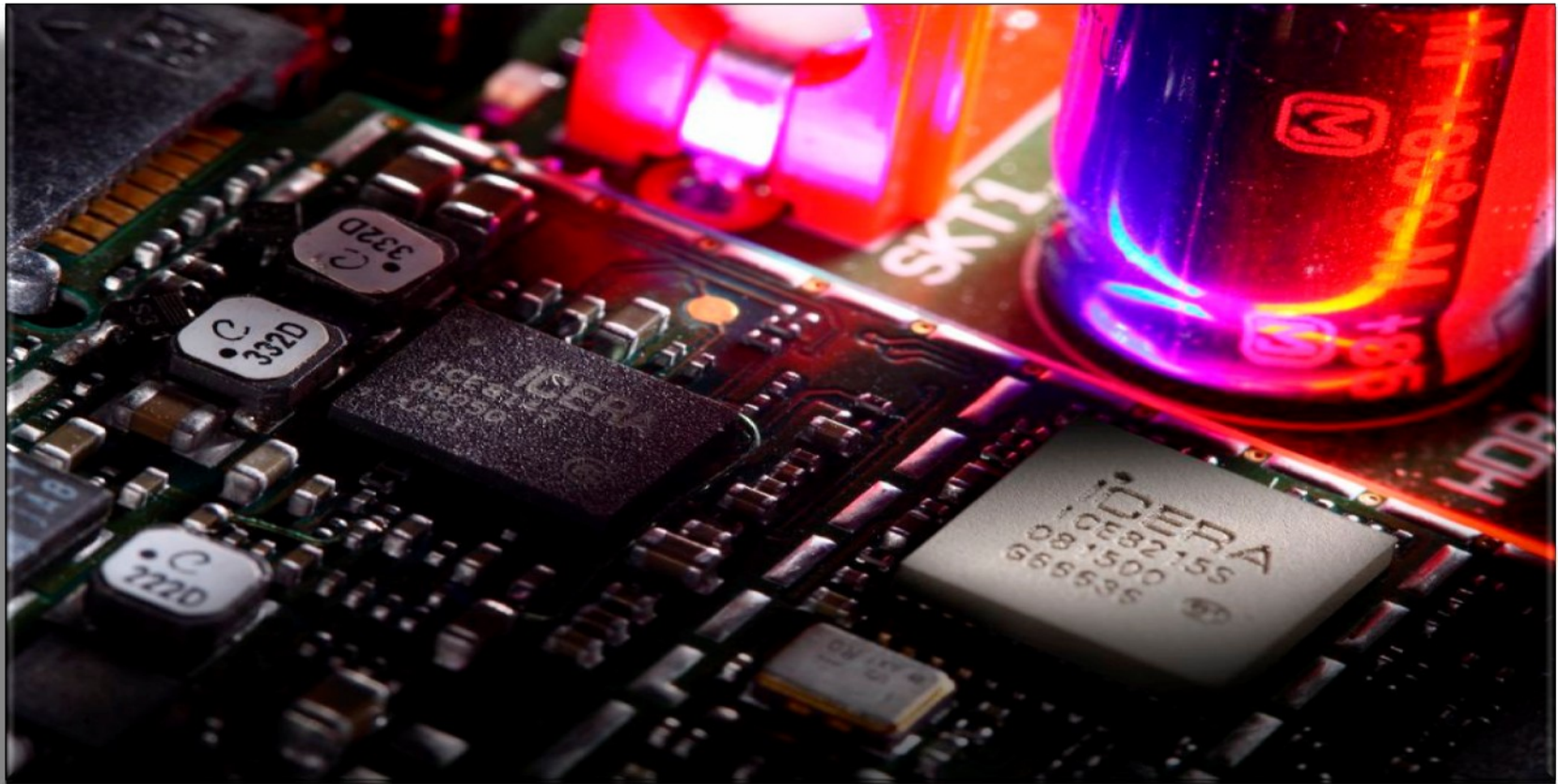


# Adopting SystemVerilog / OVM

---





# Overview

---

- The “Verification Challenge” at Icera
- Why we are interested in SystemVerilog and OVM
- Plus points
- Minus points
- Making the switch less painful
- Conclusions

# Verification @ Icera

---



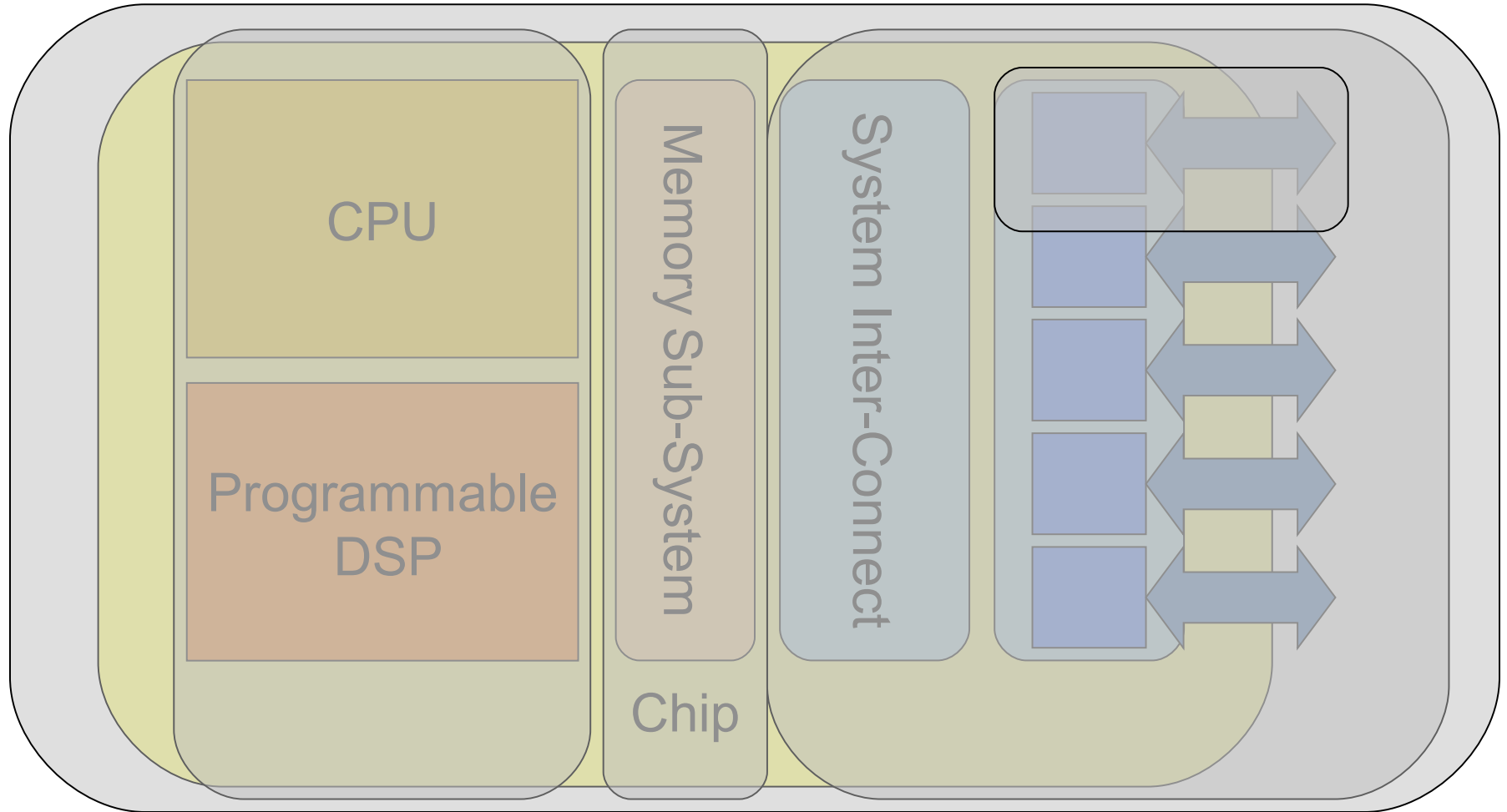


# Verification @ Icera

---

- Two silicon design sites
- Small design team
- Even smaller verification team
- All designers must help with verification
- Verification split into 3 tasks:
  - Designers own “smoke” testing
  - Sign-off verification environments
  - Formal Property Checking
- Cadence Incisive customer

# Verification @ Icera



# Why SystemVerilog / OVM?

---





# Why SystemVerilog / OVM?

---

- The “Open” in OVM
  - Vendor / License independence
  - Visibility of source code
  - A World full of developers
- Methodology seemed sound
  - Evolved from the eRM/URM we have previously used
  - Incorporates another mature methodology in AVM
- SystemVerilog is still fundamentally a HDL
  - More familiar to “part-time” verification engineers
  - Native to the simulator
- SystemVerilog class based approach familiar to C++ programmers

# Plus Points

---

- Getting started is quick
  - LRM in general provides good SystemVerilog examples
  - The OVM Reference and User Guide are pretty good
  - Can get good feedback from the OVM Forum
- Successfully created “real” verification environments
  - Roughly equivalent development time as equivalent e environments
  - Including multiple layers of UVCs



# Plus Points

---

- Constraints aren't as complex
  - Strangely this can make both the development and debug easier
  - So far haven't encountered a blocking issue
- SystemVerilog seems a natural testbench language
- The DPI is a massive improvement over the PLI/VPI

# Minus Points

---

- Vendor support
  - Supported language features aren't always known before you find them
  - Can make design choices hard – lots of simple test cases
  - Debug features aren't as mature as they are for other supported languages
- AVM and URM
  - Not 100% harmonious – you will be referred to the old documentation
  - Some concepts are blurred – sequences/scenarios, comparators

# Minus Points

---

- Few good quality examples of more complex concepts
  - Those provided with the source code vary in style, quality and documentation level
  - OVM Forum examples can be hard to find
  - The Factory: where, when and how to use isn't always obvious
  - The definition of a master / slave
- Coverage / Covergroups
  - Statically defined
    - Goes against the factory based approach of OVM
    - Can make creating coverage base classes tricky
  - Flexibility of bin definitions
  - Multi-cycle coverage requires support code

# Making the Switch Less Painful

---





# Making the Switch Less Painful

---

- Start from the bottom
  - A UVC doesn't have to relate to a specific design block
  - Get your base classes right
    - There is no such thing as a “soft” constraint
    - It's easier to add something than take it away
- Randomized scenario driven verification over constrained random
  - Start with what the design has to achieve and work outwards. With the more limited SystemVerilog constraints, this is a more natural fit
- Combinatorial / multi-cycle interfaces are still awkward
  - Try and keep to a TLM even when the interface isn't a simple fit
- The print statement is your new best friend!

# Conclusions

---



# Conclusions

---

- You can create an “Advanced Verification Environment” using SystemVerilog and OVM as it stands today in roughly the same timescales as other advanced verification languages
- Part-time verification engineers **do** seem more comfortable creating SystemVerilog infrastructure than with other advanced verification languages
- Vendor support is getting better
- Yet to see the vendor-specific value-add
- Yet to see how “Open” OVM really is
- SystemVerilog / OVM is just a tool and not a miracle
  - **There is no substitute for knowing what you are verifying and why**

# THANK YOU

---

