Tampereen yliopisto
Tampere University

COMP.CE.250
System-on-Chip Design

Clock and Power Domains

Arto Oinonen

Mobile SoC Power Consumption

Snapdragon X Elite (2024):

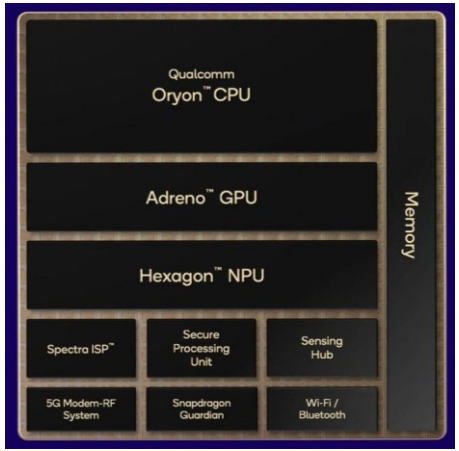
- 12 Qualcomm Oryon (ARMv8 ISA) cores
- Targeted to laptops
- Cores boost over 4GHz
- Thermal Design Power (TDP) **23W**
- Oryon cores are designed to scale from automotive systems to smartphones

A typical smartphone battery (5000mAh ≃ 20Wh):

- 23W -> 52 minutes
- 100W -> 12 minutes
- Expected battery life > 48h -> < **0,42W** on average

Also thermal considerations: the power (heat) has to be also moved somewhere

- Smaller devices: less space for thermal management



Qualcomm

PC Components > CPUs

Snapdragon X Elite pushed past 100W shows us what the CPU can offer on the desktop — almost 4X more power for 10% to 30% more performance

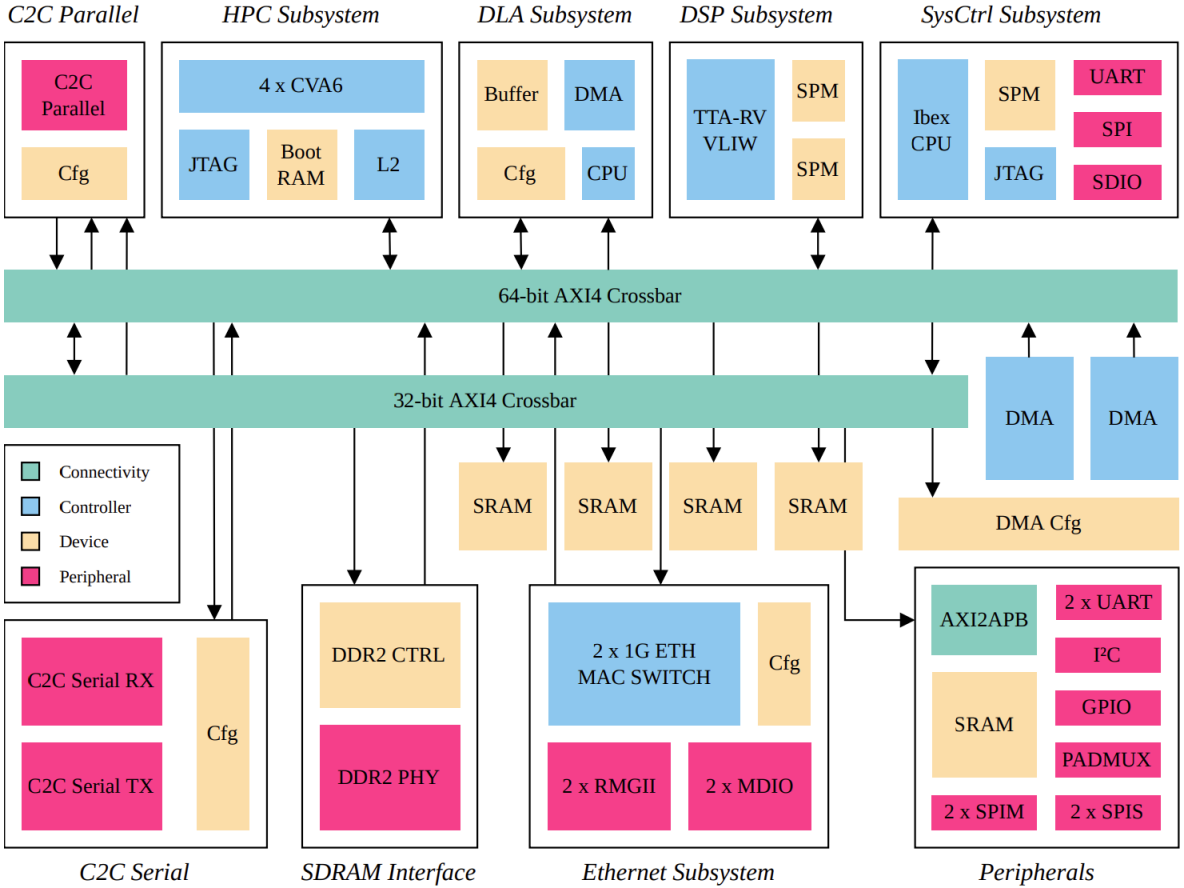
News By Matthew Connatser published October 1, 2024

Desktop users aren't missing out on much.

(Image credit: Qualcomm via Arrow.com)

https://www.tomshardware.com/pc-components/cpus/snapdragon-x-elite-pushed-past-100w-shows-us-what-the-cpu-can-offer-on-the-desktop-almost-4x-more-power-for-10-to-30-more-performance

Headsail SoC



- Estimated **1,5W total peak power** for full system workloads
- Highest clock frequencies
  - **3GHz:** SerDes
  - **1GHz:** HPC, DSP
- Reference clock input: **30MHz**

M. Käyrä *et al.*, "Headsail: One-Year Tape-Out of a 25-mm2 Linux-Capable RISC-V MPSoC," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, doi: 10.1109/TVLSI.2026.3653630.

Outline

- CMOS power consumption
- Active Power Management

- Logic synthesis recap: Clocking and synchronization
- Clock Domains

- Power Domains
- UPF Basics
- Some architectural pondering

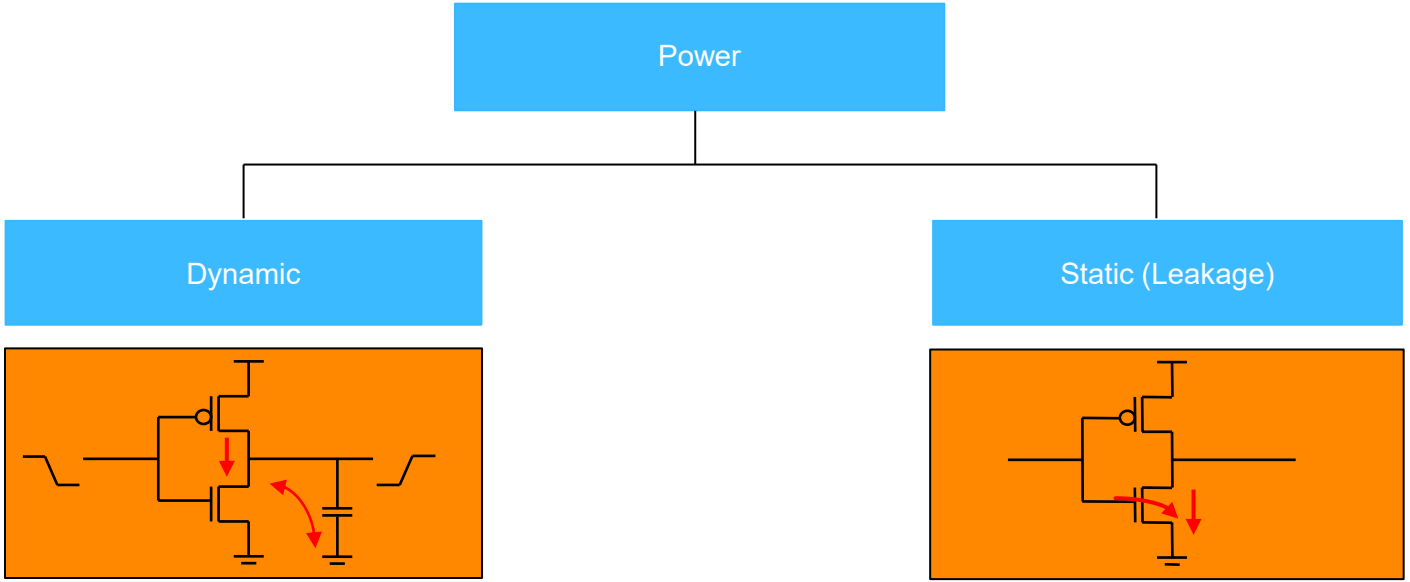


Figure: Mohsin Abbas (COMP.CE.510 lecture 12)

Tampereen yliopisto
Tampere University

CMOS Power Consumption

Power Consumption Sources

$$P_{total} = P_{sw} + P_{sc} + P_{static}$$

P_{sw} –Switching power. Switching leads to charging and discharging the load capacitances

P_{sc} – Short circuit (*crowbar*) power is due to the direct-path short current, which arises when both NFET and PFET transistors are simultaneously active, conducting current directly from supply to ground

P_{static} – is due to leakage current, which arises from reverse bias diode currents, sub-threshold fleet, gate tunneling

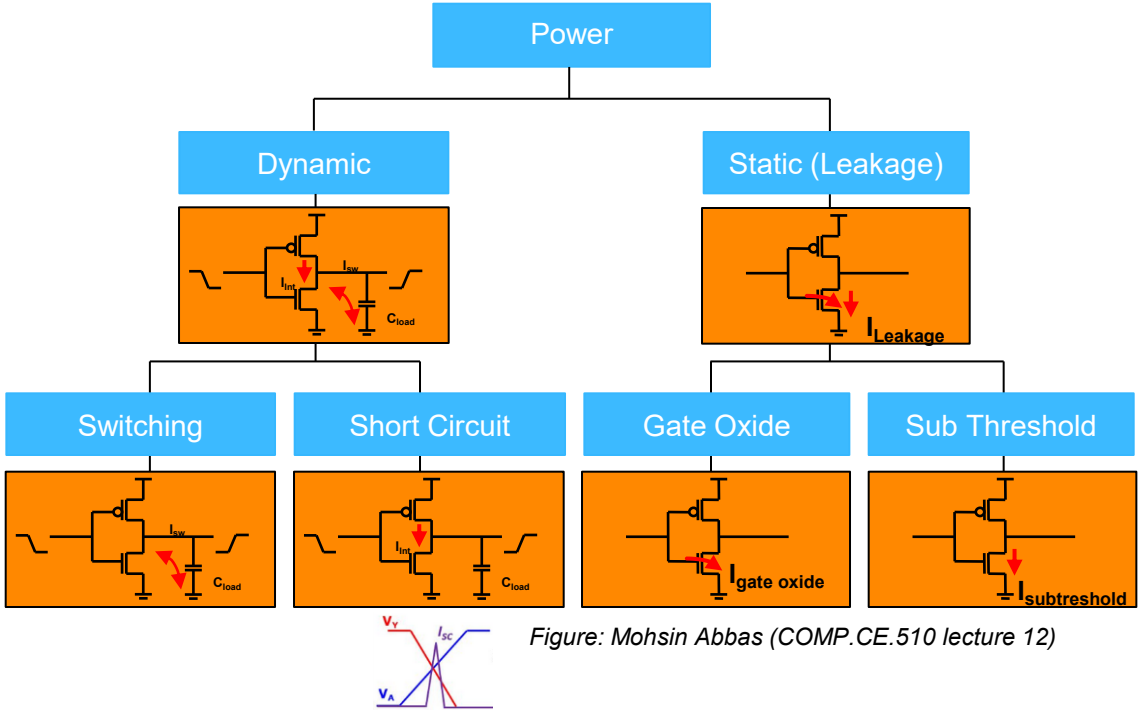


Figure: Mohsin Abbas (COMP.CE.510 lecture 12)

https://resources.pcb.cadence.com/blog/2023-cmos-power-calculation
https://schaumont.dyn.wpi.edu/ece574f24/09power.html

Power Consumption Sources

$$P_{total} = P_{sw} + P_{sc} + P_{static}$$

$$P_{sw} = f_{sw} C_L V^2 = \alpha f_{ck} C_L V^2$$

α = activity factor

$$P_{sc} = T_{sc} I_{peak} V = \alpha f_{ck} E_{sc}$$

E_{sc} = short circuit energy

T_{sc} = short circuit time period

$$P_{static} = V_{cc} I_{cc}$$

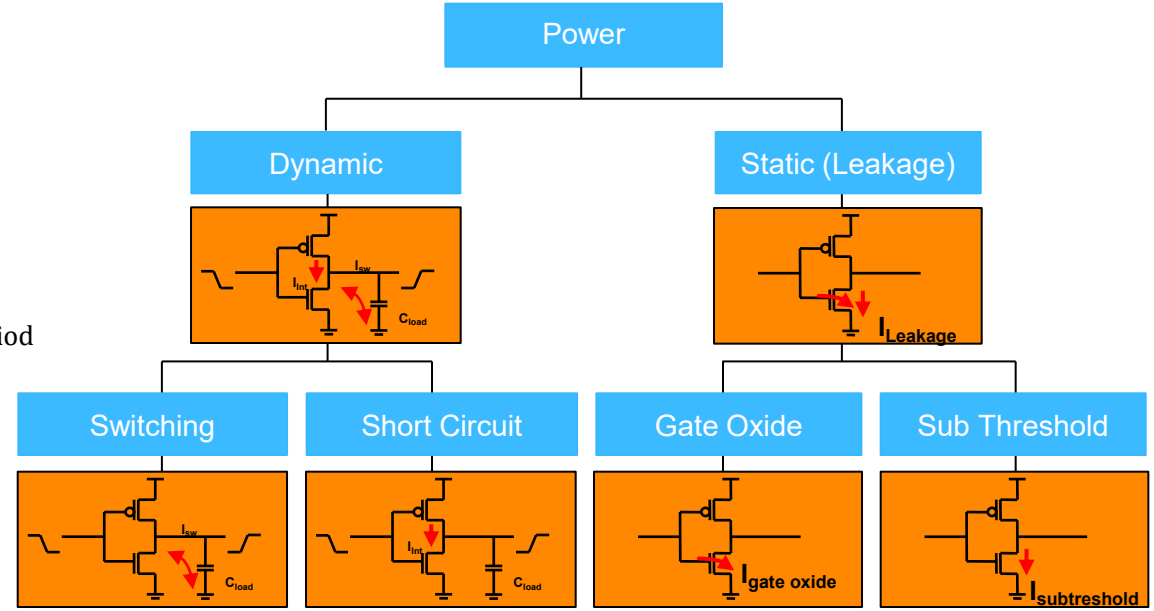


Figure: Mohsin Abbas (COMP.CE.510 lecture 12)

https://resources.pcb.cadence.com/blog/2023-cmos-power-calculation

Power Consumption Sources

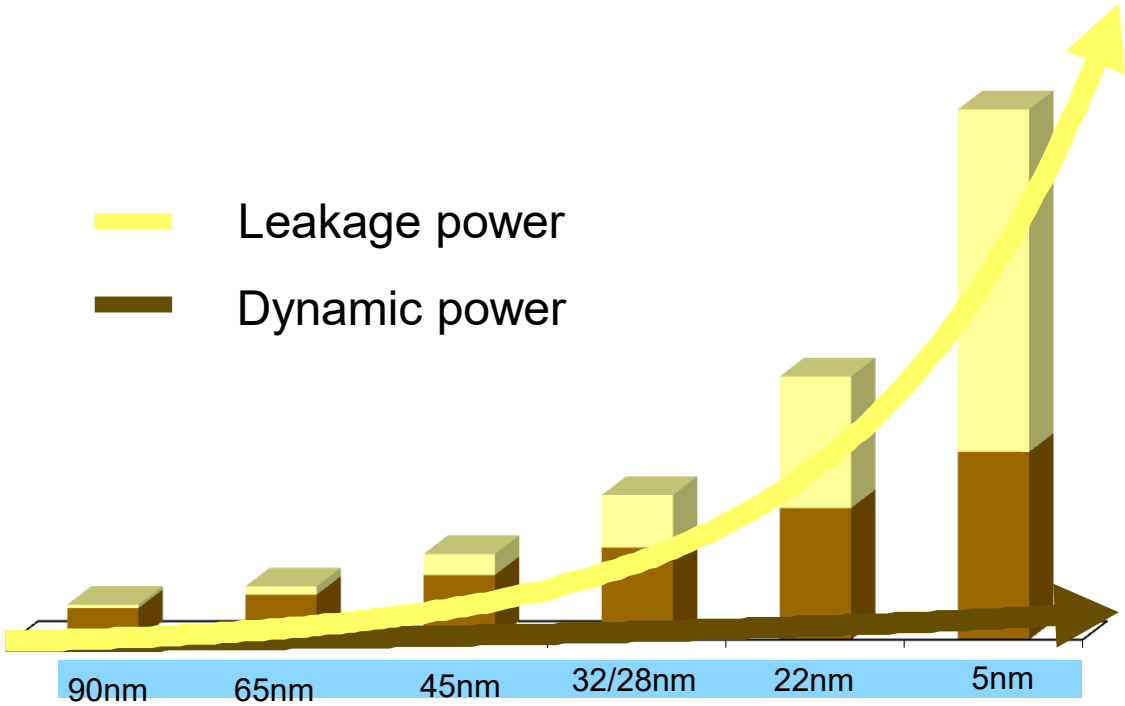


Figure: Mohsin Abbas (COMP.CE.510 lecture 12)

Active Power Management

$$P_{total} = P_{sw} + P_{sc} + P_{static}$$

$$P_{sw} = f_{sw} C_L V_{cc}^{\ 2} = \alpha f_{ck} C_L V_{cc}^{\ 2}$$

α = activity factor

$$P_{sc} = T_{sc} I_{peak} V_{cc} = \alpha f_{ck} E_{sc}$$

E_{sc} = short circuit energy
T_{sc} = short circuit time period

$$P_{static} = V_{cc} I_{cc}$$

Clock Gating

$f_{sw} = 0$

Turn off the clock to avoid unnecessary switching activity

Power Gating

$V_{cc}, I_{cc} = 0$

Turn off the power when logic is not in use to avoid static leakage

Multi-Voltage Design

$V_{cc} \rightarrow 0$

Portions of the design operate at different voltages based on performance needs

(Dynamic Voltage and Frequency Scaling)

DVFS

$V_{cc}\ f_{sw} \rightarrow 0$

Adjust voltage and frequency dynamically based on performance needs

Clock Gating

- Even if the IP is not actively doing anything, the clock keeps switching
- ➢ Disabling the clock for inactive blocks will avoid unneccessary switching activity

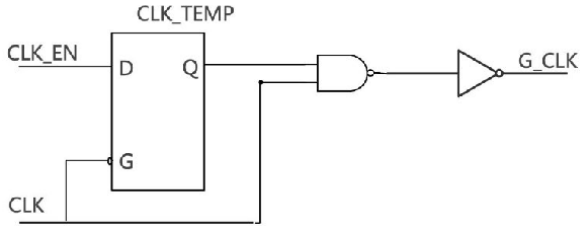
Clock Gating

$f_{sw} = 0$

Turn off the clock to avoid unnecessary switching activity

Some penalties

- The area of the cg components
- Possible clock glitches (use dedicated cg cells (ICG))
- Clock skew



Clock Gating circuit
An Integrated Clock Gating (ICG) cell integrates the latch and AND gate together

Wang,S. (2025). Research on Low-Power Design Methods for Integrated Circuits. Applied and Computational Engineering,128,18-23.
https://teamvlsi.com/2021/08/integrated-clock-gating-icg-cell-in-

Power Gating

- Leakage power is a real issue on newer nodes
- Power gating is becoming very effective
  - Only power on the parts that are currently active
  - ➢ Reduce leakage to save battery
  - ➢ Relax the thermal budget to allow the active parts to generate more heat

Power Gating

$V_{cc}, I_{cc} \rightarrow 0$

Turn off the power when logic is not in use to avoid static leakage

Some penalties

- Voltage drop (IR drop) caused by the gating structures
- Area overhead
- Rush current when powered back on
- Ramp-up latency: waking up is not immediate
- Additional structures required for state retention and isolation
  - (more about this later)

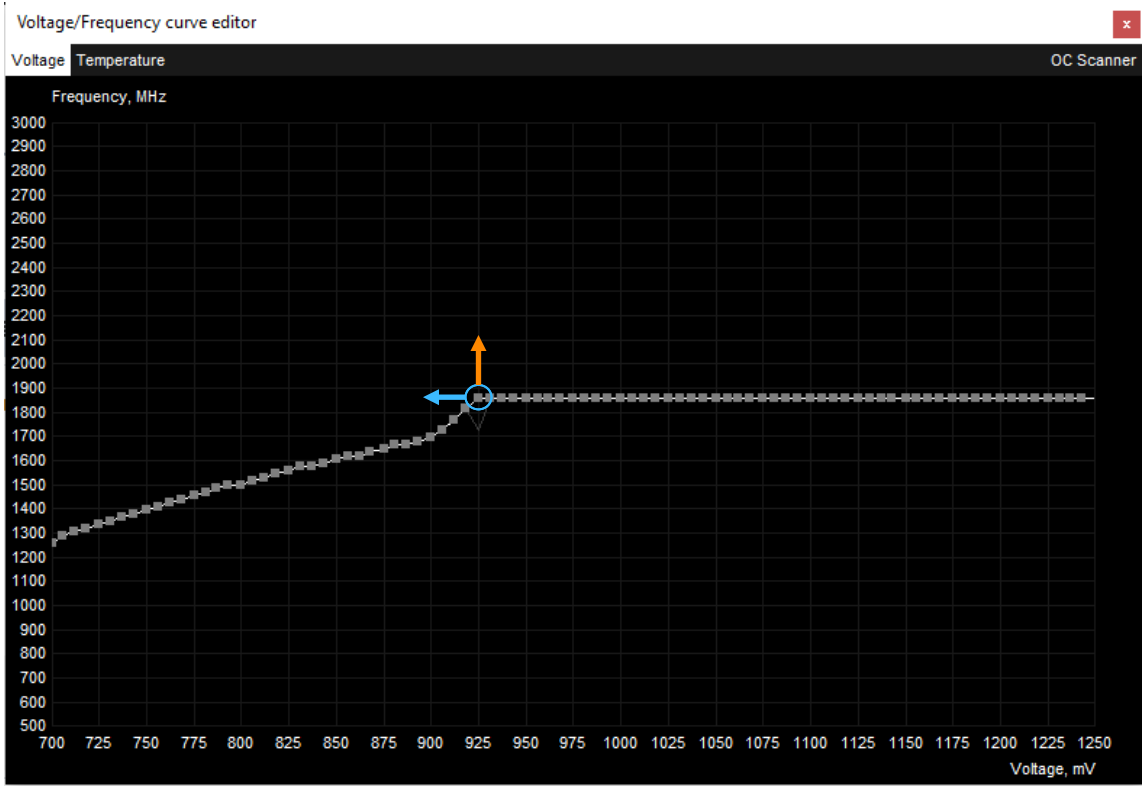
Dennard scaling (1971) *stated that as transistors shrink, their power density stays constant because voltage and current scale down with size. Broke down in ~2006*

Dark silicon *is the amount of circuitry of an integrated circuit that cannot be powered-on at the nominal operating voltage for a given thermal design power (TDP) constraint*

Jawad Haj-Yahya et al., "DarkGates: A Hybrid Power-Gating Architecture to Mitigate the Performance Impact of Dark-Silicon in High Performance Processors," in HPCA 2022
https://people.inf.ethz.ch/omutlu/pub/DarkGates_hpca22.pdf

DVFS: GPU Voltage/Frequency Curve

- An example of a user-adjustable DVFS
- A single clock/voltage curve
  - Modern CPUs have a similar idea but curves can be core-specific
- *”Undervolting”*/*”Overclocking”* the GPU: modifying the curve outside the manufacturer’s specification to reach
  - the same frequency with less voltage, or
  - faster frequencies with the same voltage, and
  - severely reduced MTBF



Linus Tech Tips

ACPI Package C-states: Intel Skylake

- ACPI C-state defines an idle power state of the system
- Recommendations defined in Advanced Configuration and Power Interface (ACPI) standard

- In general, deeper sleep states allow more aggressive power management, but lead to longer enter/exit latencies

Table 1: Package C-states in the Intel Skylake mobile SoC.

Package C-state	Major conditions to enter the package C-state
C0	One or more cores or graphics engine **executing instructions**
C2	All cores in **CC3** (clocks off) **or deeper** and graphics engine in **RC6** (power-gated). DRAM is **active.**
C3	All cores in **CC3 or deeper** and graphics engine in **RC6**. Last-Level-Cache (LLC) may be flushed and turned off, DRAM in **self-refresh**, most IO and memory domain clocks are gated, some IPs and IOs can be **active** (e.g., **DC and Display IO**).
C6	All cores in **CC6** (power-gated) or deeper and graphics engine in **RC6**. LLC may be flushed and turned off, DRAM in **self-refresh**, IO and memory domain clocks generators are turned off. Some IPs and IOs can be active **(e.g., video decoder (VD) and display controller (DC))**.
C7	Same as Package C6 while some of the IO and memory domain voltages are **power-gated**. **CPU core VR is ON**.
C8	Same as Package C7 with additional **power-gating** in the IO and memory domains. **CPU core VR is OFF**.
C9	Same as Package C8 while all IPs must be off. Most voltage regulators' voltages are reduced. **The display panel can be in panel self-refresh (PSR)** [65, 66].
C10	**Same as Package C9** while **all SoC VRs** (except state always-on VR) **are off**. **The display panel is off**.

Jawad Haj-Yahya et al., "DarkGates: A Hybrid Power-Gating Architecture to Mitigate the Performance Impact of Dark-Silicon in High Performance Processors," in HPCA 2022
https://people.inf.ethz.ch/omutlu/pub/DarkGates_hpca22.pdf

Lecture 12: Clock and Synchronization 1

Log
Tamp

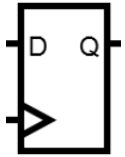
Lecture 13: Clock and Synchronization 2

COMP.CE.240 Logic Synthesis
Tampere University

Tampereen yliopisto
Tampere University

Recap: Clocking and Synchronization

DFF Timing Specification

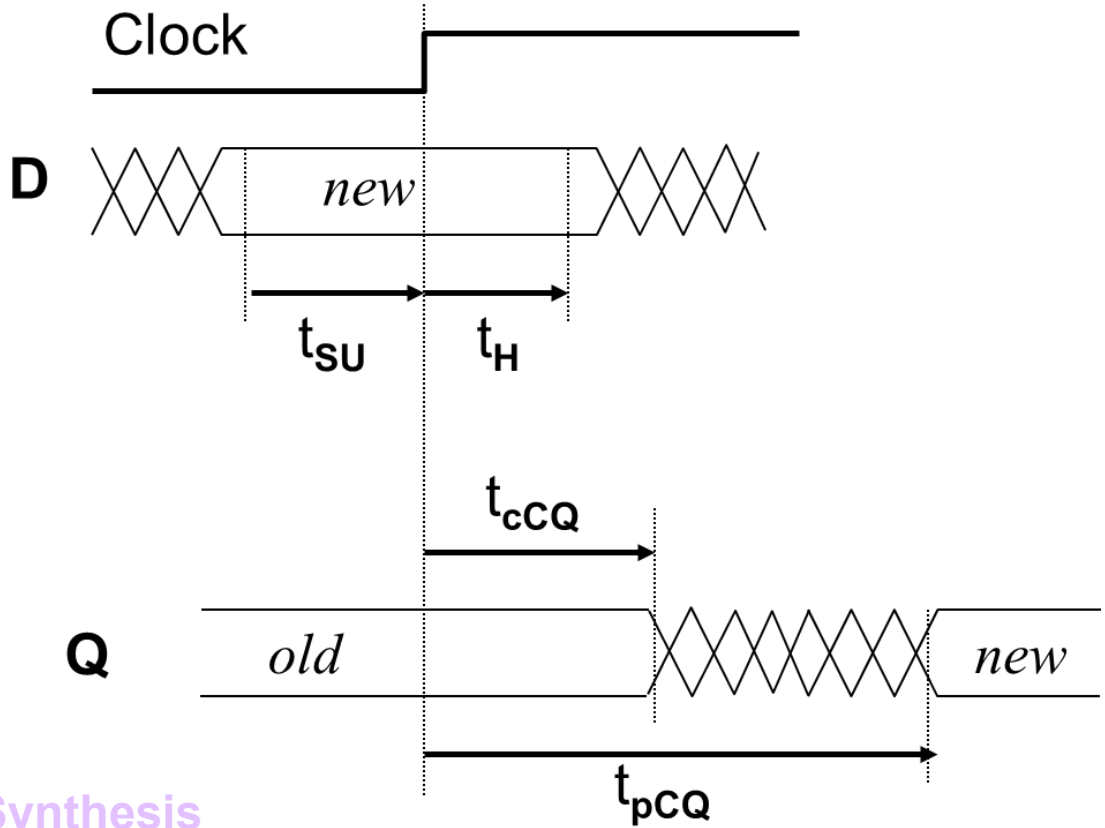


Input must be stable:

- Before edge – setup time
- After edge – hold time

Propagation delay is the time **from clock edge** to the change in output

IF you provide the FF with well-behaving inputs, THEN it will behave according to this specification, ELSE no guarantees…



Critical Path

$t_{clk\_period} \geq t_{crit.path}$

$t_{crit.\ path} = t_{p,dff} + t_{comb} + t_{su,\ dff}$

Flip-flops are the start and end point of critical path

- All flip-flops within one *clock domain* have the same clock signal (same frequency)

Ensure that $t_{clk\_priod} \geq t_{crit.path}$

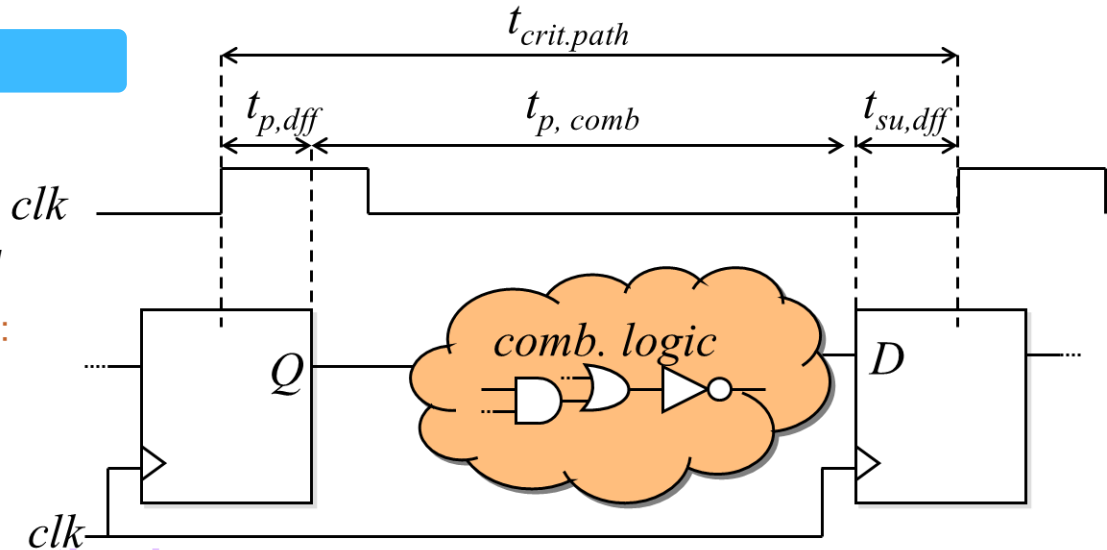
- Analyze all paths Q→D and use the longest path delay to calculate the frequency

Critical path

- Starts from DFF's Q output
- Passes through combinatorial logic
- Ends to DFF's D input
- Does not ever go through a DFF
- Paths starting or ending at IO pin may be *unconstrained*

Timing:

Structure:



Clock Distribution Network

Ideal clock: clock's rising edge arrives at FFs at the same time

Real implementation:

- Driving capability of each cell is limited
- Need a network of buffers to drive all FFs (more effort and power)
- Must balance the length of clock signal wire
  - FPGA: pre-fabricated clock distribution network. Easy to use
  - ASIC: Clock tree. Implementation needs attention

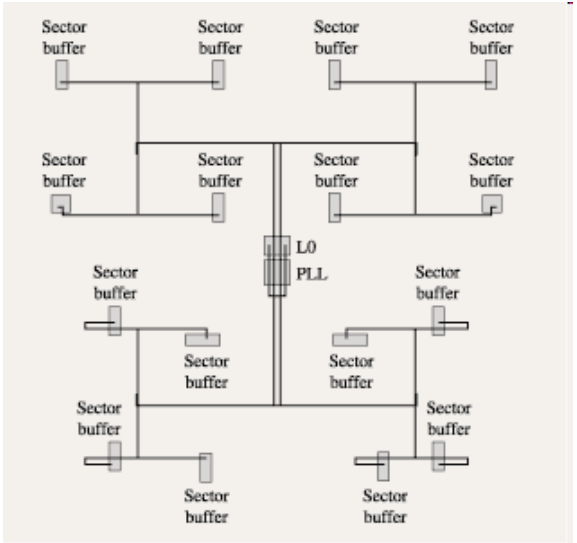
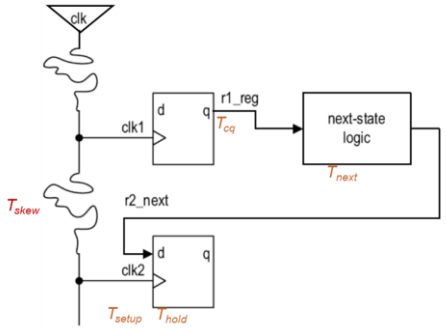
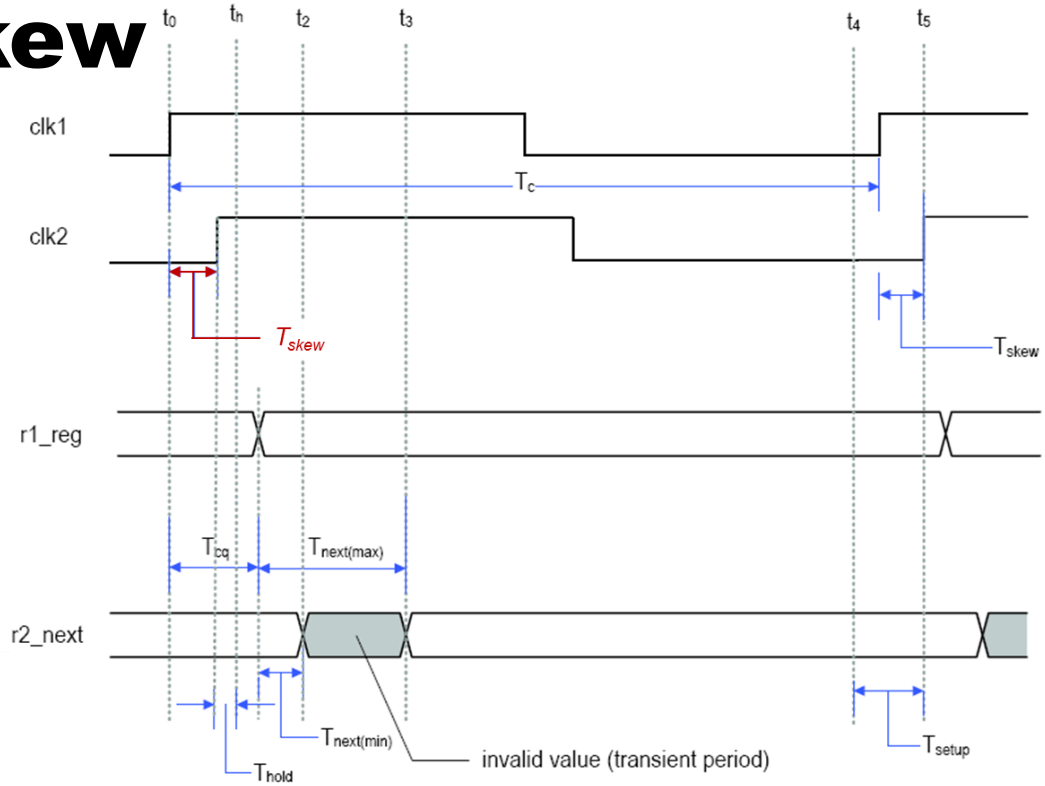


Figure 8

PLL clock distribution via H-trees to the 16 L1 sector buffers.

Clock Skew





T_{skew} = clock skew
T_{cq} = DFF clock-to-Q
T_{next} = comb. delay
T_{hold} = DFF hold time
T_{setup} = DFF setup time

Requirements:

$$T_{cq} + T_{next(min)} > T_{hold} + T_{skew}$$

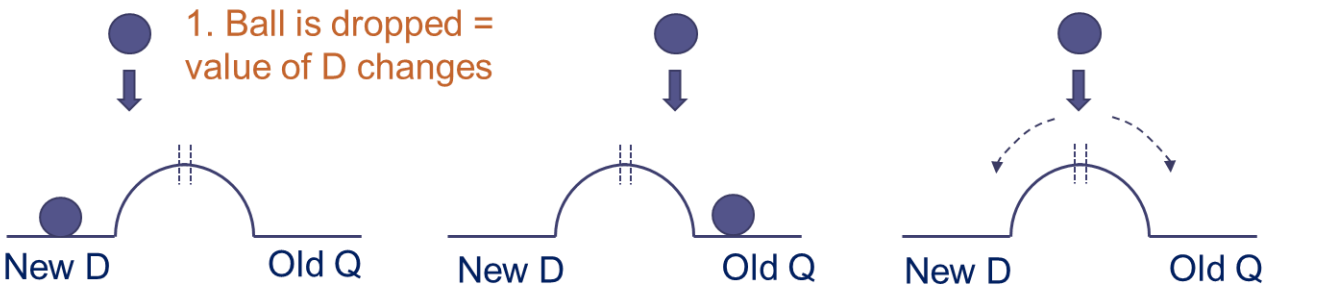
$$T_{cq} + T_{next(max)} < T_c - T_{setup} + T_{skew}$$

In general, we must assume $\pm T_{skew}$

Larger skew could violate the hold time of dff2

DFF Metastability

- Outcome is obvious if D does not change near clock edge
- Outcome is obvious if ball is not dropped accurately on the middle of the hill top



a) New value arrives *before* clk edge and gets stored into DFF

b) New value arrives *after* clk edge and Q does not change

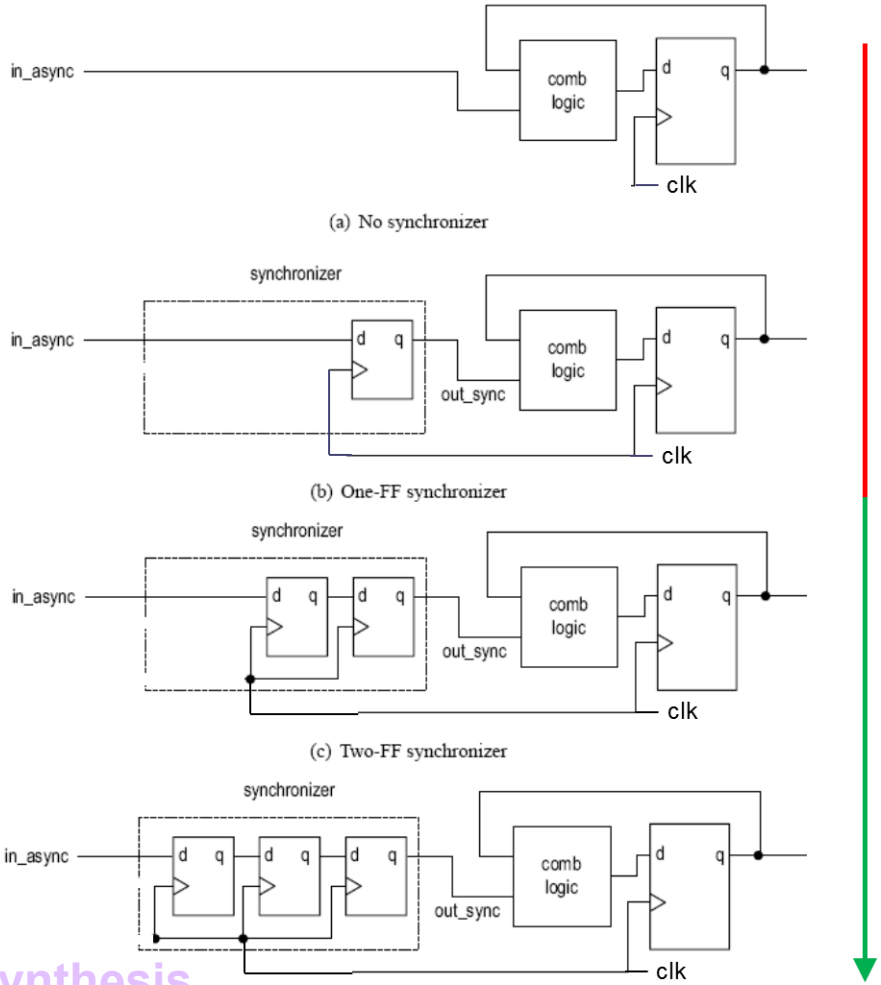
c) New value violates t_{su} or t_{hold}. Ball balances on top of the hill and goes randomly to left or right. Output is first metastable and violates t_{cq}

Synchronizers

- Synchronizing an asynchronous data input with system clock

No physical circuit can prevent metastability

- We cannot avoid it, we have to live with it
- Design should be ”metastability-tolerant”, since it cannot be ”metastability-free”
- Synchronizer just provides enough time for the metastable condition to be ”resolved”



6 Cases of Synchronizer Timing

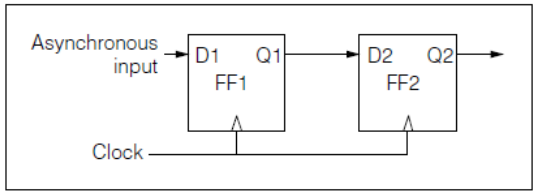


Figure 8. Two-flip-flop synchronization circuit.

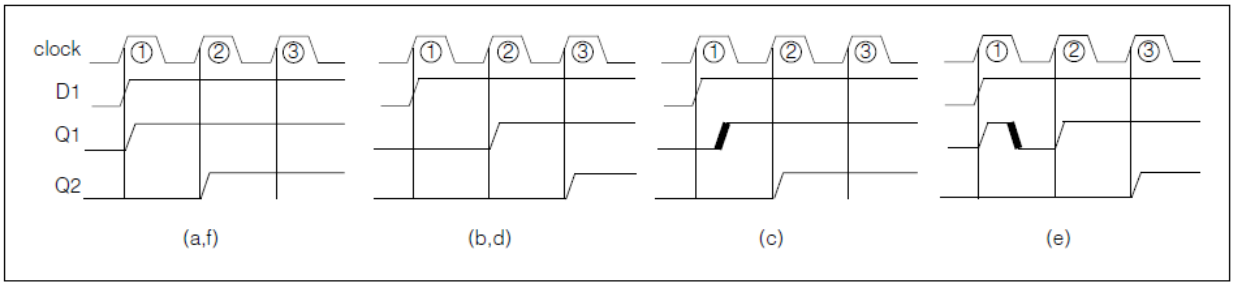


Figure 9. Alternative two-flip-flop synchronization waveforms.

Case a = Q1 goes 1 as wished
Case f = Q1 goes metastable, goes randomly to 1 and resolves to 1; looks like case a

b = Q1 misses 1 at first but rises at cycle 2
d = Q1 goes metastable, stays 0 and resolves to 0; looks like case b

c = Q1 metastable but resolves to 1 within one cycle

e = Q1 metastable and goes first 1 but then resolves to 0

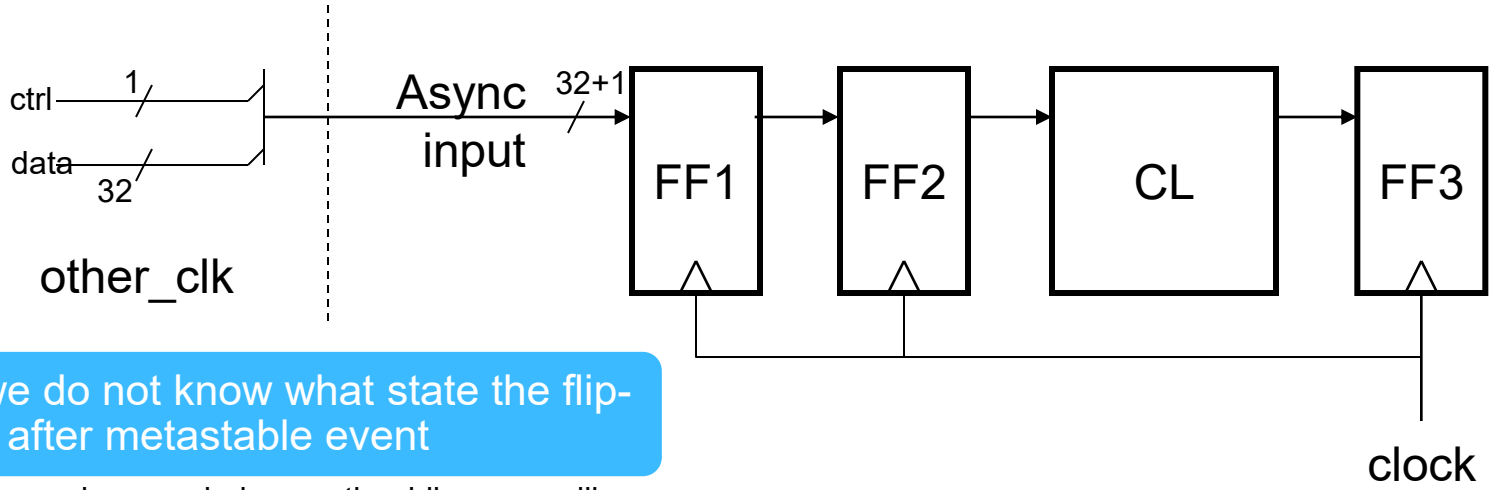
Hence, Q2 rises in all cases, but the cycle is either on 2 or 3. Note that D1 stays stable long enough

[R. Ginosar, Metastability and Synchronizers : A Tutorial, IEEE D&T Comp, Sep/Oct 2011]

Beware of WRONG Two-FF Synchronizers!

Asynchronous input contains both data and control

This does NOT work…



Recall: we do not know what state the flip-flop gets after metastable event

- some bits may be sampled correctly while some will go metastable
  - ➢ The data may get corrupted

Observations

Performance, clock cycles per transfer:

- 0-1 cc for Rx synchronizer metastability resolution
- 3 cc for Rx to issue ack (2 for synch, 1 for putting ack to outreg)
- 0-1 cc for Tx synchronizer metastability resolution
- 2 cc for Tx to synchronize ack
- ➔ 5-7 clock cycles per transfer (compare to 1 in totally synchronous)
- **Domain crossing is slow!**

Other methods for data transfer

- FIFO (synchronization needed for empty and full status signal)
  - May perform quite well for large data chunks
- Shared memory (synchronization needed for arbitration circuit)
- Dual-port memory (meta-stable condition may occur in the internal arbitration circuit)

These also need the synchronization! Domain crossing is still tricky and slow!

Observations (2): Don’t Over-Optimize

It is not beneficial to fine-tune all frequencies

E.g., increasing tx clock 475 → 500 MHz

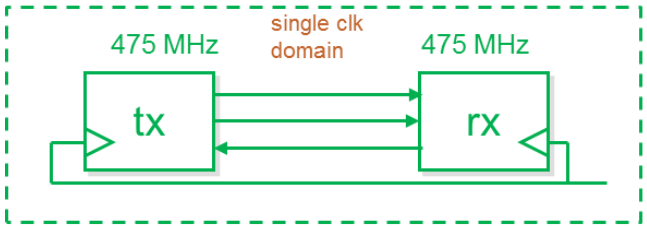
- Needs synchronizers
- ~5% increase in Tx processing power
- ~5x decrease in transfer rate Tx → Rx

The overall performance might be better with lower frequency

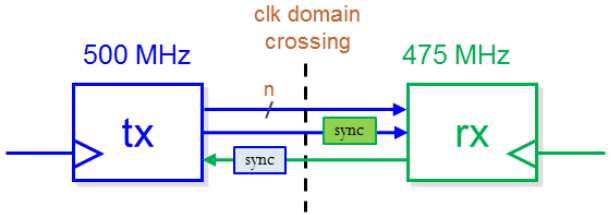
- Same phenomenon happens with supply voltages and voltage converters/regulators

One must analyze when multiple clocks pays off

- Depends on frequencies (e.g., 400 vs. 500 MHz)
- Depends on computation vs. communication ratio (e.g., ops/sent_byte)



Case1. Base system with 1 clock



Case2. Naively ”optimized” system with 2 clocks might actually be slower

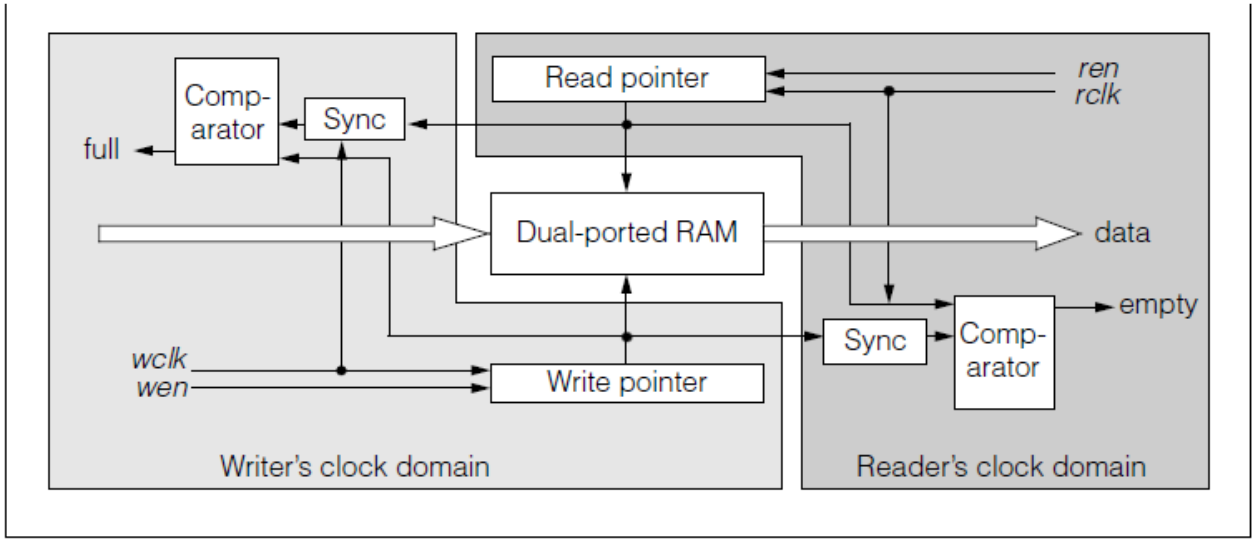
Gray FIFO

Each write increments a write pointer (read increments read pointer)

Pointers use **gray code**: only a single bit changes at a time

- Hence, the whole pointer can be synchronized although it has multiple bits

Synchronization adds latency in some cases, but will not corrupt data



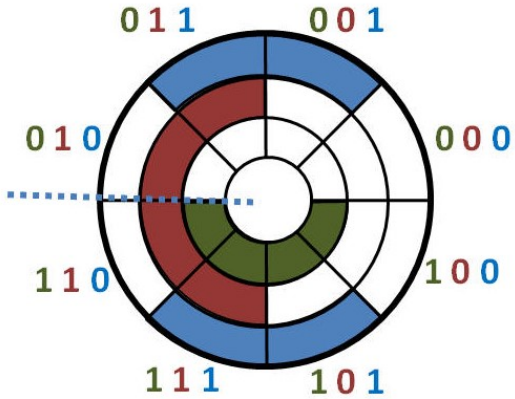
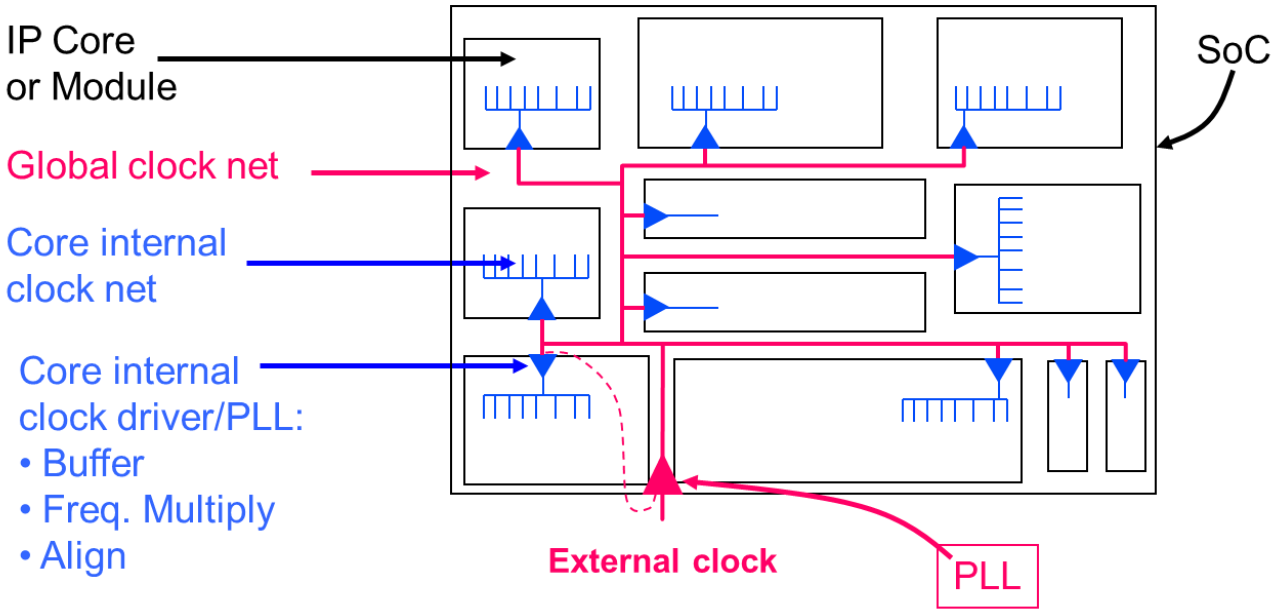


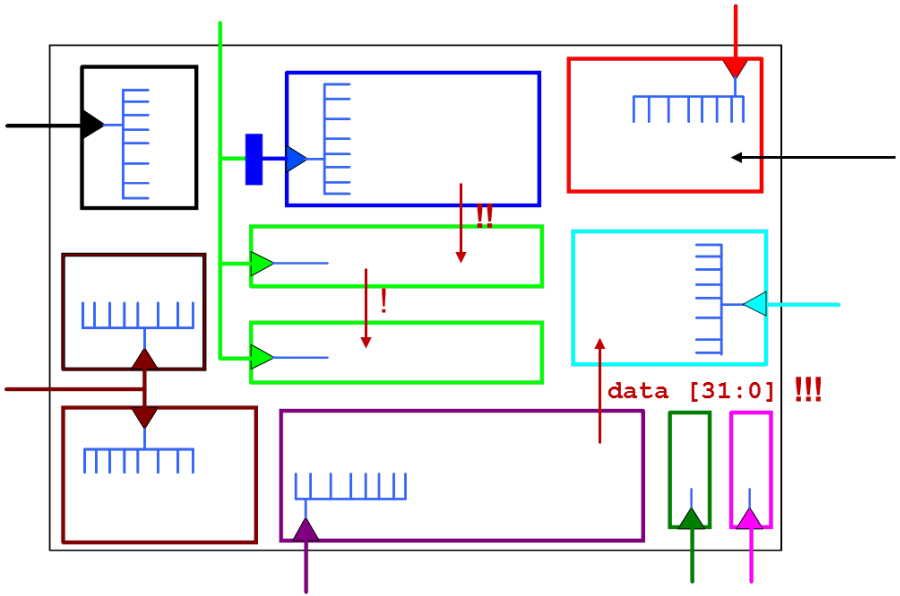
Figure 12. Two-clock FIFO synchronizer. It contains two separate clock domains and synchronizes pointers rather than data.

Synchronous (Single Clock Domain) SoC



Global Clock should arrive simultaneously to all modules! This guarantees that data can be safely communicated from one IP to another.
Clock tree balancing and buffering is not trivial (but doable in circuits so far).

SoC with Multiple Clock Domains



Communication **between domains** (e.g. data[31:0]) needs special attention.
Within one domain there is no problem.

Why Multiple Clocks

Inherent multiple clock sources

- E.g., external communication links require their own frequencies

Circuit size

- Clock skew increases with the number of FFs in a system

Design complexity

- E.g., a system with a 1 GHz application CPU, 3 GHz serial interface, 200 MHz accelerator IP, 30MHz boot CPU
- No need to optimize them all to run at multi-GHz (simpler + cheaper, possible)
- No need to run everything at 30 MHz (better performance)

Power consideration

- Dynamic power proportional to switching frequency
- Use lowest frequency allowed for each IP
- Especially useful when combined with lowered voltage!

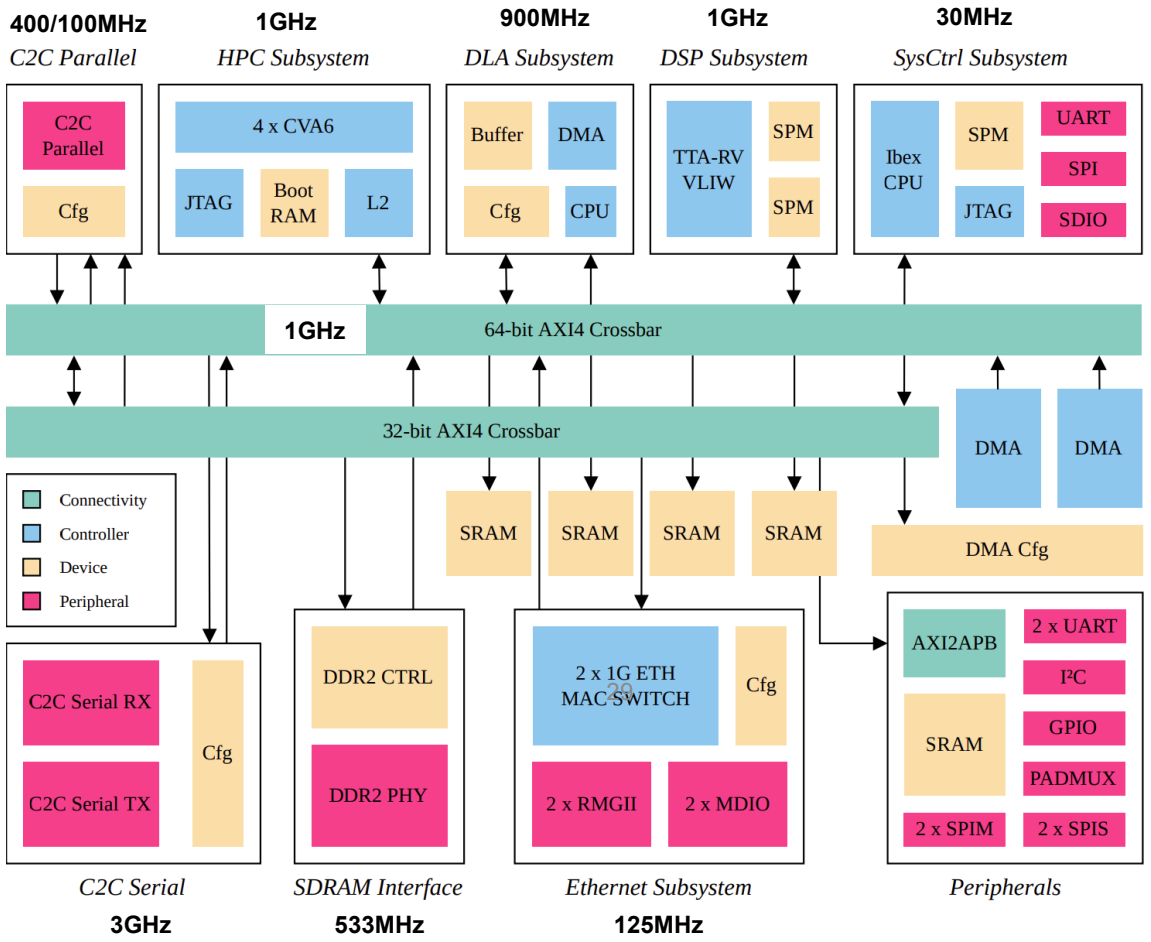
GALS

***Globally Asynchronous, Locally Synchronous* system**

- Partition a system into multiple independent subsystems with different clock domains
- Design and verify subsystem in same clock domain as a synchronous system
- Clock Domain Crossing (CDC) between subsystems

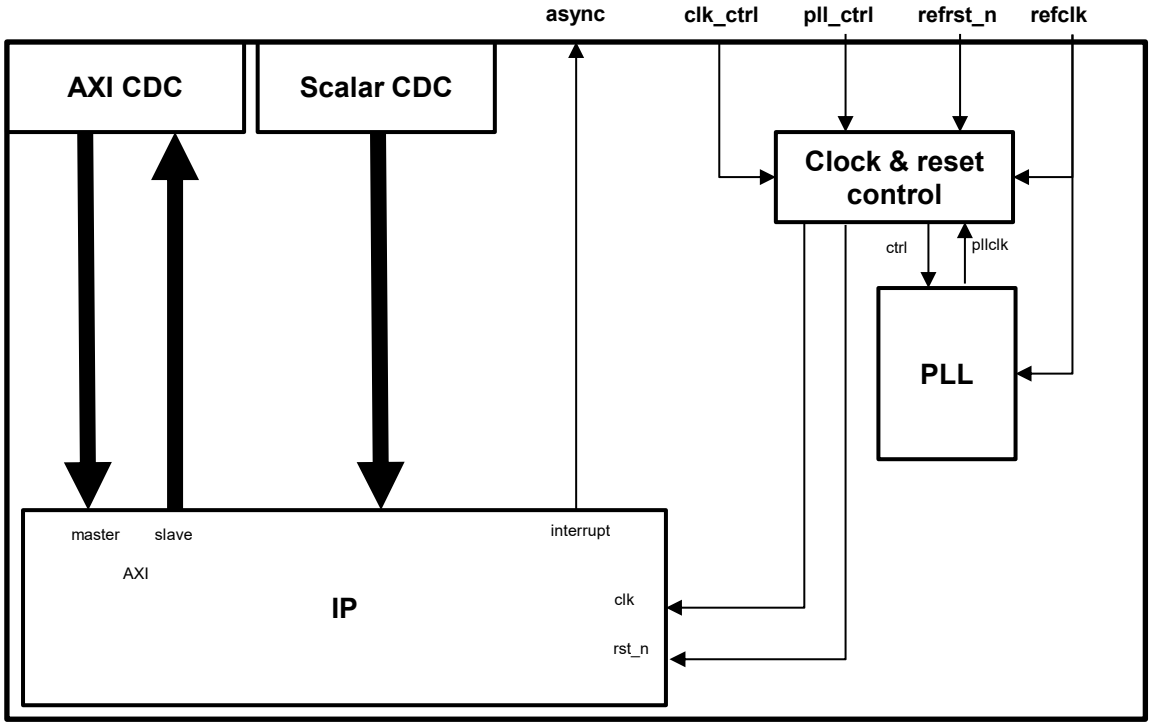
➢ **Relaxes the generation of global clock tree**

- **Headsail: A generic wrapper template for subsystems**
  - **PLL**
  - **Clock multiplexer**
  - **CDC components**
  - **Reset**



GALS Subsystem Wrapper

- Adapted from the SoC Hub wrapper
- AXI and scalar I/O pass through CDC blocks
- PLL for each subsystem, but bypassing the PLL and running on reference clock is also possible
- Clock & reset control includes
  - Clock multiplexer
  - Control registers:
    - Reset
    - Clock gating control
    - Clock mux control
    - PLL control
- On SoC level, could be controlled by
  - SysCtrl CPU (SoC Hub)
  - Control state machine
  - DIP switches on the PCB?



PULP AXI CDC

- A clock domain crossing on an AXI interface
- Included in the PULP AXI repository
- Instantiates a CDC Gray FIFO for each of the five AXI channels
  - push and pop ports in each FIFO are in different clock domains
  - A generic CDC Gray FIFO implementation is reused from the PULP Common Cells repository

- https://github.com/pulp-platform/axi/blob/master/src/axi_cdc.sv
- https://github.com/pulp-platform/common_cells/blob/master/src/cdc_fifo_gray.sv

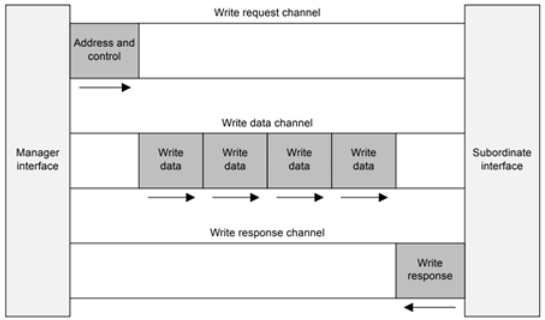


Figure A1.1: Channel architecture of writes

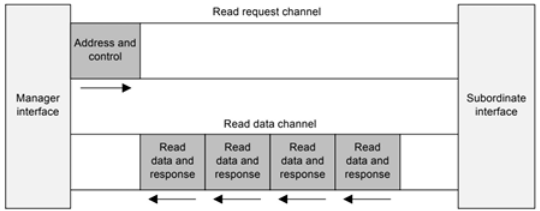
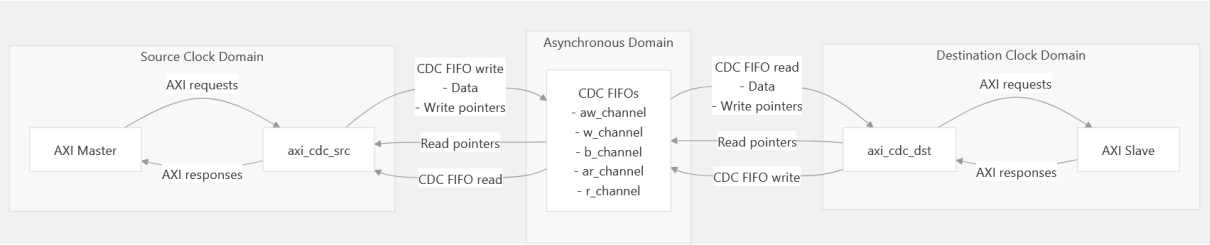


Figure A1.2: Channel architecture of reads



A rendition of the architecture by DeepWiki (AI-generated from SystemVerilog sources)
https://deepwiki.com/pulp-platform/axi/8.1-clock-domain-crossing

$$P_{total} = P_{sw} + P_{sc} + P_{static}$$

$$P_{sw} = f_{sw} C_L V^2 = \alpha f_{ck} C_L V^2$$

$$P_{sc} = T_{sc} I_{peak} V = \alpha f_{ck} E_{sc}$$

$$P_{static} = V_{cc} I_{cc}$$

Clock Gating

$f_{sw} = 0$

Turn off the clock to avoid unnecessary switching activity

Power Gating

$V_{cc}, I_{cc} = 0$

Shut off the power when logic is not in use to avoid static leakage

Multi-Voltage Design

$V_{cc} \rightarrow 0$

Portions of the design operate at different voltages based on performance needs

(Dynamic Voltage and Frequency Scaling)

DVFS

$V_{cc}, f_{sw} \rightarrow 0$

Adjust voltage and frequency dynamically based on performance needs

Back to Power Management

Active Power Management

- The RTL code does not know about the frequencies, powers or voltages

- PLL can be reconfigured by software via register interface
- Clock is constrained for Static Timing Analysis (STA) and Synthesis

➢ *Where are the power domains defined?*

➢ *How to verify a system with power management?*

Clock Gating

$f_{sw} = 0$

Turn off the clock to avoid unnecessary switching activity

Power Gating

$V_{cc}, I_{cc} = 0$

Shut off the power when logic is not in use to avoid static leakage

Multi-Voltage Design

$V_{cc} \rightarrow 0$

Portions of the design operate at different voltages based on performance needs

(Dynamic Voltage and Frequency Scaling)

DVFS

$V_{cc}, f_{sw} \rightarrow 0$

Adjust voltage and frequency dynamically based on performance needs

Unified Power Format (UPF), IEEE 1801

- Standard for specifying power intent
- Driving both verification and implementation, from RTL to layout
- Extension of tool command language (Tcl)
  - The scripting language that is already used in the majority of EDA tools

- UPF elements include
  - Power supplies: supply nets, supply sets, power states
  - Power control: power switches
  - Additional protection: level shifters and isolation
  - Memory retention during times of limited power: retention strategies and supply set power states

- Allows systems to be designed with power as a key consideration early in the process
- Defined separately from the HDL. This enables that the IP design can be reused in different systems with different power architectures.

Ashok B.Mehta: ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies, Springer International Publishing AG, 2017

UPF Methodology

- RTL is augmented with UPF
  - Define the Power Architecture for a given RTL implementation
- RTL + UPF Verification
  - Ensure that the design will work correctly under the power management (UPF) with the defined power architecture
- Gate Level Netlist + UPF Verification
  - Synthesis => Netlist. UPF will have to be updated for NL (manually or automated)
- Physical NL + UPF Verification
  - Gate Level NL + UPF => P&R => Physical Netlist.

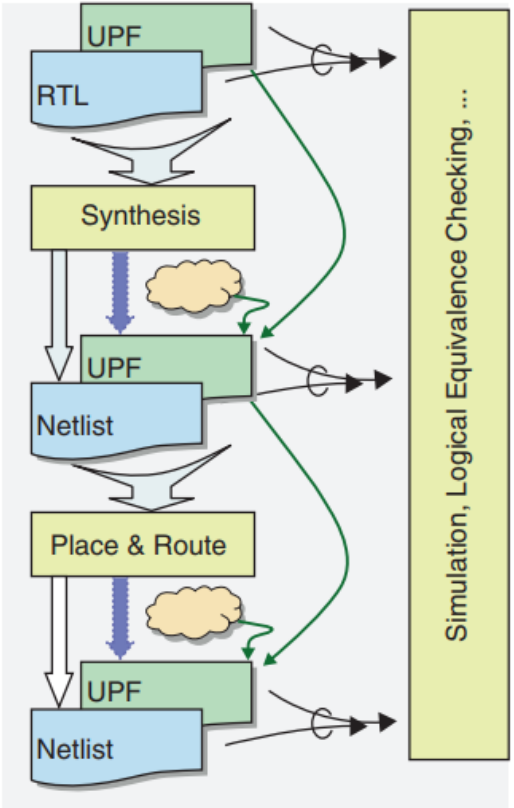


Fig. 9.3 UPF methodology

Ashok B.Mehta: ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies, Springer International Publishing AG, 2017

UPF Terminology

Power Domain

- Independently powered region

Composite Power Domain

- A power domain consisting of subordinate power domains called subdomains

Power Supply Network

- Abstract description of power distribution (ports, nets, sets, and switches)

Power State

- A subset of the functional states of an object that have the same characteristics with respect to power supply or power consumption

Power State Table

- The legal combinations of states of each power domain

State Retention

- To save essential data when power is off
- To enable quick resumption after power up

Isolation

- To ensure correct electrical/logical interaction between domains in different ON/ OFF power states
- Isolation cell is an instance that passes logic values during normal mode operation and clamps its output to some specified logic value when a control signal is asserted

Level Shifting

- To ensure correct communication between different voltage levels
- Level shifter cell is an instance that translates signal values from an input voltage swing to a different output voltage swing

Ashok B.Mehta: ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies, Springer International Publishing AG, 2017

UPF: Detailed SoC Example (Ashok B. Mehta)

Focusing on the Video_SB in this example

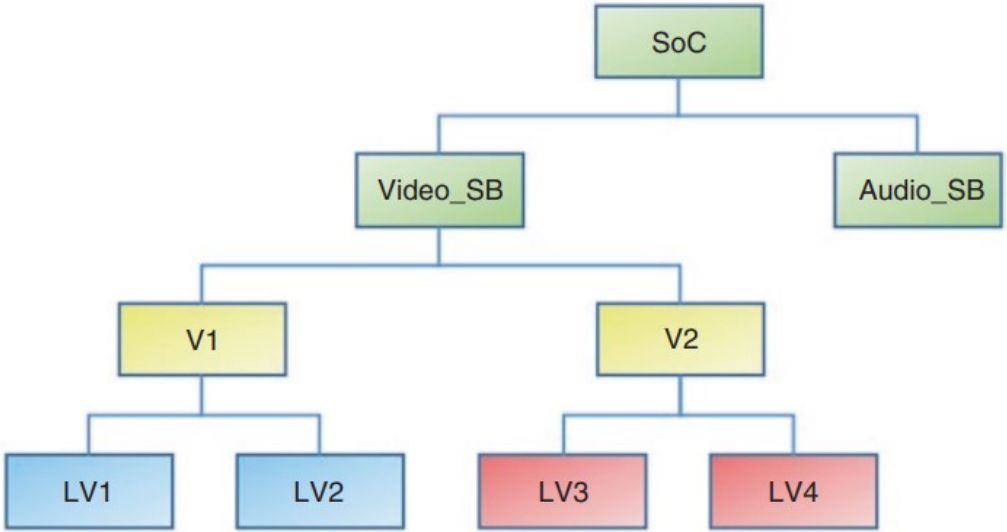


Fig. 9.4 UPF: design/logic hierarchy navigation

Ashok B.Mehta: ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies, Springer International Publishing AG, 2017

UPF: Power Domain (version 1)

```
set_scope Video_SB

create_power_domain Video_PD -include_scope

create_power_domain V1_PD \
-elements {V1}

create_power_domain LV12_PD \
-elements {V1/LV1 V1/LV2}

create_power_domain V2_PD \
-elements {V2}

create_power_domain LV34_PD \
-elements {V2/LV3 V2/LV4}
```

One way of dividing the design into power domains

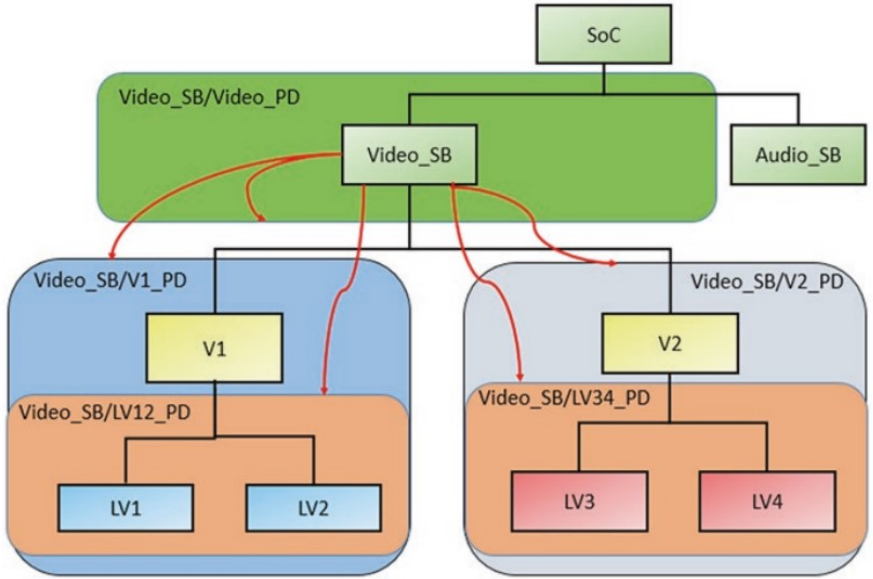


Fig. 9.5 UPF: power domain creation – 1

UPF: Power Domain (version 2)

```
set_scope Video_SB

create_power_domain Video_PD -include_scope

create_power_domain V_PD -elements {V1 V2}

create_power_domain LV_PD \
-elements {V1/LV1 V1/LV2 V2/LV3 V2/LV4}
```

Another way of dividing the design into power domains

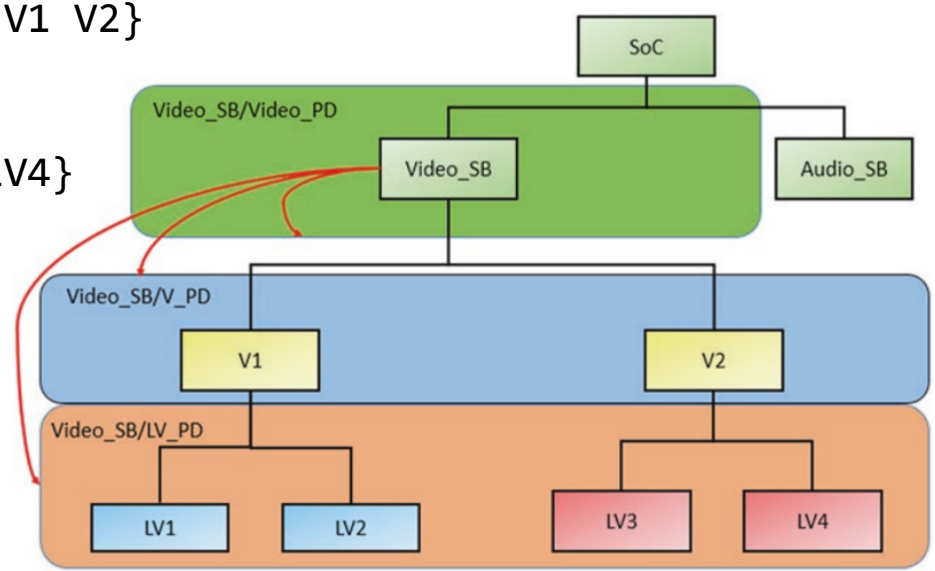


Fig. 9.6 UPF: power domain creation—2

UPF: Supply Power

```
create_supply_port VDD -direction in
create_supply_port VSS -direction in

create_supply_net Pwr -domain Video_PD
create_supply_net Gnd -domain Video_PD

connect_supply_net Pwr -ports (VDD)
connect_supply_net Gnd -ports (VSS)

set_domain_supply_net Video_PD \
-primary_power_net Pwr \
-primary_ground_net Gnd
```

Add ports (VDD,VSS) and nets (Pwr, Gnd) and establish the domain name for these nets

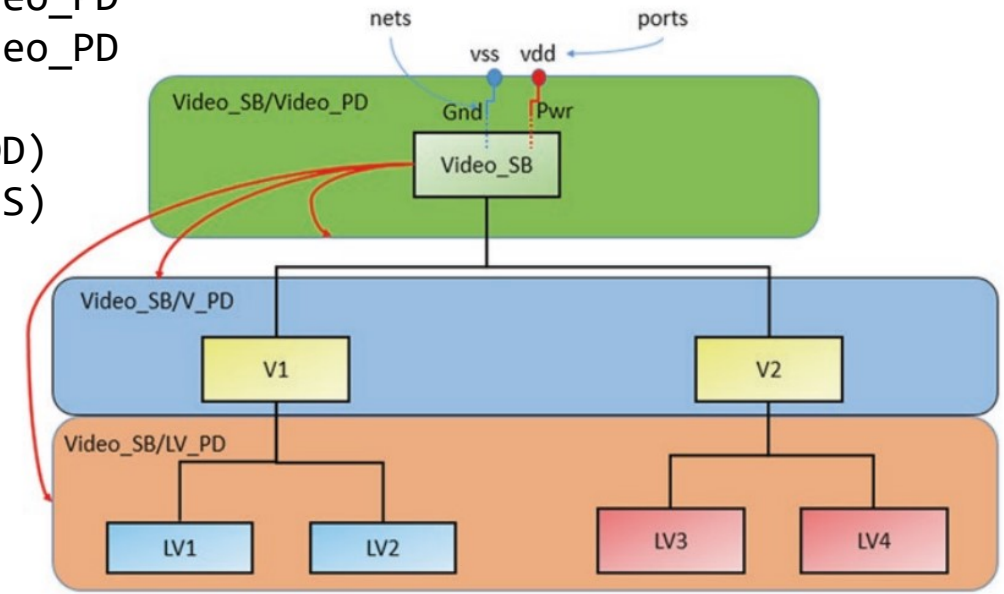


Fig. 9.7 UPF supply network

UPF: Extend the supply nets

```
create_supply_net Pwr -reuse -domain V_PD
create_supply_net Gnd -reuse -domain V_PD

set_domain_supply_net V_PD \
-primary_power_net Pwr \
-primary_ground_net Gnd
```

Extend Pwr and Gnd nets to domain V_PD: Reuse the existing nets

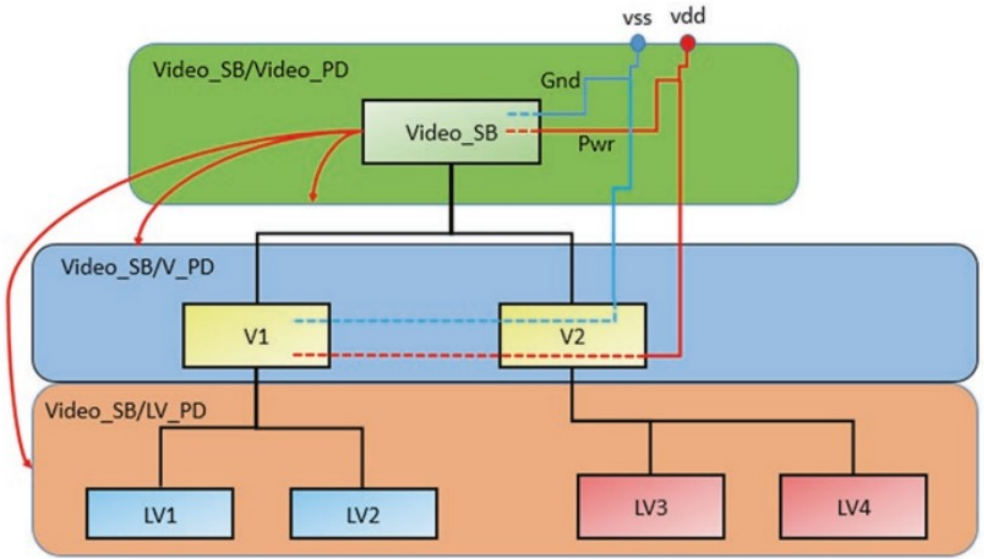


Fig. 9.8 UPF: supply network reuse

UPF: Power Switch

```
create_supply_net VDDsw -domain V_PD

create_power_switch SW -domain V_PD \
-input_supply_port {pwin Pwr} \
-output_supply_prt {pwout VDDsw }
-control_port {swctrl sw_ctrl} \
-on_state {Pwon swctrl} \
-off_state {Pwoff !swctrl}

set_domain_supply_net V_PD \
-primary_power_net VDDsw \
-primary_ground_net Gnd
```

Add a switched supply net VDDsw.
Create the switch SW that is controlled by sw_ctrl:
- Assign input, output, and control of that switch and declare on and off states.
- As before, establish the domain in which the supply nets belong

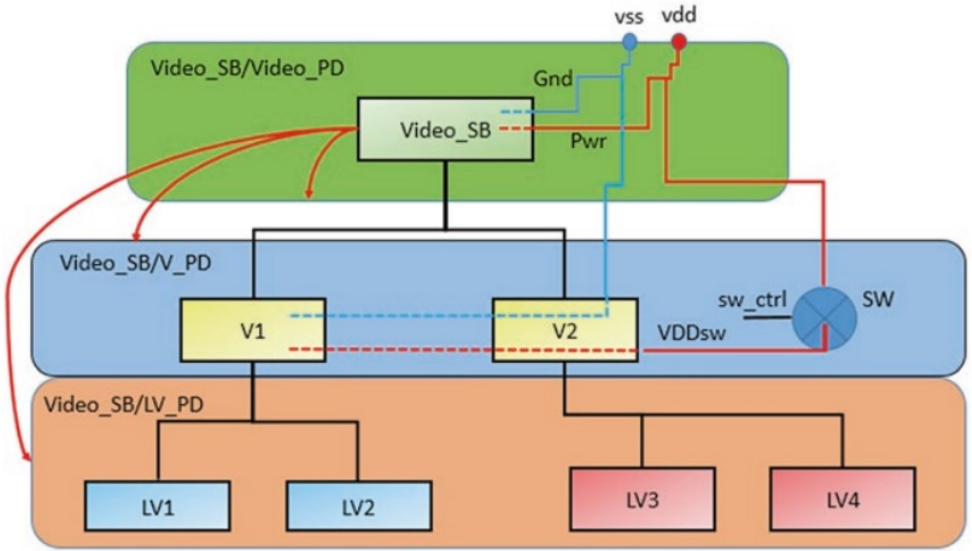


Fig. 9.9 UPF: power switch creation

UPF: Second Voltage Supply

```
create_supply_port VDD2 -direction in

set_domain_supply_net LV_PD \
-primary_power_net VDD2 \
-primary_ground_net Gnd
```

Let's create one more supply VDD and connect it to power domain LV_PD

Note: We did not create a net this time! (a port name can be used where a net name is allowed)

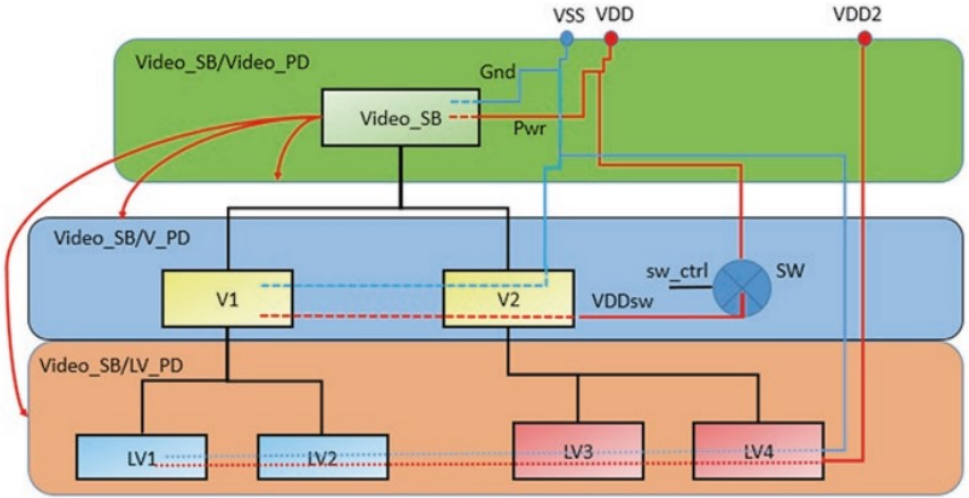


Fig. 9.10 UPF: supply port states

UPF: Supply Port States

Defining port states:
VDD always ON at 1.0V,
VSS always ON at 0V,
VDDsw can be ON(1V)/OFF,
VDD2 can be ON(0.8V)/OFF

```
add_port_state VDD -state {ON_10 1.0}

add_port_state SW/VDDsw -state {ON_10 1.0} -state {OFF off}

add_port_state VSS -state {ON_00 0.0}

add_port_state VDD2 \
-state {ON_08 0.8} \
-state {OFF off}
```

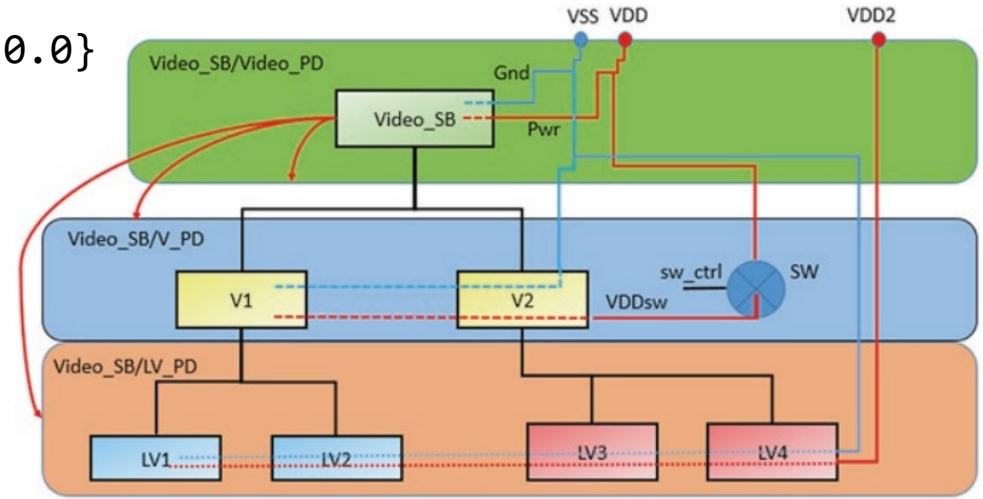


Fig. 9.10 UPF: supply port states

UPF: Power State Table

Create Power State Table PST1 and add states for Normal, Sleep, and Hibernate

```
create_pst PST1 -supplies {VDD, VDDsw, VDD2, VSS}

add_pst_state Normal -pst PST1 -state {ON_10, ON_10, ON_08, ON_00}

add_pst_state Sleep -pst PST1 -state {ON_10, OFF, ON_08, ON_00}

add_pst_state Hibernate -pst PST1 -state {ON_10, OFF, OFF, ON_00}
```

	Video_PD	V_PD	LV_PD	
State \ Supply	VDD	VDDsw	VDD2	VSS
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Fig. 9.11 UPF: power state table

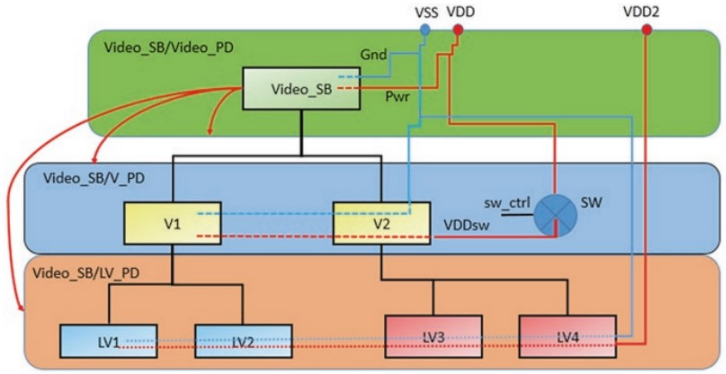


Fig. 9.10 UPF: supply port states

UPF: Retention

```
set_retention V_PD_retention \
-domain V_PD \
-retention_power_net Pwr \
-retention_ground_net Gnd

set_retention_control V_PD_retention \
-domain V_PD \
-save_signal {SRctrl posedge} \
-restore_signal {SRctrl negedge}
```

Let's assume we need to retain the state of V_PD so that it can restart quickly as soon as it wakes up from Sleep of Hibernate.
- The retention block is connected to always-on nets Pwr and Gnd (Not VDDsw)

Note: We don't use retention in LV_PD: Retention strategy is not always required.

	Video_PD	V_PD	LV_PD	
State \ Supply	VDD	VDDsw	VDD2	VSS
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Fig. 9.11 UPF: power state table

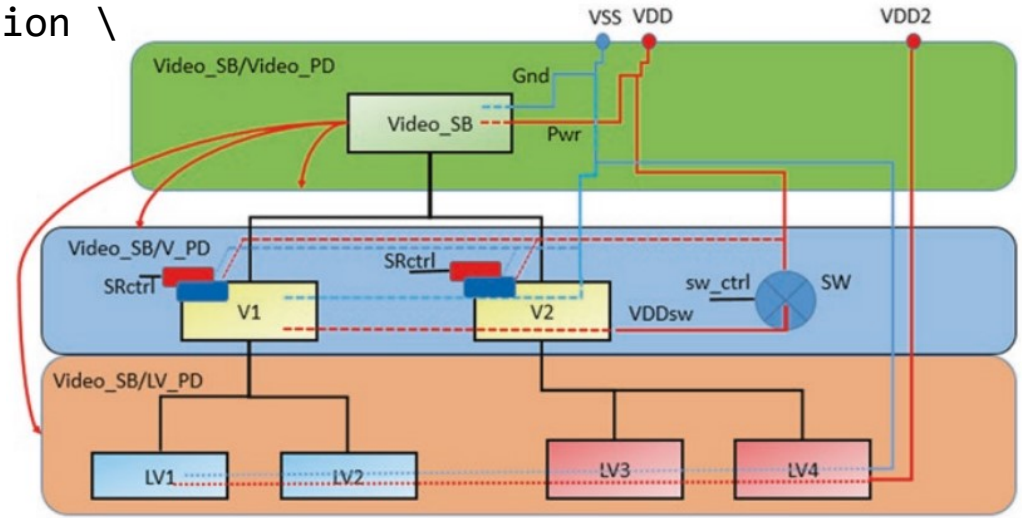


Fig. 9.12 UPF: state retention strategy

UPF: Isolation

```
set_isolation V_PD_isolation \
-domain V_PD \
-applies_to outputs \
-clamp_value 0 \
-isolation_power_net Pwr \
-isolation_ground_net Gnd

set_isolation_control V_PD_isolation \
-domain V_PD \
-isolation_signal vISO \
-isolation_sense high \
-location parent
```

When V_PD turns OFF, its outputs (inputs to Video_PD) cannot be in random (unknown) state!
- Clamp outputs to ’0’ when V_PD is OFF

Also, add control with signal vISO. These cells will be inserted to the parent (Video_PD)

	Video_PD	V_PD	LV_PD	
State \ Supply	VDD	VDDsw	VDD2	VSS
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Fig. 9.13 State table showing isolation requirements

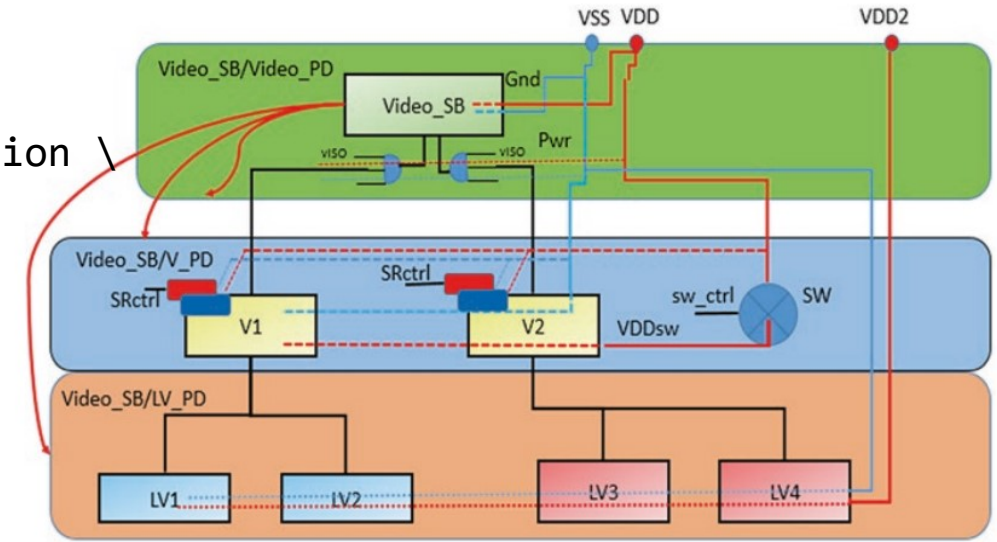


Fig. 9.14 UPF state isolation strategy

UPF: Level Shifters

```
set_level_shifter LV_PD_LS
-domain LV_PD \
-threshold 0.1 \
-applies_to both \
-rule both \
-location self
```

VDD is higher than VDD2: V_PD and LV_PD operate on different voltages.

Add levels shifters between them:
- Add them to both inputs and outputs of LV_PD
- Shift both from High_to_Low and Low_to_High
- The level shifters are inserted into LV_PD

	Video_PD	V_PD	LV_PD	
State \ Supply	VDD	VDDsw	VDD2	VSS
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Fig. 9.15 Level shifting strategy

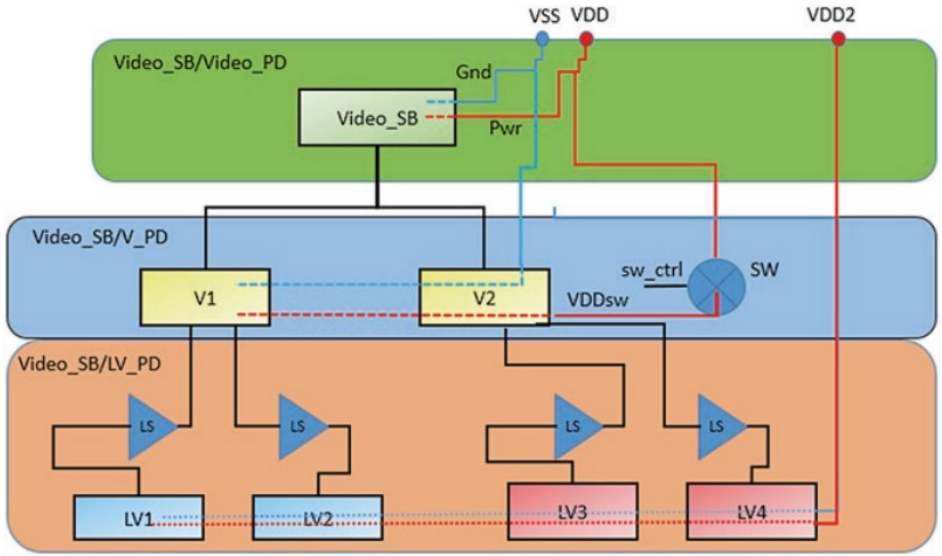


Fig. 9.16 UPF: level shifter strategy

UPF: Result

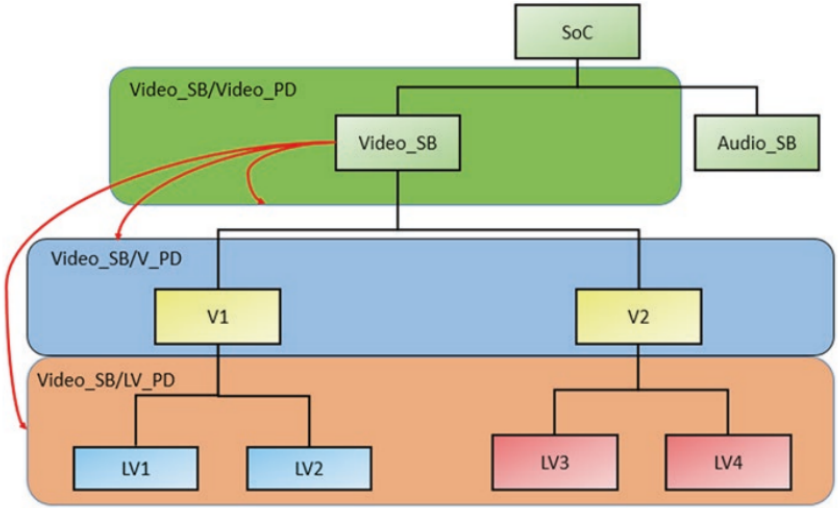


Fig. 9.6 UPF: power domain creation—2

	Video_PD	V_PD	LV_PD	
State \ Supply	VDD	VDDsw	VDD2	VSS
Normal	ON_10	ON_10	ON_08	ON_00
Sleep	ON_10	OFF	ON_08	ON_00
Hibernate	ON_10	OFF	OFF	ON_00

Fig. 9.11 UPF: power state table

Note: We did not touch the RTL at all

Of course, our RTL should be designed so that these additional structures do not cause bugs.

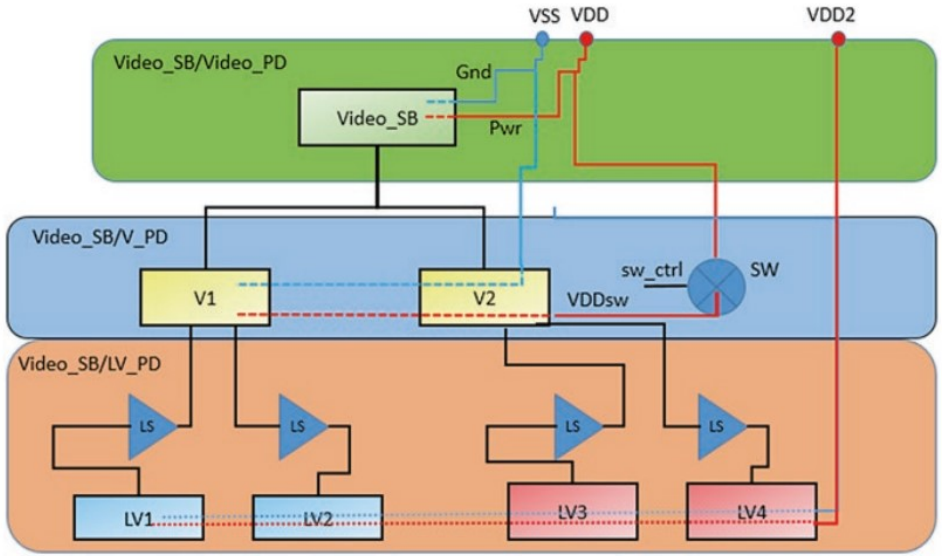
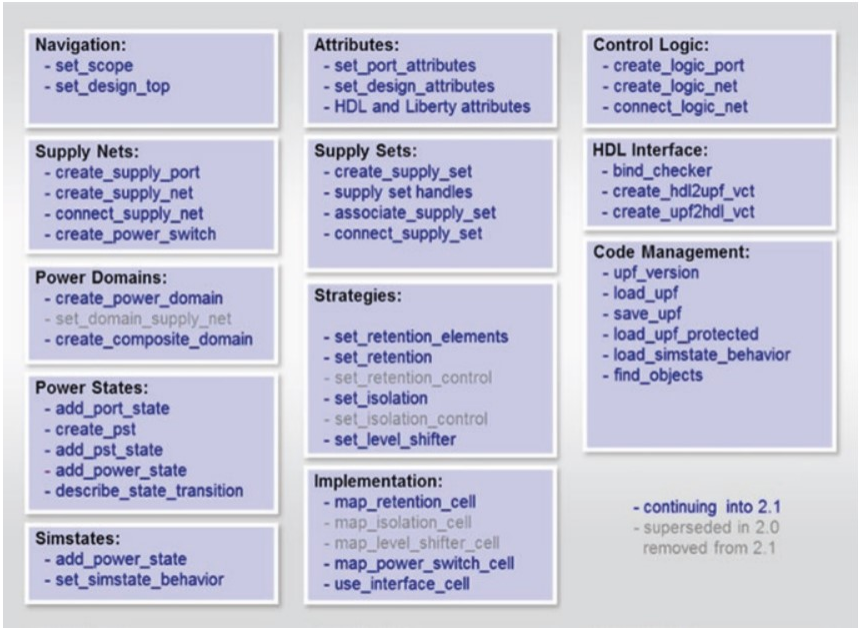


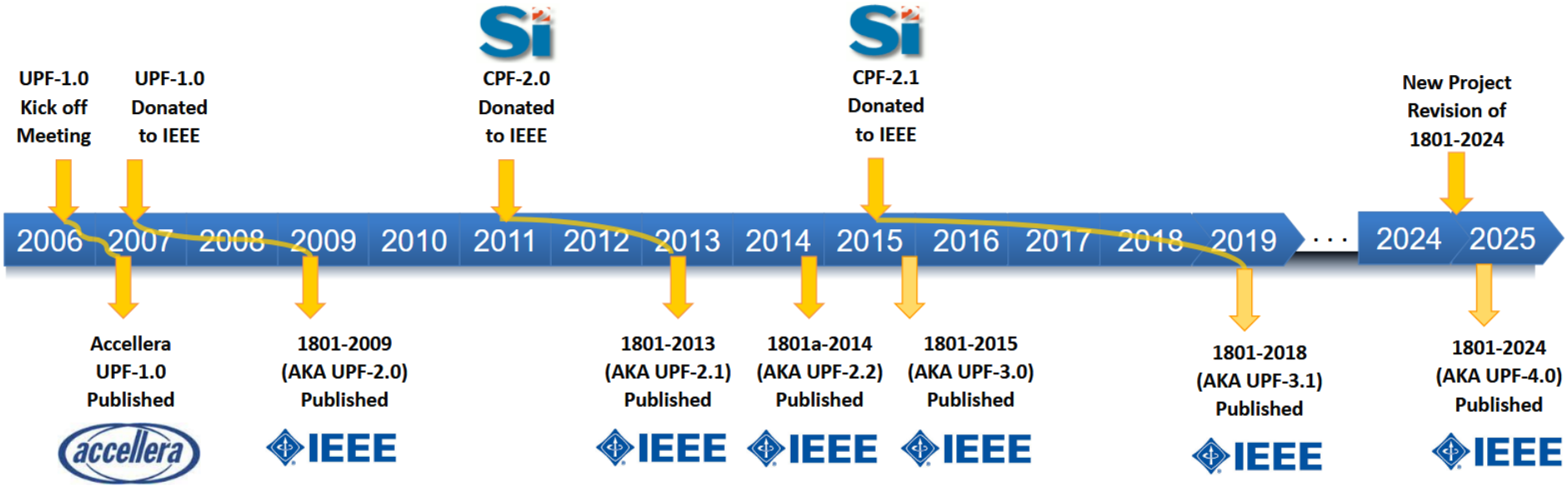
Fig. 9.16 UPF: level shifter strategy

UPF 2.0 Features (IEEE 1801-2009)



Ashok B.Mehta: ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies, Springer International Publishing AG, 2017

Evolution of the Standard



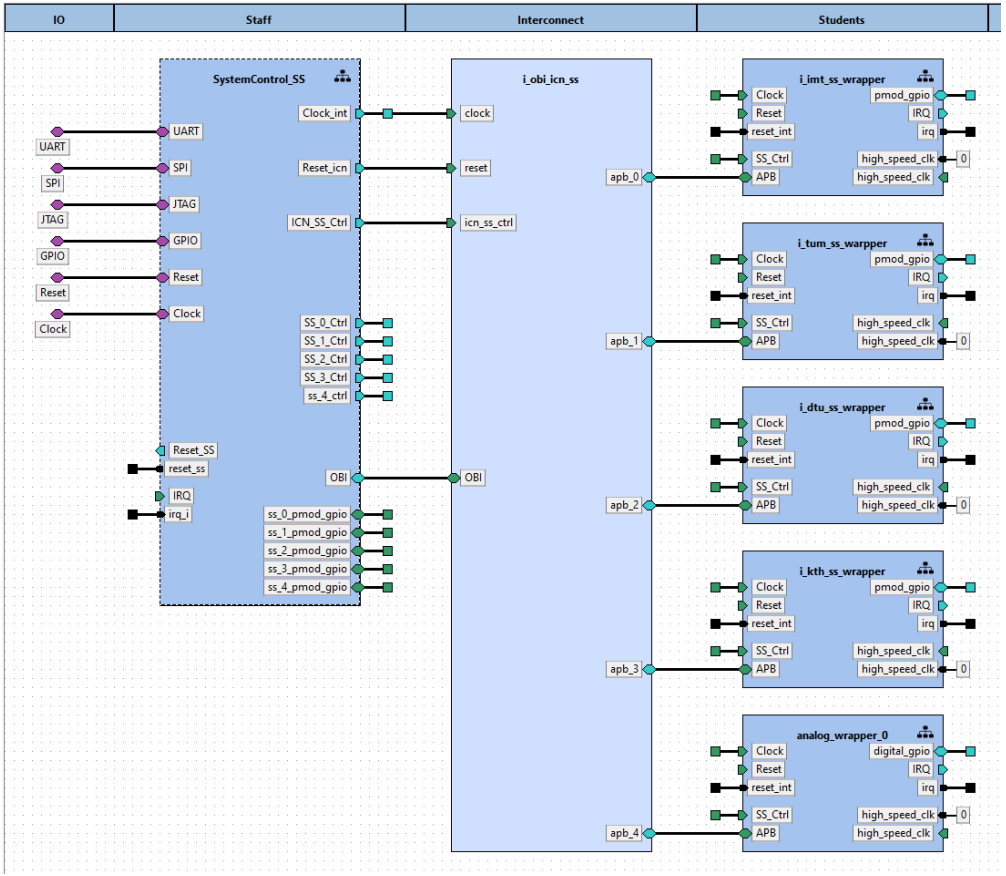
Didactic Domains

How would you divide this SoC to power domains?

Clock domains?

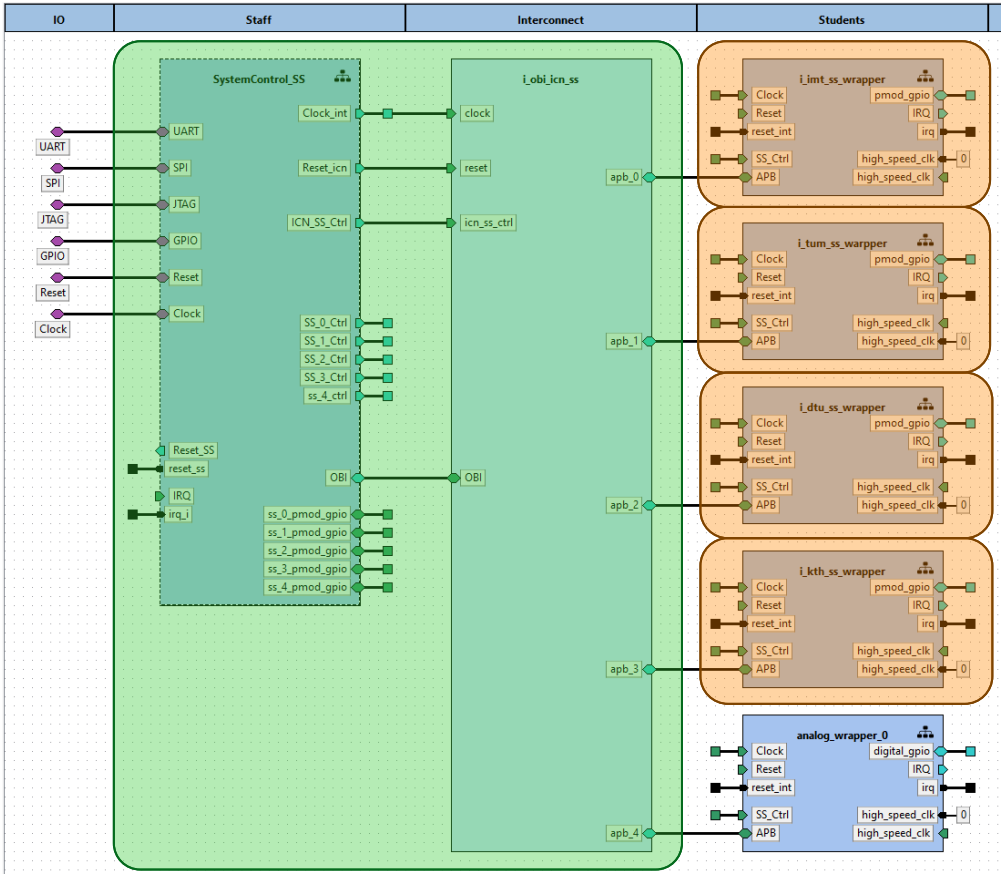
Where to put CDCs?

State Table: what should be always on, which parts can sleep?



Didactic Domains

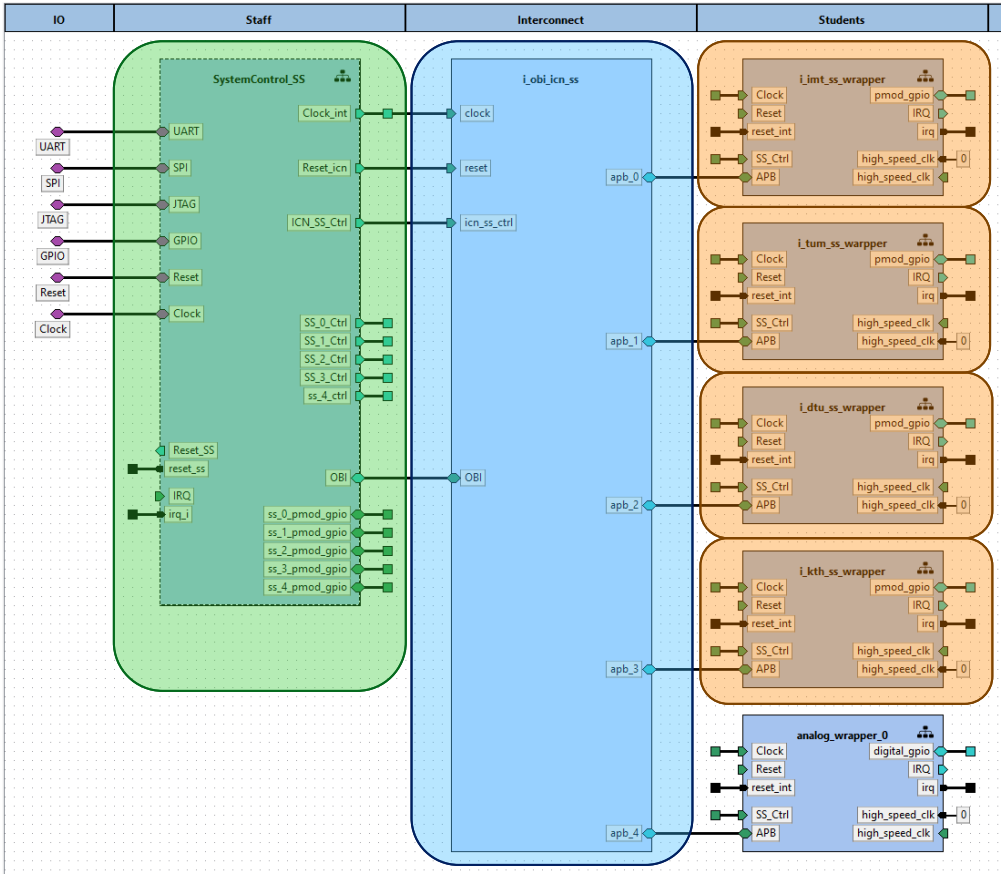
Staff ON, Students switchable?



Didactic Domains

Staff ON, Students switchable

Staff ON, Students switchable, ICN sleep?

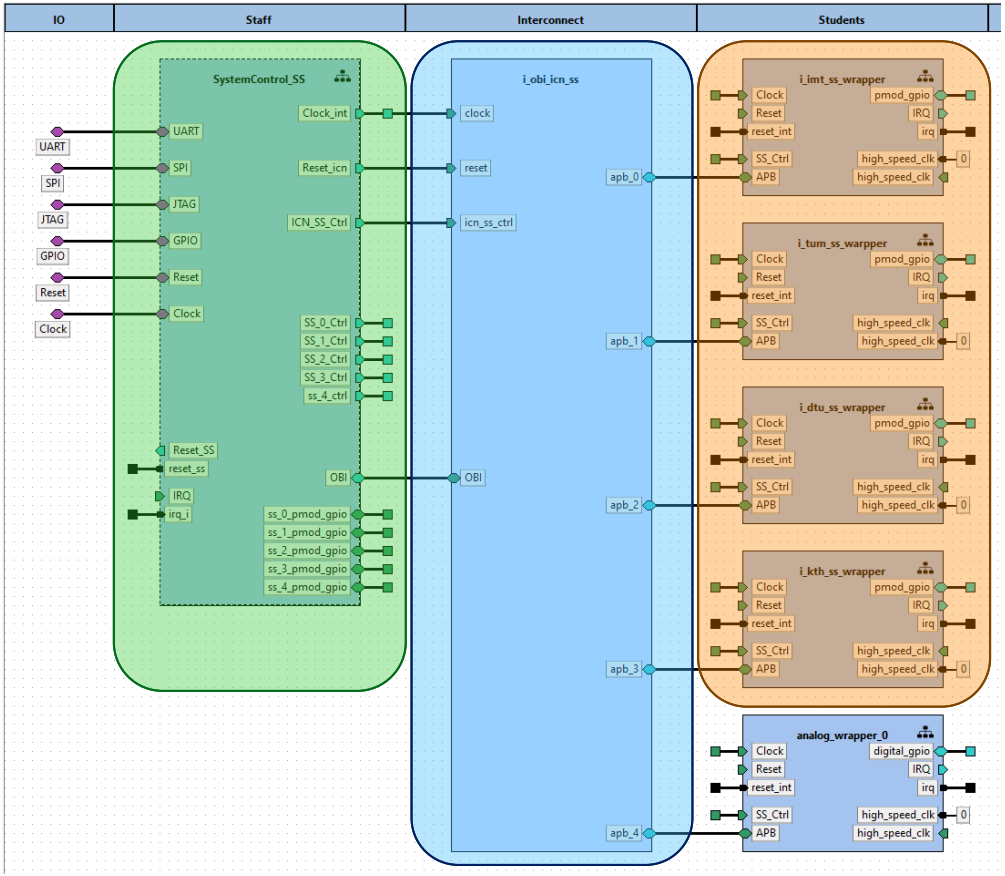


Didactic Domains

Staff ON, Students switchable

Staff ON, Students switchable, ICN sleep

Staff ON, Single switchable student domain, ICN sleep?



Didactic Domains

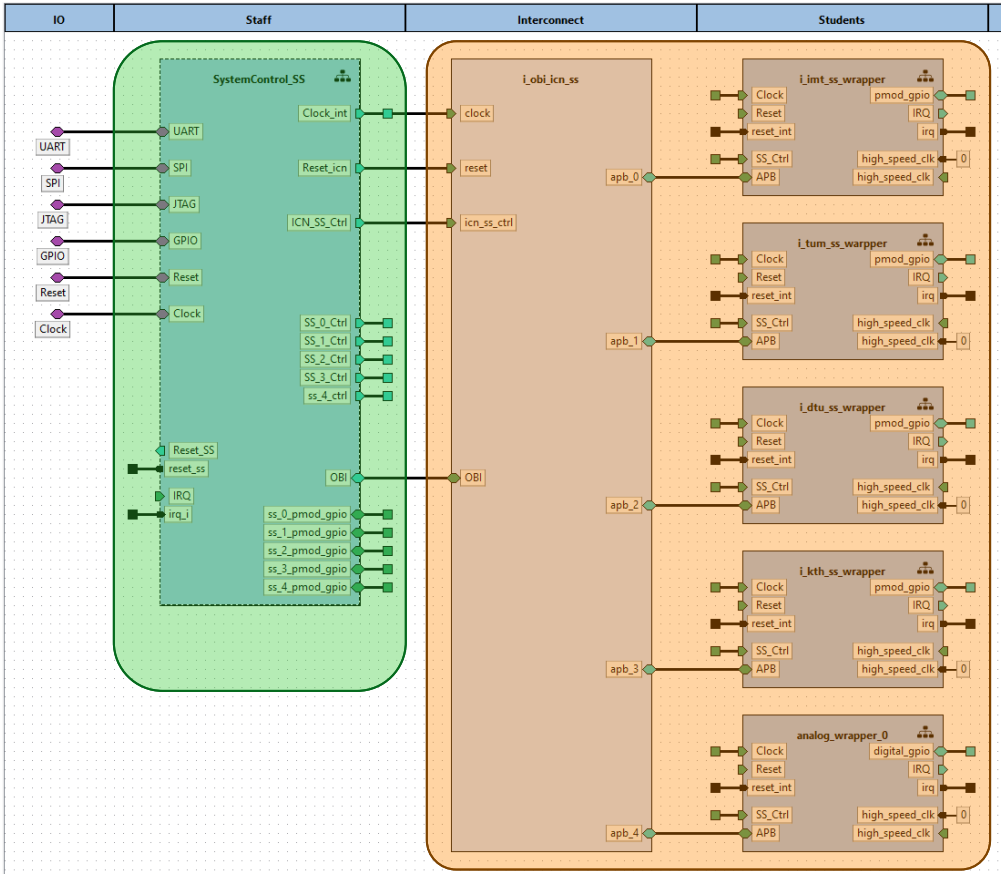
Staff ON, Students switchable

Staff ON, Students switchable, ICN sleep

Staff ON, Single switchable student domain, ICN sleep

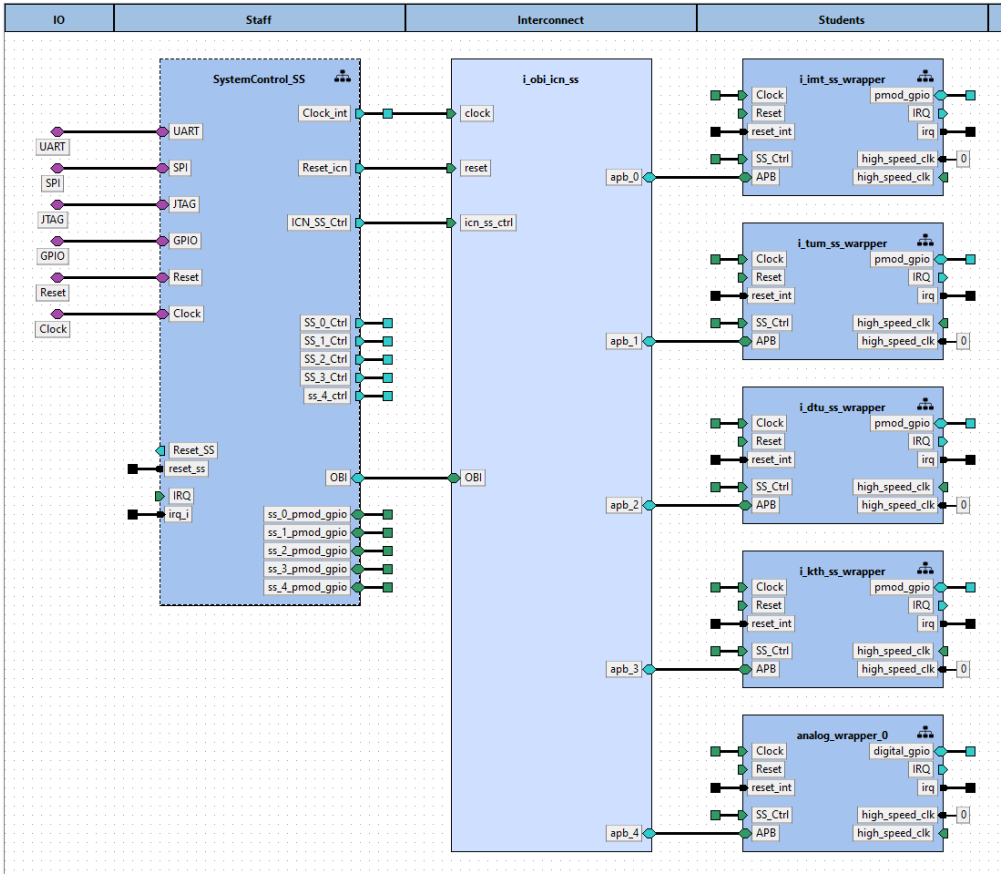
Staff ON, Single switchable student domain, ICN in student domain?

Not a single right answer.



Didactic Domains

The same exercise for clock domains



Didactic Domains

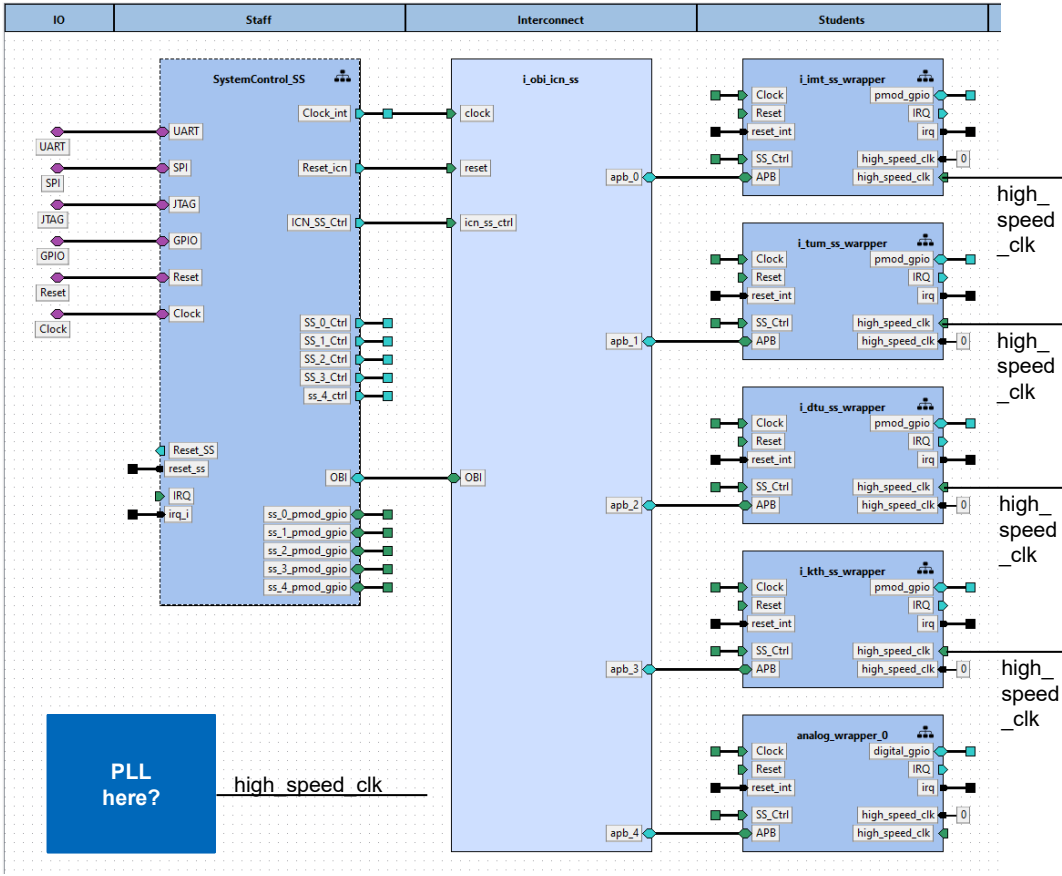
The same exercise for clock domains

- *PLL and CDCs inside each SS_wrapper?*
- *or, a single high-speed PLL on top level?*
  (the high-speed clk input is already in the wrapper)
- *both? Drive interconnect on top level PLL and subsystems with their own ones?*
  (that might be overkill here)

Staff domain: reference clock (100MHz)

- ***Interconnect also in this domain***
-

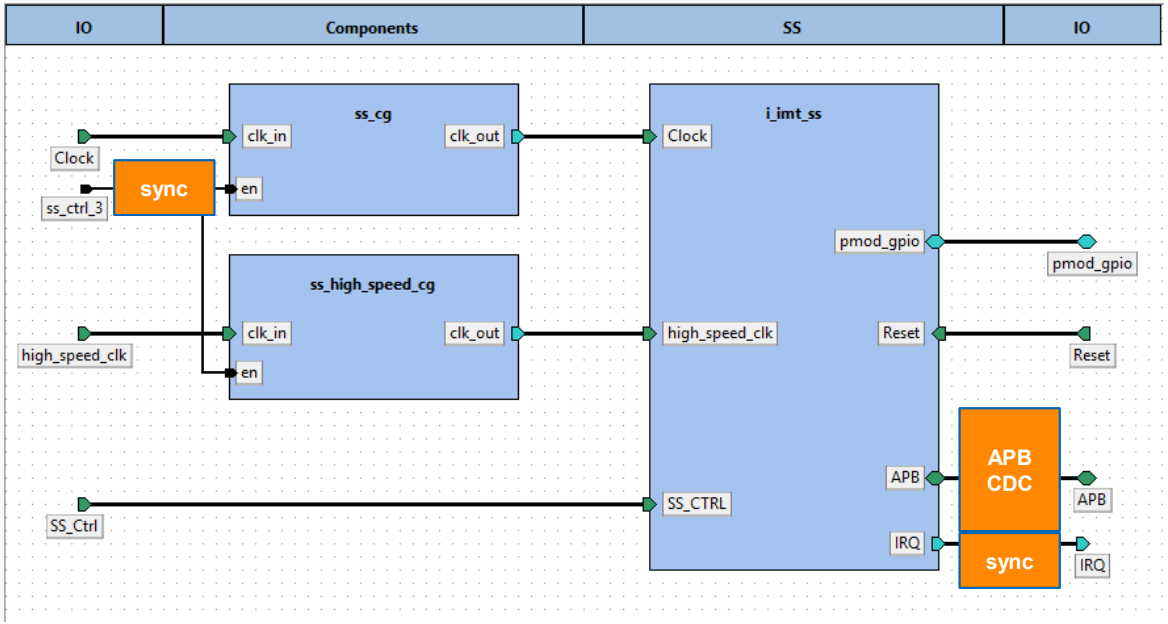
Student domain: high-speed clock (1GHz)



Didactic Subsystem Wrapper

Student subsystem as a separate clock domain

- *Clock gating already implemented*
- *Add APB CDC in the wrapper*
- *Add synchronization for the ad-hoc control signals*
- *Reset synchronization also requires special attention*



Not an official part of the exercise project, but bonus points are surely granted if you take the challenge and implement this!

Conclusion

- Power management and clocking considerations are integral to modern SoC design
  - The low-hanging fruits are on the architecture and behavioral design stage!
- SoC design is not only pure RTL but should be augmented with UPF early in the process
  - UPF follows all the way through the design flow
  - Also in low-power verification
- There are multiple ways to implement clocking, but GALS is a good principle
  - if you can divide your SoC to clear subsystems
  - The SoC Hub way: common subsystem wrapper template helps in integration
- Open-source blocks for clock gating and clock domain crossing exist for usual bus interfaces (AXI, APB)

Design abstraction stage	%power saving (%)
System design and architecture	70–80
Behavioral design	40–70
RTL design	25–40
Logic design	15–25
Physical design	10–15

Potential power savings in each abstraction stage

T. Vaibbhav, "14. Low Power Design," in Digital Logic Design Using Verilog : Coding and RTL Synthesis. 2nd ed, New Delphi: Springer India, 2022, pp. 535-558

More on clocking and low-power design in COMP.CE.510 Chip Implementation

Constraints and STA
Clock Tree Synthesis
Multi-Vt cells: optimizing each transistor
The gating structures on ASIC
Low-level details and more UPF examples
Much more