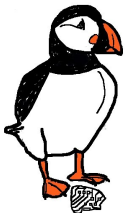

PUFFIN

Physically unclonable functions found in standard PC components

Project number: 284833
FP7-ICT-2011-C

D1.1

Scientific contribution of WP1, part 1 Exploration

Due date of deliverable: 31. July 2013
Actual submission date: 30. September 2013

WP contributing to the deliverable: WP1

Start date of project: 1. February 2012 Duration: 3 years

Coordinator:
Technische Universiteit Eindhoven
Email: coordinator@puffin.eu.org
www.puffin.eu.org

Revision 1.0

Project co-funded by the European Commission within the 7th Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission services)	
RE	Restricted to a group specified by the consortium (including the Commission services)	
CO	Confidential, only for members of the consortium (including the Commission services)	

Scientific contribution of WP1, part 1
Exploration

Daniel J. Bernstein (TUE)
Tanja Lange (TUE)
André Schaller (TUD)
Peter Simons (IID)
Anthony van Herrewege (KUL)

30. September 2013
Revision 1.0

The work described in this report has in part been supported by the Commission of the European Communities through the FP7 program under project number 284833. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Abstract

This document summarizes the scientific contribution of Work Package 1 (WP1) during the first phase (18 months) of the PUFFIN project.

Keywords: WP1, Microchip PIC16F1825, STMicro STM32F100R8, TI MSP430F5308, Atmel ATMega328p, PandaBoard, NVIDIA GTX 295

Contents

List of Figures

Chapter 1

Introduction

WP1, the Exploration work package, searches for new ways to physically identify PCs and other commodity hardware. It focuses on standard PCs, handheld devices, and embedded systems as they actually exist today and in the foreseeable future. The goal is not to modify components to make them easy to identify; the goal is to find identifiers that are already intrinsic in existing mass-market hardware. The most productive explorations so far have been of uninitialized SRAM, discussed below in more detail.

WP2 is responsible for comprehensive data analysis. However, WP1 is responsible for making a preliminary assessment of the quality of data obtained from hardware, so that WP2 can focus on the most interesting data.

WP1 has, upon request from WP3, broadened its explorations to search for randomness, and for nondeterministic behavior in general, even if the source of randomness does not seem suitable for use as an identifier. For the case of SRAM this means that even small amounts of uninitialized SRAM are potentially useful, even if those amounts are clearly not enough to support robust identification.

WP1 has successfully read out uninitialized memory from a surprisingly wide range of processors, ranging from tiny embedded processors up to graphics cards costing 500 EUR. This document describes WP1's exploration of these processors, and in particular the challenges that WP1 faced in accessing the memory on these processors.

1.1 Exploration vs. utilization

A common theme in WP1's SRAM investigations is that, after finding a potentially usable bank of SRAM, WP1 builds tools to copy the SRAM to a general-purpose laptop CPU for further analysis.

These short-term tools should not be confused with the more complex long-term tools produced by WP3. In some cases the short-term tools are a starting point for building the long-term tools, but the primary purpose of the short-term tools is for WP1 to collect data, while the long-term tools are meant to provide security for various applications. In many cases the long-term tools, when installed, will *prevent* exactly the types of copying performed by the short-term tools.

Chapter 2

Microcontroller SRAM

In this chapter, we first describe the methods used to extract SRAM start-up data from four different types of microcontrollers: Microchip PIC16F1825, STMicro STM32F100R8, Texas Instruments MSP430F5308, and Atmel ATMega328p. Afterwards, we give details about the SRAM extraction specific to each type of microcontroller.

2.1 Firmware design

Our general strategy for copying uninitialized SRAM data out of a microcontroller is as follows. The microcontroller is programmed with firmware that, on power-up, initializes the serial port and then starts transmitting the value of each SRAM byte in sequence; once finished, it enters an idle loop. Care is taken in this firmware to avoid overwriting any of the SRAM storage. This is easy to achieve on microcontrollers that have several general-purpose registers to store variables, such as a pointer to the current SRAM byte. However, some microcontrollers, such as the Microchip PIC16 family, have only a single general-purpose register. To avoid overwriting SRAM on these microcontrollers, we store some variables in unused configuration registers.

2.2 Hardware setup

We obtained initial measurements of SRAM power-up patterns as follows. We manually connected the power lines and serial port of the target microcontroller to an external serial TTL-to-USB converter, and connected the converter to a self-powered USB hub. After taking an SRAM measurement, we switched off power to the microcontroller (i.e., left the power floating) for at least 10 seconds. The goal of this discharging period is to ensure that the microcontroller has discharged completely and that the SRAM will contain fresh data on the next power-up. However, for some devices, this discharging period is insufficient: it is important to connect the power supply lines to ground in order to completely discharge any remaining energy within the microcontroller.

In order to extract start-up patterns more reliably and efficiently, we created a custom measurement board meeting the following requirements:

1. Allow connection of many microcontrollers at once.
2. Be extensible with regard to the number of attached microcontrollers.

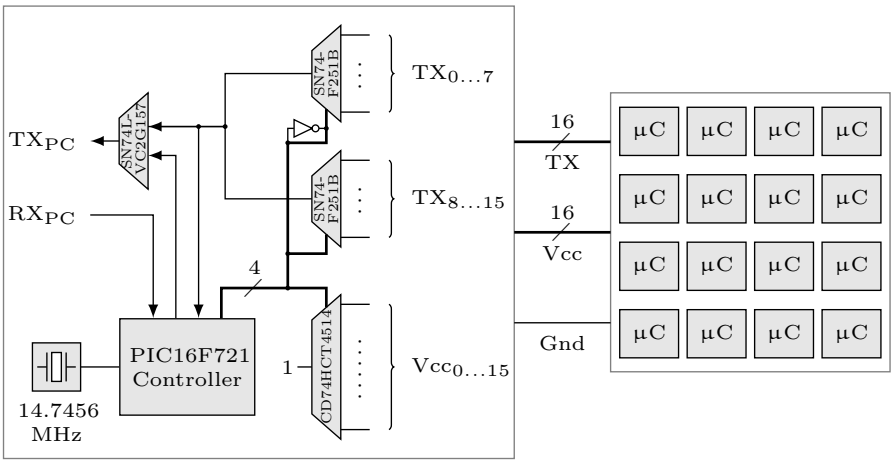

Figure 2.2.1: High-level schematic of the measurement controller board (*left*) with a board of microcontrollers to be measured attached (*right*).

3. Support remote setup.
4. Make automated, unsupervised measurements possible.
5. Support any realistic baud rate.
6. Support an arbitrary SRAM size.
7. Supply upwards-going, fast rising ($\leq$2 ms) Vcc signals.
8. Actively discharge microcontrollers that are not being measured.

Requirements 1 and 2 are satisfied by using (de)multiplexers for the power supply and serial transmission (TX) lines of the attached microcontrollers. The controller board interfaces with a PC, thereby meeting requirements 3 and 4. The controller clock signal is generated with a specialized clock, and the baud rate can also be set though the PC interface, thus fulfilling requirement 5. Requirement 6 is met by detecting when the TX line of the currently powered microcontroller goes idle, at which point the controller board advances to the next connected microcontroller. We used an oscilloscope to verify requirement 7 for our controller board; note that this is important in order to generate realistic start-up patterns. Finally, the demultiplexer on our controller board connects non-active power lines to ground, meeting requirement 8; note that this is important in order to erase the state of the SRAM completely on power-down. A simplified schematic of our design is shown in Fig. 2.2.1.

Central to the board is a PIC16F721 microcontroller which drives a 4-to-16 demultiplexer as well as two 8-to-1 multiplexers. Due to the low current requirements of the devices being measured, the outputs of the demultiplexer can be used as power supply lines. Each of the multiplexer inputs is connected to the serial transmissions port of one of the attached microcontrollers. Furthermore, a 2-to-1 multiplexer is included to allow the controller to switch serial output between either its own serial port or that of the currently powered microcontroller. Since there are some unused pins left on the PIC16F721 this design could

be extended with the help of some logic gates to allow the connection of up to at least 1024 microcontrollers.

As noted earlier, the devices should be completely discharged internally in order to get fresh SRAM power-up values. Otherwise, remnants of previously stored values might linger in memory. Thus, simply disconnecting a microcontroller from its power supply is not sufficient to ensure valid measurements during the next power-up cycle. This makes the selection of the demultiplexer crucial to being able to take valid measurements quickly. We therefore choose to use a CD74HCT4514 demultiplexer, because it connects non-selected outputs to ground, thereby discharging the attached microcontroller.

Keeping in mind future extensibility, it is preferable for the multiplexers to have a tri-state output. This allows wiring together the output of multiple multiplexers, of which only one has its output enabled. Unfortunately, 16-to-1 three-state multiplexers are not produced any more and thus we have chosen to use two 8-to-1 three-state multiplexers, more specifically the SN74F251.

The serial interface speed of the PIC16F721 controller should match that of the devices being measured. Therefore, even though the PIC16F721 has an internal 16 MHz oscillator, it is clocked by an external UART clock (i.e., 14.7456 MHz). This allows the baud rate of the PIC16F721 controller to be adapted on-the-fly to the baud rate of the microcontrollers being measured.

In order to create a flexible measurement platform which can handle any number of microcontrollers with SRAM of any size, the serial output of the current microcontroller being measured is fed into the PIC16F721 measurement controller. After the power supply for a microcontroller has been enabled and that microcontroller has been given some time to power up, the controller starts checking the serial output. If the output remains idle for too long, then either the SRAM measurement is finished or no microcontroller is available at the currently selected position, and the controller advances to the next microcontroller. This system allows for fast, repeated, unattended measurements in which measurement times are automatically adapted to allow full SRAM extraction to take place without requiring any configuration changes to the controller.

For each family of microchips to be measured, a custom PCB was designed containing just the microcontrollers and a minimum of external components (e.g., LEDs to allow debugging feedback and decoupling capacitors), thereby eliminating the change of external components interfering with the microcontroller start-up sequence.

A photograph of the measurement board attached to a prototype PCB for STM32F100R8 readout is shown in Figure 2.2.2.

In order to easily program the surface-mounted microcontrollers we built a device that we call a “programming pen”. This pen is attached to the microcontroller programmer and connects to the target IC using six pogo pins. On the target PCB, a small footprint of 6 vias is required to mate with these pogo pins. Additionally, we added an USB interface that can be used to command a PC to program the microcontroller at the press of a button embedded into the programming pen. An photograph of this device can be seen in Fig. 2.2.3.

2.3 Details on specific microcontrollers

In this section we will outline the details specific to each microcontroller family which had to be taken into account to be able to extract the complete SRAM power-up data.

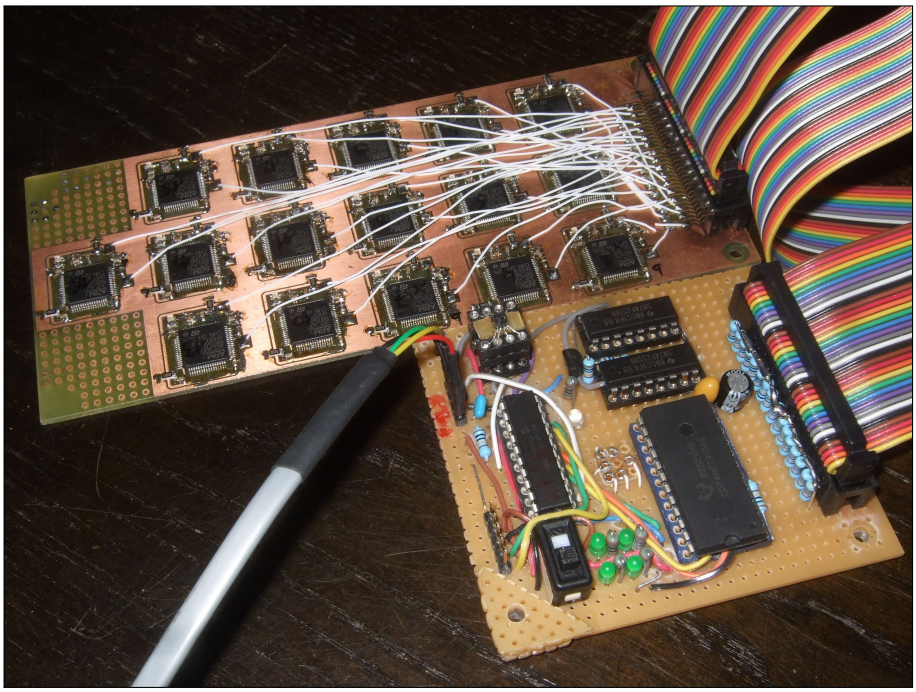

Figure 2.2.2: Measurement controller (*bottom*) connected to prototype STM32F100R8 board (*top*).

2.3.1 Microchip PIC16F1825

The Microchip PIC16 family is peculiar in that it has only a single working register. Futhermore, due to its 8-bit architecture, it requires banking in order to address the full address space. The last 16 elements of the general purpose SRAM are mapped back to bank 0. Due to this banking, the complete general purpose SRAM, which is what we want to extract, has non-sequential addresses. Furtunately, newer PIC16 architectures, such as the PIC16F1825 which we use, have a seperate linear mapping for these SRAM sections. This linear mapping excludes the shared 16 elements, so those have to be handled separately. In our extraction firmware, we first extract the shared 16 elements, and then loop over the linear mapped SRAM.

WP2 reported surprisingly low entropy in our first measurements from the PIC16F1825 chips, so we tried multiple variations of voltage curves on startup. For one such voltage curve, shown in Fig. 2.3.1 on the right, the SRAM contents turned out to have slightly higher entropy. Unfortunately, Microchip does not wish to provide any details on the internal silicon layout of their chips, so it is quite difficult to figure out what causes these effects.

We also noted that all Microchip PIC16F devices we tested kept their SRAM values for over 10 minutes when their power supply line was left floating. This observation was previously reported in dedicated SRAM devices [1], but never observed in COTS microcontrollers. Our custom measurement board eliminated this issue, ensuring proper discharge and a fresh SRAM power-up state.

Figure 2.2.3: Programming pen used to program microcontrollers through a set of pogo pins.

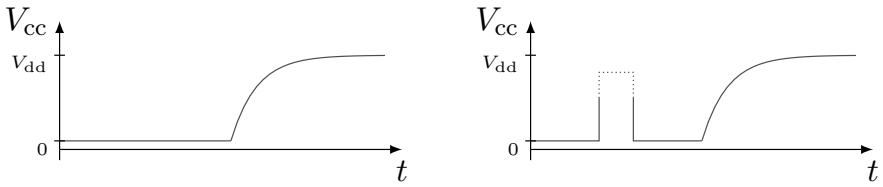

Figure 2.3.1: Powerup voltage supply variation tests on PIC16F1825. Original supply voltage curve (*left*, normal for SRAMs) and altered curve (*right*).

2.3.2 STMicro STM32F100R8

The STM32F100R8 has a 32-bit ARM Cortex-M3 architecture. We extracted SRAM by simply looping over and reading out a linear address range.

2.3.3 Texas Instruments MSP430F5308

The MSP430F5308 has a 16-bit architecture. No banking is required; we extracted SRAM by looping over and reading out a linear address range, as on the STM32F100R8.

2.3.4 Atmel ATMega328p

The ATMega328p is used on the very popular Arduino development boards. It has an 8-bit architecture. As with the two previous chips, we extracted SRAM by looping over a linear address range.

Chapter 3

Smartphone SRAM

Smartphones are much more complex devices than the microcontrollers considered in Chapter 2. WP1 decided to start its smartphone explorations with a reasonably well documented development board, the PandaBoard (ES). This board contains the same TI OMAP4460 system-on-chip used in many smartphones, and multimedia capabilities similar to modern smartphones; it was designed by TI and is sold to the general public with support from a TI subsidy.

The PandaBoard has two ARM Cortex-A9 cores; two Cortex-M3 cores for signal processing; and 2 gigabytes of DDR memory. It also contains the following on-chip memory (OCM) instances:

- 4096 bytes of "Save-and-Restore ROM", presumably not useful.
- 8192 bytes of "Save-and-Restore RAM".
- 57344 bytes of "L3 OCM RAM".

WP1 explored possible accesses to the different memory instances. Analysis showed uninitialized SRAM in the L3 OCM RAM. However, further analysis also showed that the 57344 bytes are not completely usable for fingerprint extraction since a fraction of the memory region is pre-initialized, presumably by the board's ROM code. Reading the first 13312 bytes of the L3 OCM RAM at an early stage of the boot process produced a unique SRAM start-up pattern. To achieve this, WP1 modified the bootloader (u-boot). The modification of the bootloader consisted of finding the appropriate code position for adding the read-out code such that no previous code interacted with the target memory region and thus overwrote the initial SRAM values. The added code loops a pointer through the memory region, displaying each byte on the bootloader's console for retrieval by a controlling PC for further analysis.

Chapter 4

GPU SRAM

Many users of desktop computers, laptop computers, tablets, and smartphones spend the bulk of their processing power on computer graphics, notably as a major component of video games. The only way for a chip manufacturer to provide competitive performance for these applications is to devote large amounts of chip area to the operations used in these applications: for example, heavily vectorized low-precision floating-point multiplications.

Chip manufacturers have, however, been hesitant to include these features in mass-market general-purpose CPUs. Chip area is not free; devoting large amounts of chip area to video games means taking the same area away from features that are critical for many other important CPU applications.

These pressures created a market for add-on "graphics processing units" (GPUs). All users have CPUs; many users add GPUs to provide extra processing power for computer graphics; graphics applications are designed to offload appropriate computations from the CPU to the GPU. In recent years CPU designers have begun to offer integrated chips, with varying numbers of CPU cores and GPU cores on each chip, but the GPU cores are still designed as separate special-purpose cores devoted to graphics applications.

The PUFFIN team already identified GPUs in 2010 as a possible source of uninitialized SRAM visible directly to applications. There are several relevant differences between large CPUs and GPUs:

- Large CPUs evolved various reliability and security features to support multiuser operating systems, often handling critical and sensitive data. GPUs evolved as single-user special-purpose processors, and are generally perceived as handling nothing more than video-game data.

- In particular, large CPUs include "virtual memory" providing a separate address space for each application and "memory protection" separating multiple users of the same computer. Typical GPUs do not (yet) provide either of these features.

- Allowing programs to directly read SRAM after reset could compromise CPU memory protection, so it is unsurprising for a large CPU to disable access to SRAM after reset, taking this concern away from the OS. Typical GPUs have relatively large amounts of SRAM and have no comparable reason to clear the SRAM after reset.

- Large CPUs use SRAM primarily as a cache for DRAM. Typical GPUs expose SRAM and DRAM directly to the programmer, limiting the chip area required for cache logic.

These differences provide reasons to hope that uninitialized SRAM will be more easily visible on GPUs than it is on CPUs on the same computers.

On the other hand, GPU hardware is much more poorly documented than CPU hardware. The GPU SRAM is not directly accessible by the CPU through the PCI bus; the CPU programs the GPU to access data and copy it to the CPU. GPU programing normally uses semi-portable high-level interfaces such as OpenGL, CUDA, and OpenCL; the actual low-level hardware interface is hidden behind compilers and device drivers. NVIDIA's PTX "assembly language" is actually another semi-portable high-level language, hiding most of the hardware details.

The PUFFIN proposal reported that the PUFFIN team had successfully accessed the power-on state of 1.25% of the SRAM from two NVIDIA GPUs, in total 30720 bytes from each GPU. This work took advantage of a new assembly language developed by TUE for NVIDIA Tesla-architecture GPUs, providing much more control than NVIDIA's lowest-level programming language.

After the PUFFIN project started, WP1 successfully accessed a larger fraction of the SRAM from the GPUs, and then developed a new SRAM readout tool using NVIDIA's PTX "assembly language". PTX is really another semi-portable high-level language, hiding most of the hardware details, but provides just barely enough control to access specified locations in SRAM. The resulting main loop is very simple:

```
__global__ void doit(int *results,int words)
{
  for (int i = 0;i < words / THREADS;++i) {
    int pos = threadIdx.x + i * THREADS;
    int data;
    asm("ld.shared.s32 %0, [%1];" : "=r"(data) : "r"(pos << 2));
    results[blockIdx.x * words + pos] = data;
  }
}
```

The power-on SRAM contents appear to contain large amounts of random data. Powering off and on again produces a similar, but not identical, SRAM state. Overwriting the SRAM state and resetting the GPU again produces a similar state, as if the SRAM state had never been overwritten. A different GPU has a different power-on SRAM state. These observations were consistent with what one would expect from uninitialized SRAM.

These explorations encountered a new challenge when we upgraded to the latest versions of the NVIDIA GPU drivers. These drivers appear to clear large amounts of GPU SRAM, presumably in an effort to reduce the amount of undocumented behavior exposed to GPU applications. However, the drivers do not clear SRAM bytes at positions 32 through 63 on each GPU core. The GPUs that we used for experiments each have 30 cores ("multiprocessors"), overall providing 960 bytes (7680 bits) of easily accessible uninitialized SRAM data from each GPU. We measured this data across a series of power-off-pause-power-on cycles, and forwarded the results to WP2.

Bibliography

[1] S. Skorobogatov, "Low Temperature Data Remanence in Static RAM," University of Cambridge, Tech. Rep., Jun. 2002. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.pdf