POWER ANALYSIS FOR CHEAPSKATES

The White Paper. by Colin O'Flynn

Contents

Some Front Matter, Copyright, and Referencing Information

If you wish to reference this White Paper, you can use the official title of *Power Analysis for Cheapskates: The Whitepaper*. Depending where you found out about this can affect the choice of which proceedings it was published in, for example Blackhat USA 2013. Finally you should always send people to www.chipwhisperer.com where they can find the latest version of this paper and all the information within it.

Introduction & Motivation

This white paper isn't really going to cover the theory behind Side Channel Analysis (SCA). There are lots of great references, so instead I will just point you to this.

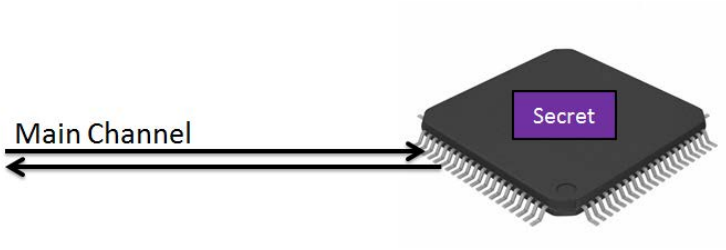
First, you could read the original paper in this field, which even if you aren't too mathematical should make sense [1]. There is also a nice book [2], all of which are discussed in more detail in the *Where to Go From Here* section of this White Paper.

Check for updates to this White Paper at www.newae.com/blackhat , or otherwise see if there is other interesting things on my website. The collection of my tools in this area has been called ChipWhisperer, so you can skip directly to side-channel things by going to www.chipwhisperer.com . There is even a mailing list to get updates and discuss the tools.

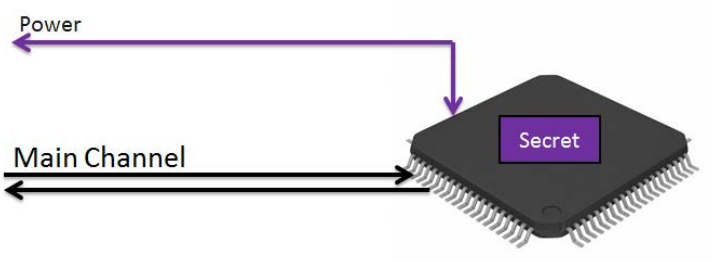
This White Paper contains only a few of the many great references in the area of Side Channel Analysis. Spend some time on Google Scholar to start exploring other interesting articles.

What is Power Analysis? What is the Side Channel?

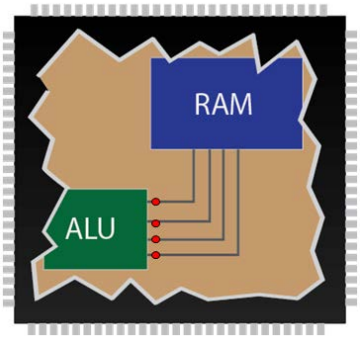
Consider a crypto device. It is trying to store a secret, and has some method of accessing it:



But that is far from the *only* way we can get information from the device. Just as nosey neighbours infer who is being unfaithful by goings and leavings of cars, we can gain some information by looking at other details of the cryptographic system. In particular we look at the 'power' side-channel. So our system now looks like this:

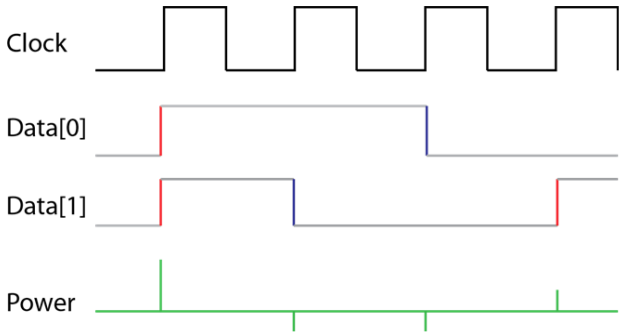


How does this system work? Well let's consider a digital block. There will be lots of stuff inside there, but we are mainly concerned with the fact that there are several blocks with bus lines interconnecting them. Here is a simplified figure of a digital system:



Well what happens when we send data across the bus? Power is transferred onto the bus from the VCC and GND lines. When we switch the states of the line it takes a charge – this means it takes power every time we want to change the state of the line.

With digital electronics, this state changes relative to the clock. Thus we could insert a resistive shunt into the power line to measure the current flowing into the chip. If we looked at two bits on the data-bus, we might expect something like this:



On each clock transition, it takes power to change the state of the line. If two lines change at once it takes more power than if one lines changes at a time. What does this mean? We actually can learn something about the data on the bus.

Depending on the architecture of system, we could learn one of two things:

- The number of lines that switch state
- The number of one's on the bus

The first one seems probably most intuitive – in the example I gave, there is a spike when lines change state. On each clock iteration you can see how many lines 'switched'. The number of bits changing is called the *hamming distance* (HD). This typically exists in hardware implementations – that is specific hardware doing cryptographic operations.

The second point exists because many architectures put the bus to a pre-charge state before actually loading data onto the bus. This pre-charge state puts about half the full charge on the bus, meaning it now takes equal energy to switch to either zero or one. This pre-charge is also an artifact of the bus having multiple drivers – the next data pulse could come from one of several sources, so you need this intermediate value when switching driver locations. You typically find that microcontrollers doing software cryptographic operations leak the *hamming weight* (HW) of the data bus.

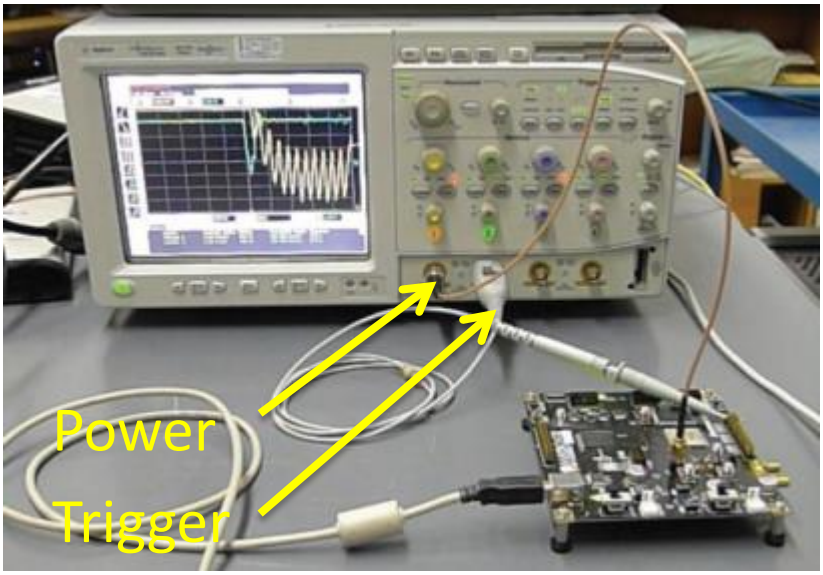
Capturing Traces: Basics & Hardware Requirements

Typical Capture Environment

Looking in papers in this field, most people are using normal oscilloscopes. When attacking hardware implementations, this often means one requires a high-speed oscilloscope. When attacking software implementations one can often 'get away' with low-cost oscilloscopes, such as 100 MSPS devices.

I have a separate paper which goes into more details around the requirements of this oscilloscope. You can see that paper at [3].

A normal environment might look something like this:

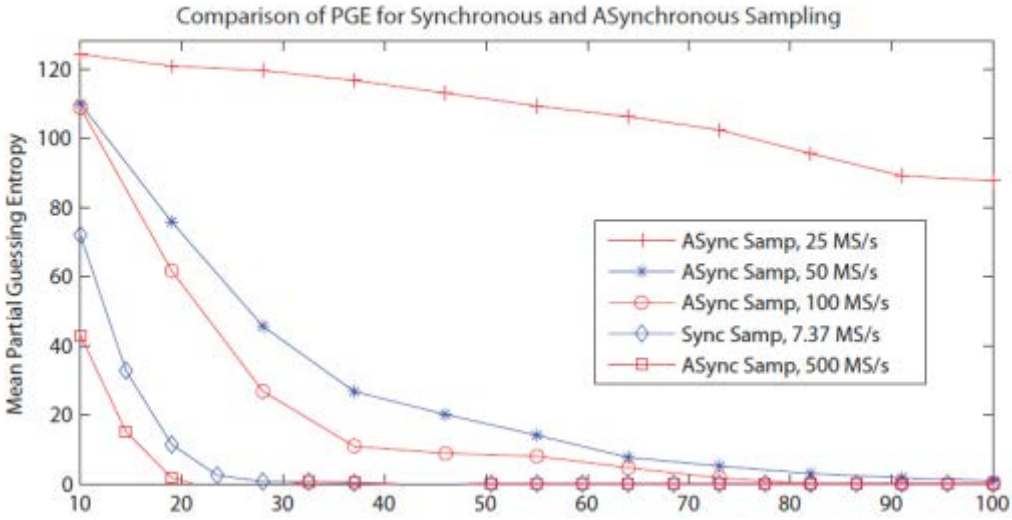


The oscilloscope is controlled by a computer off-screen.

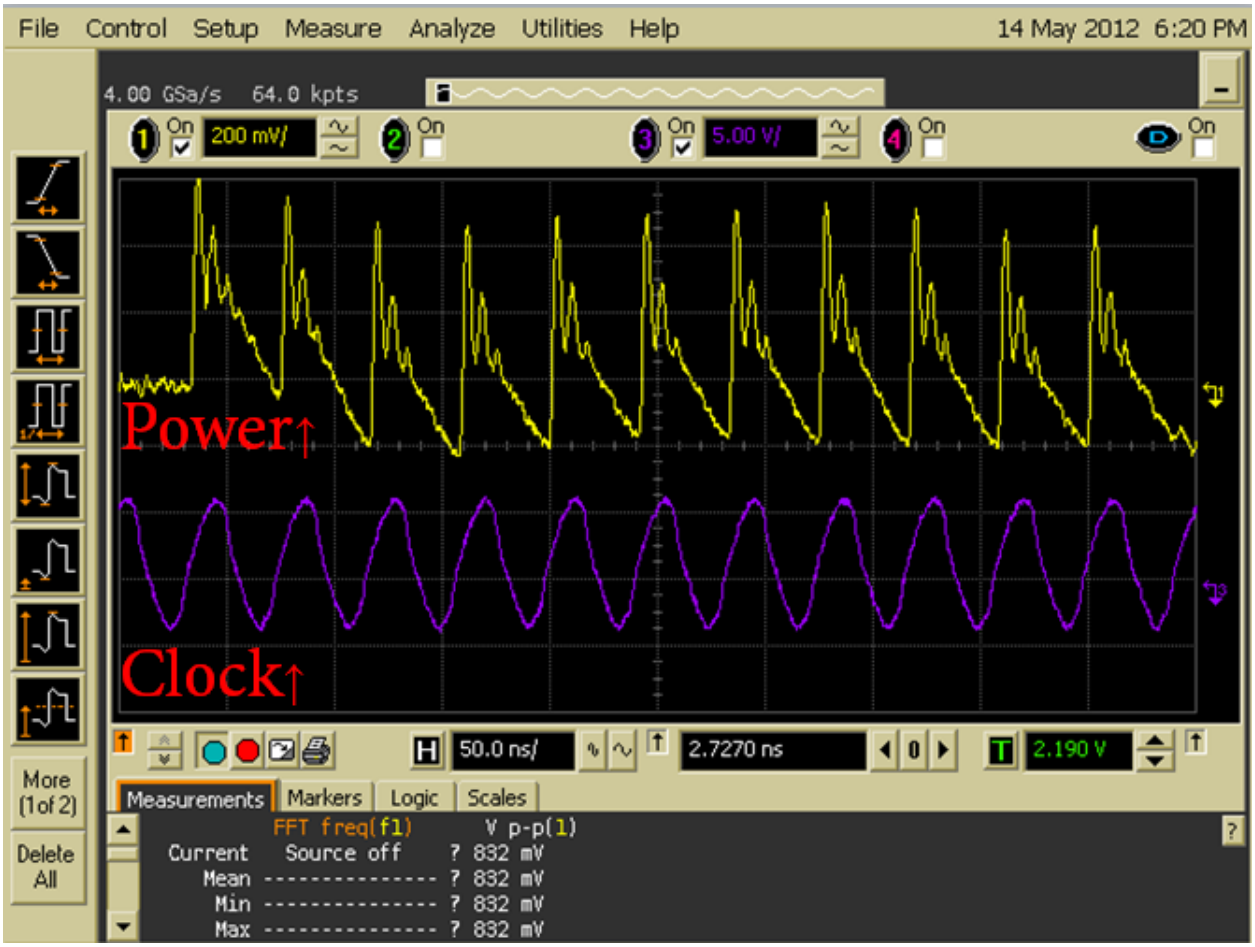
Using Low-Cost Hardware

The phrase "low cost hardware" normally means a cheap oscilloscope. This cheap oscilloscope has a limited sample rate, which will work for attacks at low clock frequencies. But what if you want to attack real implementations running at higher speeds?

As a simple example, consider the following figure. This figure shows the average 'Partial Guessing Entropy' of an attack completed using samples collected with different rates. When the 'Partial Guessing Entropy' goes to zero this means the correct key is known.



You can still use low-cost hardware, provided you carefully consider the sample clock. The following figure shows the capture of a power trace from a SASEBO-GII (FPGA) target. In addition the clock which is driving the SASEBO-GII is provided, which is running at 24 MHz:



If we could sample on the clock edges perfectly, we could collect exactly the number of samples needed. Previous work has used this idea for compressing captured traces to only keep one sample per clock cycle. This means you can capture thousands of traces without wasting a lot of hard drive space. But you still need a good oscilloscope to perform the original capture.

I designed the OpenADC, which takes this even further. It uses the device clock to decide when to sample the power consumption. This means that you don't need post-processing to keep the samples of interest. In addition it drastically reduces the sample rate requirements, meaning very low-cost hardware. To give you an example on the SASEBO-GII running at 24 MHz, sampling rates lower than 250 MSPS had been shown to be useless for SCA. With the OpenADC one can sample at 24 MSPS and still achieve a successful attack, as described in my paper at[3].

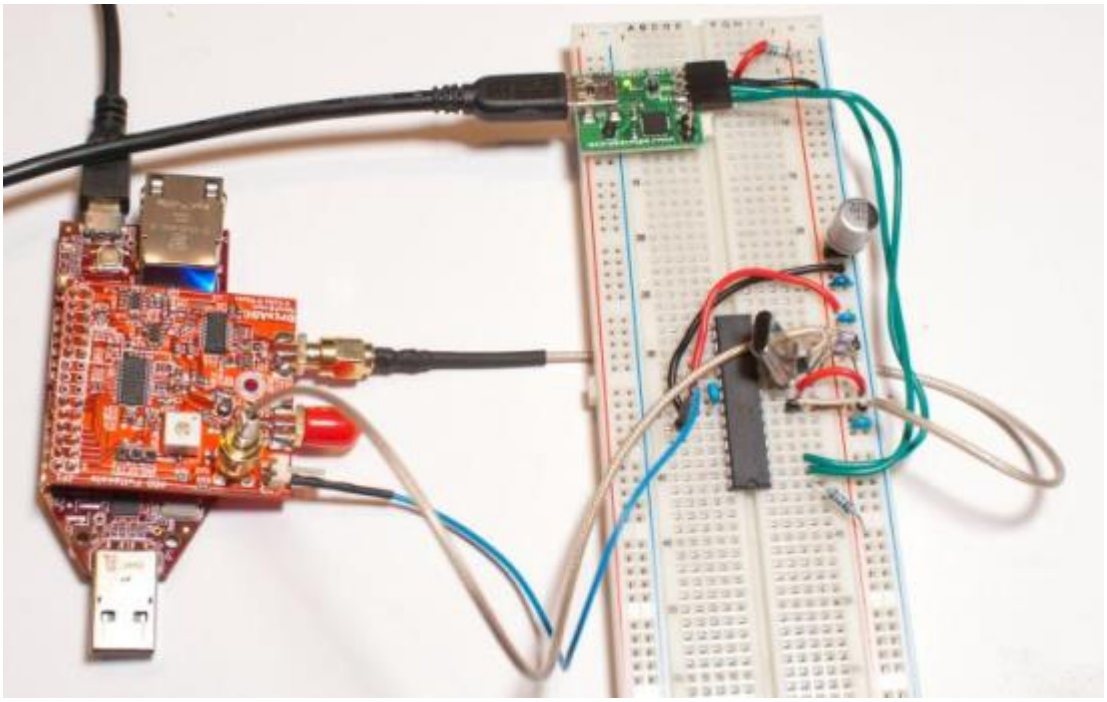
Capturing Traces: An Interlude

Sandwiched in-between the basic & advanced topics, I'm going to give a little bit of a tutorial. If you wish this would let you 'follow along' which may help you learn on your own. But even just reading through this may help you understand

Introduction

First, you'll need to setup/install the capture software. You'll also need a target.

Your target will probably be an AVR microcontroller, which means your bench looks like this:

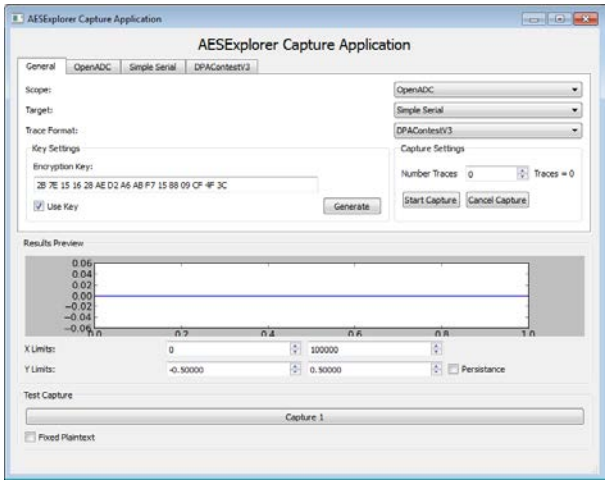


Note the following:

1. The AVR is powered from 3.3V, which is the same voltage level the FPGA I'm using with the OpenADC is running at. The clock & trigger lines of the FPGA are normally **not** 5V tolerant. If using a 5V Arduino you need to convert these voltages down, see details later.
2. I'm passing the clock from the target into the external clock input on the OpenADC.
3. The trigger line is connected to the 'POS' trigger pin on the OpenADC.
4. I'm measuring the voltage across a 75-ohm resistor in the ground pin of the AVR.

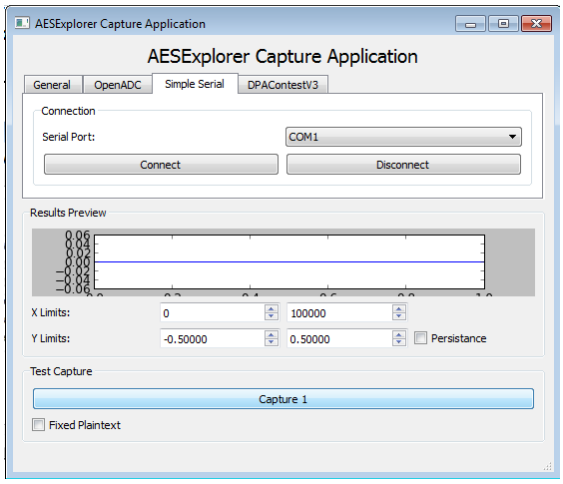
Getting Familiar

Start the aesexplorer-capture application, and your window looks something like this:

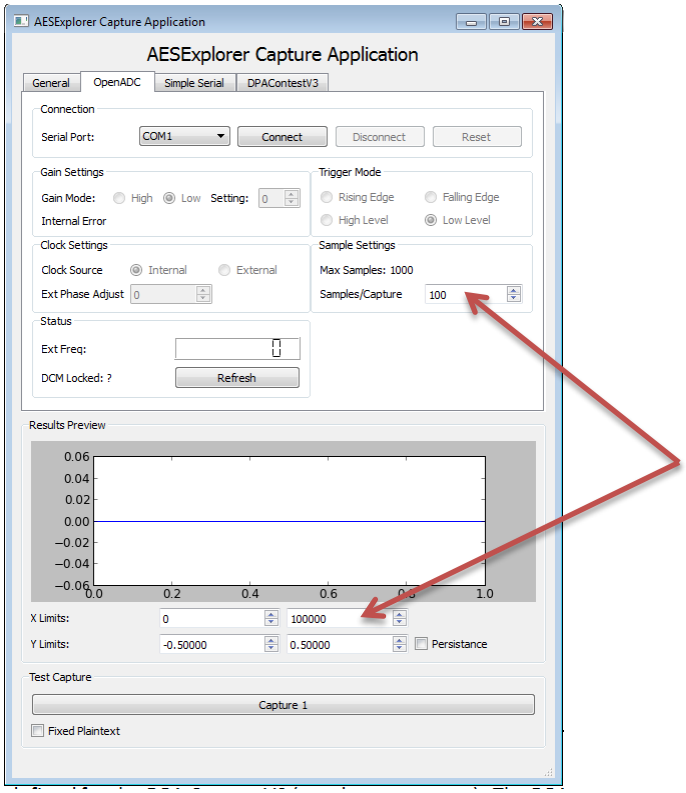


The 'scope' should be set to 'OpenADC'. The 'target' should be set to 'Simple Serial', assuming you are using the targets described here. Finally you have the choice of how traces are stored – the default 'DPAContestV3' is a format defined for the DPA Contest V3 (see dpacontest.org). The DPAContestV3 is very straight-forward, and consists of text files with the saved waveforms & other required data.

Go to the 'Simple Serial' tab, select the proper COM port and hit 'Connect'. If your serial target is attached with a USB-Serial converter, you will need to figure out which one it is.

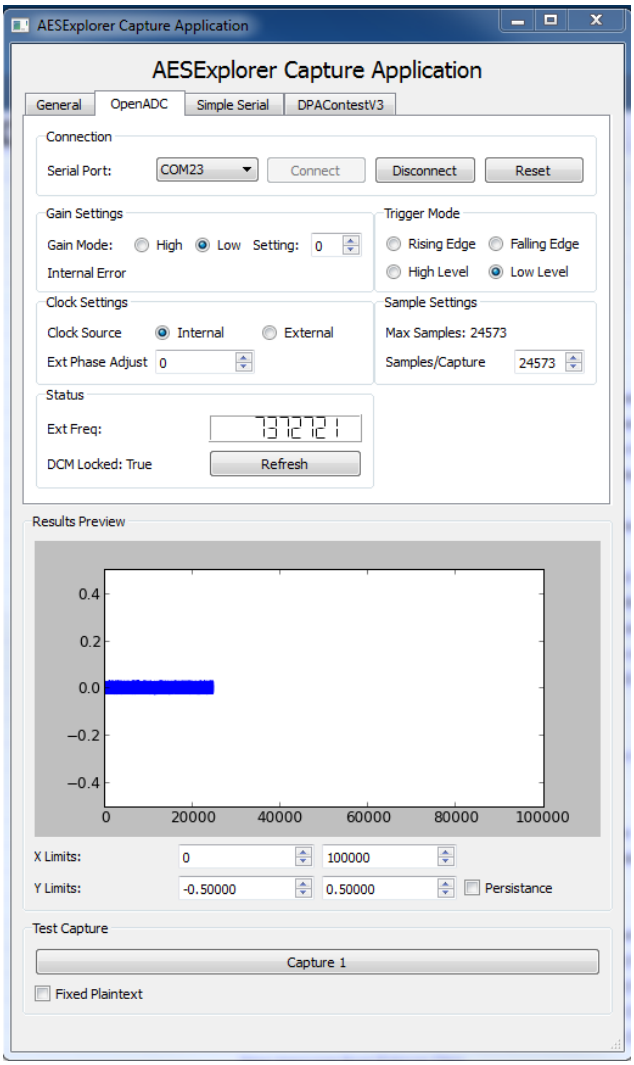


Go to the OpenADC tab, which looks like this:



Select the proper COM port and hit 'Connect', assuming you are using the LX9 microboard with the OpenADC. You will probably want to adjust the 'Samples/Capture' and the 'X Limits' of the graph, those are highlighted in the previous image.

You can hit 'Capture 1' now to try and get a trace, you should see something appear in the 'Results Preview' graph:

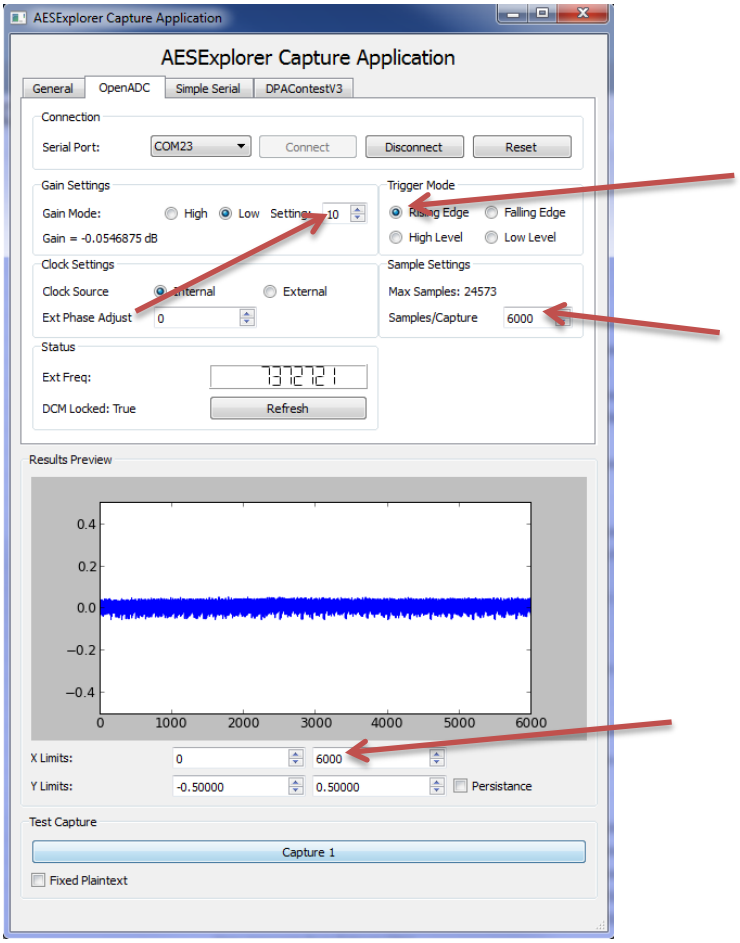


In the 'Status' field you can hit 'Refresh' and see if the external clock is working. You should see the correct frequency here should be around 7372800.

At this point the system isn't properly setup though. You should adjust the following:

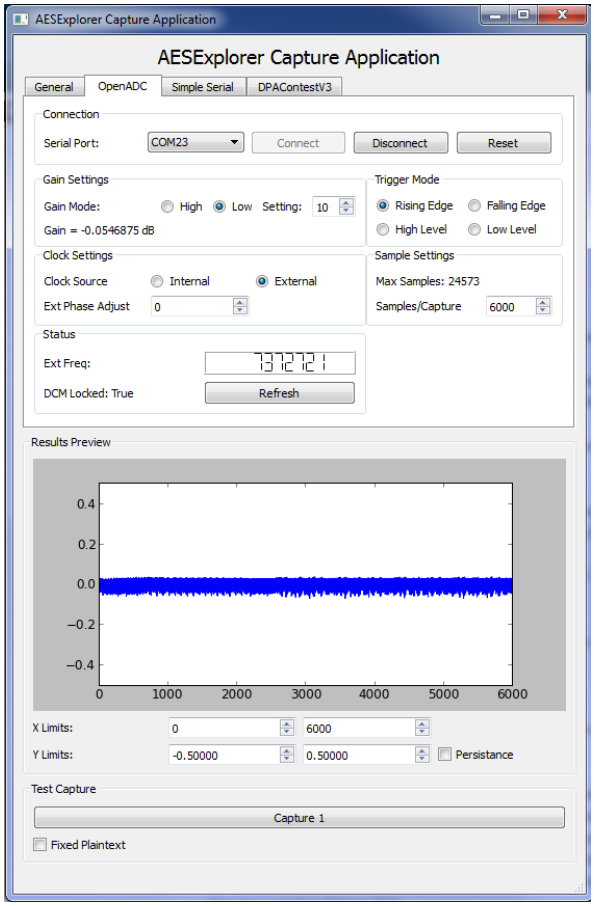
- Set Trigger Mode to 'Rising'
- Turn gain up slightly (say to '10')
- Set Samples/Capture to 6000
- Set Maximum X-Limits to 6000

Hitting 'Capture 1' should now look something like this:



The next thing to check is what happens when you set the sample clock to 'external'. Again see if 'Capture 1' works. Now that the sample clock is running slower and perfectly synchronized, you

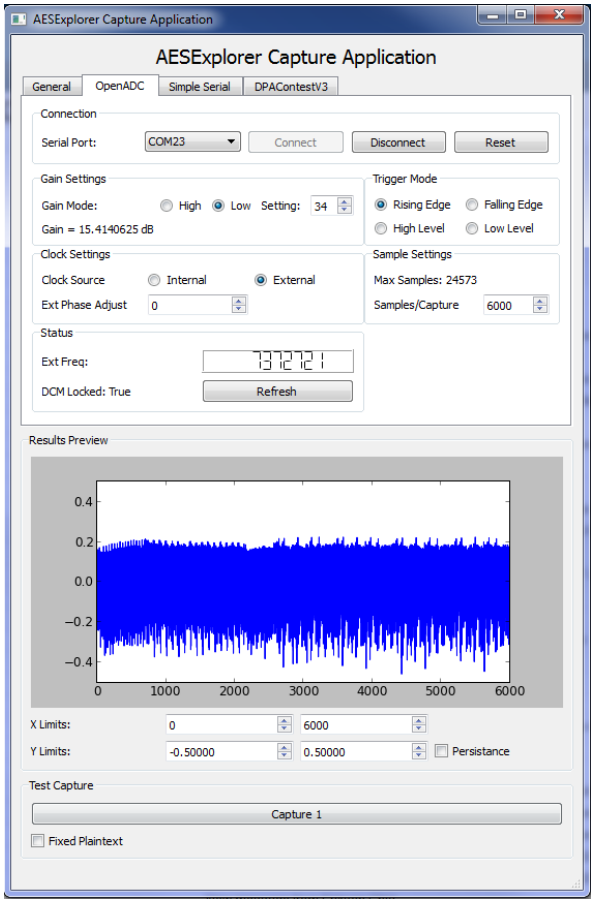
hwill see the signal change:



Ensure clock frequency is correct and 'DCM Locked' says 'True'.

WARNING: Current revision of SW has clock frequency correct for LX9 only. This will be fixed…

Adjust the gain upwards a bit:

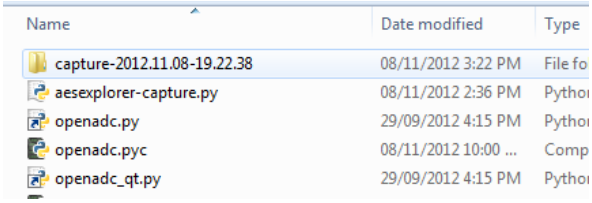


You can also adjust the 'Phase Adjust'. This is highly dependent on your specific target. Some targets, such as the MEGA163 card, are very particular to the phase adjust.

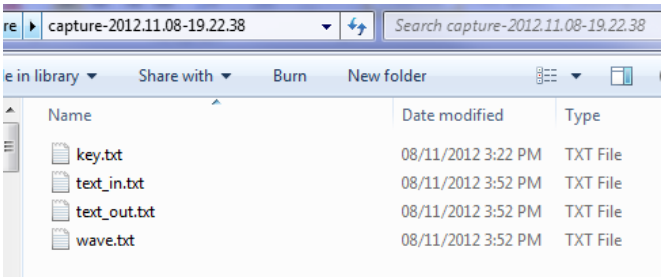
You can play around with them a little to see visually what they do. You can zoom in on the graph using the mouse button to Window around a specific area.

Now all you need to do is capture several traces. On the 'General' tab set the number of traces to 3000 for example. Then hit 'Start Capture', you will see the Trace Number increase, and each time the new trace will be written to the preview window.

It will take some time for this to complete, so go have a coffee (or beer). Once it completes the files will be written to a directory in the same file:



You can see the files in this directory contain all the data required for the attack:



For now we won't be doing the attack though, just the capture part. Now that you've got an idea of what is required, let's look at more details of the capture.

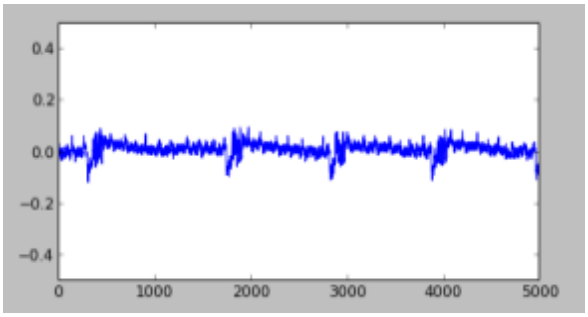
Capturing Traces: Capture Hardware

To get the traces into your computer, you need to use some capture hardware. Basically you can use any FPGA board you want, but sticking to some of the common ones might make your life easier.

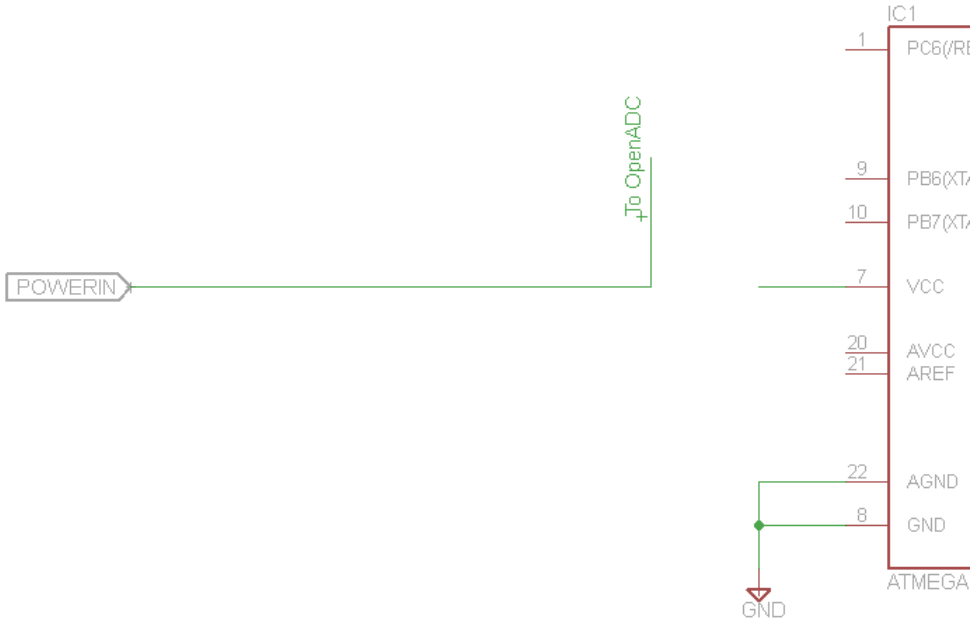
Capturing Traces: Advanced Topics

Considering Noise

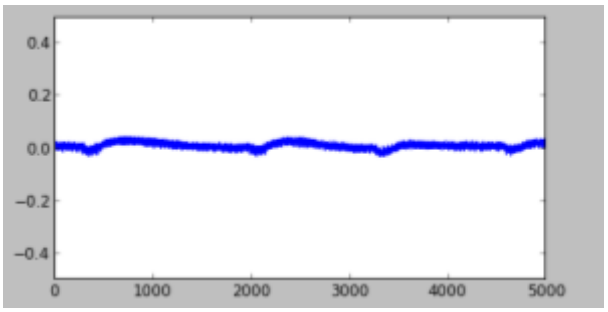
Noise can come from many sources, and you need to minimize it in your measurement system. Consider the following capture. This was done by measuring just the power rail of a target, without the microcontroller itself present:



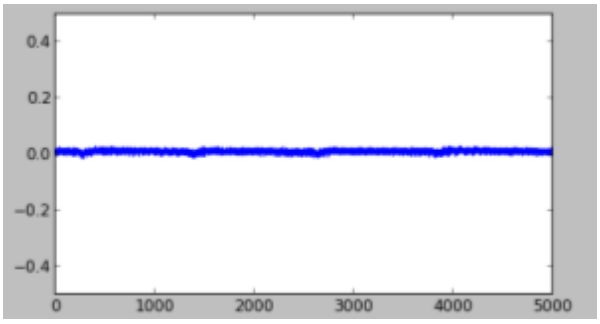
This is bad! There is far too much noise in this measurement! The voltage regulator for this example comes from a USB chip (CP2102), which has a fairly poor regulator! This is where all the noise comes from, so your systems will probably have less noise. Schematically, this looks like the following:



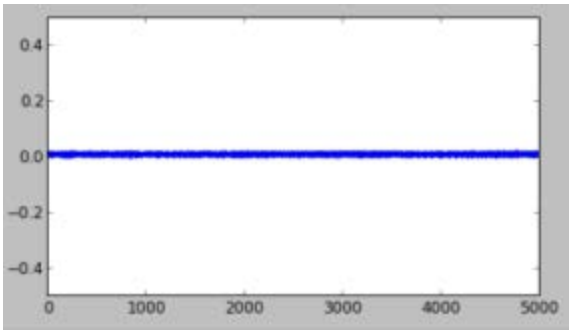
Now, we add a 2.2uF ceramic capacitor at the power rail. The trace now looks like this:



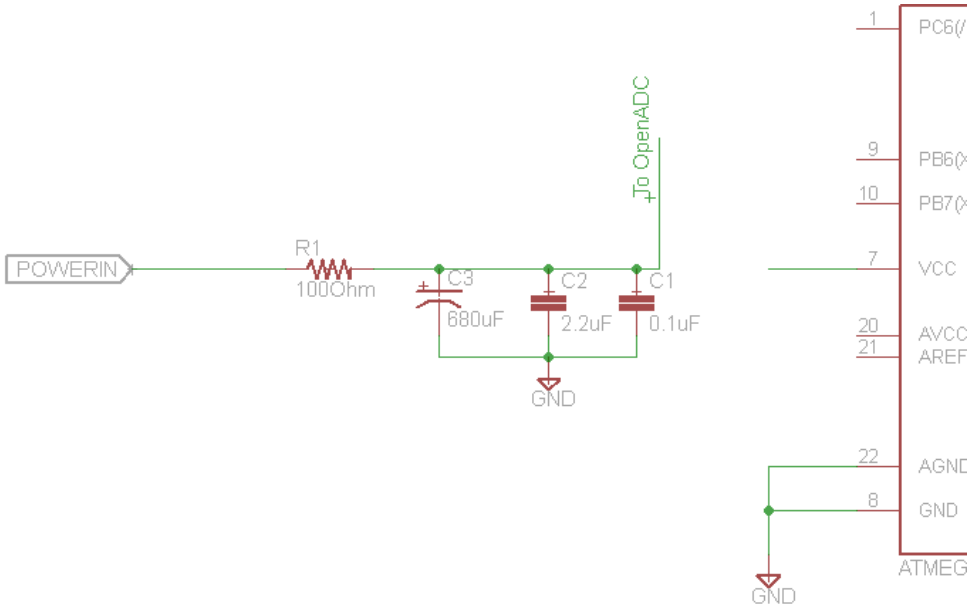
A slight improvement. Next we add a 680uF capacitor to try and even out the lumpy bits:



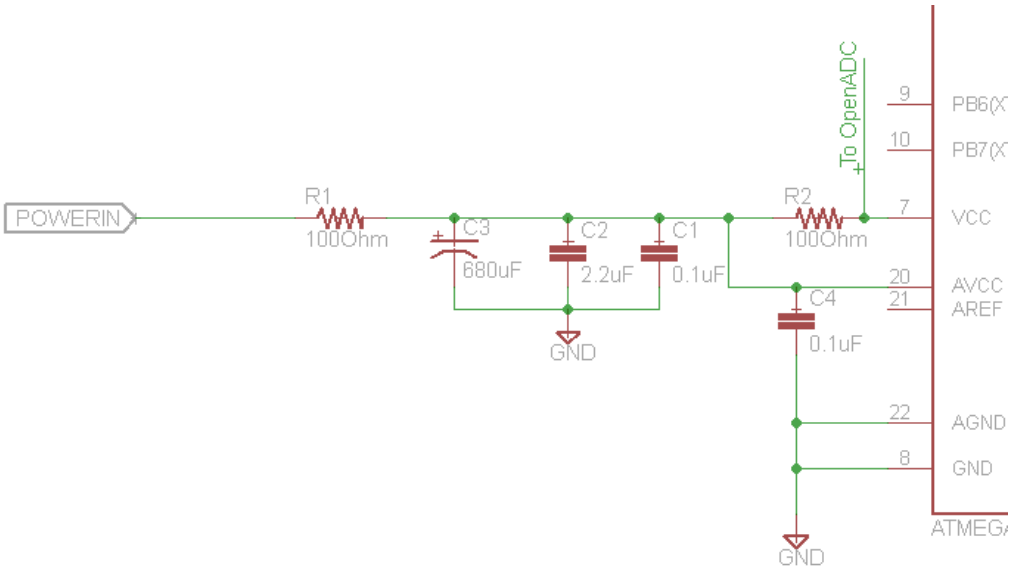
Finally we add a resistor in-line. This helps decouple us from the regulator to give us the final clean signal:



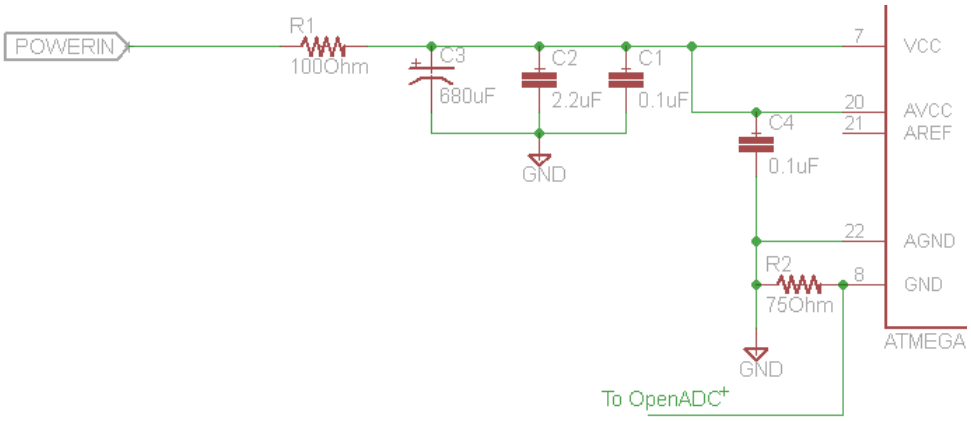
With the resulting schematic looking like:



So to do the actual measurement we insert a shunt (and also connect up AVCC), and measure the voltage at the microcontroller side of the shunt:



Note this example is using the shunt in the VCC line. You may get better results with the shunt in the GND line instead, which would look like:

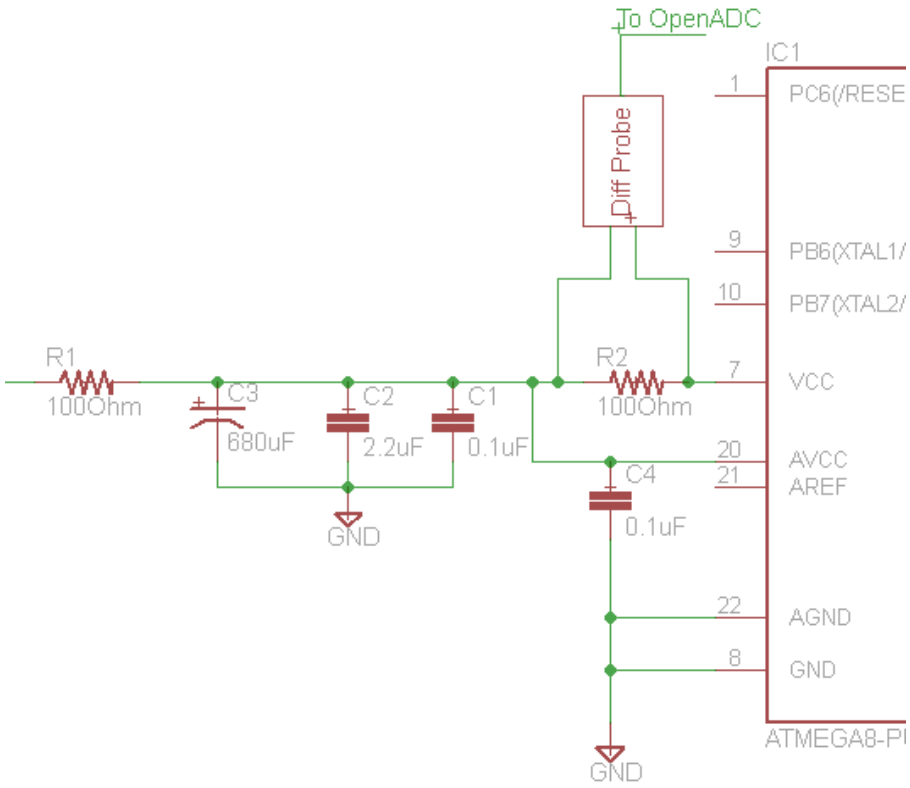


Differential Probe

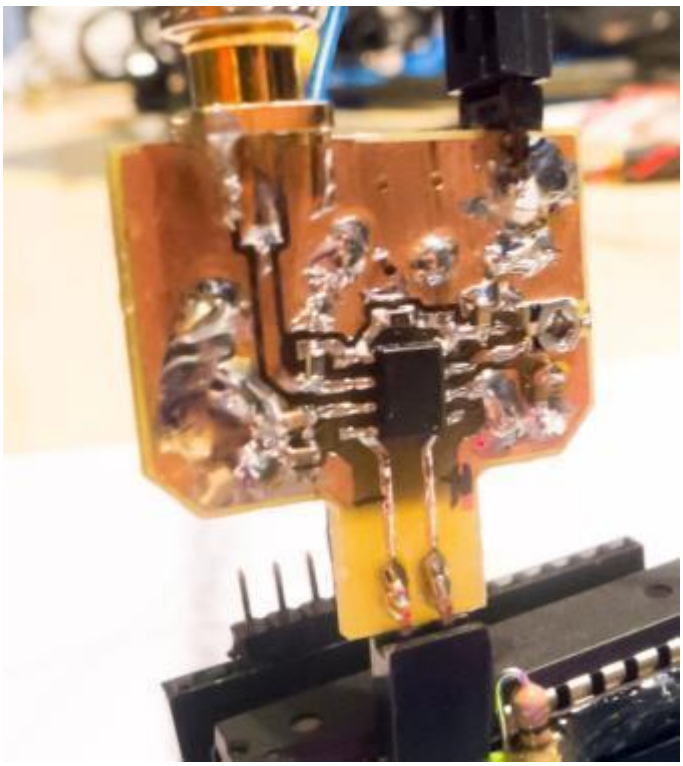
The previous diagram showed measuring the voltage across a resistor. This was measured 'single-ended'. Besides just measuring the drop across the resistor, you will also measure any variation in voltage. This could be due to the regulator, environmental noise, or voltage variations from other areas of the circuit switching.

Rather than use a single-ended probe, a differential probe ignores that variation in voltage that is *common* to both sides of the resistor, also called 'common mode' voltage. Based on the same

schematic as before, here is what the differential probe would look like:



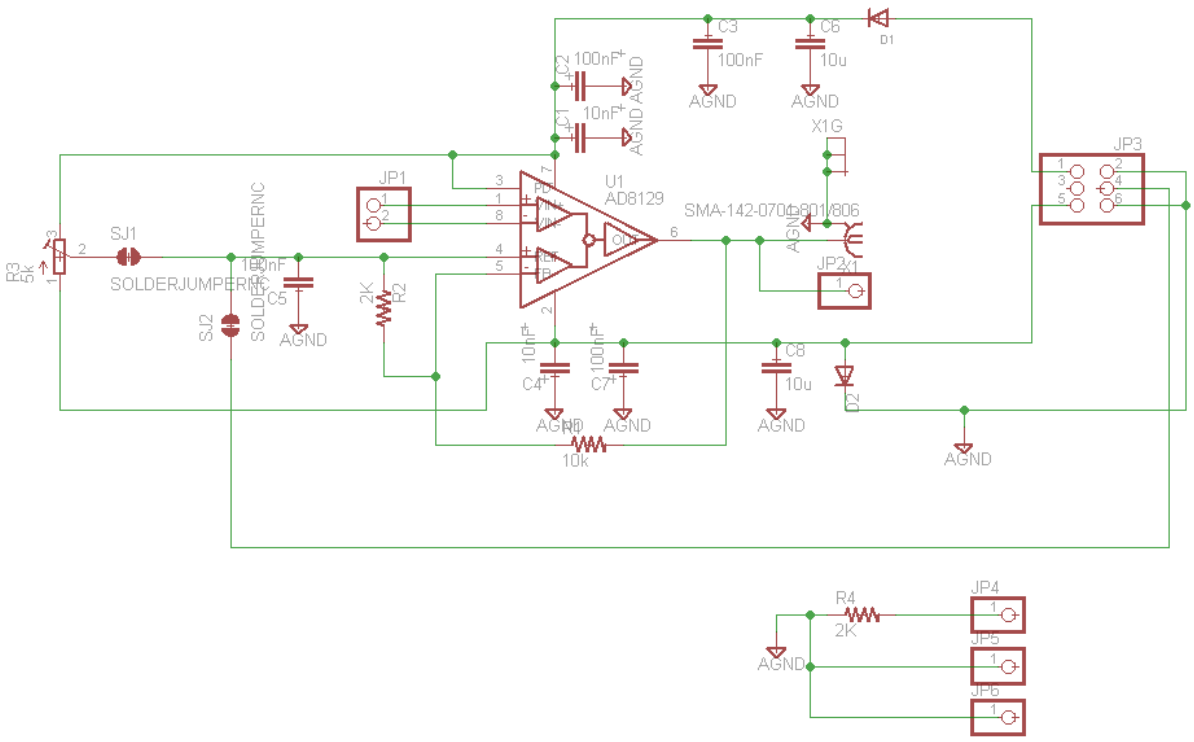
You can buy differential voltage probes, but they are pretty expensive. So here I will show you how to build one for around $20 (depending what you've got on-hand). The end result might look something like this:



The critical part is the AD8129/AD8130 differential amplifier chip. You will typically add some gain right here, as that will reduce the ability of noise to affect the signal. How much gain to add is up to you – with the OpenADC you don't want too much gain, as the maximum input signal swing is limited.

The AD8130 part will work if you want gain of < 10 (recommended for your first probe with the OpenADC). If you want higher gain use the AD8129 part, which is only stable for gains > 10. If you are building a differential probe for a generic scope, you'll probably want gains of more than 10.

The schematic is shown in the following image:



Note that R3 should be a multi-turn trimmer pot (unlike the single-turn one shown in the images). The gain is set with:

$$G = 1 + (R1 / R2)$$

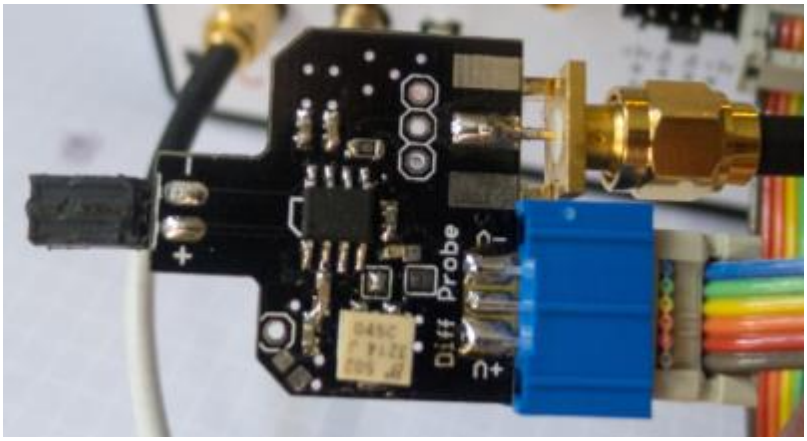
Note the gain for the example schematic would thus be about 6. This should actually be built with an AD8130 part, not the AD8129. This schematic is based on my actual design, and if you are starting again should modify it.

The AD8129 must have supply voltages with at least about 1.3V of headroom. In the given design –VS can be connected to ground, and you only need to supply a single positive voltage. When this is done you COULD NOT use the provided design on a shunt in the ground path, since the lowest common mode voltage it could measure is around 1.3V. Additional details of using this design will be posted in the Wiki.

You must also thus power it at a higher voltage than the operating voltage of the system you are measuring. If you are measuring a 3.3V system, set VCC of the probe to 5V. If measuring a 5V system, power the probe from 7V.

If you want to use the probe on a ground shunt, you will need to generate a negative supply of at least -2V and connect that to –VS.

The PCB Layout in the ChipWhisperer wiki looks something like this when built:



There are a few extra features of this PCB layout, which is based on the given schematic. Specifically:

- If powering from a single-ended supply, bridge the 'V-' pin to ground, and apply power between GND & 'V+'. If using a single-ended supply you **cannot** use this to measure current on a ground shunt as previously mentioned.

- The testpoint on the lower left of the image can be used to connect your system ground to the probe ground. Note you need to ensure the common-mode input voltage is within the 'allowed' range (e.g. at least 1.3V away from either supply voltage). If you are powering the differential probe with another power supply, you **must** make sure they are referenced to each other by connecting the 0V/GND points together. Using a resistor reduces a small differential from causing a large current to flow if you accidently have ground loops (multiple connections of system grounds).

- The adjustment voltage can come from the connector, so you can mount a multi-turn variable resistor somewhere more convenient to adjust this.

Usage Instructions

The following is a brief guide to using the probe.

1. Determine the device power supply, and power the differential probe by at least 1.3V more than this. If using the device in the ground shunt you'll need a suitable negative supply.
2. Determine where you will plug it in – this means normally across a shunt inserted into the VCC line of the device under attack

3. Connect the positive differential input to the higher voltage side of the shunt (e.g.: side of shunt connecting to target VCC)
4. Connect the negative differential input to the lower voltage side of the shunt (e.g.: side of the shunt connecting to the device under attack)
5. Connect your oscilloscope to the output if you have one. If you are using the OpenADC, instead connect a multimeter at this point.
6. Make sure you have grounds of differential probe & target somehow connected.
7. Power everything up.
8. Adjust resistor R3. If you start with the resistor at one extreme, you will see the output start at some fixed limited voltage. This is the op-amp trying to drive the output beyond what it is capable of. Typically this will be either around 1V or VCC-1V (e.g.: if powering from 5V, you'll see around 4V at the output). You want to adjust resistor R3 until the output is half-way between the two voltage supplies of the differential amplifier chip.
    a. If +VS is 5V and –VS is 0V (GND), this means you want the output to be around 2.5V
    b. If +VS is 7V and –VS is 0V (GND), this means you want the output to be around 3.5V
    c. If +VS is 5V and –VS is -2V, this means you want the output to be around 1.5V
9. Note the resistor will be very finicky! If you didn't use a multiturn resistor here you will be kicking yourself, as the tinniest turn of the screwdriver may cause the output to go to the other rail
10. If you cannot get the previous steps to work, the differential voltage across the input may be too high. Short the inputs together as a test, or substitute a smaller shunt resistor.
11. Now that you've set resistor R3, connect the differential probe to the OpenADC/scope (if not already done) and capture as normal

Electromagnetic Probes

Rather than measuring the current across a shunt, you can also measure the electromagnetic field from a chip. You can build an electromagnetic probe with a length of semi-rigid coax:

Then form it into a loop and solder the shield as shown. Cut a little bit of the shield away:

You can see more details of constructing such probes at [4] and [http://www.compliance-club.com/archive/old_archive/030718.htm]. Specific to side-channel analysis, you might want to read [5] as well. These probes can be connected to the OpenADC, you will need to set gain to maximum.

Building Amplifiers

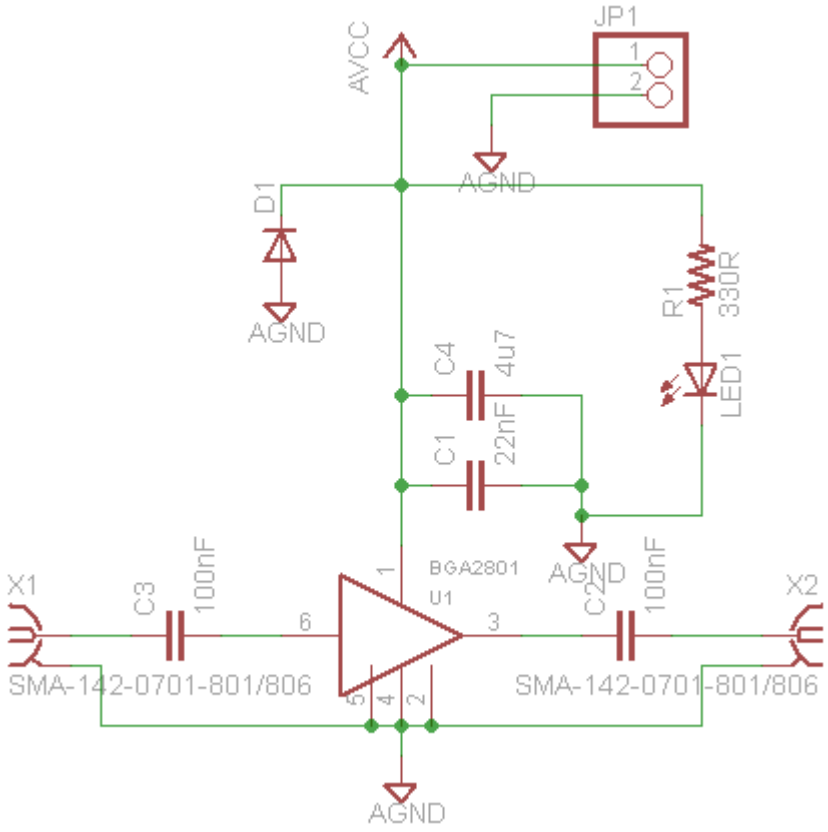
For the electromagnetic probes mentioned above, you may ultimately need an external amplifier. The signals coming from those probes is very faint, and the OpenADC may not provide quite enough gain.

The simple solution is to purchase one of these 'Block' RF amplifiers. They normally look something like this:

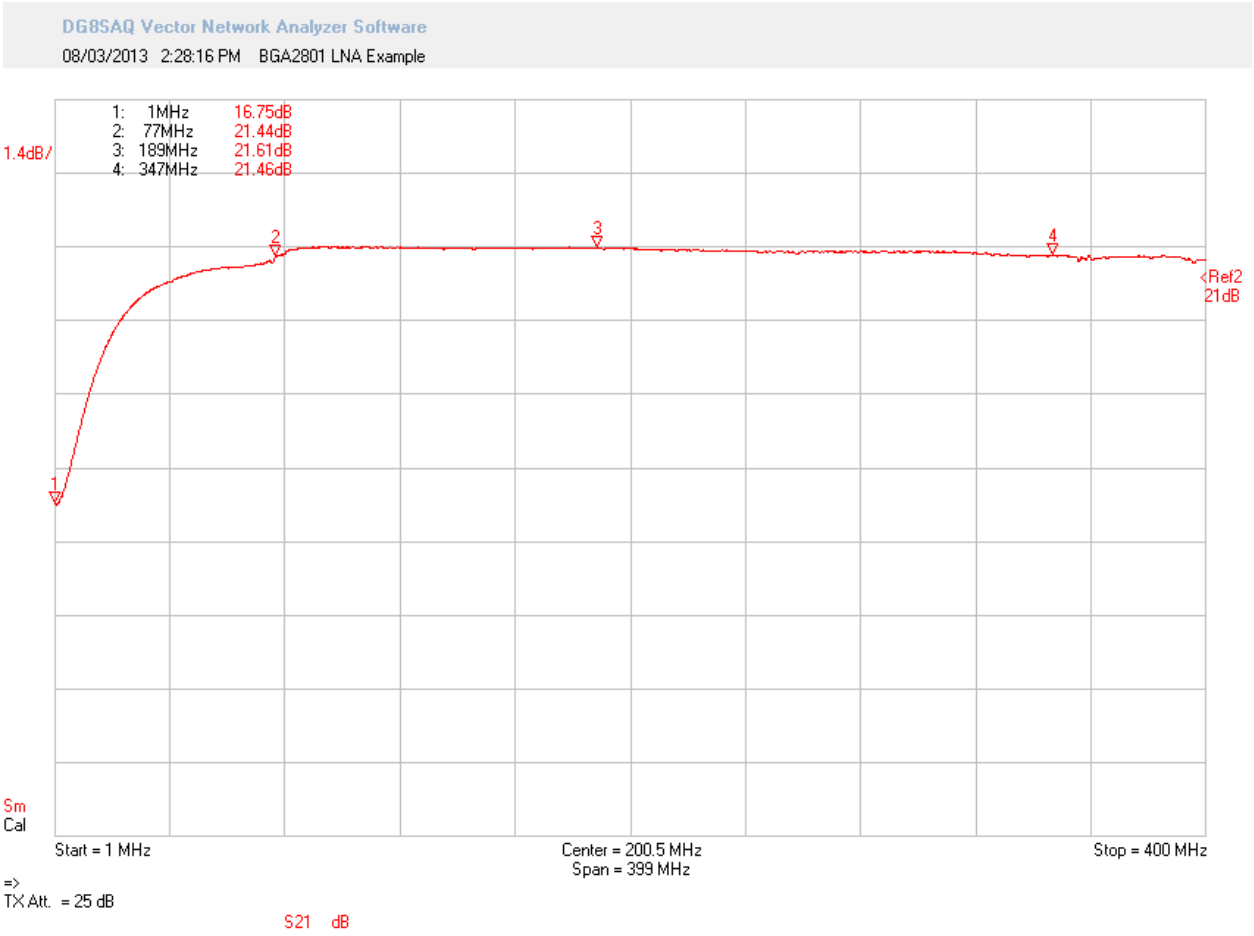
The MiniCircuits ZFL-1000LN or MiniCircuits ZFL-500LN are two examples of such amplifiers. **WARNING: As they are built for RF, they may be unstable when used with incorrectly matched outputs or inputs (not 50-ohms). Unless you are taking special precautions you will likely run into such issues. As such, I recommend to instead use some of the devices mentioned next.**

Amplifiers from MMIC Devices

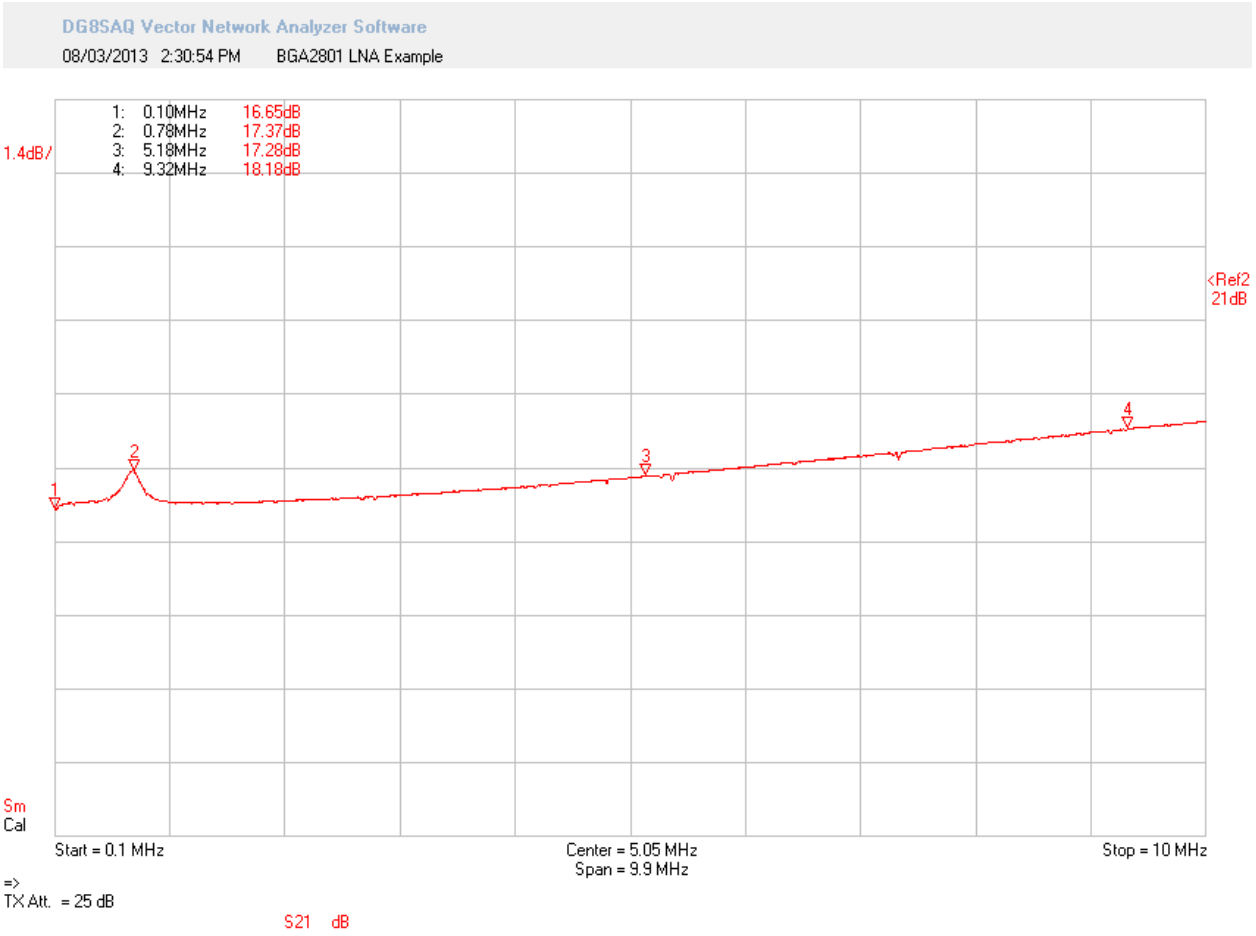
A number of single-chip amplifier solutions exist. One example I use is the BGA2801 device from NXP. This small package integrates everything into a single-chip solution, and is unconditionally stable across most of the frequencies we are interested in. These devices have lower frequencies somewhere around 0.1 MHz – 1 MHz typically – they may not be suitable for targeting slow devices. The following shows an example diagram of using the BGA2801. Note it requires a simple 3V supply, but has limited input and output power range due to this limited swing.



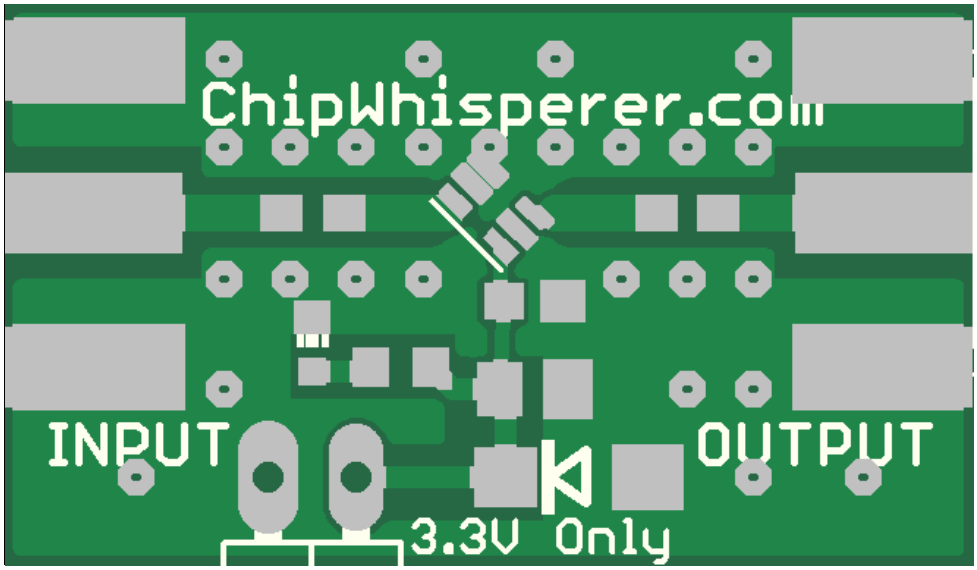
The following shows the forward gain (called S21) in dB over 1 MHz - 400 MHz.



A close-up of the gain over the range 100 kHz – 1 MHz is shown below. The gain is approximately 16dB over this range (40x in linear scale).



An example layout of the above circuit is shown here:



For more details see the Eagle project in the ChipWhisperer GIT repository, or see the ChipWhisperer Wiki page on the small-signal Low Noise Amplifier (LNA). NB: Ask on the mailing list, as this page is currently not up.

Here is a LNA + H-Probe example, along with a differential probe example:

Amplifiers from Op-Amp Devices

Simple amplifiers can also be built from op-amp devices. These amplifiers will easily work down to DC, and it is fairly straight-forward to get operation up until reasonable frequency levels (e.g.: 50 MHz).

See any standard electronics reference for building amplifiers using Op-Amps. When selecting an op-amp, one of the critical parameters to look for is the 'Gain Bandwidth Product'. This means the maximum bandwidth of the amplifier will depend on the gain of the circuit you've designed. An op-amp with a GBP of 300 MHz used to make a 40x amplifier would have a bandwidth of approximately 7.5 MHz. Thus be aware that the unity-gain bandwidth typically advertised on the front page of the datasheet *will not* be the same bandwidth you get when using the op-amp with gain. Using a circuit simulator is the best way to get an idea about the expected performance of your circuit.

Target Practice - Hardware

In order to practice all this, you will need some targets. Basically any microcontroller can be used – I'm a fan of the AVR microcontrollers for my real engineering projects, so will use them here. Any microcontroller should work, but if you want to follow along I suggest trying and duplicating these results as closely as possible.

NOTE: I haven't included complete step-by-step instructions for most of these, especially for the firmware building & programming. You should already be familiar with AVRs and know how to write C code for an AVR and download it to the AVR. If not, you should first work on getting some simple C programs working on an AVR you wire up.

A few notes specific to the AVR targets:

- We will use a port output as the 'trigger'. When that port switches it tends to contribute a lot of noise on the power trace, which is not desired. To avoid this I recommend using one of the GPIO pins shared with the ADC inputs (PortA). The digital driver lines for PORTA are powered from the AVCC (Analog VCC) for most AVRs. This means you can add lots of decoupling on the AVCC power pins. This decoupling will not affect the power traces measured on the VCC pins, but will provide a good source of power for the trigger current.

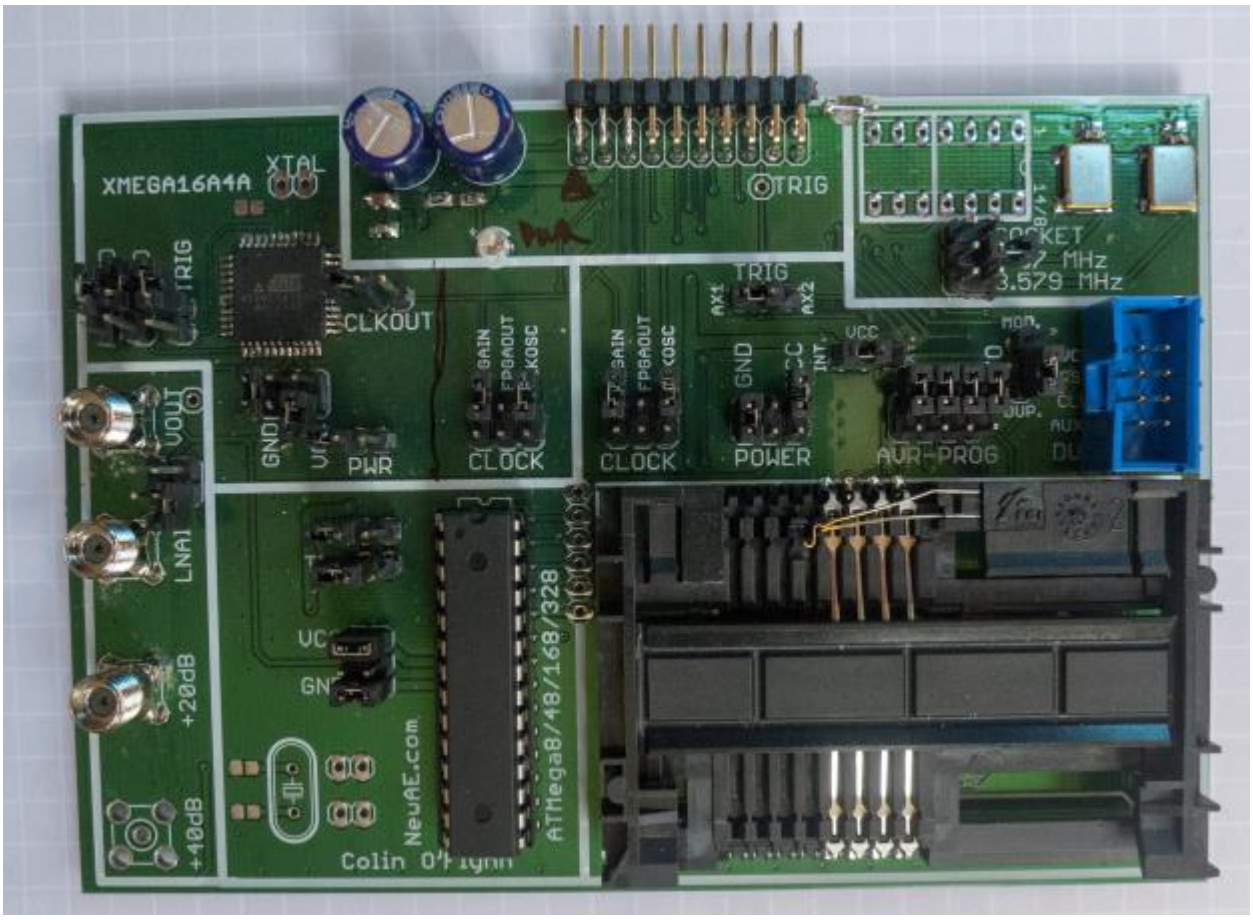
- Most AVRs can be setup to output the system clock on a certain pin (CLKOUT). Some AVRs don't have this feature. Those that don't, use the XTAL2 pin which is the output of the crystal oscillator driver. To use this properly you may need to adjust the fuses on your AVR, in particular programming the fuse that causes the signal swing to be 'full-scale'.

- Different AVR families have been made with different geometries on the silicon process. Following most semiconductor devices, older ones were made with 'larger' feature geometry. This means that the actual silicon transistors and traces on the die are physically larger. The physically larger geometry means they take more power during switching

- For real systems it is desired to minimize the powe

Multi-Target

The multi-target is a special PCB that contains multiple test devices, so you can play with AVRs, XMegas, etc. Details of this specific target board are beyond this white paper, but the following shows an implementation of this board:



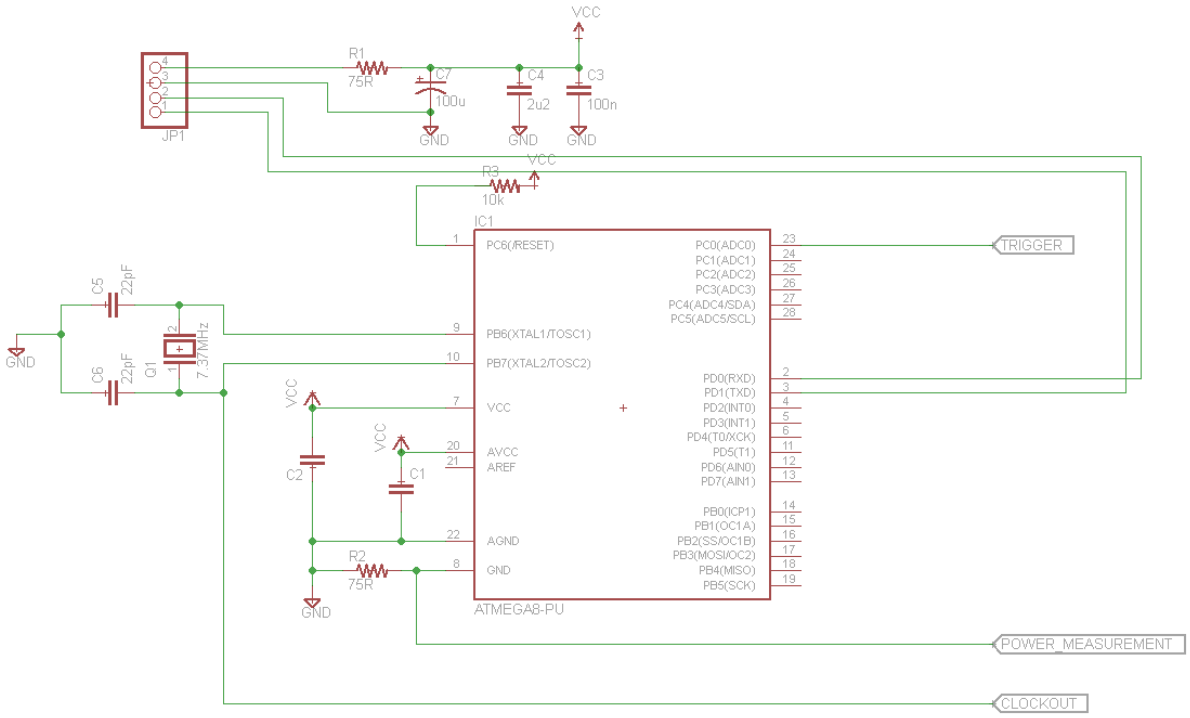
Note the board is specifically designed to work with the ChipWhisperer-Capture Rev2 Hardware. It can be used with other devices but will require some work breaking out the main connector.

This board features the following:

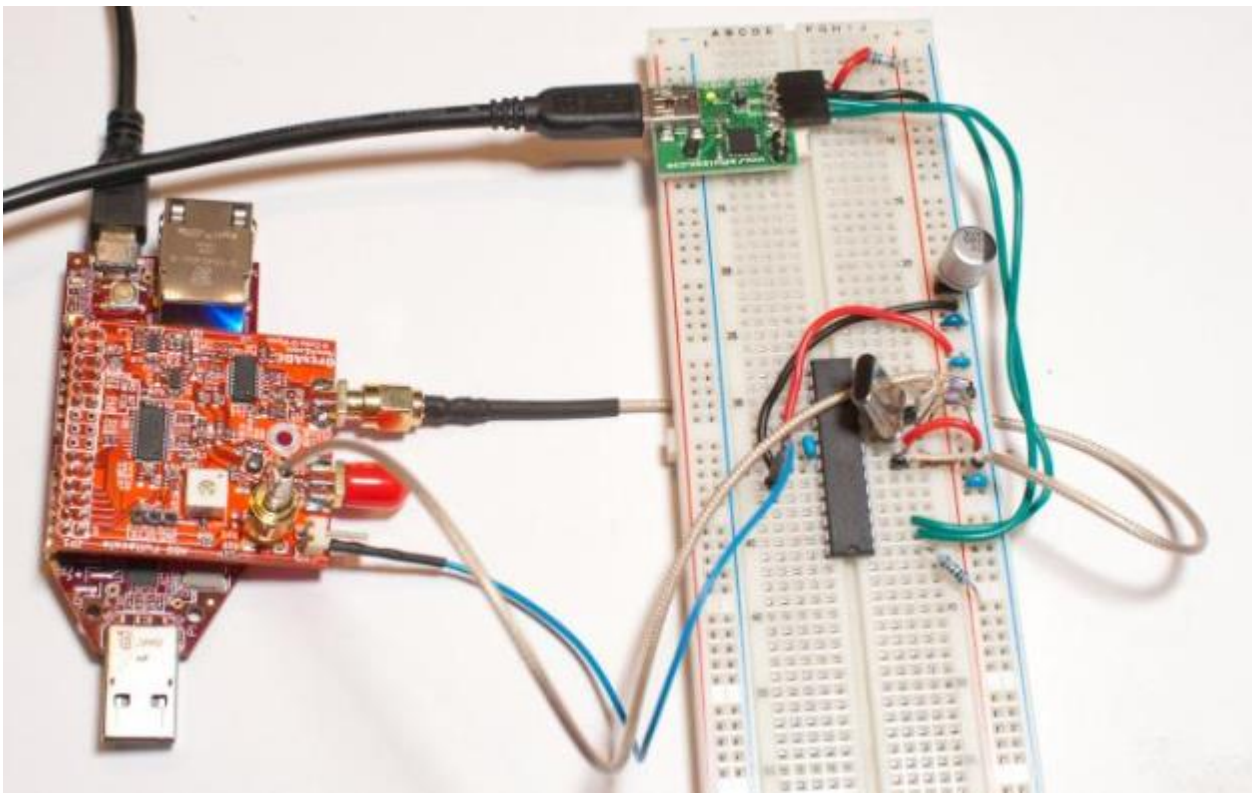
- Easily change device being attacked
- Interface to programmer built into ChipWhsiperer-Capture Rev 2
- Two LNAs built in
- Multiple oscillator sources
- Smartcard Socket, with adapter allowing pass-through attacks on Smart Cards

Simple AVR Target

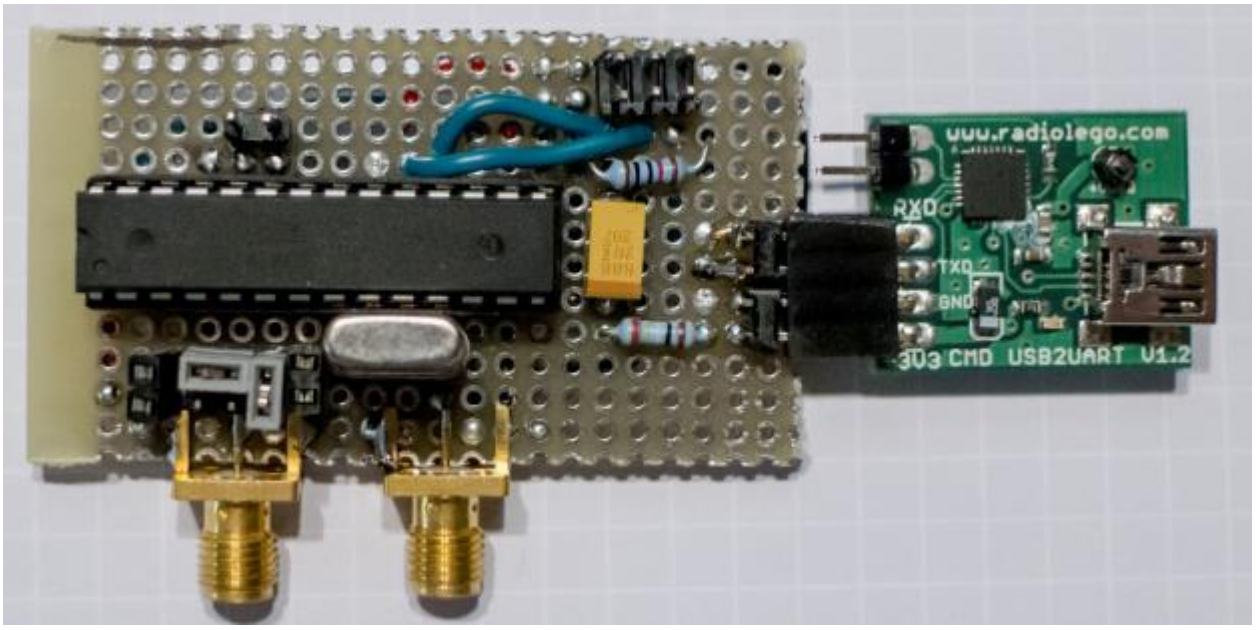
The first target to demonstrate is based on the Atmel AtMega8-16PU target. You will need a programmer tool for this to work (see Buying Guide). The basic schematic is shown below:



You can build this on a breadboard if you wish, as the following figure shows:



Alternatively you may wish to build this on a piece of PCB material so you can transport it easily:

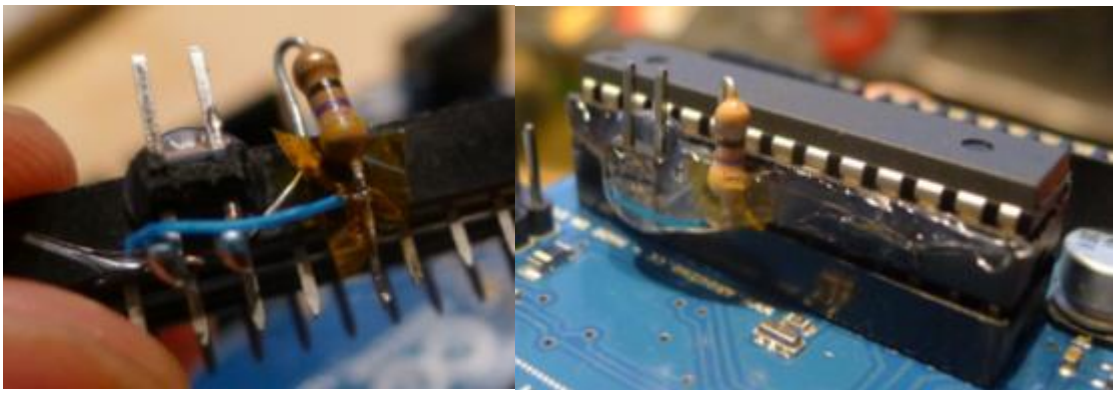


To use this target, we connect the clock, trigger, and analog line to the OpenADC.

Note: A schematic is provided for the Mega8 and for a Mega48 version. The Mega48 (or Mega88/Mega168/Mega328) is a newer chip which adds a CLKOUT pin. It is highly recommended to use this newer part if possible, and use the CLKOUT pin to get the clock source. The use of the CLKOUT pin is considerably easier, as you can easily stop the oscillator when attempting to pick the clock from the XTAL2 pin of the Mega8.

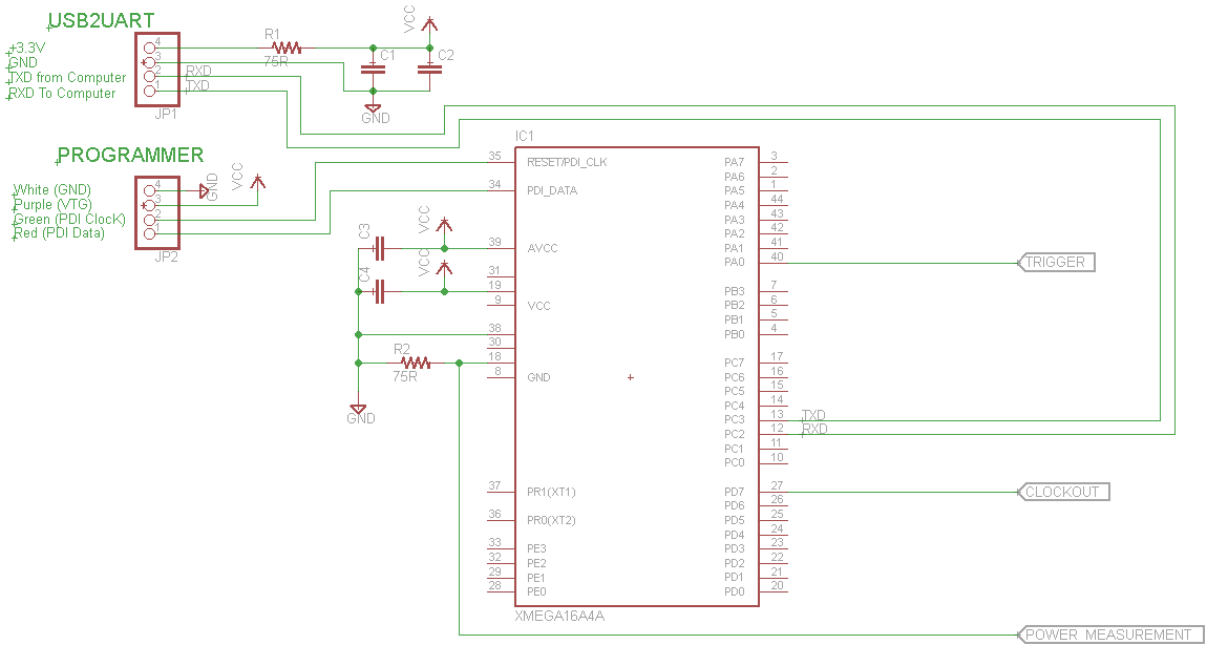
Arduino Target

The Arduino is a popular embedded device. It contains an AtMega328P device with a crystal oscillator – so we can use it to do some side channel analysis. You can either lift the pins of the DIP package IC, or build a little adapter socket that inserts a resistor in GND (preferred) or VCC (also works):

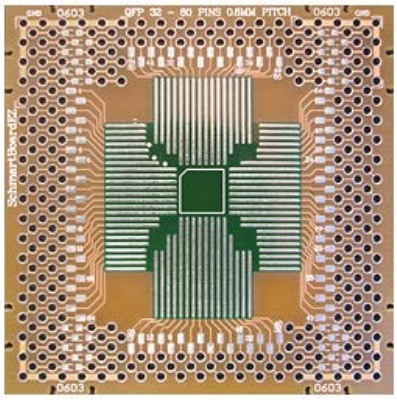


Simple XMega Target

The XMega device contains AES hardware, which has proven susceptible to Side Channel Analysis[6]. At this point we will only be demonstrating an attack of software AES on this platform, but it should be possible to expand this to attacking the hardware AES engine.

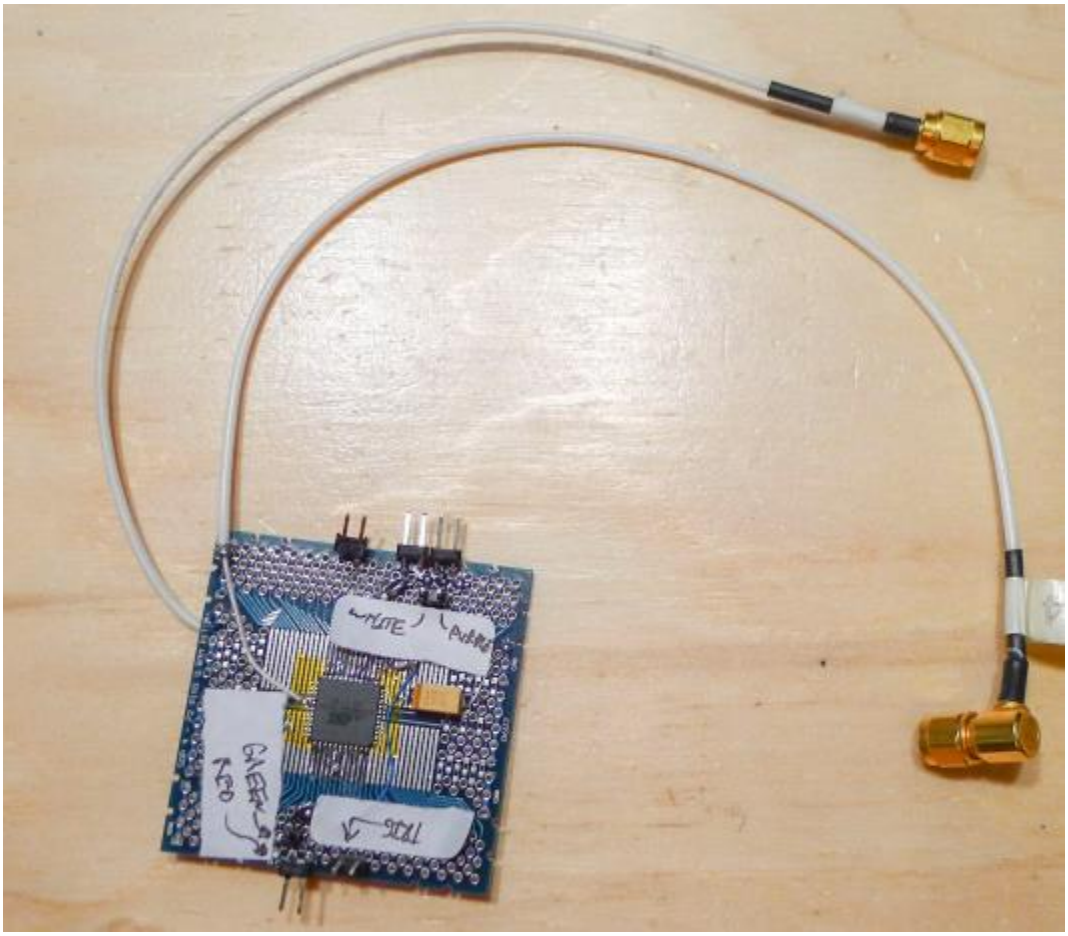


SchmartBoards are a very useful platform for building attack platforms. They allow you too easily solder SMD devices, and look something like this:



They have a very clever layout making it easy to add decoupling capacitors to ground on the backside. In addition we can use that to add a resistor in one line going to ground, like the following shows:

So the top-side of this target looks like this:



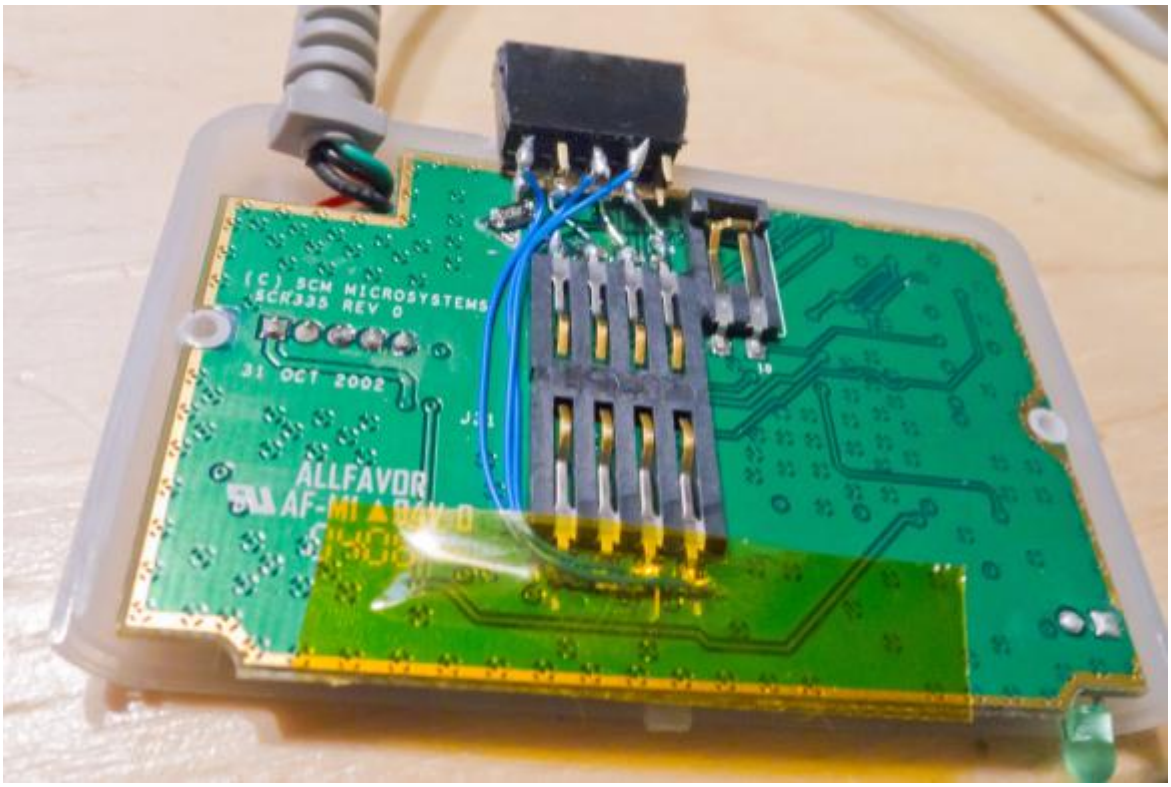
The White/Purple/Green/Red labels are indicating connections to the JTAG MK2 / JTAG3 / ISPMK2 programmer. The 'TRIG' label is the trigger. The 4-pin header on the upper right side connects to the USB2UART for serial communications and power.

SmartCard Target: A Normal Reader

Smart cards have a certain degree of 'sexyness' when it comes to demonstrating attacks. A popular card for this demo is ones based on an AVR microcontroller. It's critical to understand the following:

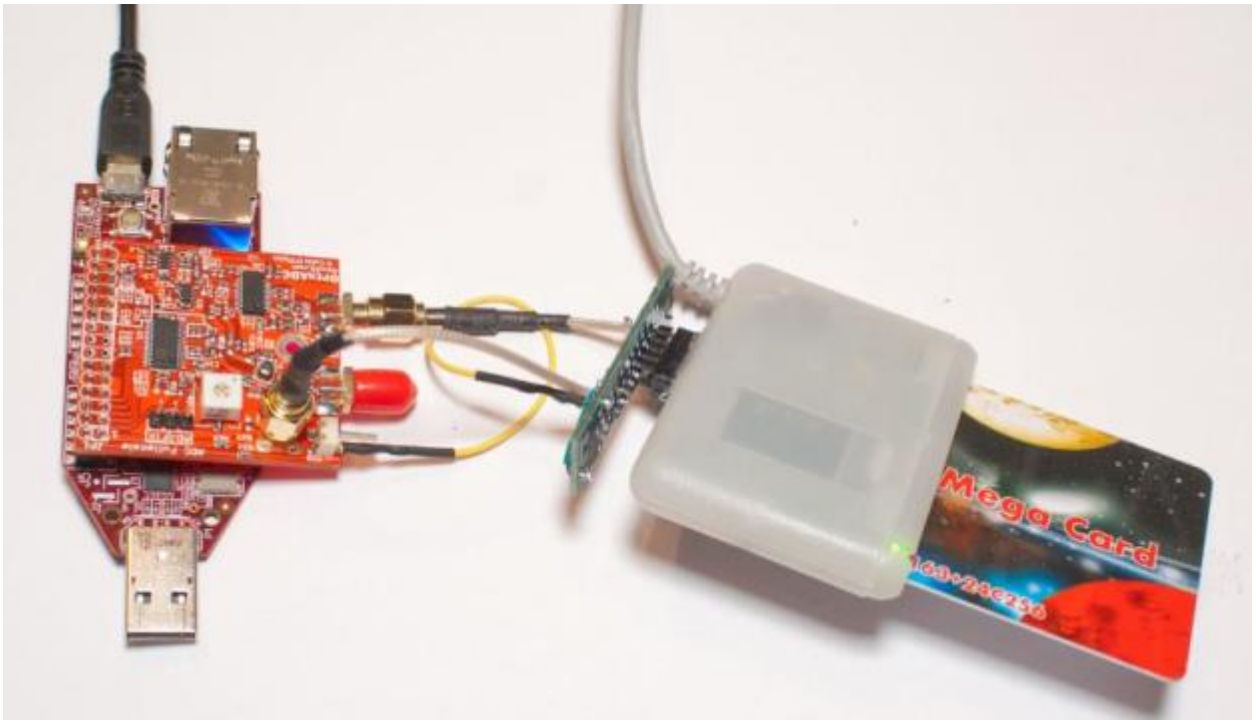
There is no difference between demonstrating an attack on one of these Smart Cards and demonstrating an attack on a discrete AVR. It's much easier to get a hold of discrete AVRs, so I don't even recommend bothering with the smart card attacks. Certain microcontrollers only come in the Smart Card form factor, so with those you **do** need to attack the smart card.

To attack the smart card, you simply insert a resistor in the power lines. Also bring out the clock pin:

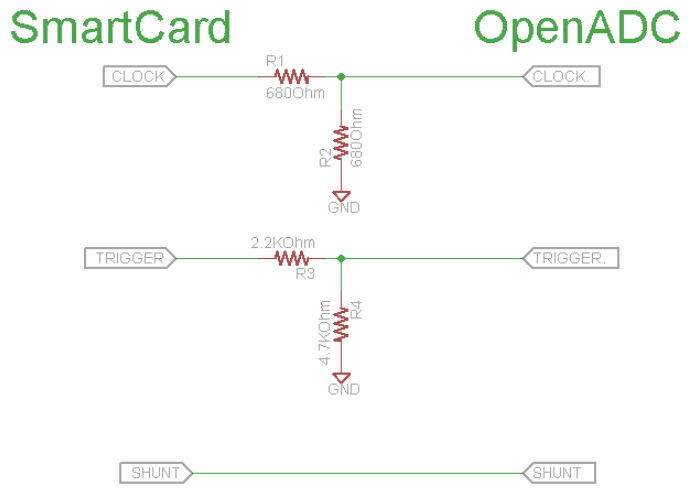


Close up the Smart Card reader, I added a connector on the backside:

WARNING: Your Smart Card will probably run at 5V. The FPGA connected to the OpenADC will be destroyed by 5V, it only accepts a 3.3V input signal. You need to add some resistive dividers on the Clock and Trigger line most likely. The analog input is fine at 5V due to being AC coupled.



My example resistor board on the backside has this basic schematic:

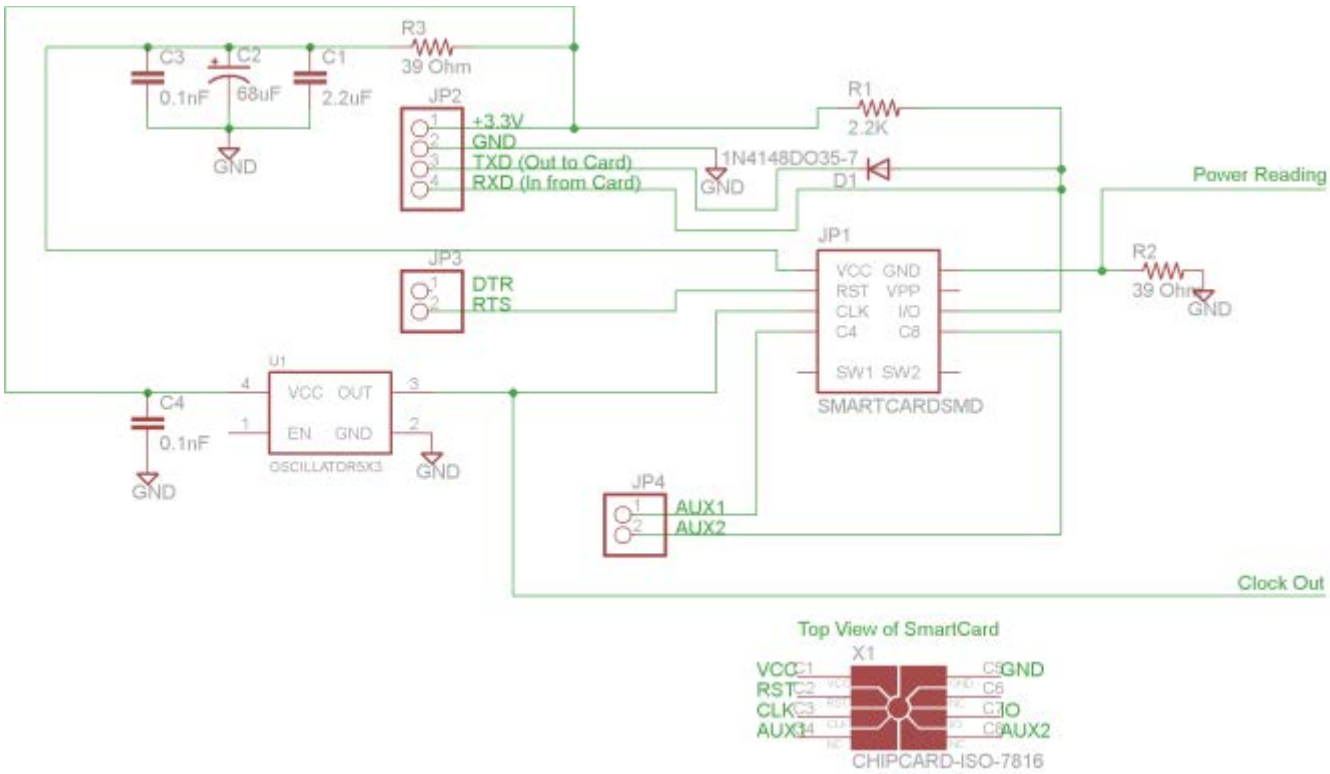


You MUST ensure the grounds of the two systems are connected for this to work! Note because the OpenADC as an AC-coupled analog input, the shunt line does not require any conditioning. Thus even though the shunt line is sitting at 5V, that will be fine.

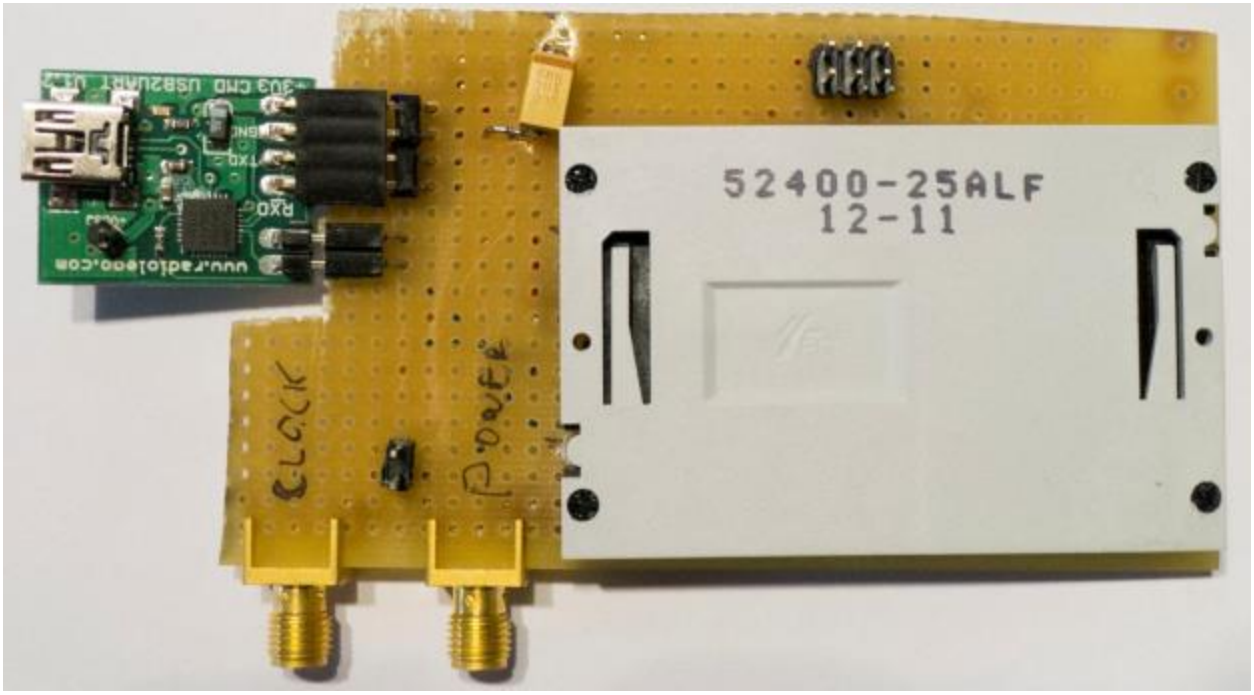
Smartcard Target - Cheapskate

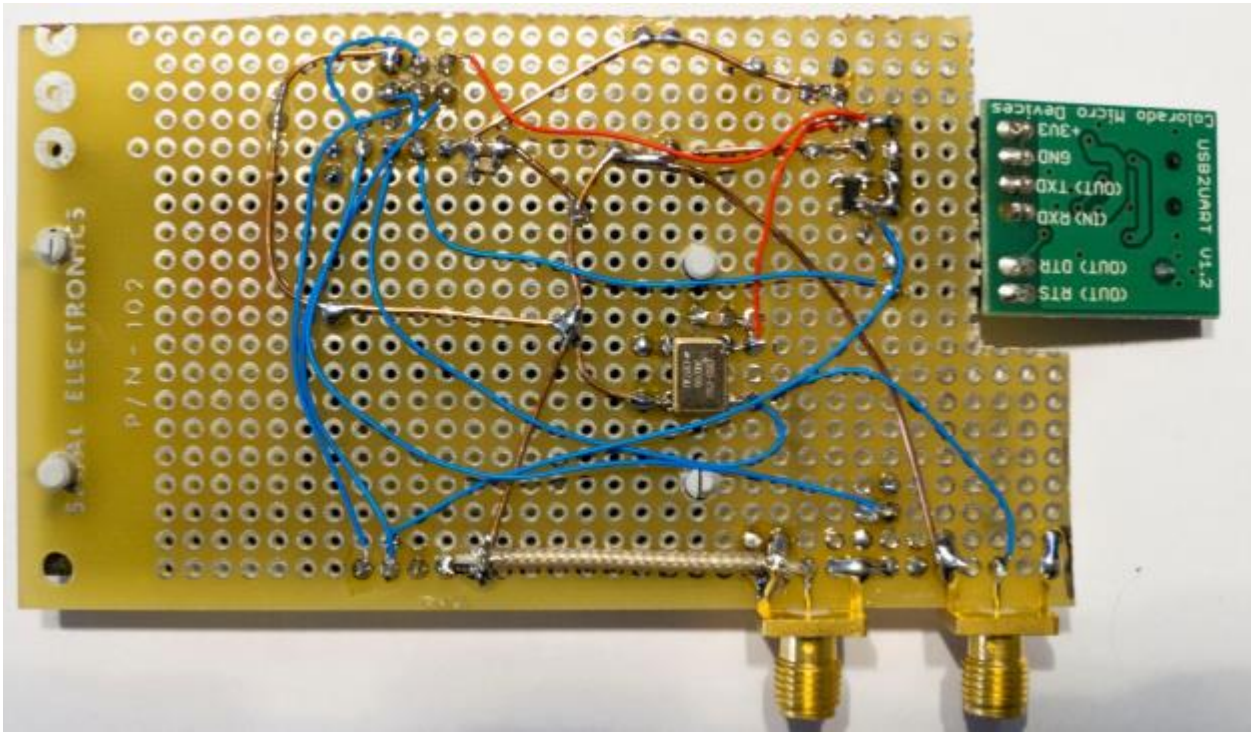
A cheap reader can be built with a logic-level serial interface, smartcard connector, and a few discrete components. This smartcard reader can be run at 3.3V, meaning it directly interfaces to the OpenADC.

The reader uses a diode and pullup resistor to convert the unidirectional serial I/O lines from the computer to the single bidirectional I/O line of the smartcard. With this method any data sent on the TX pin will also be echoed to the RX pin, so the software on the computer must account for this.



- 3.579545 MHz 3.3V Oscillator (e.g. Digikey Part No. CTX929LVCT-ND)
- SmartCard Socket (e.g. Digikey Part No. 609-1414-ND)
- Small Signal Diode (e.g. 1N914 or similar)
- SMA Connectors (for OpenADC, not needed for scope)
- Resistors, Capacitors



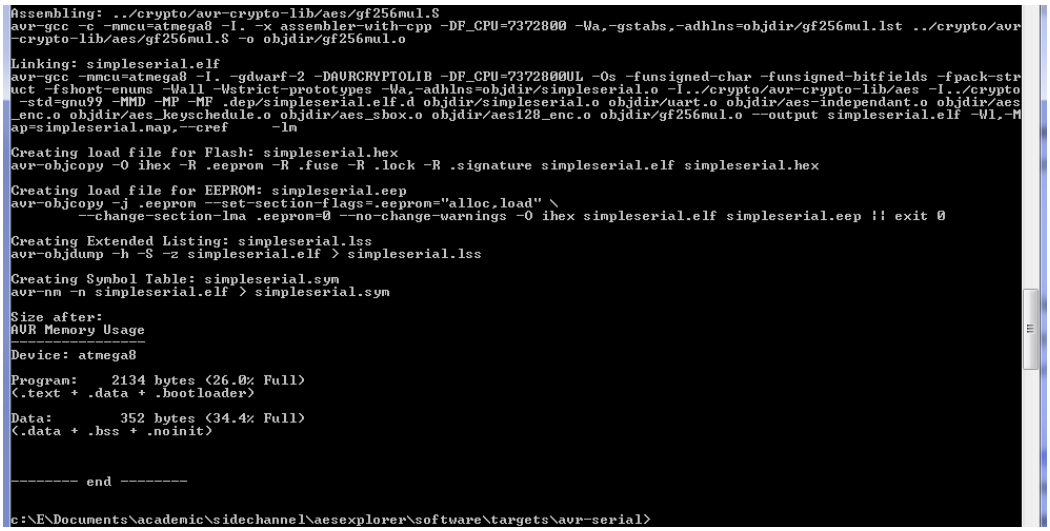


Target Practice - Firmware

The firmware examples provide the example source code you can attack. Note the most recent source will always be available from the AESExplorer project (http://www.assembla.com/spaces/aesexplorer). This open-source project means you can contribute ports to other architectures or now attacks.

How to Build

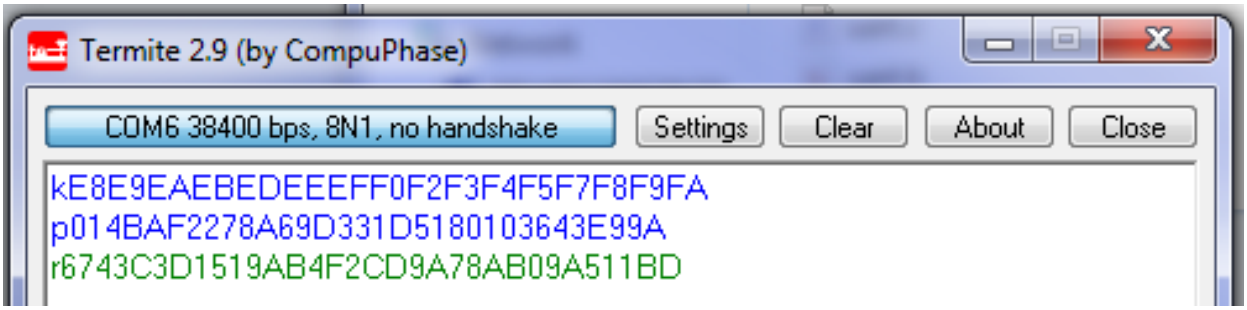
All the current targets require an avr-gcc compiler. On Windows the easiest method is to download WinAVR 20100110. Then you can cd to the appropriate directory and simply type 'make'. If you've done this correctly everything will build.



SimpleSerial Firmware

Description

Currently there are two target folders: avr-serial and xmega-serial. Both these implement the 'Simple Serial' protocol running at 38400 baud rate. In this protocol sending in ASCII 'kE8E9…F9FA' would load the encryption key E8:E9:…:F9:FA. There will be no response by the target device. Then sending 'p01BA…9A' would cause the device to encrypt with the input plaintext of 01:BA:…:9A, and send the result with an 'r' infront of it. See the following diagram for an example:



For the avr-serial firmware, you can change the target in the Makefile and clock frequency the AVR is running at.

Programming AVR & XMega

To program the AVR target, you will need an AVR programmer capable of 'ISP' programming such as the AVRISP-MK2. You can use AVR Studio 4 for programming, or the newer Atmel Studio 6. Note certain AVRs may be removed from AVR/Atmel studio – for example to program the AtMega8, which does not appear on the list of valid AVR devices, select the 'AtMega88' instead. You may have to use AVR Studio 4 in this specific example to 'override' the device signature check. If programming succeeds it will still verify the flash OK.

The XMega device can also be programmed through the AVRISP-MK2. See the help file for details of the pinout, as this uses the 'PDI' mode .

SmartCard Firmware

Description

The smart card firmware is currently not part of this project. You can download the firmware & source from http://www.morita-tech.co.jp/SAKURA/en/hardware/SASEBO-W.html (see bottom of page – "Software for ATMega163 card".

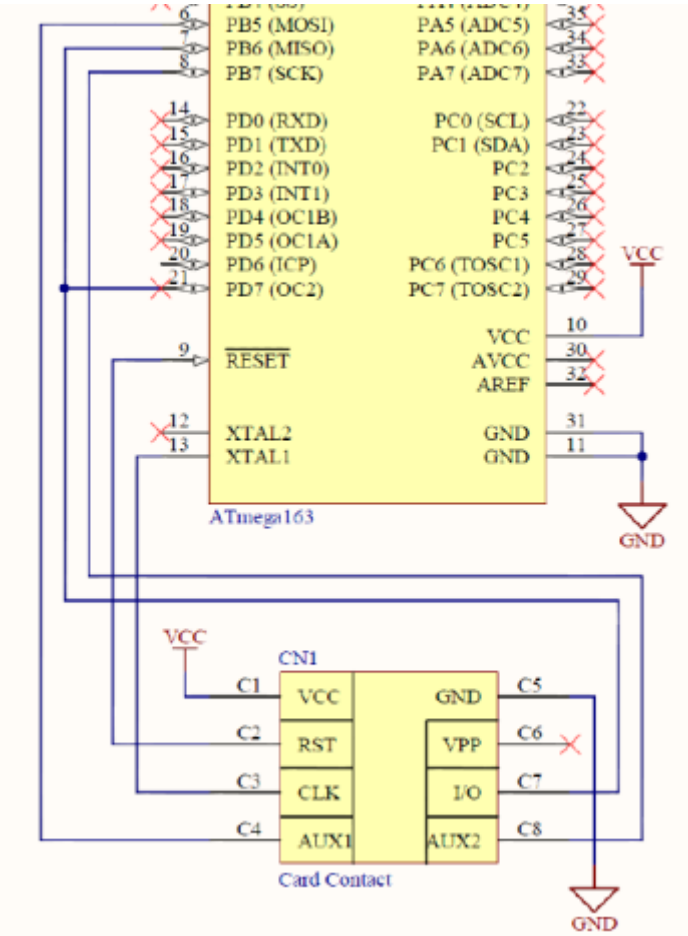
Programming

If you have a SASEBOW-W, it has already been designed with programming capabilities, see the QuickStart guide.

If you are buying a normal smartcard programmer, see the user guide for that device.

If you are building the cheapskate smartcard reader, you can add a 6-pin ISP header. Note the cheapskate reader provides a suitable clock already, all you need to do is wire out the RESET, SCK, MISO, and MOSI pins from the smartcard.

When programming, use AVR Studio 4. It will not support the Mega163 device, so instead select the 'Mega363'. Again tell it to ignore the issues of incorrect signature bytes, the programming should still succeed.



The cryptography implementation being used in these targets is selectable – it's all based on open-source AES implementations. Many real devices are made by an engineering typing into Google 'AES AVR' and looking at the results. In this case you have three options for the crypto on the targets, although more are being added:

1. avr-crypto-lib C implementation from http://avrcryptolib.das-labor.org . This is the default library used, and is a crypto library written in C. Only the AES part is included here. This is the recommended first target, as it is fairly easy to crack, but still a 'real' cryptographic library probably used in products.
2. 'Straight-forward' C implementation from http://cs.ucsb.edu/~koc/cs178/projects/JT/avr_aes.html . The version as written is subject to timing attacks, so has been quickly modified to make it timing-independent. Note this mod-

ification is not done in a safe way, so if you use a different compiler version it may fail. This implementation

3. avr-crypto-lib ASM implementation from http://avrcryptolib.das-labor.org . This assembly version of AES is much faster than the C implementation, and requires more traces to completely break. It is not protected however, and can still be broken with DPA.

The crypto library to use is selected in the Makefile. Look for the following line:

```
##############################################################################
######################### CHOOSE CRYPTO LIB ##################################
#Uncomment the lines below for whichever crypto lib you'd like to attack

########
#From http://avrcryptolib.das-labor.org, C Version
#*This is the recommended first version to test*
CRYPTO_LIB = avr-crypto-lib/aes
SRC += aes_enc.c aes_keyschedule.c aes_sbox.c aes128_enc.c
ASRC += gf256mul.S
CDEFS += -DAVRCRYPTOLIB
```

Simply comment out the old crypto implementation and uncomment the crypto implementation you wish to use.

Software, Attack, and the CD

The Example Attack

Once you have the traces, you can perform the attack. The only example given here is a simple DPA attack written in Python.

To use this:

1. Copy files wave.txt & text_in.txt into same directory as dpa-attack.py
2. Open info.xml from directory wave.txt and text_in.txt came from:
    a. Record/Remember the 'NumPoint' field value, which says the length of each trace
    b. Also note the 'NumTrace' field, which says how many traces there were
3. In dpa-attack.py edit the line that says:
   ```
   pointend = 3000
   ```
   To the number specified in 'NumPoint'
4. Run the attack (hit F5 from IDLE)
5. If you want, you can try reducing the number of traces required. This is set with the 'traceend' variable, which by default uses all traces
6. You can also play around with what points are used in the attack

Further Attacks

The ChipWhisperer software, which isn't fully described in this white paper, contains a more powerful attack called 'Correlation Power Analysis'.

In addition there is some work at the following sites to demonstrate more powerful attacks:

1. OpenSCA at http://www.cs.bris.ac.uk/home/eoswald/opensca.html
2. Examples from the DPA Book at http://www.dpabook.org
3. The DPAContest have several references available at http://www.dpacontest.org

A Buyers Guide

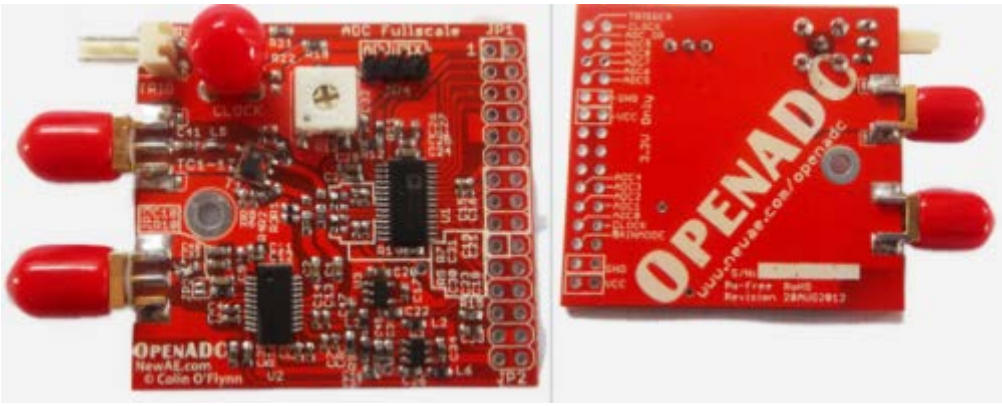
This very briefly goes over what you might want to purchase.

Scope

I designed the OpenADC for side-channel analysis. If you want, you'll NEED a FPGA board which you can use with the OpenADC, I reluctantly recommend the Spartan 6 LX9 Board from AVNET (part number AES-S6MB-LX9-G). The reluctant part is due to slower download speeds – I'm currently moving to using a faster FPGA board (from ZTEX.de see below):

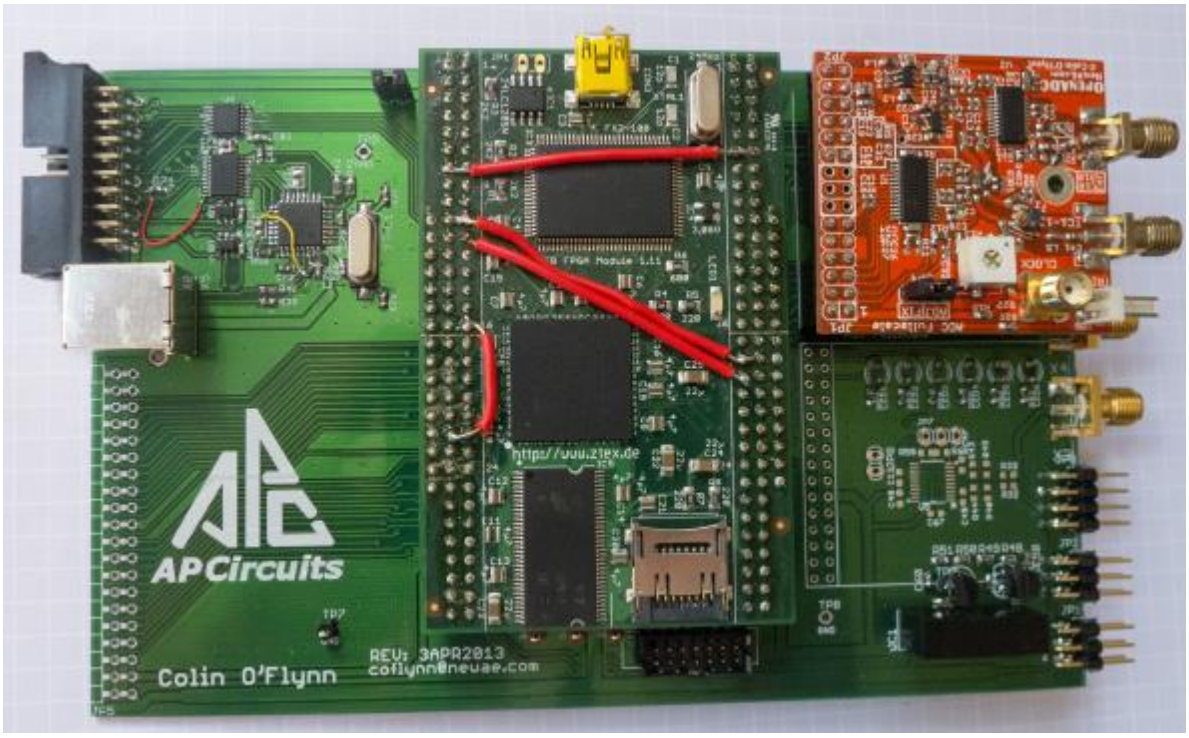
The LX9 Board is $90 and looks like this:

The OpenADC just plugs on top. Note the USB-serial interface the download speed is pretty limited, so it won't be remotely as fast as a proper oscilloscope. The OpenADC looks like this:



For a faster download, you can use a ZTEX USB 1.11 board, either the LX9 or LX25 is recommended. Currently you need to build a little carrier board for this though, but on the plus side get much faster download speeds:

There is a nicer carrier board you can use too:



Which fits into a nice case to look pretty:

If you want a generic scope, there are lots of nice USB ones. Expect to spend over $1000 USD depending how fancy you want.

One USB oscilliscope is the Cleverscope, which has an option to add an external sample clock input (option CS810). The cleverscope is the lowest-cost oscilliscope with this syncronous sample option:

TiePie makes scopes with a higher sampling rate, but more limited synchronous ability. The reference clock must be around 10 MHz, and the TiePie generates higher sample frequencies based

on this. For example the HS5 is a high-speed scope (500 MS/s):

Another alternative is the PicoScope brand, which is often used in these projects. They have a range of sample rates available (up to 5 GS/s of publication), with a fairly low cost. Some of their oscilliscopes (PS6000 series) have a generic 'Clock Input' port available too! Check with Pico Technology to be sure this feature can be used as desired.

Magnetic Field Probe

Search for terms such as 'SMA Male RF405', 'SMA Male Semi-Rigid', 'SMA Male Semi-Flex'. Suggested just to look on EBay:

Use self-fusing tape & Polyimide Tape for insulation purposes:

http://dx.com/polyimide-heat-resistant-high-temperature-adhesive-tape-8mm-33m-260-c-21351?item=2

Or try coating entire thing in rubber/plastic, such as with:

Plasti Dip

Pin It

Low Noise Amplifier

Buying Commercially

Easiest thing is to buy a LNA from somewhere like Mini-Circuits, such as ZFL-500LN or ZFL-1000LN. Combine that with a variable attenuator if you need less amplification (output is too big).

As a hack you can turn the voltage down on the LNA. As the supply voltage goes down the LNA gain goes down due to biasing – you will also see some performance degradation, so this can't be done too much!

As mentioned be aware of the limits of using such a LNA – specifically they typically expect a well-matched input and output, or they may oscillate. You need to check the *stability factor* of the amplifier to make sure it is unconditionally stable – if so it shouldn't give you oscillations when being unused unmatched.

Using MMIC

Finding a suitable MMIC device can be done through a search engine such as Digikey, Farnell, etc. Note there is a designed PCB for the BGA2801 device available on the ChipWhisperer Wiki, and a PCB is available to purchase .

Using Op-Amp

As an alternative to the MMIC, an op-amp can be used for providing amplification. Note that it may be desired to use a differential amplifier chip to provide a differential probe for the shunt, which will have some level of built-in amplification.

AVR Stuff

In North America, the easiest place to buy AVR is at Digikey. You can also buy the programmers here too. Your options are:

- ATAVRISP2 – Atmel ISP MK2 ($35). Only usable for programming.
- ATAVRDRAGON – Atmel Dragon ($50). Usable for programming and debugging *most* devices. Doesn't come in a case (bare PCB).
- ATJTAGICE3 – Atmel JTAG 3 ($99). Usable for programming & debugging all AVR/XMega devices, comes in a nice case. You probably want this if you might be doing some debugging.

SmartCard Stuff

You can buy a normal 'commercial' reader online. The specific one I used (SCR335) was purchased from EBay, several of them will come up when searching that.

There are many sources online for the 'FunCard AtMega163' used here, called 'ATMega Card' also. You will need a way to program it – you can purchase specific programmers (see Infinity USB SmartCard Programmer, or DuoLabs Dynamite +Plus). Note you can program these cards with an AVR programmer if you connect the proper pins to the normal 6-pin ISP header and provide a clock.

Alternatively, you can use the SASEBO-W board, see below.

The SASEBO Project

A company called SAKURA makes a number of useful boards for side-channel analysis. While they aren't exactly 'cheapskate', they are very high performance. For instance the SASEBO-W

contains a Spartan 6 LX150 on which you can run attack algorithms, and the OpenADC directly interfaces to this board:

There are a variety of other targets of interest. See http://www.morita-tech.co.jp/SAKURA/en/hardware.html

Where to Go From Here

You've reached the end! So where to go for more information?

First, there is the original paper in the field, available at http://www.cryptography.com/public/pdf/DPA.pdf which a shorter version at http://www.cryptography.com/public/pdf/DPATechInfo.pdf .

The 'DPA Book' is described at http://dpabook.org/ which contains a lot of useful information, going from the bare basics onward. So if you are just starting might be an easy way forward!

You can also explore on Google Scholar (or similar) different papers in this field. Papers describing more advanced technics

There is lots of experimentation that can be done too. Before going out and purchasing lots of stuff, some of this you can do on your own. There are some example traces on the CD, and more will be posted to my website.

If you're not too solid on your cryptography, I highly recommend picking up 'Understanding Cryptography', which will give you a very good background in various cryptographic principles. It covers a wide range of topics and is an enjoyable read. See http://wiki.crypto.rub.de/Buch/ for the book website, which includes recorded lectures corresponding to chapters.

Plus please subscribe to the ChipWhisperer mailing list, and ask questions there.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *Advances in Cryptology—CRYPTO'99*, pp. 789–789, 1999. Available: http://www.cryptography.com/public/pdf/DPA.pdf

[2] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*, vol. 31. Springer-Verlag New York Inc, 2007. http://www.dpabook.org

[3] C. O'Flynn and Z. Chen, "A Case Study of Side-Channel Analysis using Decoupling Capacitor Power Measurement with the OpenADC," *Foundation and Practices of Security*, 2012. Available: http://www.newae.com/tiki-index.php?page=Articles

[4] D. C. Smith, "Signal and noise measurement techniques using magnetic field probes," *Electromagnetic Compatibility, 1999 IEEE International Symposium on*, vol. 1, pp. 559–563, 1999.

[5] E. De Mulder, "Electromagnetic Techniques and Probes for Side-Channel Analysis on Cryptographic Devices," 2010.

[6] I. Kizhvatov, "Side channel analysis of AVR XMEGA crypto engine," *Proceedings of the 4th Workshop on Embedded Systems Security*, p. 8, 2009.

Revision History

6MAR2013 – Update version for Blackhat EU. Adds more recent information, adds information about ChipWhisperer-Analyzer.

30NOV2012 – Updated version, includes information about firmware in target examples, updates figures to reflect this firmware, add SmartCard interface. This version on Blackhat EU CD-ROM.

15NOV2012 – Original version included on Blackhat Abu Dhabi CD-ROM