When hardware must “just work”

David Kaplan
Farhan Rahman

Introduction

- Hardware development is different than software
  - Long development timeline
  - $$$
  - Difficult to test everything before fabrication

- X86 cores are especially challenging
  - Highly complex (~60M NAND equivalent gates, ~1M lines of code)
  - Super fast clock rate
  - Must work ~100% of the time

Are CPUs perfect? LOL

- High profile CPU bugs
  - Intel Pentium divide bug (1994)
  - AMD Phenom TLB bug (2007)

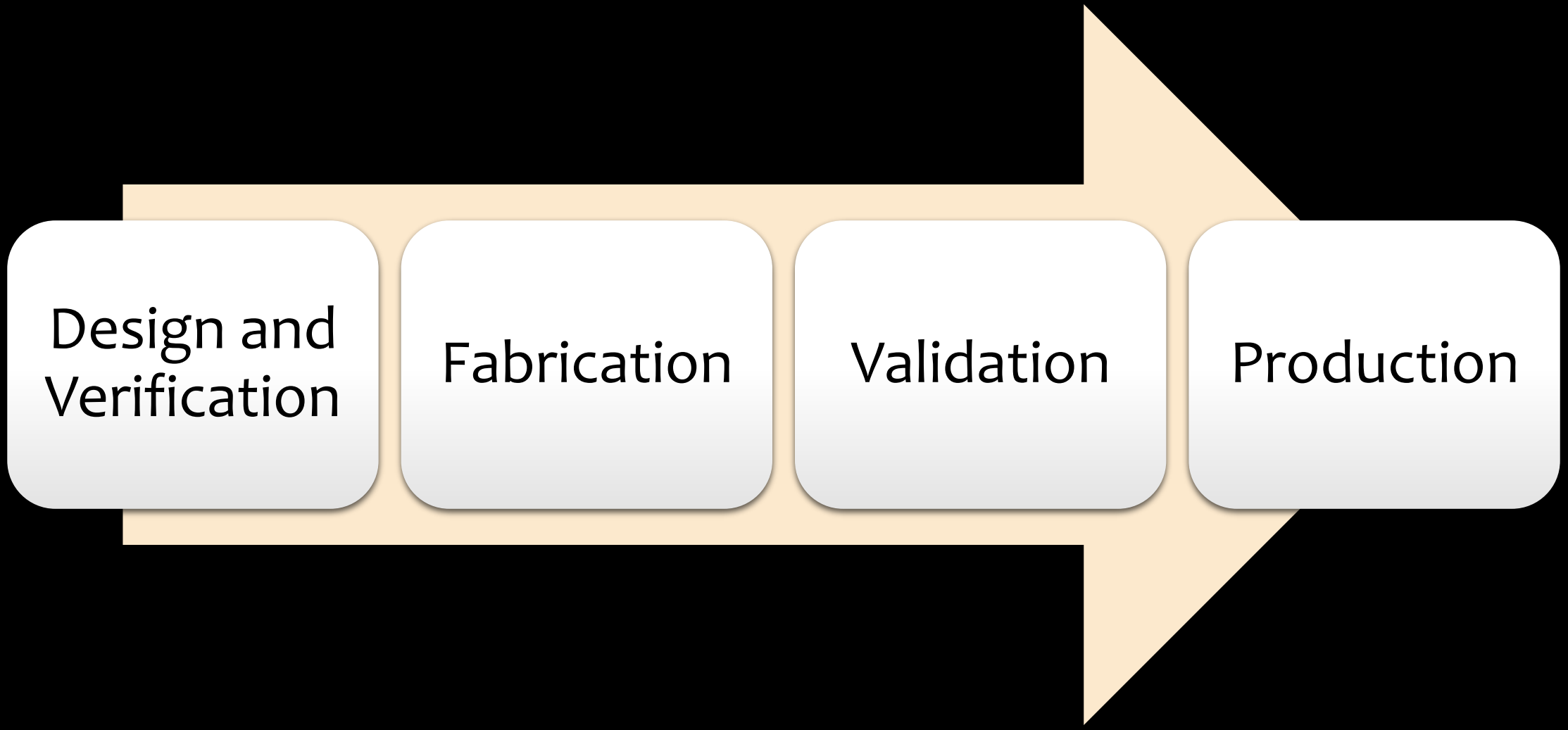
- Many other minor bugs reported in errata guides
  - They're minor though…

http://support.**amd**.com/TechDocs/51810_16h_00h-0Fh_Rev_Guide.pdf

Table 5. Cross-Reference of Processor Revision to Errata

No.	Errata Description	CPUID Fn0000_0001_EAX
		00700F01h (KB-A1)
77	Long Mode CALLF or JMPF May Fail To Signal GP When Callgate Descriptor is Beyond GDT/LDT Limit	No fix planned
361	Breakpoint Due to an Instruction That Has an Interrupt Shadow May Be Lost	No fix planned
541	IBS Registers May be Unpredictable After CC6 State	No fix planned
638	Processor May Violate Trp During Dynamic Mode Switch	No fix planned
737	Processor Does Not Check 128-bit Canonical Address Boundary Case on Logical Address	No fix planned
756	Machine Check Information May Show Inconsistent Signature from an Older Corrected Error	No fix planned
757	L2 Tag Error Machine Check Status May Be Incorrect	No fix planned
767	Processor APM Behavior May Be Incorrect After CC6	No fix planned
776	Incorrect Processor Branch Prediction for Two Consecutive Linear Pages	No fix planned
778	Processor Core Time Stamp Counters May Experience Drift	No fix planned
779	Initial Time Stamp Counter Frequency May Be Incorrect	No fix planned
780	Processor May Cache Guest Write Combining Memory Type	No fix planned
781	Improper Handling of ECX Value During RDPMC Instruction	No fix planned
785	USB Interrupt Status May Not Be Set After a Short Packet	No fix planned
786	APIC Timer Periodic Mode is Imprecise	No fix planned
789	G3 to S5 Power State Transition May Observe Real Time Clock Errors and Other Unpredictable System Behavior	No fix planned
791	RDTSCP Instruction Does Not Serialize	No fix planned
792	DRAM Scrubbing May Overwrite CC6 Core Save State Data Resulting in Unpredictable System Behavior	No fix planned
793	Specific Combination of Writes to Write Combined Memory Types and Locked Instructions May Cause Core Hang	No fix planned
794	Performance Monitor PMCx076 May Be Inaccurate and Lose Overflow Interrupts When Halted	No fix planned
795	Core Performance Monitor Counters May Appear to be Cleared to Zero in the Least Significant 32 Bits	No fix planned

Hardware design process

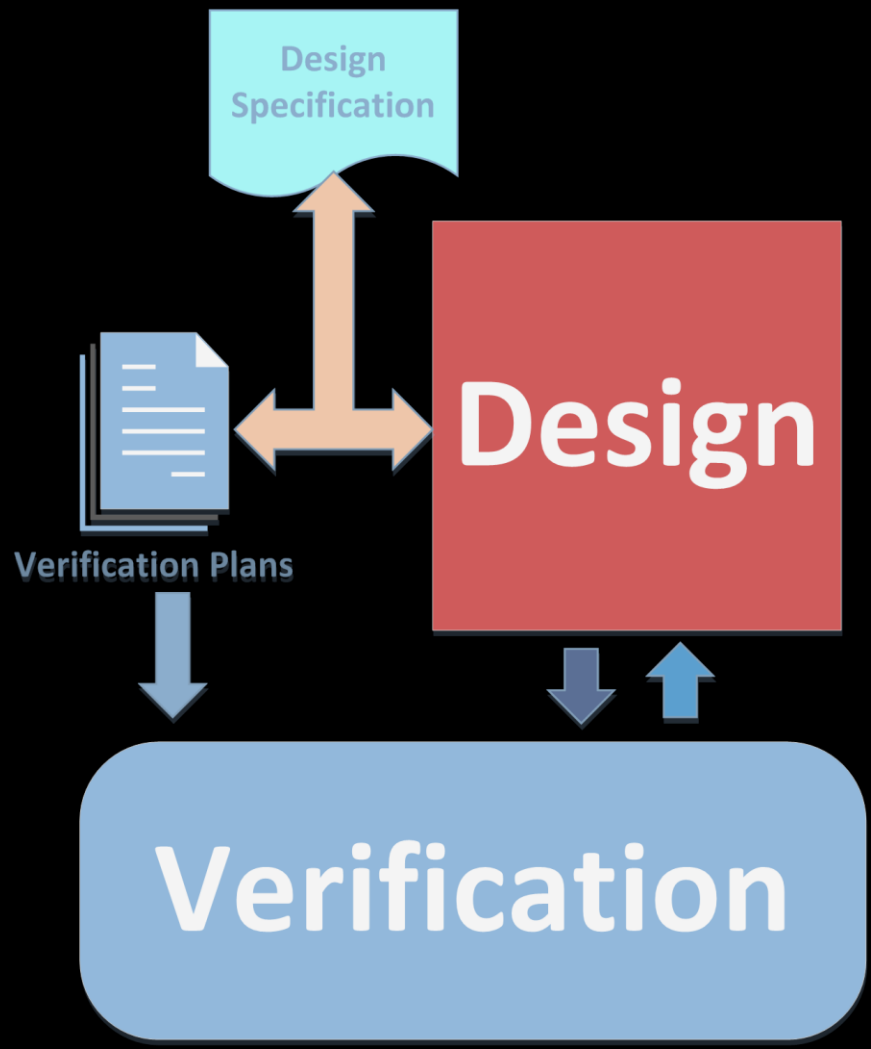


Verification (Pre-silicon)

What is verification?

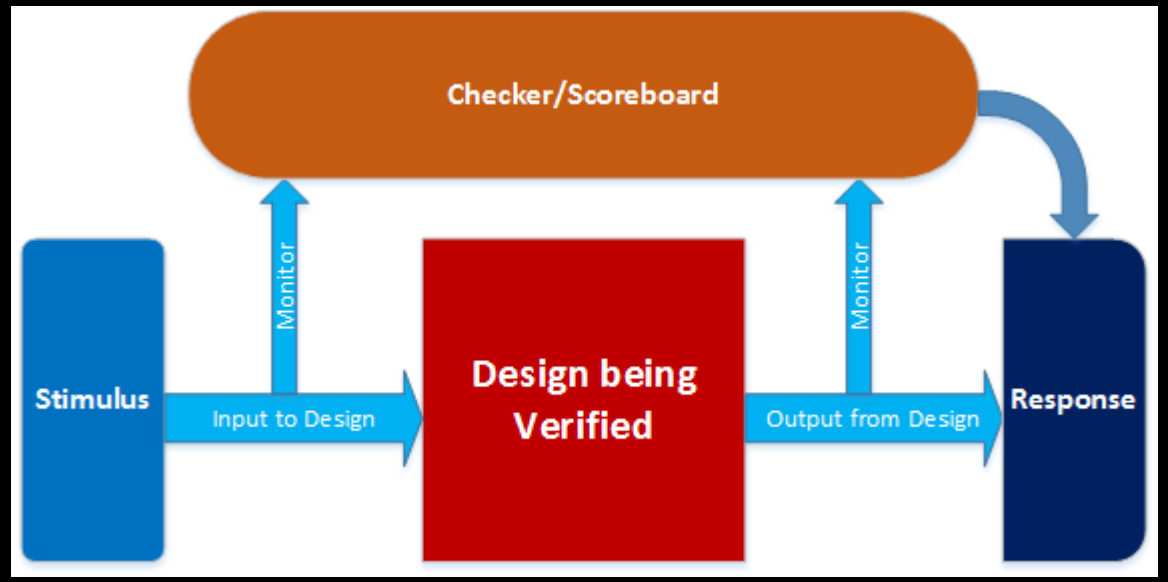
- Verification is a specific discipline in silicon design that ensures a design conforms with the **design specification** (contains functional, power, and performance details)

- Goal of verification is to find design defects (aka, Bugs!)



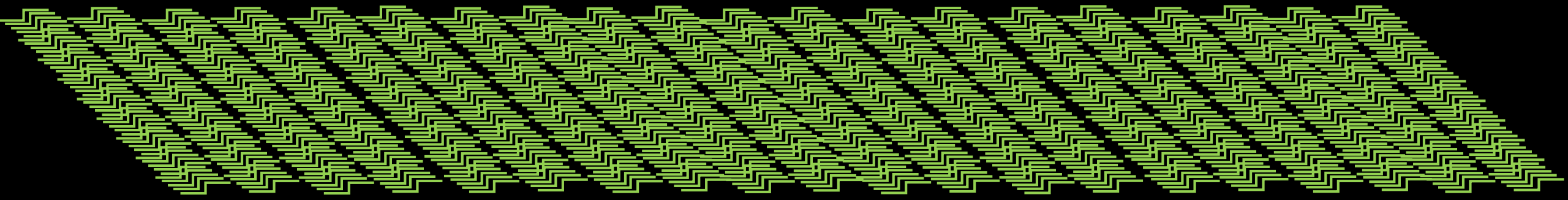
What is a testbench?

- Testbench:
  - Generate stimulus
  - Apply stimulus to Design being Verified (DBV)
  - Capture the response
  - Check for correctness
  - Coverage analysis: Hit/analyze 100% as defined as functional coverage in verification plan.



Testbench speed

- System level testing (~1Hz)
  - Full design, verify system level features
- Core level testing (~10Hz)
  - CPU core only, verify single core behavior
- Multi-unit level testing (~50Hz)
  - Multiple CPU blocks (e.g. Instruction Fetch + Decode)
- Unit level testing (~100Hz)
  - Single CPU block (e.g. Decode)
- Actual silicon (~3000000000Hz)



Emulation

- Synopsys Zebu
  - Emulate smaller IPs (core, etc.)
  - Very fast, limited debug visibility

http://www.synopsys.com/Tools/Verification/hardware-verification/emulation/Pages/zebu-server-asic-emulator.aspx

- Cadence Palladium XP
  - Emulate a full System on a chip
  - Fast; great debug visibility

http://www.cadence.com/products/sd/palladium_xp_series/pages/default.aspx

Emulation accelerates simulation and helps find corner-case bugs

What about formal verification?

	Functional	Formal
Operate on large designs over 10M nand gates	Yes	No
Verification scheme	Stimulus: - Directed (hand written) - Random (auto generated)	Constraint driven exhaustive testing
Designs	Full cpu, IP, SOC, etc.	Small blocks: Multiply, Divide, CRC, Floating-point, etc.
Development work	Testbench, stimulus, reference models, etc.	Lemmas, constraint, etc.
Connectivity checks	Yes	Yes
Power aware verification	Yes	No

What's not found?

Easy to find bugs

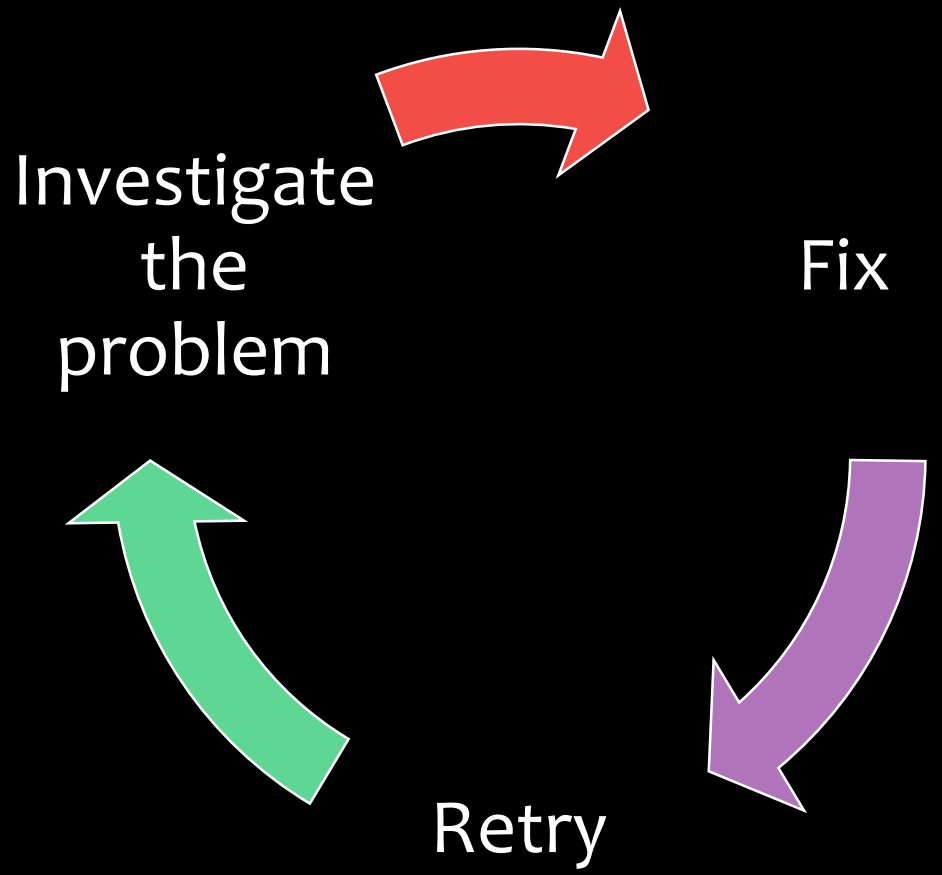
- Basic functional behavior (functional verif)
  - Does this mode work?
  - Are exceptions correctly generated?
  - Cache coherency
- Formal proofs (formal verif)
  - Is multiplier output correct?
- Coverage holes
  - Can all exceptions be generated?
  - Are all instructions executed?

Hard to find bugs

- System level behavior
  - Protocol violations
  - FIFO overruns/underruns
- Multiple random events
  - 5 unlikely asynchronous events in a row
- Long runtime events
- Statistically unlikely matches
  - Multiple loads where 20 address bits match

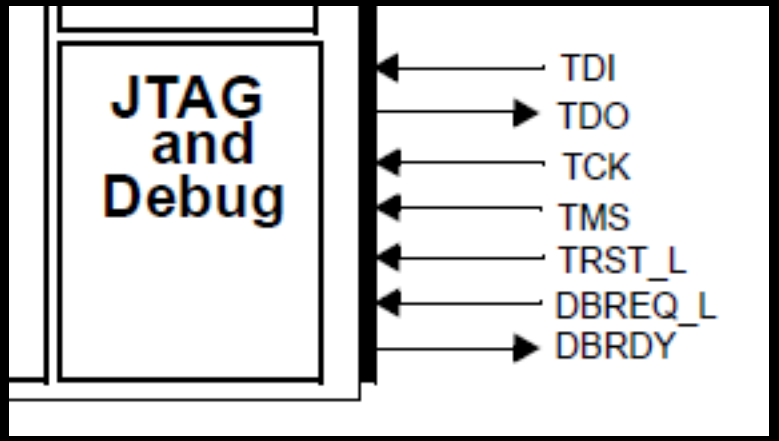
Validation (Post-silicon)

Debug cycle



What happened?

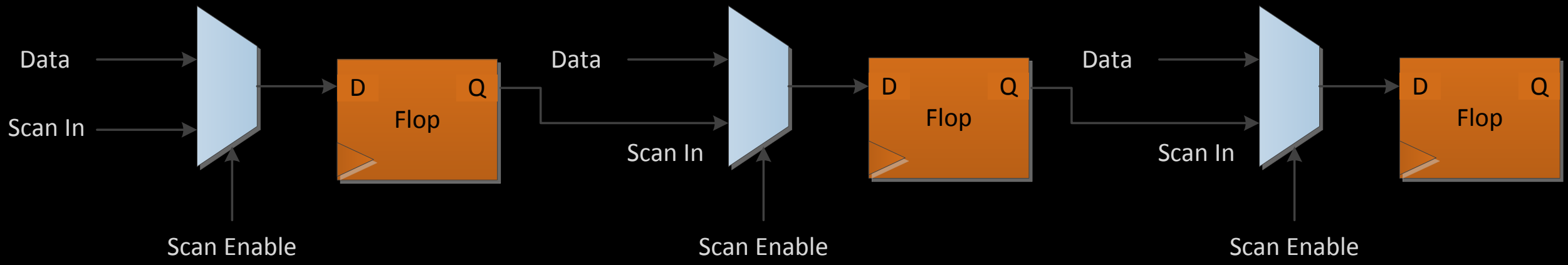
- Modern processors support JTAG (IEEE 1149.1)
- JTAG hardware implements various commands
  - BYPASS
  - EXTEST
  - IDCODE
  - Anything else the vendor wants
- JTAG commands can be used to implement a simple debugger
  - Read/write CPU register state
  - Read/write memory
  - Read/write I/O
  - Single-step execution
  - Set debug breakpoints
  - Etc.



http://support.amd.com/TechDocs/31412.pdf

What if the CPU is hung?

- Answer: Scan!
- Scan enables reading all flip-flop state in the CPU (like a crash dump)
- Flops are chained together to enable shifting out data through the JTAG port
- Limitations:
  - Only get data in flops
  - Only from a single point in time
  - Interesting information may be long gone



Ok, we found the problem...now what?

- To fix the issue, we may need to re-spin the silicon
  - But we may not need to change the entire design
  - Typical chips may have a base layer and up to ~9 metal layers

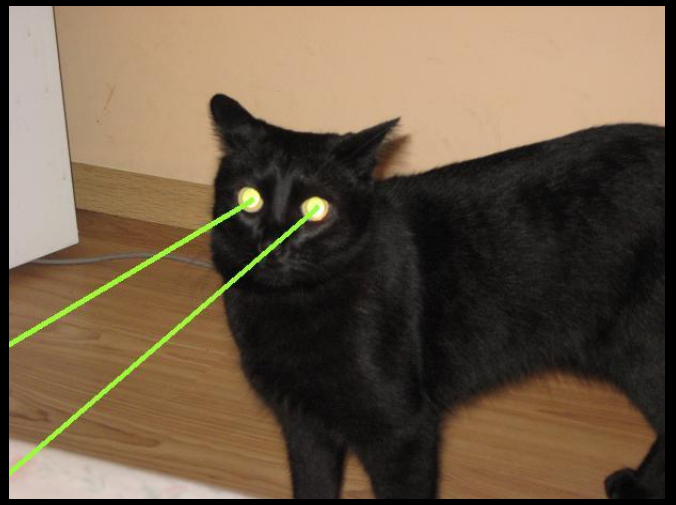
- **Base layer**
  - Implements the logic gates in the design
  - $$$ to remake, long delay through the fab
- **Metal layer(s)**
  - Wires up the various logic gates in the design
  - $$ to remake, shorter delay through the fab

- Often, extra gates will be built so simple changes can be made in metal

```
if (~Reset)
   DoStuff<=1'b0;
```

```
/* MUCH better now */
if (~Reset)
   DoStuff<=1'b1;
```

Less costly solutions



http://nimfpa.deviantart.com/art/Laser-cat-201298166

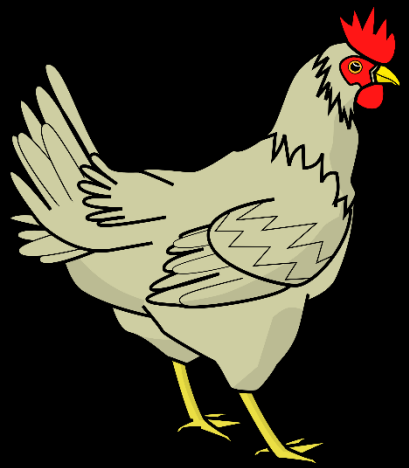
By english User:Cm the p (English version) [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/)], via Wikimedia Commons

Practical, less costly solutions

- Designers will often build “disable” bits for risky features (aka chicken bits)
- Typically used for performance or power enhancements

MSRC001_1022 Data Cache Configuration (DC_CFG)

Bits	Description
63:16	Reserved.
15	**DisPfHwForSw**. Read-write. Reset: 0. 1=Disable hardware prefetches for software prefetches.
14	Reserved.
13	**DisHwPf**. Read-write. Reset: 0. 1=Disable the DC hardware prefetcher. BIOS: See 2.3.3 [Using L2 Cache as General Storage During Boot].
12:5	Reserved.
4	**DisSpecTlbRld**. Read-write. Reset: 0. 1=Disable speculative TLB reloads. BIOS: See 2.3.3 [Using L2 Cache as General Storage During Boot].
3:0	Reserved.



http://support.amd.com/TechDocs/49125_15h_Models_30h-3Fh_BKDG.pdf

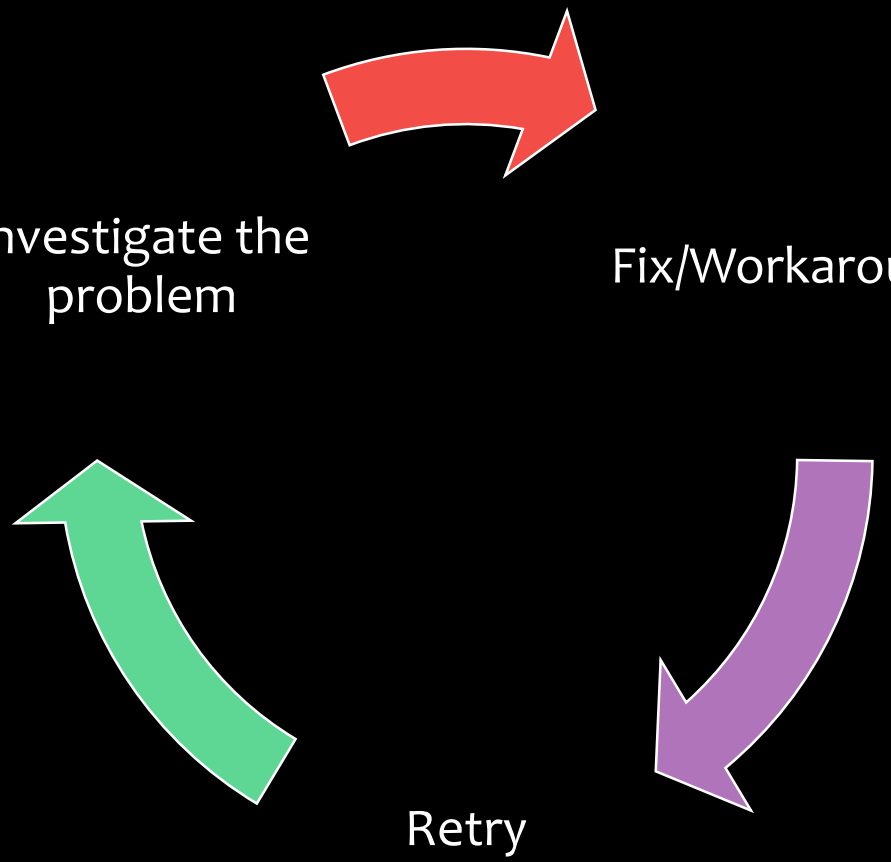
Microcode patch

- Modern x86 CPUs use microcode to implement more complex functionality such as
  - Complex x86 instructions (e.g. IRET, RSM, etc.)
  - Interrupt delivery
  - Power management
- Microcode is built into the silicon, but a patch RAM exists
  - Can be used to replace/supplement existing microcode to fix bugs
  - Limited in size and capability
- Microcode can implement workarounds like:
  - Serialize the machine after CLFLUSH to work around a memory bug
  - Handle a rare corner case during a hardware task switch
  - Etc.

- For more info on microcode patch loading, see the Linux patch loaders under arch/x86/kernel/cpu/microcode/

Putting it together

JTAG debugging
Scan



Base spin
Metal spin
Microcode patch
Chicken bits
FIB

A little note on security

- *Do you really know who's using your ~~backdoors~~ debug interfaces???*

- Debug interface security should always be considered and tested
- Typical solutions
  - Disable some or all JTAG commands (including scan) on production parts
  - Ensure debug access to sensitive information is blocked in production
  - Require authentication to use a debug interface
  - Require signed CPU microcode updates

Takeaways

Takeaways

- Testing
  - Break down large designs into small, manageable chunks
  - Use tools to ensure you're getting the most out of your test time
  - Know your weaknesses!
- Design for failures
  - Build in debug features if appropriate
  - Anticipate the risk areas and how you can fix them
  - A software upgrade is not always the answer

- Please secure your debug interfaces ☺

Further resources

- Actual CPU technical documentation
  - BIOS and Kernel Developers Guide (BKDG)
    - e.g. http://support.amd.com/TechDocs/49125_15h_Models_30h-3Fh_BKDG.pdf
  - CPU Revision Guides
    - e.g. http://support.amd.com/TechDocs/51810_16h_00h-0Fh_Rev_Guide.pdf
    - Or http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/4th-gen-core-family-desktop-specification-update.pdf
    - Or just google it
- Verification resources
  - https://www.youtube.com/watch?v=dKFwQNsXaNU (Overview of Modern Functional Verif)
  - https://www.youtube.com/watch?v=266ub4vb_H8 (Challenges in Functional Verif)
  - https://www.youtube.com/watch?v=Q2m10MEB2Ak (Technology Evolution in Functional Verif)
  - Or just google “what is functional verification”