

# Design

## Simplicity and Executability: Cornerstones of Quality

**Mike Keating**  
**Synopsys**



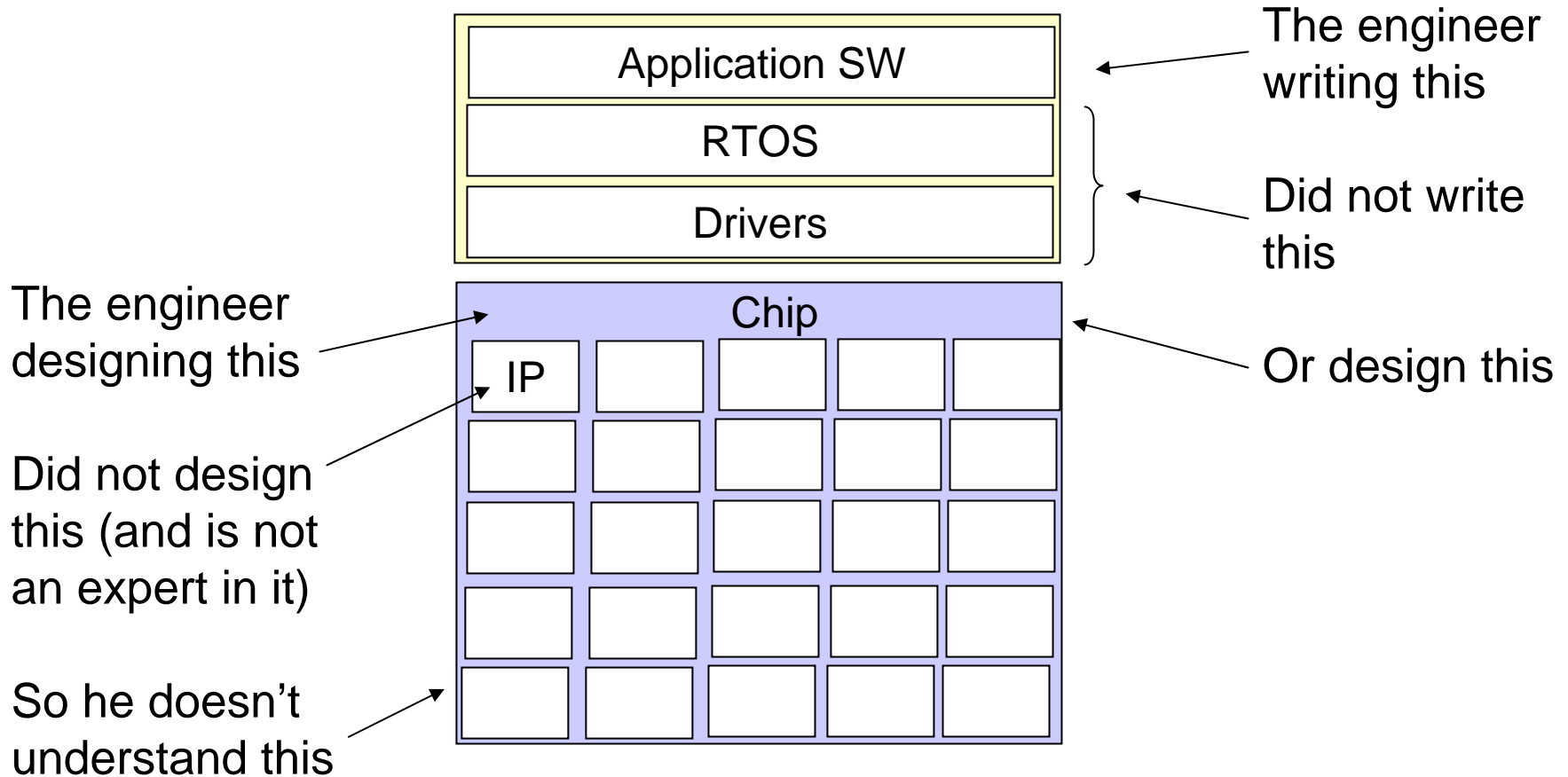
# Two Basic Rules of Design

- **If it's not tested, it's broken.**
- **If it's not simple, it will never work.**
  
- **The most important principles**
- **The most often violated.**

# IP-based SoC Design

- **Every SoC design is IP based**
  - **Internal IP**
  - **Third Party IP**

# Discontinuities of IP-based Design

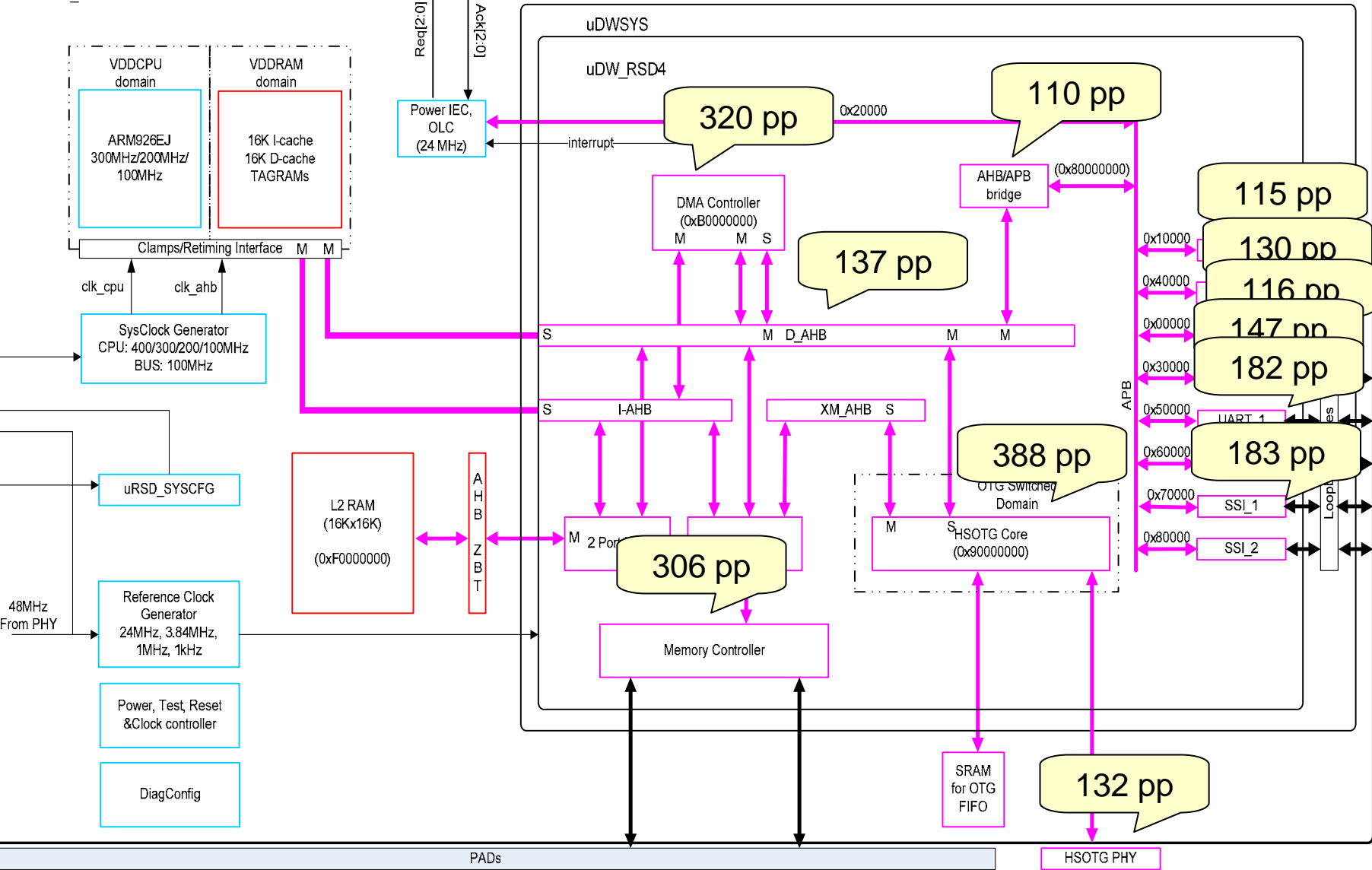


**Result: No One Knows What's Going on**

# IP Meets the Real World

- **Test chip projects**
- **Let's start by reviewing the documentation**

uRSD926\_CORE



132 pp

HSOTG PHY

# Integrating, Testing an IP-based SoC

- **I didn't read 2500 pages of documentation**
- **I used example designs – VTB**
- **I used documentation as a reference**
  - **But found discrepancies with VTB**
  - **Found errors in description of new feature**

# Debugging an IP-based Design

- I didn't read 2500 pages of documentation
- AE's could help identify the symptoms
- Then read the code
  - Faster than reading the documentation
  - More accurate than the documentation
  - It can be explored through simulation
  - Documentation missing some key info
    - `defines for PHY

# The Documentation Trap

- **Documentation is not testable**
- **Therefore it is not tested**
- **Therefore it is broken**
  
- **User docs can be guidelines to the code, but they do not solve the communication challenge between IP provider and SoC designer**

# Implications for User Docs

- **User docs need to provide good overviews**
  - **Block diagrams**
- **Data should be extracted from code**
  - **Pin definitions**
  - **Register definitions**
  - **Parameters and configuration options**
- **Usage information should be in examples**

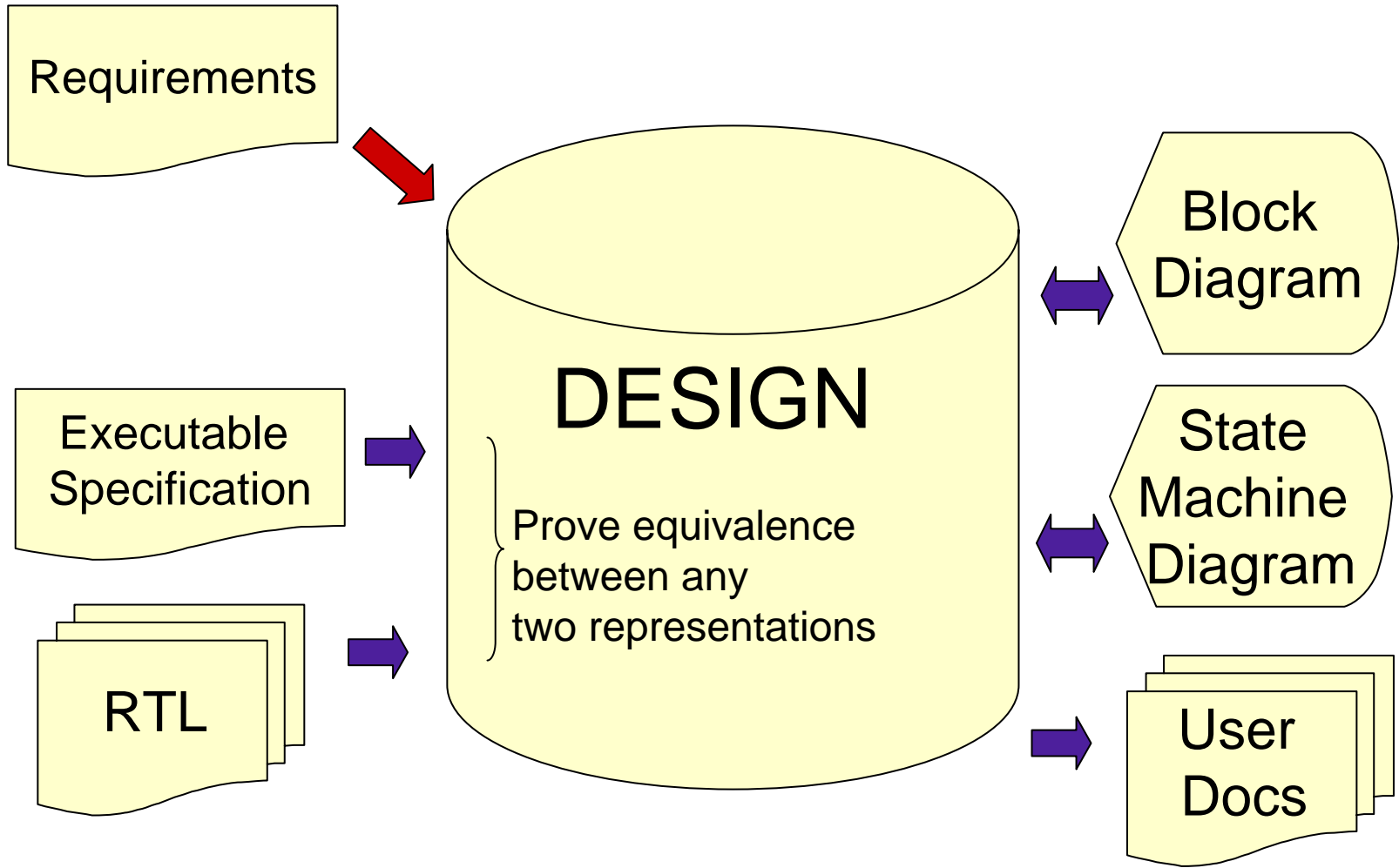
# Implication for Other Docs

- **Requirements**
  - **Fundamentally different from design**
  - **Need to be text-based document**
- **Specification**
  - **It it's not tested, it's broken, so it's broken**
  - **Executable specification**
  - **Key information guaranteed to be consistent with code**
  - **Code should be extracted from specification**

# Implication for Code reviews

- **Code/design review as proof of compliance with spec is fundamentally flawed**
- **Compliance with requirements must be testable**
  - **Plug-fests, gate count, MHz, Power**
- **Compliance with specification must be testable**
  - **Executable spec and code run same diags**
  - **Proof of equivalence with formal tools**
- **Comments**

# Design as a Database



# Part 1: Conclusions

- **“If it’s not tested it’s broken”**
  - **The non-testability of documents severely limits their effectiveness in the design cycle**
- **The specification-code-review process needs to be (and is being) re-worked around the concept of an executable specification**
- **User documentation is not the key to solving the IP usage model and the problems with IP-based SoC design**

# The Big Conclusion

- **Code quality counts**
- **Code ultimately is:**
  - **The specification**
  - **The user documentation**
  - **Code reviews are still the best chance to find the bugs testing doesn't**

Code – and data extracted from the code – is the core form of communication between designers in the IP-based SoC domain.

# Design

## Part II: Simplicity



# What is a Good Design

- **The simplest design that implements the required functionality and meets other design requirements**
- **Easier to:**
  - **Get right**
  - **Fix**
  - **Use**

# What is Good Code?

- **Code that implements the correct function**
- **Code that can be understood by**
  - **The IP architect**
  - **The IP implementer**
  - **The IP verification team**
  - **The IP maintenance team**
  - **The SoC architect**
  - **The SoC implementer**
  - **The SoC verification team**

We are writing code for an audience now.

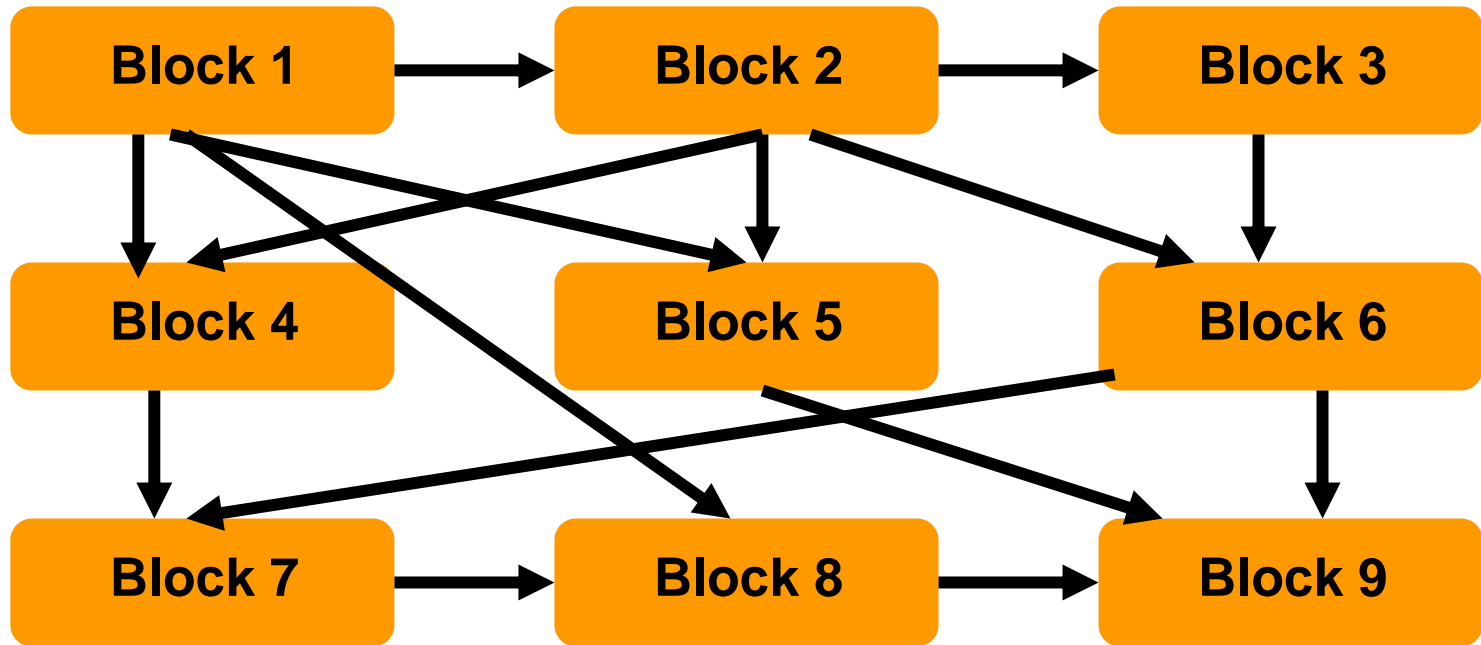
# Good Code is Simple Code

- **Written once, read many times**
- **Must be read by others**
  
- **Simplest code has the fewest bugs**
- **Simple code is more likely to work in untested scenarios if it works in tested scenarios**

# What is Simple

- **Lack of implicit structure (all structure is explicit)**
- **Presence of a clearly perceptible pattern**

# Bad Structure can be Invisible



# Explicit Structure

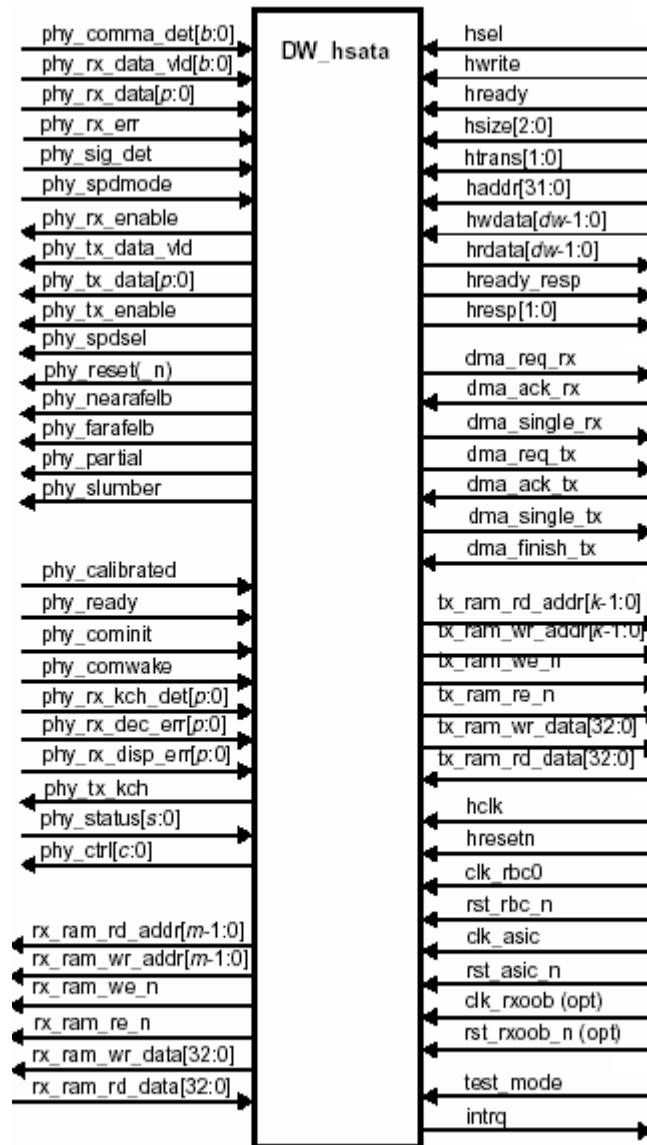


Figure 18: DW\_hsata I/O Diagram

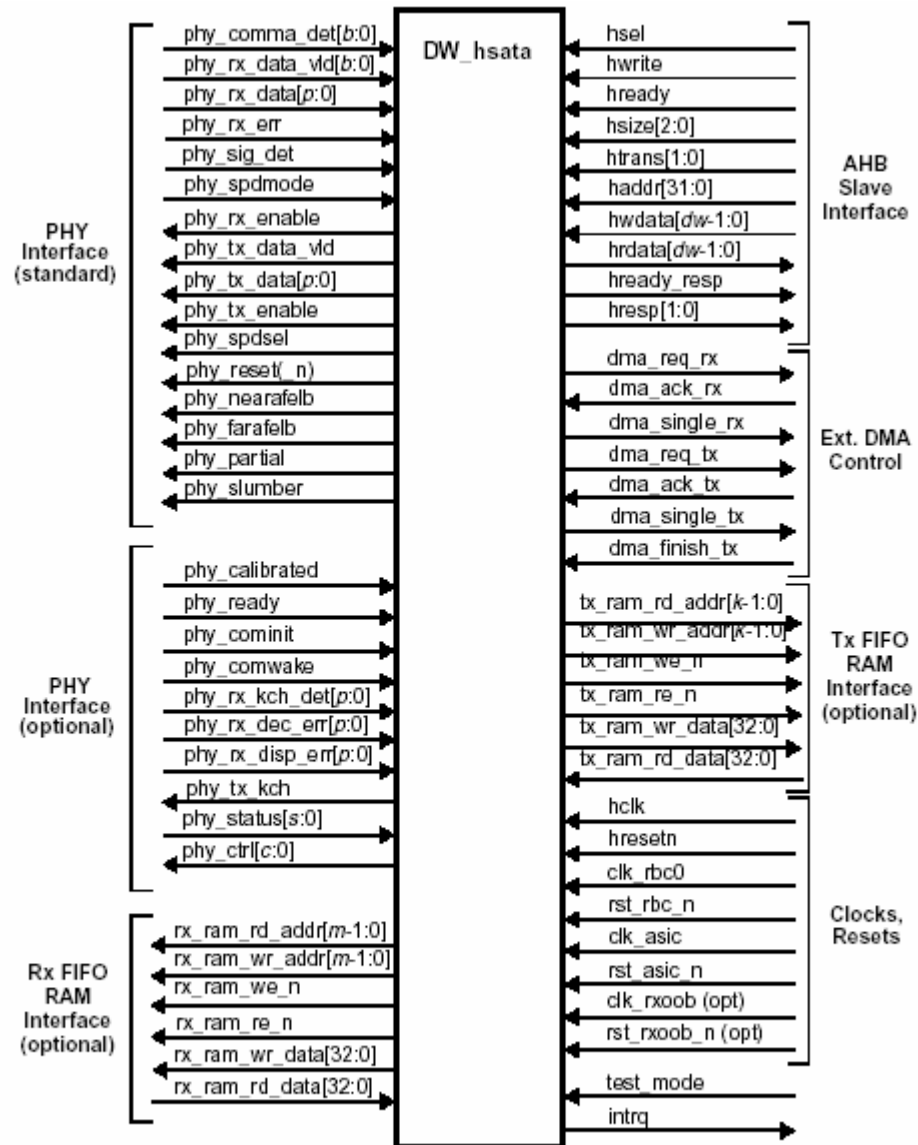
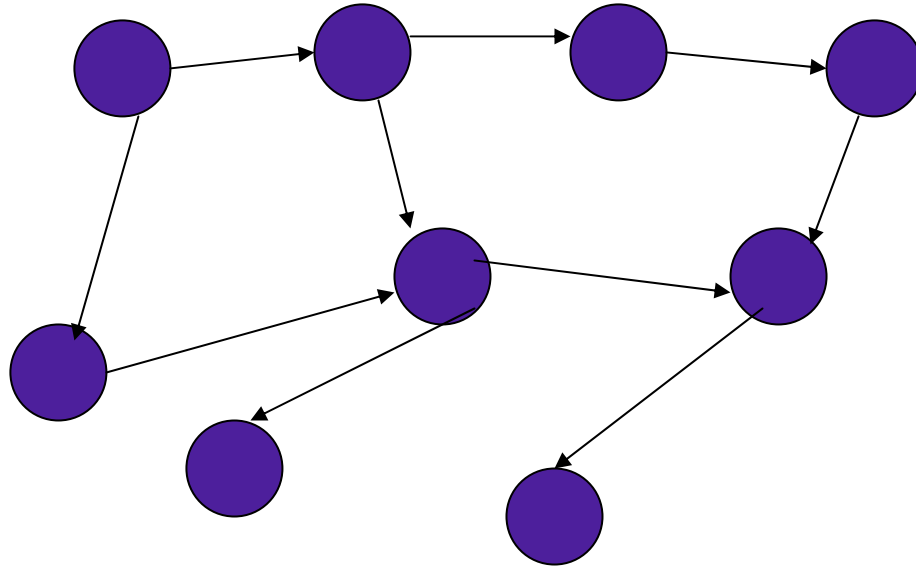


Figure 18: DW\_hsata I/O Diagram

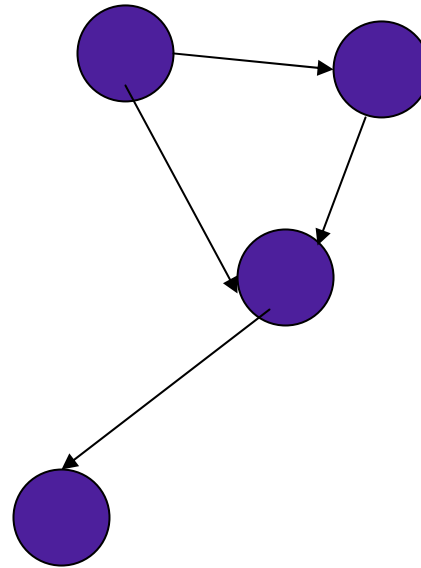
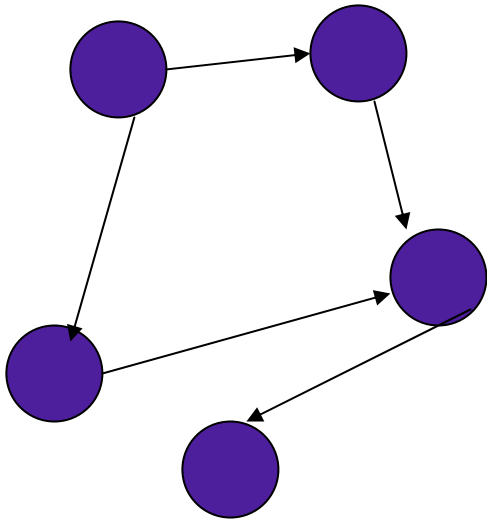
# State Machines – One Big, Flat One



State space is explicit

Substructure is not

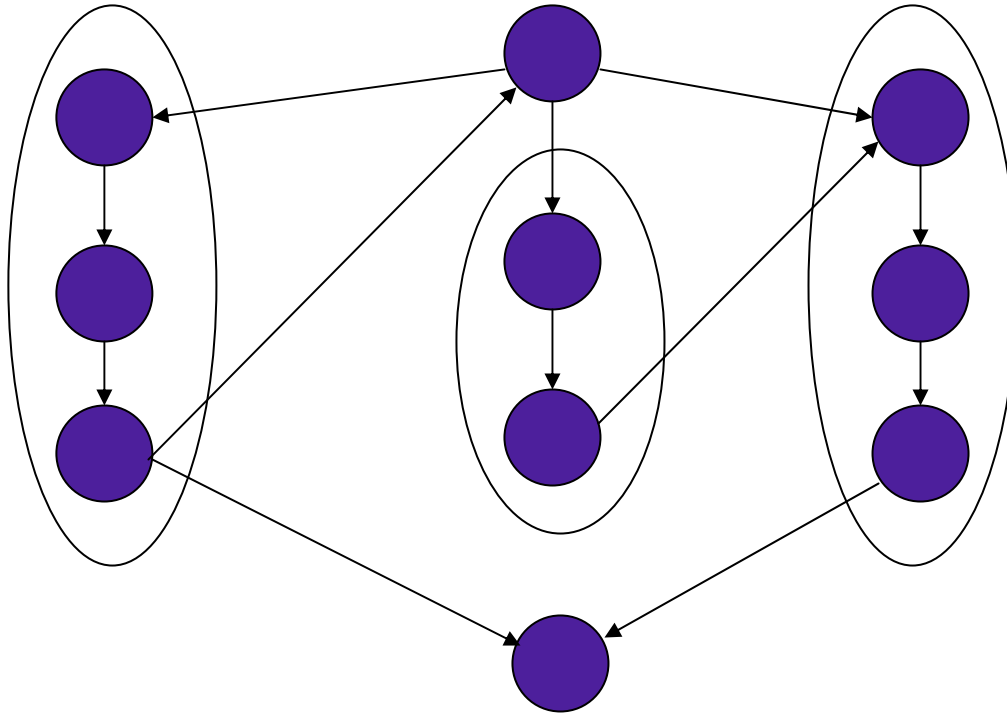
# Multiple State Machines



Structure is explicit

State space is not

# Hierarchical State Machines



State machine consists of 2 state and 3 composite states

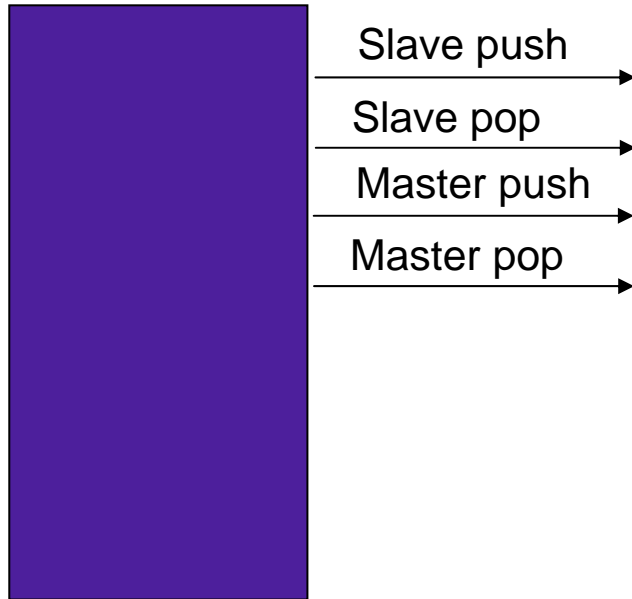
Each composite state has exactly one entry state and one exit state

Only one state is active at a time – no concurrency

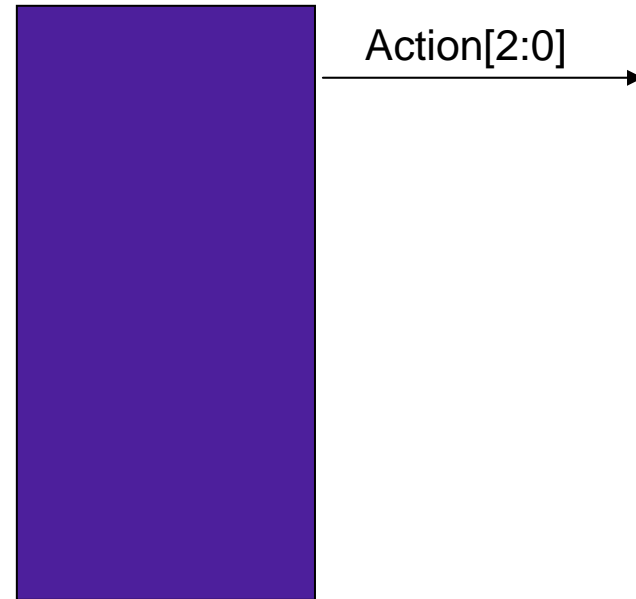
State space is explicit

Structure is explicit

# Wires vs. Messages



Wire States (16)



Messages (5) :  
Slave Push  
Slave Pop  
Master Push  
Master Pop  
None

# Perceptible Patterns

2 x 11 x 17

- You must be able to see it all at once
- Pattern must consist of a reasonable number of elements
- Regularity of structure

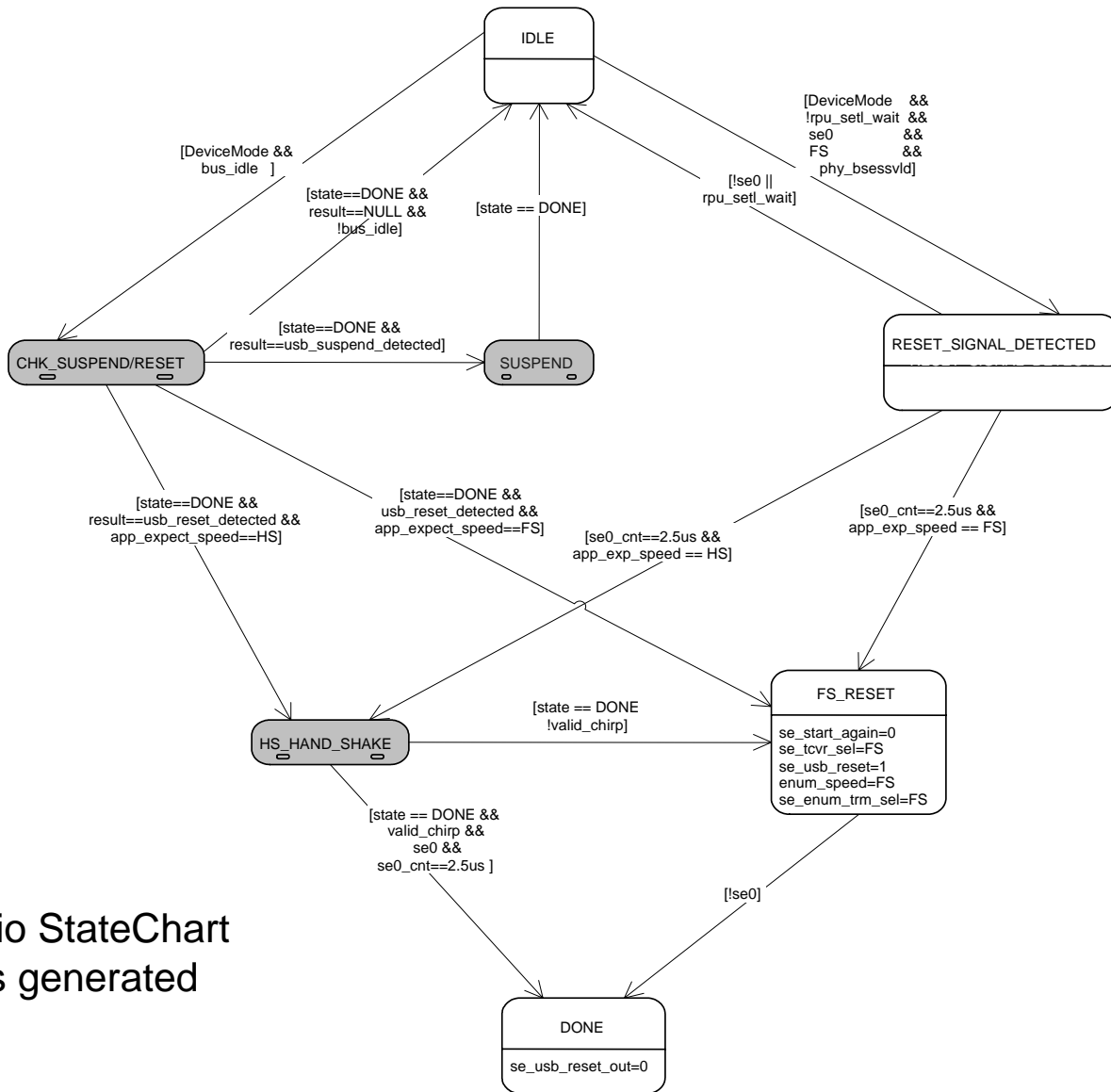
Rule of 7

Similar things  
similarly  
represented

# Perceptible Patterns

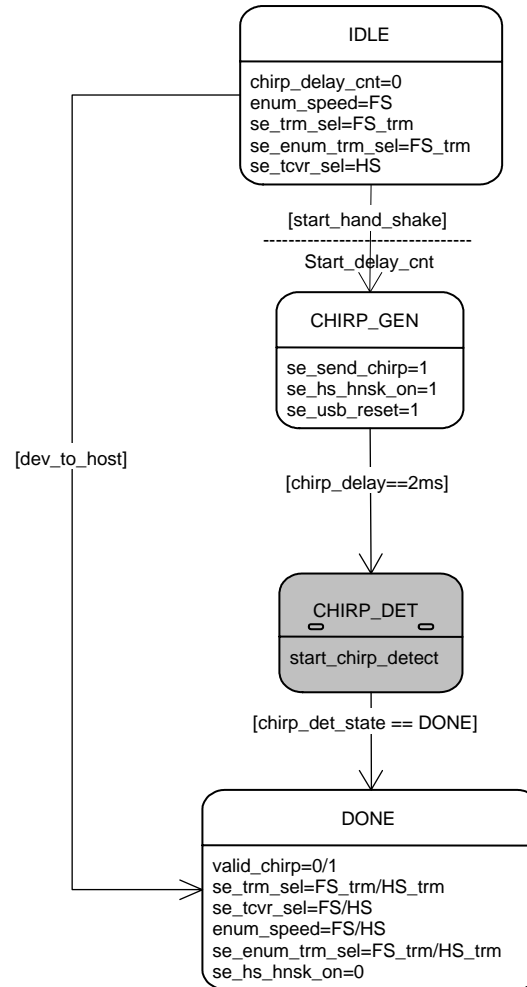
- **You can create structure in code**
- **But you can't see structure in code**

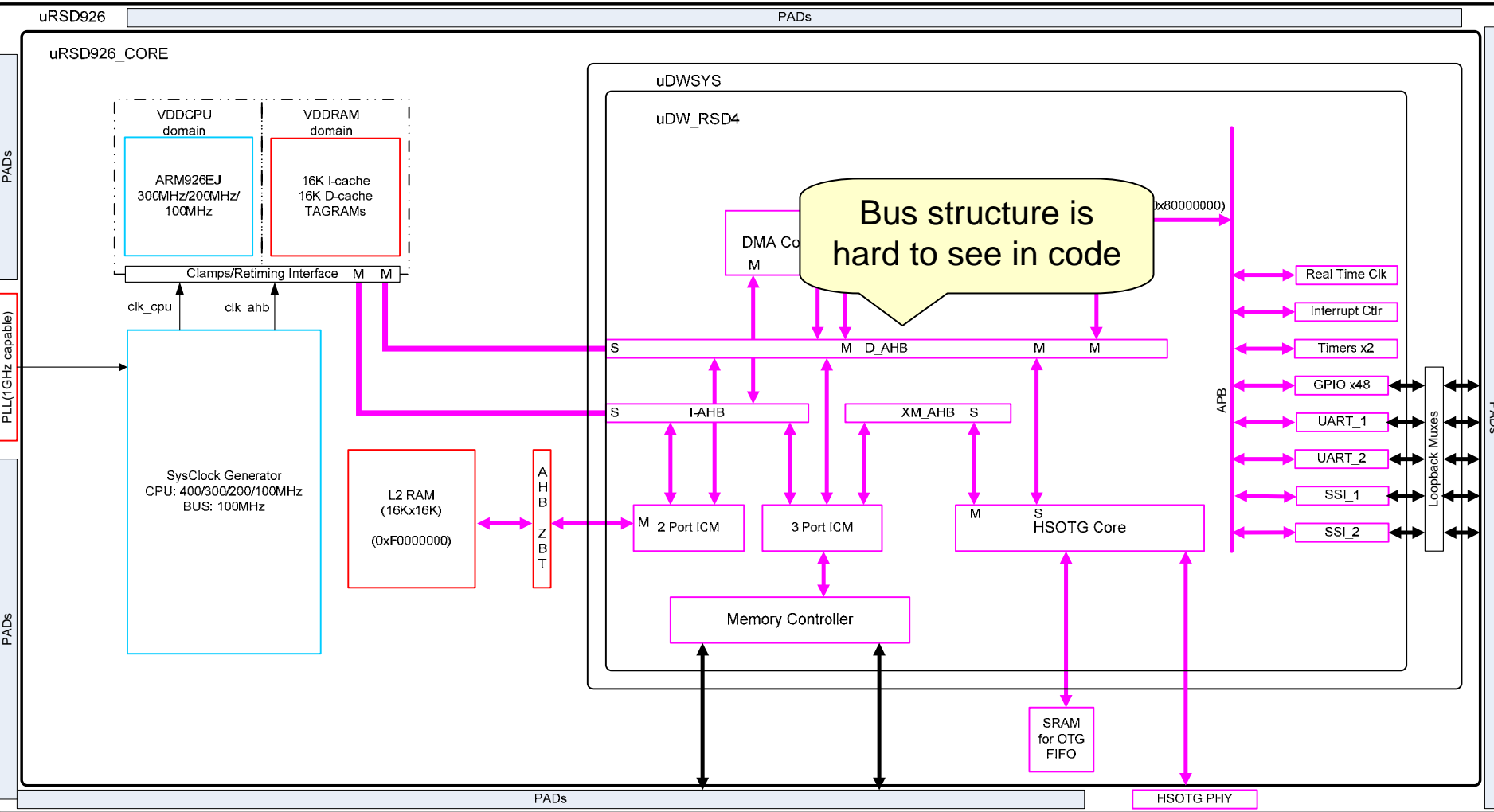
# Suspend/Resume State Machine - Top



Created in Visio StateChart  
Verilog code is generated  
by add-on

# Sub State-Machine (HS\_Hand\_Shake)





Bus structure is hard to see in code

# System Verilog

- **Interfaces**
- **ANSI C port declarations**
- **Structs, unions, etc.**
- **Explicit flops (always\_ff)**

# Memory Controller Instantiation

6 lines  
System Verilog

60 lines  
Verilog

```
memctl i_memctl(  
    .hclk      (hclk),  
    .hresetn   (hresetn),  
    .sdram_ret_clk (sdram_ret_clk),  
    .ahb       (ahb_memctl),  
    .memctl    (memctl)  
);
```

```
DW_memctl dut  
(.hready_resp (dut_hready_resp),  
 .hresp      (dut_hresp),  
 .hrdata     (dut_hrdata),  
 .s_ras_n    (dut_s_ras_n),  
 .s_cas_n    (dut_s_cas_n),  
 .s_cke      (dut_s_cke),  
 .s_wr_data  (dut_s_wr_data),  
 .s_addr     (dut_s_addr),  
 .s_bank_addr (dut_s_bank_addr),  
 .s_dout_valid (dut_s_dout_valid),  
 .s_sel_n    (dut_s_sel_n),  
 .s_dqm      (dut_s_dqm),  
 .s_we_n     (dut_s_we_n),  
 .s_sa       (dut_s_sa),  
 .s_scl      (dut_s_scl),  
 .s_rd_ready (dut_s_rd_ready),  
 .s_rd_start (dut_s_rd_start),  
 .s_rd_pop   (dut_s_rd_pop),  
 .s_rd_end   (dut_s_rd_end),  
 .s_rd_dqs_mask (dut_s_rd_dqs_mask),  
 .s_cas_latency (dut_s_cas_latency),  
 .s_read_pipe (dut_s_read_pipe),  
 .sm_addr    (dut_sm_addr),  
 .sm_oe_n    (dut_sm_oe_n),  
 .sm_we_n    (dut_sm_we_n),  
 .sm_bs_n    (dut_sm_bs_n),  
 .sm_dout_valid (dut_sm_dout_valid),  
 .sm_rp_n    (dut_sm_rp_n),  
 .sm_wp_n    (dut_sm_wp_n),  
 .sm_wr_data (dut_sm_wr_data),  
 .sm_adv_n   (dut_sm_adv_n),  
 .s_sda_out  (dut_s_sda_out),  
 .s_sda_oe_n (dut_s_sda_oe_n),  
 .gpo       (dut_gpo),  
 .debug_ad_bank_addr (dut_debug_ad_bank_addr),  
 .debug_ad_row_addr  (dut_debug_ad_row_addr),  
 .debug_ad_col_addr  (dut_debug_ad_col_addr),  
 .debug_ad_sf_bank_addr (dut_debug_ad_sf_bank_addr),  
 .debug_ad_sf_row_addr (dut_debug_ad_sf_row_addr),  
 .debug_ad_sf_col_addr (dut_debug_ad_sf_col_addr),  
 .debug_hiu_addr     (dut_debug_hiu_addr),  
 .debug_sm_burst_done (dut_debug_sm_burst_done),  
 .debug_sm_pop_n     (dut_debug_sm_pop_n),  
 .debug_sm_push_n    (dut_debug_sm_push_n),  
 .debug_smc_cs       (dut_debug_smc_cs),  
 .hclk               (dut_hclk),  
 .hresetn            (dut_hresetn),  
 .haddr              (dut_haddr),  
 .hsel_mem           (dut_hsel_mem),  
 .hsel_reg           (dut_hsel_reg),  
 .hwrite             (dut_hwrite),  
 .htrans             (dut_htrans),  
 .hsize              (dut_hsize),  
 .hburst             (dut_hburst),  
 .hready             (dut_hready),  
 .hwdata             (dut_hwdata),  
 .sm_clken           (dut_sm_clken),  
 .sm_ready           (dut_sm_ready),  
 .sm_data_width_set0 (dut_sm_data_width_set0),  
 .s_rd_data          (dut_s_rd_data),  
 .s_sda_in           (dut_s_sda_in),  
 .sm_rd_data         (dut_sm_rd_data),  
 .gpi                (dut_gpi),  
 .remap              (dut_remap),  
 .power_down         (dut_power_down),  
 .sm_power_down      (dut_sm_power_down),  
 .clear_sr_dp        (dut_clear_sr_dp),  
 .big_endian         (dut_big_endian));
```

# Visual Design

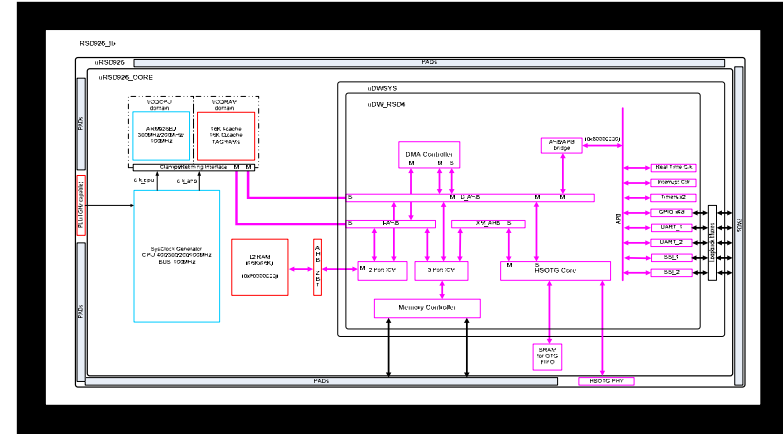
- **Drawings, diagrams are key to seeing patterns**
- **Must be provably equivalent to code**
- **Standard approach: generate code from drawings**
- **Result: very low adoption**
- **Why: Mixes content with presentation**

# Visual Design – A Better Path

- **Generate drawings from code**
  - **Possible with SystemVerilog – minimal clutter**
  - **Requires some discipline in coding style**
- **Reverse engineering (visualization) tools**
  - **Key to IP-based SoC design**
  - **Code view, Function View**
  - **Clock and reset View**

# Visual Environment

```
ahb_data i_ahb_data (  
  .hclk      (hclk),  
  .hresetn  (hresetn),  
  .ahb_m1   (ahb_m_cpu),  
  .ahb_m2   (ahb_m_otg),  
  .ahb_s1_s2 (ahb_memctl),  
  .ahb_s3   (ahb_apb),  
  .ahb_s4   (ahb_s_otg)  
);
```



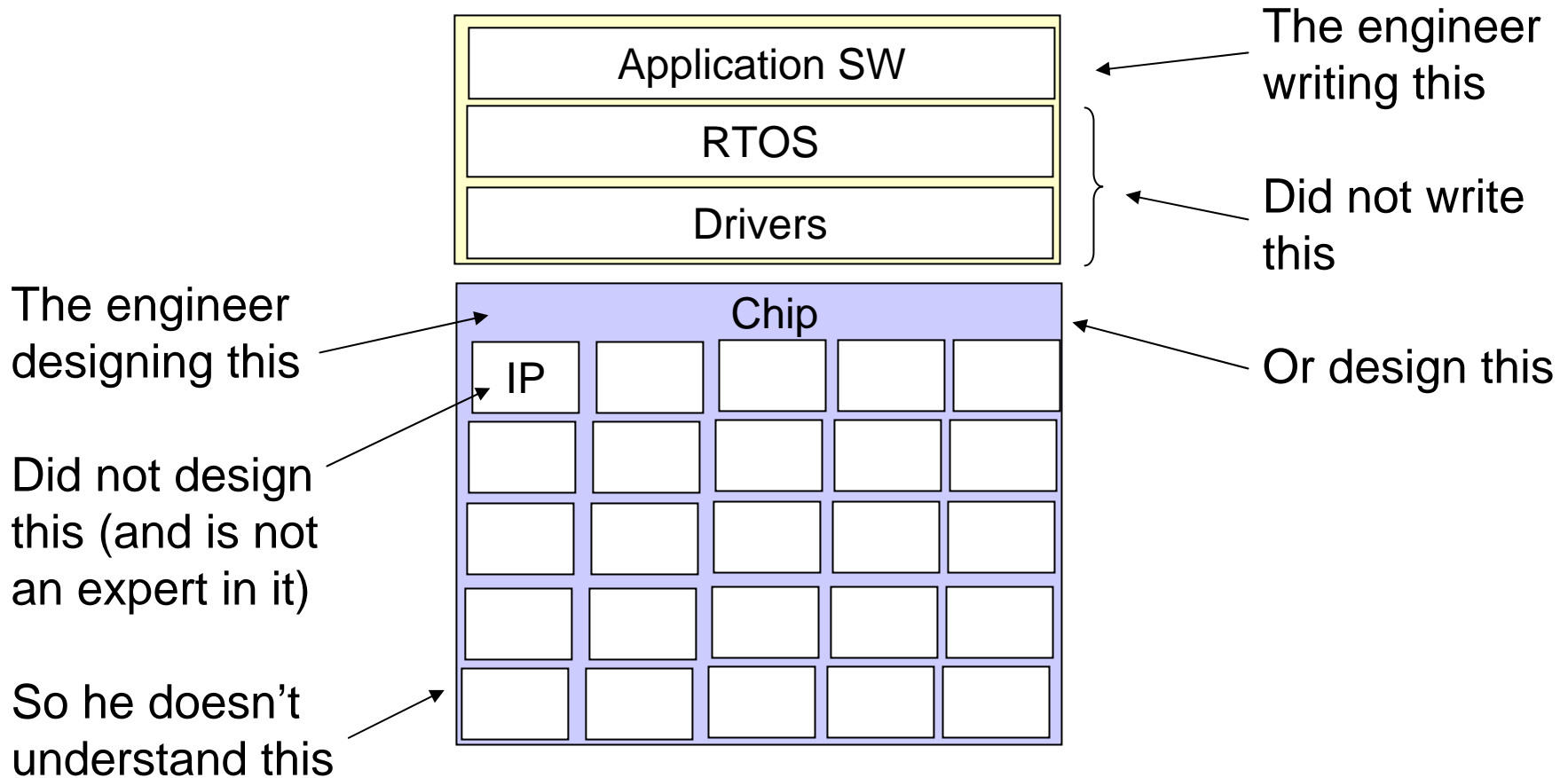
Two monitors, 23"  
11x17 color printer

# Design

## Summary

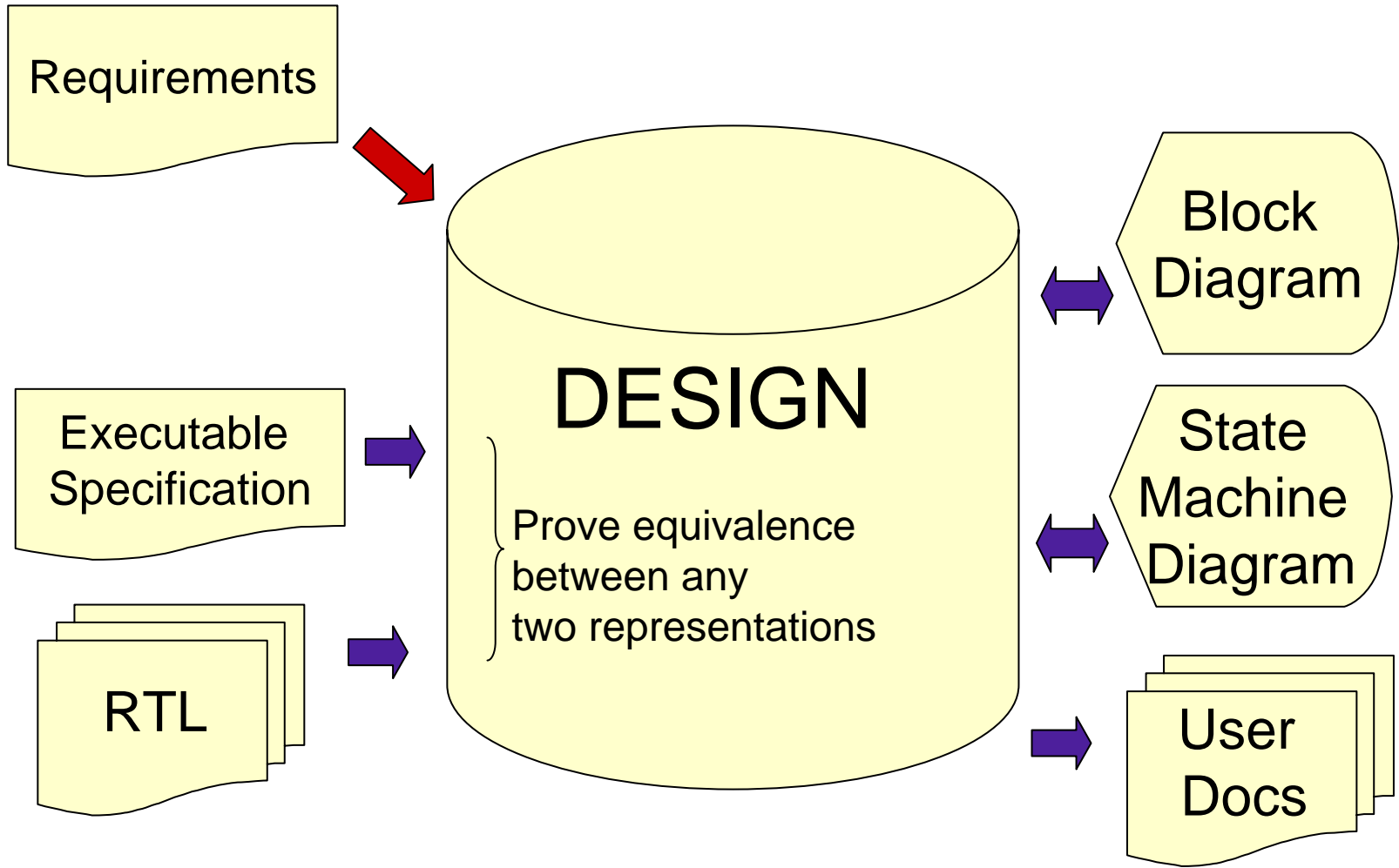


# Problems of IP-based Design



You're always debugging someone else's code

# Design as a Database



Ultimately, the quality of a design depends on the simplicity of it's execution

The art of design is the art of making the very complex appear very simple

Design

Thank You

