Introduction to the Methods Course

Per Larsson-Edefors

Dept. of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden

perla@chalmers.se

Motivation

- Billions of transistors on single chips.
  - First reaction: “Great! More performance and functionality!”
  - ... but how do we make use of such a complex platform?
- Electronic system designers are forced to make use of *design automation tools* to manage *design complexity* and meet, e.g., strict timing, power and time-to-market budgets.
- To apply the right tools in the right context and in the right sequence has become a *methodological challenge* that rivals traditional design challenges intrinsic in logic and circuit design.

Course Learning Outcomes

1. describe the algorithmic principles of a number of important EDA concepts, such as behavioral and logic synthesis, logic simulation, static timing analysis, timing closure and power dissipation analysis
2. describe contemporary EDA design flows and their fundamental weaknesses and strengths
3. apply Linux-based EDA tools, including simple shell scripts, for design and verification of digital electronic systems
4. perform timing-driven synthesis and power dissipation analysis for digital circuits
5. critically and systematically integrate knowledge, to model, simulate, and evaluate features of digital ASIC design flows
6. write a technical report containing introduction, background, method, results and conclusion, with proper commentary of data and reference handling

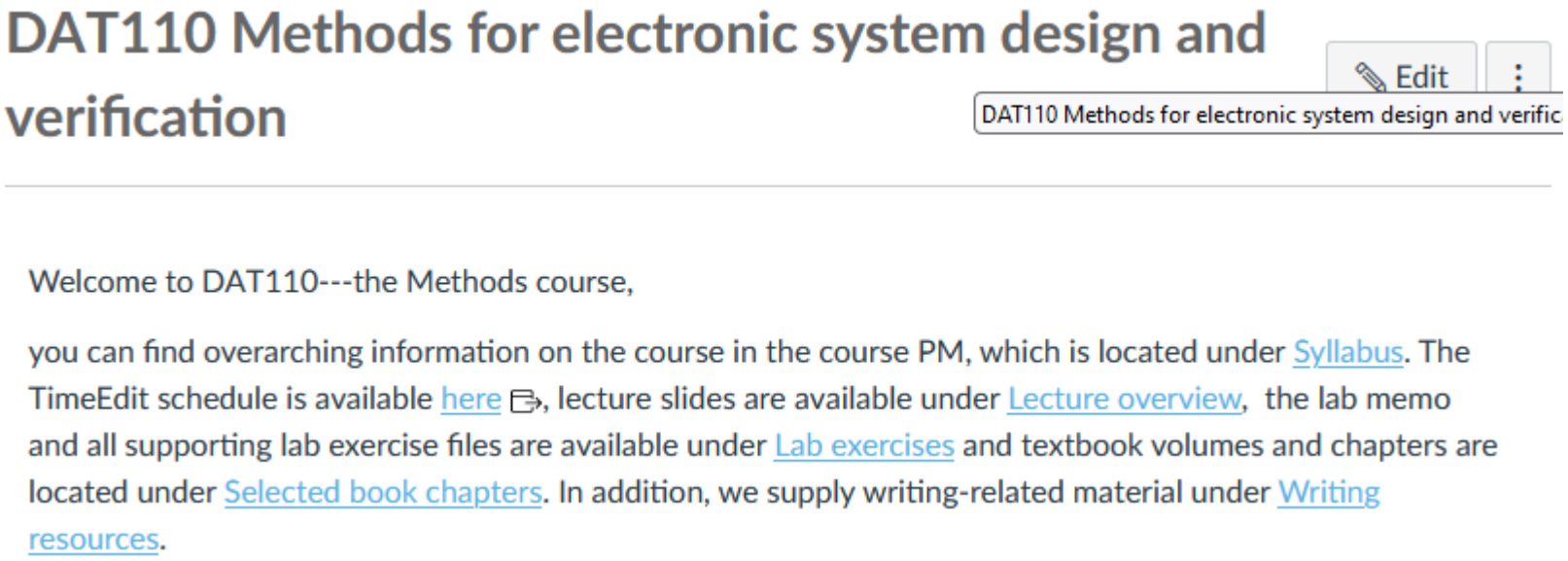
EDA What? Some Terms …

- EDA = Electronic Design Automation (*cf* CAD).
- RTL = Register Transfer Level.
- P&R = Placement and Routing / Place and Route.
- Design flow = EDA tools arranged in sequence defined by scripts.
- Synthesis = refinement (often a reference to automated refinement).
- Verification = check that requirements are fulfilled.
- Heuristic = a problem-solving algorithm with unpredictable outcome.

Teachers and Canvas

- Instructor:
  - Per Larsson-Edefors.
- Technical writing lecturer:
  - Anne Hsu Nilsson (Gothenburg University).

- Canvas site: https://chalmers.instructure.com/courses/20977

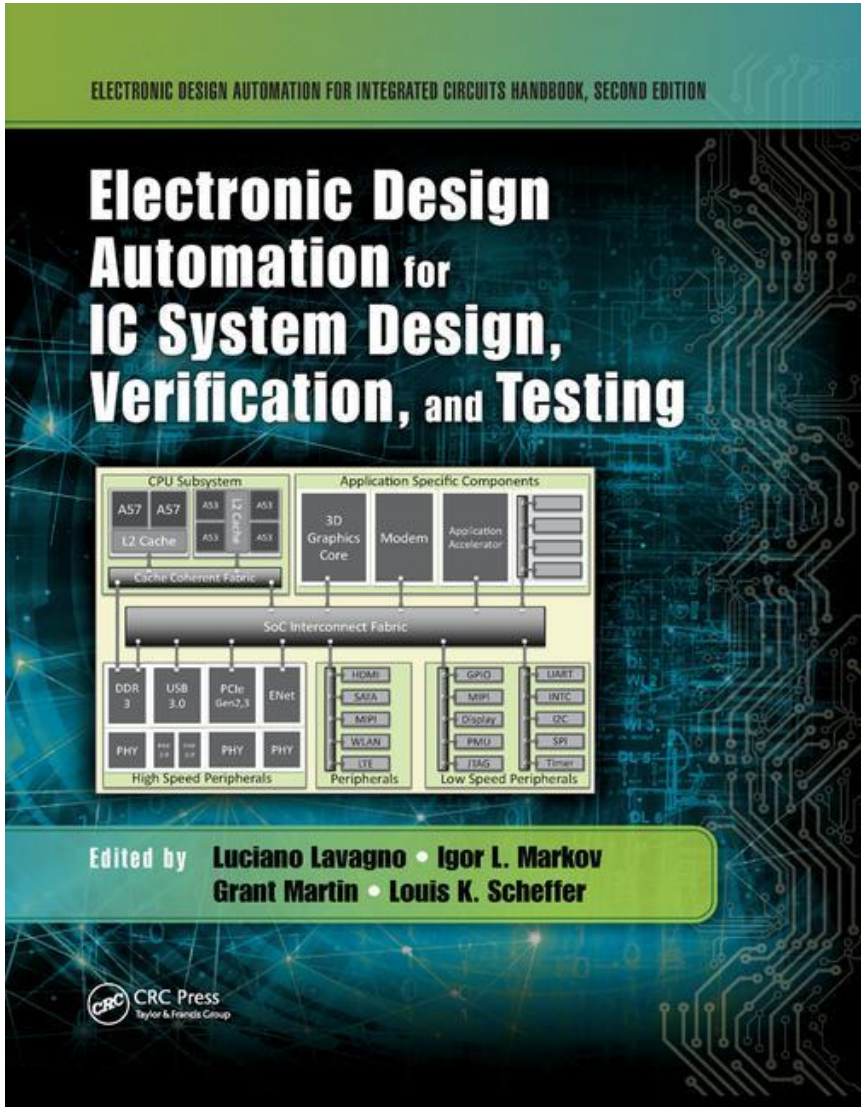


Course Organization

- Lectures.
  - Design and verification context of advanced electronic systems.
  - Technical writing.
- Labs.
  - Comprehensive hands-on training.
  - Best design practice using state-of-the-art EDA tools.
  - Emphasis on properties like timing and power/energy.
  - Technical writing with external sources.

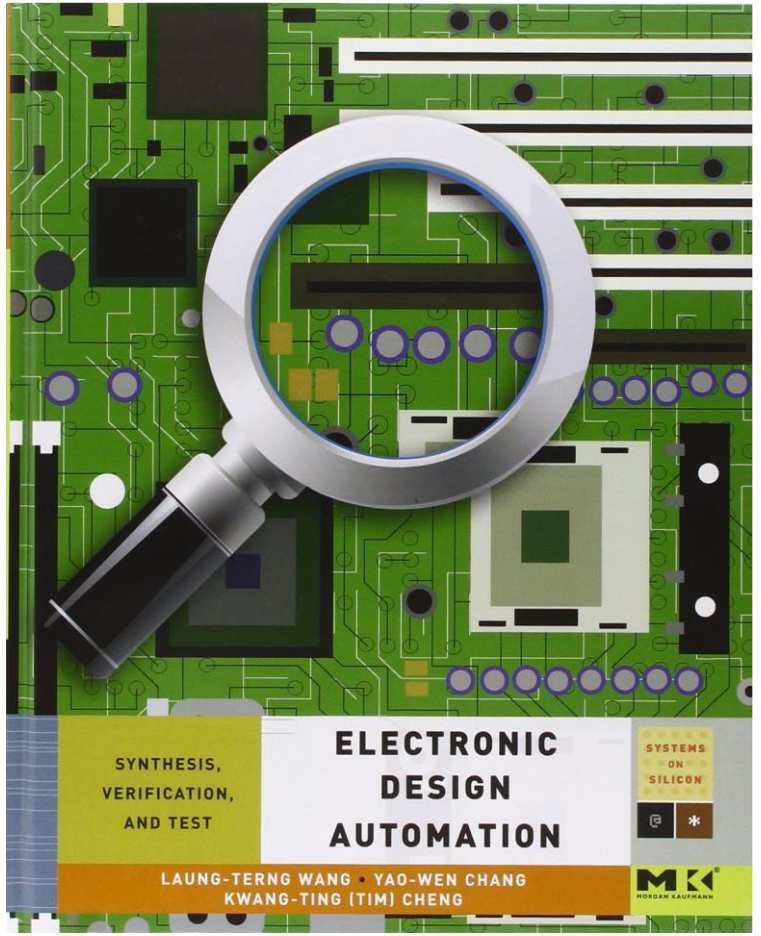
Main Textbook from 2016

- EDA for IC System Design, Verification, and Testing (Vol 1)
- EDA for IC Implementation, Circuit Design, and Process Technology (Vol 2)
- Both volumes downloadable from Taylor and Francis.
  - See instructions in Canvas under *Selected book chapters*.

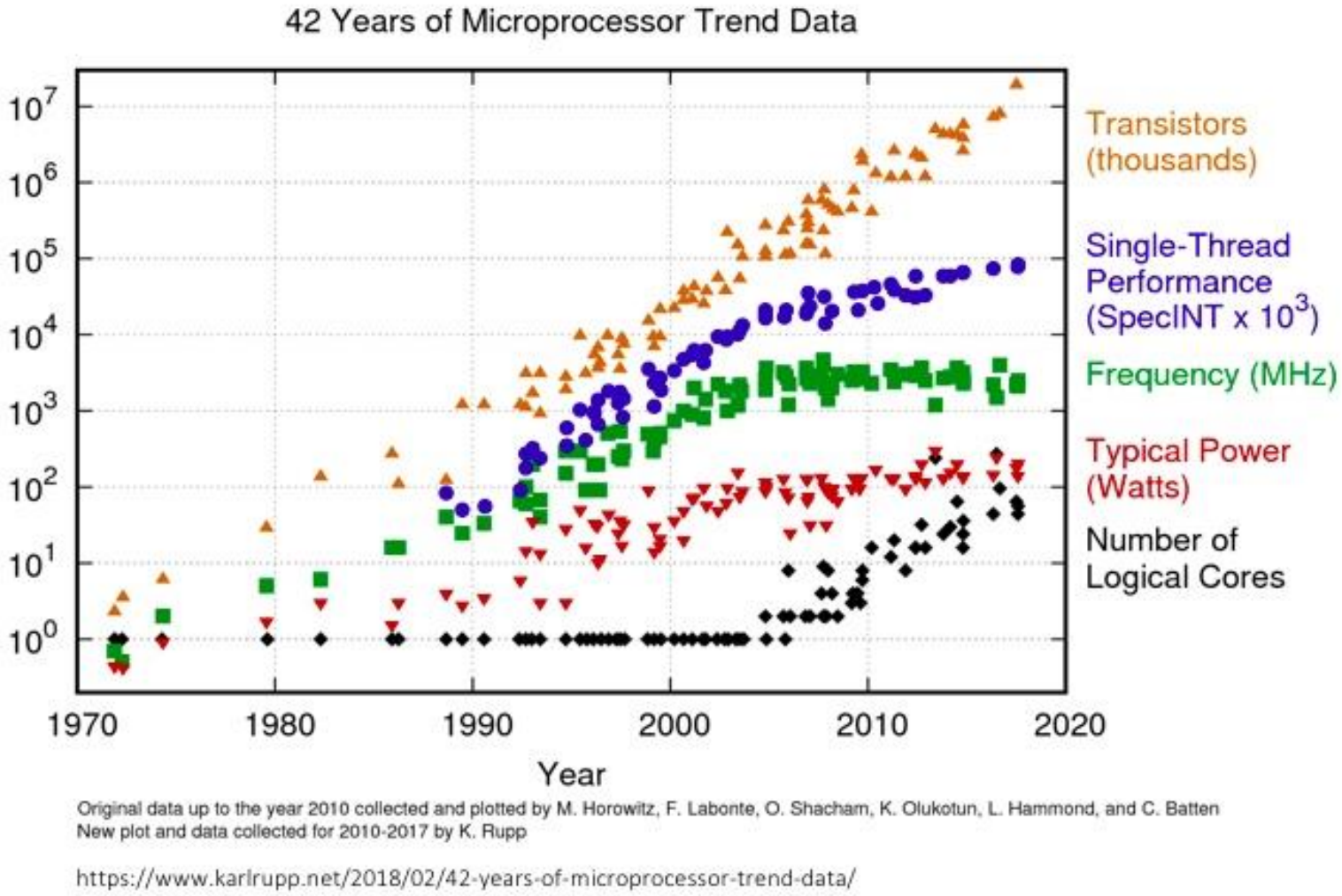


Select Chapters

- Additional textbook resource.
  - Use ScienceDirect database; use link from Chalmers library.
  - Relevant chapters available in Canvas under *Selected book chapters*.



VLSI Trends … “Moore’s Law”



Billions of MOSFETs: Use Abstractions!

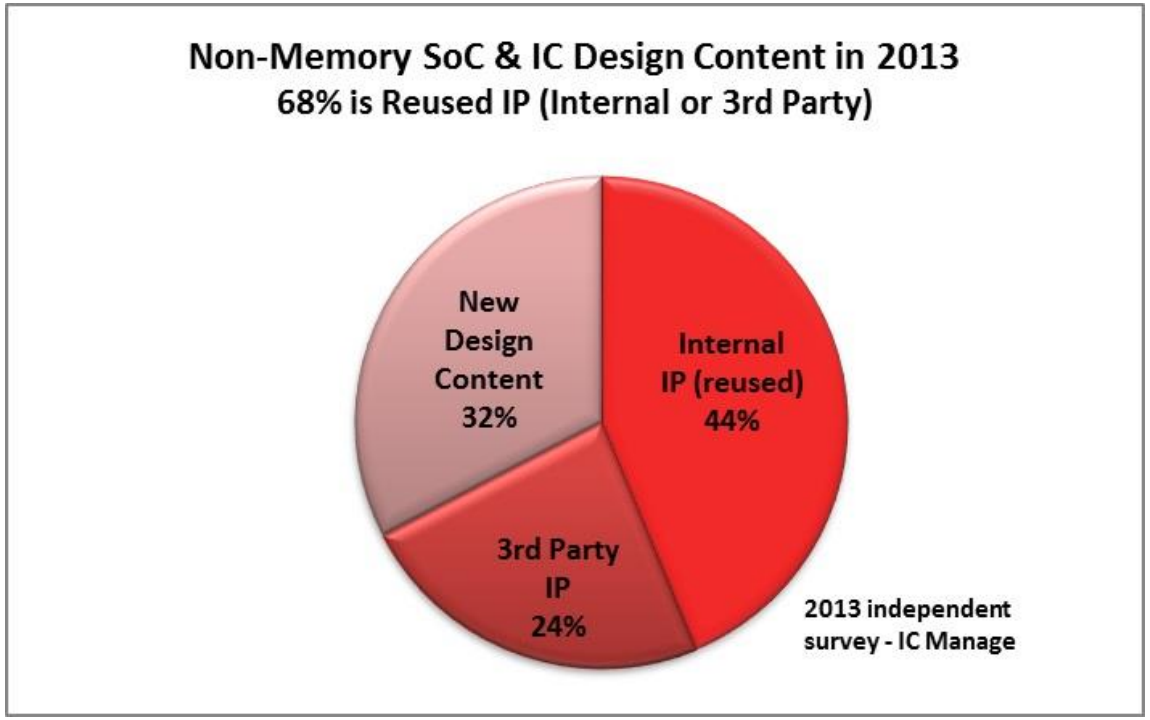
- Signals are 0 or 1.
- The overall behavior is synchronous.
- Gates are characterized as stand-alone elements (cell library).
- Hardware description languages (VHDL, Verilog).
- Design hierarchy:
  - hides details and improves productivity.
  - encourages use of modules and “black boxes”, which is essential for verification.
  - lab preparation: draw system block diagram

Design Flow: The Order of Tasks Matters

- Bottom up:
  - first, standard cells developed, next, cell libraries for synthesis.
  - slow approach if entirely sequential,
    but important for new implementations.
- Top down:
  - e.g., bring an algorithm to silicon.
  - identify and bring in intellectual property (IP) blocks.
  - *timing closure*: what wire assumptions
    can be made in early stages of design?
- EDA tools for chip design: Tools lined up in a specific order,
  a *design flow*, known to give good results.

Bottom-Up Example: IP Blocks

- Trend from 2013 maintained;
  IP reuse makes up 60-70% of block.
- Merging EDA and semiconductor IP strategically important
  - Synopsys made $500M on IP products in 2016.
- Process foundries develop more and more IP and point EDA tools to move up the value semiconductor chain.



Top-Down Example: SW Prototyping

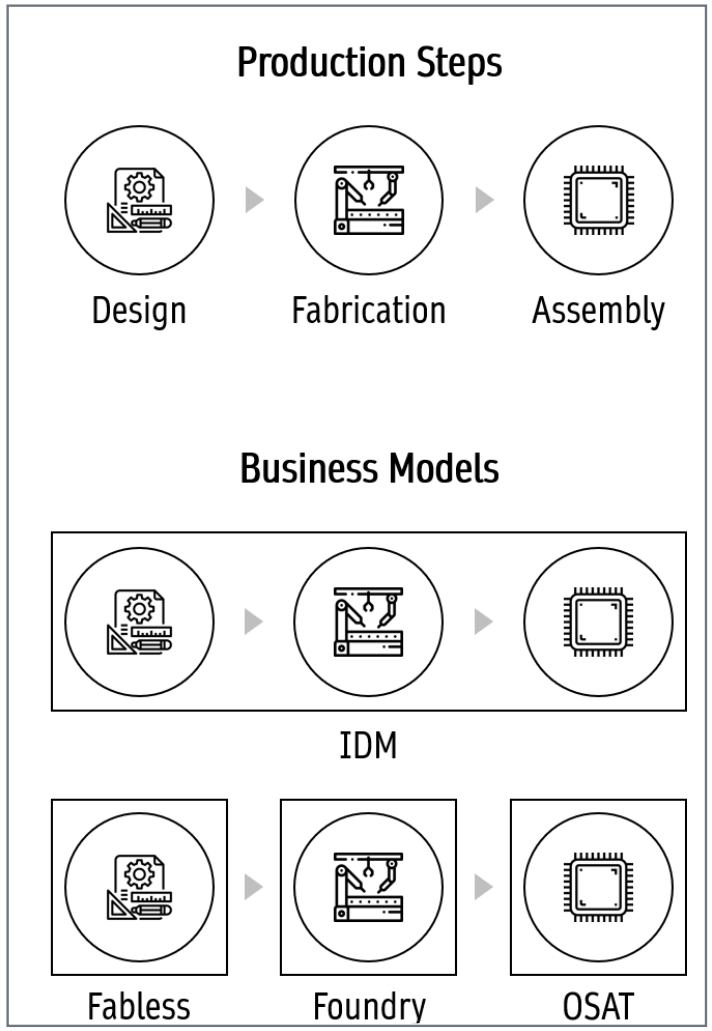
- Start SW development early,
  before the ASIC has been developed.
- Virtual prototypes:
  Implement prototype on FPGAs to
  reduce SW development time.
  - Protium (Cadence).
  - HAPS (Synopsys).
  - Veloce (Siemens Mentor).

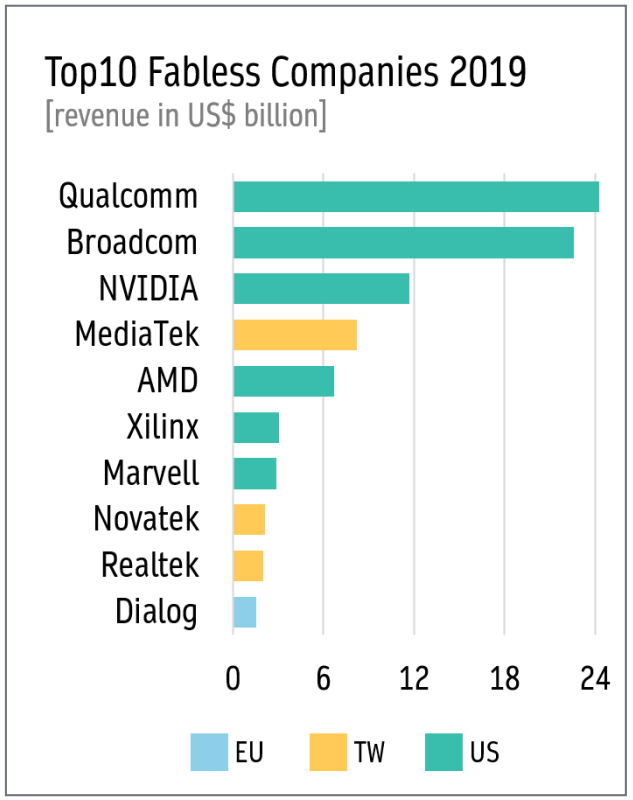
source: Mentor Siemens

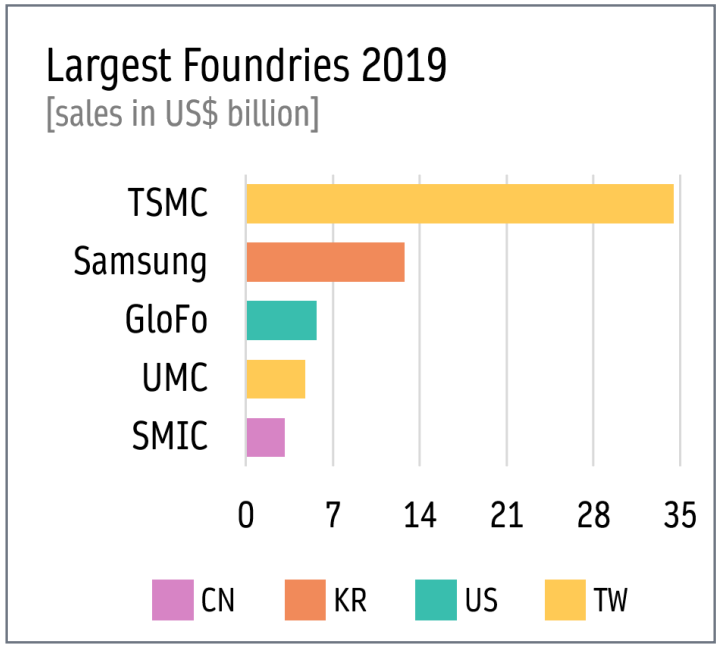
source: Cadence

source: Synopsys

Semiconductor Value Chains

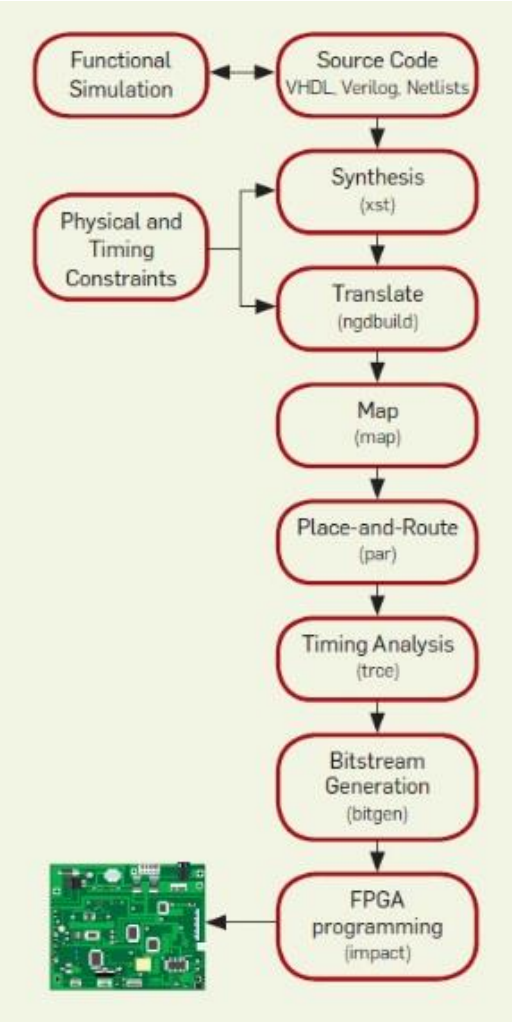


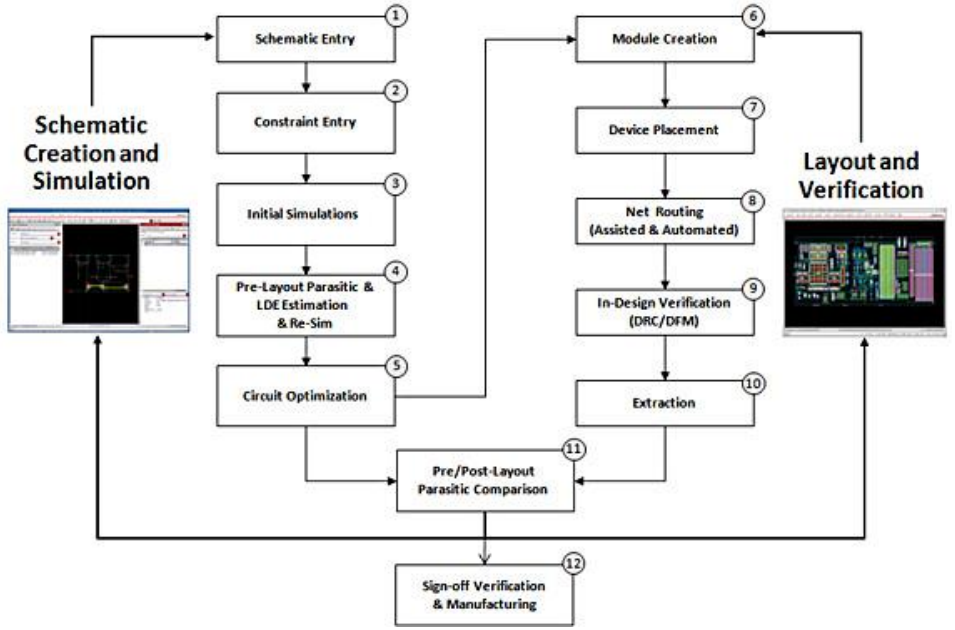




source: "The global semiconductor value chain", Kleinhans and Baisakova, 2020

EDA Design Flows



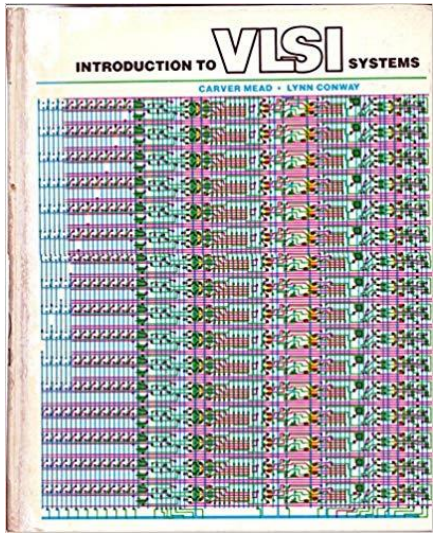


source: Hiroshi Ishikawi, Cadence Community, 2013

source: FPGA Programming for the Masses, by Bacon et al., 2013

Design Methodology for VLSI

- Mead and Conway: seminal book "Introduction to VLSI Systems" in 1979.
- Daisy Systems and Valid Logic Systems (schematic capture, logic simulation) with proprietary HW/SW systems.
- *Mentor Graphics*: Spin off (1981) from Tektronix (Beaverton OR). IDEA 1000 simulation software running on Apollo workstations.
- VLSI Technology Inc.: Early foundry. Cell library design (IP + EDA).



source: https://www.computerhistory.org/

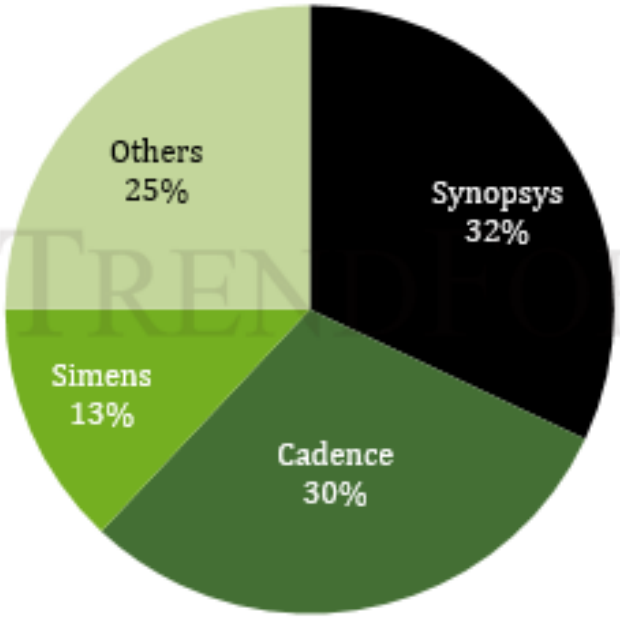
The Rise of Two EDA Giants

- Early EDA businesses had problems caused by poor scalability; dedicated software (assembly) running on rigid hardware.
- Two successful companies offered software only:
  - EDAC (Dracula layout checker) 1982 and SDC (layout design framework) merged in 1988 and became *Cadence*.
    Some key acquisitions:
    Gateway (with Verilog) in 1989. Valid Logic Systems in 1991.
  - Optimal Solutions, a spin off from GE in NC (1986).
    After moving to Mountain View CA 1987, this became *Synopsys*.
    The SOCRATES system was a technology mapper [Lecture 4] and became the starting point of Synopsys Design Compiler.
    Some key acquisitions:
    Epic (1997), Avanti (2002), Magma (2012).

Three Dominant EDA Companies

- Synopsys.
- Cadence.
- Siemens (formerly Mentor Graphics).

Figure 2: Global EDA Software Market Share, 2021



Source: TrendForce, Aug. 2022

EDA Acquisitions

(ESNUG 588 Item 5) ---------- [09/19/19]

Subject: Badru wants PnR users to know Nitro-SoC is alive and well and #3!

CHOICE IS GOOD: A whole bunch of digital PnR users were quite happy to see that AtopTech ... err... correction ... make that "Avatar" was back in the EDA game. There were two groups of users commenting on this:

1. the older earlier ATOP users who were generally happy that their PnR tool of choice could now compete on a technology level instead of on a "who has the sleaziest lawyers level".

2. and a number of potentional *new* Avatar users who love the idea of having a viable 3rd choice in PnR over the CDNS vs. SNPS duopoly -- especially ~~now that MENT Olympus and Nitro SoC are gone.~~

   (01/30/2019 Edit: Mentor says **Olympus** and **Nitro-SoC** are NOT gone.)

   - Avatar/Atop is back; but MENT Nitro-SoC is dead (01/30/19)

From: [Badru Agarwala of Mentor Calypto]

John, John, John...

Oh, the chaos you cause is so fun, my friend. Our sales department has a different view of you, however (but we won't go into that).

Siemens EDA

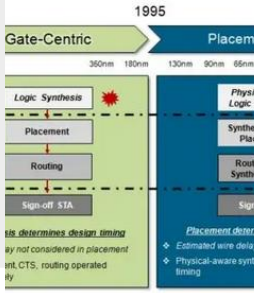
Update on Mentor's Acquisition of Avatar Integrated Systems

by Daniel Nenni on 09-23-2020 at 10:00 am
Categories: Siemens EDA

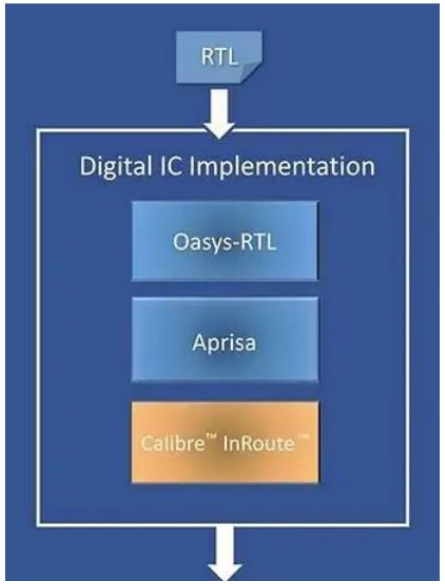
aphics, a Siemens Business, has completed their acquisition of EDA Avatar Integrated Systems. I recently spoke with Joe Sawicki, Executive VP tor IC EDA segment, about the acquisition strategy and IC Design platform ntegration of the Avatar products.

rmerly AToptech) focused on physical implementation tools for complex, designs – e.g., floorplanning, placement, clock-tree synthesis, routing, and . Specifically, the foundation of the Aprisa Product was to build their gorithms on a *route-centric, hierarchical data model.* The right-hand side of below highlights the Avatar strategy.

on of Design Closure



Joe then described the IC Design product strategy. *'The Nitro-SoC platform will be supported through the 16/14nm node. Going forward, Aprisa will be the SAPR solution for 7nm and below. The DRC engine that was internal to Aprisa will be replaced by Calibre InRoute."*



News | Events | Editorials | Tutorials | Jobs | IP | Catalogs | V

emens' Aprisa digital implementation solution
rtified for Samsung Foundry's advanced 4nm
ocesses

| Rate | Review | More EDA News

Share Tweet

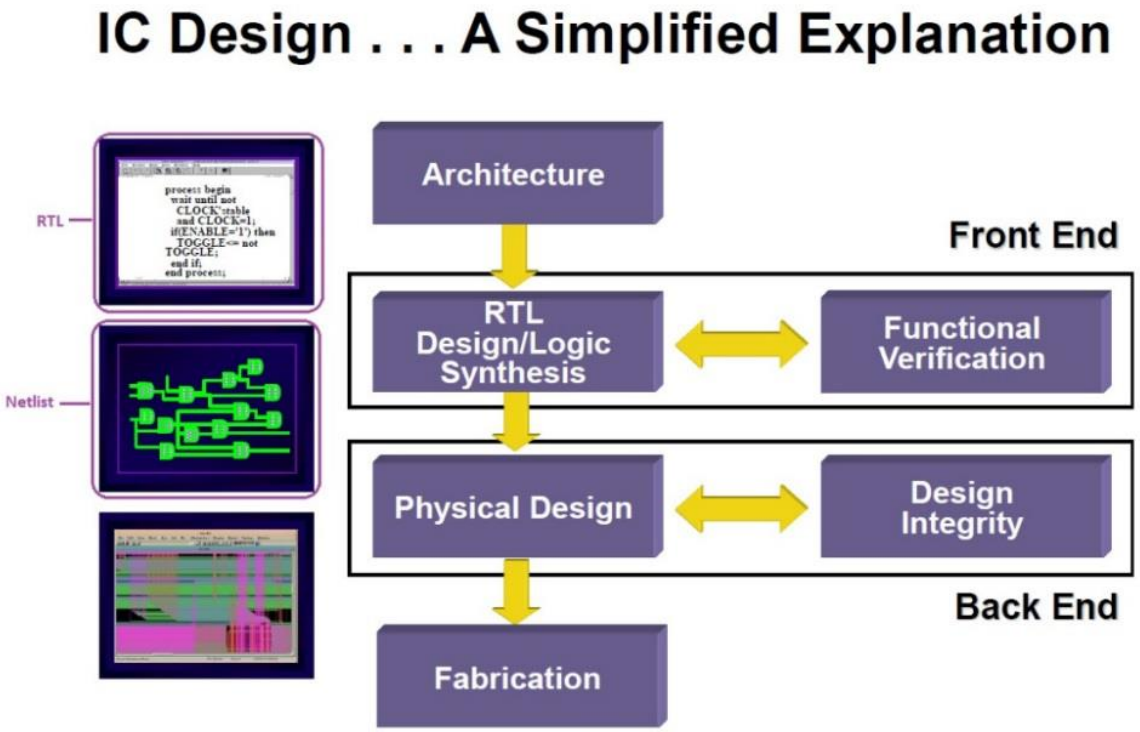
ware today announced that Samsung Foundry solution for the foundry's advanced 4nm FinFET tification, customers using Aprisa for digital jects at 4nm with fully certified technology that nd features of Samsung Foundry's most advanced form.

e latest in a series of milestones reflecting it to make Aprisa the industry's leading digital completing its acquisition of the portfolio in nore than doubled the Aprisa R&D team, while certification programs and product

its value to the Samsung Foundry ecosystem by utions in support of our most advanced vice president of the Foundry Design Technology 'By continuing to innovate leading-edge electronic nologies for our new processes, Siemens is again nsung and our mutual customers."

Focus of This Course

- What parts of the design flow are addressed?
  - Behavioral testbench VHDL.
  - Verification of RTL + TB code.
  - Verify system specs:
    Clock rate (via timing analysis) and power budget (via power analysis).
  - *Optionally (TBD):*
    Place and route.
    Coverage.
    SDF-based power analysis.



source: https://iliketoknow.wordpress.com/

Hardware Description Languages

- HDLs can describe hardware from behavioral to gate level,
  but mainly the range is RTL to gate level.
- There are two dominant HDLs:
  - VHDL (VHSIC HDL):
    originally for specification/documentation, based on Ada,
    inception 1981, US defence contract involving e.g. IBM and TI,
    became IEEE standard in 1987.
  - Verilog HDL:
    originally for simulation, inspired by C,
    developed at Gateway Design Automation 1983-85,
    Cadence acquires Gateway in 1989 and makes Verilog HDL public in May 1990.
    Cadence launches the Leapfrog VHDL simulator in 1993.

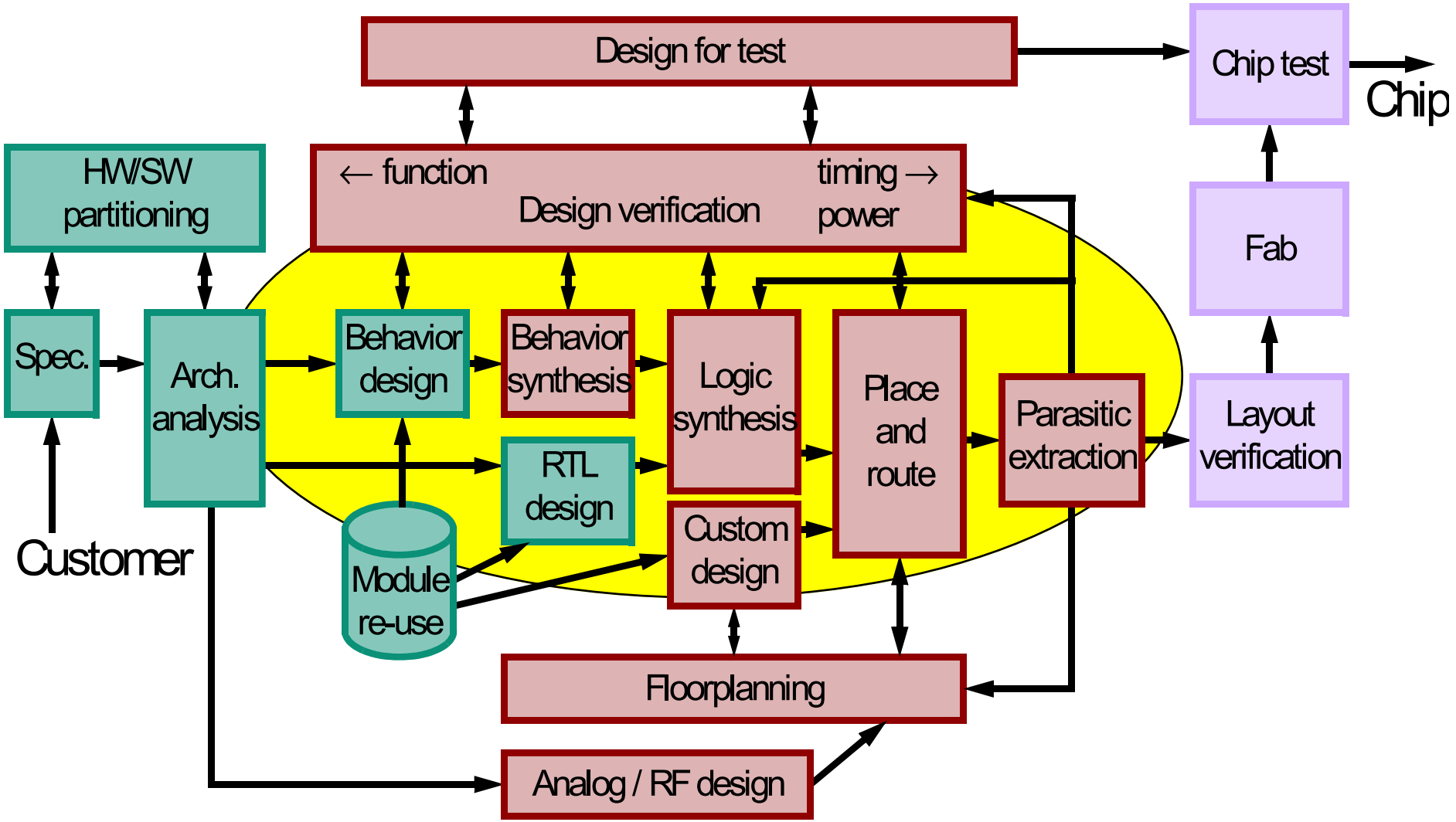
RTL vs Behavioral VHDL

- Efficient RTL code is preceded by the definition of a digital system architecture or algorithm functionality.
  - While sharing features of coding, *hardware description is not the same as programming*.
  - Always picture your system, using block and timing diagrams, before writing RTL code.
- The testbench code is different since it is behavioral which means it is not intended to be synthesized to gates.

Synthesis - “Compilation of HW”

- General design flow
  1. Behavior (C code or HDL) $\rightarrow$ RTL — *High-level synthesis*
  2. RTL $\rightarrow$ Physical implementation — *Logic synthesis*
- Conventional design flow for ASICs
  1. RTL-specification $\rightarrow$ Generic gate netlist.
  2. Generic gate netlist $\rightarrow$ Cell library.
  3. Cell library $\rightarrow$ Placement and routing.

Complete ASIC Design Flow



Labs – Core of the Course

- Preparation Study Week 1
  - Develop block and timing diagrams and initiate testbench design.
- Lab 1: Verification Using Testbench and Test Vectors SW 2
- Lab 2: Synthesis of Adder RTL Code SW 3
- Lab 3: Verification of Netlist SW 4
- Lab 4: Power Analysis SW 5
- *Optional labs (TBD):*
  - *P&R, coverage and SDF-based power analysis.*
- Lab report.

Lab Schedule

- For labs, you can choose either to work Wednesday 13-17 or Friday 13-17.
- Lab hours in 4220:
  - Lab 1: Wed. 13-17 and Fri. 13-17 in SW 2.
  - Lab 2: Wed. 13-17 and Fri. 13-17 in SW 3.
  - Lab 3: Wed. 13-17 and Fri. 13-17 in SW 4.
  - Lab 4: Wed. 13-17 and Fri. 13-17 in SW 5.
  - *Optionals: Wed. 13-17 and Fri. 13-17 in SW 6.*
  - *Backup: Wed. 13-17 and Fri. 13-17 in SW 7.*
- Since labs are completely redesigned this year, no TAs will be used and my in-hall supervision will be focused on the first half of each occasion.

Lab Assignments

- Preparation task Lab 1 (see lab memo).
  - Class discussion Friday Nov. 4 13:30-15:00 via Zoom. https://chalmers.zoom.us/j/69388246617
  - Be prepared to share screen with your solutions.
- Lab report:
  - 6-page individual lab report.
  - Describe your lab work holistically.
  - Consider lab learning outcomes and supporting lectures.
  - Use book chapters and papers as references for your discussions.
  - Deadline Thursday Jan. 5, 2023.
  - Comments will be given but no resubmission is possible after Jan. 5.

Grading Principles for Lab Report

- Performing labs 1-4 satisfactorily is a requirement to pass.
  - Addressing optional labs in the report likely leads to a higher grade.
- Report should convey …
  - (grade 3) understanding of lab work performed.
  - (grade 4) good understanding of lab work performed.
  - (grade 5) very good understanding of lab work performed.
- The quality with which you relate your lab work to book chapters, papers etc. will impact the grade.

Criteria	Weighting
Overall impression	20%
Content & understanding	40%
Structure	20%
Language	20%
Combined grade	100%

Chalmers HISS criteria

[more on this on Lecture 3]

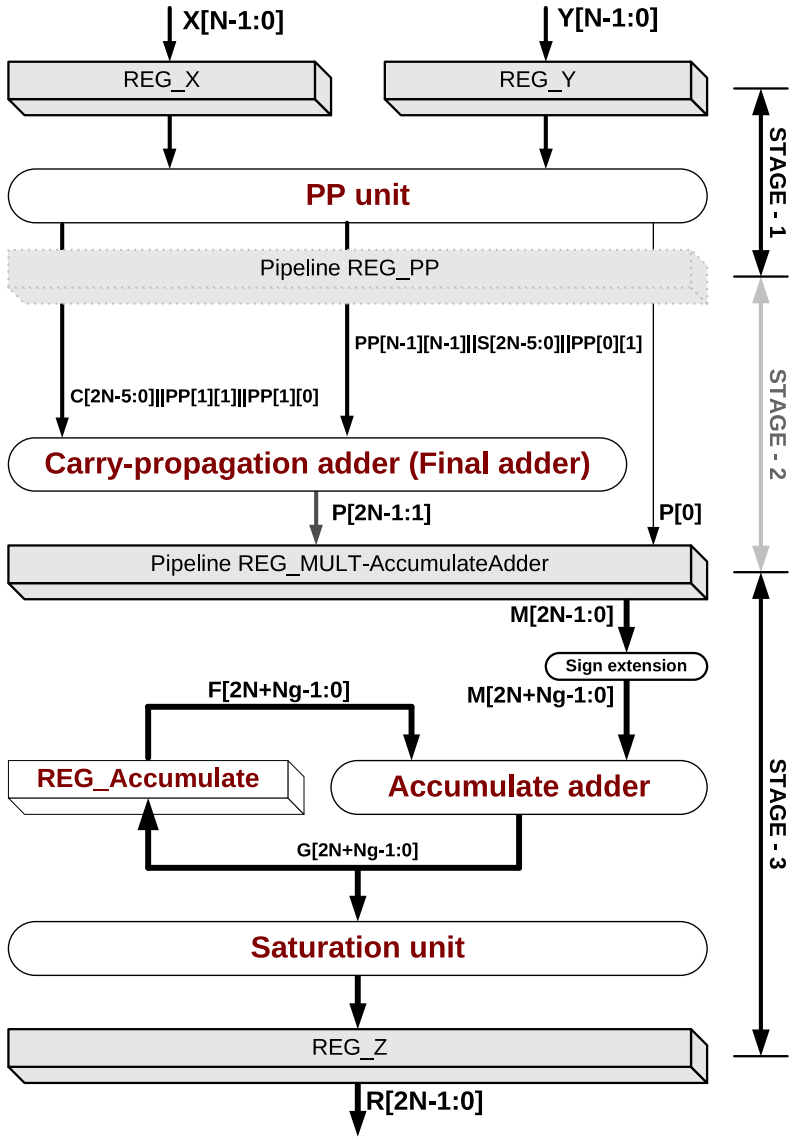
Technical Writing

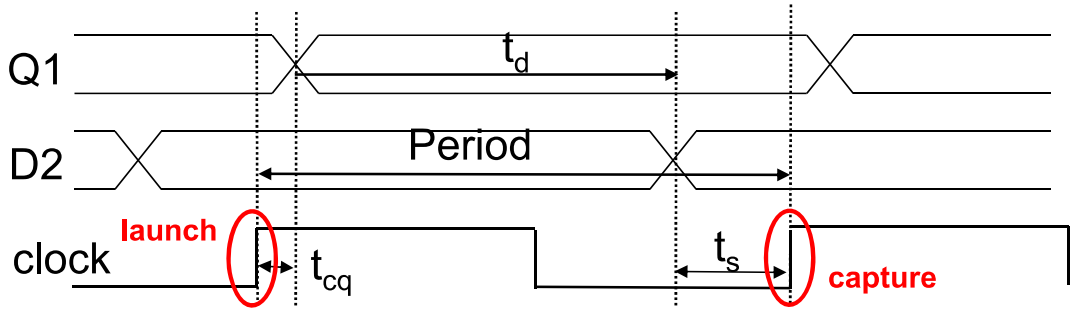
- In addition to the practical work in the labs,
  we will work actively on technical writing;
  follow the lectures and read up on sources
  like chapters and research papers.
  - A first technical writing lecture next Tuesday, Nov. 8 [Lecture 3].
  - One lecture (Nov. 22) and a final workshop on technical writing (Dec. 9)
    with Anne Hsu Nilsson, GU.
- The writing training will prove useful in your future career,
  not least in the EESD project and during the MSc thesis work.

Writing Tools

- I provide a LaTeX quick start guide, in case you want to use LaTeX to write your lab report.
  - In Canvas, check out *Writing resources*.
  - There is this software called Word out there. If you intend to work with figures, stay away from Word.
  - Additionally, consider what software you can use to draw figures: Perhaps Inkscape (https://inkscape.org/)...
- LaTeX is exclusively used for the MSc thesis, so you may just as well start using it.

Get Going with Labs — Block and Timing Diagrams





Start of TB Skeleton

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_textio.all;
use STD.textio.all;

-- entity declaration
-- architecture start
-- component declarations, for example
  component wrapper is
    generic (WL : positive);
    port(
      clk  : in  std_logic;
      a    : in  std_logic_vector(WL-1 downto 0);
      b    : in  std_logic_vector(WL-1 downto 0);
      cin  : in  std_logic;
      cout : out std_logic;
      sum  : out std_logic_vector(WL-1 downto 0));
  end component;
```

```
-- constant and type declarations, for example

 constant WL         : positive := 16;
   -- adder wordlength

 constant CYCLES     : positive := 1000;
   -- number of test vectors to load

 type word_array is array (0 to CYCLES-1) of std_logic_vector(WL-1 downto 0);
   -- type used to store WL-bit test vectors for CYCLES cycles

 file LOG : text open write_mode is "mylog.log";
   -- file to which you can write information
```

TB Functions

```
function to_std_logic (char : character) return std_logic is
  variable result : std_logic;
begin
  case char is
    when '0'    => result := '0';
    when '1'    => result := '1';
    when 'x'    => result := '0';
    when others => assert (false) report "no valid binary character read" severity failure;
  end case;
  return result;
end to_std_logic;
```

```
function load_words (file_name : string) return word_array is
  file object_file : text open read_mode is file_name;
  variable memory  : word_array;
  variable L       : line;
  variable index   : natural := 0;
  variable char    : character;
begin
  while not endfile(object_file) loop
    readline(object_file, L);
    for i in WL-1 downto 0 loop
      read(L, char);
      memory(index)(i) := to_std_logic(char);
    end loop;
    index := index + 1;
  end loop;
  return memory;
end load_words;
```

Practical Advice

```
-- testbench code

 verification_process : process
   -- ...
   variable index : natural := 0;
   variable L     : line;
 begin
   -- ...
   write(L, string'("index = "));
   write(L, index);
   writeline(LOG, L);  -- example on how you can write a mix of string text and variables
                       -- from your testbench to file LOG, which was opened above
```

- Use assert to test hypotheses, for example,

```
assert (X = Y) report "Error" severity warning;
```

This will send a warning to the screen.

- You can also save error information to file, for example, by

```
if X /= Y then write(L, string'("Error!"));
```

Useful Reference Textbook

- You can find “The Designer's Guide to VHDL”, by Peter J Ashenden in the Chalmers e-library:
  - https://ebookcentral.proquest.com/lib/chalmers/detail.action?docID=452921
- Consider especially ch. 16 on file I/O and ch.18 on testbenches as supporting material for developing the testbench.

File and Directory Structure (1)

```
> mkdir lab1
> cd lab1

> mkdir RCA
> cd RCA
```

.. perform lab assignments ⇒

```
> ls
FA.vhdl   RCA.vhdl   Wrapper.vhdl

> emacs TB.vhdl
> ls
FA.vhdl   RCA.vhdl   TB.vhdl Wrapper.vhdl
```

Later move on to Sklansky…

```
> cd ..
> mkdir SKL
> cd SKL
```

File and Directory Structure (2)

```
> ls
lab1 lab2 lab3 vhdl
```

Refer to separate directory for VHDL files $\Rightarrow$

```
> cd lab 1
```

Give relative reference

```
> ls ../vhdl
FA.vhdl   RCA.vhdl   Wrapper.vhdl
```

or refer to home directory for absolute reference (assuming you perform labs under DAT110):

```
> ls ~/DAT110/vhdl/
```