WhitePaper

How the analysis of electrical current consumption of embedded systems could lead to code reversing?

A practical approach of Power Analysis dedicated to reverse Engineering

Experimental content (no math!), proof of concept, tools, limits, protections and prospective

Authors
Yann ALLAIN (yann.allain[at]opale-security.com)
Julien MOINARD (julien.moinard[at]opale-security.com)

Brief Description/Abstract

The purpose of our study is to try to show how the analysis of electrical consumption of an embedded system enables us to find parts of the codes that it executes; this is done by presenting an operating mode, tools, a solid analysis, results, counter-measures and future research axes. It is all about trying to find another approach to the audit system. This approach aims at acquiring the code (reverse engineering) without having a physical access to the internal system components.

Our Whitepaper content will consist in making a quick presentation of the physical phenomenon at the origin of this type of information leak, confirming whether a sequence of instructions (opcode and data) can be found (reversed) by the analysis of electrical current used by the embedded system during the execution of a program., assessing then overcoming the technical difficulties in its achievement (Signal Acquisition, treatment and analysis, limits…), presenting a proof of concept and possible countermeasures to limit the risks.

Our intervention has a number of technical aspects linked to electric phenomena associated with the functioning of the electronic components, hardware hacking concepts etc…

To begin, we will present our purposes and the studies that are already published. Then, we will outline our approach, the targets we want to reach, the technical means necessary for the implementation of the analysis as well as a concrete example (Proof of concept). We will finish by a presentation of the means of limiting this type of attack and we will offer new orientations for future works linked to this field (Prospective)

Table of Contents

Introduction

The analysis of electrical consumption for a given system can be the cause of critical information leaks. Anglo Saxon terminology generally uses the expression: “Side Channel Attacks.”

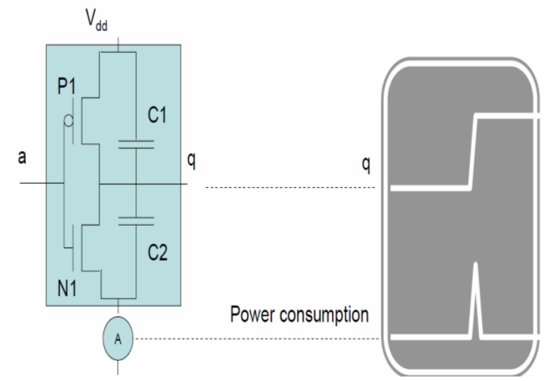
This sort of analysis is most often used to “find” keys in the encryption / decryption systems (Crypto processors, Smartcards…). There are a variety of methods to extract these codes: Simple Power Analysis (SPA), differential Power Analysis (DPA) …

The purpose of our experiment was to extrapolate on these methods in an attempt to find the code and the data executed by an embedded system and not just the algorithms or the encrypting keys.

Origin of the phenomenon

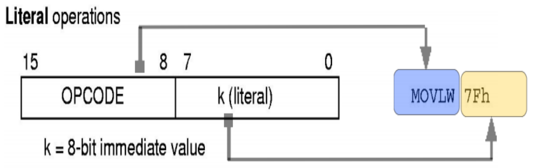
The technology used in microcontrollers/microprocessors is based on component units: The transistors; often in CMOS technology. These component units are grouped into logical functions. These logical functions deal with data and instructions. The treatment, implying the execution of an instruction or data manipulation, impacts the electricity consumption during transitions (passage from binary value 0 to binary value 1). As a consequence, current peaks are created.

See illustration below:



(Oswald, http://www.cs.bris.ac.uk/Research/Seminars/departmental/2007-03-29_DeptSeminar_Elisabeth_Oswald.pdf)

The consumption of an embedded system is therefore **theoretically** proportional to the number of bit transitions which will go from 1 to 0 or from 0 to 1 when code or data are processed. This phenomenon can be applied to data as well as to instructions which are also coded as bits.



(Microchip, http://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf)

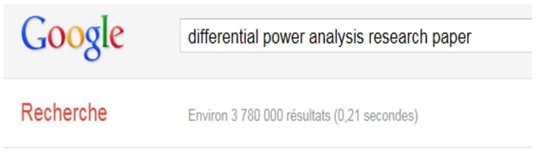
What is the interest of this experiment and why should we do this?

- Why not…
- To have an alternative from classical (and henceforth boring) XSS and SQL Injections attacks…
- It is not always possible to “open” a system to do audits: The clients can refuse the opening of an electronic system during an audit

- Anti-opening protections (Physical Tamper Resistance devices) are implemented and can have, as a consequence, the destruction of the program and of the data (Cf. payment terminals and CryptoSystems…)
- The physical accesses to codes can be protected by encryption systems which prevent (or slow?) the classical reverse engineering analyses (code extraction in EPROM or Flash memory…)
- The debugging hardware interfaces can often be suppressed from the systems when they are placed on the market. (no more JTAG access…)
- For fun…measure a current = read the code!

(Rapid!) Analysis of the pre-existing works on this topic

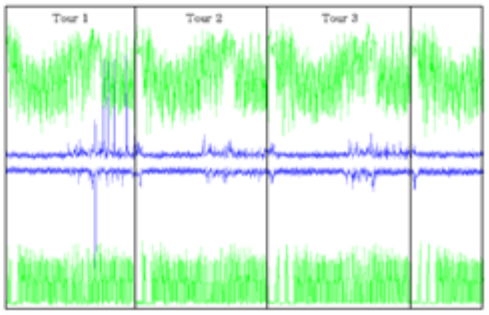
A large amount of research ("Whitepapers") and documents on attacks **aiming at finding encryption keys**: 3,780,000 answers in Google for only one type of attack!


Google differential power analysis research paper

Recherche Environ 3 780 000 résultats (0,21 secondes)

As we can see, the specific techniques to find an encryption key are widely published and accessible.

For instance, below is an extract of a publication which aims at showing the relation between a trace of the power consumption of a crypto processor and the execution of a DES algorithm.



(Clavier, http://www.prism.uvsq.fr/fileadmin/CRYPTO/these-cc-s.pdf)

Our bibliographical research (see details at the end of the document), which is certainly not exhaustive, seems to show that there are far fewer publications on the use of techniques of analysis of power consumption (power analysis) for reverse engineering.

However, we have "spotted" three interesting documents linked to our specific topic:

- The following article deals exclusively with the identification of instructions managed by a PIC (a well-known microcontroller): (Thomas Eisenbarth, http://math.fau.edu/~eisenbarth/pdf/SideChannelDisassembler.pdf)

- The following document underlines the uses of electricity analysis techniques to do some reverse engineering, but without revealing too many details. Furthermore, the aim is the discovery of information on the encryption keys: (Valette, http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lcr/dalemuva05.pdf)

- And finally, an example adapted to JAVACARDS technology: (Vermoen, http://ce.et.tudelft.nl/publicationfiles/1162_634_thesis_Dennis.pdf)

Most of these publications are **full of mathematical formulae**, which are more or less complex (from our point of view!)

E.g. : Inference of the secret by current analysis by correlation (!)

$$\rho_{WH'} = \frac{\mathrm{cov}(aH+b,H')}{\sigma_W \sigma'_H} = \frac{a}{\sigma_W}\frac{\mathrm{cov}(H,H')}{\sigma'_H} = \rho_{WH}\rho_{HH'} = \rho_{WH}\frac{m-2k}{m}.$$

$$\hat{\rho}_{WH}(R) = \frac{N\sum W_i H_{i,R} - \sum W_i \sum H_{i,R}}{\sqrt{N\sum W_i^2 - (\sum W_i)^2}\sqrt{N\sum H_{i,R}^2 - (\sum H_{i,R})^2}},$$

(Source: http://www.prism.uvsq.fr/fileadmin/CRYPTO/these-cc-s.pdf)

Finally, **the analysis of experiments/ documents existing on this subject highlights certain "shortcuts".** These shortcuts, that we could also call "experimental choices", do not question the conclusions presented by the authors. But **they can have an impact on the achievability in "real life" during a security audit**; for instance, we have noted:

- A decrease in the frequency used by the microcontrollers. This action is impossible (or quiet difficult) with no physical access to internals parts of the embedded system – so why boring with a highly difficult power analysis if they can "dump" the memory from the EPROM they have access to ;-)
- Elimination of the decoupling capacitors on electronic circuits (impossible if there is no physical access to the electronic components)
- Reduction of the analysis only to minor the length of keys or restrict the analysis to some and few instructions

What we think of this quick bibliographical analysis?

Point 1: The aim of the community of researchers therefore seems to be more centered on encryption issues (> 3 Million links vs. around 10 on Google for the aspects of reverse engineering)[1]. **The use of these techniques to extract the code seems to be a secondary issue in the authors' minds**…

Point 2: Can we achieve any of this without having a 12-year doctorate in mathematics? **Is there space for a more experimental approach?**

Point 3: Is it **really** possible to extract the executed code from an embedded system via the analysis of the power consumption?

[1] If we consider that the indexation obtained via search engines such as Google is representative… or not ;-)

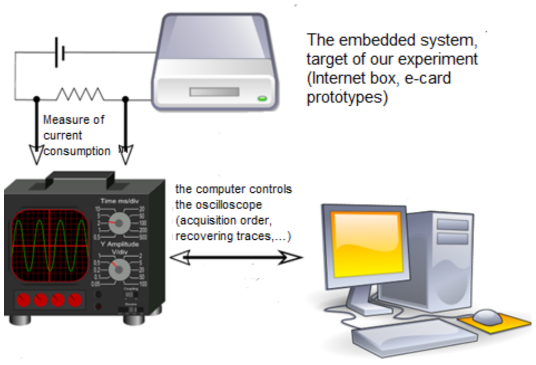
Presentation of our study

Reminder of the targets: our goal is to validate the possibility (or lack of one) of doing code reverse engineering through the analysis of the current consumption of an embedded system.

First, we need to find a way to acquire the electric signals

The acquisition process for electric signals (current, voltage)

Generally, the acquisition process for this type of analysis is the following:


The embedded system, target of our experiment (Internet box, e-card prototypes)
Measure of current consumption
the computer controls the oscilloscope (acquisition order, recovering traces,...)

Picture "freely adapted" from https://commons.wikimedia.org/wiki/File:Differential_power_analysis.svg

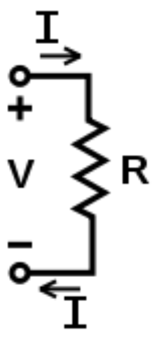
A simple resistance[2] "before" the embedded system makes this measure possible. But be careful, this resistance must be placed between the 0v and the embedded system's ground input! (If not, there is a risk of creating a short-circuit as soon as the measuring device is plugged in: another mass is created)

[2] See http://en.wikipedia.org/wiki/Electrical_resistance for more details on what is a "resistance"

Another possible choice is to use a differential sensor (more costly and more complex to implement) to note the difference in voltage across the resistance.

The working principle of the measure

The oscilloscope measures, and enables us to see the voltage between the resistance's fuse holders. The voltage is directly proportional to the current used according to Ohm's law: V (the voltage = R (resistance value in Ohms) x I (the value of the current in amperes)[3]



$$V = IR$$

(http://en.wikipedia.org/wiki/Ohm's_law)

Our first measurements show that the variations of these currents are extremely low and that is why we choose a resistance of 50 ohms to "amplify" the phenomenon (U= 50 X I)

[3] For more rigour, if the current varies the ohm law is written: U(t) = R * i(t) All measures become a function of time. R remains constant it is unnecessary to note the (t).

Some of the notions of measurement which influence the choice of measurement devices

According to Ohm's law we know that for a 1mA current we will have 50mV (via our 50 ohm resistance). So the **voltage we have to measure remains low** compared to the power supply orders for the embedded systems: A digital electric circuit is generally supplied in 3.3V and 5V -> our variations will therefore be around 1% of the general supply…

According to our first experiments, there is a voltage DC component which is "added" to the measured current. It actually reveals the average consumption of the prototype we used (PIC in our case).

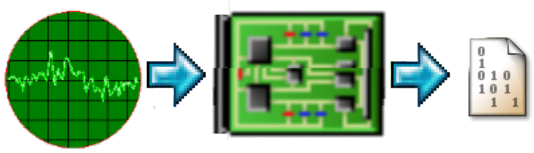
We are looking for variations around this value; we must not over-amplify the measurement (cf. the value of the resistance) as we will also increase this average voltage. The consequence of this increase would be to bring our signal beyond the range of input voltages that the oscilloscope can measure, and we would have a distorted signal.

When we launch a program in the embedded system, **the current variations are around 0.1mA (or** more or less 5mV[4] to 10mV to be measured).

[4] 1 mv = 0.001 Volt

How do we choose an oscilloscope adapted to this type of experiment?

To take this measurement, we are going to use a digital oscilloscope. The digital conversions require a sampling of the data. The choice of the oscilloscope will depend on the speed of the system, as it transforms (by conversion) an analogical signal (Voltage) into a digital value measured in 8, 16, 32 bits.

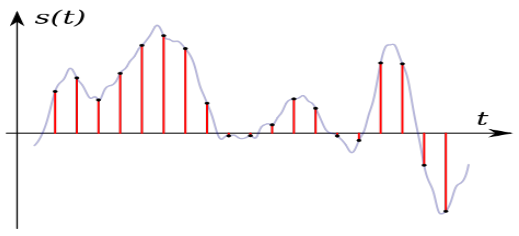


This digital value can be used for**:**

- Setting up displays (curbs, Traces…)
- Calculating (averages, maximum/ minimum values…)
- And much more (Fourier transformations…)

Be careful with the sampling speeds!

The sampling principles of an analogical signal for it to be converted to a digital signal need to follow Shannon's law:



Source:
http://en.wikipedia.org/wiki/File:Analog_digital_series.svg

"To avoid a signal being disturbed by the sampling, the sampling frequency should be superior to the double of the highest frequency contained in the signal"(*http://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_de_Shannon)*

To summarize, **if we choose an oscilloscope that does not take enough samples per second** (= number of analogic to digital conversions per second), **there is a loss of crucial information**.

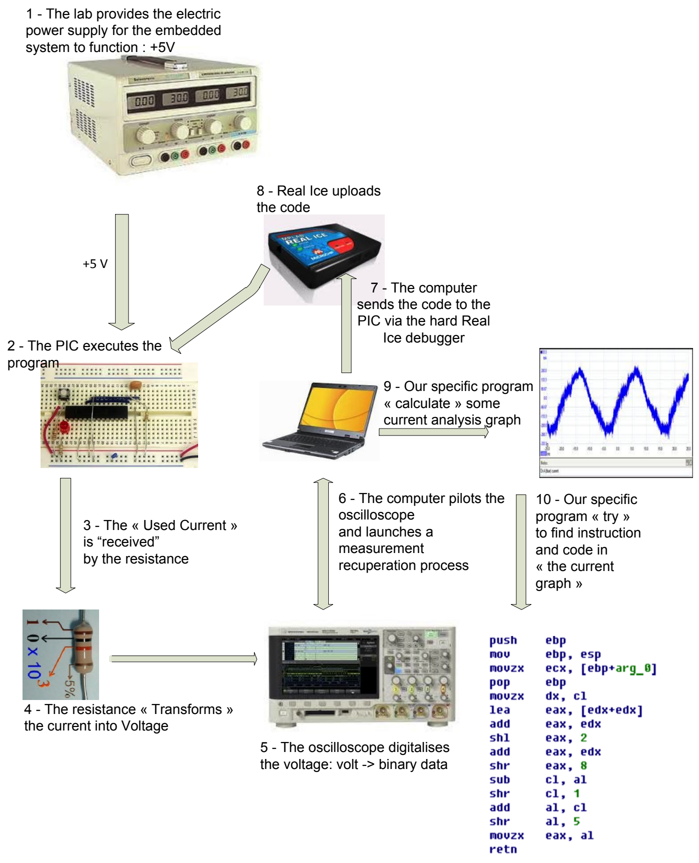
Within the framework of our experiment, this can impact:

• The quality of the measures, and therefore our capacity to spot electrical transitions (or not).
• The "repeatability" of the measurements (coherence of the measures between two trials).

In other terms, the oscilloscope never displays the same thing since it never sees (in fact it does not always measure) the same phenomenon (the transition is too fast and lacks synchronization).

After some unsuccessful attempts with cheap oscilloscopes (<€450, USB type…) which were not adapted to our needs, we chose to buy an oscilloscope from Agilent Technologies: More precisely, the model DSO3024A with a 2Gs/sec or 4Gs/sec sampling according to the model (around €4,000!).

Our experimentation system designed to create a “disassembler based only on the analysis of current consumption “


1 - The lab provides the electric power supply for the embedded system to function : +5V
+5 V
8 - Real Ice uploads the code
7 - The computer sends the code to the PIC via the hard Real Ice debugger
2 - The PIC executes the program
9 - Our specific program « calculate » some current analysis graph
3 - The « Used Current » is “received” by the resistance
6 - The computer pilots the oscilloscope and launches a measurement recuperation process
10 - Our specific program « try » to find instruction and code in « the current graph »
4 - The resistance « Transforms » the current into Voltage
5 - The oscilloscope digitalises the voltage: volt -> binary data

```
push    ebp
mov     ebp, esp
movzx   ecx, [ebp+arg_0]
pop     ebp
movzx   dx, cl
lea     eax, [edx+edx]
add     eax, edx
shl     eax, 2
add     eax, edx
shr     eax, 8
sub     cl, al
shr     cl, 1
add     al, cl
shr     al, 5
movzx   eax, al
retn
```

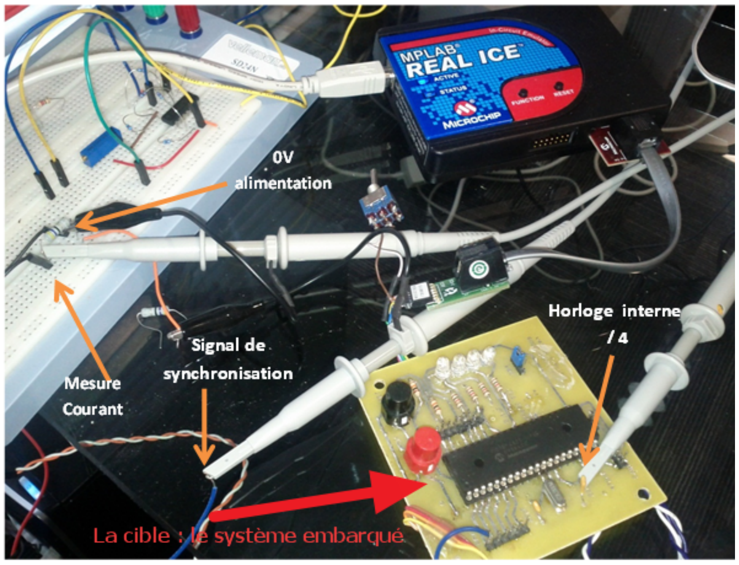
A bit of hardware!

Our tests use a “home-made” embedded system. It is based on a PIC18F4620 type microcontroller (Microchip). The embedded system’s function was to make the LED flash and to control the inputs/outputs. However, the use of the embedded system does not impact our experimentation.

List of components:

• Dso3024a Oscilloscope from AGILENT TECHNOLOGIE,
• A Windows 7 operated computer,
• A simple embedded system based on a MICROCHIP (PIC) microcontroller,
• A REAL ICE Programmer/ Debugger,
• We use the internal 1 MHz clock from the PIC,
• Laboratory electricity supply
• Some discrete components (resistance…),
• Test holed attachment plate (Breadboard)
• Various wires and other electronics stuffs

In reality this is what it looks like:



Principles of signal acquisition

Step 1: the REAL ICE programmer enables the upload of a program in the embedded system (in the PIC). It is used to send a code that we control to the embedded system.

Step 2: the execution of the program is launched on the embedded system (Run)

Step 3: during the execution, the code should cause a variation of the electricity consumption according to the instructions which have been executed and the data already treated. The resistance 'transforms' the used current into Voltage.

Step 4: this voltage is "Representative" of the consumption of the embedded system during the execution of a program. The oscilloscope's sensors recuperate this voltage

Step 5: the computer pilots the oscilloscope to start the measurements and recuperate the data (the digital conversion of the voltage by the resistance's fuse holders) -> [V(t),V(t1)…,V(tn)]

Step 6: a program on the computer gathers a number of voltage measurements. The same program calculates the differences between these voltage measurements and displays them as a graph.

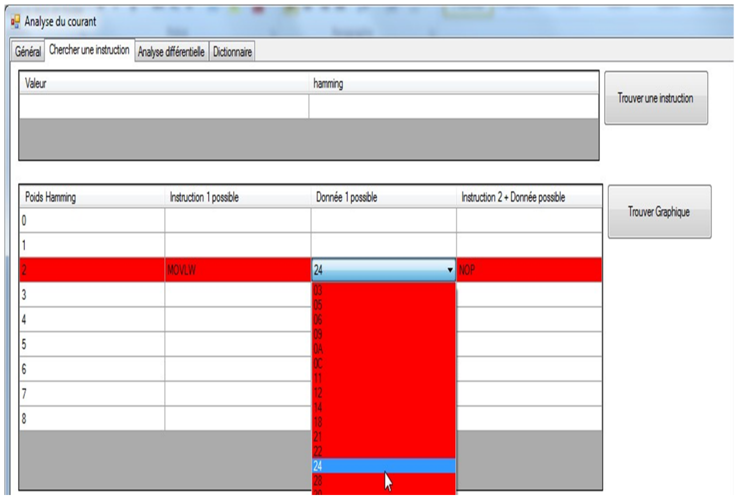
Note: all these measurements are "synchronized" with the embedded system clocks (cf. synchronization signal on the previous photo).

A little software too!

To take our measurements, we have developed a piece of software in VB.net which pilots the oscilloscope in order to:

•Acquire the averaged measurement of current.
•Differentiate 2 measurements of current.
•Display the measurement curbs.

GUI Screenshot

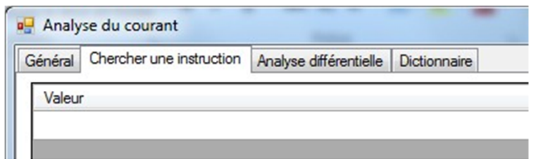


Valeur	hamming

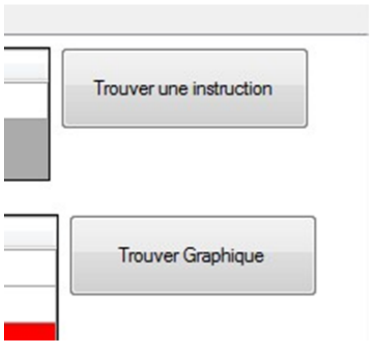
Poids Hamming	Instruction 1 possible	Donnée 1 possible	Instruction 2 + Donnée possible
0			
1			
2	MOVLW	24	NOP
3			
4			
5			
6			
7			
8			

Zoom on specific parts of the GUI

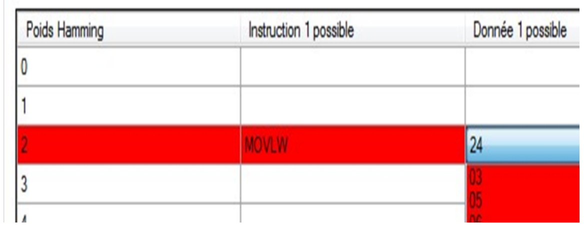
Several functions in Menu
"Find an Instruction", "Make a differential power analysis", "See and create dictionary"



Main Action Button for the user:
"Find instructions" (the disassembler like function), "Show graphical trace of current consumption associated"



Below, here is the “result window” of “disassembling”. It contains all type of instructions and data that could correspond to current consumption acquired



Poids Hamming	Instruction 1 possible	Donnée 1 possible
0		
1		
2	MOVLW	24
3		03 05

Our results

What we are going to do?

Reminder of our target: show the correlations between power consumption and the executed instructions and data processed.

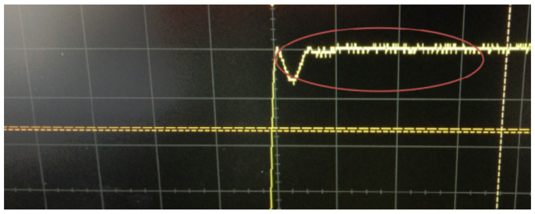
To begin, we are going to highlight the relation between current consumption and executed instructions.

Then we will show that the way the instructions are decoded by a microcontroller and how this impacts on the electricity consumption (because of the decoding pipeline)

Finally, we will demonstrate the effect of the bit values for data or instructions on consumption. Our purpose here will be to mention the notion of Hamming weight (Representation model of the electricity consumption according to bit value)

How we succeed to reduce parasites?

This type of low amplitude current measurement implies a large amount of parasites to be dealt with, and which can distort the measures (see below)



The solution to limit the impact of the latter is to calculate averages to prevent the imprecision of the measurement.

We have two possible choices:

- 1st choice: take frequent measurements and calculate the average thanks to the computer.
- 2nd choice: have this done by the oscilloscope itself.

Our choice was to leave these calculations to the oscilloscope, since for our 1st tests the recording capacities of our devices are sufficient. We are going to attempt to find 2 or 3 instructions only (so around ten clock cycle).

So our program just has to go round in loops to have a periodic current consumption.

How to make those loops?

Trigger a reset on the embedded system regularly (Power off, Power on)

Make a test assembly programs that use a loop (to repeat the same instructions cycle)

For our experiments, we chose the last solution (easier to manage)

The oscilloscope has thus been set to calculate an average on 8000 measures.

All the current curbs that you will see will therefore be averages. This enables us to have results that are **easy to reproduce and relatively precise** (**with few parasites**)

Measurement 1: Analysis of a program with NOP instructions

For this measure, we download a program in the Microcontroller. It contains:

• 4 `nop` instructions. The `nop` instruction corresponds to an assembler's instruction which does not do any operation (no operation)

• 2 assembler instructions commanding one of the microcontroller outputs. These two instructions control the value of one microcontroller's pin. They enable the positioning of its value to 1 or to 0. It is a question of the creation of a synchronization signal enabling us to know when the 4 `nop` instructions have been executed. This signal is sent toward the synchronization inputs on the oscilloscope.

Program 1

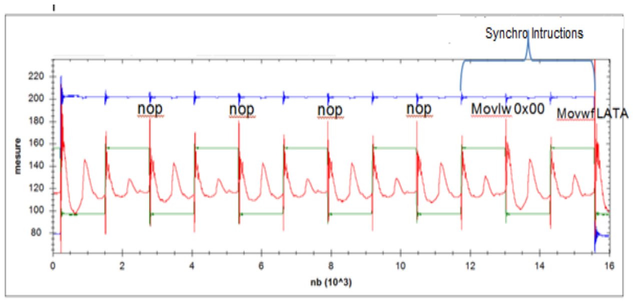
```
nop
nop
nop
nop
+
Synchronization instructions
```

This program is executed in loops on our embedded system. Here is the trace that we get on one loop.

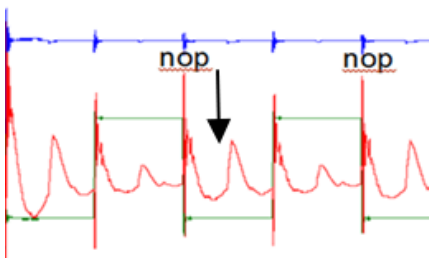
•In red, we measure the used current during the execution.
•In Blue, we have our synchronization signal (which goes to zero to the end of the graph)
•In Green, we visualize the clock of the embedded system

The graph below corresponds to the execution time of our 4 instructions `nop` + 2 instructions for synchronization.

This graph is visualized on our oscilloscope or inside our specific GUI



If we make a zoom



Conclusion 1: we can find instructions and codes inside a current consumption trace with a practical approach

The above trace reveals an obvious and repetitive link (the peaks are in red) between the execution of the code and the electricity consumption. The shape and the periodicity in the consumption time shows that the instructions executed at the moment of each tick of the clock (timing).

So it is possible to find a correlation between the execution of a code in the embedded system and the electricity consumption by a simple and practical approach.

We can see that **we can detect <u>where</u> the instructions are just by analyzing the shape** of the trace of the used current. But we are still not able, for now, to translate this trace into instructions (the value corresponding to the measured trace)

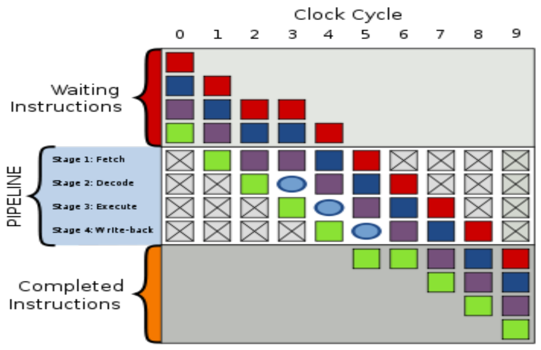
Measure 2: Influences of the Pipeline for reversing instructions and its drawbacks for our measurements

Technical note on what a pipeline is:

Most of microcontrollers use a pipeline. According to the Wikipedia definition, a pipeline is "*one of the elements of an electronic circuit in which data advance one after the other to the rhythm of the clock signals. In the microarchitecture of a microprocessor, it is more precisely the element in which the instruction execution is divided into stage*".

The purpose of the pipeline is (still according to Wikipedia): *"…a concept inspired by the functioning of an assembly line. Let's consider that the assembly of a car is composed of three stages: installing the engine - installing the bonnet - fixing the tires (in this order, with maybe intermediary stages). A car on this assembly line can only be in one position at any given time. Once the engine is installed, the car Y continues for the bonnet to be installed, leaving the "engine" position available for a car X. The car Z is having the tires fixed (Wheels) whilst the second car (Y) is at the bonnet stage. Simultaneously, the car X is starting the engine phase. If the installation of the engine, the bonnet and the wheels take - respectively – 20, 5 and 10 minutes, the completion of three cars will take (if they follow one another on the assembly line) 105 minutes (1h45)=(20+5+10)x3=105. If we place a car on the assembly line as soon as the level where the car should be is free (pipelining principle), the total time to make the three cars will be of 75 minutes (1h15)…"*

The purpose of the pipeline is therefore to allow a quicker execution of the instructions within a microcontroller or a microprocessor. (See illustration below)



Source : http://upload.wikimedia.org/wikipedia/commons/thumb/6/67/Pipeline,_4_stage_with_bubble.svg/350px-Pipeline,_4_stage_with_bubble.svg.png

Let's return to our experiment,

Within the framework of this new measurement, we are going to make **the difference in current consumption between** a program which executes 4 `nop` instructions (cf. measure 1 of the previous chapter) and a new program containing other instructions. For instance, a `movlw 0x00` :

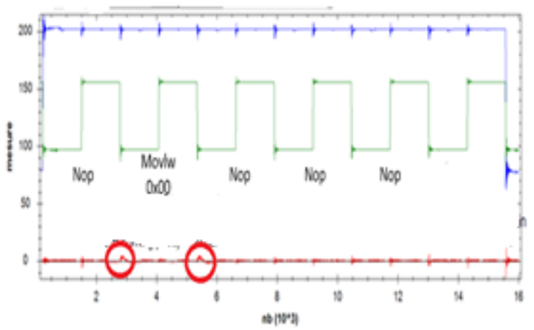
Program 2

```
nop
movlw 0x00
nop
nop
+
Synchronization
instructions
```

This measure aims to find the difference between the electricity consumption for the program 1 (`nop` only) and the electricity consumption for the program 2 (`nop + one mov intruction`).

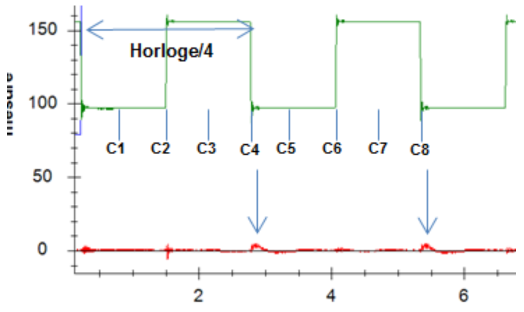
It is calculated by the program which pilots the oscilloscope. The two measures of the oscilloscope come in two charts, we memorize the values and then the program makes the different between each point.

Below, we show the trace corresponding to the difference in consumption between the programs 1 and 2:



In red trace above, the 2 circled small current's peaks represent the consumption which is theoretically proportional to the number of bits transitions which are going from 1 to 0 or from 0 to 1 : in our case, it's correspond to the number of bits of `nop` and a `movlw` instructions.

Let's zoom in this trace,



C1, C2…, C8: represents the steps for decoding an instruction on a PIC: An instruction is executed every four clock cycles on this type of microcontroller. Each cycle corresponds to specific decoding step (this is the pipeline!) . In comparison to the program 2 (`nop, movlw 0x00, nop, nop`), this is how the instructions are dealt with on the pipeline

C1	C2	C3	C4
Decoding	Read k here 0x00 (`movlw 0x00` ou k =0)	CPU Calculation	CPU write the work in registers

Reminder: the consumption is theoretically proportional to the number of transitions of the bits which will move from 1 to 0 or from 0 to 1 (cf. chapter “origin of the phenomenon”)

In our situation, the transitions are the following

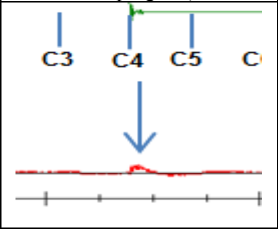
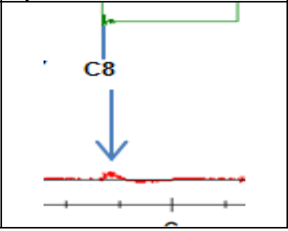
`Nop` instruction binary encoding is

0000 0000 0000 0000

`movlw 0x00` instruction is

0000 1110 0000 0000

So, I we make a zoom on peaks on latter graph, we have

1st peak	2nd Peak
This current peak linked to the decoding of the MOV during the first cycle of execution of the NOP (1st NOP in the program)	This current Peak **(wave form is identical!)** linked to the decryption of another NOP during the execution cycle of the MOV
	

Analysis of the above trace, and highlighting of the influence of the pipeline on consumption

•In C4 we write the result of the operation in the work register, but the microcontroller does not actually execute anything, as the `nop` does not have a result.

•We can observe an electricity peak in the 4th cycle. However, the `nop` instruction does not write in any register, so why do we have a power peak?

•In a first analysis (without taking into account the way the pipeline works), we should have had it in C5 if we had had four instructions per cycle. It is the principle of the functioning of the microcontroller’s Pipeline which is already looking for the following instruction in the ROM to fit it into a register that can be read by ALU (arithmetic and logical unit).

•The electricity peak in C4 is due exclusively to the decryption of the instruction `movlw` (because of the pipeline)

•In C8, as there is a `nop` after the `movlw` (encoding only with 0s), we always have the same variation (= same number of bits coming through which go from 1 to 0 on the microcontroller’s internal register: **so we measure the same peak twice while the microcontroller decode two different instructions!**

Conclusion 2: the power measurements taken at a given time depend on the previous instructions executed and data processed

As we see, the power measurements taken at a given time depend on the previous instructions. Indeed, the latter are dealt with by the microcontroller’s pipeline in advance of the stages (before the actual execution). It is **a major problem** that can quickly limit (or at least complicate) the extraction of the code by an embedded system’s analyses of current **consumption** … But all is not lost! (cf. chapter below)

Measure 3: Influences of the bits values on current consumption (Hamming weight!)

The difference in instructions or in data impacts current consumption of current. This impact is directly proportional to the bit value for the instructions (hexa values for instructions) or for data (value of the data) and mainly for transitions: this means the number of bits which go from 1 to 0 (or the contrary) between two ticks of the clock.

This concerns the idea of Hamming weight:

For instance, the two following byte 0 1 0 1 0 1 0 1 and 0 0 0 0 0 1 1 1 have a hamming weight of 3 (there are 3 different bits at value=1)

The greater the weight (in relation to two sets of data to be compared) the higher the electricity consumption will be.

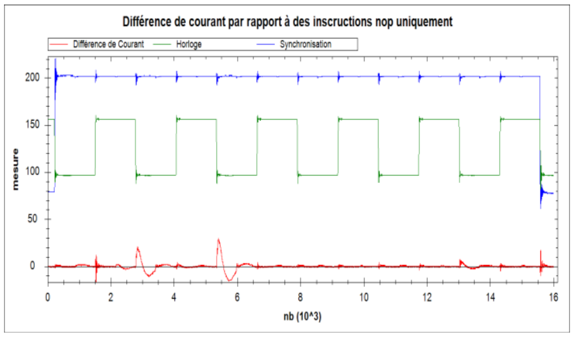
There are existing electric consumption models based on the Hamming notion of distance (see reference below)
(Jie Li et al., http://www.scientific.net/AMM.121-126.867)

Below we have a practical demonstration:

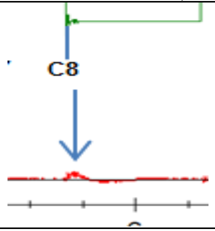
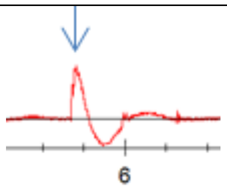
Let's compare the consumption of a program with 4 `nop` and a 2nd program with `nop - movlw 0xFF - nop - nop`

Encoding of the `nop` instruction
=> 0000 0000 0000 0000
for the instruction `movlw 0xFF`
=> 0000 1110 1111 1111

Measurement graph is



In relation to the second program (which contained a `movlw 0X00`), we can see a difference in measurements linked to the difference in the number of 0 bits and 1 bits between the two instructions

Electricity peak linked to the `movlw 0x00` (cf. measurement 2)	Electricity peak linked to the `movlw 0xFF` (cf. measurement 3)
	

Conclusion 3: There is dependence between the values of data and instructions in relation to the measured consumption

Therefore we have a validated dependence between the values of data and instructions in relation to the measured consumption

Global interpretations of our 1st set of results and limitations showed

It therefore seems possible for us to "find" the data and the instructions in the traces of the electric consumption.

However, creating a disassembler is more complex as all the measurements always depend on the instruction which had previously been decoded (because of the pipeline)

How to progress regarding our objectives? (See below)

Is there a solution to improve our "disassembler" based only on the analysis of current consumption?

Create a dictionary: we applied the rainbow table principal to memorize a "footprint" of current consumption for each pair of instructions that could be executed

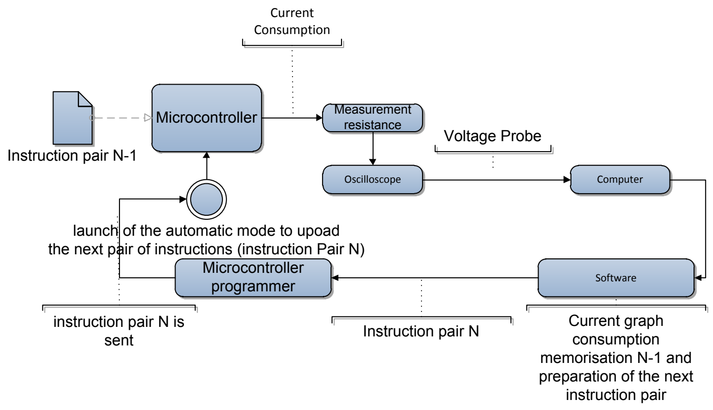
We need to create a dictionary of current consumption for each combination of possible pair instructions for a specific embedded system (rainbow table principal)

Here are some ideas that we are going to experiment in order to advance in this study. The goal is to create a disassembler which would be based only on the analysis of used current to "find" the code or data which is executed on an embedded system.

The main problem is the sequential aspect: state/previous instruction which impacts the used current at a t+1 time.

The idea is to memorize a signature of electricity consumption for each pair of consecutive instructions in an exhaustive way. The idea is to create a sort of dictionary (this principle is similar to a pre calculated hash tables or rainbow tables).

To create these dictionaries, the principle is the following:



Current Consumption
Instruction pair N-1
Microcontroller
Measurement resistance
Voltage Probe
Oscilloscope
Computer
launch of the automatic mode to upoad the next pair of instructions (instruction Pair N)
Microcontroller programmer
Software
instruction pair N is sent
Instruction pair N
Current graph consumption memorisation N-1 and preparation of the next instruction pair

In the follow up of this analysis, for more simplicity[5] , we will only look into instructions which last just one machine cycle. If, as a minimum, we want to find all the possible pairs of two instructions with the matching data, we need 256^2 or 65536 measurements.

Then, we just need to compare the current consumption "footprint" of

[5] This our "experimental choice ! »

an “audited” system with the dictionary we have created.

This dictionary will only enable us to distinguish a list of 2 instructions, so it then becomes obvious that to carry out those measurements properly, we will have to continue developing our software to be able to “find” more instruction.

However, as rainbow tables take time to generate, our current consumption dictionary too!

Moreover, the programs that create the dictionary must be able to synchronize the signals on its own but more particularly to send the right program to the microcontroller before the measurement is sent to extract the electricity signature. Thus we create an automatic mode.

But finally, there is no interest in creating all the instruction couples (for a proof of concept ;-) , because this type of dictionary will be very long in spite of our software which automates this task.

We must not omit the Hamming weight. In truth we only need to make a Hamming weight related dictionary if we take the example of our three `nop` with a `movlw` instruction that we want to identify:

As an example of Hamming weight equal to 1 we have `movlw 1, 2, 4, 8, 16, 32, 64, 128` so we only take the “footprint” of one of these instructions, then for the weight of 2 we have `movlw 3, 5, 6, 9, 10, 12`… we soon realize that if we proceed like this, our dictionary will be quicker to create but will be far less precise, according to the instruction we looked for.

For instance, the 0 has a 0 weight and it is the only one. The 255 (0xFF) is also the only one to have the 8 bits at 1. The Hamming weight 1 only has 8 values. But let's see a summary of this in the table below, to have a better understanding.

Hamming Group	**Number of instruction or data value by hamming groups**
0	1
1	8
2	28
3	56
4	70
5	56
6	28
7	8
8	1

We soon realize that according to the Hamming weight of the instruction we are looking for, the value we have found has variations in precision.

For the following of our study we have created a dictionary of all possible permutations of program that’s included instructions and data with `nop` and `movlw xx`.

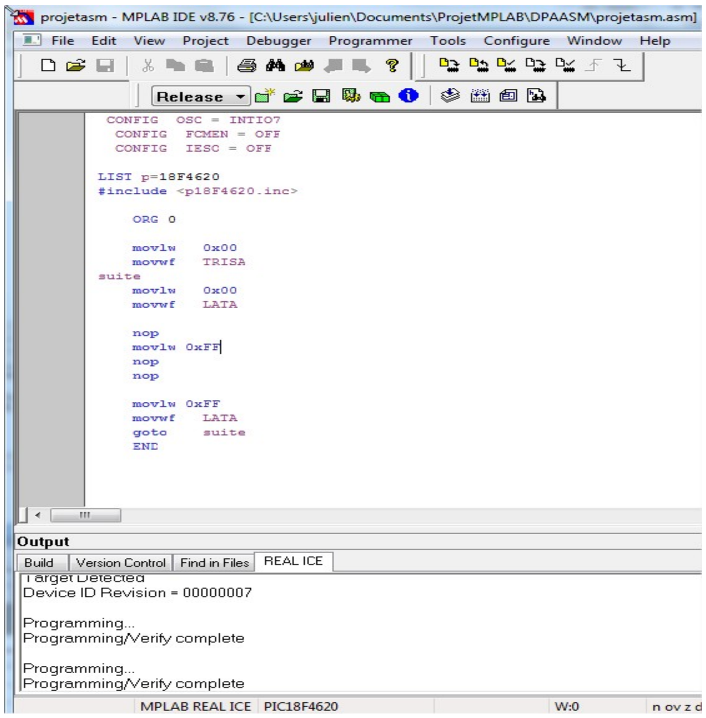
We need to mention that the use of a dictionary imply that our method could only be adapted to reverse the code of embedded system based on well know board or ready to use system (FGPA based board, Developpement board, Pre designed embedded system board…). Why? Because, we need to be able to create a dictionary. And for that, we need to upload our X Pair instructions as described above…

Examples of instruction discovery with our “ultra-basic disassembler based only on the analysis of current BUT with the use of our dictionary”

Measurement 4: How to find an unknown instruction inserted after 3 `nop` with this technic?

At first, we program the microcontroller with an instruction pair which is available in the dictionary previously created. So, our software will try the match the “unknown instruction” between `nop` (in our case a `movlw 0xFF` but our “disassembler” don’t know it!)

Here is the program with the “unknown instruction” that we will “upload” to the PIC


projetasm - MPLAB IDE v8.76 - [C:\Users\julien\Documents\ProjetMPLAB\DPAASM\projetasm.asm]
File Edit View Project Debugger Programmer Tools Configure Window Help
Release
Output
Build Version Control Find in Files REAL ICE

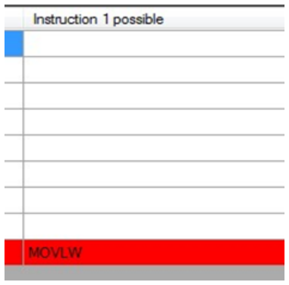
```
 CONFIG  OSC = INTIO7
  CONFIG  FCMEN = OFF
  CONFIG  IESC = OFF

LIST p=18F4620
#include <p18F4620.inc>

    ORG 0

    movlw   0x00
    movwf   TRISA
suite
    movlw   0x00
    movwf   LATA

    nop
    movlw 0xFF
    nop
    nop

    movlw 0xFF
    movwf   LATA
    goto    suite
    END
```

```
Target Detected
Device ID Revision = 00000007

Programming...
Programming/Verify complete

Programming...
Programming/Verify complete
```

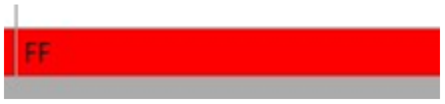
MPLAB REAL ICE PIC18F4620 W:0 n ov z d

Then, we launch the software to capture the current: and we launch the graphical instruction search (a result which is easier to interpret because very visual)

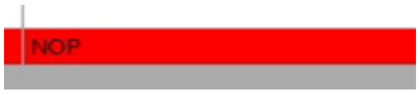
The software found the “unknown instruction”



The Data found is



And the “next instruction” found is



The program has analyzed the current and has inferred the executed instruction.

Here we are talking of a `movlw 0xFF` followed by a `nop`. According to our tables of hamming groups, this result is 100% true, the program only proposes FF as a solution.

However we are going to make another attempt.

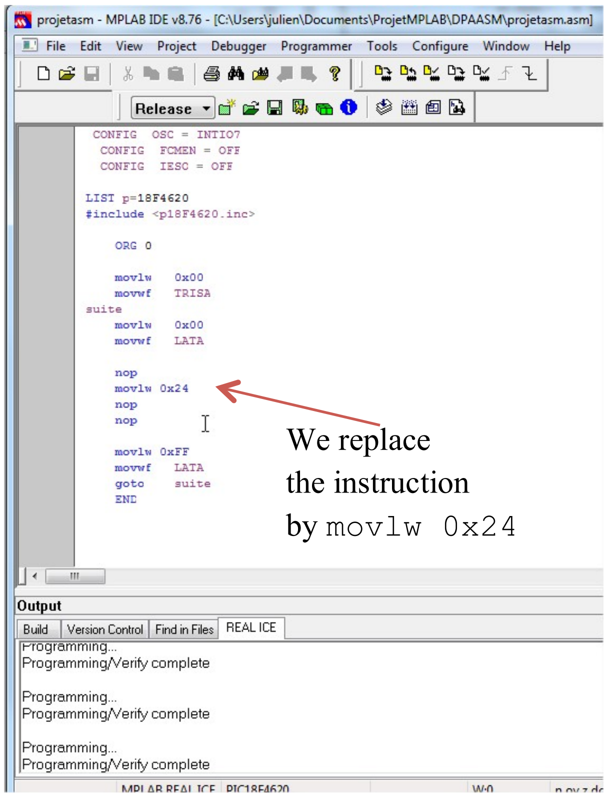
For instance, we try to find the following couple instruction `movlw 0x24, nop`

The hexadecimal number 24 equals in binary

0	0	1	0	0	1	0	0

That corresponds to a Hamming weight of 2, let's see what the "dissassembler" gives us:

Here is the program with the "unknown" instruction that we will "upload" to the PIC


projetasm - MPLAB IDE v8.76 - [C:\Users\julien\Documents\ProjetMPLAB\DPAASM\projetasm.asm]

File Edit View Project Debugger Programmer Tools Configure Window Help
Release

```
 CONFIG  OSC = INTIO7
  CONFIG  FCMEN = OFF
  CONFIG  IESC = OFF

LIST p=18F4620
#include <p18F4620.inc>

    ORG 0

    movlw   0x00
    movwf   TRISA
suite
    movlw   0x00
    movwf   LATA

    nop
    movlw 0x24
    nop
    nop

    movlw 0xFF
    movwf   LATA
    goto    suite
    END
```

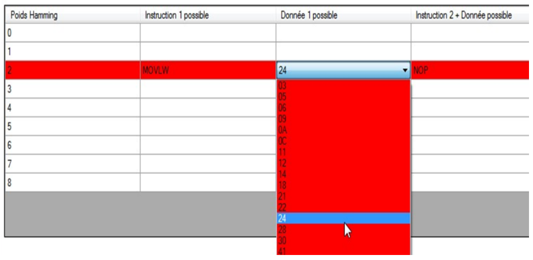
We replace the instruction by `movlw 0x24`

Output
Build
Version Control
Find in Files
REAL ICE

```
Programming...
Programming/Verify complete

Programming...
Programming/Verify complete

Programming...
Programming/Verify complete
```

We launch again our "disassembler"

The software results are

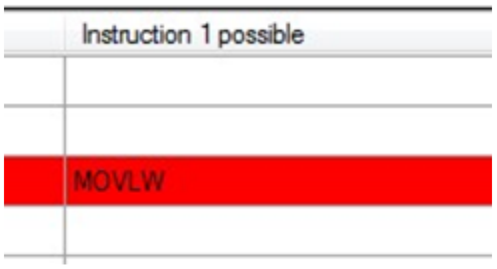


Poids Hamming	Instruction 1 possible	Donnée 1 possible	Instruction 2 + Donnée possible
0			
1			
2	MOVLW	24	NOP
3			
4			
5			
6			
7			
8			

```
03
05
06
09
0A
0C
11
12
14
18
21
22
24
28
30
41
```

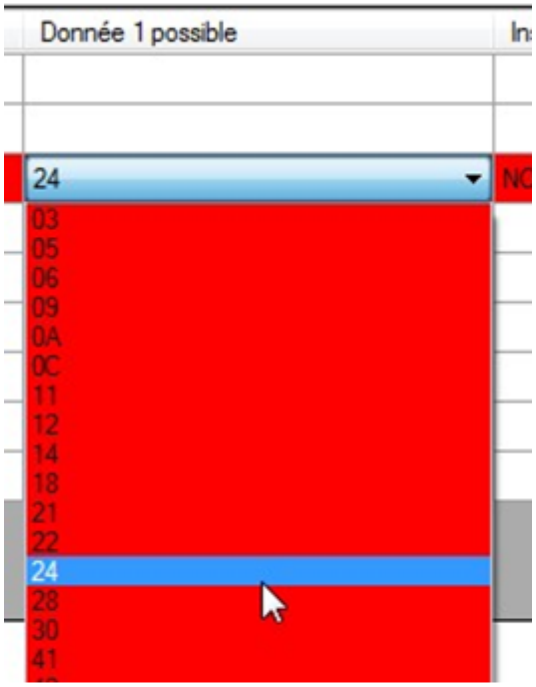
If we make a zoom on GUI,

Instruction found by the program is

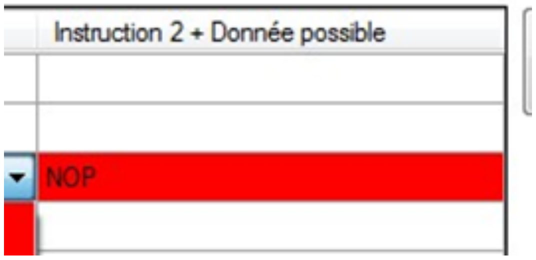


Instruction 1 possible
MOVLW

Data found is in the hamming group of 2 (with contain 28 possibilities)



And the “next instruction” found is

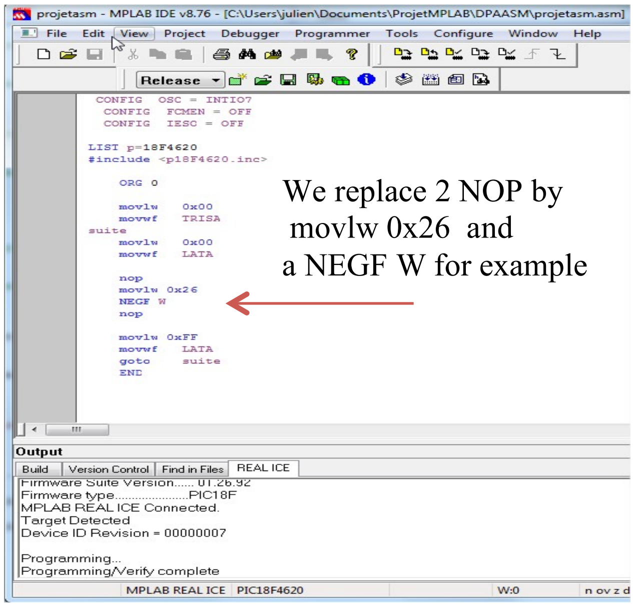


The program has inferred an instruction with a Hamming weight of 2 for the data.
But remember, a Hamming weight of 2 also means 28 possible instructions. But we have our 0x24 in this Hamming Group. So we are still “good", but less accurate.

We will continue with a more complicated case

Measurement 5: How to find 2 “unknown instructions” inserted inside a list of `Nop`?

We will therefore program our microcontroller with two instructions which are in our dictionary.


projetasm - MPLAB IDE v8.76 - [C:\Users\julien\Documents\ProjetMPLAB\DPAASM\projetasm.asm]
File Edit View Project Debugger Programmer Tools Configure Window Help
Release

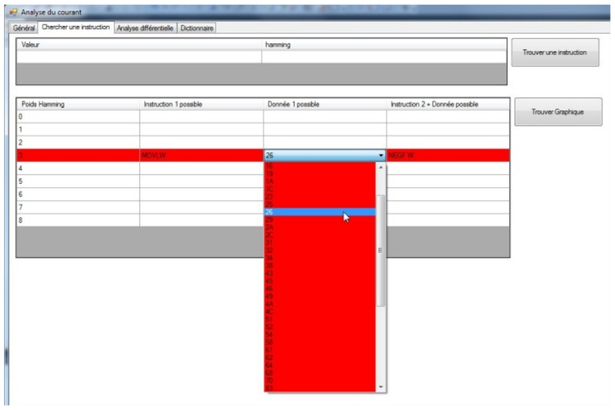
```
 CONFIG   OSC = INTIO7
  CONFIG   FCMEN = OFF
  CONFIG   IESC = OFF

LIST p=18F4620
#include <p18F4620.inc>

    ORG 0

    movlw    0x00
    movwf    TRISA
suite
    movlw    0x00
    movwf    LATA

    nop
    movlw 0x26
    NEGF W
    nop

    movlw 0xFF
    movwf    LATA
    goto     suite
    END
```

We replace 2 NOP by
movlw 0x26 and
a NEGF W for example

Output

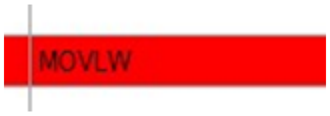
```
Firmware Suite Version...... 01.26.92
Firmware type....................PIC18F
MPLAB REAL ICE Connected.
Target Detected
Device ID Revision = 00000007

Programming...
Programming/Verify complete
```

We re-launch the software to get an analysis

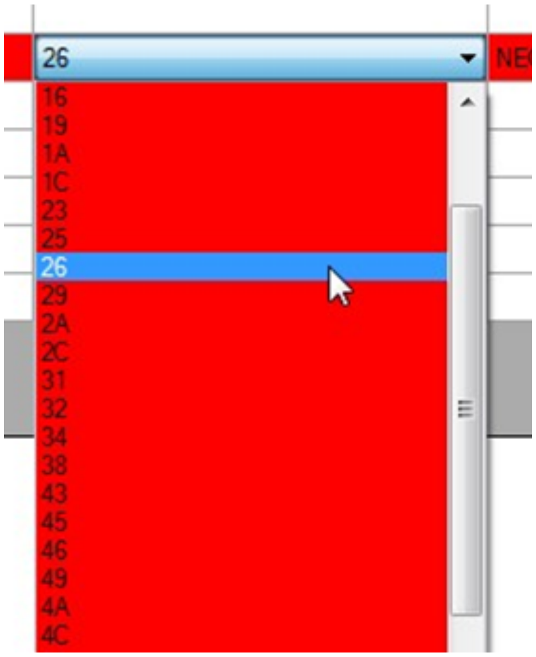


If we zoom in the GUI

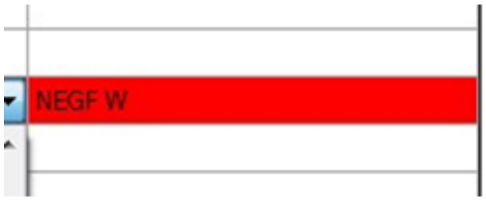
The “dissassembler” found the following intruction



And the following possible data



And finally the “Next instruction”



Thus, our application is able to find the pair `movlw 0xDD` (where DD could be one of the possibility in the hamming group (which include de 0x26 value!) followed by a `NEGF W : Good !`

Global conclusion

We are perfectly capable of finding certain instruction pairs. It is rather encouraging but according to the type of instruction and of data, groups have formed (accuracy decreased!)

However, to be able to design a complete disassembler with this type of method, we need to overcome some issues regarding several specific set of instructions: Branch and Jump instructions, I/O manipulation instruction, more than 1 cycle instruction. The influence on current consumption for those later would be different for sure (further investigation need to be scheduled!)

We need to mention that the use of a dictionary imply that our method could only be adapted to reverse the code of embedded system based on well know board or ready to use system (FGPA based board, Developpement board, Pre designed embedded system board…). Why? Because, we need to be able to create a dictionary. And for that, we need to upload our X Pair instructions as described above…

How can we move further?

We have to find another method to subdivide the Hamming groups even further in order to obtain increased accuracy.

Maybe that could be done by using another physical phenomenon such as the fact that all the memorization latch (or storage flip flop) (transistor based) do not commute simultaneously. It could be possible, but it must be synchronized. It will probably be very difficult to identify this structure. To be followed up. (A more advanced submission on another conference ;-)

Some solutions for protection against this type of attack

There are a variety of countermeasures. For instance, those which are used in the field of encryption key protection. Licenses have already been submitted for counter measures for electricity analysis, measures against the appropriation of keys (cf. notably for the site: http://www.cryptography.com).

Here is a non-exhaustive list of certain types of counter-measures:

- Leakage reduction: there are techniques to make the totality of a sequence of operations independent of the key as well as the balancing techniques for hardware and software, in order to reduce the variations in energy consumption for different sets of data.

- The introduction of noise: there are techniques enabling to allow different types of noises to "interfere" with the measures of energy consumption available for the attacker.

- The incorporation of random events: these are randomization techniques for the data manipulated by the device.

In the context of our study, the creation of a microcontroller or of microprocessors with integrated internal protections could be very costly (with the necessity of adding hardware elements). However, the integration of protection solutions in the FPGA software processors seems more easily achievable as they already have programmable elements.

So one solution would be to create a "software processor" with integrated protections, knowing that the "creation" of this type of "soft-core" processor is exclusively based on programming (FPGA principle)6

[6] next conference?

Annex 1: Speakers' resume

Yann ALLAIN, founder and current director of the OPALE SECURITY company (www.opale-security.com). He graduated from a computer and electronic engineering school (Polytech - Université Pierre et Marie Curie). After a time in the electronic industry as an engineer in embedded system conception, he made a career move towards ICT. He started as a production manager for a company in the financial sector (Private Banking), and evolved towards ICT security when he became part of the ACCOR group. He was in charge of applicative security for the group. He has an 18-year experience, 14 of which dedicated to IT system and embedded system security. OPALE SECURITY are security consultants who deal with research projects linked, amongst other things to the security of embedded systems (http://www.opale-security.com/innovation-securite-systemes-information.html)

Julien MOINARD, an electronics technician with a solid background in this field (over 7 years) associated with many personal and professional experiments in the field of microcontrollers. Furthermore, he contributes to training 1st year students in an electrical engineering and industrial computing DUT (2-year technical degree). He is in the 2nd year of this program.

Annex 2: Bibliography and references

Clavier, C. (http://www.prism.uvsq.fr/fileadmin/CRYPTO/these-cc-s.pdf). *De la séecurité physique des crypto-systèmes embarqués.*

Jie Li et al., 2. A.-1. (http://www.scientific.net/AMM.121-126.867). *Hamming Distance Model Based Power Analysis for Cryptographic Algorithms.*

Microchip. (http://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf). *PIC18F4520, see Datasheet.*

Oswald, E. (http://www.cs.bris.ac.uk/Research/Seminars/departmental/2007-03-29_DeptSeminar_Elisabeth_Oswald.pdf). *Power Analysis Attacks.* Computer Science Department: University of BRISTOL.

(s.d.). *Source: http://www.prism.uvsq.fr/fileadmin/CRYPTO/these-cc-s.pdf.*

(s.d.). *Source: http://www.prism.uvsq.fr/fileadmin/CRYPTO/these-cc-s.pdf.*

Thomas Eisenbarth, C. P. (http://math.fau.edu/~eisenbarth/pdf/SideChannelDisassembler.pdf). *Building a Side Channel Based Disassembler.*

Valette, R. D. (http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lcr/dalemuva05.pdf). *Side Channel Analysis for reverse Engineering (SCARE).*

Vermoen, D. (http://ce.et.tudelft.nl/publicationfiles/1162_634_thesis_Dennis.pdf). *Reverse engineering of Java Card applets using power analysis.*