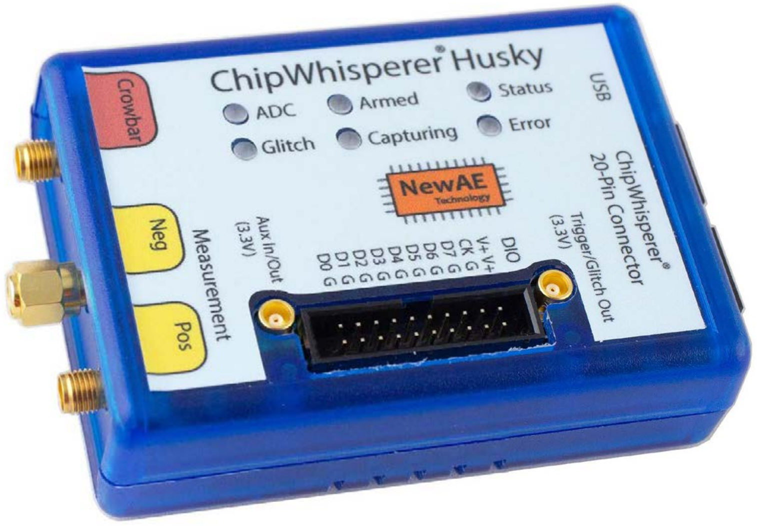
ChipWhisperer-Husky

Care & Feeding Instructions



Last Update: March 28/2023

[EN] WARNINGS:

- Any external power supply used with the product shall comply with relevant regulations and standards applicable in the country of intended use. The power supply should provide a voltage of 5VDC and minimum rated current of 500mA.
- Do not expose this product to water or moisture.
- Do not allow conductive materials such as wires or aluminum foil to come into contact with the product.
- Take care when handling the product to avoid mechanical or electrical damage.
- Avoid electrostatic discharge damage by handling the device only in an electrostatic discharge protective area.

WEEE Directive Statement for European Union

This marking indicates the product should not be disposed with other household wastes throughout the EU. To prevent possible harm to the environment or human health, recycle it responsibly to promote the sustainable reuse of material resources. Please contact us at compliance@newae.com and we will provide a return service, or will otherwise ensure you can safely dispose of this product at no cost to yourself.

[DE] WARNUNG:

- Das zur Stromversorgung verwendete externe Netzteil muss den Vorschriften und Normen des Landes entsprechen, in dem es verwendet wird. Es muss 5 V (Gleichstrom) und einen Mindest-Nennstrom von 500mA liefern.
- Das Produkt darf nicht in Kontakt mit Wasser gelangen oder Feuchtigkeit ausgesetzt werden.
- Achten Sie auf einen sorgsamen im Umgang mit dem Produkt, so dass keine mechanischen und elektrischen Schäden am Produkt entstehen können.
- Achten Sie darauf, dass keine leitenden Materialien mit dem Produkt in Berührung kommen.
- Arbeiten Sie zur Vermeidung von Schäden durch elektrostatische Entladung (ESD) ausschließlich in ESD-geschützten Bereichen bzw. an dafür vorgesehenen ESD-Arbeitsplätzen.

Entsorgung:

Das Produkt soll einer umweltgerechten Wiederverwertung zugeführt werden. Entsorgen Sie es nicht im Restmüll/Hausmüll!

Bei Fragen zur Entsorgung setzen Sie sich bitte per E-Mail mit uns in Verbindung (compliance@newae.com), damit wir sicherstellen können, dass eine sichere und umweltgerechte Entsorgung des Produktes ohne zusätzliche Kosten möglich ist.

Nur für EU-Länder:

Gemäß der Europäischen Richtlinie 2012/19/EU über Elektro- und Elektronik-Altgeräte und ihrer Umsetzung in nationales Recht müssen nicht mehr gebrauchsfähige Elektro- und Elektronik-Produkte getrennt gesammelt und einer umweltgerechten Wiederverwertung zugeführt werden.

Bei unsachgemäßer Entsorgung können Elektro- und Elektronik-Altgeräte aufgrund des möglichen Vorhandenseins gefährlicher Stoffe schädliche Auswirkungen auf die Umwelt und die menschliche Gesundheit haben.

Introducing the ChipWhisperer-Husky

The *ChipWhisperer* started as an open-source project capable of performing advanced hardware attacks like power analysis and fault injection. It's grown to include a wide range of both hardware and software, always with the goal of making hardware security *accessible*.

We make our tools accessible by ensuring we can commercially support it, making it well documented, avoiding artificial limitations, and keeping it as *open as we reasonably can*. While the production design files for ChipWhisperer-Husky aren't open source, the interesting parts you might want to modify, such as the FPGA codebase, microcontroller firmware, and target board design files *are* open-source.

We hope you enjoy using this tool, and thank you for sharing our vision of a future where hardware security tooling is available to students and engineers everywhere!

** Alex Dewar*	** Jean-Pierre Thibault*	** Claire Frias*
** Melissa Armbruster*	** Hilary Taylor*	** Colin O'Flynn*

Special thanks to the QA team:

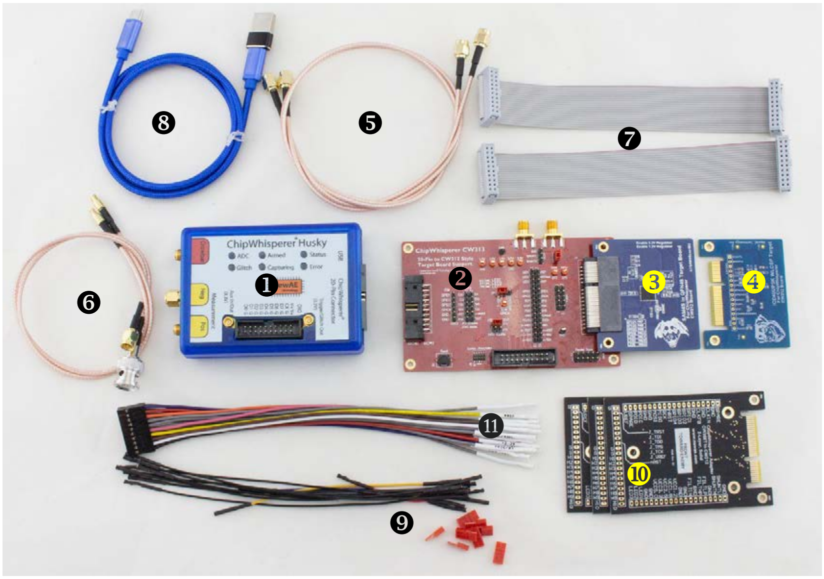
** Luna*	** Bergen*	** Hero*
		** Foster (consultant)*

Table of Contents

About This "Manual"

Most of the project documentation is available online, which will always be the most up-to-date. This manual is partly a quick reference and partly an introduction to what you can do with the ChipWhisperer-Husky. You may need additional resources, which we've linked to if you haven't worked with power analysis or fault injection before.

Kit Contents



①	ChipWhisperer Husky	⑥	MCX to SMA & MCX to BNC
②	CW313 Board	⑦	20-Pin Cables (×2)
③	SAM4S Target (Inserted)	⑧	USB-C Cable (with USB-A adapter)
④	iCE40 Target	⑨	Jumper Wires & Caps
⑤	SMA Cables (×2)	⑩	CW308 Target to CW312 adapter
⑪	ChipWhisperer 20-Pin Connector breakout wires		

Some parts (especially jumpers & cables) may look slightly different. Some adapters are pre-installed, for example the USB-C to A adapter can be pulled off the USB-C cable if you don't need a USB-A end.

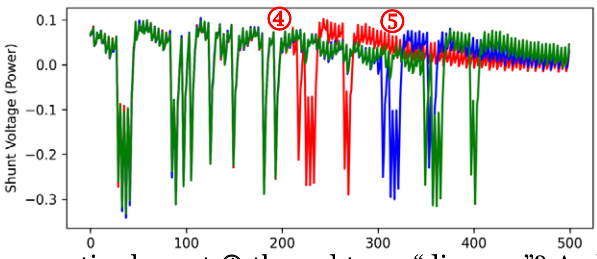
Power Analysis Side Channels

Power analysis is a method of finding *secrets* using a *power* side-channel. It turns out that digital devices consume different amounts of power depending on what they are doing. As a simple example, take a look at this comparison loop:

```
bool is_password_ok(char input *)
{
    char known_pw[] = "doggy"; ①
    for(int j = 0; j < 5; j++){
        if (input[j] != known_pw[j])
            return false; ②
    }
    return true; ③
}
```

This simple function has a known password stored at ①(doggy), which as soon as a character is wrong with the input, it returns false at ②. If it never sees a wrong character, it returns true at ③.

Different operations actually take different amounts of power; even a small microcontroller shows obvious differences if we look closely. Take a look at a recording made of the power used by the `is_password_ok()`function, with three different inputs:

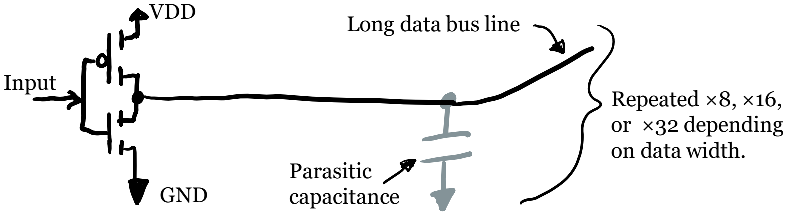


Whoa – notice how at ④ the red trace "diverges"? And then the blue trace diverges at ⑤? The red trace is the guess daddy, the blue trace is dog, and green is doggs. As more correct characters are in the guess, we can see the loop progress further. This lets us brute force the password one character at a time.

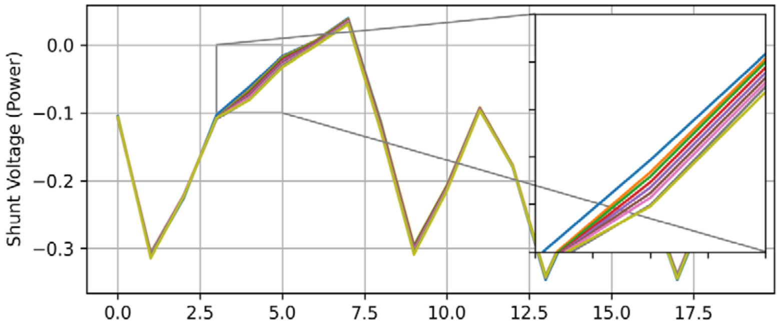
You'll do this yourself in SCA101, Lab 2.1B

Differential Power Analysis

You'll often hear of *differential power analysis*, which refers to the fact we use different data with the same operation. This is even crazier than before – not only do devices tell you *what operations* they are doing, but you can also determine the *exact data* being processed! It works because internally moving data around means charging and discharging data lines – this is normally done with something like this:



Charging and discharging the lines takes physical *power*. Taking a simple operation, such as loading a single byte with different values, and plotting what the power consumption looks like will give us something like this:



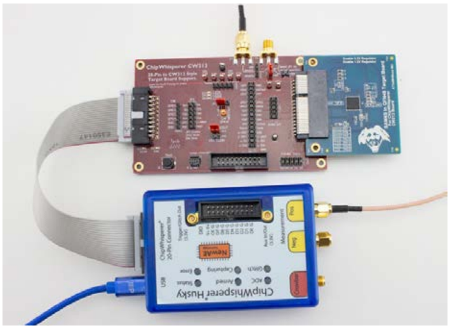
It might be hard to see, but there are nine distinct "splits" the power trace takes. These nine distinct splits correspond to how many 1-bits are in the byte (you can have zero 1-bits, one 1-bit, etc., up to eight 1-bits, so you end up with nine groups). We can thus learn the number of 1's in a byte being processed.

You'll see this split in SCA101, Lab 3.1

You'll use this data to break AES SCA101, Lab 3.3

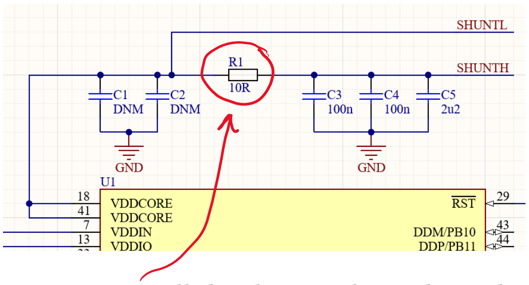
Power Analysis with the Husky

Capturing these detailed power traces is what the ChipWhisperer-Husky is designed for! A typical setup will look like this on your desk:



The ChipWhisperer-Husky performs the analog measurement (like an oscilloscope), where the target device is running your code with a secret inside it. For all the labs, the ChipWhisperer-Husky also serves as a communication and programming interface.

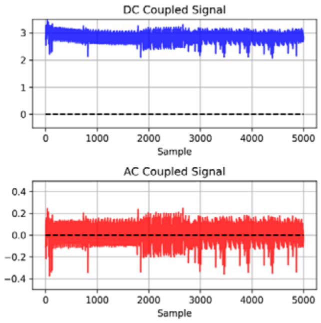
The Target board will normally consist of the CW313 "baseboard" along with a target plugged into it. To make power measurements easier, the target boards are specially designed to provide you with clean power traces. Take a look at part of the SAM4S schematic:



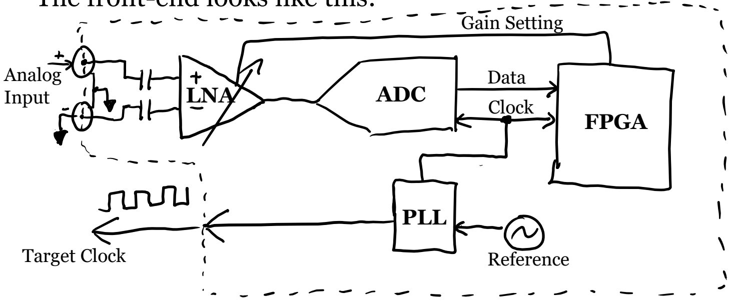
A *shunt resistor* (R1) will develop a voltage depending on the current flowing through it. This is measured at SHUNTL, which connects to the ChipWhisperer-Husky through the SMA cable.

Power Analysis with the Husky

We need to measure the difference across the shunt resistor. To simplify the setup, the input of the ChipWhisperer-Husky is *AC coupled*, which means it sees only the variations in the *SHUNTL* value. This works because the power on the "other side" (SHUNTH) is very clean, thanks to a filter in the CW313 baseboard.

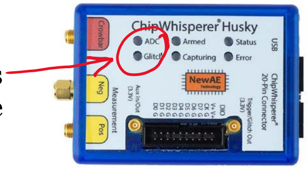


The ChipWhisperer-Husky front-end contains a powerful Low Noise Amplifier (LNA), with up to 70 dB of gain (3000×). Unlike an oscilloscope, the ChipWhisperer can *only* measure very small signals. This makes it ideal for power analysis work. The front-end looks like this:



The analog to digital converter (ADC) has a fixed input range, so the adjustable gain of the LNA lets you get the best signal out of the ADC by adjusting the gain so that it's not clipping (going beyond the ADC range). ADC clipping is flagged as an error. Any errors cause the ADC and glitch LEDs to blink together.

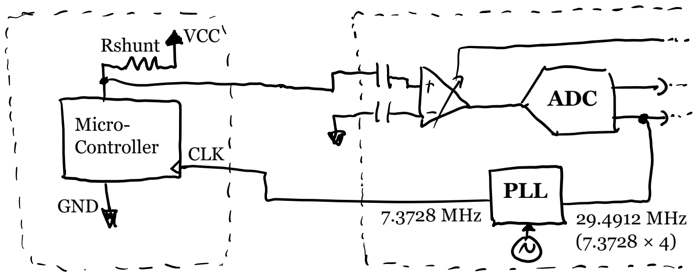
If you see these LEDs blinking, check & clear the error from Python.



Synchronous Sampling

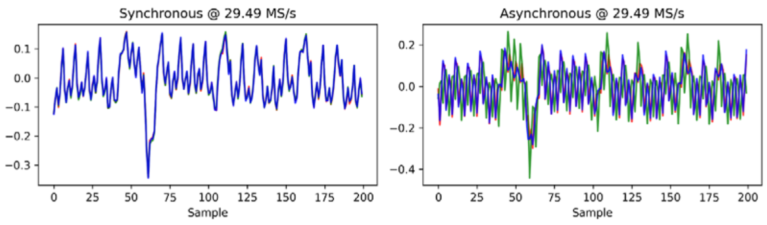
The ChipWhisperer-Husky also contains a unique feature not found in oscilloscopes: the ability to sample synchronously to the target device.

On the previous page, you might notice that a Phase Locked Loop (PLL) drives both the ADC and the Target clock. Our standard setup looks like this:



7.3728 MHz is a value that easily divides down to common baud rates – the reason for such a funky number.

The same clock source is used to drive the target microcontroller as is used to time the samples of the power measurement. Here is an example comparing the exact same operation recorded three times with synchronous and with asynchronous sampling:

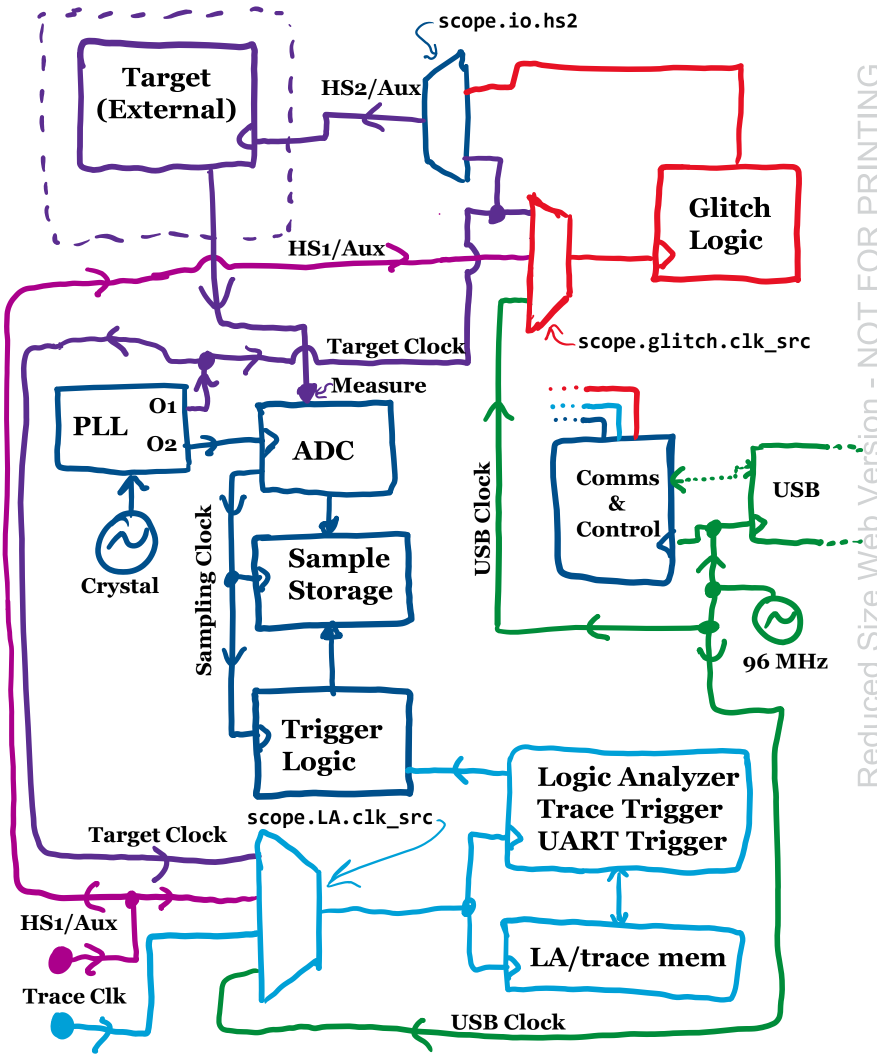


With synchronous sampling the samples of the power traces are perfectly aligned. For more details on this see the paper:

C. O'Flynn, Z. Chen. A case study of Side-Channel Analysis using Decoupling Capacitor Power Measurement with the OpenADC. 2013.

ChipWhisperer-Husky Clocking

Clocking in the ChipWhisperer-Husky is very complex. This page is a quick reference for what sources you can use and how they connect.



Glitching (or Fault Injection)

The ChipWhisperer is also a tool for performing glitch (also called fault injection) attacks. These attacks concentrate on the fact that we can actually cause a microcontroller to perform *incorrect* operations. This becomes a problem with lots of security-conscious code, such as checking if a signature is valid:

```
if(sig_ok){
    boot_os();
} else {
    while(1);①
}
```

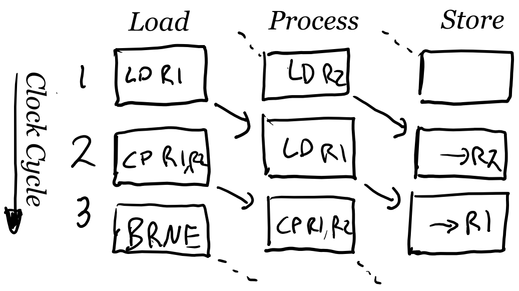
We could "glitch" the device such that it incorrectly calls `boot_os()`, by corrupting the value of `sig_ok`, or even skipping an instruction. Take a look at the assembly code for the above:

```
        ldr     r3, .SYM_sig_ok
        ldr     r3, [r3]
        cbnz    r3, .L6
.L2:
        b       .L2 ②
.L6:
        bl      boot_os() ③
        movs    r0, #0
```

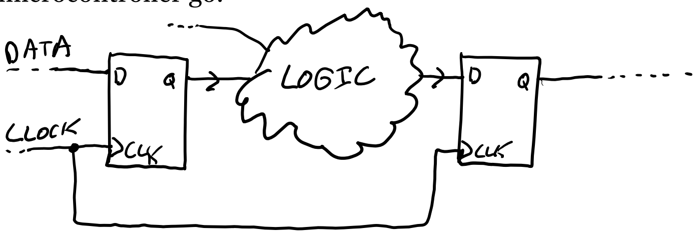
The source code has an infinite loop at ①. The compiler optimizes the code-flow, and now if the jump instruction is skipped in the assembly version at ② it will call that important function at ③ that should never have been called. Oops!

This often works because flip-flops and other digital elements have important timing constraints, such as a flip-flop requiring that the data is present before the clock. If we can make those constraints fail, incorrect data will be loaded (or processed). And by "data" we can mean anything – like loading the wrong instruction, loading the wrong argument, or loading the program counter (PC) value!

Even a simple microcontroller has a *pipeline*, which does different operations on a given clock cycle in order to load, process, and store the instructions that make up your program. This makes it a sensitive area for glitches. You'll often find glitching directly impacts the program flow itself.



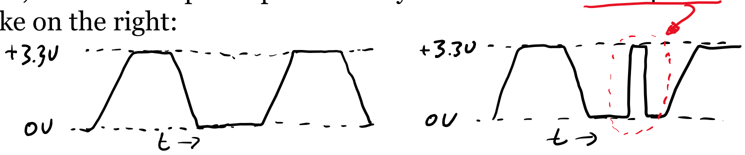
In the simple pipeline above, there would be sets of registers that hold data moving between each stage. Each register is made up of many flip-flops. In the following diagram, you can see how the same clock drives two different flip-flops, with the logic between them. Logic contains gates, such as AND, OR, and XOR gates, used to build up functions that make the microcontroller go.



Normally the data processes through the logic and arrives at the input (D) before the clock pulse. Glitching tries to either impact the *propagation delay* so that the data or clock arrives at the wrong time, change the clock timing, or add extra pulses such that the clock arrives before the data is ready.

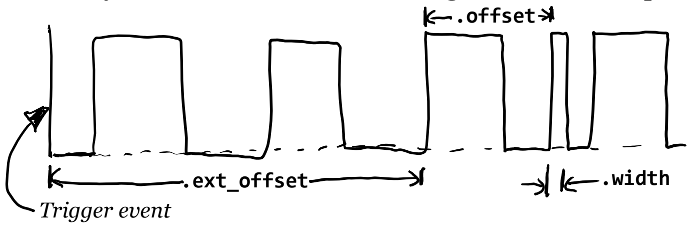
Clock Glitching with the Husky

Clock glitching tries to insert extra clock pulses into the clock of the device. A typical clock looks nice and steady like on the left, but with ChipWhisperer-Husky we can insert extra pulses like on the right:



The ChipWhisperer-Husky uses an FPGA to do this, by phase-shifting the "original" clock into multiple copies. This ensures the glitch always scales with the frequency.

You can adjust the width & offset of the glitch within the pulse:



You can also glitch multiple cycles (repeat), and choose the offset between the trigger and when the glitches start to flow (`ext_offset`).

Because you might not have a fast enough logic analyzer to see what is happening on the hardware, the ChipWhisperer-Husky has a 300 MS/s logic analyzer built in.

The "glitchy clock" is sent over the 20-pin cable to the target's clock input for the labs.

Clock glitching is less common these days, as many devices run on internal oscillators or don't directly use the external clock.

Check out the following labs to see clock glitching in action:

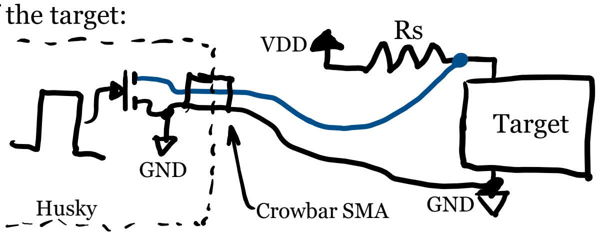
You'll explore the parameters in Fault101, Lab 1.1

You'll attack actual code in Fault101, Labs 1.2 & 1.3

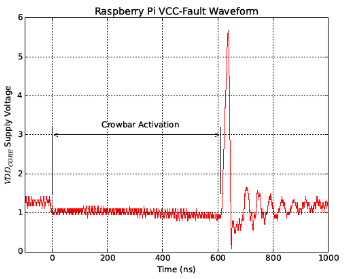
Voltage Glitching with the Husky

Voltage glitching is trying to impact the internal power rails of the target device. This can be done in many ways, such as driving positive or negative voltage spikes onto the power rails.

The ChipWhisperer-Husky uses a "crowbar" mechanism that was pioneered in the ChipWhisperer hardware. It simply uses a MOSFET (electronic switch) to short the power rails of the target:



Because the targets for ChipWhisperer have a resistive shunt (Rs), you get very clean power drops. External targets may have more ringing, such as the Raspberry Pi waveform from the right.



You often need multiple glitches. You can try multiple narrow glitches, but most of the time we instead use a long glitch created with a mode called `enable_only`. Voltage glitching can work on almost any target – see the link below for an example of glitching a Raspberry Pi running Linux using voltage glitching:

https://www.youtube.com/watch?v=dVkCNiM0PL8

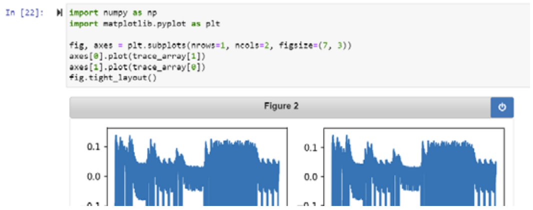
Check out the following labs to see voltage glitching in action:

You'll explore the parameters in Fault101, Lab 2.1
You'll attack actual code in Fault101, Labs 2.2 & 2.3
You'll attack RSA in Fault201, Labs 2.1

Installing ChipWhisperer

When working with ChipWhisperer, you will often be *interactively* working with the target. This means rather than just passively observing like with an oscilloscope, you'll be sending data, triggering, and toggling power, while observing different outputs.

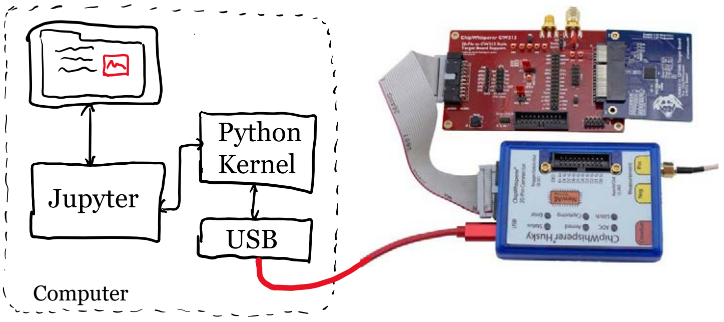
For this reason using ChipWhisperer is best done through programming scripts. To make this easier we use something called Jupyter, that interactively runs Python, shows plots, and can call system commands (like compilers). Your view will look like this:



```
In [22]:   import numpy as np
           import matplotlib.pyplot as plt

           fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))
           axes[0].plot(trace_array[1])
           axes[1].plot(trace_array[0])
           fig.tight_layout()
```

The general connection looks like this:



The USB cable provides the connection to the ChipWhisperer-Husky. Through this several interfaces are exposed, including serial ports used for debugging and interacting with the target.

Of course, you can use Python scripts directly without the Jupyter interface. We use Jupyter to provide all our example labs, but the actual interface and API is a standard Python package.

Installing on Windows

As you need many tools which are *not* typically installed on Windows, we recommend using the latest ChipWhisperer Windows Installer (available on the ChipWhisperer GitHub under “releases”).

This will install Python, Jupyter, the ChipWhisperer repository, and the typical compilers you will need to build the samples. It also ensures you have the correct working version of all these tools.

These tools all get installed in a single directory, so even if you already have tools installed (such as make), the ChipWhisperer ones will be isolated from existing tools on your system. The driver should automatically load on plugging in the Husky.

Installing on *nix

On most Linux or Mac systems, the tools which you need to run ChipWhisperer are available from the repositories. You can easily install the required tools by following the latest install instructions at the links below. You will need to provide permission for USB access, as well.

Using Virtual Machine

A Virtual Machine (VM) which runs the Jupyter system is available. This can be useful in training environments. See links below for more information.

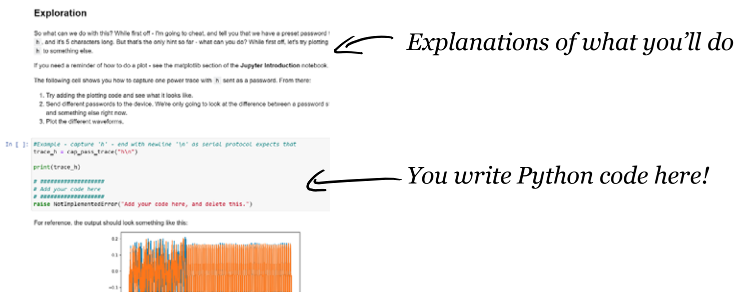
Installation Links

The latest links are available from **ChipWhisperer.com** or the ChipWhisperer GIT Repository. Currently, the installation instructions are part of the ChipWhisperer Software Documentation:

https://chipwhisperer.readthedocs.io/

Getting Started

We always recommend starting with our labs – they walk you through how Power Analysis & Fault Injection work. For example, the description of that password check is part of Lab 2-1B in SCA101:



Exploration

So what can we do with this? While first off - I'm going to cheat, and tell you that we have a preset password `h`, and it's 5 characters long. But that's the only hint so far - what can you do? While first off, let's try plotting `h` to something else.

If you need a reminder of how to do a plot - see the matplotlib section of the **Jupyter Introduction** notebook.

The following cell shows you how to capture one power trace with `h` sent as a password. From there:

1. Try adding the plotting code and see what it looks like.
2. Send different passwords to the device. We're only going to look at the difference between a password s and something else right now.
3. Plot the different waveforms.

```
In [ ]: #Example - capture 'h' - end with newline '\n' as serial protocol expects that
        trace_h = cap_pass_trace("h\n")

        print(trace_h)

        # ###################
        # Add your code here
        # ###################
        raise NotImplementedError("Add your code here, and delete this.")
```

For reference, the output should look something like this:

The labs typically include a *solution* notebook. We've purposely left some code out of the lab notebooks, but if you get stuck you can always look at the solution notebook. You can also run the *solution* notebook to see how things should work!

You'll need to modify the **PLATFORM** line for each lab and set it to the target platform, **CW308_SAM4S** . This makes sure the right compiler and settings are selected.

```
SCOPETYPE = 'OPENADC'
PLATFORM = 'CW308_SAM4S'
```

Our first baseboard was the CW308 UFO board, so many target platforms start with **CW308_**. With the Husky we planned the new prefix of **CW312_**, but some targets still use CW308 if they were originally designed for the CW308 (even if they are in CW312 form factor).

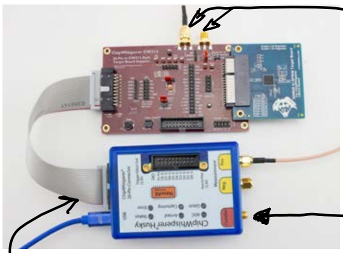
We suggest starting with the **SCA101** course first, as it includes some ChipWhisperer basics! Courses and labs are generally designed to be stand-alone, however, so you can jump around as things interest you.

Getting Started

Physically, you'll connect the ChipWhisperer-Husky to the CW313 board. Into the CW313 board you plug the target – we suggest starting with the SAM4S target board, which is what we use to validate the labs.

Before the ChipWhisperer-Husky, most ChipWhisperer hardware used a STM32F3 target as a default. Due to supply chain challenges the STM32F wasn't available when we were doing production of the Husky.

Your setup should look like this:

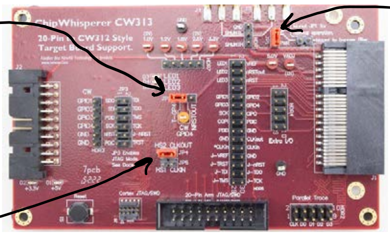


You can use either of these SMA connectors on the CW313.

The **Crowbar** output is connected only for voltage glitching labs.

Ensure you use the 20-pin connector on the side, not the one on the front of the ChipWhisperer-Husky.

The default jumpers on the CW313 should be setup like this:



JP2 Left 2 Pins Shorted

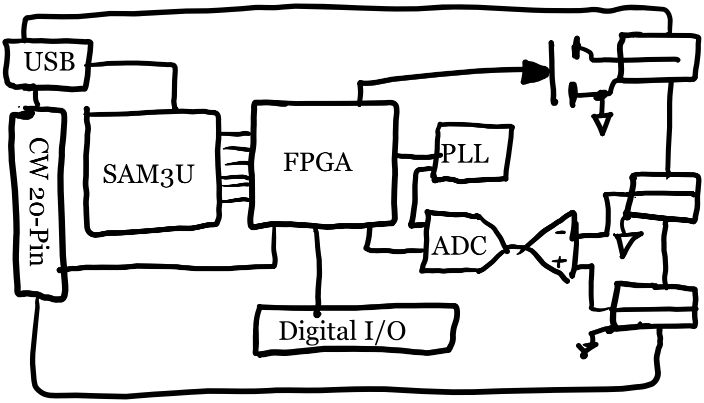
JP4 Shorted

JP1 Shorted

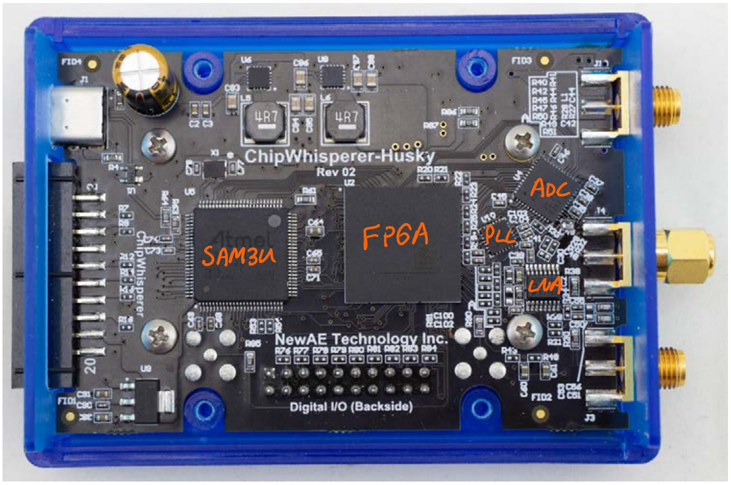
The SAM4S microcontroller is programmed using a serial protocol, as it includes a *hardware bootloader*. Sometimes you can trigger the "erase" by accident when removing and inserting the board. If your target stops responding, try reprograming it (one of the steps in each lab).

CW-Husky Hardware

The main pieces of the ChipWhisperer-Husky are an FPGA (Artix A35), Microcontroller (SAM3U), PLL clock chip, ADC, and variable gain amplifier:

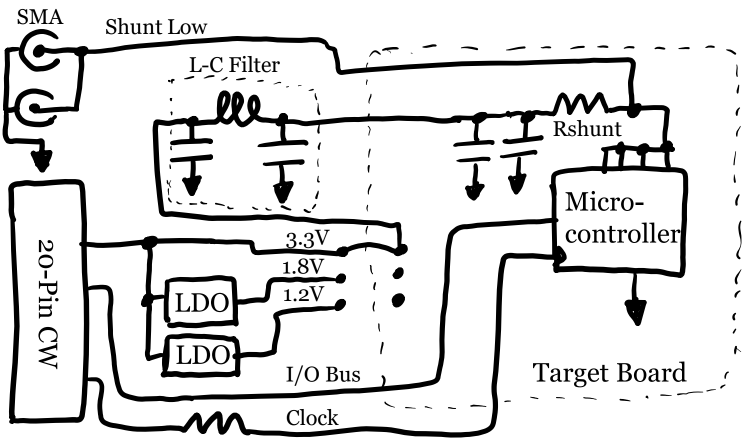


You can see most of them in this photo of the PCB itself:



CW313 & Target Hardware

The CW313 board is a simple interposer to connect the ChipWhisperer 20-pin connector to a target board:



The CW313 includes various useful headers to simplify this, along with additional voltage regulators to generate 1.8V and 1.2V (common core supply voltages). It also includes a L-C filter which the target board can use to filter the power supply.

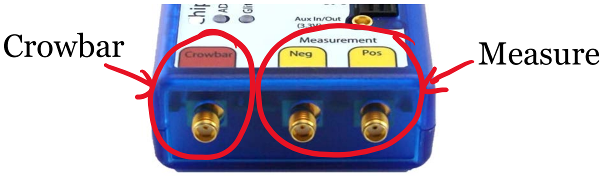
The actual shunt resistor is located on the target board, and normally some additional filtering will be present on the target board itself. The two SMA connectors on the CW313 connect to the same point. There are two of them to allow you to perform voltage glitching & power analysis at the same time.

The specifics of which I/O pins are connected will vary with target boards. Some have more I/O than others, and many boards have different programming interfaces.

The CW313 does not perform level shifting. All NewAE targets use a 3.3V I/O voltage level, even if the core voltage is lower.

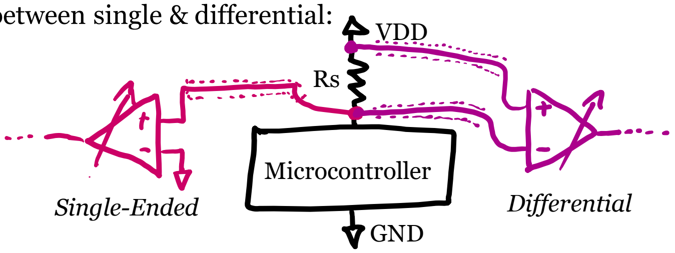
Connectivity – Analog & Glitching

The ChipWhisperer-Husky has three SMA connectors on the front-end:

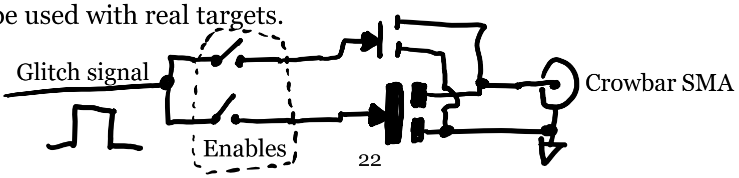


The "measure" connectors route to the differential variable gain amplifier (VGA). They are AC-coupled in hardware – you can change this to DC coupling with a solder mod, but this has very limited DC range and needs additional signal conditioning.

Most of the time we keep a **shorting cap** on the negative input, and just use the positive input in a single-ended mode. The single-end mode is more robust against setup errors, so we use this when possible. Here is the setup difference between single & differential:

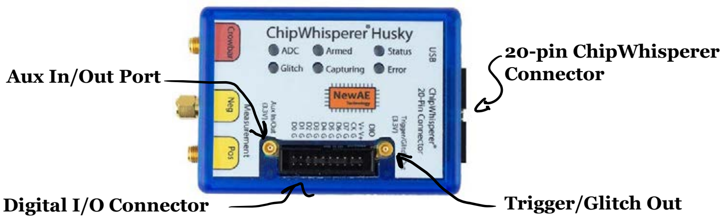


The "crowbar" connector connects to two MOSFETs (electronic switches) which can be used as a **crowbar** for **voltage glitching**. The two MOSFETs have different strengths. The "low-power" MOSFET has a quicker response but is less powerful, and it works well with devices with a built-in shunt resistor (such as any of our targets). The "high-power" MOSFET is slower, but more powerful and can often be used with real targets.



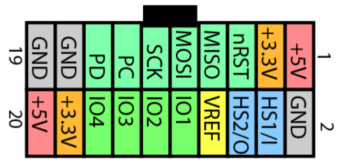
Connectivity - Digital

There are four main digital I/O ports you can use:



All I/O Lines are 3.3V Voltage Level

The 20-pin ChipWhisperer connector is used on most NewAE products, and provides I/O to the target, along with clock & power:

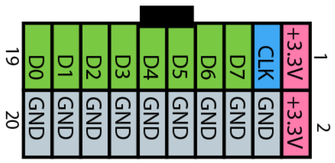


Connector View

Pin	Standard Usage
IO1	Serial RX or TX
IO2	Serial RX or TX
IO4	Trigger Input
HS2	Clock Output (to Target)
+3.3V	Target Power

+5V & VREF are not internally connected on the Husky.

The 20-pin Digital I/O connector can be used as a logic analyzer input or as additional I/O for various modes. The pinout is compatible with a 20-pin JTAG connector, and can be used with “MPSSE mode” to provide OpenOCD support.



Connector View

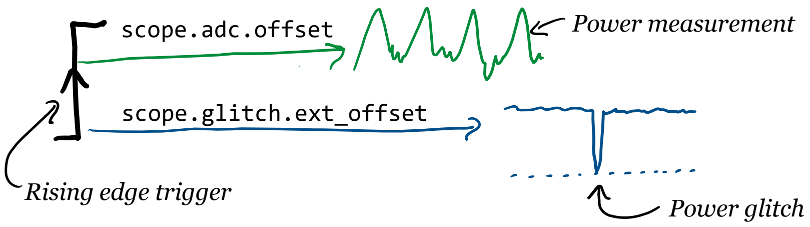
Pin	JTAG / SWD Mode
D7	TDI
D6	TMS / SWDIO
D5	TCK / SWCLK
D3	TDO
+3.3V	Constant +3.3V Output

+3.3V on this connector is not switched with the target power.

The two MCX connectors provide logic-level input or outputs and must be configured in the software.

Triggering

The ChipWhisperer-Husky uses triggering for both the power analysis and fault injection. You'll configure a trigger source, and this can then have a configurable delay before both a power analysis and fault injection event.

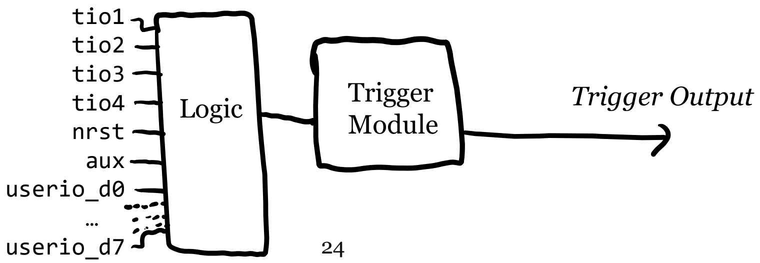


The simple rising edge trigger is used by default on all the labs, where the target firmware is configured to generate a trigger on an I/O pin. This I/O pin is typically connected to the ChipWhisperer "GPIO4" pin on the 20-pin connector.

You can route other pins to the trigger logic, GPIO4 is just what we usually use. You might use the nRST signal or a serial line (GPIO1 or GPIO2) instead. This is configured with the `scope.trigger.triggers` parameter.

```
scope.trigger.triggers = "tio4" #Trigger on GPIO4
scope.trigger.triggers = "nrst" #Trigger on nRST pin
# Use logical AND of AUX connector AND GPIO1:
scope.trigger.triggers = "aux AND tio1"
```

You can combine the inputs using basic logic as well (see docs). These inputs are fed to one of several trigger modules.



Triggering

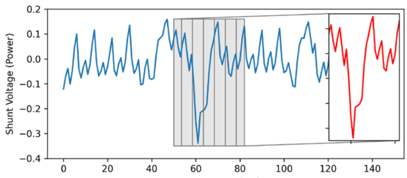
ChipWhisperer-Husky has several advanced trigger sources beyond the standard edge events, these are:

Edge Count: This counts the total number of edges. This can be helpful to trigger on events such as a certain number of SPI transactions, or even arbitrary digital protocols.

UART Trigger: Triggering on serial (UART) data ca be used to trigger on either active communications with the target, or passive output such as a boot message.

Analog ADC Level: Triggering on an ADC level is similar to a standard oscilloscope trigger.

Analog Pattern Match (SAD): The SAD match triggers on a waveform in real-time. This can be used to match certain characteristic functions, for example.

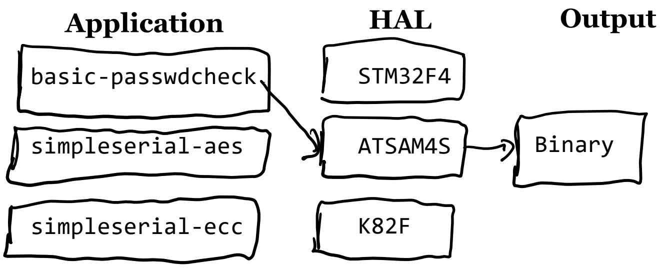


Arm ETM Trace (TraceWhisperer):
The TraceWhisperer core can understand (decode) the Arm ETM trace format. This allows you to trigger on trace events, such as hitting a certain program counter value.

You can add your own module into the FPGA code if you need to trigger on new protocols. See the "modifying the FPGA" section on page 32.

Target Firmware & HAL

The ChipWhisperer repository includes a variety of example applications. These applications typically run some form of cryptographic code, often with additional instrumentation. The applications are built with a Hardware Abstraction Layer (HAL) which we've created and verified for many different processors:



Many of the applications use the '**SimpleSerial**' protocol. This protocol started as a basic way of triggering cryptographic functions on a target. It now includes a *binary* mode (**SimpleSerialV2**) which allows very fast communication.

We include the `SimpleSerial` (and `SimpleSerialV2`) Python interface, so you can quickly build new applications for working with arbitrary blocks inside the target.

To add a new device which you *don't* find a HAL already at `chipwhisperer/hardware/victims/firmware/hal`, you only need to provide these functions:

```
void platform_init(void); // Setup clock etc
void init_uart(void); //Setup UART
void trigger_setup(void); // Setup trigger pin (I/O)
void trigger_low(void); // Set I/O pin low
void trigger_high(void); // Set I/O pin high
void putch(void); //Send single char, blocking
void getch(void); //Receive single char, blocking
```

You do *not* need working interrupts for the HAL to function.

Target Firmware & HAL

Some useful existing applications you can find in `chipwhisperer/hardware/victims/firmware`:

basic-passwdcheck: Shows a simple application that *doesn't* use SimpleSerial. This is closer to a real-life prompt.

simpleserial-aes: Runs AES on the target. You can choose various cryptographic libraries, including using the hardware accelerator where available.

simpleserial-rsa: Runs RSA on the target, this along with `simpleserial-ecc` show how you can test asymmetric algorithms.

simpleserial-trace: Shows how to enable the parallel trace, which can be read or triggered on with the TraceWhisperer functionality.

simpleserial-glitch: Runs various glitch demos. In particular the glitch_loop() function is useful as a calibration when testing new targets.

glitch-simple: A version of the glitch code without SimpleSerial. If you enable the glitch loop (`glitch_infinite()`) you get a simple output you can try to corrupt with a glitch.

```
unsigned int k = 0;
//Declared volatile to avoid optimizing away loop.
//This also adds lots of SRAM access
volatile uint16_t i, j, cnt;
while(1){
    cnt = 0;
    trigger_high();
    trigger_low();
    for(i=0; i<200; i++){
        for(j=0; j<200; j++){cnt++;}
    }
    uart_printf("%u %u %u %u\n", cnt, i, j, k++);
}
```

Adding a New Target

The following assumes you will build a new target for the ChipWhisperer. Doing so will require:

1. Understanding the Clock
2. Understanding the Power Source
3. Finding a Trigger & IO
4. Calibration or Experimentation

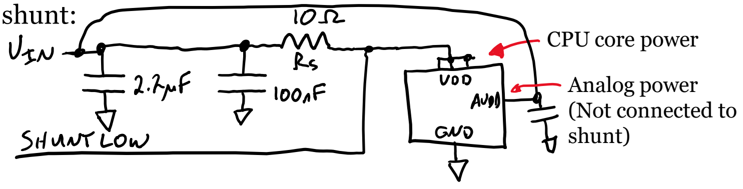
Clock

We typically want to provide some relationship between the ChipWhisperer clock & the target clock. Normally we'll feed a clock into the clock input pin, but on some devices they have no clock input. You may be able to enable a clock output, or even enable something like a very fast PWM signal that ChipWhisperer can use to synchronize to the internal clock.

Power Source

You'll want to add a shunt resistor on the power supply to the core logic. Sometimes it's not clear which power pin(s) this is, so you may want to make the board with various jumpers.

The shunt value varies with device complexity; more complex devices (such as large FPGAs) may need small shunt values. We typically start with around 10 Ohms. Be sure to include filtering (decoupling capacitors) on the high side of the shunt:



The design files for all our individual target boards are available & make a good reference for various filtering options.

Adding a New Target

Trigger & IO

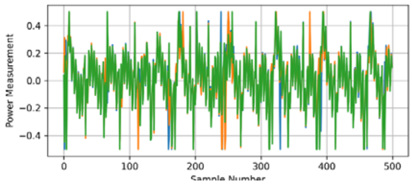
Building the sample code will require you to have a trigger, which simply requires access to the GPIO pins and being able to toggle them.

The default firmware also includes a working UART. The ChipWhisperer-default 7.3728MHz frequency can be used to generate standard baud rates from even very old UARTs.

ChipWhisperer can work with arbitrary baud rates and clocks. A trick we sometimes use is simply to write code to output fixed data from the UART, and measure that baud rate. You can then use the “natural baud rate” of the device; it doesn’t have to be a standard baud rate.

Calibration & Experimentation

With your new target, try performing some basic power analysis. You should get traces which overlap and “appear” low-noise. If not, try running an AES-128 attack, as you may find the signal is still present even with the noise there. A TVLA test (see **SCA 203**) is an even better technique.



Example trace from a STM32F2 that is a little noisier than other examples.

For glitching, we normally start with running the glitch loop test using voltage glitching. At first you have no idea how to tune this. Start by making sure you can reset the device with glitching (this is too far!), then back off until you see only occasional resets. Hopefully, you’ll get some glitches in the calibration code at that point.

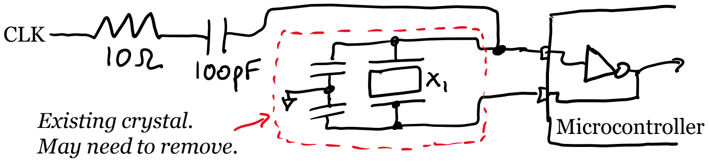
See Fault 101 Lab 2.1 for this sort of calibration.

Connecting an Existing Board

If you have an existing board (be it development board or a real target), the steps are similar to those in "adding a new target". The difference is that we have additional constraints about what we can do. We almost always build a custom target board for a new chip we are working with before attempting to modify an existing board, as there are many "unknown unknowns" when working with a new board and chip together.

Clock

We typically want to provide some relationship between the ChipWhisperer clock and the target clock. This requires us to control the clock. If your device has a crystal already, note that this typically means you can simply "overpower" it or force it to phase-lock to your own clock:



The resistor and capacitor may or may not be required. Note you need to feed this into the input pin, which often won't be specified (even in the datasheet). In which case you can experiment to determine this.

Power

For power analysis, it's much easier to have a very clean power source. You may want to feed an external supply in, and adding filtering on the physical board is very helpful.

See the "Hardware Hacking Handbook", Chapter 11 for more discussion of this type of setup.

Connecting an Existing Board

Finding a Trigger

You'll need to trigger on the device's operations. Some common triggers you might consider include:

- Triggering on the reset pin going high
    - Check if the device has a *reset out*, which can be an even more accurate source of reset.
    - Devices may have internal delays from reset pin to boot processing.
- Triggering on UART data
    - Triggering on something the device sends to you is best, as it is more likely to be synchronized to internal logic.
- Triggering on some external I/O (such as an activity LED)

The analog trigger module (SAD) can be helpful as well.

Calibration & Exploration

Attempting to directly "attack" some specific function is rarely successful. Instead, first decide on some simple functionality you can use to confirm that your power analysis or fault injection is working.

For example, if you're working with a board and you want to confirm your glitch attack will be successful, check the following:

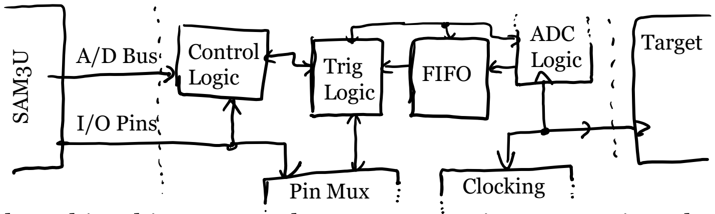
- Can you corrupt a start-up prompt without it rebooting?
- Can you corrupt a checksum or CRC operation? They are typically slow (easy to "hit") and have useful error messages.

For power analysis, you might consider:

- Can I observe the device boot and see various operations?
- Can I force the device into different states (secure and insecure) and see *where* in time that switch happened?

ADV: Modifying the FPGA

The Field Programmable Gate Array (FPGA) provides all the high-speed data processing for the ChipWhisperer-Husky.

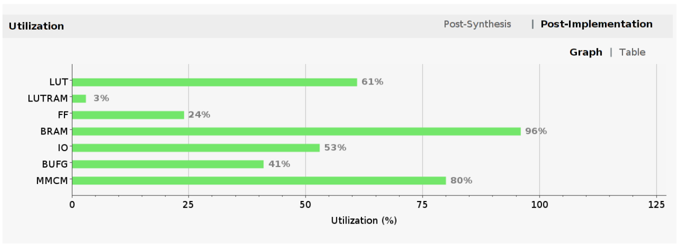


The ChipWhisperer-Husky-FPGA repository contains the FPGA code. There is a simulation and test regression environment to validate correct functionality of the existing design. Implementation is done with Vivado.

Once you have an updated FPGA bitfile, further automated testing can be done with the `test_husky.py` script; this runs tests on the actual hardware for things that are harder to verify in simulation.

The stock FPGA bitfile is very full so if you wish to make substantial changes you may need to disable some features.

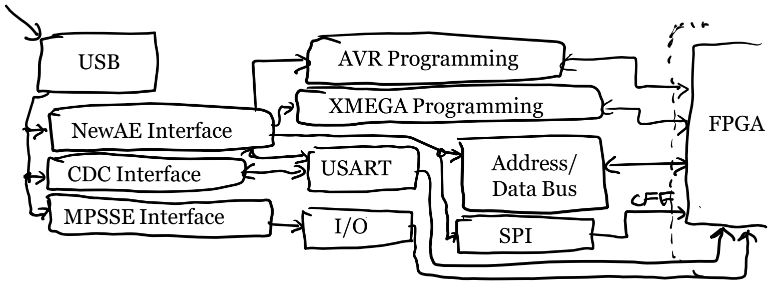
If you need more BRAMs (which are almost fully utilized), you can define TINYFIFO to drastically reduce the ADC sample storage size.



https://github.com/newaetech/chipwhisperer-husky-fpga

ADV: Modifying the SAM3U

In addition to the FPGA, the Husky has a SAM3U microcontroller. Its main purpose is to handle USB communication, program and communicate with the FPGA, implement target programming, and provide UART communication with target boards.



The SAM3U firmware is available from the ChipWhisperer-Husky repository. The same compilers are required as used for ChipWhisperer targets, so you should already have the toolchain setup.

To build firmware, navigate to `chipwhisperer-husky/ChipWhisperer-Husky/src`, make sure you've got the `naeusb` submodule up to date, and run:

```
$ make -j
```

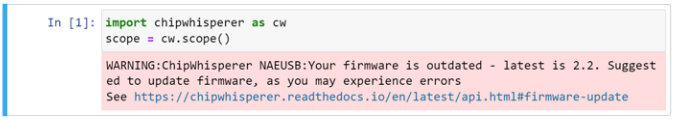
We use the naeusb repository to hold common build systems. To get familiar with the layout of the repo and the build system, check out `naeusb/README.md` `naeusb/naeusb.md`

You can reprogram the SAM3U using the built-in bootloader system:

```
import chipwhisperer as cw
cw.program_sam_firmware(fw_path='SAM3U.bin')
```

Updating SAM3U Firmware

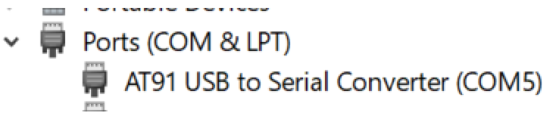
You might get a message about a firmware update for the ChipWhisperer-Husky when you connect. You'll see something like this:



```
In [1]: import chipwhisperer as cw
        scope = cw.scope()

WARNING:ChipWhisperer NAEUSB:Your firmware is outdated - latest is 2.2. Suggest
ed to update firmware, as you may experience errors
See https://chipwhisperer.readthedocs.io/en/latest/api.html#firmware-update
```

See the online documentation for the procedure. If the firmware update stalls or fails, the device will often revert to the hardware bootloader. You can tell this has happened since the device appears as a serial port:



Ports (COM & LPT)
AT91 USB to Serial Converter (COM5)

In very rare cases you need to force the bootloader to re-enter. Depending on your hardware revision, you either need to short a pad on the PCB, or you can use a pin to press a recessed button. See the online documentation for this.

Note the hardware bootloader in the SAM3U cannot be erased. There is no risk of a failed firmware update permanently bricking your ChipWhisperer-Husky.

There is no firmware update needed for the FPGA, as the FPGA is always loaded when the ChipWhisperer software connects. The FPGA bitstream is always the most up-to-date.

Common Problems

Problem: Device stops being detected in Jupyter

First, try unplugging & replugging it. This performs two 'fixes': (1) ensures all connections on the computer are closed, and (2) does a hard reset of the microcontroller. When switching Jupyter notebooks, run scope.dis() in the "old" one to close the connection and to avoid (1).

If that isn't successful, check that the driver has loaded (Windows) or permissions are OK.

Problem: No device in `Device Manager` or `dmesg`

Check that the status LED is blinking (indicating USB is OK). If light is solid, try plugging into another computer. If this works, try a different port on your computer, or try removing other attached USB devices that could be conflicting. See online documentation for more USB troubleshooting.

Problem: Device LEDs are rapidly blinking on & off

This is typically a power or sleep problem; the computer is attempting to power it up and then shutting it down right away. Try a different port, or if you're using a USB hub try directly connecting to your USB port.

Problem: Device is dead after a firmware update

See online documentation and page to the left. You may need to manually trigger the bootloader, which is always present in the device.

Problem: SAM4S Target Not Responding

The SAM4S seems to sometimes need a power cycle (but sometimes works better without it?). Try modifying the reset code to toggle `scope.io.target_pwr` (or remove it if already there).

Getting Help & More

Learning About Security

You might have more fundamental questions about this whole thing! Beyond the ChipWhisperer-Jupyter courses which are freely available, other resources include:

- **The Hardware Hacking Handbook** by Jasper van Woudenberg & Colin O'Flynn, published by No Starch Press.
- **ChipWhisperer Training** (ChipWhisperer.io & in-person)
- General Books & Material on embedded development will be helpful in understanding how lots of things work! Much of what we do in security is really related to embedded engineering.

NewAE Specific Assistance

Hardware Documentation includes documentation for various hardware and target boards, including the ChipWhisperer-Husky, CW313, and the included targets.

https://rtfm.newae.com

ChipWhisperer Documentation is built from the ChipWhisperer repository. It includes lots of getting started advice, along with how to use the software, and the full API documentation that is also available through the Python help() feature.

https://chipwhisperer.readthedocs.io

NewAE Forum is our preferred support method. Answers here can be seen by the wider community.

https://forum.newae.com

Issues on GitHub projects are used for specific problems (bugs); raise an issue to ensure they aren't dropped. Please don't use issues for general questions.

NewAE Discord is a chat room which also has some support offered. For specific questions, the Forum is typically easier, as we can provide inline images and access the thread at a later date.

NewAE Support Tickets are used when you have hardware problems or similar. If you need a hardware replacement, you will need to use our support ticket system.

https://support.newae.com

Links to Sources & Repos

ChipWhisperer Repositories

You might notice some of the resources are a bit spread out! Here are some of the more important resources you don't want to miss:

ChipWhisperer Repository

This (overloaded) codebase includes many of our older design files including hardware sources, FPGA examples for the CW305, the target firmware and HAL, and then the entire ChipWhisperer software stack.

https://github.com/newaetech/chipwhisperer

ChipWhisperer-Jupyter Repository

This repository includes the "Jupyter notebooks" organized into complete courses. This gets installed when you install ChipWhisperer by default.

https://github.com/newaetech/chipwhisperer-jupyter

ChipWhisperer-Husky Repository

This repository includes the SAM3U firmware and other hardware information. The FPGA design is a submodule of this repository.

https://github.com/newaetech/chipwhisperer-husky

ChipWhisperer CW308/CW312 Target Repository

This repository includes the source files for all target boards for the CW308 and CW313. This is a useful reference if designing your own targets.

https://github.com/newaetech/chipwhisperer-target-cw308t

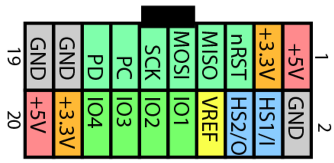
ChipWhisperer TraceWhisperer Repository

TraceWhisperer is our trace decode logic, used for both debug and triggering capability.

https://github.com/newaetech/tracewhisperer

Quick Reference for Connectors

20-Pin ChipWhisperer Connector (Side)

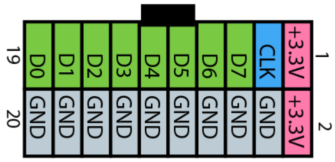


Connector View

Pin	Standard Usage
IO1	Serial RX or TX
IO2	Serial RX or TX
IO4	Trigger Input
HS2	Clock Output (to Target)
+3.3V	Target Power

The "+3.3V" is switched on/off with `scope.io.target_pwr` The I/O pins will go High-Z when target power is switched off to prevent powering the target.

20-Pin Digital I/O Connector (Front)



Connector View

Pin	JTAG / SWD Mode
D7	TDI
D6	TMS / SWDIO
D5	TCK / SWCLK
D3	TDO
+3.3V	Constant +3.3V Output

+3.3V on this connector is **not** switched with the target power.

See page 23 for connector details.

Using JTAG (MPSSE) Mode

ChipWhisperer can operate as a FTDI-compatible debug probe for usage with OpenOCD (we call it 'MPSSE mode'). You enable this via Python, and can choose to use either the ChipWhisperer connector pins OR the Digital I/O pins with this mode.

USB limitations mean that USB-Serial mode & MPSSE mode cannot be used together. On Windows only a single process can talk to the USB device, meaning you need to disconnect in Python when using OpenOCD (& vis-versa) on Windows.

LIMITED WARRANTY AND LIMITATION OF LIABILITY
Each NewAE Technology Inc product is warranted to be free from defects in material and workmanship under normal use and service. The warranty period is one year and begins on the date of shipment. This warranty extends only to the original buyer or end-user customer of a NewAE Technology Inc authorized reseller, and does not apply to probes, exposed circuit boards, fault injection targets, or to any product which, in NewAE Technology Inc's opinion, has been misused, altered, neglected, contaminated, or damaged by accident or abnormal conditions of operation or handling (including failing to observe required ESD handling procedures).

Authorized resellers shall extend this warranty on new and unused products to end-user customers only but have no authority to extend a greater or different warranty on behalf of NewAE Technology Inc. NewAE Technology Inc.'s warranty obligation is limited, at NewAE Technology Inc.'s option, to refund of the purchase price, free of charge repair, or replacement of a defective product which is returned to a NewAE Technology Inc. within the warranty period. To obtain warranty service, contact NewAE Technology Inc.

If NewAE Technology Inc. determines that failure was caused by neglect, mis-use, contamination, alteration, accident, or abnormal condition of operation or handling, including failures caused by use outside the product's specified rating, or normal wear and tear of mechanical components, NewAE Technology Inc will provide an estimate of repair costs and obtain authorization before commencing the work.

THIS WARRANTY IS BUYER'S SOLE AND EXCLUSIVE REMEDY AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. NEWAE TECHNOLOGY INC SHALL NOT BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES OR LOSSES, INCLUDING LOSS OF DATA, ARISING FROM ANY CAUSE OR THEORY.

Since some countries or states do not allow limitation of the term of an im-plied warranty, or exclusion or limitation of incidental or consequential damages, the limitations and exclusions of this warranty may not apply to every buyer. If any provision of this Warranty is held invalid or unenforceable by a court or other decision-maker of competent jurisdiction, such holding will not affect the validity or enforceability of any other provision.

NewAE Technology Inc.
127 Joseph Zatzman Dr
Dartmouth, NS. Canada

sales@newae.com
1-888-GLITCHY (US/CA)
or
+1 902 800 8880

ChipWhisperer® Husky User Manual

What's this? A manual on how to use side-channel analysis to understand what your husky is talking about? Whoa that would be awesome – what a great idea!

Unfortunately, this is just a book about a hardware tool, the ChipWhisperer-Husky. We see where the confusion comes from.

We'll get started on that first one though; that is probably going to be a big seller!

In the meantime, you can hopefully use this book to understand what the heck power analysis and fault injection is, and how you can start learning about them using a low-cost† tool.

† Relative to the cost of everything else in 2022.

About the Author

Bergen is The Assistant (to the) QA Manager at NewAE Technology Inc. At NewAE, she has led the work-life balance program by example. She is originally from the Canadian territory of Nunavut, and always dreamed of competing in dogsled races. She can be seen practicing her dogsled-pull many times throughout her daily walks and has never let her short stature get in the way of her training regime. She lives in Halifax, NS, Canada.

Printed in Nova Scotia, Canada.

NewAE Technology Inc.
Self-Publishing Division.
127 Joseph Zatzman Dr.
Dartmouth, NS. B3B 1W1.
Canada
Phone: 1-888-GLITCHY

$19.99 CAD / £14.99 GBP / $99.99 USD

ISBN: 978-1-9994176-1-1

9 781999 417611