **NV_I2C_INFO_V3::i2cDevAddress**

The address of the I2C slave. The address should be shifted left by one. For example, the I2C address 0x50, often used for reading EDIDs, would be stored here as 0xA0. This matches the position within the byte sent by the master, as the last bit is reserved to specify the read or write direction.

Figure 16 - NVIDIA's documentation on the address of the I^{2}C slaves their programs can interface with.

9 Investigating GvWriteI2C

As previously established, GvWriteI2C is the actual function that changes the LEDs. And, it seems like our LED address is 90, so using this we can investigate the function further and attempt to deconstruct its references to memory. Change a colour, in this first example I will be setting the Cyan colour. On the first breakpoint we'll have a look at its memory references.



GvWriteI2C | GvDisplay.dll

Module: OC_GURU.exe
Process: Graphics Card Smart Tuner. (OC_GURU.exe)
Process ID: 880 Kill
Thread ID: 12592 Kill

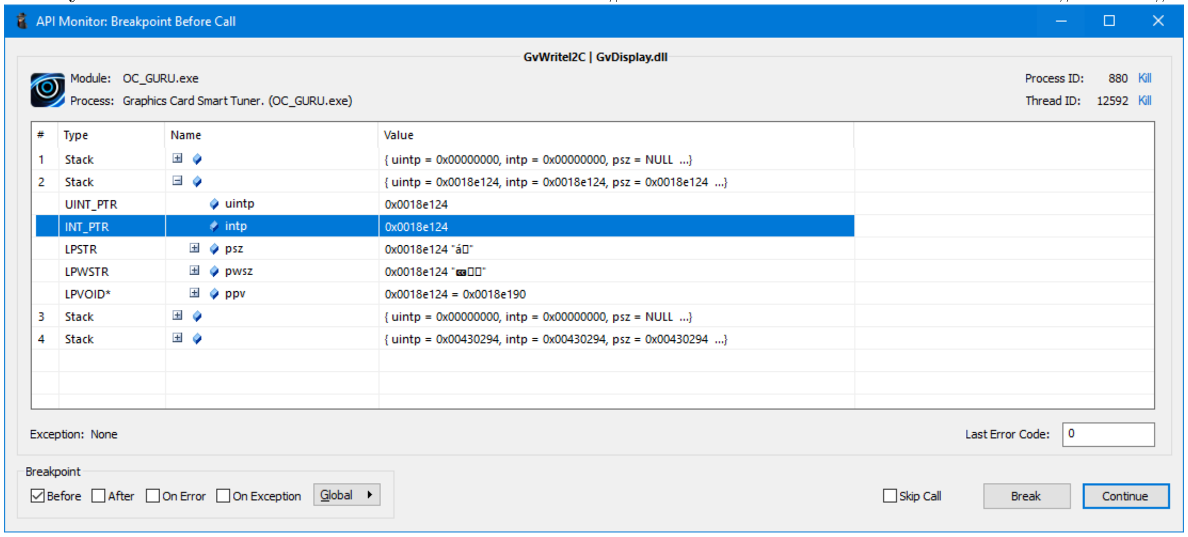
#	Type	Name	Value
1	Stack	⊞	{ uintp = 0x00000000, intp = 0x00000000, psz = NULL ...}
2	Stack	⊞	{ uintp = 0x0018e124, intp = 0x0018e124, psz = 0x0018e124 ...}
3	Stack	⊞	{ uintp = 0x00000000, intp = 0x00000000, psz = NULL ...}
4	Stack	⊞	{ uintp = 0x00430294, intp = 0x00430294, psz = 0x00430294 ...}

Exception: None
Last Error Code: 0

Breakpoint
[✓ Before] ☐ After ☐ On Error ☐ On Exception

Figure 17 - First breakpoint when changing colour to Cyan.

The first bit of info, #1 has no information in it at all. Expanding the menu shows a reference to 0x00000000 and every other value is NULL. This is the same with #3. So this means we want to look at #2 and #4.



GvWriteI2C | GvDisplay.dll

Module: OC_GURU.exe
Process: Graphics Card Smart Tuner. (OC_GURU.exe)
Process ID: 880 Kill
Thread ID: 12592 Kill

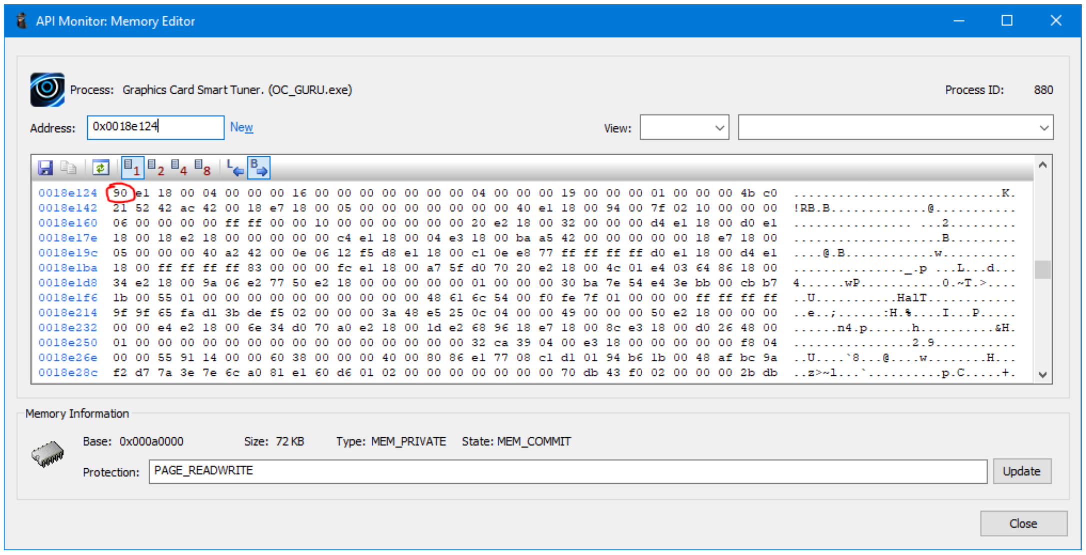
#	Type	Name	Value
1	Stack	⊞	{ uintp = 0x00000000, intp = 0x00000000, psz = NULL ...}
2	Stack	⊟	{ uintp = 0x0018e124, intp = 0x0018e124, psz = 0x0018e124 ...}
	UINT_PTR	uintp	0x0018e124
	INT_PTR	intp	0x0018e124
	LPSTR	⊞ psz	0x0018e124 "á□"
	LPWSTR	⊞ pwsz	0x0018e124 "[illegible]□□"
	LPVOID*	⊞ ppv	0x0018e124 = 0x0018e190
3	Stack	⊞	{ uintp = 0x00000000, intp = 0x00000000, psz = NULL ...}
4	Stack	⊞	{ uintp = 0x00430294, intp = 0x00430294, psz = 0x00430294 ...}

Exception: None
Last Error Code: 0

Breakpoint
[✓ Before] ☐ After ☐ On Error ☐ On Exception

Figure 18 - Expanded menu of the Cyan colour change breakpoint, displaying memory references.

Investigating #2, and we see a reference to 0x0018e124 and 0x0018e190, so right clicking on it, opening the memory editor and navigating to these addresses reveals:


API Monitor: Memory Editor
Process: Graphics Card Smart Tuner. (OC_GURU.exe)
Process ID: 880
Address: 0x0018e124
New
View:
Memory Information
Update
Close

```
0018e124  90 e1 18 00 04 00 00 00 16 00 00 00 00 00 00 00 04 00 00 00 19 00 00 00 01 00 00 00 4b c0  ............................K.
0018e142  21 52 42 ac 42 00 18 e7 18 00 05 00 00 00 00 00 00 00 40 e1 18 00 94 00 7f 02 10 00 00 00  !RB.B.............@...........
0018e160  06 00 00 00 00 ff ff 00 00 10 00 00 00 00 00 00 20 e2 18 00 32 00 00 00 d4 e1 18 00 d0 e1  ................ ...2.........
0018e17e  18 00 18 e2 18 00 00 00 00 00 c4 e1 18 00 04 e3 18 00 ba a5 42 00 00 00 00 00 18 e7 18 00  ....................B.........
0018e19c  05 00 00 00 40 a2 42 00 0e 06 12 f5 d8 e1 18 00 c1 0e e8 77 ff ff ff ff d0 e1 18 00 d4 e1  ....@.B............w..........
0018e1ba  18 00 ff ff ff ff 83 00 00 00 fc e1 18 00 a7 5f d0 70 20 e2 18 00 4c 01 e4 03 64 86 18 00  ..............._.p ...L...d...
0018e1d8  34 e2 18 00 9a 06 e2 77 50 e2 18 00 00 00 00 00 01 00 00 00 30 ba 7e 54 e4 3e bb 00 cb b7  4......wP...........0.~T.>....
0018e1f6  1b 00 55 01 00 00 00 00 00 00 00 00 00 00 48 61 6c 54 00 f0 fe 7f 01 00 00 00 ff ff ff ff  ..U...........HalT............
0018e214  9f 9f 65 fa d1 3b de f5 02 00 00 00 3a 48 e5 25 0c 04 00 00 49 00 00 00 50 e2 18 00 00 00  ..e..;......:H.%....I...P.....
0018e232  00 00 e4 e2 18 00 6e 34 d0 70 a0 e2 18 00 1d e2 68 96 18 e7 18 00 8c e3 18 00 d0 26 48 00  ......n4.p......h..........&H.
0018e250  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 32 ca 39 04 00 e3 18 00 00 00 00 00 f8 04  ................2.9...........
0018e26e  00 00 55 91 14 00 00 60 38 00 00 00 40 00 80 86 e1 77 08 c1 d1 01 94 b6 1b 00 48 af bc 9a  ..U....`8...@....w........H...
0018e28c  f2 d7 7a 3e 7e 6c a0 81 e1 60 d6 01 02 00 00 00 00 00 00 00 00 00 70 db 43 f0 02 00 00 00 2b db  ..z>~l...`............p.C...+.
```

Base: 0x000a0000 Size: 72 KB Type: MEM_PRIVATE State: MEM_COMMIT

Protection: PAGE_READWRITE

Figure 19 - A reference to 90 (the address of our I^{2}C slave) at the memory referenced by GvWriteI2C!

A reference to 90 at 0x0018e124. If you recall, this is actually the value of presumably our LED controller (0x48) shifted to the left by 1 bit. So there we have it! It's all but confirmed that our assumption was correct.

Now that we've found the beginning of the transmission, let's find out what data it sends by setting a few more colours, and looking at what the differences are on that line. But first, I'd like to have a look at what the line looks like on the other multiple calls to GvWriteI2C when the colour is shifting. It reveals that the position of the value 16 in that previous screenshot (address 0x0018e12c, on the first line) cycles between 16, 81 and 83 and does that twice for a total of 6 calls.

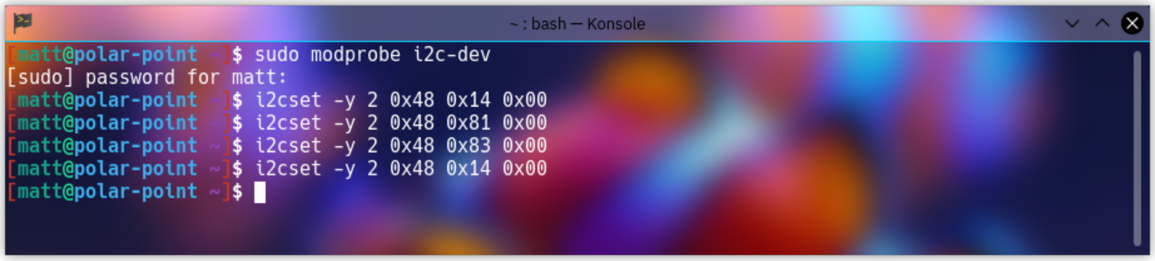
So I tried setting other colour settings, and it turns out that value at 0x0018e12c is the one that changes between colour settings. If the LED mode is set to "On", changing the LEDs gives these values at that position:

Colour	ID at 0x0018e12c
Blue	14
Red	11
Green	12
Yellow	13
Magenta	15
Cyan	16
White	17

These also cycle between ID, 81, 83 for the total of 6 calls but the IDs are the only values that change between colour settings. So what now? We have our colour IDs that are sent to 0x48, but how do we do something with that? Turns out that included with i2c-tools over on Linux is a utility known as i2cset. We can play around with this and try to change the LED colours.

10 Setting the LED Colours in Linux

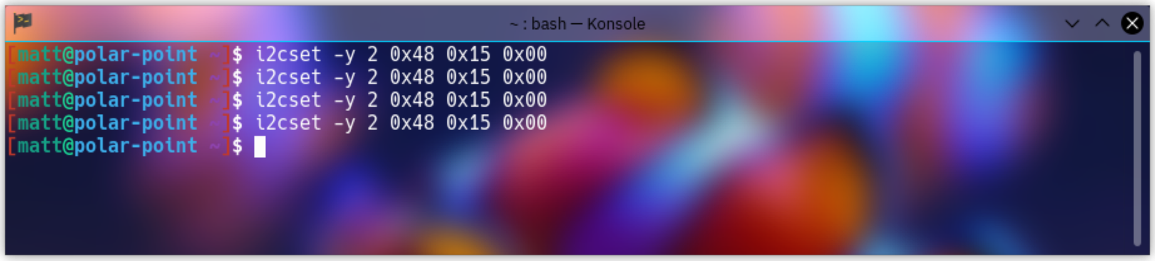
Let's review what we know so far. We know that in Linux, the LED colour controller is on i2c bus 2 at address 0x48. We also know what all the hexadecimal values for the colour modes are, and how GvWriteI2C writes to the I^{2}C 6 times in a cyclic nature with ID, 81 and 83. Logically, we might try and emulate this using the Linux utility i2cset. Let's try and get it to change to blue:



```
[matt@polar-point ~]$ sudo modprobe i2c-dev
[sudo] password for matt:
[matt@polar-point ~]$ i2cset -y 2 0x48 0x14 0x00
[matt@polar-point ~]$ i2cset -y 2 0x48 0x81 0x00
[matt@polar-point ~]$ i2cset -y 2 0x48 0x83 0x00
[matt@polar-point ~]$ i2cset -y 2 0x48 0x14 0x00
[matt@polar-point ~]$ 
```

Figure 20 - Successfully setting the Windforce LED to the colour Blue in Linux with i2cset.

And it worked! Using this command, I didn't even have to write data to the I^{2}C device as many times as OC Guru II did. That being said, I am sending less data, there was some info in the stuff OC Guru was sending I haven't analysed fully yet. Playing around with it a bit more, I found out that I don't even have to write the 81 and 83. If I write the colour ID 4 times, the LED will change. For example, to set the LED to magenta:



```
[matt@polar-point ~]$ i2cset -y 2 0x48 0x15 0x00
[matt@polar-point ~]$ i2cset -y 2 0x48 0x15 0x00
[matt@polar-point ~]$ i2cset -y 2 0x48 0x15 0x00
[matt@polar-point ~]$ i2cset -y 2 0x48 0x15 0x00
[matt@polar-point ~]$ 
```

Figure 20 - Successfully setting the Windforce LED to the colour Magenta in Linux with i2cset, only using the colour ID.

So simple! And this can be achieved without actually doing any coding. So naturally, the next thing to do is slap it into a shell script so that LEDs can be changed with a single command, and the scripts available on this github repository are the final products of this reverse engineering effort so far. There's more to this reverse engineering effort which will be provided in additional PDF files under the docs/ folder on the github repository: g1980ti-led

11 LED Controller Operation Theories

I'm adding this in as part of the revision on 21/03/24 to provide an educated guess as to how the LED controller actually works now that the process is done. Unfortunately my card died (of unrelated causes I assure you) so I can't go back and test these.

I see two main possibilities:

- Every colour is assigned a register and if data is written there the colour is selected
- There is only 1 data receive register and writing the colour value there will change it

I'm leaning towards the second given the dump of the addresses for the LED controller (figure 15) gave the same value for all tried addresses. It is potentially the case that the 6 calls to GvWriteI2C when the colour is set write a certain value to that single register, and the meaning of the data written there depends on the order which would explain the other 0x81 and 0x83 values. I guess we will never know :)