When Silicon Fails…

Will Robbins

I am Will Robbins and I am the VP silicon for Mindspeed in the UK, formerly Picochip.
On the Picochip website it said of me "he has consistently produced 'right first time' silicon for highly complex devices".
This is no longer true.
I have now suffered with failed silicon. It is horrible, stomach churning and very expensive.

Which of you have not put a bug on silicon?

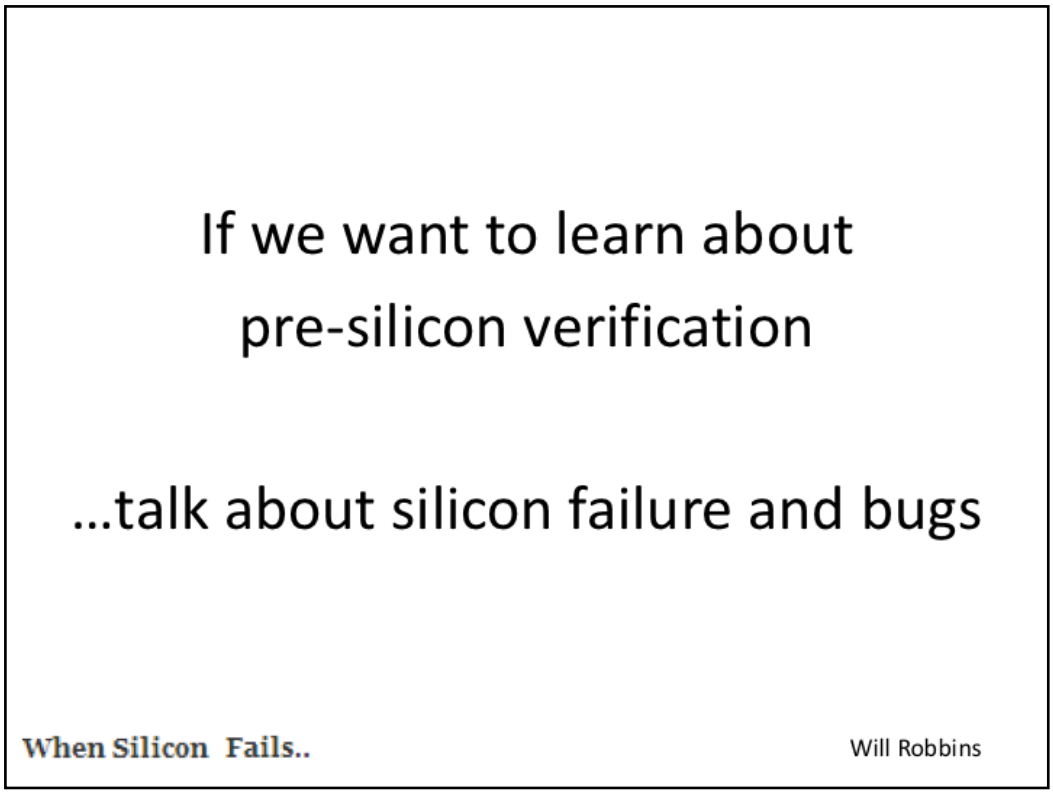
You are very lucky. We all need to help you stay lucky.

So we need to talk about "When silicon fails…"

We ALL need to talk about silicon bugs.
…. because we don't

I'm going explain:
- why I think we need to talk about silicon bugs
- why these bugs evade pre-silicon verification
- and "talk" about some real silicon bugs.

BUT my mission today is to get us talking about silicon failure..



Pre-silicon verification is only a means to an end, it is not an end in itself.

The end point is working silicon.

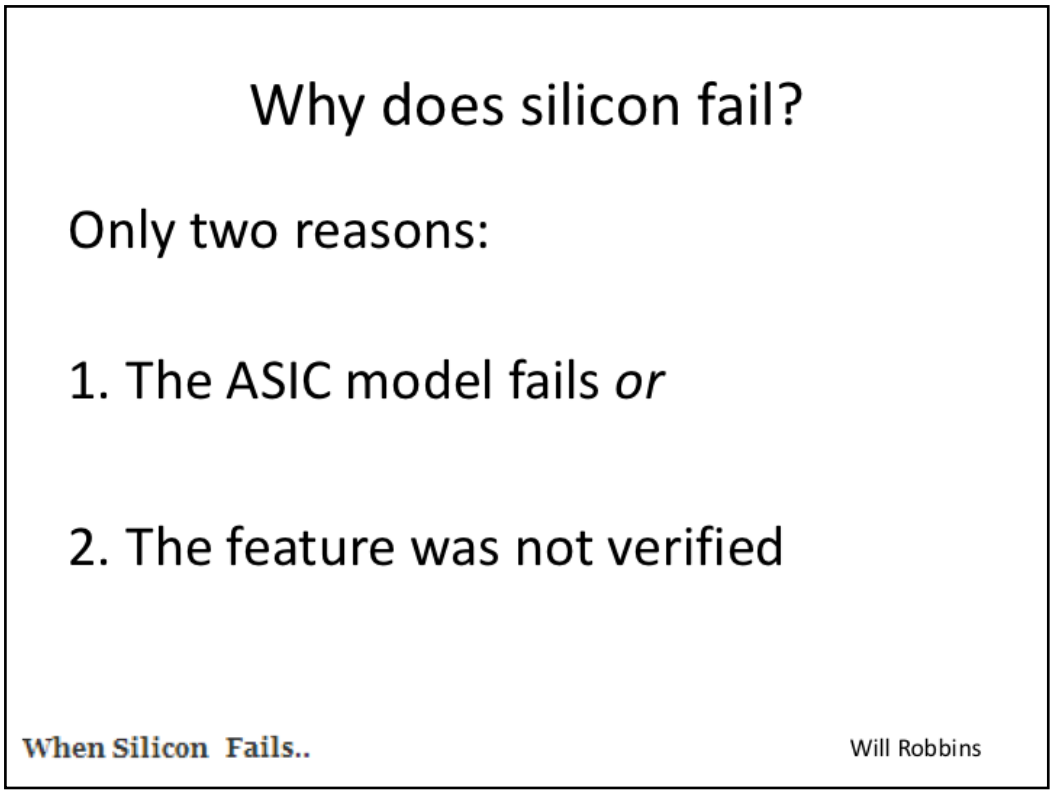
So the only way to judge the quality of pre-silicon verification is whether the silicon works.

If we want to learn about pre-silicon verification we need to talk about silicon bugs and about what could go wrong.

But talking about silicon bugs is unfashionable and bordering on the taboo.

As we have seen today pre-silicon verification has over the last fifteen years reached a very high level of sophistication.

But the silicon validation languishes in the shadows.



I would (boldly) like to propose that there are only two reasons for bugs evading the pre-silicon verification and getting onto silicon

Either "The ASIC model fails"
Or "The feature was not verified"

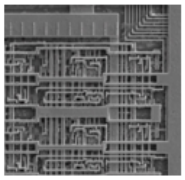
You may argue with this simple proposition, but look at it the other way around:

If a feature has been fully verified and the ASIC model holds then it will function on silicon.

ASIC model failing..

- Vast abstraction

```
always @ (posedge clk or posedge rst)
    if (rst) ct <= {size{1'b0}};
    else if (cet && cep)
        begin
            if (ct == len-1) ct <= {size{1'b0}};
            else ct <= ct + 1'b1;
        end
```



- Impossible to model everything
  - Power supply, black boxes ….
  - clock crossing, hot spots ….
- Engineering judgement is required..

When Silicon Fails.. Will Robbins

Lets consider the first of these – the ASIC model failing

By "the ASIC model" I mean the process by which you can run a zero delay RTL simulation and believe the silicon will function.

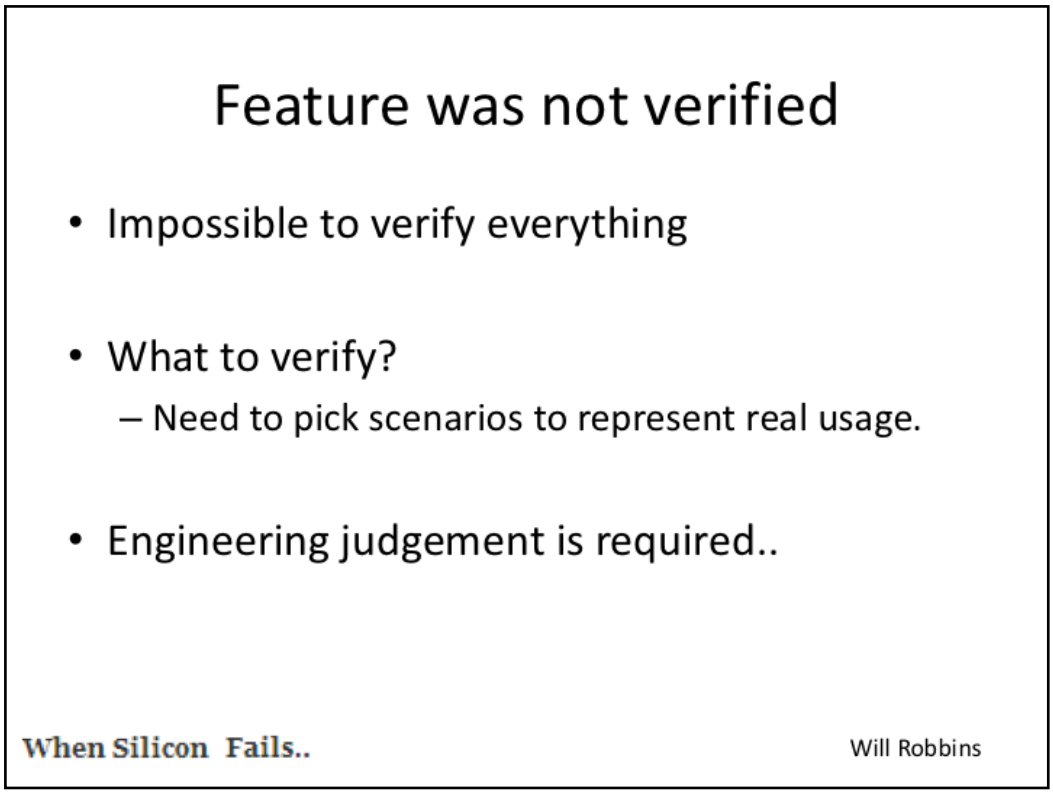
This abstraction is vast – high level code to wires, doping and transistors.

Now it is obviously both impossible and undesirable to "model everything".

But we need to consider what to model to maintain this abstraction:
- How much should the power supply bounce in every part of the chip be modelled?
- Is that block box – be it PLL or standard cell – modelled and characterised sufficiently?
- Has the clock crossing be dealt with?
- Do hot spots on the die take you out of the characterised temperature?

The crucial point here is that ENGINEERING judgement has to be made on what to model and what not to model,

.. and if they are wrong the ASIC model will break, and the silicon *could* fail.



The second reason for silicon failure is the “feature not being verified”.

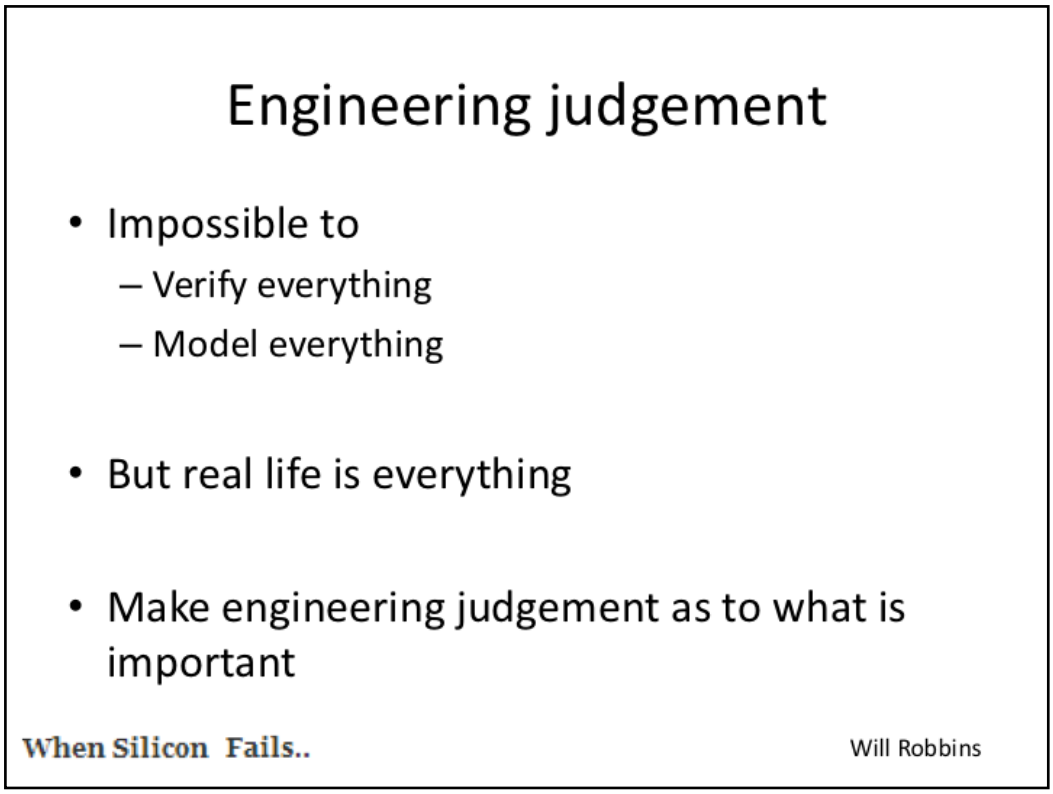
This sounds at best like a lack of coverage or at worst negligent. “What!! You didn’t verify the feature?”

But think about this – What do we fully verify that is much bigger than a NAND gate? We probably don’t even fully verify a multiplier.

So it is IMPOSSIBLE to verify everything.

Instead, as you know, your skill is to pick scenarios that represent real usage.

You have to use your ENGINEERING judgement to pick these scenarios – and if you get this wrong the silicon can fail.



So what I hope that we have established is:

• It is impossible to verify everything

• It is impossible to model everything.

But real life is everything

We have to make ENGINEERING judgements of what is important, and what is not..

So we have good news and bad news.

The good news is that there are only TWO reasons why silicon can fail.

The bad news is that BOTH of these rely on engineering judgement. So there are no GREEN LIGHTS here.

Our industry finds comfort in green lights. - 100% coverage, All tests passing, Zero timing violations, No DRC errors.

But here we do not have this, we have judgement, we have an AMBER light.

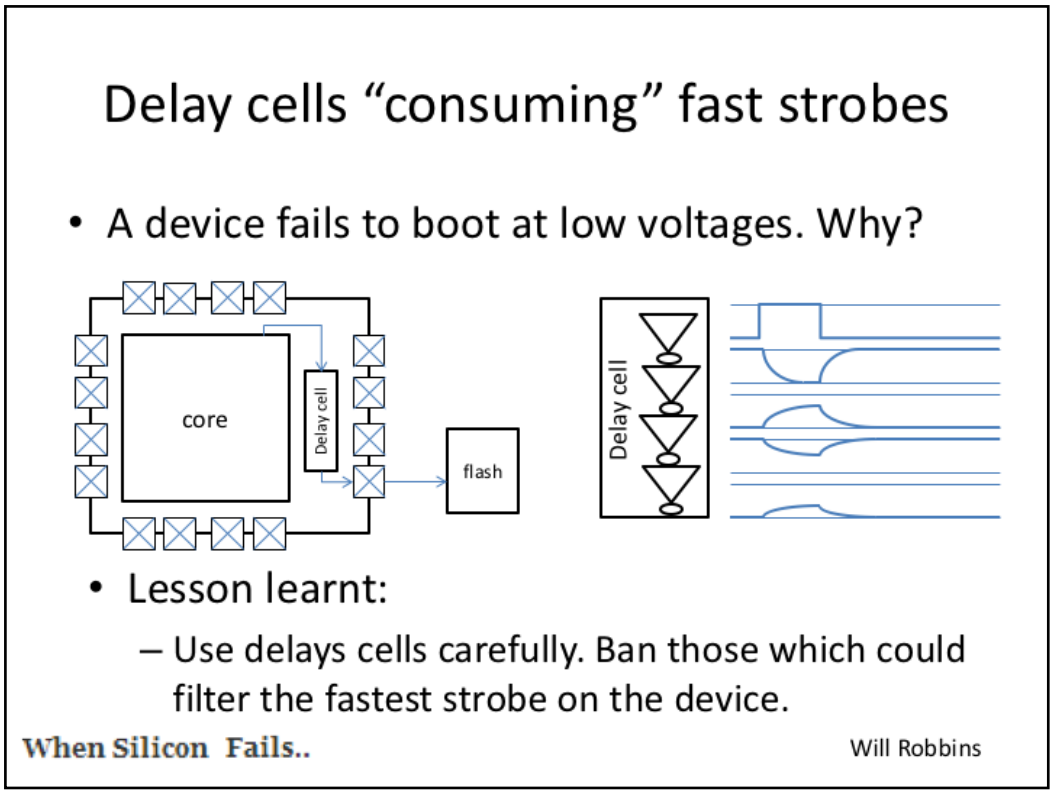
So if these "judgements" are so important how do we get better at them?

We need to learn from mistakes, learn from silicon bugs, when these judgements were wrong. But learning from your own mistakes is painful, much better to learn from everybody's mistakes. So this is why we need to talk about wrong judgement and when silicon fails….

So my mission today is to ask you to talk about and share your stories of silicon failure, and what you learnt from them. Maybe an ice beaker over lunch "tell me about your last silicon bug and what did you learn from it?"

There is an obvious issue here – confidentiality. I don't want you to think that I am ignoring it. But I have been gathering silicon failure stories on my blog (plug coming up..), and in many ways we are all doing is very similar things, we can remove the application, remove the company name, remove the IP vendor and there are still nuggets of wisdom to be found.

So enough of the high level philosophising – lets talk about some bugs. Lets do four quick failure stories before lunch.
All of the are taken from www.whensiliconfails.com



Lets start with a broken ASIC model.

Verifying an SoC, all was going well, but when the voltage is reduced it stops booting.

After lots of investigation the clock to the Flash was found to firstly shorten and then disappear completely over a very short voltage window.

But ALL the pre-silicon verification passed, ALL the STA passed.

(You will understand that here I here condensing several weeks of angst down into 45 seconds).

It was found that a fast clock in the padring was taken through the longest delay cell on the chip – a smoking gun was found.

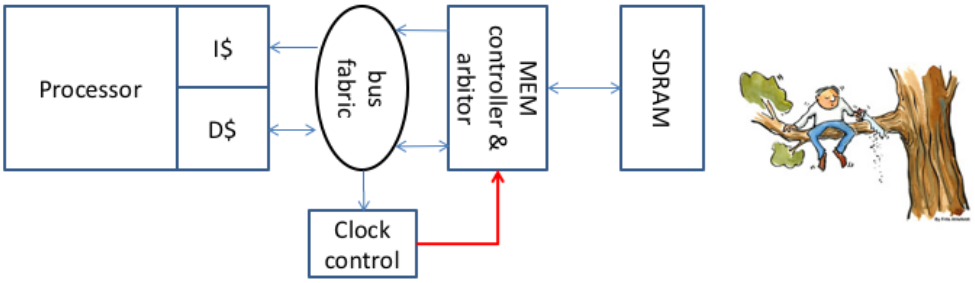
Further investigation with Spice showed that at low voltage this delay cell simply filtered out the clock pulse.
The delay cell no longer performed like its model.

The ASIC model and so the silicon was broken….

The lesson learnt here is to be very careful with delays cells and ban the dangerous ones.

Don’t saw through the branch on which you are sitting

- Requirement to have a variable DDR clock controlled from the processor.



- Lesson learnt:
  - Verify more closely against the requirements

When Silicon Fails..

Will Robbins

This bug is entitled “Don’t saw through the branch on which you are sitting”.

Or in other words “Don’t remove a resource on which you are relying - even temporarily”.

The requirement was to be able to control the DDR clock from the processor.

Unfortunately changing this clock’s speed entailed stopping and restarting it.

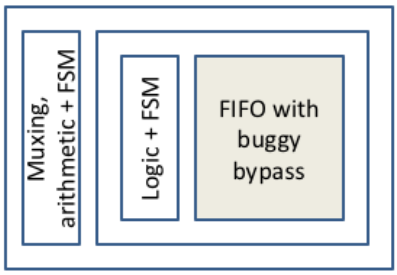
Stopping was fine. But trying to read code to restart it from a SDRAM that was disabled was not going to work.

This may sound foolish and simplistic. But this was put on silicon.

What can we learn from this? Verification should be tied more closely to the requirements.

Those difficult to reach corners … or plain too complicated

- Complicated block with multiple state machines.
- FIFO looses entries when full and simultaneously pushed and popped.
- Lesson learnt:
  - The block was too complicated.
  - Needs more unit testing.



When Silicon Fails..

Will Robbins

For this bug we are looking at a significant block in a wireless Phy.

This block parsed an input stream, de-multiplexed it, and kept track of resources in the SDRAM.

And this was verified at this block level.

The silicon could occasionally see resources run out unexpectedly after maybe an hours operation.

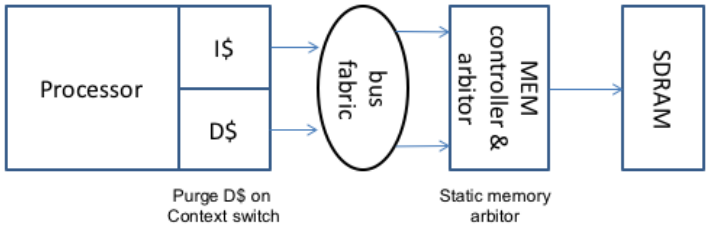
This was tracked down to a FIFO deep in the block loosing entries when it was full and was pushed and popped simultaneously.

It would have been very difficult to specify scenarios that would have stimulated this situation – but it happened in real life.

The lesson to learn here is that the block was to complicated; the FIFO needed unit test.

Processor performance crippled by large caches

- Processor showed good performance, until running more than a few processes when performance plummeted.



- Lesson learnt: Wrong scenario was selected, due to not analysing the end application and operating system sufficiently.

When Silicon Fails.. Will Robbins

Last bug – another "not verified" bug. And one of my favourites as it is counter intuitive.

The performance of this processor was crippled by LARGE caches, not by small ones.

This SoC had very good performance numbers – in pre-silicon verification AND in post-silicon validation

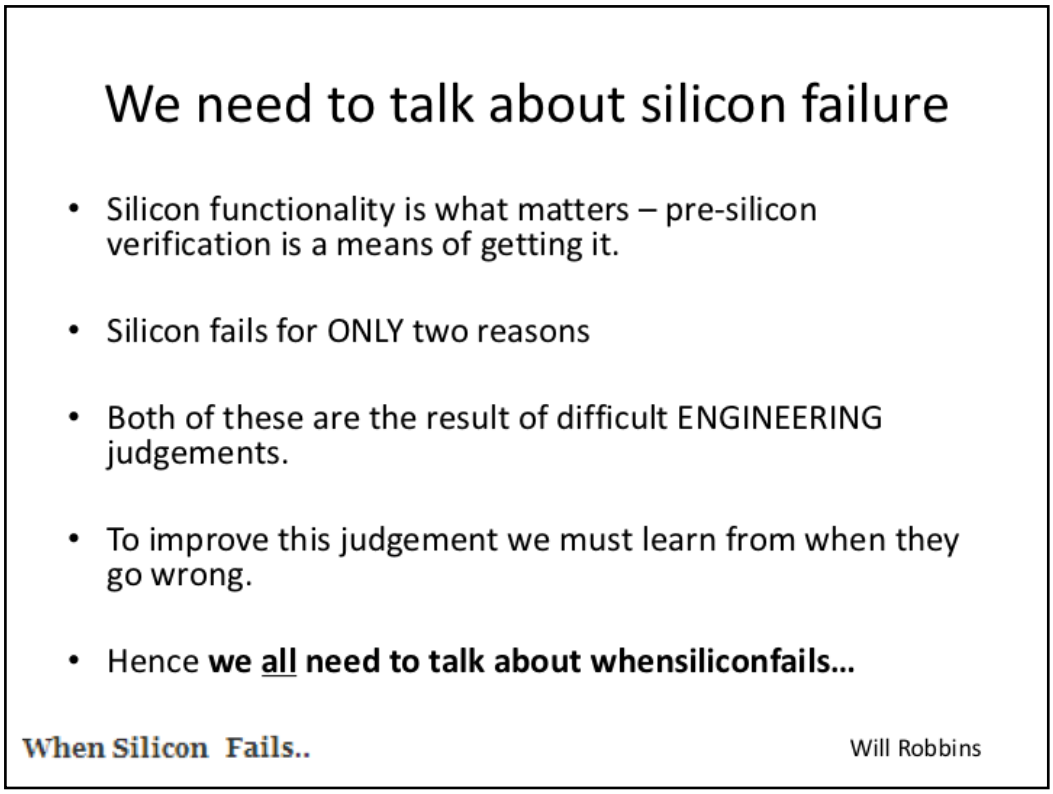
BUT when the number of processes reached about five the performance plummeted.

Again much angst.

What was discovered was that on every context switch the cache was purged of all "dirty" lines. So there were many consecutive writes to memory. And because they were large caches there many, many writes.

The memory controller was optimised for reads.
It could not respond to this deluge of writes and performed very badly.

The verification scenario chosen was read latency. This was wrong.
The lesson here is know the application, and know what the OS is doing in detail.



IN CONLCUSION

It is silicon functionality what matters
pre-silicon verification ONLY a means to an end.

Silicon fails for ONLY two reasons

BUT both of these are the result of difficult ENGINEERING judgement – with NO green lights

To learn about verification
To learn about these judgement

WE need to look to silicon bugs

WE need to talk about silicon failure.

Thank you.

We need to talk about silicon failure

www.whensiliconfails.com

When Silicon Fails..

Will Robbins