



SOC & ASIC DESIGN AT ERICSSON

Björn Fjellborg
Ericsson AB
Kista



1

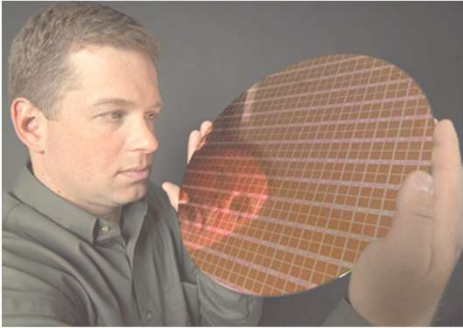
Anything happened since 1984?



2


THIS LECTURE

- › Mobile networks and HW infrastructure
- › Systemization and System design
- › Design challenges
- › SoC/ASIC design flow



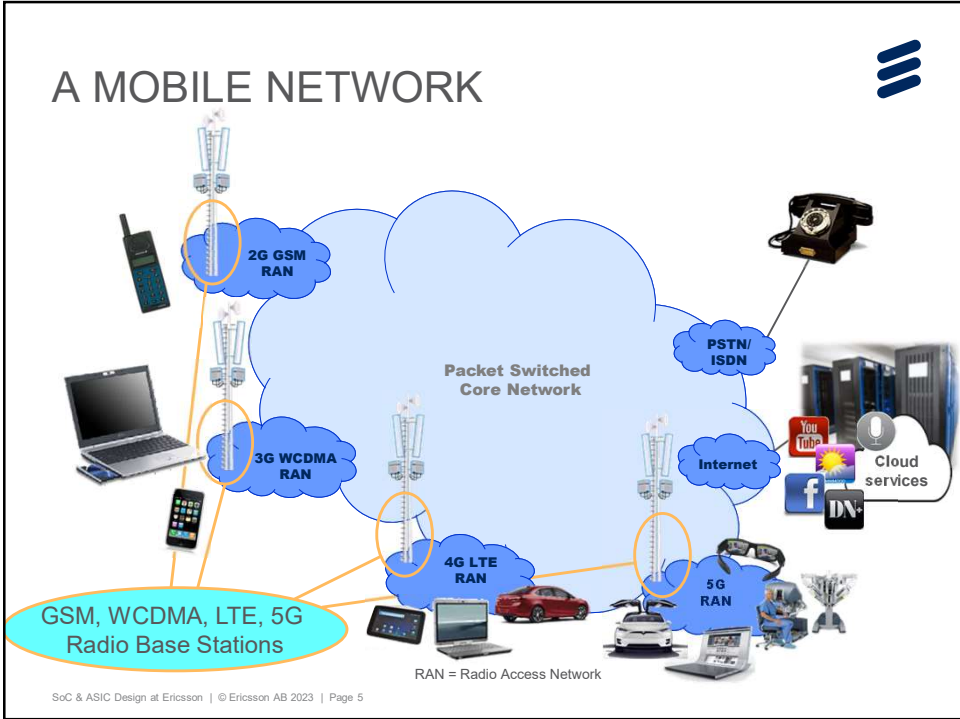
SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 3

3

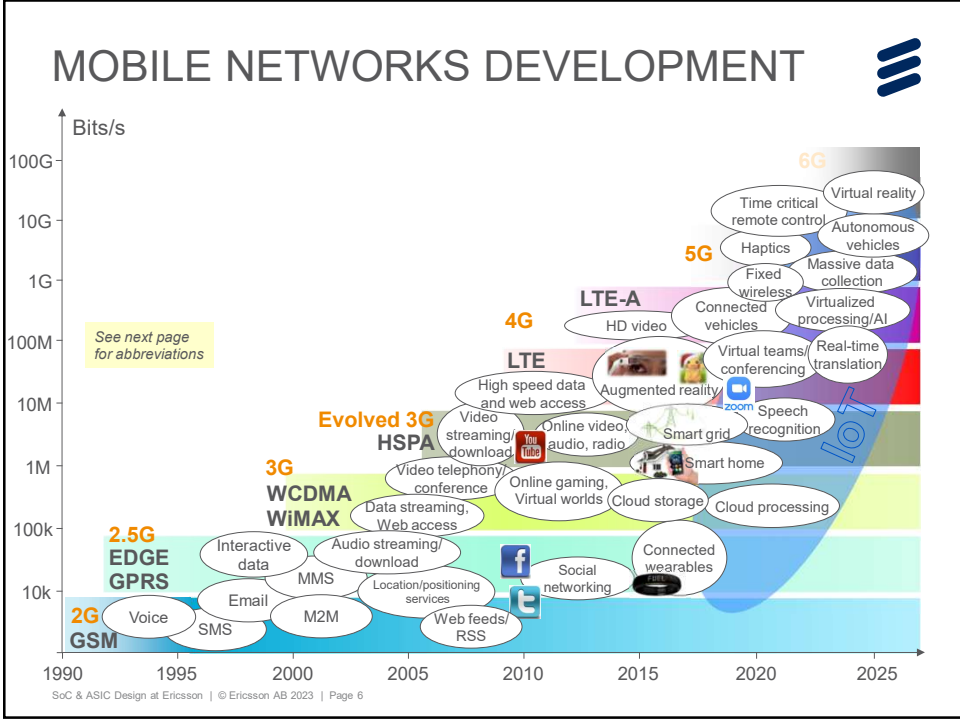


Mobile Networks and
HW Infrastructure

4



5



6

ABBREVIATIONS



- › AI Artificial Intelligence
- › EDGE Enhanced Data rates for GSM Evolution
- › GPRS General Packet Radio Service
- › GSM Global System for Mobile communications
- › HD High Definition
- › HSPA High-Speed Packet Access
- › IoT Internet of Things
- › LTE Long Term Evolution
- › LTE-A Long Term Evolution Advanced
- › MMS Multimedia Messaging Service
- › M2M Machine to Machine
- › RSS Really Simple Syndication
- › SMS Short Messaging Service
- › WCDMA Wideband Code Division Multiple Access
- › WiMAX Worldwide Interoperability for Microwave Access


SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 7

7

ERICSSON RADIO BASE STATIONS



The most visible parts of a base station are usually the antennas (and antenna mast)



INDOOR



Cabinets come in different sizes, for different capacity



SMALL INDOOR

The cabinet housing the HW and SW is located at a sealed off area below the antenna mast or inside a nearby building



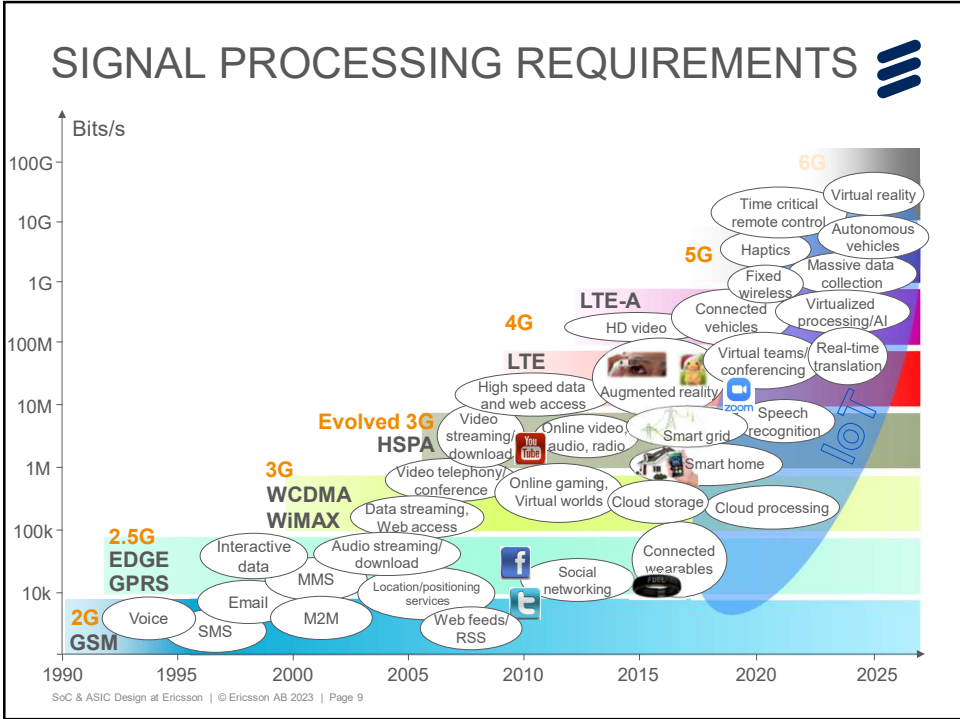
OUTDOOR



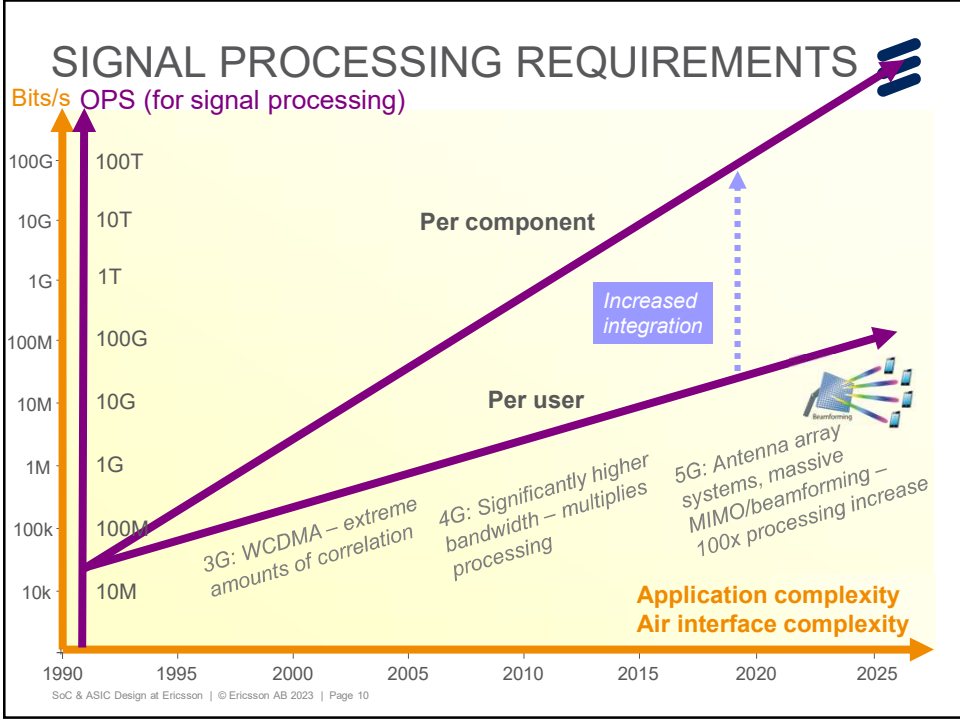
RADIO DOT

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 8

8




9



10


ERICSSON SILICON INDUSTRY LEADING SYSTEM ON A CHIP



Ericsson Many-Core Architecture (EMCA)

Hundreds of digital signaling processors (DSPs)

4G/5G HW Accelerators



Digital front-end

Tight SW/HW co-design

Software development kit (SDK) for designing massively parallel radio algorithms

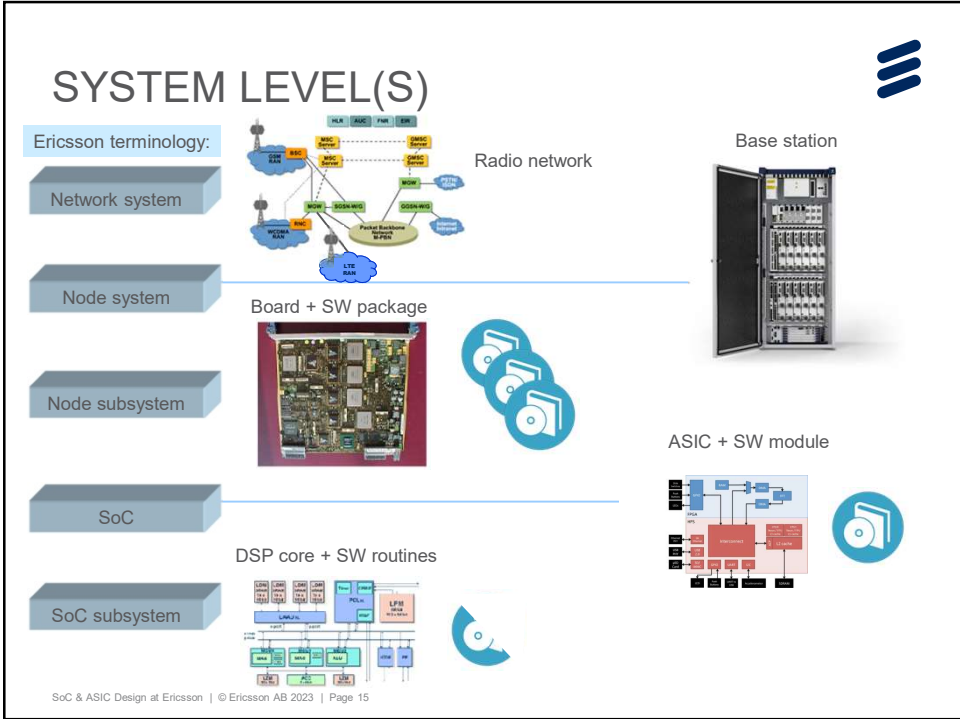
SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 11

11

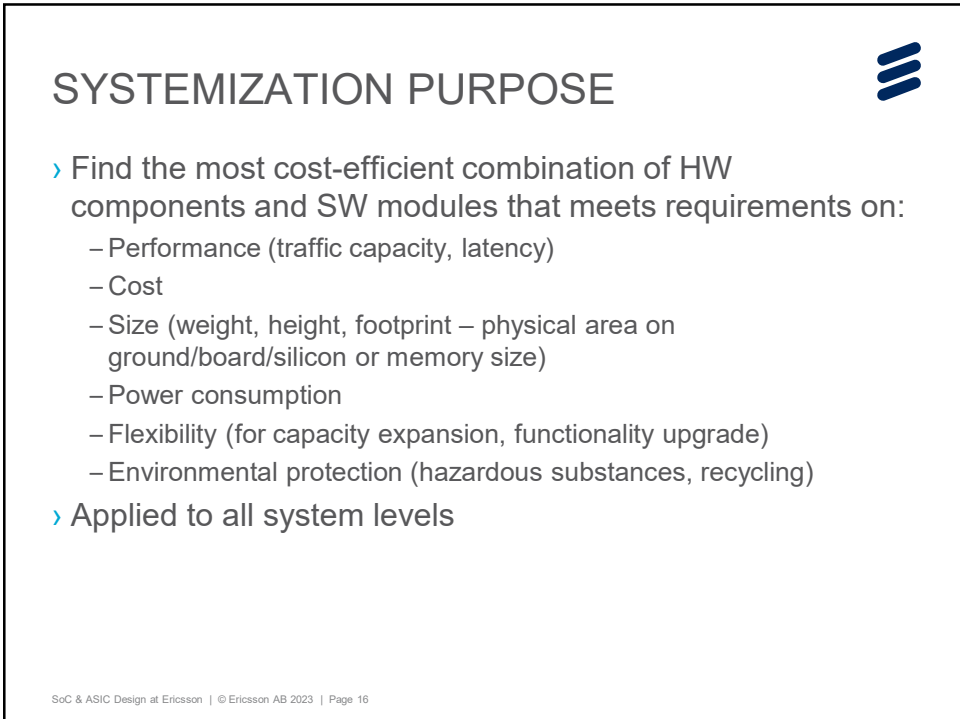


Systemization and System Design

14



15

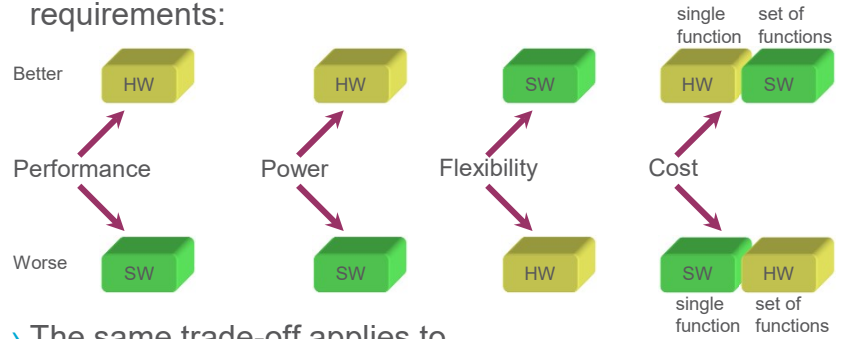


16

HW/SW TRADE-OFFS



› System design is in many cases a trade-off between HW and SW solutions to best meet the systemization requirements:



- › The same trade-off applies to
 - ASIC/FPGA vs DSP/CPU
 - Hardwired logic vs DSP/CPU cores

17

HW VS SW PERFORMANCE



- › HW has higher performance than SW because:
 - HW tailored to a specific functionality requires fewer gates than a general instruction execution engine
 - ⇒ Higher capacity/gate
 - Tailored HW can exploit parallelism to a higher degree than a DSP/CPU
 - ⇒ Lower latency (faster execution of complex functions)

18

HW VS SW POWER



- › HW has lower power dissipation than SW because:
 - Tailored HW uses few transistor switches/function (HW optimized for the specific function)
 - ⇒
Low energy/function step

 - SW that runs on an instruction execution engine induces many transistor switches/function (several SW instructions to load, decode, and execute)
 - ⇒
High energy/function step

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 19

19

HW VS SW FLEXIBILITY



- › SW has higher flexibility than HW because:
 - Correcting errors in SW requires no new HW
 - ⇒
Faster correction at customer site

 - Upgrading functionality in SW may require more memory but no new HW
 - ⇒
No HW production cost for upgrading, say, 100 000 customer sites (provided enough memory was designed in)

 - Note: Functionality upgrades in SW do not necessarily get to customer faster than HW upgrades, because development and verification times are comparable to those of HW (for complex systems)!

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 20

20

HW VS SW COST



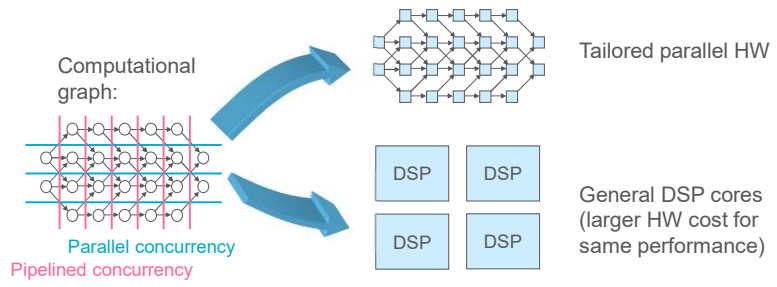
- › HW has lower cost than SW when:
 - The functionality can be implemented on a tailored piece of HW that is cheaper/smaller than a general purpose DSP/CPU
- › A set of functions may have lower SW cost (a DSP/CPU):
 - Re-uses the same HW (instruction execution engine) for multiple functions
 - HW tailored to each function
 - ⇒ Cost = Σ tailored HW blocks > 1 DSP/CPU

21

HW VS SW COST FURTHER CONSIDERATIONS



- › Concurrency-based allocation:
 - Has inherent concurrency: Use HW (utilize HW concurrency)
 - › May require a large number of DSP/CPU to obtain same performance
 - › Typical for filter functions
 - Has inherent sequentiality: Use SW (no speed-up with tailored HW)



22

EXAMPLE: RANDOM ACCESS & DEMODULATION (RA-DEM)



› WCDMA Uplink Baseband Antenna Near Signal Processing

- › A result from node subsystem systemization for baseband is that HW support for Random Access and Demodulation is needed in the receiver chain (for WCDMA receivers).
- › A RA-DEM detects and correlates all reflections of coded signals into one signal per transmitter
- › A RA-DEM performs a number of functions:
 - Preamble detect
 - Message search
 - Rake finger despread
- › The RA-DEM functions imply a number of operations:
 - Code match filtering
 - Interference estimation
 - Coherent and non-coherent accumulation
 - Interpolation
 - Squaring
 - ...

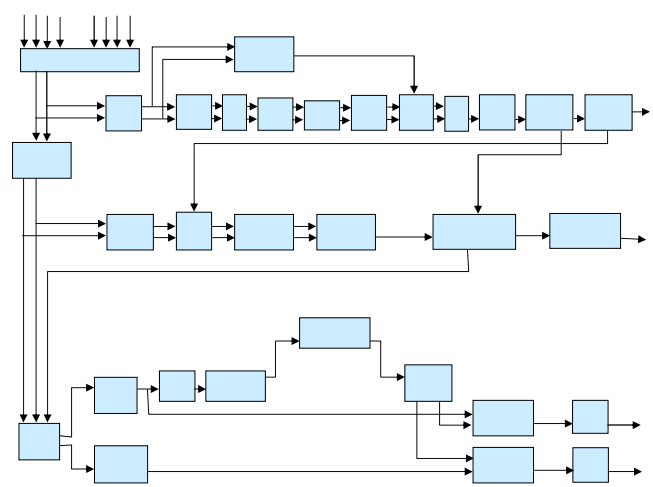
SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 23

23

RA-DEM ARCHITECTURE



First SoC/SoC subsystem systemization*:



- Node subsystem systemization has resulted in a RA-DEM algorithm, given requirements at that level
- SoC systemization has defined subfunctions (boxes in the picture)
- All subfunctions are performed inside the SoC ASIC

• System design task:
 For each box, decide whether to use

- hardwired logic
- SW (DSP core)

* Subfunctions are anonymized for confidentiality reasons
SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 24

24

RA-DEM DESIGN GOAL

- › Main goal: Minimize cost
- › Constraints:
 - Performance (can be determined with high precision per subfunction from the algorithm and max input data rate)
 - Power (total for the SoC; if the SoC power budget does not hold, re-systemization has to be done on the SoC or even the node subsystem level)
 - Flexibility (flexibility requirements are limited to those subfunctions that have to change in case of future upgrades)

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 25

25

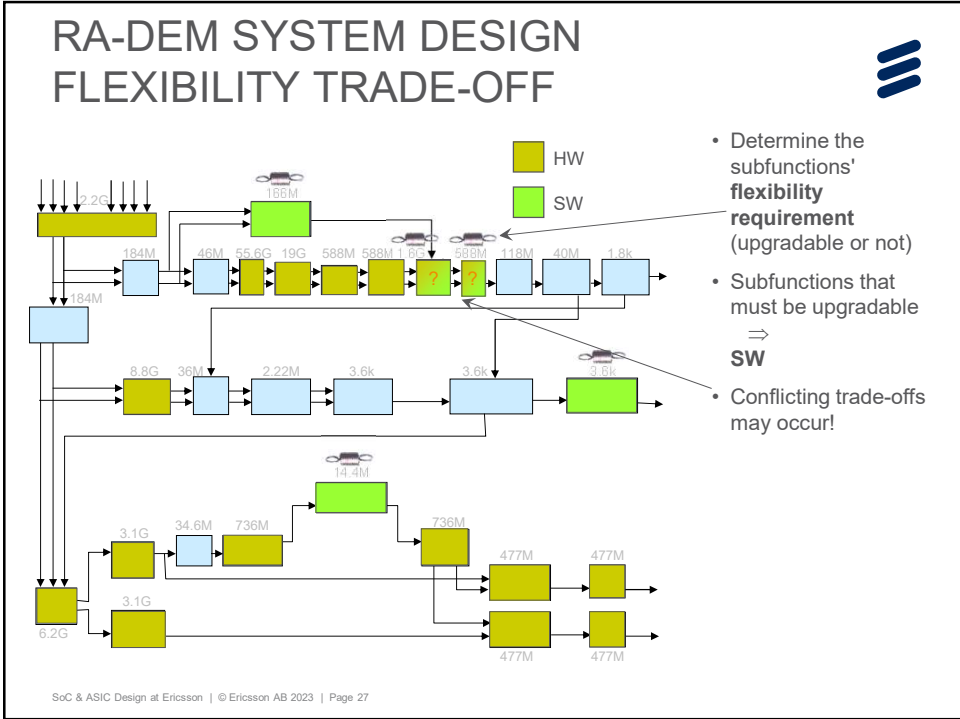
RA-DEM SYSTEM DESIGN PERFORMANCE TRADE-OFF

- Determine the subfunctions' **peak performance*** requirement* (in OPS)
- Subfunctions that exceed one DSP core's performance (350 MOPS) => **HW** (to avoid splitting over several DSPs)

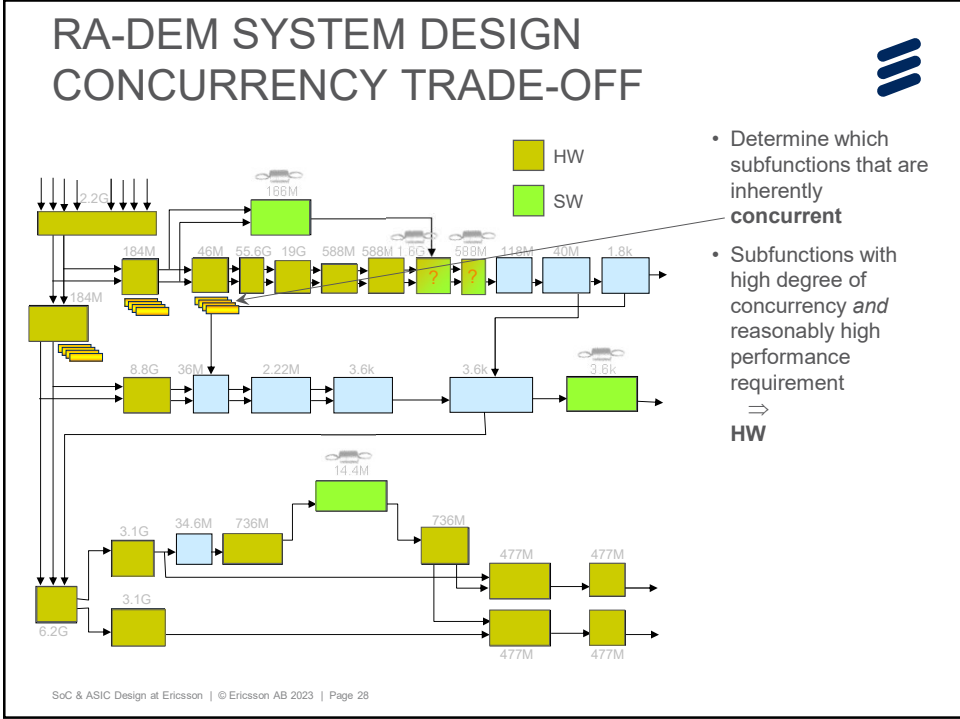
* A function that requires 10 MOPS and has to finish in 0.1 s has a peak performance of 10/0.1 = 100 MOPS

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 26

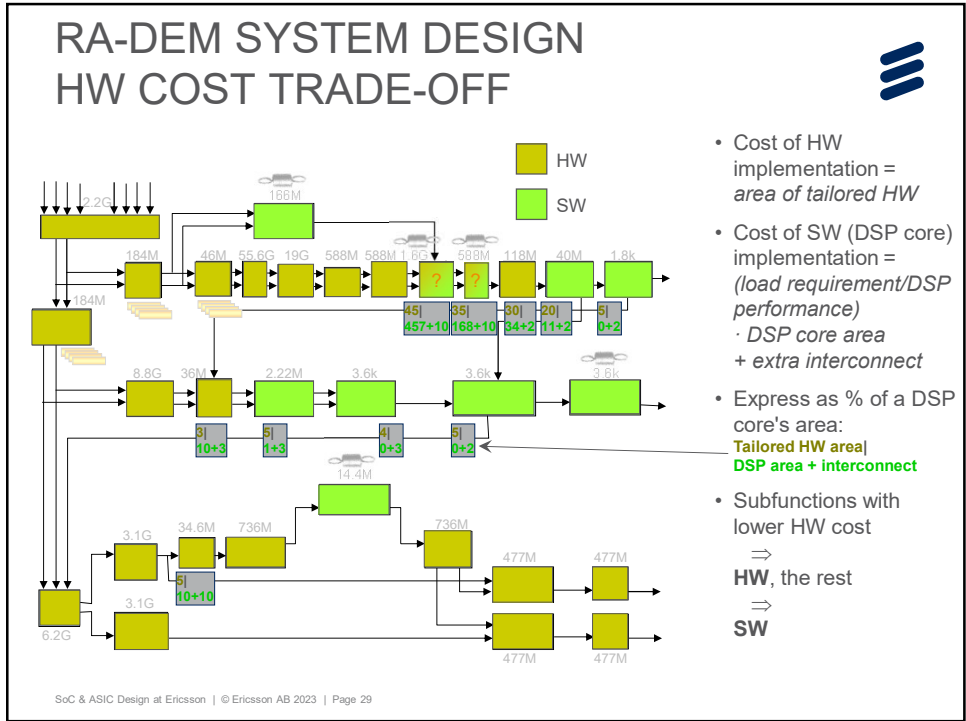
26



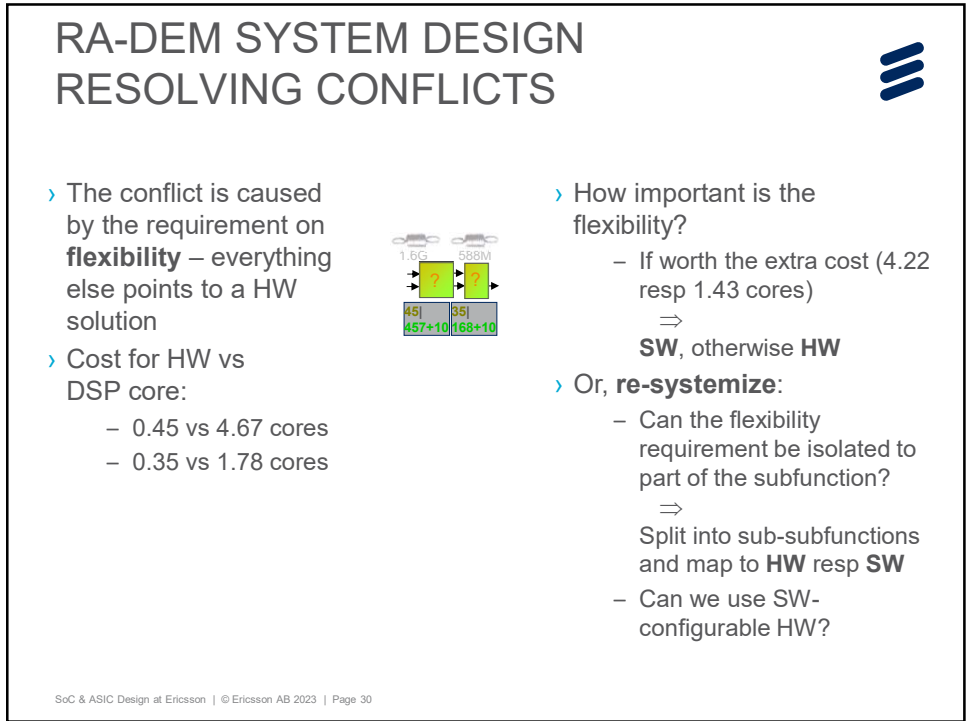
27



28



29

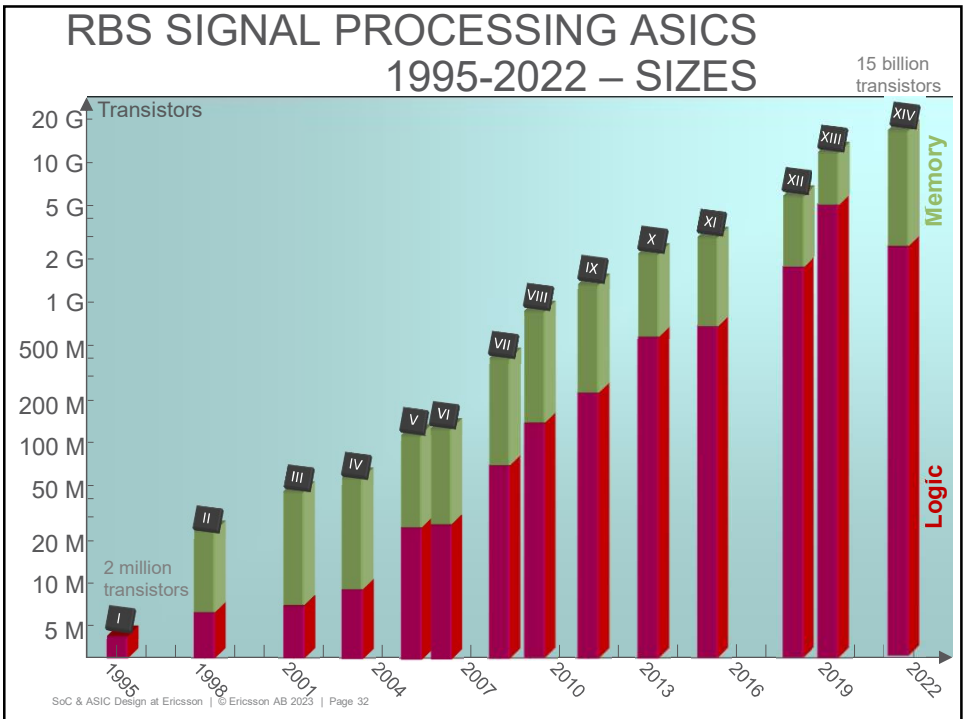


30



Design Challenges for SoCs

31



32

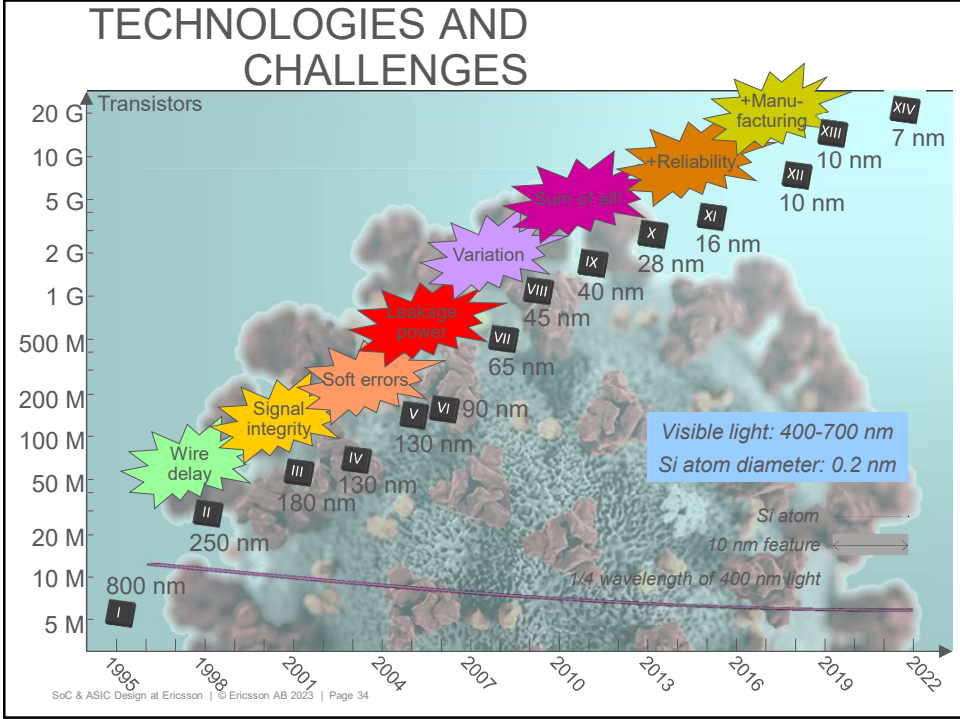
MULTI-CHIP MODULES

- › Integrate multiple chips in a single package
 - Denser design
 - Faster interconnect

42 billion transistors

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 33

33



34

WIRE DELAY, CROSSTALK, SOFT ERRORS



- › Wire delay dominates over logic delay at ≤ 350 nm
 - Solution: New delay models for timing analysis, interconnect-driven design
- › Signal integrity, e.g. crosstalk (capacitive coupling between parallel wires) that cause excessive delays and false pulses, at ≤ 180 nm
 - Solution: New design rules (wire spacing) and analysis methods
- › Soft errors (charged particles hit the silicon with enough energy to change the logic state) at ≤ 130 nm
 - Solution: Error correction coding of memory content

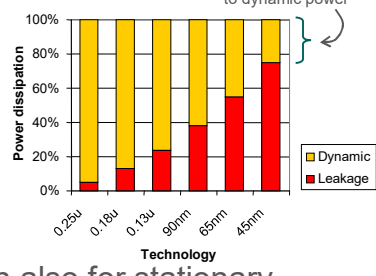
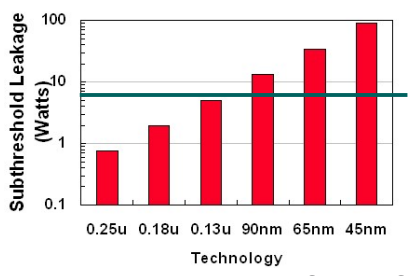
35

LEAKAGE POWER



- › Leakage current (transistor turn-off current) is a major contributor to power dissipation at ≤ 90 nm

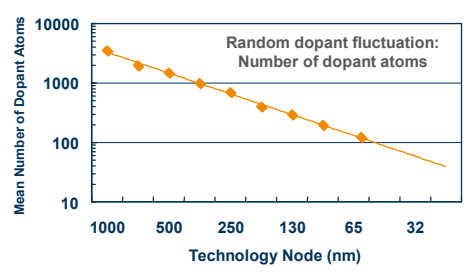
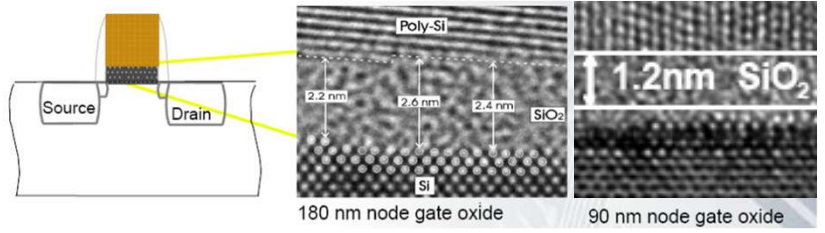
Conventional power reduction methods apply to dynamic power



- › Power is a limiting factor for design also for stationary equipment (i.e., not battery operated)
 - Cooling
 - Energy consumption

36

PROCESS VARIATION AT THE ATOMIC SCALE



- › Parameter variation increases with decreasing geometry:
 - Dopant
 - Instrumentation
 - Mask precision
 - Lithography
 - Variations between fabs and batches
- › Timing characteristics vary significantly from chip to chip

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 37

37

RELIABILITY AT 16 NM AND BELOW



- › Increased risk of device failure after years of operation, due to the small geometry making the device more sensitive to:
 - Electromigration: Can cause opens in wires due to gradual displacement of metal ions caused by high current density
 - Electrostatic discharge: Can cause shorts; increased risk due to small margin between operating voltage and breakdown voltage
 - Thermal heating: High current density leads to local heating which can lead to defects such as cracks (due to heat expansion)
- › Hard to completely eliminate, but effects can be reduced by proper physical design and geometry of transistors and wires



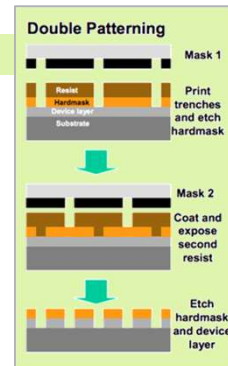
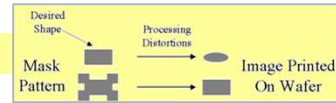
SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 38

38

MANUFACTURING AT 16 NM AND BELOW



- › Silicon structures are manufactured with a lithographic process that uses 193 nm UV light. Creating 10 nm features with a 193 nm "pen" is challenging!
- › **Optical proximity correction (OPC)** compensates for image errors due to diffraction
- › **Double or multiple sets of masks** reduce the demands on individual mask resolution
- › **Extreme UV (EUV) light (11-14 nm)** is now used for some 7 nm technologies.
- › **X-rays (< 1 nm)** may be a future method. It presents huge challenges for the optical equipment used to focus the light.



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 39

39

WITH ALL THESE PROBLEMS – WHY MAKE AN ASIC?






- › 95 % of the functionality on a radio base station receiver board is signal processing
- › Developing an ASIC takes 1-1.5 year and costs several 10 MSEK
- › Buying a high-performance DSP costs 2000 SEK and it is available off the shelf



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 40

40

ASIC VS DSP

> ASICs are:

- Custom made
- Tailored to the application

> Pros:

- High functionality/transistor (tailored HW)
- Few transistor switches/function (optimized HW; **low energy**)
- Low component price (100-1000 SEK)

> Cons:

- Expensive to introduce (high development cost)
- Available after 1-2 year development
- Not upgradable for new functionality

> DSPs are:

- Commodity products
- Used for general applications

> Pros:

- Cheap to introduce
- Available at once (but SW will likely take at least 1 year to develop...)
- Upgradable for new functionality (through SW)


> Cons:

- Less functionality/transistor (general SW)
- Many transistor switches/function (several SW instructions to load, decode, and execute; **high energy**)
- High component price (500-3000 SEK)

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 41

41

ASIC VS DSP PERFORMANCE





> A 100 GOPS ASIC:

- Uses massive parallelism (1000s of parallel threads in concurrent HW)
- Can run at moderate frequency (100 MHz-1 GHz)
 - ⇒ Use low to medium performance Si process (high V_t transistors)
 - ⇒ **Low leakage current**
 - ⇒ **Low to medium power dissipation (5-50 W)**

> A 100 GOPS DSP:

- Uses moderate parallelism (pipelining, co-processors)
- Must run at high frequency (2-20 GHz)
 - ⇒ Use high performance Si process (low V_t transistors)
 - ⇒ High leakage current
 - ⇒ **High power dissipation (100-1000 W)**

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 42


42




SoC/ASIC Design Flow

43

FROM IDEA \Rightarrow COMPONENT ...

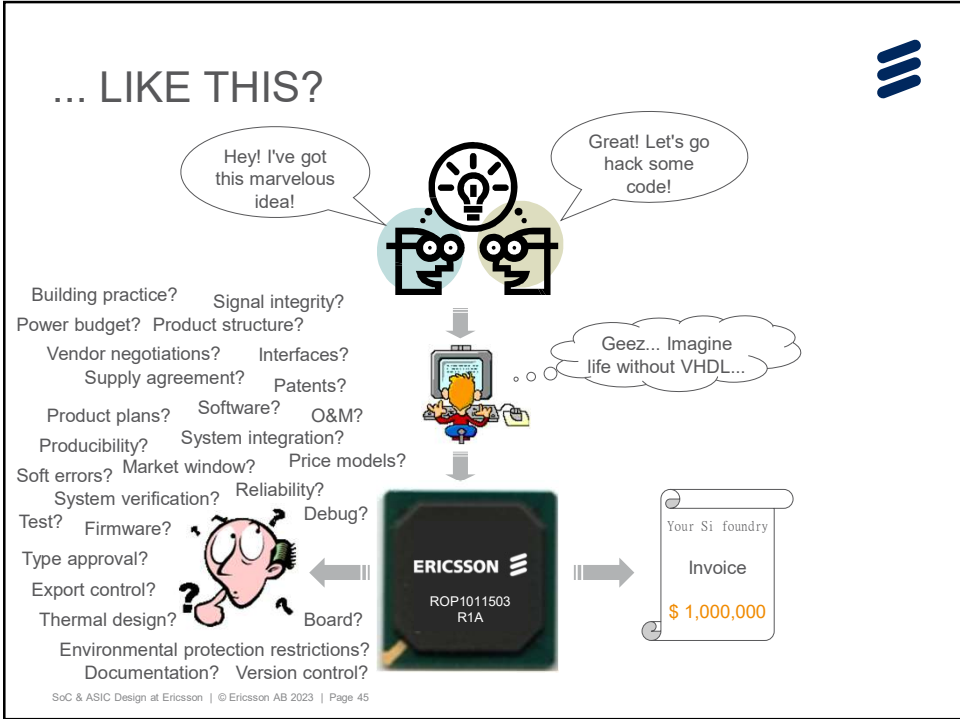


ERICSSON 

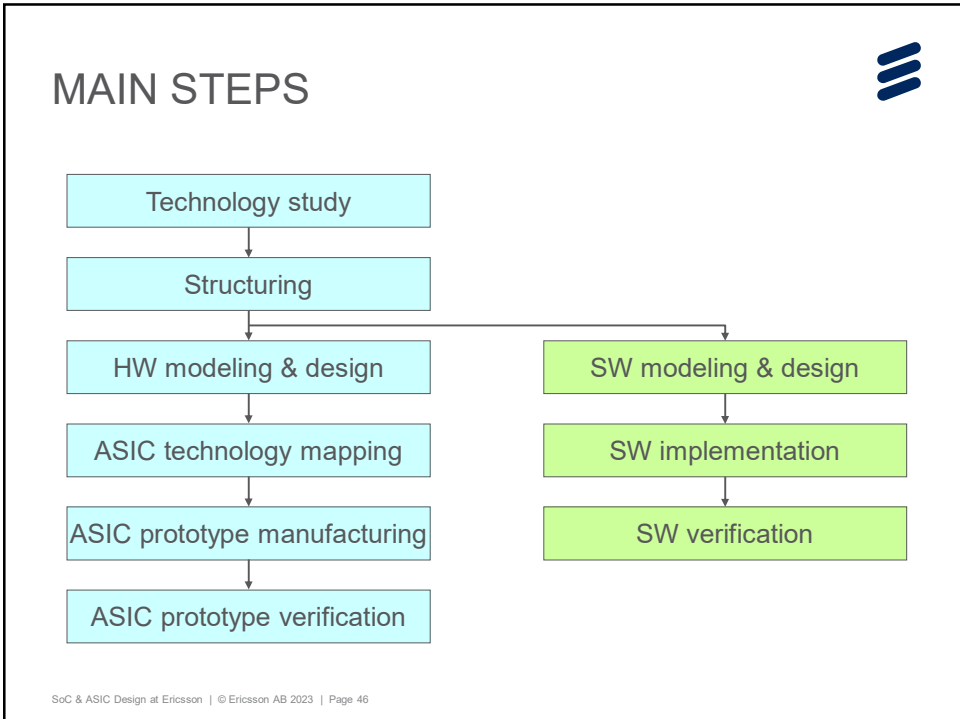
ROP1011503
R1A

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 44

44



45

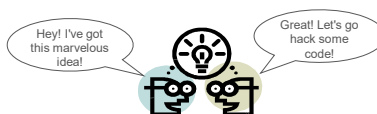


46

TECHNOLOGY STUDY



- › Analyze functional and performance requirements
- › Analyze technological possibilities
- › Investigate different technical solutions, consider:
 - standards
 - production costs
 - life cycle costs
 - reliability
 - environment
 - legal directives
- › Create a high level description of the system architecture (text and graphics)
- › Provide a decision base for project launch decision



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 47

47

STRUCTURING



- › Partition into HW and SW
- › Define HW/SW interface
- › Partition into functional blocks
- › Create behavioral models to evaluate critical functionality (Matlab, C, VHDL)
- › Select IPs, IOs and memories, consider re-use of previous designs
- › Select ASIC technology and silicon foundry
- › Choose package
- › Investigate patent opportunities
- › Specify system testability and diagnostics in production, field, and repair
- › Choose test strategy: scan test, memory test, boundary scan, logic BIST



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 48

48

SYSTEM DESIGN TOOLS AND MODELS

Node subsystem

SoC

- › Signal processing algorithms are designed and analyzed in Matlab
 - Functionality and performance
 - Based on GSM/WCDMA/LTE standards
- › The signal chain is modelled at algorithmic level in C/C++
 - Includes model of the air (radio channel mobile device – base station)
 - Constitutes a reference model for HW and SW implementation
- › Systemization alternatives are evaluated in various ways
 - Spreadsheets for cost, power, capacity
 - High-level architectural models for performance (transaction level models in SystemC)
- › SW Virtual Platforms
 - High-level functional models for SW development (transaction level models in SystemC)

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 49

49

TYPICAL ASIC DESIGN FLOW

IP (macros)

Processor cores
BIST & TAPC
RAMs
PLL

Specification

↓

Coding

↓

Synthesis

↓

Floorplanning

↓

Place & Route

↓

Production *Silicon foundry*

↓

Prototypes

Front-end

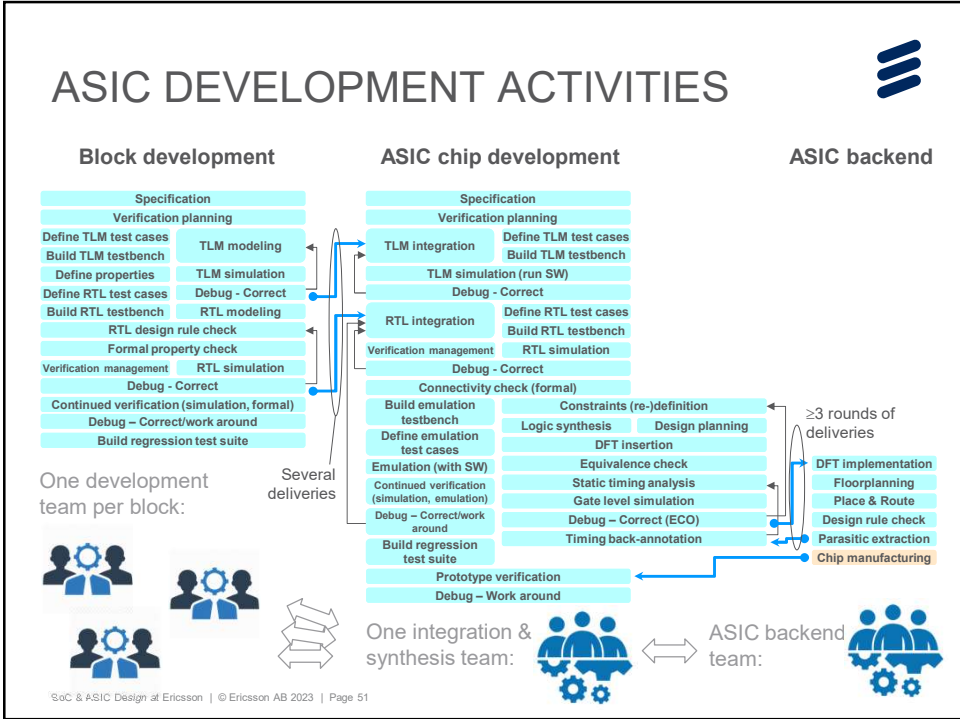
Back-end

Design & verification of

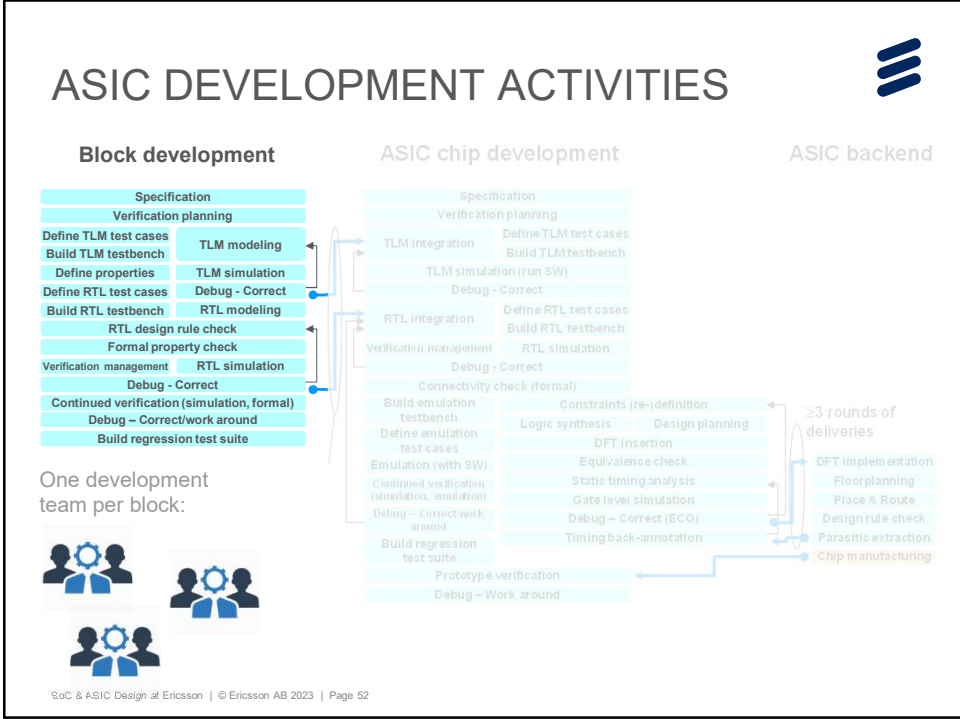
- function
- timing
- power

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 50

50



51



52

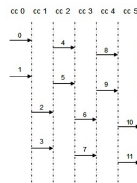
SPECIFICATION



› Each function block has a *Design Specification*

- There are also *Interface Descriptions*, *Verification Specifications*, *Verification Reports* + a number of other documents

› Example of content:



4.2.2.7 ox_dil

Table of Contents

1	Revision History	2
2	Scope	2
3	Short functional description	3
3.1	Overview	3
3.2	Block diagram	4
3.3	Performance	4
4	Design Description	6
4.1	The control logic	6
4.1.1	ox_dil	6
4.1.2	ox_cm_arbitration	6
4.2	The FFT core Bauta	8
4.2.1	Functional description	8
4.2.2	ox_bauta	9
5	Interface Description	20
5.1	External Interface	20
5.1.1	Inputs	21
5.1.2	Outputs	22
5.2	HW/SW interfaces	24
6	Environment Dependent Functions	24
6.1	Reset	24
6.2	Clocking	24
7	References	24
8	Terminology	24

The de-interleaving block `ox_dil` is included in the last stage of the pipeline. The de-interleaving block is rearranging data before it is written to the output scratch memory `ox_omem`. The sample value is calculated as the sum of the first 3 counters. The sum is set to `out_data.sample`. The fourth counter is set to `out_data.index`. Sample and index are then multiplied and scaled with the `dp` value to create the twiddle factor.

53

VERIFICATION PLANNING



Define how to reach verification goals within available time and resources:

1. Analyze the design requirements to identify **features**
2. Analyze the features' characteristics: typical and extreme data/conditions
3. Set verification goals
 - Include all types of **coverage**
4. Design tests to achieve verification goals
5. Plan resources for verification (personnel, computers, licenses)
6. Commence verification

54

VERIFICATION FEATURES AND COVERAGE



- › Features of a design
 - Operations to perform
 - Data to process
 - Protocols to follow
- › Verification goals
 - Cover x % of typical cases
 - Cover y % of corner cases
 - Cover z % of interactions between features
- › Verification complete when cover goals reached (and bugs attended to)

Three different verification features

For example, functional coverage might track whether the verification process has:

- Filled and emptied every FIFO in the design
- Transmitted all packet types across a particular channel of the design
- Transmitted packets across all channels in the design
- Transmitted all packet types across all channels (cross-coverage)

Interacting features

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 55

55

TRANSACTION LEVEL MODELING (TLM)



- › Model *functionality* at a high level
- › Communication modeled as *transactions*
- › Use SystemC and TLM-2 standard
 - Timing can be added
- › Simulates fast (10-1000 x faster than RTL)
- › Example SystemC code:
 - Communication modeled as function calls

```
struct sender: sc_module {
    sc_out<pkt> pkt_out;
    sc_in<sc_int<4> > source_id;
    sc_in_clk CLK;
    SC_CTOR(sender)
    { SC_CTHREAD(entry, CLK.pos()); }
    void entry();
};

void sender::entry() {
    pkt pkt_data;
    while(true) {
        // Data generation code ...
        pkt_out.write(pkt_data);
        wait();
    }
}
```

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 56

56

REGISTER TRANSFER LEVEL (RTL) MODELING



- Model functionality and implementation at HW level
- Communication modeled as signaling
- Use VHDL, Verilog, SystemVerilog
 - Clock-based timing
- Example VHDL code (excerpts) for a serial interface:

Signal assignments:

```

romb_read_req <= not sr_data(65) when req_trigger_d1 = '1' else '0';
romb_write_req <= sr_data(65) when req_trigger_d1 = '1' and sr_data(64) = '0' else '0';
romb_address <= sr_data(63 downto 32) when req_trigger_d1 = '1' else (others => '0');
romb_wmask_req <= sr_data(64) when req_trigger_d1 = '1' and sr_data(65) = '1' else '0';
romb_wmask <= sr_data(31 downto 16) when req_trigger_d1 = '1' and sr_data(65) = '1' else (others => '0');
romb_wdata <= sr_data(15 downto 0) when req_trigger_d1 = '1' and sr_data(65) = '1' else (others => '0');
    
```

Entity with ports:

```

entity serdes_rif is
port (
    clk : in std_logic;
    reset_n : in std_logic;
    cif_has_stdb : in std_logic;
    cif_has_data : in std_logic_vector(cif_di_width_c_dwn);
    cif_has_ack_stdb : in std_logic;
    cif_has_data_out : out std_logic_vector(cif_do_width_c_dwn);
    cif_has_req : out std_logic;
    cif_has_halt : out std_logic;
    rc_rstb : in std_logic;
    rc_reset_n : in std_logic;
    romb_address : out std_logic_vector(31 downto 0);
    romb_wdata : out std_logic_vector(15 downto 0);
    romb_wmask_req : out std_logic;
    romb_wmask : out std_logic;
    romb_read_req : out std_logic;
    romb_ack : in std_logic;
    romb_result : in std_logic;
    romb_rdata : in std_logic_vector(15 downto 0);
);
end serdes_rif;
    
```

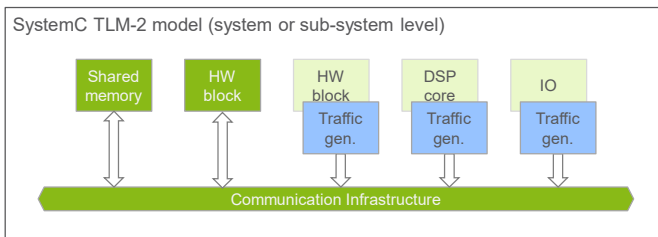
FSM controller (part):

```

rc_ctrl_fsm : process (state_rs, sr_data(65),
sr_data(64), req_trigger_d1, romb_ack,
romb_result, rc_domain_clear_s)
begin -- process rc_ctrl_fsm
nextstate_rs <= state_rs;
read_data_valid <= '0';
case state_rs is
when idle_w =>
if rc_domain_clear_s = '0' then
if req_trigger_d1 = '1' then
if sr_data(65) = '1' then
nextstate_rs <= wait_acknowledge_w_r;
else
nextstate_rs <= wait_acknowledge_r_w;
end if;
end if;
when wait_acknowledge_w_r =>
if rc_domain_clear_s = '0' then
if romb_ack = '1' then
read_data_valid <= '1';
nextstate_rs <= idle_w;
end if;
else
nextstate_rs <= idle_w;
end if;
when wait_acknowledge_r_w =>
if rc_domain_clear_s = '0' then
if romb_ack = '1' then
nextstate_rs <= wait_read_data_w;
end if;
else
nextstate_rs <= idle_w;
end if;
when wait_read_data_w =>
if rc_domain_clear_s = '0' then
if romb_result = '1' then
read_data_valid <= '1';
nextstate_rs <= idle_w;
end if;
else
nextstate_rs <= idle_w;
end if;
when others => nextstate_rs <= idle_w;
end case;
end process rc_ctrl_fsm;
    
```

+ about 400 additional code lines of declarations, synchronization processes, meta-stability handling, etc

SYSTEM PERFORMANCE ANALYSIS



CCI* information exchange:
 Registers
 Internal state
 Transaction logging
 etc

*CCI = Configuration, Control & Inspection

Latency Contention Throughput Bandwidth
 Utilization (time, memory) Transaction tracing

DEFINE PROPERTIES

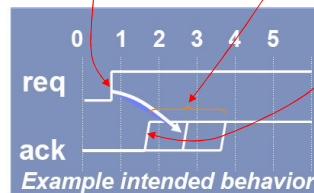
› Properties are used for three purposes:

- **Assertions:**
Define conditional expected events and sequences (what you want to test)
- **Assumptions:**
Constraints (what states the design can have)
- **Covers:**
Check that certain states are reached (can be prerequisites for tests)

Example **assertion**: Check that a bus request is followed by an acknowledge 1-3 cycles after the request

SystemVerilog code

```
property req_ack;
  @(posedge clk) $rose(req) |-> ##[1:3] $rose(ack);
endproperty
req_ack_asrt: assert property(req_ack);
```



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 59

59

BUILDING THE RTL TEST BENCH

- › Create test benches to verify functional behavior and RTL implementation
- › Feature based verification
- › Usually built with SystemVerilog
- › Stimuli generation
 - Specify data format rather than exact content
 - Random data generators with constraints
- › Checkers verify stimuli/response relationship
 - Basis for functional coverage analysis

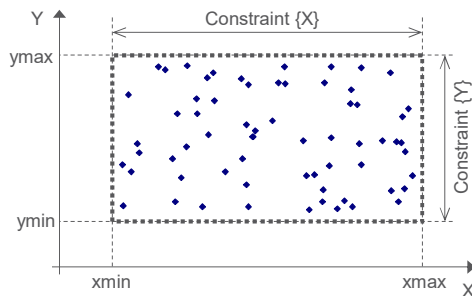
SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 60

60

BUILDING THE RTL TEST BENCH CONSTRAINED RANDOM TESTING



- › Instead of specifying which variable **values** to test, specify constraints for the interesting **ranges**
- › Auto generate **random** values within the constraints



SystemVerilog code:

```
class CRT_example;
  rand shortint X;
  rand shortint Y;

  constraint x1
    {X inside {[xmin:xmax]};}
  constraint y1
    {Y inside {[ymin:ymax]};}
endclass: CRT_example
```

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 61

61

RTL DESIGN RULE CHECK



- › Designability = Ability to design
- › Rule check: Find design patterns that may cause problems in the design flow
 - Static check of HDL code
- › Typically several hundred rules
- › Rule examples:
 - Avoid latches (incomplete HDL conditions may cause unintentional latches)
 - Memory inputs should be observable (for test)
 - Use specified identifier suffixes, e.g. "_n" for active low
 - One file per HDL design unit (entity/module etc)
 - Use fixed indentation
 - Use IEEE Numeric_std packages

ID	Rule	Alias	Severity	File	Line	Wt	Message
[13]	W06		Warning	/u2/Work/synth_1018_issue/Synth_1018/Multidim.vhd	18	2	Not all the elements of signal 'rai_alpha' a
***** Template-New_SIL/Maturity Level_2/ST_Ericsson_design_rules/eng_pam *****							
[1]	Re_2_2_2_3-1	Recommendation		/u2/Work/synth_1018_issue/Synth_1018/Multidim.vhd	10	2	Signal beta does not follow man
[2]	Re_2_2_2_3-1	Recommendation		/u2/Work/synth_1018_issue/Synth_1018/Multidim.vhd	13	2	Signal alpha does not follow re
[3]	Re_2_2_2_3-1	Recommendation		/u2/Work/synth_1018_issue/Synth_1018/Multidim.vhd	14	2	Signal wea_beta does not follow
[4]	Re_2_2_2_3-1	Recommendation		/u2/Work/synth_1018_issue/Synth_1018/Multidim.vhd	18	2	Signal rai_alpha does not follo
[5]	Re_2_2_2_3	Rule		/u2/Work/synth_1018_issue/Synth_1018/Multidim.vhd	1	2	File /u2/Work/synth_1018_issue/
***** Template-New_SIL/Maturity Level_2/ST_Ericsson_design_rules/eng_opernore *****							
[14]	TypeComment	(OpenOFF 5.2.5.1)	Rule	/u2/Work/synth_1018_issue/Synth_1018/Multidim.vhd	11	5	Type definition is not co
[15]	TypeComment	(OpenOFF 5.2.5.3)	Rule	/u2/Work/synth_1018_issue/Synth_1018/Multidim.vhd	16	5	Type definition is not co

SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 62

62

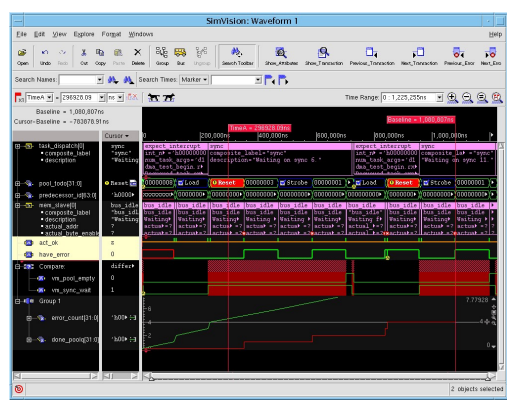
RTL VERIFICATION AND DEBUG



- › First verify assertions by formal property check
 - Mathematical proof that a behavior always/never occurs

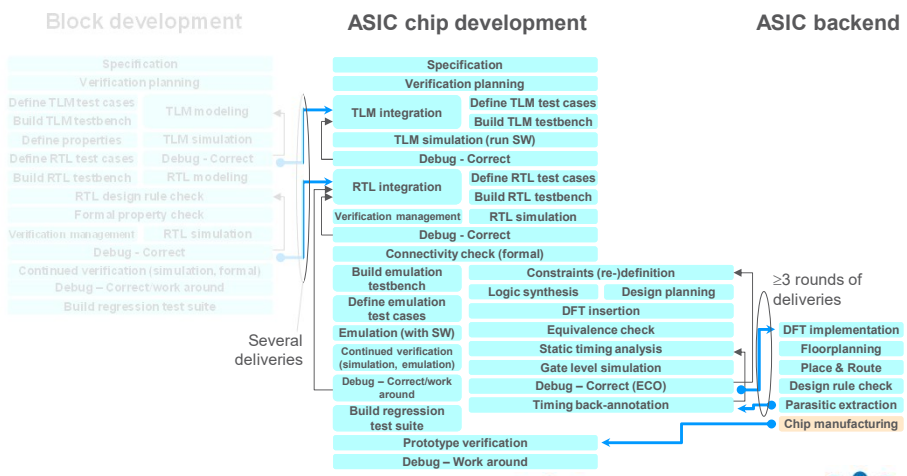
- › Then simulate what's left
 - Mathematical computations
 - Performance

- › Debug is typically waveform based



63

ASIC DEVELOPMENT ACTIVITIES

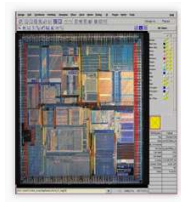
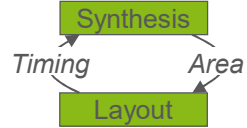
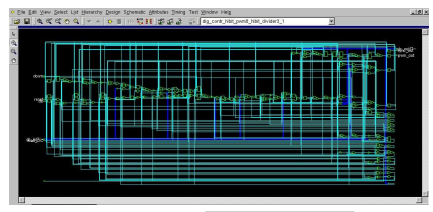


64

LOGIC SYNTHESIS AND DESIGN PLANNING



- › Map RTL to optimized gate level netlist
 - Needs cell library for target technology
- › Work towards goals
 - Typically minimized area or power
- › Find result within constraints
 - Typically timing or power
- › Synthesis result depends on physical layout
 - Layout affects timing constraints
- › Physical layout depends on synthesis
 - Synthesis determines size of blocks
- › Good result requires interaction synthesis – design planning (= preliminary layout)

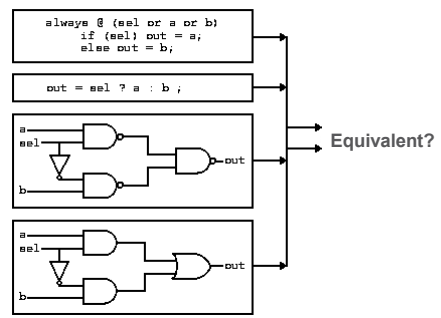


65

EQUIVALENCE CHECK



- › Checks functional equivalence of two designs
 - RTL - gate (before and after synthesis)
 - Gate - gate (before and after netlist modification)
 - (RTL - RTL)
- › Formal methods
 - Reference design logically implies* compared design
 - No test bench, no test vectors
 - Verifies complete design



* *Implication* (\Rightarrow) means that *all* reference functionality is present in the compared design, but the compared design may have *additional* functions not present in the reference design (such as test logic).

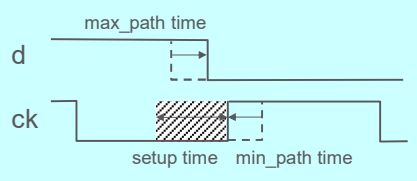
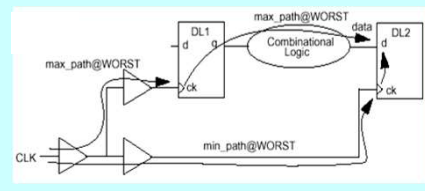
66

STATIC TIMING ANALYSIS



- › Verify timing
- › Gate level
- › Estimated or back-annotated timing from layout
- › Calculate delay along all clock and data paths
 - Check for setup and hold violations, delays, pulse widths
 - Locate critical paths
 - Analysis for worst case, best case, on-chip variation

Timing path for worst case (setup check)

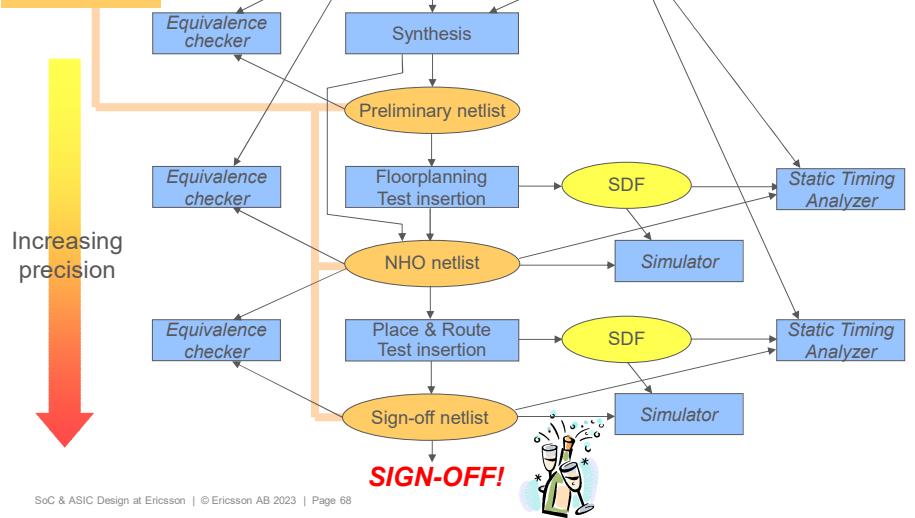


67

BACKEND DESIGN & VERIFICATION



Typically at least 3 deliveries to ASIC backend



68

TOP-LEVEL VERIFICATION



› Four different platforms for the top level:

TLM simulation RTL simulation RTL/GL emulation ASIC prototype

```

void sender::entry() {
    pkt pkt_data;
    while(true)
        // data generation code ...
        pkt_out.write(pkt_data);
        wait();
}

rc_ctrl_fsm : process (state_rs,
    sr_data(is), sr_data(0),
    req_trigger_d1, rmb_ack,
    rmb_result, rc_domain_clear_s)
begin -- process rc_ctrl_fsm
    nextstate_rs <= state_rs;
    read_data_valid <= '0';
    case state_rs is
        when idle_w =>
            if rc_domain_clear_s = '0' then
                if req_trigger_d1 = '1' then
                    if sr_data(is) = '1' then
                        nextstate_rs <=
                            wait_acknowledge_w_w;
                    end if;
                end if;
            end if;
        end case;
    end process;
end rc_ctrl_fsm;
    
```



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 69

EMULATION



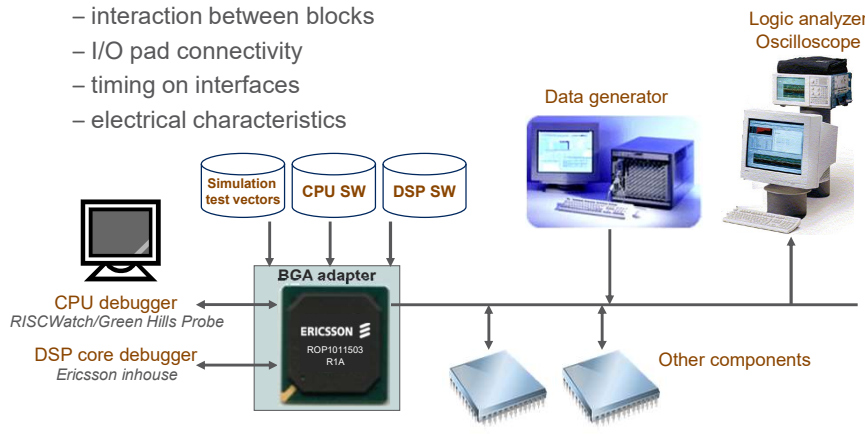
- › RTL or gate level verification
- › **Acceleration:** Replace design in simulation (keep simulated test bench)
- › **In Circuit Emulation:** Replace circuit on board with model in emulator; connect emulator to board
- › Emulates design with configurable hardware arrays (FPGAs or custom developed)
 - Extremely fast, ~1-10 M cycles/s
- › Requires long test cases for efficient use
 - Random or real data streams, application SW



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 70

PROTOTYPE VERIFICATION

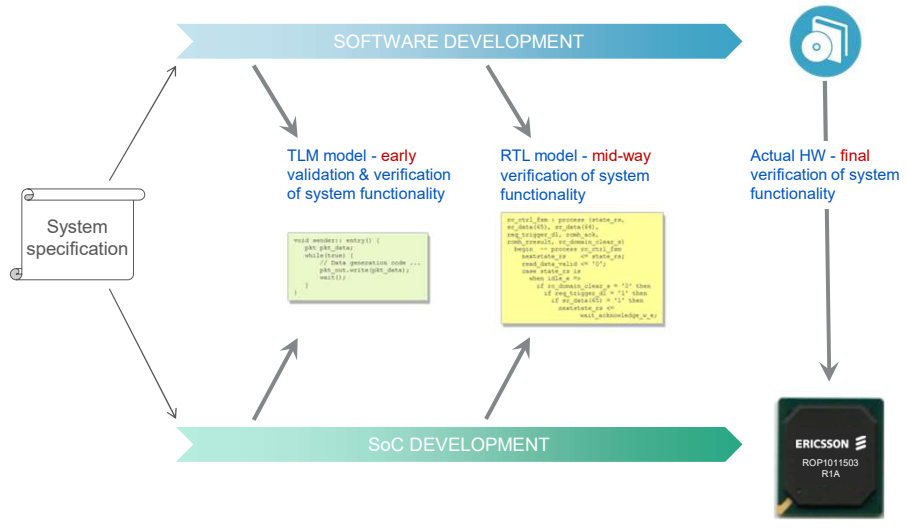
- › Establish that ASIC prototypes meet specifications:
 - functionality at full speed
 - interaction between blocks
 - I/O pad connectivity
 - timing on interfaces
 - electrical characteristics
- › Tested in lab environment:
 - production board or test board



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 71

71

HW/SW/SYSTEM VERIFICATION

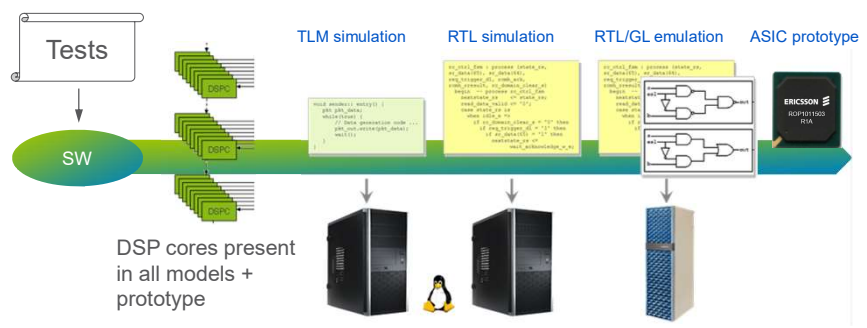


SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 72

72

SW-DRIVEN VERIFICATION

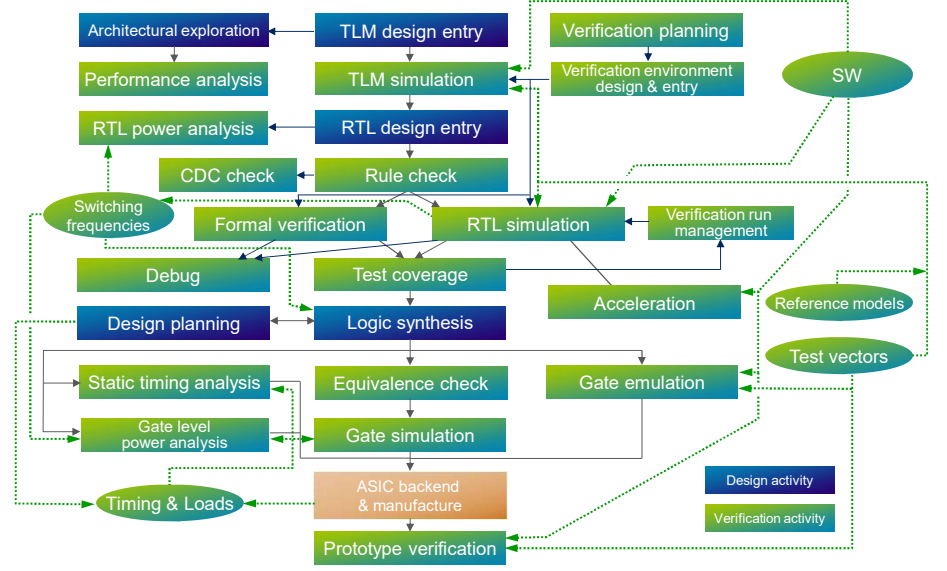
- › Use the built-in DSP cores to run test programs on top level
- › Provides a platform-independent test environment
- › The same tests can be run on all four top-level verification platforms



SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 73

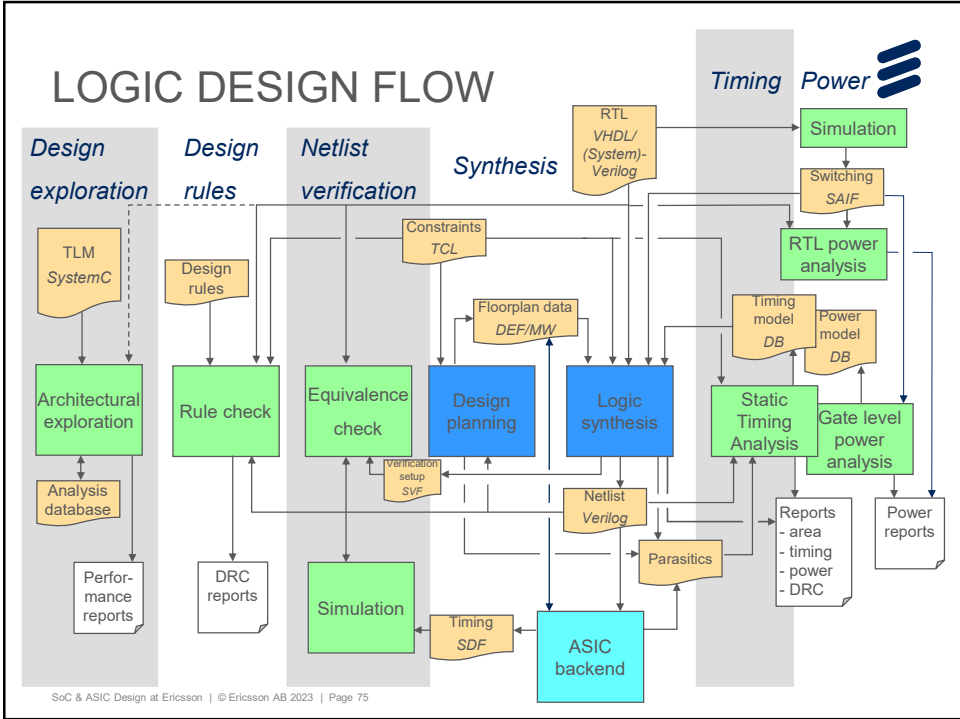
73

ASIC DESIGN FLOW – OVERVIEW

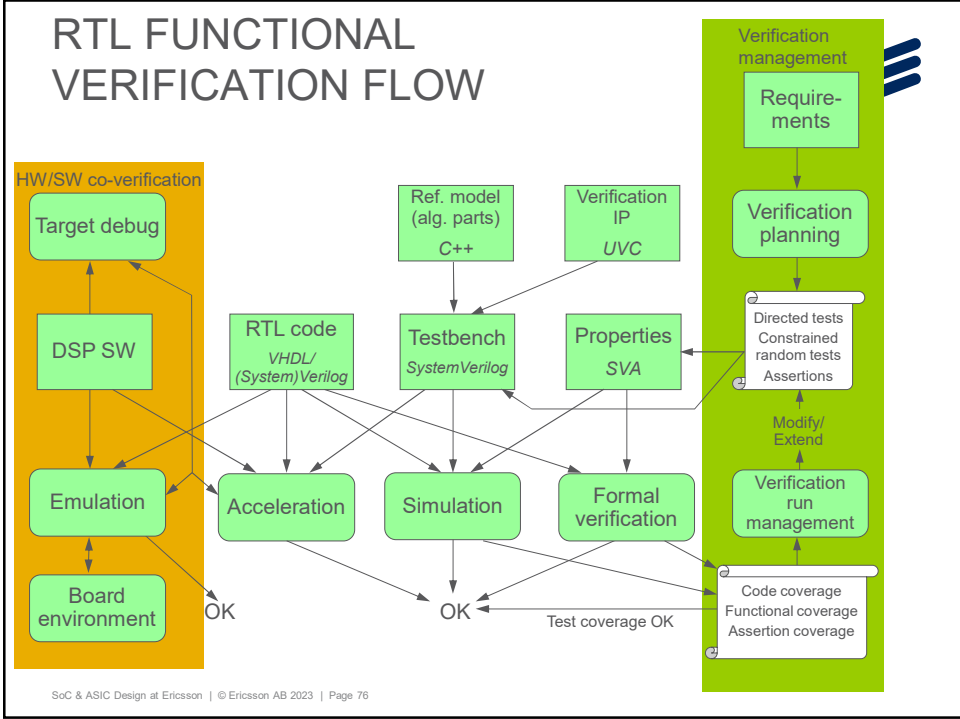


SoC & ASIC Design at Ericsson | © Ericsson AB 2023 | Page 74

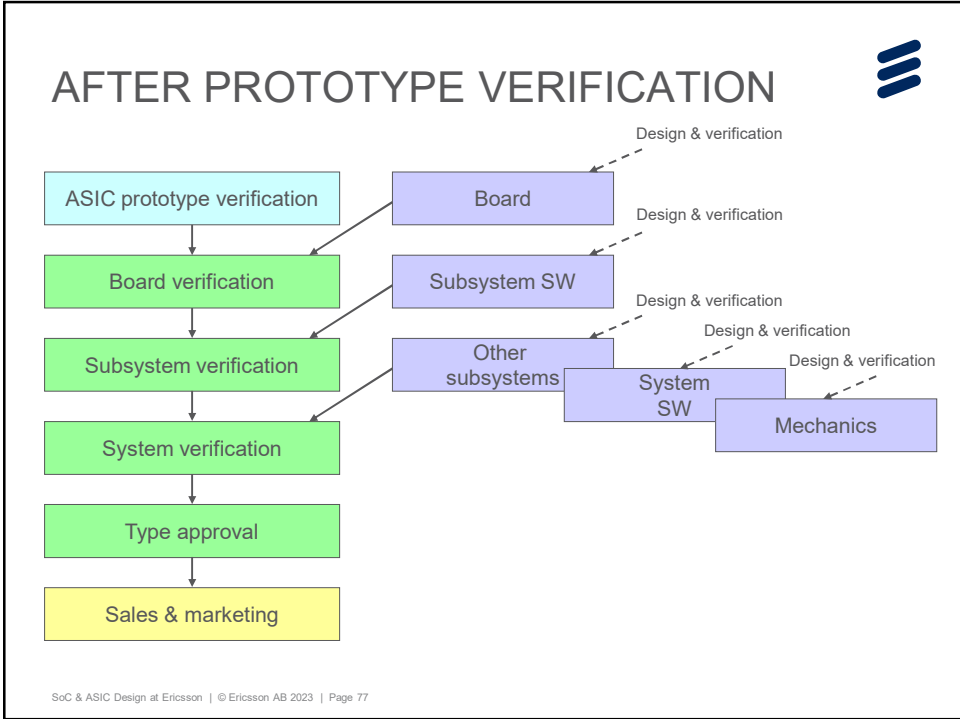
74



75



76



77

WANT TO PARTICIPATE?

- › Do you want to:
 - Contribute to the world-wide expansion of mobile communications?
 - Use the absolutely latest microelectronics technology?
 - Be part of a world class ASIC design team?
 - Work in a global environment with daily international contacts?
 - Work at the world leading telecommunications company?
- › Look for job opportunities and thesis works at jobs.ericsson.com !

78



79