Applying Crowbar Voltage Glitches to nRF52 and CC253x/4x Microcontrollers

Jonathan Rudman

April 21, 2022

University of Birmingham

Student ID: 1907314
Supervisor: Prof. David Oswald
Programme: 610C - BSc Com Sc w DTP FT DA (PwC)
Word Count: 9464

Abstract

Embedded systems utilising microcontrollers have become ubiquitous; they are found in everything from watches to washing machines, and are commonly connected to the Internet. Manufacturers of devices containing microcontrollers typically enable some form of code readout protection (CRP) to protect their intellectual property, hiding their firmware and disabling a level of debugging access within these microcontrollers. Voltage glitching is an often successful fault injection method used to bypass this readout protection and allow an adversary, security researcher or consumer to extract the firmware from the microcontroller's internal memory.

This report explores how "crowbar" voltage glitches—momentarily shorting the CPU core of a microcontroller to ground—can be used to bypass readout protection on members of two families of microcontrollers: Nordic Semiconductor's nRF52 series and Texas Instruments' CC253x/4x series. A successful glitch attempt is replicated on the nRF52832, faulting the CRP data and allowing the extraction of firmware. Unsuccessful glitch attempts are made on the CC2541—for which there are no crowbar glitches leading to a CRP bypass published at the time of writing—with a repeatable glitch outcome of full flash memory erasure documented. A consistent methodology for both glitching attempts is outlined, starting with identifying a suitable glitch width and moving on to describe identifying potential critical sections in power traces. Two different glitch generators are used: the Pico Debug'n'Dump and an FPGA-based board using GIAnT, the Generic Implementation Analysis Toolkit. Finally, improvements to the ease of use of voltage glitching hardware are proposed; these intend to apply some direction to the problem of making voltage glitching kits more consumer-friendly, increasing the ease with which defunct consumer devices may be repurposed.

Contents

1 Introduction

Embedded systems based around microcontroller units (MCUs) are ubiquitous due to their low cost and versatility when compared to purpose-built circuits. Microcontroller firmware is often protected from being read by the consumer with code readout protection (CRP), usually in the form of disabling debugging access to the microcontroller. This means that independent security researchers cannot easily extract the firmware to assess it for security vulnerabilities and consumers of many Internet of Things (IoT) devices are left with e-waste: devices which cannot be repurposed if the manufacturer goes bankrupt and can no longer provide the original service. This service, in the current age of IoT, is often a webserver which augments or authorises the capabilities of the embedded system, and the address to this server would need to be changed if the webserver goes dark—something which cannot be done if the original firmware cannot be read and modified.

There are many potential remedies for this problem which would allow consumers and researchers to access the MCU firmware, but this report focuses on a fault injection method called voltage glitching. Fault injection is an active implementation attack whereby a fault in the operation of a computer is caused by putting it under unusual operating conditions. This fault may then lead to an observable failure such as skipping the CRP check of a microcontroller under stress, enabling its debugging capabilities when ordinarily they would be disabled. Voltage glitching is a method of fault injection where a fault is caused by changing the operating voltage of a device to be outside of its recommended range.

Voltage glitching was chosen as the area of research for this report, as opposed to invasive implementation attacks and legislative solutions—for example, forcing manufacturers to publish source code or allow debugging access to consumers—for a few reasons. Legislative solutions are slow, expensive and require many people to back them in order to gain traction and, even then, can fail easily. Right to repair, for example, is a movement that's gained more traction over the last year, but it is not a new idea. Fault injection methods on the other hand can be done quickly and relatively cheaply by one person; they can be a very reliable way to change the behaviour of a microcontroller. Voltage glitching is an inexpensive and quickly repeatable active implementation attack, especially when compared to invasive implementation attacks such as using laser or UV-C light to flip bits.

There are a variety of voltage glitching techniques including overvolting and undervolting, but this report researches the feasibility of using a "crowbar" voltage glitch [1] to bypass the CRP of two MCUs: Nordic Semiconductor's nRF52832 and Texas Instruments' CC2541. A crowbar voltage glitch involves momentarily shorting the CPU voltage to ground in order force the CPU to miscalculate or skip an instruction. In order to learn more about voltage glitching, the first chip for which a crowbar glitch to bypass CRP will be attempted is the nRF52832, due to the existence of documentation relating to executing this type of glitch [2], [3]. The second experiment will be to apply the same techniques learned during work on the nRF52832 to the CC2541, to see whether there is a similar vulnerability present on a microcontroller with a different CPU core and from a different manufacturer. Any methods which bypass CRP are assumed to also be applicable to the rest of the microcontroller family of each target: nRF52 and CC253x/4x. This report also outlines ways in which voltage glitching techniques could be made more accessible to the consumer who is trying to repurpose defunct hardware which has CRP enabled.

1.1 Related work

The nRF52832, used in Apple AirTags, is susceptible to a crowbar glitch to skip its CRP check, demonstrated by Roth [3]. Roth also cites earlier work from a blog, LimitedResults [2], which focuses on a similar system-on-a-chip (SoC), the nRF52840, and presents information about the nRF52 family in the context of voltage

glitching and CRP. Replicating Roth's and LimitedResults' work is the first experiment of this project and was used as a framework for learning about many aspects of voltage glitching for the first time before moving onto a novel experiment: applying a similar glitch to the Texas Instruments CC2541.

Voltage glitching can also be used in conjunction with binary analysis—a so-called grey-box approach—on embedded bootloaders [4]. Although the nRF52832—this report's first voltage glitch target—has a CRP check before any optional bootloader, Van Den Herrewegen et al. outline voltage glitches on three different MCUs, presenting useful information and methods for someone learning about voltage glitching. In the aforementioned paper the GIAnT [5] is also used for the attacks; GIAnT is also used throughout this report as the primary glitching system for glitching the nRF52832 and CC2541, although the Pico Debug'n'Dump [6] is used as well against the nRF52832 for comparison.

O'Flynn details the application of crowbar circuits—which are used to short-circuit a component to ground—in voltage glitching methods [1]. Similar to O'Flynn's usage of the n-channel MOSFET on the ChipWhisperer, both the Pico Debug'n'Dump and GIAnT boards utilise an n-channel MOSFET in their implementations of a crowbar circuit. The application of these crowbar voltage glitches is the main focus of this report.

The exploratory experiments on the CC2541 outlined later in this report (power analysis, glitching a test application to find suitable parameters, etc.) were inspired by a glitch campaign carried out by GitHub user debug-silicon against the SiLabs C8051F340 [7]. Much like the Texas Instruments CC2541, the SiLabs C8051F340 is also an 8051-based chip. Methods outlined by Wouters et al. [8], although applied to Texas Instruments microcontrollers based around ARM Cortex-M CPUs rather than 8051 CPUs, were also helpful in compiling this report. Reading both of these papers helped in understanding what often makes a voltage glitch targeting CRP checks possible—neither the similarities between CPUs or manufacturers, but rather the nature of the CRP check itself (which could be done outside the CPU) or the movement of CRP data.

1.2 Success criteria

The success criteria, or aims of this project, have changed since the project proposal, but settled on the following:

- Replicate the crowbar attack Roth carried out on an nRF52832 on a development board [3].
- Add to the knowledge of which chips are vulnerable to crowbar voltage glitch attacks.
  - Answer whether the same crowbar technique applied to the nRF52 series by both Roth [3] and LimitedResults [2] could be repeated and applied to another family of microcontrollers, the CC253x/4x series from Texas Instruments.
- Evaluate how it might be made easier and more reliable to carry out crowbar attacks.

Previously, this project's success criteria were focused on successfully glitching and then dumping the firmware of the nRF52832 and, later, the CC2541. In contrast, the bullet points above are more learning-oriented, focusing on investigation and the various activities that need to be executed to find a voltage glitch vulnerability, should one exist.

2 Replicating a crowbar attack on the nRF52832

The nRF52832 is a member of the nRF52 series from Nordic Semiconductor, a series based around the ARM Cortex-M4 processor and a 2.4GHz transceiver. This section outlines a successful attempt at replicating the crowbar voltage glitch attack on the nRF52832, initially demonstrated by Roth [3], to fault its CRP check

and allow debugging access in the serial bootloader. Roth showed how the nRF52832 could be glitched in situ on the AirTag PCB and built on the efforts of LimitedResults [2] who used the nRF52840, but the following experiment aims to glitch the nRF52832 on a development board. It is likely that, due to the similarities in features, processors and memory layouts, a voltage glitch which allows CRP bypass on one member of the nRF52 series is also applicable, with little to no adjustments, to any other member of the nRF52 series of microcontrollers. These similarities are why LimitedResults' method could also be applied to the nRF52832.

Without a background in electronics and fault injection, the reproduction of Roth's and LimitedResults' experiments proved to be a learning experience. Following this process would go on to provide knowledge and practise in preparation for investigating whether similar attacks would be possible on the CC2541.

2.1 Glitch method

The glitch method for nRF52 microcontrollers, similar to the method outlined by LimitedResults [2], is to:

1. Power on the microcontroller.
2. Wait until the memory controller starts transferring the user information configuration register values—including APPROTECT, the value of which determines whether the debug access port is protected—to the core. This time period is called the *glitch offset*.
3. Short the CPU power to ground for a time period, the *glitch width*.

2.1.1 Generic setup

Before using a glitch generator such as the Pico Debug'n'Dump or GIAnT, it is important to identify which pins will need to be accessed for the experiment. The following pins must be identified on the target:

- Power supply and ground connections because the microcontroller needs to be powered and operational while glitching.
- Debug access pins, for dumping firmware and sending other commands to the target.
- A decoupling circuit for the target.

A general purpose computer is used to configure the glitch parameters on the glitch generator and communicate with the nRF52832 over a debugging interface which, in the case of the nRF52832, an ST-LINK V2 connected over USB to the computer.

2.1.2 Benchmarking a successful glitch

Before running a glitch campaign against the nRF52832, the following console command is executed while the nRF52832 is connected to the general purpose computer over an ST-LINK V2. This command enables CRP by writing `0x01` to the user information configuration register (UICR) APPROTECT, which is used to enable access port protection, disabling debug access. Any value other than 0xFF will enable the protection.

```
1 openocd -f interface/stlink.cfg \
2   -f target/nrf52.cfg \
3   -c 'init;halt;flash fillw 0x10001208 0xFFFFFF00 0x01;exit'
```

A crowbar glitch applied to the CPU of the nRF52832 which causes a bypass of CRP is considered a successful glitch. This is checked using the following command:

```
1 openocd -f interface/stlink.cfg \
```

```
2   -f target/nrf52.cfg \
3   -c 'init;dump_image nrf52_dump.bin 0x0 0x1000;exit'
```

This command reads the flash memory and code RAM contents to `nrf52_dump.bin`, and unlike in Roth's example [9], the command was always found to be successful even if CRP was still enabled. There is still hope however, because `nrf52_dump.bin` will start with a certain sequence if the binary is empty, indicating that CRP is still enabled. This sequence, `b"\x00\x00\x00#"` when matched in Python, is checked against the first four bytes of `nrf52_dump.bin` and if it doesn't exist, CRP is disabled and real flash memory has been dumped.

2.1.3 Modifications

Five wires needed to be attached to the nRF52832 development board for easy connection to other jumper wires. These wires had to be trace back to the following pins on the nRF52832 chip itself:

- `VDD`: Power supply
- `VSS`: Ground
- `SWDCLK`: Serial wire debug clock input, for debugging
- `SWDIO`: Serial wire debug I/O, for debugging
- `DEC1`: 0.9V digital supply decoupling (CPU voltage regulator)

Identifying which pins these are requires referring to the nRF52832 Product Specification and using a multimeter to find the development board pins they are attached to. The header pins on the nRF52832 development board were too narrow and close together for any of the jumper wires available, so jumper wires were cut and soldered onto them for easier access.

Glitching the CPU voltage regulator decoupling circuit allows an attacker to specifically target the CPU, rather than the full system, which means that the CPU is more likely to feel the effect of the glitch. The decoupling capacitor was also removed from the `DEC1`, to prevent damping of the glitch and the voltage traces when reading the oscilloscope.

2.2 Using Pico Debug'n'Dump

The video demonstrating the nRF52832 glitch, uploaded by Roth [3], introduces the Pico Debug'n'Dump, a $25 commercially available device [6] Roth created to make crowbar voltage glitching cheaper and easier. Roth had also published a glitching application [9], which was flashed to the Debug'n'Dump by the general purpose computer. Below is a table of the wiring setup.

Debug'n'Dump	nRF52832	ST-LINK	Description
`Trig`	`VDD`		Supplies nRF52832 with 3.3V
`Glitch`	`DEC1`		For shorting the CPU
`GND`	`GND`		Grounds the nRF52832
	`SWDIO`	`SWDIO`	Serial wire debug I/O
	`SWCLK`	`SWCLK`	Serial wire clock input from ST-LINK

The first run didn't seem to have any effect on the nRF52832, but an oscilloscope wasn't available for troubleshooting at the time so there was no way of knowing whether the glitch was being applied, or when.

After acquiring an oscilloscope and connecting it to the `Glitch/DEC1` wire, it was observed that there was only a small pulse happening on `DEC1` at the glitch offset; the short circuit wasn't being applied. An issue was created on the airtag-glitcher GitHub repo documenting these findings [10] in the hope of communicating with Roth about this problem and fixing it.

It was established during communication on the GitHub issue that there was a manufacturing issue with some Debug'n'Dump units, where a pull-down resistor for the Glitch pin was rated at 10kΩ rather than 100Ω, preventing the short circuit from being applied. This resistor could be removed because it was redundant, due to the fact that the Pico pins are internally pulled down. To avoid wasting time while waiting for Roth to reply, the approach changed to use GIAnT, an FPGA-based glitching board and implementation analysis toolkit, to replicate the attack. Returning to the Debug'n'Dump—after succeeding with GIAnT—and removing its redundant resistor fixed the issue and not only was the short circuit of `DEC1` clearly visible on an oscilloscope, but CRP was bypassed within a few minutes.

2.3 Using GIAnT

Generic Implementation Analysis Toolkit (GIAnT) is an implementation analysis toolkit for use with FPGA-based boards [5], and was used as the glitch generator for attacking the nRF52832 after experimenting with the Pico Debug'n'Dump. Boards used by GIAnT are custom and cannot currently be found commercially; the one used in this experiment extends the ZTEX USB-FPGA Module 2.04, which costs €129 alone [11]. The GIAnT board used for this experiment didn't originally have a transistor soldered to it for the purposes of a crowbar circuit (but had the solder pads available), so a MOSFET had to be added to the trace leading to pin `T1`, along with the capability to use it in the FPGA bytestream.

Wiring the nRF52832 to the GIAnT is similar to the Pico Debug'n'Dump: `VDD` on the MCU is plugged into the DAC output on the GIAnT and `DEC1` is wired to `T1` on the GIAnT for the CPU crowbar.

After flashing the glitching firmware to the GIAnT development board, a Python library is used to interact with the board over USB and configure DAC voltage and the glitching parameters: the pin address for triggering the glitch, the glitch offset and the glitch width. The application [12] configures the GIAnT board to keep shorting the nRF52832's CPU to ground momentarily at an increasing offset and with multiple pulse widths. Slowly working through offsets and pulse widths increases the likelihood of finding glitch parameters which bypass CRP. The initial offset is configured to be immediately before the critical section (see section 2.3.2). Once there is a glitch success—defined by the extraction of valid firmware—the application on the general purpose computer stops running, and a dump of the nRF52832's firmware is left in the working directory.

2.3.1 Finding a suitable glitch width

Although the parameters will change during runtime, a suitable *minimum* glitch width should be found, which is the smallest glitch width at which a CPU instruction will be skipped or miscalculated. If the glitch width is too small, there is a chance that it will have no effect (or is too small for the glitch generator to output), but if it is too large the microcontroller might reset or the CPU might fail until the next reset. On each iteration, the glitch width is increased by 10ns to increase the likelihood of a successful glitch, should it require a longer glitch width. The difference between the smallest and largest glitch width is not large because it could extend runtime unnecessarily; the minimum glitch width should already reliably glitch a test application and so there's little need to iterate over many widths.

The process of finding a suitable glitch width can also be used to diagnose any problems with the glitching setup because in most cases where the CPU voltage is glitched, a fault should occur in the execution of the running application. O'Flynn uses a `for` loop containing a counter incrementation, nested in another `for` loop, to show how the counter incrementation is skipped when both loops are complete (or when the jump to the start of the loop is skipped) and the counter is printed [1]. Due to the added complexity of relying on printing to a buffer—requiring the use of another serial connection in this case—a simple GPIO pin toggling application was created, flashed and an oscilloscope was connected to that pin to observe the effect of a glitch [13]. Normal operation of this application is shown in figure 1.

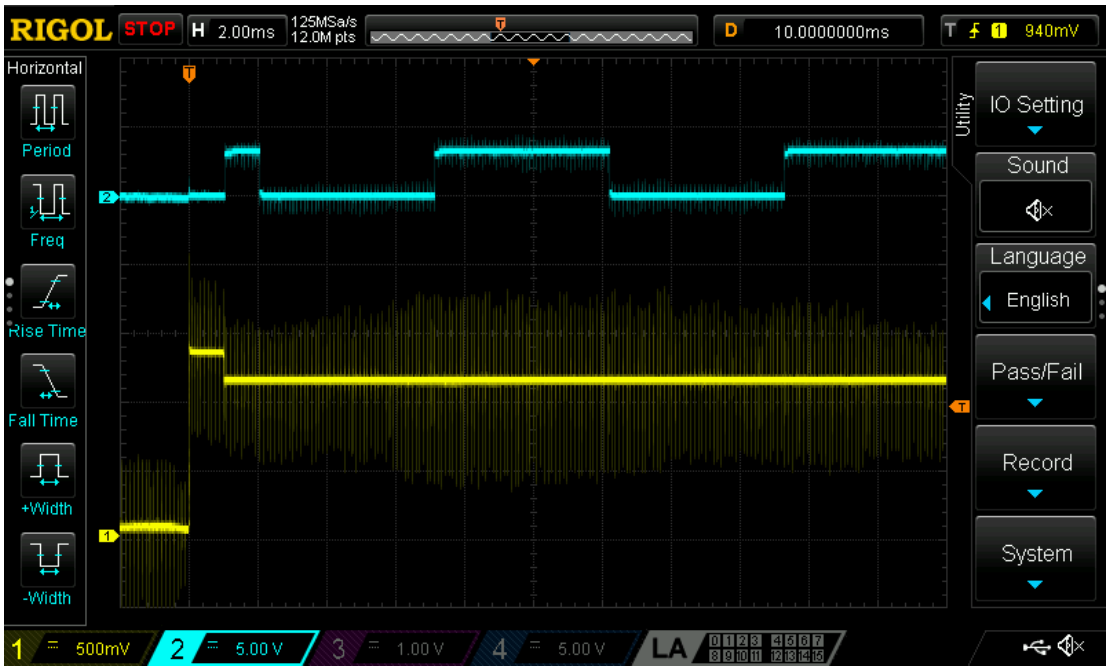

Figure 1: The pin toggling application functioning normally, with no glitch. Trace 1 (yellow) is the CPU core voltage and trace 2 (blue) is the pin being toggled by the application. Trigger T points to the moment the nRF52832 becomes powered.

After creating a script to interface with the GIAnT board to try a given glitch width over a range of offsets [14], the script was run against the nRF52832 running its pin toggling application. Manually increasing the value of `width` in the script from 10ns—the GIAnT board cannot produce glitches shorter than 10ns—showed that the minimum glitch width to successfully fault the CPU execution, at least in the case of application execution, is 50ns, as shown in figure 2. The result of the fault is that the application jumps out of the `while` loop altogether, even at different offsets. This could be due to skipping a `for` loop jump instruction, but it is likely that the fault occurs during the 5ms delay routine—which doesn't just use `NOP`s in the case of the nRF52832—and jumps out of it, especially because this effect is observed at many offsets. At widths as large as 140ns, the CPU would permanently fail until the next reset, as shown in figure 3.

Later investigation has found that a width of 50ns is typically too short to reliably fault the NVMC activity and therefore the CRP, and glitch widths as large as 180ns do not always cause the CPU to permanently fail until reset when targeting the critical section. This is why it is important to cycle through a number of glitch widths when approaching a critical section; some may have different outcomes when not applied to normal applications.

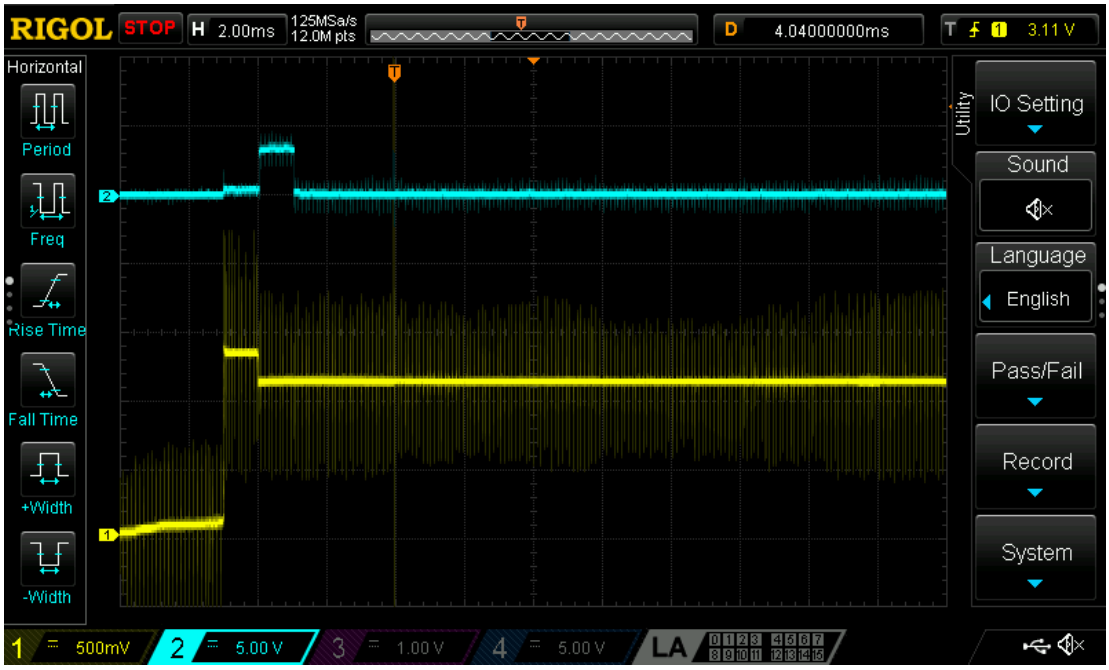

Figure 2: The pin toggling application being faulted, with a 50ns glitch applied to the CPU core at trigger T. Trace 1 (yellow) is the CPU core voltage and trace 2 (blue) is the pin being toggled by the application. Normal operation shown in figure 1.

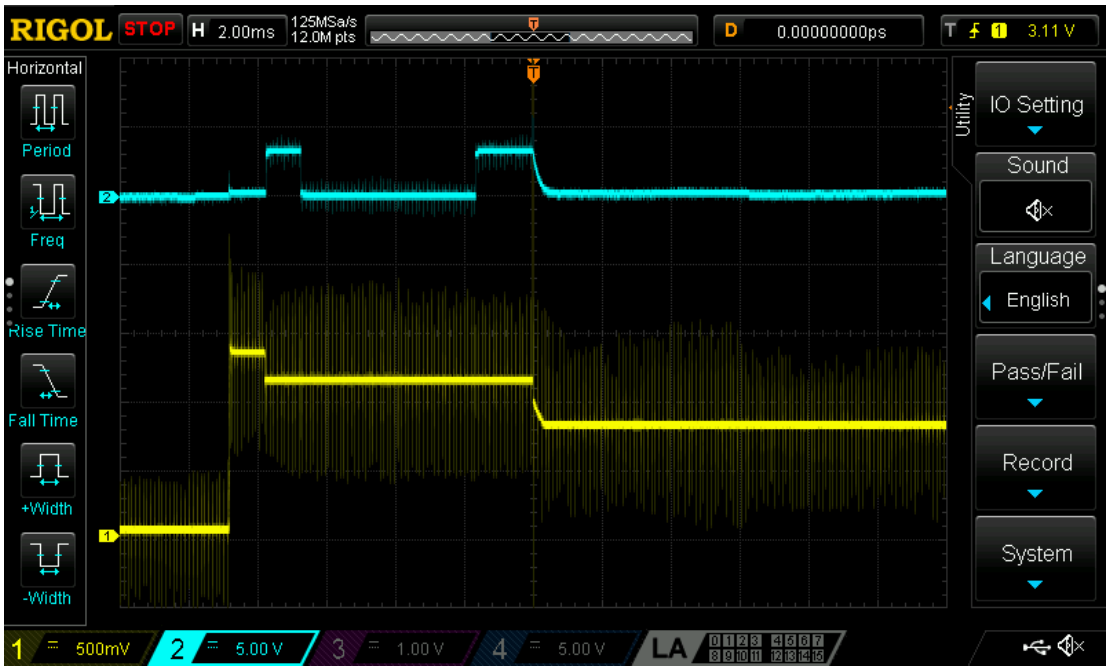

Figure 3: The pin toggling application being faulted, with a 140ns glitch applied to the CPU core at trigger T, causing the CPU to permanently fail until next reset. Trace 1 (yellow) is the CPU core voltage and trace 2 (blue) is the pin being toggled by the application. Normal operation shown in figure 1.

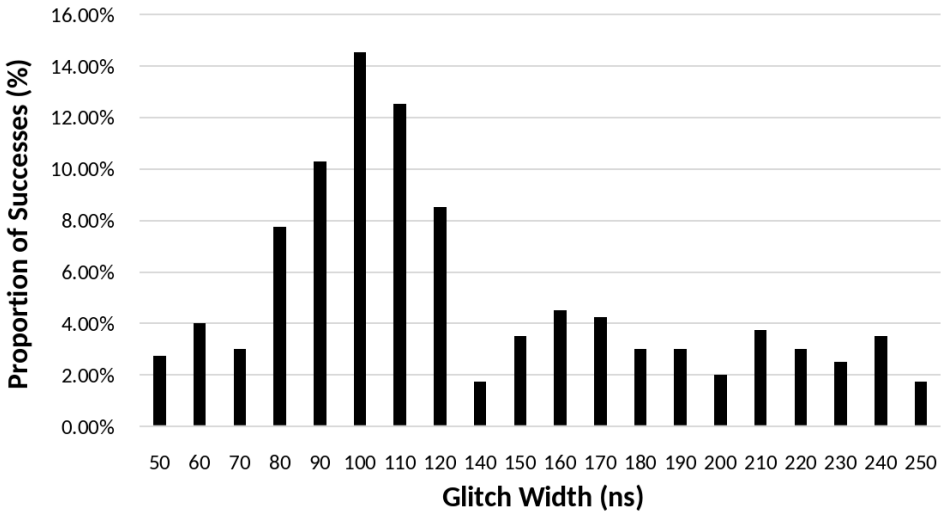

Figure 4: Glitch width vs. proportion of successes (out of 399 successes) for the nRF52832.

After applying a successful glitch (see section 2.4), widths for 399 successful glitches were recorded and the proportion of glitch successes for each glitch width is shown in figure 8. A successful glitch is a glitch on the NVMC activity outlined in section 2.3.2 which bypasses CRP and allows the firmware to be dumped. Figure 4 shows that the optimal glitch width is between 80ns and 120ns. It's important, for future crowbar glitch campaigns on the nRF52832 CPU core, that the glitch width range is reduced to this identified range, if the intent is to speed up causing a CRP bypass. Although there have been successes outside of the $80 - 120$ns range, successful glitches can be made even more likely in the $80 - 120$ns range by repeating with the same parameters, as this counteracts jitter (the critical section occurring earlier or later in the next power cycle).

2.3.2 Finding a suitable glitch offset

The glitch offset for the nRF52832 is the amount of time after supplying power to its development board until the glitch is applied. In practise, this is the time difference between powering on and just before the start of the critical section to compensate for jitter and ensure a glitching opportunity isn't missed.

Because this experiment was carried out to replicate work by Limited Results [2] and Roth [3] and apply a successful glitch, the focus was to find the same critical sections that they have highlighted in their work. For example, LimitedResults identified the section of non-volatile memory controller (NVMC) activity in figure 7, before the CPU core voltage drops and execution of the application starts [2]. This NVMC activity, according to LimitedResults, is likely to contain the transfer of user information configuration register (UICR) data, including the CRP value APPROTECT, to the CPU core. A similar section was identified for the nRF52832 and has been highlighted in figure 5, and is shown, zoomed-in, in figure 6. The noise present in figures 5 and 6 is due to the nRF52832 `VDD` being connected to the GIAnT DAC, rather than the ST-LINK V2.

The nRF52832's critical section lasts for about 2µs, although glitches are usually successful in the later part of the critical section. The offsets for 399 successful glitches were recorded and the proportion of glitch successes for each glitch offset is shown in figure 8. Looking at figure 8, the glitch offset can be said to start at 1029µs after supplying power to the nRF52832, and increase by 1µs until 1044µs.

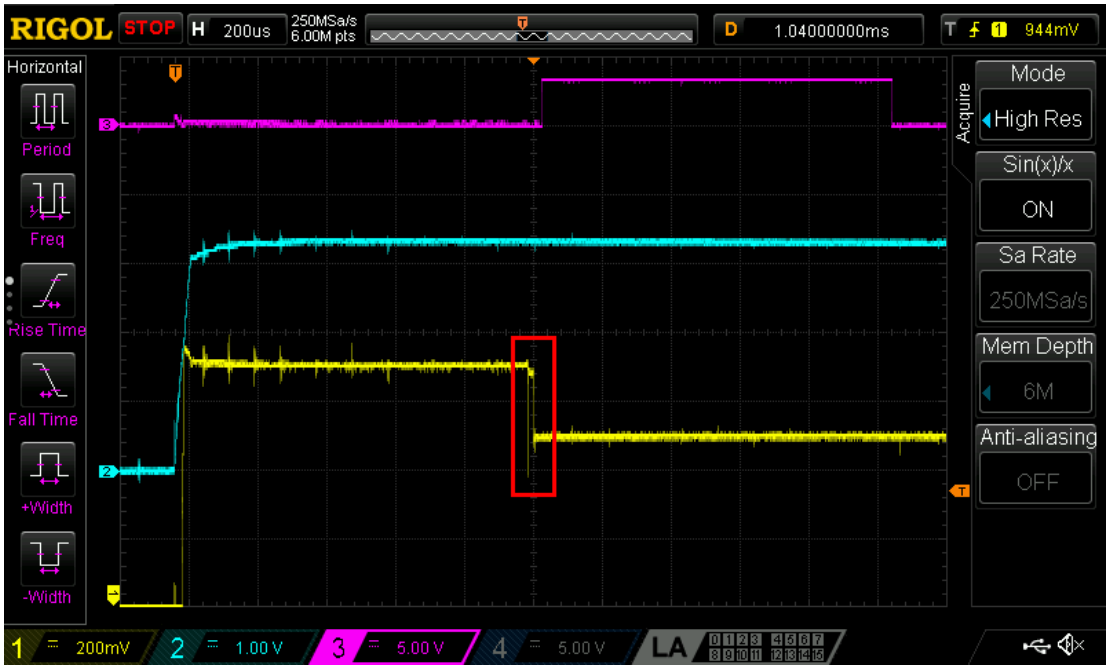

Figure 5: Red box highlighting critical section after nRF52832 power-on. Trace 1 (yellow) is the CPU core voltage, trace 2 (blue) is the voltage of `VDD`, trace 3 (purple) is the pin being toggled by the application on the nRF52832.

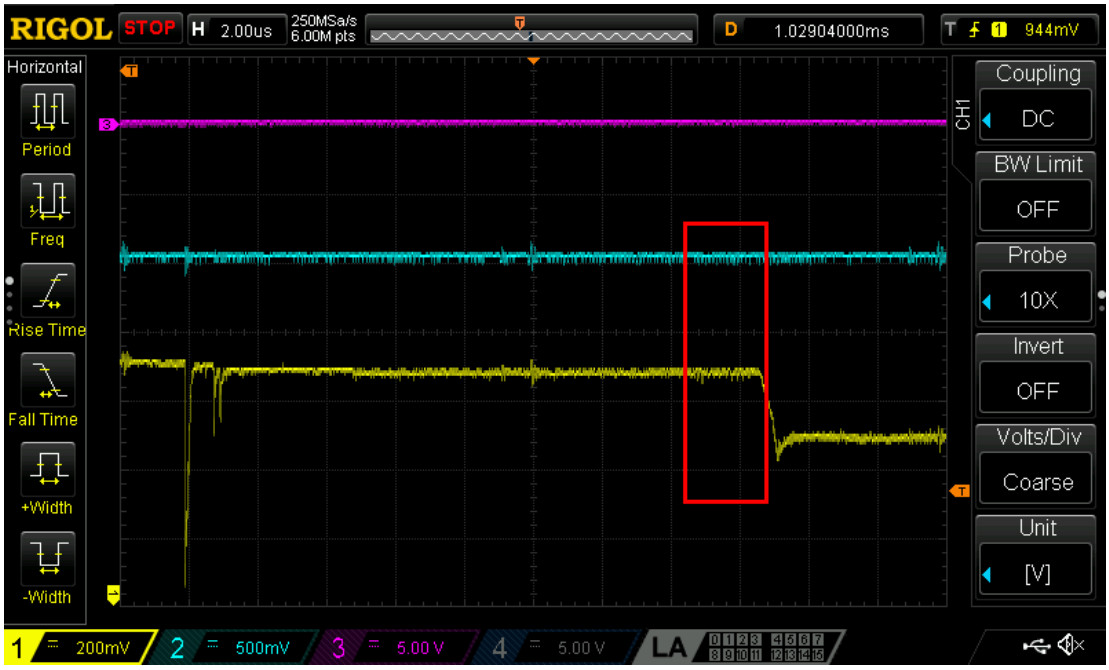

Figure 6: Zoomed-in version of the traces in figure 5. Critical section (NVMC activity) highlighted in red box.

Figure 7: Screenshot of the nRF52840 critical section, captured by LimitedResults [2]. Trace 2 (light blue) is the CPU core voltage at `DEC1` and trace 4 (dark blue) is the system power voltage at DEC4. Compare with figure 6.

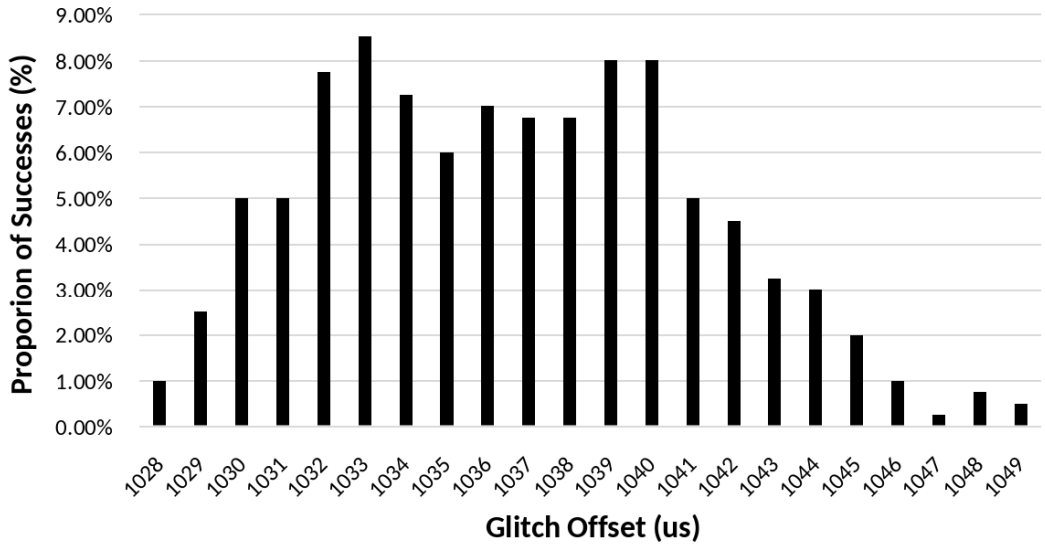

Figure 8: Glitch offset vs. proportion of successes (out of 399 successes) for the nRF52832.

2.4 Glitch attempts and observations

A script [12] was written to control the GIAnT board using the parameters discussed in sections 3.4 and 2.3.2. The finalised code for this script iterates over an offset range of 1027µs – 1100µs at an interval of 1µs, and over a width range of $50ns$ – 200ns at an interval of 10ns. The reason for the large offset range compared to figure 8 is that over the course of a few months, the critical section has been at varying offsets; successful glitches months apart have occurred at offsets of $1029\mu s$ – 1045µs, 1057µs during project demonstration week and 1071µs in December.

Within minutes of starting the aforementioned script, a successful glitch was found and an example is shown in figure 9. The script has since been added to a list of examples in the GIAnT revision B repository [15]. Compare this to figure 6; notice how the CPU core voltage does not drop after the glitch. LimitedResults calls the CRP check a "pure hardware process" due to the absence of boot ROM [2], but the successful glitch seems to also bypass the whole section of slightly dropped voltage before the voltage drops more steeply for application execution. There is no mention of boot ROM—where bootloader code would be retrieved from and executed by the CPU—in the user guide but the lack of a voltage drop after the successful glitch looks more like a branch, and thus a section of code, has been skipped. Much of this can only be speculation, without details provided by the manufacturer.

If some of the Pico Debug'n'Dump shipments had not had the resistor specification issue mentioned in section 2.2, a successful crowbar voltage glitch would have been much more quickly and cheaply attained. On the other hand, focusing on using GIAnT allowed the development and testing of dormant crowbar glitch capabilities on its FPGA-based board, which proved equally effective to the Debug'n'Dump at applying a crowbar glitch.

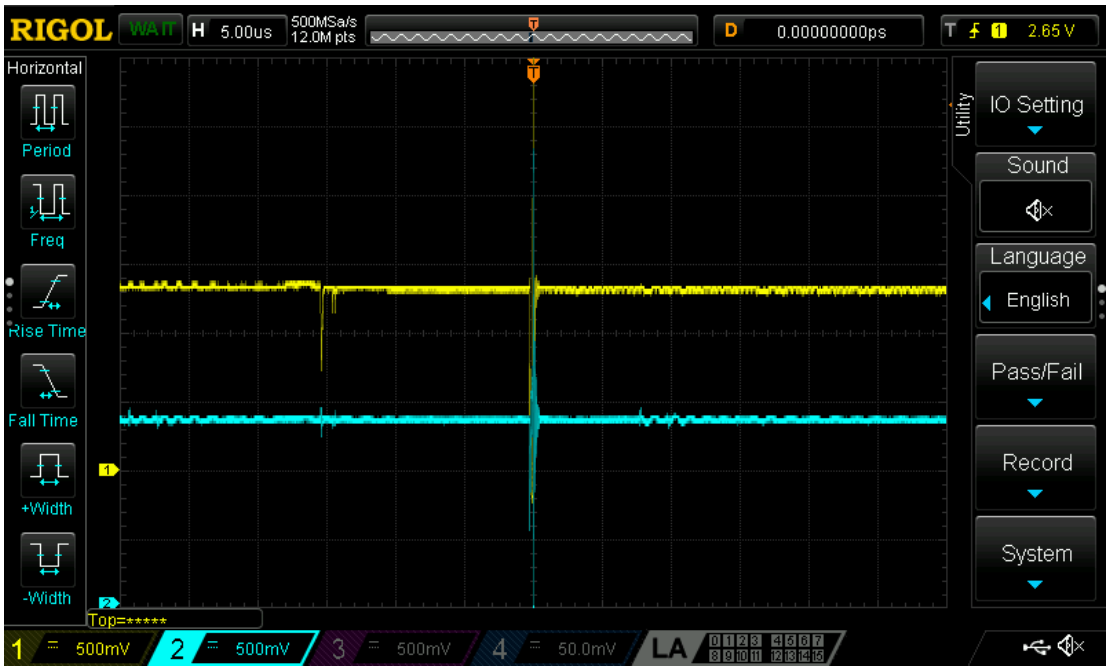

Figure 9: A successful glitch on the nRF52832. Trace 1 (yellow) is the CPU core voltage through `DEC1` and trace 2 (blue) is the system voltage.

3 CC2541 glitch attempts

The CC2541 microcontroller sits within Texas Instruments' CC253x/4x line-up of SoCs, which are all based on a modified 8051 microcontroller, differing from the nRF52832 which uses an ARM Cortex-M4. The main difference between the CC253x and CC254x series is that CC253x SoCs are intended for applications which implement IEEE 802.15.4-based protocols and the CC254x series is intended for bluetooth low-energy applications. As previously stated, the CC2541 is based around a different processor than the nRF52832, but the two chips are similar in that they don't contain boot ROM; there's no read-only bootloader by default on either of the devices. This means that, for a crowbar glitch applied to the CPU to be successful, there needs to be some handling of the CRP data by the CPU.

There were three lessons learned during the nRF52832 glitch attempts, which informed the exploratory glitch campaign on the CC2541:

- When looking for a suitable glitch offset, use an oscilloscope from the beginning; study the power traces for notable features when using the SoC for various activities, especially immediately after reset and power-on. This is known generally as simple power analysis (SPA).
- Read the data sheet and user guide more thoroughly before attempting a voltage glitch, especially for information about bootloaders, which have logic commonly executed by the CPU and as a result can be glitched using the crowbar technique on the CPU decoupling circuit.
- Use the debugging interface to get used to reading the flash contents from the target and define a benchmark against which to check whether CRP is still enabled.

These learning outcomes encouraged a deeper analysis of how the SoC or microcontroller functions *before* leaving the glitch generator running to brute-force search the glitch parameter space.

3.1 Modifications

The CC2541 is similar to the nRF52832 in that it also has a digital supply decoupling pin, `DCOUPL`, for use with a decoupling capacitor. Decoupling capacitors are intended to smooth out any sudden changes in voltage, so it's important that the decoupling capacitors are removed in order to see useful voltage traces when using an oscilloscope, and because they might prevent a successful voltage glitch. See figure 10 for a comparison of `DCOUPL` traces before and after removing the decoupling capacitor. Initially the removal of the capacitor connected to `DCOUPL` was neglected and the voltage trace on it looked very featureless; voltage trace readings were repeated after removing the decoupling capacitor. For easy connection to jumper wires and oscilloscope probes, wires were connected to the following areas on the CC2541 development board:

- `VDD`: Power supply—there was already a header pin for this
- `GND`: Ground—there was already a header pin for this
- `DD`: Debug data pin, for debugging
- `DC`: Debug clock pin, for debugging
- `RESET_N`: Reset pin, for debugging
- `DCOUPL`: 1.8V digital supply decoupling circuit

For all pins other than `VDD` and `GND`, a solder pad on the development board was used to reduce mechanical strain on the chip legs, found using continuity testing between legs and pads. Unlike for the nRF52832 `DCOUPL` was the only decoupling circuit listed in the data sheet, so it was assumed that if a successful crowbar voltage glitch is possible, it would be there.

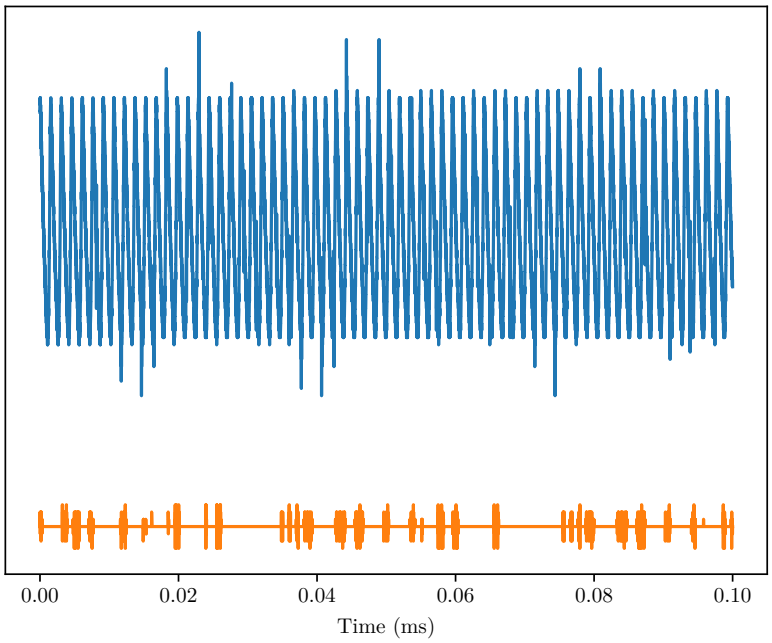

Figure 10: A comparison of the `DCOUPL` power trace before (orange, bottom) and after (blue, top) removing the decoupling capacitor connected to `DCOUPL`. Captured at 250MSa/s.

3.2 Wiring

After soldering work to attach jumper wires, the following connections were made between the GIAnT board, the CC2541 development board and the CC Debugger. The CC Debugger is required for debugging and flashing, due to the lack of bootloader and reliance on proprietary debugging circuits. It can be seen as taking the place of the ST-LINK V2 from the nRF52832 glitch campaign.

GIAnT	CC2541	CC Debugger	Description
`DAC out`	`VDD`	`SENSE`	Supplies CC2541 with 3.3V and the CC Debugger senses it
`T1`	`DCOUPL`		Applies the glitch
`GND`	`GND`	`GND`	Grounds the CC2541
	`DD`	`DD`	Debug data connection
	`DC`	`DC`	Debug clock connection
`GPIO1_6`	`RESET_N`	`RST`	Allows the CC Debugger to reset CC2541 and GIAnT to sense it

Initially, the CC2541 was powered exclusively by the CC Debugger, with DCOUPL and RESET_N being connected to the GIAnT from the CC2541. This was done so the CC Debugger could have full control over the CC2541, while allowing GIAnT to trigger when RESET_N was pulled high. After unsuccessful glitch attempts during and after reset it became apparent that it would be impossible, in this configuration, to apply a glitch triggered by powering on the CC2541—like the nRF52832 attack—because VDD would always remain high if powered by the CC Debugger. The finalised configuration in the table above allows the GIAnT to power up or down the CC2541, while allowing the CC Debugger to reset the CC2541. Even though power is supplied by the GIAnT DAC, the CC Debugger still needs to be connected with the SENSE pin—used when power is supplied to the target by another device—in order to preserve debugging functionality. When the target is being supplied with power by another device, the power supply on the CC Debugger breakout board must be turned off.

Preserving debugging functionality required the following connections to be present for the CC Debugger:

- VDD or SENSE
- GND
- DD
- DC
- RST

As long as SENSE is high (i.e. GIAnT DAC is supplying power to CC2541) when the CC Debugger is first manually reset with its reset button and is high when CC Debugger next communicates with the CC2541, the CC Debugger will correctly interface with the CC2541.

3.3 Glitch experiments plan

For the CC2541, the CRP data is stored as a "debug lock bit": bit 127 (zero-indexed) in the "upper available flash page," according to the user guide. The following experiments will be attempted:

1. Try to glitch a simple application while it's running on the CC2541 to find a suitable glitch width.
2. Use simple power analysis (SPA) to look for potential critical sections—areas where a voltage glitch might cause a CRP bypass.
3. Attempt to glitch the potential critical sections using parameters found in experiment 1.
4. If experiment 3 is unsuccessful, apply differential power analysis (DPA) to look for more definitive critical sections to glitch.

Differential power analysis has been placed last in the list of experiments due to its time-consuming nature; thousands of traces need to be recorded from the oscilloscope, which also must be interfaced with the general purpose computer. Learning how to do this would be time-consuming and specific to experiment 4.

3.3.1 Benchmarking a successful glitch

When setting up the CC2541 for a CRP bypass glitch campaign, the console command `cc-tool -e -f -w application.bin` erases the flash and writes a test application `application.bin` to it, in fast mode. The command `cc-tool -l debug` enables the debug lock bit, enforcing CRP by disabling most debugging functionality, including reading the flash memory to retrieve the application.

A crowbar glitch on DCOUPL which causes a bypass of CRP is considered a successful glitch. In the glitch helper applications [16], [17], which control the GIAnT board, CRP is considered to be disabled if, and only if, the output from the console command `cc-tool -f -r dump.bin --log` contains "Reading flash." Otherwise

CRP is assumed to still be enabled. The flag `--log` is useful for printing more detailed log messages if there is any unusual response, such as an exception or the text "Target is locked" is not in the output. An example log for CRP enabled is listed in appendix A.1 and a log for CRP disabled is listed in appendix A.2. If `cc-tool -f -r dump.bin --log` succeeds and CRP is disabled then the contents of the flash memory are read into `dump.bin` in the working directory. If `dump.bin` is the same as the application written to flash memory before the glitch campaign, the flash memory has been preserved throughout the glitch attempts, and the glitch is considered successful.

3.4 Finding a suitable glitch width

The reasoning behind finding a suitable minimum glitch width is outlined in section 2.3.1. Much like the code written for finding the minimum glitch width for the nRF52832, a test application was written to toggle a pin on the CC2541 at 5ms intervals [18], and a script was created for using the GIAnT board to glitch the CC2541 at an increasing offset [19]. The pin toggling application is configured to toggle pin 0.2 on the CC2541, which is connected to a header pin on its development board labelled "state." The first two pin toggles are only 1ms apart to show more clearly when the application starts. Normal operation of the pin toggling application is shown in figure 11.

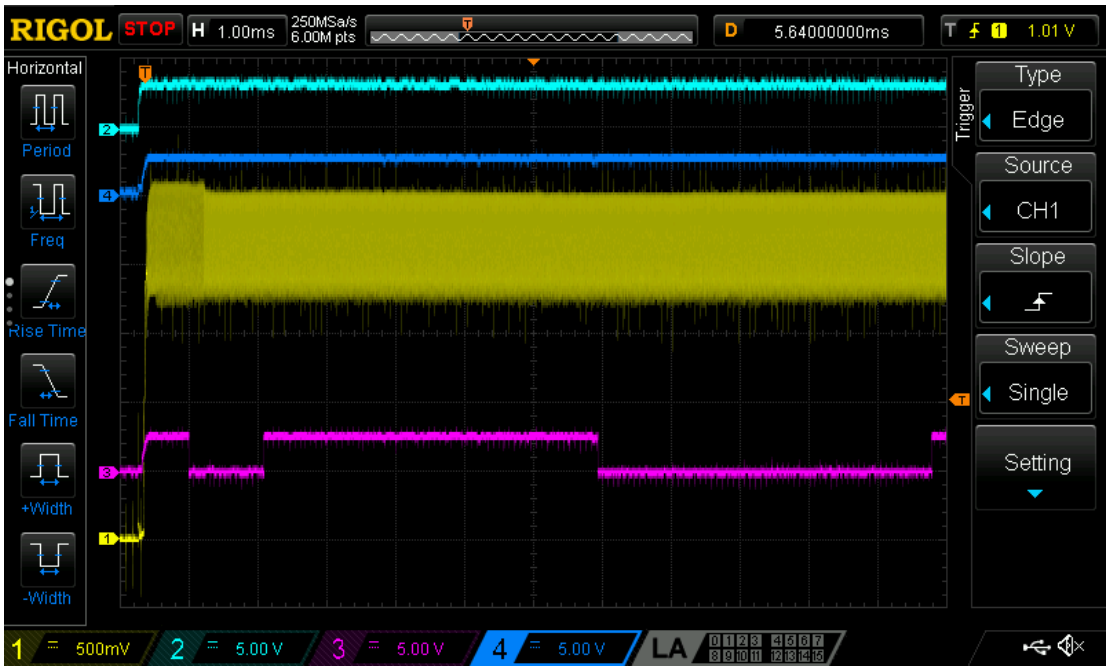

Figure 11: The CC2541 pin toggling application without a glitch. Trace 1 (yellow): `DCOUPL`, trace 2 (light blue): `RESET_N`, trace 3 (purple): STATE on the development board (the pin being toggled), trace 4 (dark blue): Debug Data.

Due to the use of `NOP`s in the `delay_ms` function written by Grapsus [20] for the CC2541 and used by the application, most offsets will not show a discernible glitch outcome, unlike for the nRF52832. This means that, unless the glitch occurs near to within a single clock cycle, or about 31ns, before the "state" pin toggles, the trace will continue to look like the application is running as expected, because only a `NOP` or two are skipped. The most that will happen is that the next pin toggle may come a couple of cycles sooner, which isn't very visible. Glitching within a few clock cycles before the "state" pin toggles does however show a

successful glitch in the application, seen in figure 12. The smallest glitch width for a successful glitch to the pin toggling application was 50ns. 80ns is the maximum glitch width before the microcontroller resets, as seen in figure 13, where a 90ns width was used. The initialisation pattern is visible in trace 1 (`DCOUPL`) in figure 13 and figure 11, which shows the same pattern when powered on.

The reason there are no success rates listed is because the success of each glitch can't easily be determined for the test application (pin toggling) and there was no success with a CRP bypass to be measured.

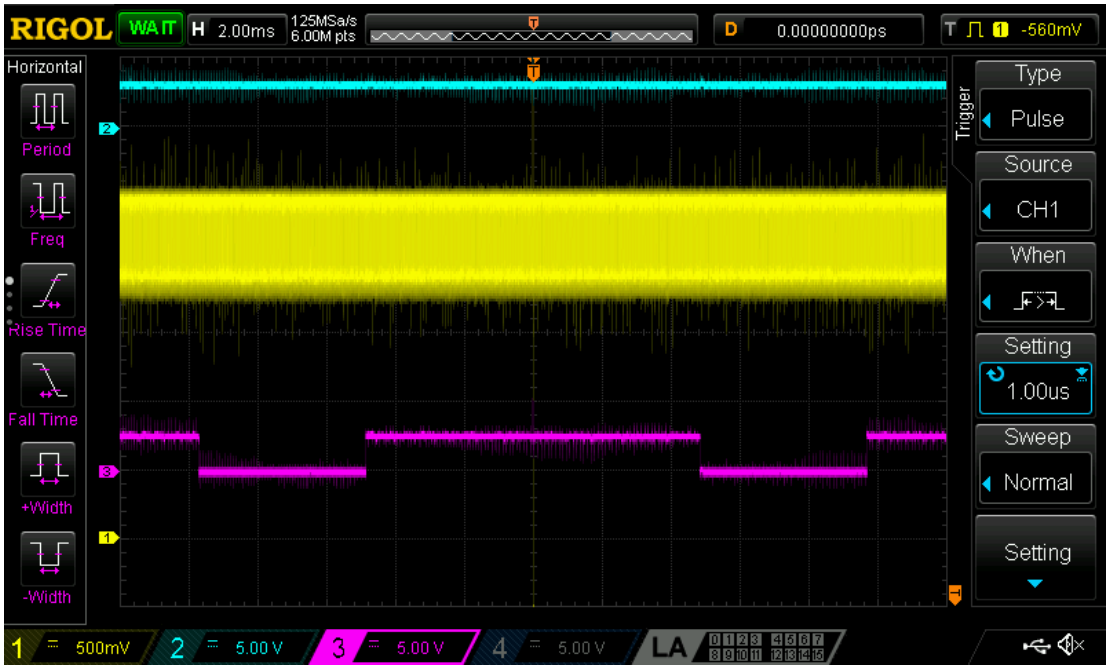

Figure 12: The CC2541 pin toggling application with a 50ns crowbar glitch at trigger T, causing the pin toggle instruction to be skipped and the STATE pin remains high for another 5ms. Trace 1 (yellow): `DCOUPL`, trace 2 (light blue): `RESET_N`, trace 3 (purple): STATE on the development board (the pin being toggled).

3.5 Finding a suitable glitch offset

Generally speaking, potential critical sections for bypassing CRP are found before the application starts running. This is because, even without boot ROM, any CRP check or movement of CRP data to the CPU is an initialisation step, which would need to occur before normal CPU execution and potential debug connection.

In order to identify these potential critical sections and infer more about how the CC2541 works, an oscilloscope was connected to measure the voltages of `DCOUPL`, `VDD`, `RESET_N` and the state pin used by the pin toggling application. The voltage traces should be analysed during the most likely areas in which a CRP check might exist:

- powering on the CC2541 (cold boot, figure 11) or after reset;
- debugging communication using `cc-tool` from the general purpose computer with the CC Debugger (figure 14).

Figure 13: The CC2541 pin toggling application with a 90ns crowbar glitch at trigger T, causing a reset, visible in the pattern in trace 1 and the initial 1ms pin toggle in trace 3, at the start of the application. Trace 1 (yellow): `DCOUPL`, trace 2 (light blue): `RESET_N`, trace 3 (purple): STATE on the development board (the pin being toggled), trace 4 (dark blue): Debug Data.

Figure 14: Using `cc-tool` to read basic info about the CC2541: what its debug status is and its name and flags. Trace 1 (yellow): `DCOUPL`, trace 2 (light blue): `RESET_N`, trace 3 (purple): STATE on the development board (the pin being toggled), trace 4 (dark blue): Debug Data.

In figure 14, it can be seen that before and after every debugging command is sent to the CC2541 by the CC Debugger, `RESET_N` is held low while two falling edge transitions are forced on Debug Clock. It can be assumed that after every reset out of debug mode, the CC2541 reinitialises and CRP data is moved from flash again, if it is handled by the CPU at all. The assumption of this reinitialisation is fair because setting the debug lock with `cc-tool --lock debug` enables CRP for all subsequent debug commands, as seen below.

```
1 $ cc-tool -r out.bin
2   Programmer: CC Debugger
3   Target: CC2540
4   Target is locked.
5   No operations allowed on locked target without erasing
```

In order to check whether a glitch has been successful at bypassing CRP, a non-error response from `cc-tool` which does not contain "Target is locked" is used as the benchmark in the GIAnT scripts [16], [17]. Unless the area of memory containing the debug lock bit is permanently changed—something not likely with a crowbar glitch on the CPU—any outcome from a glitch before `cc-tool` is used will be reverted by the reset before each debug command, which includes the check to see whether the glitch has been successful. Therefore any glitch on a cold boot is unlikely to bypass CRP in a meaningful way, because checking the debug lock bit will reset the chip and re-enable the CRP.

The result of this is that the best chance of a successful glitch on CRP is during the time from the reset before the debug command is sent to the CC2541, until the debug command is processed by the CC2541. This is the period of time between trigger T on figure 14 and 8ms later.

3.6 Glitch attempts

Before carrying out differential power analysis (DPA), glitch attempts were made across the offset space outlined in section 3.5, cold boot and during and immediately after a debugging-triggered reset. Glitch attempts were made using GIAnT interface scripts on both the reset caused by debugging command execution [16] and cold boot [17]. Both scripts cycled, for every glitch offset in those *potential* critical sections, through glitch widths of 50ns to 80ns at 10ns intervals, based on the findings presented in section 3.4 and repeated each glitch width 2 times.

Debugging command-triggered glitches were triggered by falling edge transitions on `RESET_N`, connected to `GPIO1_6` on the GIAnT board. The actual crowbar glitches were applied at offsets ranging from 1µs to 8.1ms at intervals of 0.5µs after the trigger (reset being pulled low), in order to attempt glitches during both reset and debug command handling. Cold boot-triggered glitches were triggered by enabling the DAC on the GIAnT board, supplying `VDD` on the CC2541, and glitches were applied at offsets ranging from 100µs to 700µs at intervals of 0.5µs after the trigger (CC2541 receiving power). Glitching on a cold boot—not during the reset and subsequent debug communication—was never likely to produce a CRP bypass, as described in section 3.5, but was carried out as a fallback after glitching during debug communication didn't produce a CRP bypass.

For all of the offsets at which a glitch was applied, no CRP bypass was achieved. There were two observed outcomes in which the microcontroller didn't reset and there was no "Target is locked" message, but neither of these were a CRP bypass. Neither of these outcomes described occurred after a cold boot glitch; both were during debugging communications.

3.6.1 Outcome: Corrupted debug status

The first time time there was no "Target is locked" message was when the glitch was applied during debug communications (after `RESET_N` has been pulled high again and during Debug Data activity) The `cc-tool` logs show that glitches during debug communications often cause the response or the processing of a request to be corrupted. Notice, in the log message in appendix A.3, the following lines:

```
1 [21.02 13:29:34:99695] programmer, debug status, 00h
2 [21.02 13:29:34:99695] programmer, halt failed
```

See how the debug status returned is `00h` and not `26h` (CRP enabled) or `22h` (CRP disabled). When glitching around the time of this USB read, the value sent back to the CC Debugger can be corrupted and be any other value. This isn't helpful, because even if it is corrupted to `22h`, that doesn't mean that CRP has actually been disabled for subsequent flash reads, only that the message has been corrupted.

3.6.2 Outcome: Erased flash

The second time `cc-tool` did not print a "Target is locked" message after glitching was also when the glitch was applied during debug communications, specifically around the section of activity on trace 4 (Debug Data) and marked by trigger T in figure 15. CRP was only disabled at the expense of erasing the flash, which defeats the purpose of the attack because the flash containing the original application can't then be dumped. Figure 15 shows that after the second time `RESET_N` is pulled low, the pin toggling application doesn't start again. Future power cycles also don't restart the pin toggling application and the device remains unlocked, and any dump of the flash produces an empty binary.

There are two possible reasons that this happened. The first is that repeated glitching for long enough might corrupt the flash, causing it to be wiped. This isn't likely because this outcome only seems to happen during handling of debug commands. The second, more likely reason is that a debug command gets corrupted and interpreted as an erase command, or that erasing is the default operation when receiving such a corrupted command.

Figure 15: The CC2541 glitch during debug communication which was initially thought to be successful. Notice how trace 3 doesn't start toggling again because the application has been wiped. Trace 1 (yellow): `DCOUPL`, trace 2 (light blue): `RESET_N`, trace 3 (purple): STATE on the development board (the pin being toggled), trace 4 (dark blue): Debug Data.

3.7 Differential power analysis

According to Kocher et al., differential power analysis (DPA) is a "statistical method for analyzing sets of measurements to identify data-dependent correlation," and involves splitting a set of power traces into subsets, then computing the difference of the average of these subsets [21]. In this case, the initial set of power traces is a set of power traces for a given time period of the CC2541's power consumption, and there are two subsets: traces where CRP is disabled and traces where CRP is enabled. After applying glitches throughout offsets covering all of the potential critical sections as described in section 3.6, DPA was applied in an effort to reduce the search space, thus focusing glitch attempts on a more likely critical section. Looking for differences between the averages of these two sets could help identify a critical section for a crowbar glitch; a pattern being different in one subset when compared to the other could indicate a glitch-able branch related to CRP checking or handling of CRP data.

3.7.1 Interfacing with the DS1074Z oscilloscope

In order to record accurate power traces and average them, it was necessary to record the points on the trace directly from the oscilloscope, the Rigol DS1074Z Plus, to the general purpose computer for data processing. After experiencing issues with a variety of Python USB interfaces for Rigol oscilloscopes, success at transferring traces over USB was found in the PyVisa interface class at the website *Rocking Wombat* [22], which has also been adapted for the purposes of this project [23].

For a single trace of 3 million points to transfer over USB took around 18 minutes. 3 million points were needed for the time period captured for analysis, because the sample rate needed to be 250MSa/s to capture

the details of a CPU running at 32MHz. DPA requires, ideally, thousands of traces for each subset being compared, in order to average the constituent traces without removing important features from the average trace, especially when there is jitter between each trace. A thousand traces of 3 million points would therefore take 300 hours to capture, which makes accurate DPA prohibitively time-expensive for this particular oscilloscope.

3.7.2 Results

The results of the DPA were inconclusive; due to time constraints only 149 traces were recorded for during and after reset (each contain 3 million points), and 1,246 for cold boot (each contain 300000 points). Graphs displaying the difference between CRP disabled and enabled are shown for during reset and after, when processing debug commands, in figure 16 and during power on in figure 17. The desired outcome of DPA is for there to be one or a few prominent peaks, but there are many similarly-sized peaks in most places on both graphs. This is due to a combination of factors, which couldn't be explored and eliminated due to time constraints:

- Time period of the trace recording: while the cold boot recordings were shorter than the reset recordings (accounting for the increased number of traces) because they focused on a section of initialisation activity about 0.8ms long, it is still a long period of time and clock cycles aren't clearly visible on the graph yet.
- Too few recordings: for cold boot and reset recordings, less than 1000 traces were recorded for each CRP setting. Peaks on the difference trace for cold boot are more prominent than for reset, and even then, it indicates very little.
- It is possible there is no difference between CRP disabled and CRP enabled activity.

The next steps would have started with reducing the trace capture period for the reset to purely the debug communication section (starting at about 7ms in figure 16), skipping the flat line where `RESET_N` is low and allowing for more traces to be recorded in the same amount of time. The glitch which causes a flash erase, found in section 3.6, encourages deeper exploration of the debug communication time period of the voltage traces. Following on from that, recording more of both cold boot and debug communication traces would help eliminate more of the aforementioned limitations.

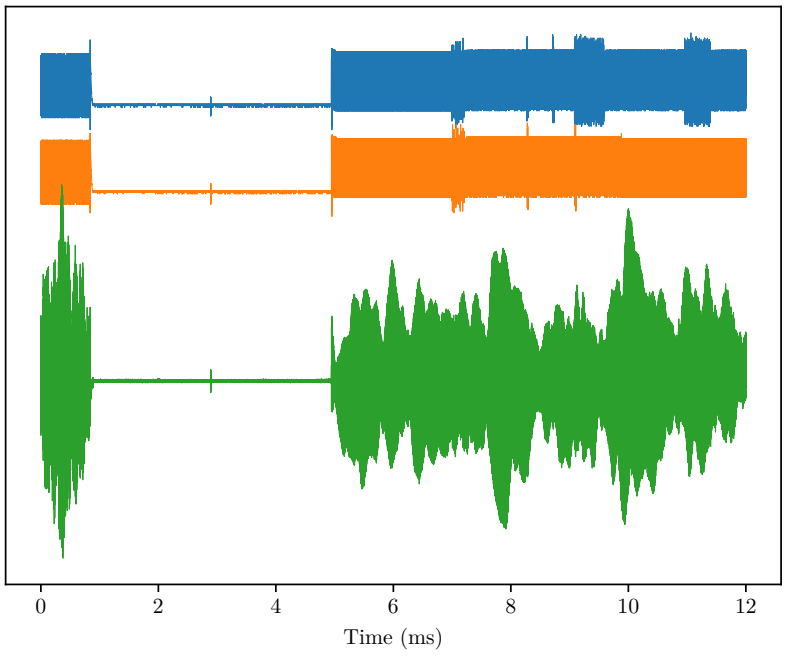

Figure 16: DPA traces of CC2541 during and after reset. Blue trace (top): a single example of CRP disabled, orange trace (middle): a single example of CRP enabled, green trace (bottom): difference between the averages of CRP disabled and enabled traces, multiplied by 15.

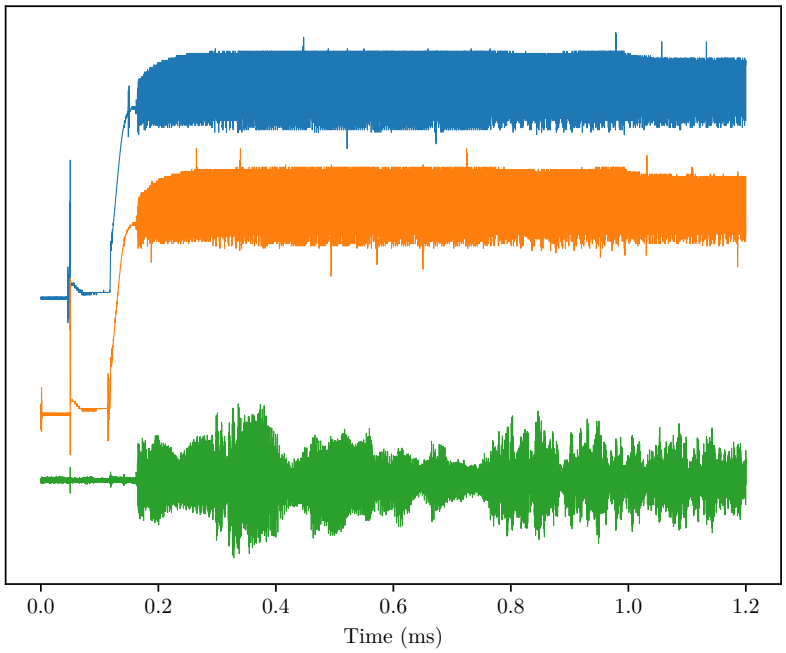

Figure 17: DPA traces of CC2541 after power on. Blue trace (top): a single example of CRP disabled, orange trace (middle): a single example of CRP enabled, green trace (bottom): difference between the averages of CRP disabled and enabled traces, multiplied by 15.

3.8 Areas of difficulty

A number of time-consuming areas of difficulty presented themselves during this project, but many were specific to the analysis of the CC2541. The following were rectified before conclusions were drawn in any of the previous sections.

During initial analysis, removing the decoupling capacitor was seen as an optional step, which it can be if a voltage glitch is found without needing to study voltage traces more deeply. The problem with this was that removing the decoupling capacitor was postponed until it was forgotten, and the first traces recorded for differential power analysis were featureless due to the capacitor's damping effect. Many steps had to be repeated after removing this capacitor, and its removal made power analysis much easier, demonstrated in figure 10.

Removal of the original wire soldered to `DCOUPL` and then its decoupling capacitor put too much strain and heat onto the solder pad used, which consequently lifted off the PCB. The jumper wire is now soldered to a much smaller and less disposable contact: the `DCOUPL` leg on the side of the chip itself. There is no backup for this, but at least the original decision to solder onto the PCB rather than directly onto the pin left a fallback for when the pad came off.

A lot of time was spent trying to glitch some kind of bootloader, which the CC2541 didn't have, just like the nRF52832. It would have been a better idea to reason about causing a CRP bypass by targeting a transfer of CRP data, as done for the nRF52832, because there wasn't likely to be any logic executed by the CPU which could be glitched to cause a CRP bypass, because there was no bootloader embedded on the device.

For the majority of the time spent applying glitches to the CC2541, the glitch width was far too high and was constantly causing either a reset or a full CPU failure until the next reset. This was caused by an initial misunderstanding of how long the glitch was, and then mistaking a glitch-induced reset for a successful glitch, at which point the glitch width stopped being re-evaluated as an independent variable. A closer and more patient analysis of the traces on the oscilloscope would have avoided this.

After seeing that every `cc-tool` command causes a reset before and after the CC2541 handles the debug operations, much time was spent on removing these resets, under the incorrect assumption that they were optional. This was, of course, not true and removing the resets and re-compiling `cc-tool` caused complete failure of debug communication. The user guide makes it clear that putting the CC2541 into debug mode requires `RESET_N` to be held low while applying two falling edge transitions on the Debug Clock pin, but this information was not found until after `cc-tool` had been modified.

Differential power analysis is very difficult to carry out using the oscilloscope used throughout this project, the Rigol DS1074Z Plus. DPA requires, ideally, thousands of recordings of power traces, so that averaging them doesn't completely remove any features from the resulting traces. The DS1074Z oscilloscope is slow at transferring points from power traces over USB, resulting in at least a few minutes to transfer a single trace of 3 million points, which makes DPA prohibitively time-expensive. Switching to another oscilloscope wasn't possible due to time constraints.

All of these time-consuming tasks and mistakes were learning experiences and now it is much easier to recognise various patterns in oscilloscope traces and avoid introducing systematic error.

3.9 Observations

Due to the lack of successful CRP bypass with a crowbar glitch, there are a number of questions still unanswered. It is clear that there is no boot ROM, containing an embedded bootloader, present on the CC2541, much like the nRF52832. This is evident, not only by the lack of mention in the data sheet and user guide, but the lack of any obvious branching in simple power analysis, when comparing between voltage traces of CRP enabled and CRP disabled activity.

However, a crowbar glitch can be successfully applied to the nRF52832, so why is the CC2541 not susceptible to the same kind of attack? It is possible that there is no easily target-able memory copy of CRP data, unlike the nRF52832, or that the CPU core has nothing to do with the CRP data, although this is unlikely, considering the fact that CRP data must still be mapped into addressable memory to be enabled or disabled, and the debug interface communicates with the CPU core. After seeing that some glitch attempts made in section 3.6 cause flash to be completely erased, it is also possible that there is some kind of tamper-proofing for unexpected activity around protected memory, which wipes all flash memory.

It would be arrogant to conclude that a crowbar glitch on `DCOUPL` cannot cause a CRP bypass on the CC2541, but it is very unlikely, considering the number of offsets at which a glitch has been applied. There are other types of attacks that can be attempted in search of a CRP bypass vulnerability. Crowbar voltage glitches are only one type of voltage glitching, and slight overvolting or undervolting could also be attempted, not just on `DCOUPL` but `VDD` too. An example of research which could be executed quickly is to check whether Goodspeed's reported RAM vulnerability of CC2430 [24] is still present on newer chips in the same family, such as the CC2541. Another area of research could be the extraction of secrets through power analysis of `DCOUPL` while the AES encryption and decryption core on the CC2541 is being used.

4 Ease of use of voltage glitching hardware

When it comes to the accessibility of voltage glitching, two factors need to be considered: ease of use of equipment and cost of equipment. This section compares the Pico Debug'n'Dump [6], GIAnT [5] and PocketGlitcher [25], based on those two factors.

The Pico Debug'n'Dump, introduced in section 2.2, is available as a kit—which must be soldered together by the user—costing $25. After shipping and buying the not-included Raspberry Pi Pico, its effective cost is just under £30, making it the cheapest device of the three. It is also very easy to use, due to extensive documentation and simple APIs which are familiar to users who have worked with other Raspberry Pi products.

The FPGA-based board constructed for use with GIAnT software, introduced in section 2.3, is based on the ZTEX USB-FPGA Module 2.04, which costs €129 before designing and attaching custom circuits for the desired applications. This easily makes it more expensive than the other two devices. FPGA configuration is also typically more complicated and most people have less experience with it, making setup of the device much more specialised and difficult. Its more complex, free-form nature also makes it versatile; a board designed for GIAnT can be configured for a wider variety of applications. Using an FPGA rather than a CPU also allows the device to be much more accurate with timing for real-time applications, such as voltage glitching.

The PocketGlitcher consists of a PocketBeagle [26] and adapter board, which has already been designed but requires ordering of parts and assembly. Together, these components would cost just below £50. At the time

of writing, I do not have personal experience with this kit, but it is similar to the Pico Debug'n'Dump in hardware and implementation, with fewer pins than a GIAnT board, so can be considered to have similar ease of use and setup as the Pico Debug'n'Dump.

While the Pico Debug'n'Dump and PocketGlitcher do not have such deterministic and fine-grained timing as FPGA-based boards used with GIAnT, this report has shown that they do still work for crowbar voltage glitching, with the Debug'n'Dump being used successfully to bypass the CRP of an nRF52832 within minutes. The low cost and ease of use definitely make the Debug'n'Dump the best choice of the three for someone with little experience of voltage glitching and hardware, as long as its target is known to be vulnerable to a CRP bypass with a voltage glitch and the only type of glitch needed is a crowbar glitch (e.g. nRF52). For exploration of targets without known voltage glitch vulnerabilities, GIAnT is the better choice due to its ability to generate many different glitches and its timing accuracy.

4.1 Suggested improvements to ease of applying crowbar attacks

While the Debug'n'Dump makes the execution of crowbar attacks easier and cheaper for a user who might wish to repurpose or reverse-engineer consumer hardware, below are suggestions for increasing that ease of use.

- Production-ready, pre-built voltage glitch generators: while soldering is still often needed to attach wires to the target device, soldering a voltage glitch generator kit is time-consuming. If the Debug'n'Dump were available pre-built, this would lower the barrier to entry.
- Reduction of the need to debug each device: there are many components involved in voltage glitch campaigns, such as the debugging adapter, glitch generator, target and all the software needed to interface with them. There are multiple factors that assure a user that they won't need to spend long debugging any of those components.
  - Comprehensive user guides: so far, most resources surrounding voltage glitching are demonstrative, rather than how-to guides. Producing guides on how to use a pre-built glitch generator with various targets would reduce the margin of error for the user.
  - Ready-to-use, purpose-built software: to prevent the need to own an expensive oscilloscope, the software for the glitch generator hardware should be written in a way which ensures that, if the user has followed the guide properly, the software will do the rest (i.e. glitch at conservatively early offsets to prevent the need for manually configuring offsets).

Using these suggestions should make voltage glitching kits more accessible to consumers. For security researchers and those who wish to tinker with newer hardware, custom solutions are still relevant and can be used to inform the designs of purpose-built glitching kits.

5 Evaluation

5.1 Comparing outcomes to success criteria

The crowbar attack on nRF52 microcontrollers shown by LimitedResults [2] and Roth [3] was successfully repeated for the nRF52832 on a cheaply available development board using both the Pico Debug'n'Dump and GIAnT.

This project has shown that neither the CC2541, nor the family of SoCs of which it is a member, are susceptible to crowbar voltage glitches to bypass CRP without erasing the flash memory. In addition, I have learned many new skills throughout:

- Using an oscilloscope for both simple power analysis and voltage trace extraction over USB.
- Deeper understanding of how SoCs and microcontrollers work and their architectures, especially for the nRF52 and CC253x/4x product families.
- Practical experience of voltage glitching using FPGA-based solutions and less real-time microcontroller-based solutions.
- Using all of the software tooling required for the above devices.

Methods for making crowbar attacks and other voltage glitches easier have also been outlined in section 4. Generally, those methods involve turning a voltage glitching kit into a production-ready, pre-built device.

5.2 Future work

If there were more time for this project, it would have extended to include the following work:

- Using a faster oscilloscope to collect more traces in order to produce more accurate DPA results on the CC2541.
- Building on the knowledge gained about how the CC2541 works to attempt to glitch `VDD` on the CC2541, undervolting or overvolting—not using a crowbar attack—to set the debug lock bit.
- Checking whether the CC2541 has the same vulnerability where the RAM isn't erased with the flash, as demonstrated by Goodspeed for the CC2430 [24].
- Improving the ease of use of voltage glitching by refactoring GIAnT into a Python package and adding to the currently sparse documentation.

Future work also mentioned in sections 3.7 and 3.9 includes reducing the trace capture period to only capture debug communication, applying different types of voltage glitch to the CC2541, researching whether a particular RAM vulnerability exists on chips as new as the CC2541 [24] and the extraction of secrets through power analysis of the CC2541.

5.3 Conclusion

This report has demonstrated that the work carried out by LimitedResults [2] and Roth [3] on nRF52 microcontrollers can be repeated with similar results on the nRF52832 using both the Pico Debug'n'Dump and GIAnT. It has been re-affirmed that the code readout protection of the nRF52832 can be bypassed using a crowbar voltage glitch. Aside from the issue encountered with the Debug'n'Dump arriving with a resistor of the incorrect specification, the Debug'n'Dump was a far easier and simpler tool to use in the context of applying a crowbar voltage glitch. An evaluation of ease of use has also been carried out, concluding that the Debug'n'Dump is more accessible than GIAnT when an attacker only wishes to carry out a crowbar glitch,

but GIAnT has a versatility better suited to exploratory research into various implementation attacks on microcontrollers. Improvements to voltage glitching hardware and software have been suggested, and point to making glitching hardware more production-ready: pre-built, with extensive documentation.

Research into and glitch attempts on the CC2541 with the aim of bypassing its CRP have also been carried out. After finding a reliable glitch width for faulting applications and applying glitches of this width across a wide range of offsets, it has been concluded that the possibility of bypassing CRP with a crowbar glitch on the CC2541 is unlikely. A number of reasons why the CC2541 isn't susceptible to this kind of attack have also been proposed, citing the lack of boot ROM and corresponding embedded bootloader, alongside the possibility that its CRP data isn't handled directly by the CPU in a way which can easily be glitched.

Microcontrollers more likely to be vulnerable to this kind of attack include those which have bootloaders embedded in the device in boot ROM (such as LPC176x/5x [27]), which are always run before any application, regardless of the application.

6 References

[1] C. O'Flynn, "Fault Injection using Crowbars on Embedded Systems," 810, 2016. Accessed: Mar. 22, 2022. [Online]. Available: https://eprint.iacr.org/2016/810

[2] LimitedResults, "nRF52 Debug Resurrection (APPROTECT Bypass) Part 1," Jun. 10, 2020. https://limitedresults.com/2020/06/nrf52-debug-resurrection-approtect-bypass/ (accessed Oct. 15, 2021).

[3] *How the Apple AirTags were hacked*, (May 11, 2021). Accessed: Oct. 15, 2021. [Online]. Available: https://www.youtube.com/watch?v=_E0PWQvW-14

[4] J. Van den Herrewegen, D. Oswald, F. D. Garcia, and Q. Temeiza, "Fill your Boots: Enhanced Embedded Bootloader Exploits via Fault Injection and Binary Analysis," *TCHES*, pp. 56–81, Dec. 2020, doi: 10.46586/tches.v2021.i1.56-81.

[5] D. Oswald, "Giant," 2016. https://sourceforge.net/projects/giant/ (accessed Apr. 20, 2022).

[6] T. Roth, "Pico Debug'n'Dump." https://pdnd.stacksmashing.net/ (accessed Oct. 15, 2021).

[7] debug-silicon, *SiLabs C8051F34x code protection bypass*. 2021. Accessed: Jan. 19, 2022. [Online]. Available: https://github.com/debug-silicon/C8051F34x_Glitch

[8] L. Wouters, B. Gierlichs, and B. Preneel, "On the Susceptibility of Texas Instruments SimpleLink Platform Microcontrollers to Non-invasive Physical Attacks," in *Constructive Side-Channel Analysis and Secure Design*, vol. 13211, J. Balasch and C. O'Flynn, Eds. Cham: Springer International Publishing, 2022, pp. 143–163. doi: 10.1007/978-3-030-99766-3_7.

[9] T. Roth, *Airtag-glitcher*. stacksmashing, 2021. Accessed: Oct. 15, 2021. [Online]. Available: https://github.com/stacksmashing/airtag-glitcher

[10] "Glitch pin not shorting to ground · Issue #4 · stacksmashing/airtag-glitcher." https://github.com/stacksmashing/airtag-glitcher/issues/4 (accessed Apr. 08, 2022).

[11] "USB-FPGA Module 2.04: Spartan 6 FPGA Board with EZ-USB FX2 and DDR SDRAM." https://www.ztex.de/usb-fpga-2/usb-fpga-2.04.e.html (accessed Apr. 20, 2022).

[12] J. Rudman, "nRF52832 Full GIAnT Script," 2022. https://github.com/jontyrudman/voltage-glitch-nrf52-cc254x-paper/tree/master/nrf52832/giant/example_nrf52832.py (accessed Sep. 07, 2023).

[13] J. Rudman, "nRF52832 Blink Pin Code," 2022. https://github.com/jontyrudman/voltage-glitch-nrf52-cc254x-paper/tree/master/nrf52832/blink_pin (accessed Sep. 07, 2023).

[14] J. Rudman, "nRF52832 Minimum Glitch Width Helper Script," 2022. https://github.com/jontyrudman/voltage-glitch-nrf52-cc254x-paper/tree/master/nrf52832/giant/nrf52832_min_glitch_width.py (accessed Sep. 07, 2023).

[15] D. Oswald, "Giant-revB: The GIAnT fault injection board in the new revision," 2021. https://github.com/david-oswald/giant-revB (accessed Oct. 15, 2021).

[16] J. Rudman, "CC2541 Command-Triggered Glitch GIAnT Script," 2022. https://github.com/jontyrudman/voltage-glitch-nrf52-cc254x-paper/tree/master/cc2541/giant/example_cc2541.py (accessed Sep. 07, 2023).

[17] J. Rudman, "CC2541 Cold Boot Glitch GIAnT Script," 2022. https://github.com/jontyrudman/voltage-glitch-nrf52-cc254x-paper/tree/master/cc2541/giant/example_cc2541_coldboot.py (accessed Sep. 07, 2023).

[18] J. Rudman, "CC2541 Blink Pin Code," 2022. https://github.com/jontyrudman/voltage-glitch-nrf52-cc254x-paper/tree/master/cc2541/applications/blink_state (accessed Sep. 07, 2023).

[19] J. Rudman, "CC2541 Minimum Glitch Width Helper Script," 2022. https://github.com/jontyrudman/voltage-glitch-nrf52-cc254x-paper/tree/master/cc2541/giant/cc2541_min_glitch_width.py (accessed Sep. 07, 2023).

[20] Grapsus, "Cc254x_sdcc." https://github.com/Grapsus/cc254x_sdcc (accessed Apr. 13, 2022).

[21] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *J Cryptogr Eng*, vol. 1, no. 1, pp. 5–27, Apr. 2011, doi: 10.1007/s13389-011-0006-y.

[22] S. Riester, "DS1074Z und PyVisa." http://www.rocking-wombat.de/PythonPyVisa.html (accessed Apr. 19, 2022).

[23] S. Riester, "DS1074Z Interface Class." https://github.com/jontyrudman/voltage-glitch-nrf52-cc254x-paper/tree/master/general/record_and_dpa/DS1074Z.py (accessed Sep. 07, 2023).

[24] T. Goodspeed, "Extracting Keys from Second Generation Zigbee Chips," p. 3.

[25] LimitedResults, "The PocketGlitcher," Mar. 14, 2021. https://limitedresults.com/2021/03/the-pocketglitcher/ (accessed Oct. 15, 2021).

[26] "BeagleBoard.org - pocket." https://beagleboard.org/pocket (accessed Apr. 20, 2022).

[27] "UM10360 LPC176x/5x User manual," vol. 2016, p. 851, 2016.

A `cc-tool` logs

A.1 CRP enabled

```
1 [19.04 13:20:15:73697] main, cc-tool 0.26
```

```
2 [19.04 13:20:15:73697] main, command line: cc-tool -i --log
3 [19.04 13:20:15:73718] usb, open device, VID: 0451h, PID: 16A2h
4 [19.04 13:20:15:73718] usb, set configuration 1
5 [19.04 13:20:15:73720] usb, claim interface 0
6 [19.04 13:20:15:73723] usb, get string descriptor 2, data: CC Debugger
7 [19.04 13:20:15:73723] programmer, request device state
8 [19.04 13:20:15:73723] usb, control read, request_type: C0h, request:
    C0h, value: 0000h, index: 0000h, count: 8
9 [19.04 13:20:15:73725] usb, control read, data: 40 25 CC 05 44 00 01 00
10 [19.04 13:20:15:73725] device, name: CC Debugger, ID: 0100, version:
    05CCh, revision: 0044h
11 [19.04 13:20:15:73725] programmer, set debug interface speed 0
12 [19.04 13:20:15:73726] usb, control write, request_type: 40h, request:
    CFh, value: 0001h, index: 0000h, count: 0
13 [19.04 13:20:15:73727] programmer, connect target
14 [19.04 13:20:15:73727] programmer, enter debug mode
15 [19.04 13:20:15:73727] usb, control write, request_type: 40h, request:
    C5h, value: 0000h, index: 0000h, count: 0
16 [19.04 13:20:15:73729] usb, control write, request_type: 40h, request:
    C8h, value: 0001h, index: 0000h, count: 48
17 [19.04 13:20:15:73729] usb, control write, data: 43 43 32 35 34 30 20 20
    20 20 20 20 20 20 20 20 20 44 49 44 3A 20 30 31 30 30 20 20 20 20 20 20
    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
18 [19.04 13:20:15:73733] programmer, reset target, debug mode: 1
19 [19.04 13:20:15:73734] usb, control write, request_type: 40h, request:
    C9h, value: 0000h, index: 0001h, count: 0
20 [19.04 13:20:15:73745] programmer, read debug status
21 [19.04 13:20:15:73745] usb, bulk write, count: 2, data: 1F 34
22 [19.04 13:20:15:73746] usb, bulk read, count 1: data: 26
23 [19.04 13:20:15:73746] programmer, debug status, 26h
24 [19.04 13:20:15:73746] programmer, target is locked
25 [19.04 13:20:15:73746] main, start task processing
26 [19.04 13:20:15:73746] programmer, check if unit locked
27 [19.04 13:20:15:73746] programmer, read debug status
28 [19.04 13:20:15:73746] usb, bulk write, count: 2, data: 1F 34
29 [19.04 13:20:15:73746] usb, bulk read, count 1: data: 26
30 [19.04 13:20:15:73746] programmer, debug status, 26h
31 [19.04 13:20:15:73746] main, finish task processing
32 [19.04 13:20:15:73746] programmer, reset target, debug mode: 0
33 [19.04 13:20:15:73746] usb, control write, request_type: 40h, request:
    C9h, value: 0000h, index: 0000h, count: 0
```

A.2 CRP disabled

```
1 [19.04 13:18:09:47648] main, cc-tool 0.26
```

```
[19.04 13:18:09:47648] main, command line: cc-tool -i --log
[19.04 13:18:09:47669] usb, open device, VID: 0451h, PID: 16A2h
[19.04 13:18:09:47669] usb, set configuration 1
[19.04 13:18:09:47671] usb, claim interface 0
[19.04 13:18:09:47674] usb, get string descriptor 2, data: CC Debugger
[19.04 13:18:09:47674] programmer, request device state
[19.04 13:18:09:47675] usb, control read, request_type: C0h, request:
    C0h, value: 0000h, index: 0000h, count: 8
[19.04 13:18:09:47676] usb, control read, data: 40 25 CC 05 44 00 01 00
[19.04 13:18:09:47676] device, name: CC Debugger, ID: 0100, version:
    05CCh, revision: 0044h
[19.04 13:18:09:47677] programmer, set debug interface speed 0
[19.04 13:18:09:47677] usb, control write, request_type: 40h, request:
    CFh, value: 0001h, index: 0000h, count: 0
[19.04 13:18:09:47678] programmer, connect target
[19.04 13:18:09:47678] programmer, enter debug mode
[19.04 13:18:09:47678] usb, control write, request_type: 40h, request:
    C5h, value: 0000h, index: 0000h, count: 0
[19.04 13:18:09:47680] usb, control write, request_type: 40h, request:
    C8h, value: 0001h, index: 0000h, count: 48
[19.04 13:18:09:47680] usb, control write, data: 43 43 32 35 34 30 20 20
    20 20 20 20 20 20 20 20 44 49 44 3A 20 30 31 30 30 20 20 20 20 20 20
    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
[19.04 13:18:09:47685] programmer, reset target, debug mode: 1
[19.04 13:18:09:47685] usb, control write, request_type: 40h, request:
    C9h, value: 0000h, index: 0001h, count: 0
[19.04 13:18:09:47696] programmer, read debug status
[19.04 13:18:09:47696] usb, bulk write, count: 2, data: 1F 34
[19.04 13:18:09:47697] usb, bulk read, count 1: data: 22
[19.04 13:18:09:47697] programmer, debug status, 22h
[19.04 13:18:09:47697] programmer, write debug config, 22h
[19.04 13:18:09:47697] usb, bulk write, count: 3, data: 4C 1D 22
[19.04 13:18:09:47697] programmer, read xdata memory at 6276h, count: 2
[19.04 13:18:09:47697] usb, bulk write, count: 47, data: 40 55 00 72 56
    E5 92 BE 57 75 92 00 74 56 E5 83 76 56 E5 82 BE 57 90 62 76 4E 55 E0
    5E 55 A3 4F 55 E0 5E 55 A3 D4 57 90 C2 57 75 92 90 56 74
[19.04 13:18:09:47700] usb, bulk read, count 2: data: 4C 07
[19.04 13:18:09:47700] programmer, read xdata memory, data: 4C 07
[19.04 13:18:09:47701] programmer, read xdata memory at 6249h, count: 1
[19.04 13:18:09:47701] usb, bulk write, count: 41, data: 40 55 00 72 56
    E5 92 BE 57 75 92 00 74 56 E5 83 76 56 E5 82 BE 57 90 62 49 4F 55 E0
    5E 55 A3 D4 57 90 C2 57 75 92 90 56 74
[19.04 13:18:09:47704] usb, bulk read, count 1: data: 22
[19.04 13:18:09:47704] programmer, read xdata memory, data: 22
[19.04 13:18:09:47704] programmer, read xdata memory at 624Ah, count: 1
```

```
[19.04 13:18:09:47704] usb, bulk write, count: 41, data: 40 55 00 72 56
    E5 92 BE 57 75 92 00 74 56 E5 83 76 56 E5 82 BE 57 90 62 4A 4F 55 E0
    5E 55 A3 D4 57 90 C2 57 75 92 90 56 74
[19.04 13:18:09:47707] usb, bulk read, count 1: data: 8D
[19.04 13:18:09:47707] programmer, read xdata memory, data: 8D
[19.04 13:18:09:47707] target, name: CC2540, chip ID: 8Dh, rev. 22h,
    flash: 256, ram: 8, flags: 07h
[19.04 13:18:09:47707] main, start task processing
[19.04 13:18:09:47707] programmer, check if unit locked
[19.04 13:18:09:47707] programmer, read debug status
[19.04 13:18:09:47707] usb, bulk write, count: 2, data: 1F 34
[19.04 13:18:09:47709] usb, bulk read, count 1: data: 22
[19.04 13:18:09:47709] programmer, debug status, 22h
[19.04 13:18:09:47709] programmer, read info page
[19.04 13:18:09:47709] programmer, read xdata memory at 7800h, count: 128
[THE INFO PAGE IS READ, HEX DUMP REMOVED FROM LOG]
[19.04 13:18:10:48681] main, finish task processing
[19.04 13:18:10:48681] programmer, reset target, debug mode: 0
[19.04 13:18:10:48681] usb, control write, request_type: 40h, request:
    C9h, value: 0000h, index: 0000h, count: 0
```

A.3 Corrupted debug status

```
Command '['cc-tool', '-i', '--log']' returned non-zero exit status 1.
[21.02 13:29:34:99664] main, cc-tool 0.26
[21.02 13:29:34:99664] main, command line: cc-tool -i --log
[21.02 13:29:34:99672] usb, open device, VID: 0451h, PID: 16A2h
[21.02 13:29:34:99672] usb, set configuration 1
[21.02 13:29:34:99680] usb, claim interface 0
[21.02 13:29:34:99682] usb, get string descriptor 2, data: CC Debugger
[21.02 13:29:34:99682] programmer, request device state
[21.02 13:29:34:99682] usb, control read, request_type: C0h, request:
    C0h, value: 0000h, index: 0000h, count: 8
[21.02 13:29:34:99682] usb, control read, data: 40 25 CC 05 44 00 01 00
[21.02 13:29:34:99682] device, name: CC Debugger, ID: 0100, version:
    05CCh, revision: 0044h
[21.02 13:29:34:99682] programmer, set debug interface speed 0
[21.02 13:29:34:99682] usb, control write, request_type: 40h, request:
    CFh, value: 0001h, index: 0000h, count: 0
[21.02 13:29:34:99682] programmer, connect target
[21.02 13:29:34:99682] programmer, enter debug mode
[21.02 13:29:34:99682] usb, control write, request_type: 40h, request:
    C5h, value: 0000h, index: 0000h, count: 0
[21.02 13:29:34:99683] usb, control write, request_type: 40h, request:
    C8h, value: 0001h, index: 0000h, count: 48
```

```
18 [21.02 13:29:34:99683] usb, control write, data: 43 43 32 35 34 30 20 20
      20 20 20 20 20 20 20 20 44 49 44 3A 20 30 31 30 30 20 20 20 20 20 20
      20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
19 [21.02 13:29:34:99684] programmer, reset target, debug mode: 1
20 [21.02 13:29:34:99684] usb, control write, request_type: 40h, request:
      C9h, value: 0000h, index: 0001h, count: 0
21 [21.02 13:29:34:99694] programmer, read debug status
22 [21.02 13:29:34:99694] usb, bulk write, count: 2, data: 1F 34
23 [21.02 13:29:34:99695] usb, bulk read, count 1: data: 00
24 [21.02 13:29:34:99695] programmer, debug status, 00h
25 [21.02 13:29:34:99695] programmer, halt failed
```