Universität Bamberg



The Search for Perfection

MOS65xx and C64 Emulation Over Time

Michael Engel (michael.engel@uni-bamberg.de)
Lehrstuhl für Praktische Informatik, insb. Systemnahe Programmierung
https://www.uni-bamberg.de/sysnap

Licensed under CC BY-SA 4.0
unless noted otherwise



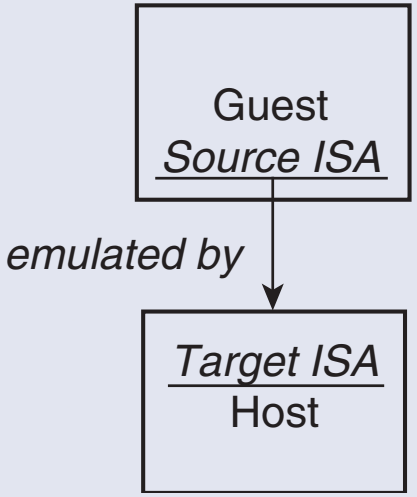
Definitions: Emulation vs. Simulation

One can find the following definitions: [1]

- **Emulation**
  - *The process of implementing the interface and functionality of one system or subsystem on a system or subsystem having a different interface and functionality*
  - It is usually capable of **replacing** the emulated system
  - It omits unimportant details for the sake of performance
- vs. **Simulation**
  - *Precise reproduction of an existing system including its structure and parameters* (e.g. flight simulator)
  - Not necessarily a replacement for the real system – can be significantly slower
  - Allows to analyze and reason about the internals of a system

Processor Emulation

- **CPU emulation is a central part of an emulator**
  - ***Instruction set emulation*** is a key aspect of many virtual machine implementations
  - The virtual machine must support a ***program binary*** compiled for an instruction set that is different from the one implemented by the host processor(s)
  - Emulation allows a machine implementing one instruction set, the target instruction set, ***to reproduce the behavior of software compiled to another instruction set***, the source instruction set

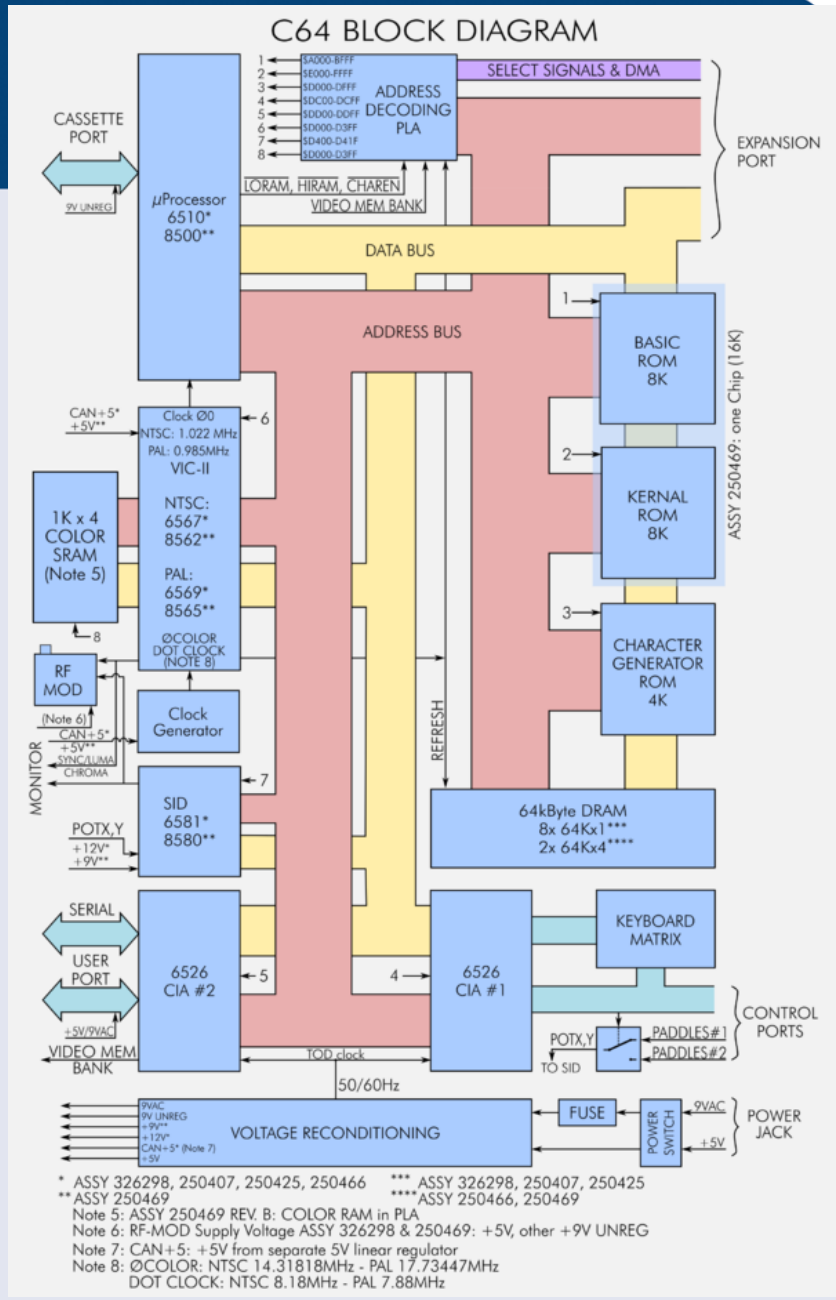


C64 System Emulation

Not only CPU emulation…

- Other mainboard components
  - VIC-II, SID, CIAs, PLA, …
- External devices
  - Keyboard, joysticks, …
  - Especially floppy drives
    - These have their own 6502!





A Very Simple 6502 CPU Emulator

Universität Bamberg

- **Basic structure:** Big while loop + switch statement
  - As simple as it gets
  - More sophisticated approaches described in [1,10]

```
pc = m[0xFFFC]+256*(uint16_t)m[0xFFFD]; // fetch first instruction
     // from address stored at 0xFFFC: 2 bytes, little endian!
r[RA] = 0; r[RY] = 0; … // initialize internal 6502 CPU state

while (1) {
  uint8_t opcode = m[pc];  // fetch instruction opcode from memory
  switch (opcode) {
    case 0xAD: // LDA xxxx (absolute)
         …     // implement semantics here
         break;
    case 0x69: // ADC #xx (immediate)
         …     // implement semantics here
         break;
    …
    default:
         // unknown opcode - print error message
  }
}
```

Emulation Performance

Problem:

This basic emulation is **slow** ……..



```
start:
  CLC
  LDA #0x41
loop:
  STA 0xC000
  ADC #0x01
  CMP #0x5B
  BCC loop
  BRK
```

- Example code: print the letters A-Z
- Execution comprises
  2 + 26 * 4 + 1 = **107** 6502 instructions
  (cycles left as an exercise for the audience! 😀)
- The execution of our emulator used about
  ~5800 instructions on a x86-64 system

…so on average about 50 host instructions are required to execute **one** guest instruction – and this is only the CPU emulation

Emulation Performance



Simple emulation is **slow…**

Why is it so slow?

- Each instruction is read from memory and decoded every time
  - *also in loops – for every iteration!*
- Parameters are separately fetched from memory
- Overhead of while loop and of switch statement
- Overhead of PC updates
- Flag updates even if the updated flags are never evaluated

Can we do better?

- optimizations of the interpreter, static translation (binary translation), dynamic compilation (JIT) [1,10]

⇒ **Tradeoffs** between **emulation speed** and **emulation accuracy!**

Emulation Glitches

- **Problems of inaccurate emulators**
  - Programmers directly accessed the hardware and relied on timing
  - Used side effects, timing loops counting machine instructions, …
- Example: "Air Strike Patrol" for SuperNES: bomber jet shadow helps to aim
  - Generated using mid scanline effects, requires significant overhead

Original





Impresice emulation

6502 System Emulation Challenges

Emulating a seemingly simple CPU can cause subtle problems due to details not described in the official documentation…

- Timing of instructions, buses and peripherals
- CPU Bugs and undocumented features
- "Illegal" Opcodes
- Residual bus data
- Copy protection methods
- Peripheral Emulation (+ 1541…)
- CPU differences: e.g. to the 65C02 and 65816
- …

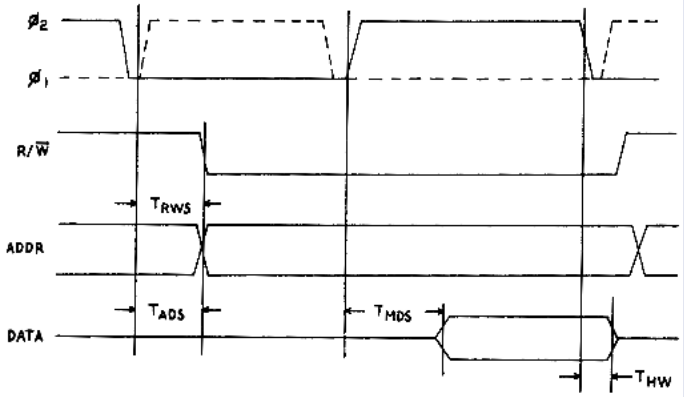
6502 Timing

Universität Bamberg

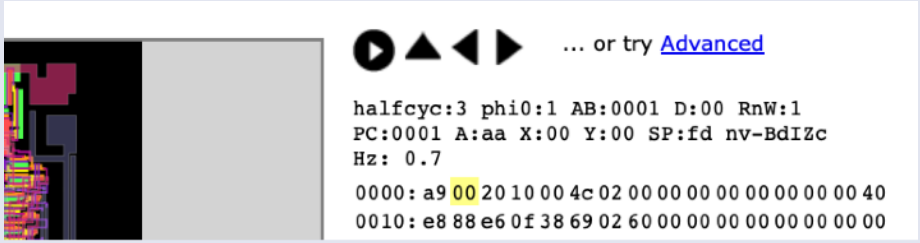
In most systems, memory accesses of the 6502 are controlled by another system component (e.g. the VIC-II in the C64)

- The 6502 (like the Motorola 680x) can share the bus with other devices in an unusual way
  - The CPU only uses the bus when the Φ_2 clock signal is high
  - Other accesses (e.g. video) can occur in the low phase of Φ_2

    → accurate emulation requires interlocking of the "half cycles"



6502 Write Timing



Half cycles in the Visual6502 simulator

6502 Timing Challenges

Apple II – Residual data on the bus [11,12]

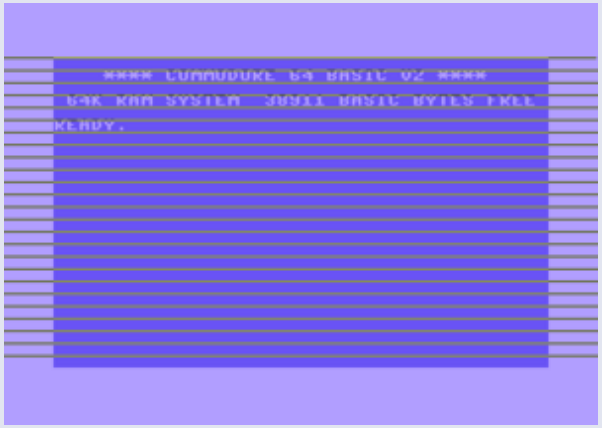
- Detecting vertical blanking on an Apple II, relies on capacitance on the bus and very strict timing
  - On a real Apple II the value read from a non-existent memory address is not always 0, sometimes it's the last value read, which is sometimes the last value read by the video system which is overlapped with the CPU
  - To emulate this right, you need to emulate the whole video system lock-step and cycle accurate with the CPU

Apple II – clock stretching

- The video logic extends every 65th CPU cycle so that the color values generated are in sync with the b/w video signal

C64 – badlines [13]

- During a badline the CPU is frozen for an entire scanline and no memory accesses are performed
- This happens every 8th scanline, but the timing differs between PAL and NTSC versions of the VIC-II (6567/6569)



6502 ~~Bugs~~ Features

Over-fetches

- Indexed addressing across page boundaries does an extra read of an invalid addr.

Read/write sequences

- Read–modify–write instructions perform one read and two write cycles.
- First, the unmodified data that was read is written back, then modified data is written

Addressing mode wrapping/errors

- The memory indirect jump instruction, JMP (<ad>), is partly broken
  - If <ad> = $xxFF the 6502 will not jump to the address stored in xxFF and xxFF+1 as expected, but rather the one defined by xxFF and xx00
  - e.g. JMP ($10FF) would jump to the address stored in 10FF and 1000

BCD (Binary Coded Decimal) addition and subtraction

- The N (negative), V (overflow) and Z (zero) status flags are meaningless when performing arithmetic ops while the CPU is in BCD mode – these flags reflect the binary, not BCD, result
- If the 6502 is in BCD mode when a hardware IRQ occurs, it will not revert to binary

Illegal Instructions

Universität Bamberg

HI	LO-NIBBLE															
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	BRK impl	ORA X,ind				ORA zpg	ASL zpg		PHP impl	ORA #	ASL A			ORA abs	ASL abs	
1-	BPL rel	ORA ind,Y				ORA zpg,X	ASL zpg,X		CLC impl	ORA abs,Y				ORA abs,X	ASL abs,X	
2-	JSR abs	AND X,ind			BIT zpg	AND zpg	ROL zpg		PLP impl	AND #	ROL A		BIT abs	AND abs	ROL abs	
3-	BMI rel	AND ind,Y				AND zpg,X	ROL zpg,X		SEC impl	AND abs,Y				AND abs,X	ROL abs,X	
4-	RTI impl	EOR X,ind				EOR zpg	LSR zpg		PHA impl	EOR #	LSR A		JMP abs	EOR abs	LSR abs	
5-	BVC rel	EOR ind,Y				EOR zpg,X	LSR zpg,X		CLI impl	EOR abs,Y				EOR abs,X	LSR abs,X	
6-	RTS impl	ADC X,ind				ADC zpg	ROR zpg		PLA impl	ADC #	ROR A		JMP ind	ADC abs	ROR abs	
7-	BVS rel	ADC ind,Y				ADC zpg,X	ROR zpg,X		SEI impl	ADC abs,Y				ADC abs,X	ROR abs,X	
8-		STA X,ind			STY zpg	STA zpg	STX zpg		DEY impl		TXA impl		STY abs	STA abs	STX abs	
9-	BCC rel	STA ind,Y			STY zpg,X	STA zpg,X	STX zpg,Y		TYA impl	STA abs,Y	TXS impl			STA abs,X		
A-	LDY #	LDA X,ind	LDX #		LDY zpg	LDA zpg	LDX zpg		TAY impl	LDA #	TAX impl		LDY abs	LDA abs	LDX abs	
B-	BCS rel	LDA ind,Y			LDY zpg,X	LDA zpg,X	LDX zpg,Y		CLV impl	LDA abs,Y	TSX impl		LDY abs,X	LDA abs,X	LDX abs,Y	
C-	CPY #	CMP X,ind			CPY zpg	CMP zpg	DEC zpg		INY impl	CMP #	DEX impl		CPY abs	CMP abs	DEC abs	
D-	BNE rel	CMP ind,Y				CMP zpg,X	DEC zpg,X		CLD impl	CMP abs,Y				CMP abs,X	DEC abs,X	
E-	CPX #	SBC X,ind			CPX zpg	SBC zpg	INC zpg		INX impl	SBC #	NOP impl		CPX abs	SBC abs	INC abs	
F-	BEQ rel	SBC ind,Y				SBC zpg,X	INC zpg,X		SED impl	SBC abs,Y				SBC abs,X	INC abs,X	

What happens if we try to execute an opcode for which there is a gap in the instruction set table?

The 6502 has 105 undocumented opcode slots called "illegal/unofficial/undocumented opcode/instruction"!

Illegal Instructions

Universität Bamberg

HI	LO-NIBBLE															
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	BRK impl	ORA X,ind	JAM	SLO X,ind	NOP zpg	ORA zpg	ASL zpg	SLO zpg	PHP impl	ORA #	ASL A	ANC #	NOP abs	ORA abs	ASL abs	SLO abs
1-	BPL rel	ORA ind,Y	JAM	SLO ind,Y	NOP zpg,X	ORA zpg,X	ASL zpg,X	SLO zpg,X	CLC impl	ORA abs,Y	NOP impl	SLO abs,Y	NOP abs,X	ORA abs,X	ASL abs,X	SLO abs,X
2-	JSR abs	AND X,ind	JAM	RLA X,ind	BIT zpg	AND zpg	ROL zpg	RLA zpg	PLP impl	AND #	ROL A	ANC #	BIT abs	AND abs	ROL abs	RLA abs
3-	BMI rel	AND ind,Y	JAM	RLA ind,Y	NOP zpg,X	AND zpg,X	ROL zpg,X	RLA zpg,X	SEC impl	AND abs,Y	NOP impl	RLA abs,Y	NOP abs,X	AND abs,X	ROL abs,X	RLA abs,X
4-	RTI impl	EOR X,ind	JAM	SRE X,ind	NOP zpg	EOR zpg	LSR zpg	SRE zpg	PHA impl	EOR #	LSR A	ALR #	JMP abs	EOR abs	LSR abs	SRE abs
5-	BVC rel	EOR ind,Y	JAM	SRE ind,Y	NOP zpg,X	EOR zpg,X	LSR zpg,X	SRE zpg,X	CLI impl	EOR abs,Y	NOP impl	SRE abs,Y	NOP abs,X	EOR abs,X	LSR abs,X	SRE abs,X
6-	RTS impl	ADC X,ind	JAM	RRA X,ind	NOP zpg	ADC zpg	ROR zpg	RRA zpg	PLA impl	ADC #	ROR A	ARR #	JMP ind	ADC abs	ROR abs	RRA abs
7-	BVS rel	ADC ind,Y	JAM	RRA ind,Y	NOP zpg,X	ADC zpg,X	ROR zpg,X	RRA zpg,X	SEI impl	ADC abs,Y	NOP impl	RRA abs,Y	NOP abs,X	ADC abs,X	ROR abs,X	RRA abs,X
8-	NOP #	STA X,ind	NOP #	SAX X,ind	STY zpg	STA zpg	STX zpg	SAX zpg	DEY impl	NOP #	TXA impl	ANE #	STY abs	STA abs	STX abs	SAX abs
9-	BCC rel	STA ind,Y	JAM	SHA ind,Y	STY zpg,X	STA zpg,X	STX zpg,Y	SAX zpg,Y	TYA impl	STA abs,Y	TXS impl	TAS abs,Y	SHY abs,X	STA abs,X	SHX abs,Y	SHA abs,Y
A-	LDY #	LDA X,ind	LDX #	LAX X,ind	LDY zpg	LDA zpg	LDX zpg	LAX zpg	TAY impl	LDA #	TAX impl	LXA #	LDY abs	LDA abs	LDX abs	LAX abs
B-	BCS rel	LDA ind,Y	JAM	LAX ind,Y	LDY zpg,X	LDA zpg,X	LDX zpg,Y	LAX zpg,Y	CLV impl	LDA abs,Y	TSX impl	LAS abs,Y	LDY abs,X	LDA abs,X	LDX abs,Y	LAX abs,Y
C-	CPY #	CMP X,ind	NOP #	DCP X,ind	CPY zpg	CMP zpg	DEC zpg	DCP zpg	INY impl	CMP #	DEX impl	SBX #	CPY abs	CMP abs	DEC abs	DCP abs
D-	BNE rel	CMP ind,Y	JAM	DCP ind,Y	NOP zpg,X	CMP zpg,X	DEC zpg,X	DCP zpg,X	CLD impl	CMP abs,Y	NOP impl	DCP abs,Y	NOP abs,X	CMP abs,X	DEC abs,X	DCP abs,X
E-	CPX #	SBC X,ind	NOP #	ISC X,ind	CPX zpg	SBC zpg	INC zpg	ISC zpg	INX impl	SBC #	NOP impl	USBC #	CPX abs	SBC abs	INC abs	ISC abs
F-	BEQ rel	SBC ind,Y	JAM	ISC ind,Y	NOP zpg,X	SBC zpg,X	INC zpg,X	ISC zpg,X	SED impl	SBC abs,Y	NOP impl	ISC abs,Y	NOP abs,X	SBC abs,X	INC abs,X	ISC abs,X

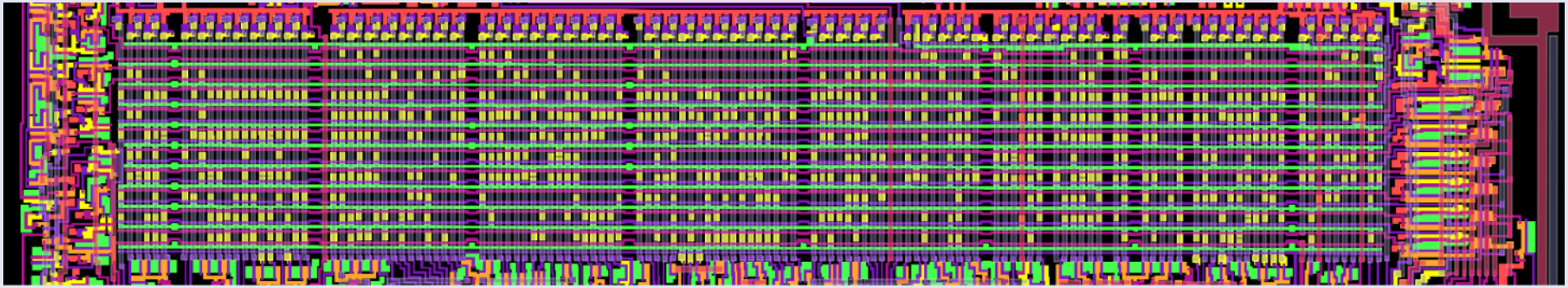
Examples:

- LAX – load operand into A **and** X reg's (A7,B7,AF,BF,A3,B3)
- ARR – AND operand + ROR (B6)
- JAM (also KIL) – hangs the CPU when executed

Illegal Instructions

- 6502 uses a **PLA** to decode opcodes (programmable logic array)
  - a compact and regular way to construct ~130 NOR gates whose inputs come from the Instruction Register and the Timing Generator
  - PLA outputs are only defined for 151 of the 256 available opcodes

- **Saving transistors with lazy decoding**
  - Control logic for 6502 was designed to contain the minimal number of states to try to simplify the actual gate logic once it is compiled



An Early 6502 Emulator

Peter Koch's Apple II emulator (University of Kaiserslautern 1993)

- Emulates Apple][,][+
- NMOS 6502 emulation with/without illegal instructions + 65C02
- VSYNC emulation (every 17000 clock ticks)
- No clock stretching
- No addr. mode wrapping for JMP (xxFF)
- No over-fetches

```
opcode=get6502memory(PC++);

DISASM

switch (opcode)
{
    /* opcodes not matching any adrmode */
    case 0x0a: /* ASL A */
        if (A & 0x80) set6502c; else clr6502c;
        A = A << 1;
        set6502nz(A);
        DISASM
        break;
    case 0x00: /* BRK */
}
```

Performance:
on Sun ELC (SPARC, ~12 MIPS) – approx. 80%
on Sequent S27 (80386, ~5 MIPS) – approx. 30%
on Sun 3/260 (68020, ~4 MIPS) – approx. 20%.

https://mirrors.apple2.org.za/ground.icaen.uiowa.edu/Mirrors/uni-kl/emulators/apple2.tar.Z

First C64 Emulation Projects

Universität Bamberg

- **The 64 Emulator** (ReadySoft)
- **GO-64!** (Software Insights Systems, no SID)
  - ~1986 on Amiga, 2-5 x slower than C64
- **VICE**
  - Started 1993 by Jarkko Sonninen
  - Today also emulates C64-DTV, C128, VIC20, ~all PET models, Plus4, CBM-II
  - Available on a large number of platforms
- **CCS64**
  - 1995 by Per Håkan Sundell
  - C64 emulator for PC – shareware
  - Master thesis at Chalmers University [14]



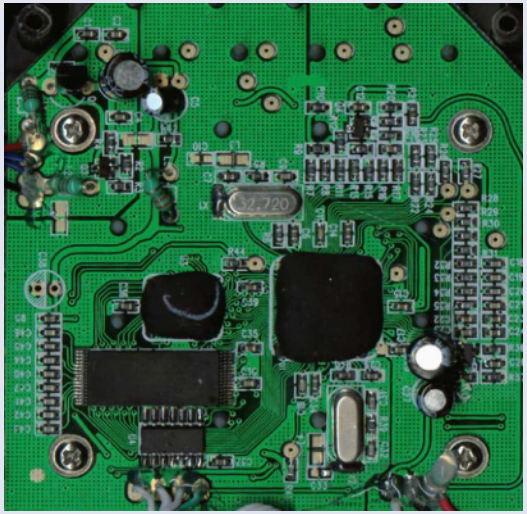
Denise		2.3
VICE (x64sc)		3.8
Hoxs64		1.1.2.6
Z64K		2.0.0
VirtualC64		5.0b7
VICE (x64)		3.8[N 1]
Emu64		git
Kernal64		1.7.3b16
Micro64		1.00.2018.03.16 Build 719
BizHawk		2.9.1
CCS64		V3.9.2
Hyper64		1.00.2008.08.08 Build 20
MAME		0.267
Power64		4.9.5
Frodo		4.1b
Breadbox (VICE based)		1.0
CSFEC		20240422

ASIC Implementation

C64DTV (2004)

- Single-chip ASIC implementation of the C64 implementing 6510, VIC-II, SID, CIA, and PLA
- Contained in a joystick case
- Emulation issues: Many games (up to 30%) do not work properly, no SID filters emulated [5]







ASIC designed by Jeri Ellsworth

FPGA-based C64 Implementations

Universität Bamberg

- **C-One** (2002) – chipset only, uses real 6502
  by Jeri Ellsworth and Jens Schönfeld
- **Syntiac FPGA-64** (2006)
  by Peter Wendrich
- **Turbo Chameleon** (2013)
  by Jens Schönfeld

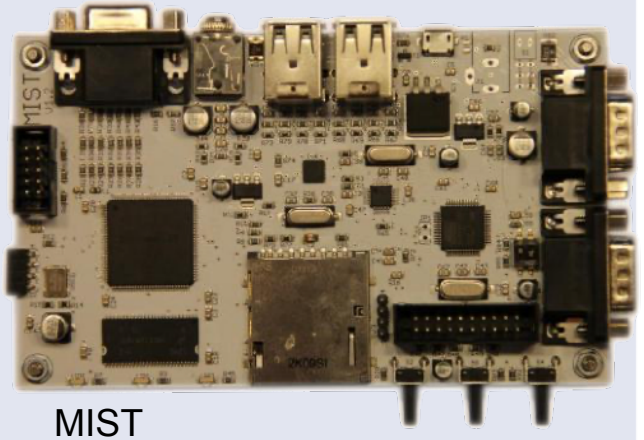


Turbo Chameleon

Ultimate 64



- **Ultimate 64** (2018)
  by Gideon Zweijtzer
- **MIST** (2014) [6] based on FPGA-64 core
  Versatile FPGA platform by Lotharek
- **MISTer** (2017) – successor + more features

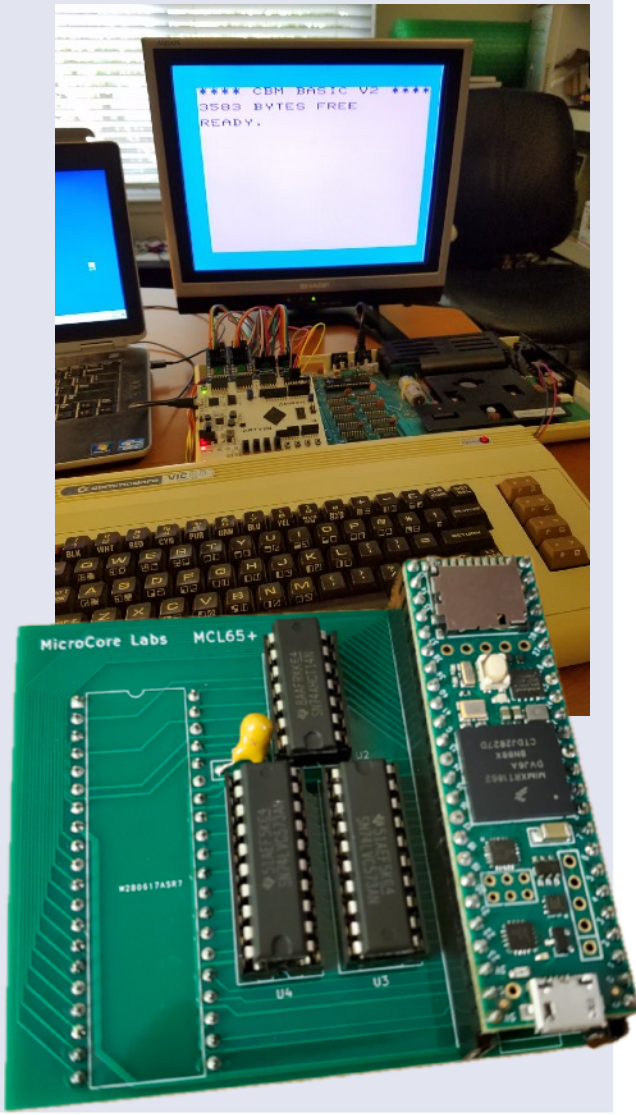


MIST

MCL65

Microcoded versions of the 6502 (and others)

- Created by Ted Fried (MicroCore Labs)
- **FPGA version** on Xilinx Spartan-7 FPGA
  - Uses only 252 LUTs (0.77%) of the FPGA!
  - Plugs into 6502 socket using adapter board
- **Software version** running on Teensy 4.1 board (+ adapter PCB to shift voltages)
  - ARM Cortex-M7 at 600 MHz (RT1062), 8 MB flash and 1 MB RAM
  - Fast enough to emulate the CPU and the 6502 bus IO in software
    https://microcorelabs.wordpress.com
    https://github.com/MicroCoreLabs/
    https://www.pjrc.com/store/teensy41.html
- Similar setups exist for AVR, STM32 [14,15]



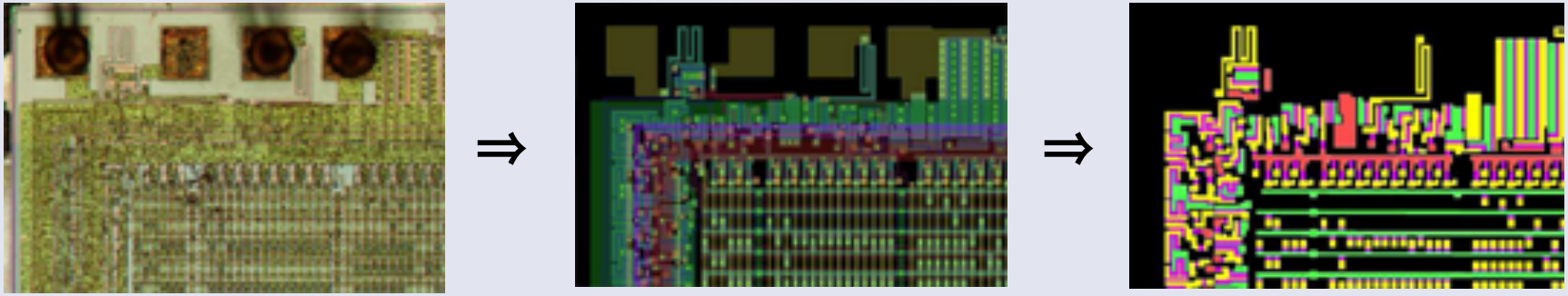
Digging Deeper…

All FPGA-based emulations have compatibility problems

- Certain demos and games have glitches or crash
- Recreation of the FPGA and ASIC cores using **behavioral models**
  - "Black box" reverse engineering from observations (+docs)

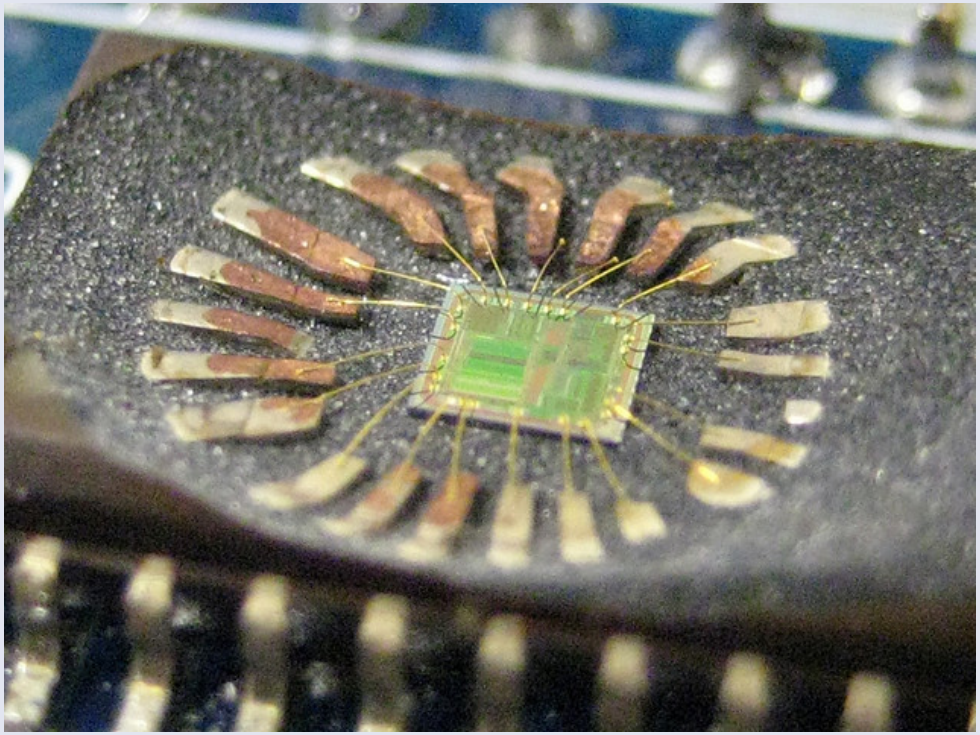
How can we achieve improved emulation accuracy?

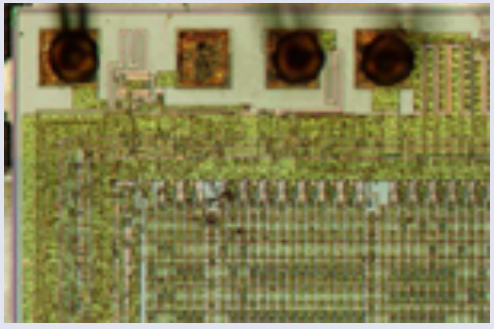
- "White box" reverse engineering:
  - "Decap" a chip using strong acids
  - Take microscope pictures and reconstruct the transistor model
  - Difficult for recent ICs, feasible for the 3µm 6502 feature width



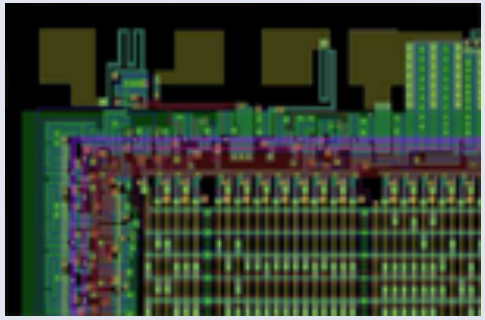
Taking a Closer Look

Universität Bamberg

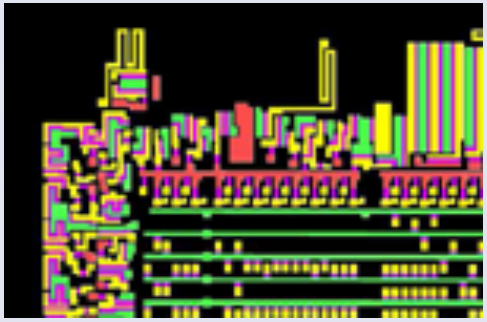




⇒



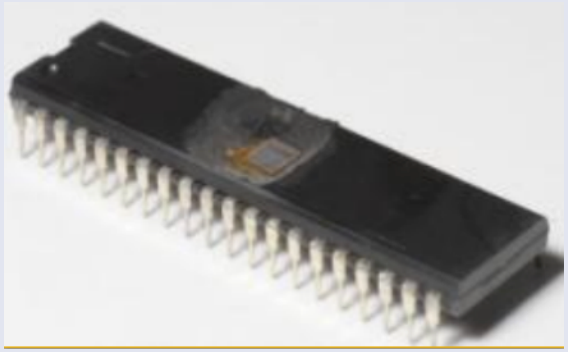
⇒

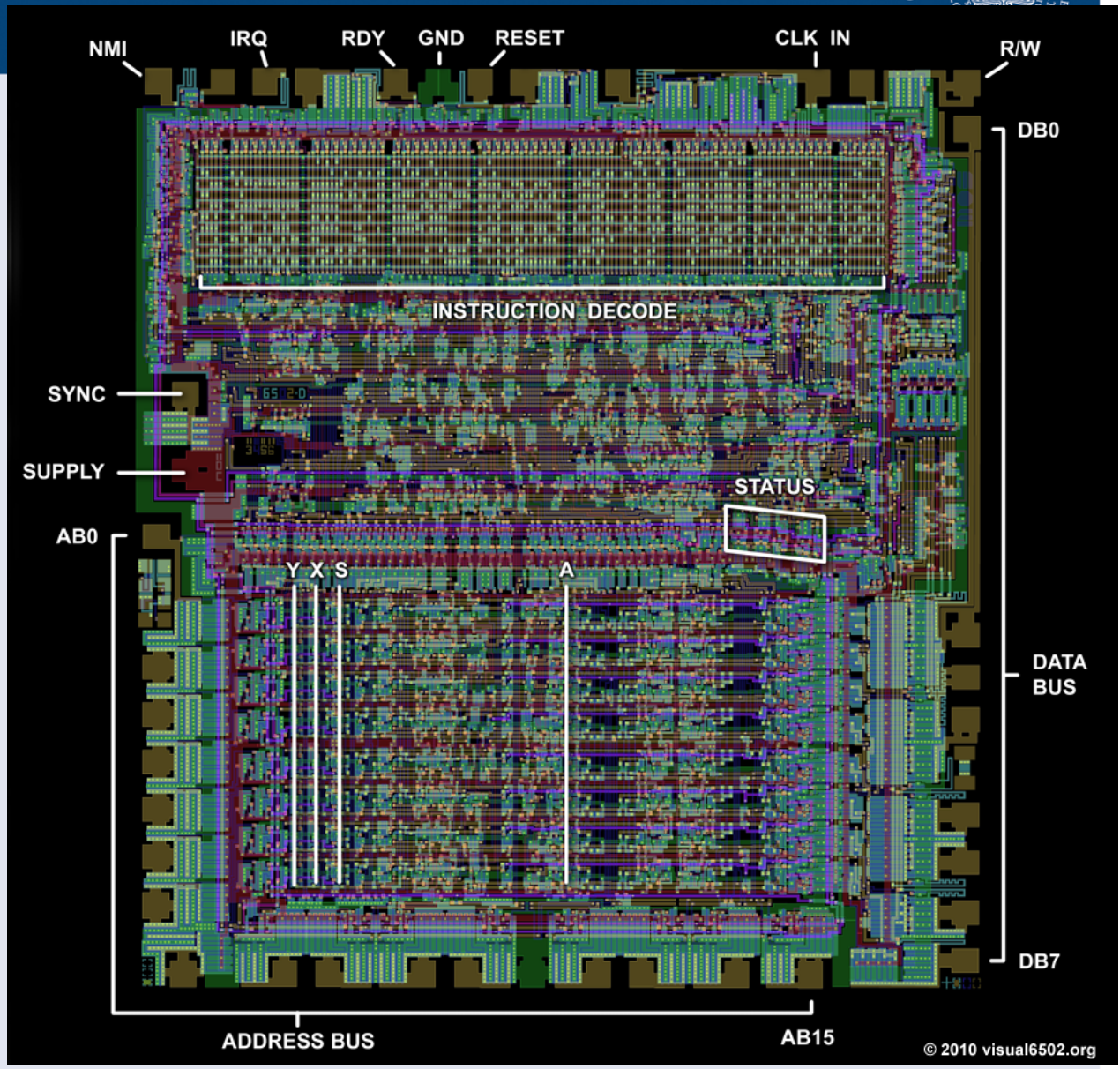


Visual 6502

- Transistor-level emulation (Javascript)
- Models the NMOS 6502 [7]
- Netlist created from die photos
- Semantic annotations of transistor functions

http://visual6502.org





So… Build Transistor-Level Hardware?

…use real transistors: the MOnSter 6502! (2015)

- Created by Eric Schlaepfer
- "Dis-integrated" circuit project
- Complete, working transistor-scale replica of the NMOS 6502
- Four layer PCB, 30 × 38 cm² – 7000 times the actual size
- 3218 transistors, 1019 resistors
- Maximum reliable clock ~ 50 kHz



https://monster6502.com/

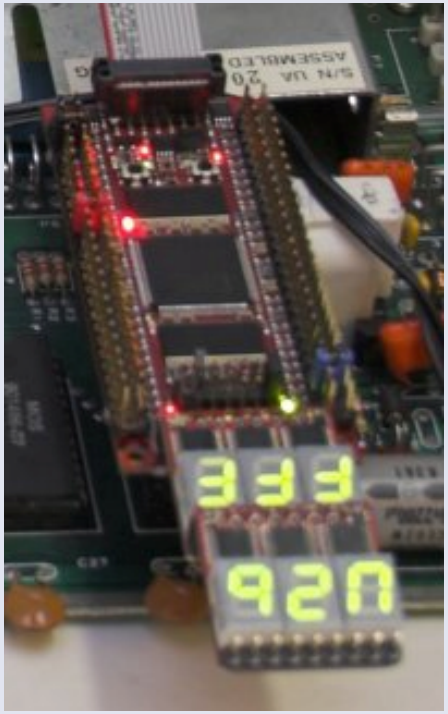
C simulation of the original NMOS 6502 netlist (2014)

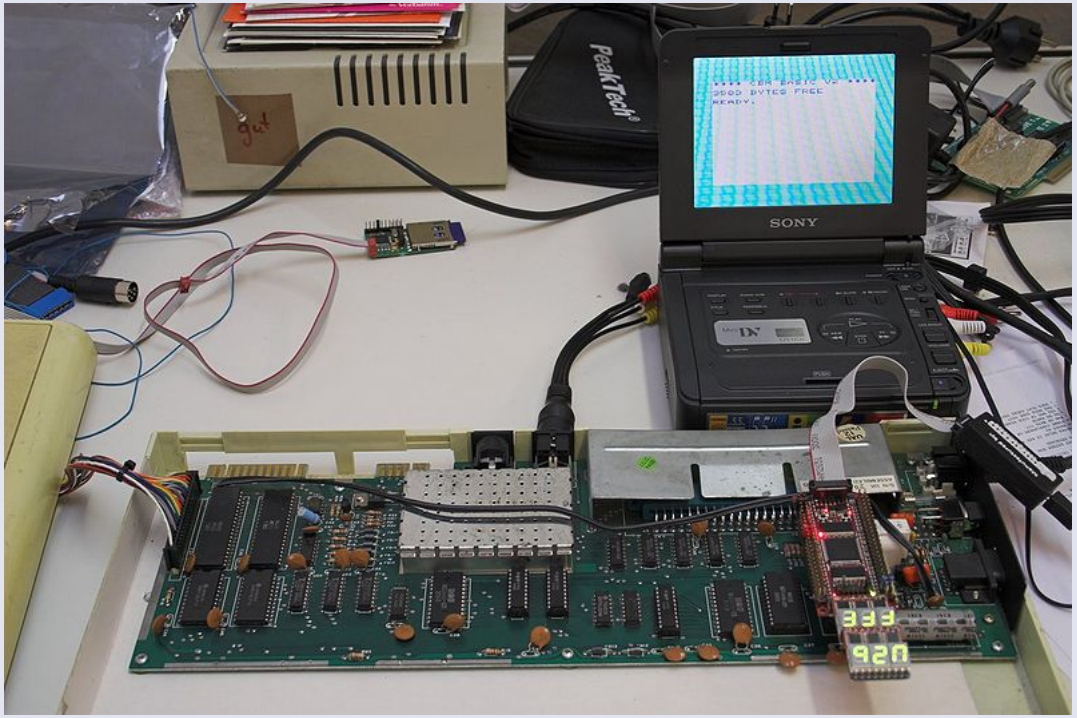
- Created by Michael Steil (https://github.com/mist64/perfect6502)
- Complete behavior, internal state and outputs of the emulated 6502 are half-cycle exact
- …But it's **slow**:
  - perfect6502 uses highly optimized C code
  - Achieves 1/30 of a 1 MHz 6502's speed on a high-end CPU of 2020
- perfect6502 can be used to run Commodore and Apple I BASIC
  - Run BASIC, e.g. in a Unix shell



FEHLER Project (TU Dortmund)

- Idea: translate Javascript-based netlist back to real hardware [8,9]
  - Verilog version of transistor netlist → FPGA (Xilinx 3S500E)
  - Replace 6502/6510 in real systems: C64, VC20, Apple II
  - Based on the GODIL FPGA+level shifter module





Other C64 Custom Chips?

- The 6502/6510 is only one of the many custom chips in a C64
  - …and you can still buy new 65C02 CPUs (+adapters for 6510 [10])
- **What about the other custom Commodore chips?**
  Implementation examples (others exist for some of the chips):
  - VIC-II (6567/6569) Video Interface Chip
    https://github.com/randyrossi/vicii-kawari
  - SID (6581/8580) Sound Interface Device
    https://www.fpgasid.de/
  - CIA (6526/8250/8251) Complex Interface Adapter
    https://github.com/daglem/reDIP-CIA
  - PLA – Programmable Logic Array
    https://ezcontents.org/recreate-c64-pla-chip-vhdl
    (82S100 marked 906114 28-pin device)
    https://frank-buss.de/c64/pla/
    (64-pin ASIC 251715/252535 in C64C)

Testing

Getting all details right is a painstaking process…

Test suites for hardware implementations and emulators exist:

- Klaus Dormann's test suite – portable, includes decimal mode, is standalone and can be assembled to a single image around 16k [T1]
- Wolfgang Lorenz' C64 suite – exhaustive, excluding decimal mode, uses C64 facilities to chain each program. 11kByte total [T2]
- Ruud Baltissen's 8k test ROM – from his VHDL 6502 core [T3]
- NES test rom – by Kevin Horton (24kbyte) [T4]
- AllSuiteA.asm – from the hcm-6502 Verilog project [T5]
- Decimal mode tests – by Bruce Clark ADC/SBC (exhaustive, tests all four affected flags and several CPU versions) [T6]
- Rob Finch's tests – from his 6502 core [T7]
- Acid800 – by Avery Lee for 8-bit Atari emulators includes 6502 tests [T8]
- ASAP tests – by Piotr Fusik include exhaustive tests for ADC, SBC and 0x6B as well as a few tests for other undocumented opcodes [T9]

Turtles All the Way Down…

Universität Bamberg

[T]he scientist, after giving a lecture on astronomy, is approached by a little old lady who says that he's got it all wrong and that the world in fact rests on the back of a giant turtle. The scientist then asks the lady what the turtle is standing on, and she answers: on the back of a second, even larger turtle. But, asks the scientist, what does *that* turtle stand on? To which the lady triumphantly answers: "You're very clever, young man, but it's no use – it's **turtles all the way down**!"

Terry Pratchett, "Little Gods"



[Public Domain, wikipedia]

Many more details (in German):
My talk "**Vom Silizium zur Simulation und zurück**" @ HASI Siegen 2017
https://www.youtube.com/watch?v=5cfvL3cMnz4

References

[1] J. Smith, R. Nair: Virtual Machines: Versatile Platforms for Systems and Processes, MKP 2005

[2] M. Steil: How MOS 6502 Illegal Opcodes really work – https://www.pagetable.com/?p=39

[3] groepaz: NMOS 6510 Unintended Opcodes – no more secrets – https://csdb.dk/release/?id=173068

[4] D. Thom, V. Numbers: The Software Protection Handbook for the C-64, PSIDAC 1984

[5] C64 emulation issues
https://web.archive.org/web/20121103095402/http://picobay.com/dtv_wiki/index.php?title=DTV_Programming#C64_emulation_issues_.28for_the_DTV2.2F3.29

[6] MIST C64 core – https://github.com/mist-devel/c64

[7] G. James, B.Silverman: Visualizing a Classic CPU in Action: The 6502, Proc. of SIGGRAPH 2010

[8] Ingo Korb, A Cross-Layer Analysis of Error Impact on the Instruction Execution in an 8-bit Microprocessor
Diploma Thesis, TU Dortmund, 2012

[9] Ingo Korb's version of the FPGA netlist tools – https://github.com/ikorb/FPGA-netlist-tools

[10] K. Hazelwood: Dynamic Binary Modification: Tools, Techniques, and Applications,
Synthesis Lectures on Computer Architecture, Morgan-Claypool/Springer 2011

[11] Apple II Vapor Lock and the Floating Bus
http://deater.net/weave/vmwprod/megademo/vapor_lock.html

[12] Ryan Carsten Schmidt: Apple II floating bus does not switch modes at the right time
https://github.com/TomHarte/CLK/issues/1204

[13] C64 Wiki - Badline – https://www.c64-wiki.de/wiki/Badline

[14] Klaus Dormann's ATmega16 6502 emulator – http://2m5.de/6502_Emu/index.htm

[15] Chris Baird's STM32 6502 emulator – https://github.com/BigEd/stm6502

[16] Håkan Sundell: Correct Emulation of Micro-Computer C64 in Software independent of Computer System
http://www.ccs64.com/Master_Thesis__Correct_Emulation_of_Micro_computer_C64_in_Software_Independent_of_Computer_System.pdf

Test Suites

[T1] Klaus Dormann – https://github.com/Klaus2m5/6502_65C02_functional_tests/tree/master

[T2] Wolfgang Lorenz – http://www.modelb.bbcmicro.com/testsuite-2.15.tar.gz

[T3] Ruud Baltissen – http://www.baltissen.org/zip/rb65-11.zip

[T4] NES Test ROM – http://nickmass.com/images/nestest.nes

[T5] AllSuiteA – http://code.google.com/p/hmc-6502/source/browse/trunk/emu/testvectors/AllSuiteA.asm

[T6] Bruce Clark's Decimal mode tests – http://www.6502.org/tutorials/decimal_mode.html

[T7] Rob Finch – http://web.archive.org/web/20070707064155/http://www.birdcomputer.ca/Projects/Prj6502/bc6502_page.html

[T8] Avery Lee's Acid800 – http://www.virtualdub.org/downloads/Acid800-1.0.7z

[T9] Piotr Fusik's ASAP tests – http://asap.git.sourceforge.net/git/gitweb.cgi?p=asap/asap;a=tree;f=test