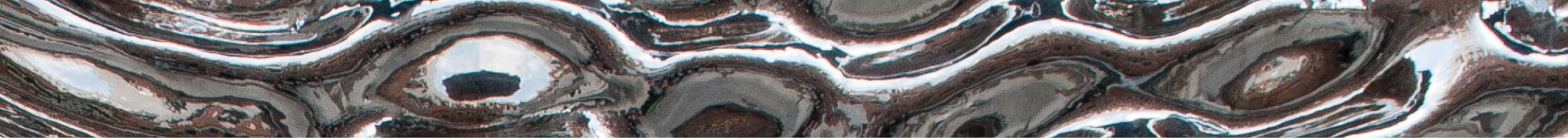


UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

IN3160, IN4160 Digital system design

# Introduction + HDL, PL and Design flow

Yngve Hafting

# Overview

- General information
  - Course management
  - Schedule
  - Course Goals
  - Curriculum
  - Lab assignments
  - Who are we
- Motivation
  - Why Digital Design?
  - Why HDL?
- Intro to programmable Logic
  - What is programmable logic?
  - Why choose programmable logic?
- Design Flow for digital designs
- Intro to our hardware…:
  - Zedboard
    - Architecture
    - Documentation
  - «Our» HDL: VHDL

- Assignments and suggested reading for this week



# Course Management

- Lecturers:
  - **Roar** Skogstrøm (II'er IFI)
  - **Alexander** Wold (II'er IFI)
  - **Yngve** Hafting  (Universitetslektor IFI/ROBIN)
- Lab supervisors / teachers:
  - "**Hoi**" Bihui Chen (student)
  - **Georg** Magneshaugen (Student)
  - **Max** Lauritz Øyen (Student)
  - **Sander** Elias Magnussen Helgesen (Student)
  - Seyed **Mojtaba** Karbasi (PhD)

## Lectures

Monday 14:15 -16:00, OJD Logo «2438»)

Friday 10:15-12:00, OJD Logo «2438»)

## Lab

LISP (2428): TBD- lab will be manned certain time slots- poll next slide

https://www.mn.uio.no/ifi/om/finn-fram/apningstider/

| Monday | Tuesday | Wednesday | Thursday | Friday |
|---|---|---|---|---|
| | | | | |
| | | | | *Lecture 10-12* |
| *Lecture 14-16* | | | | |
| | | | | |

## Web

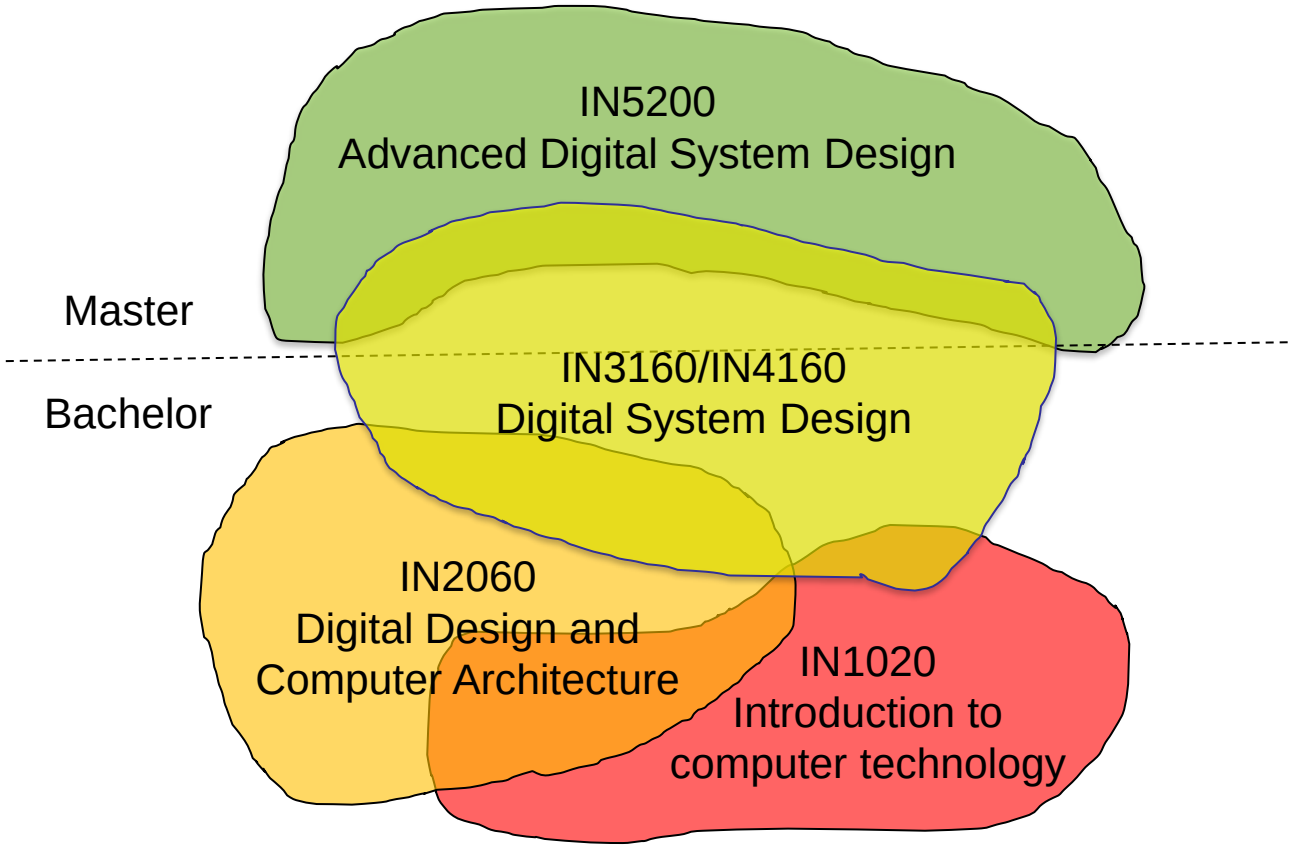
http://www.uio.no/studier/emner/matnat/ifi/IN3160/
(covers also INF4160)

# Where do we stand + lab supervision poll?

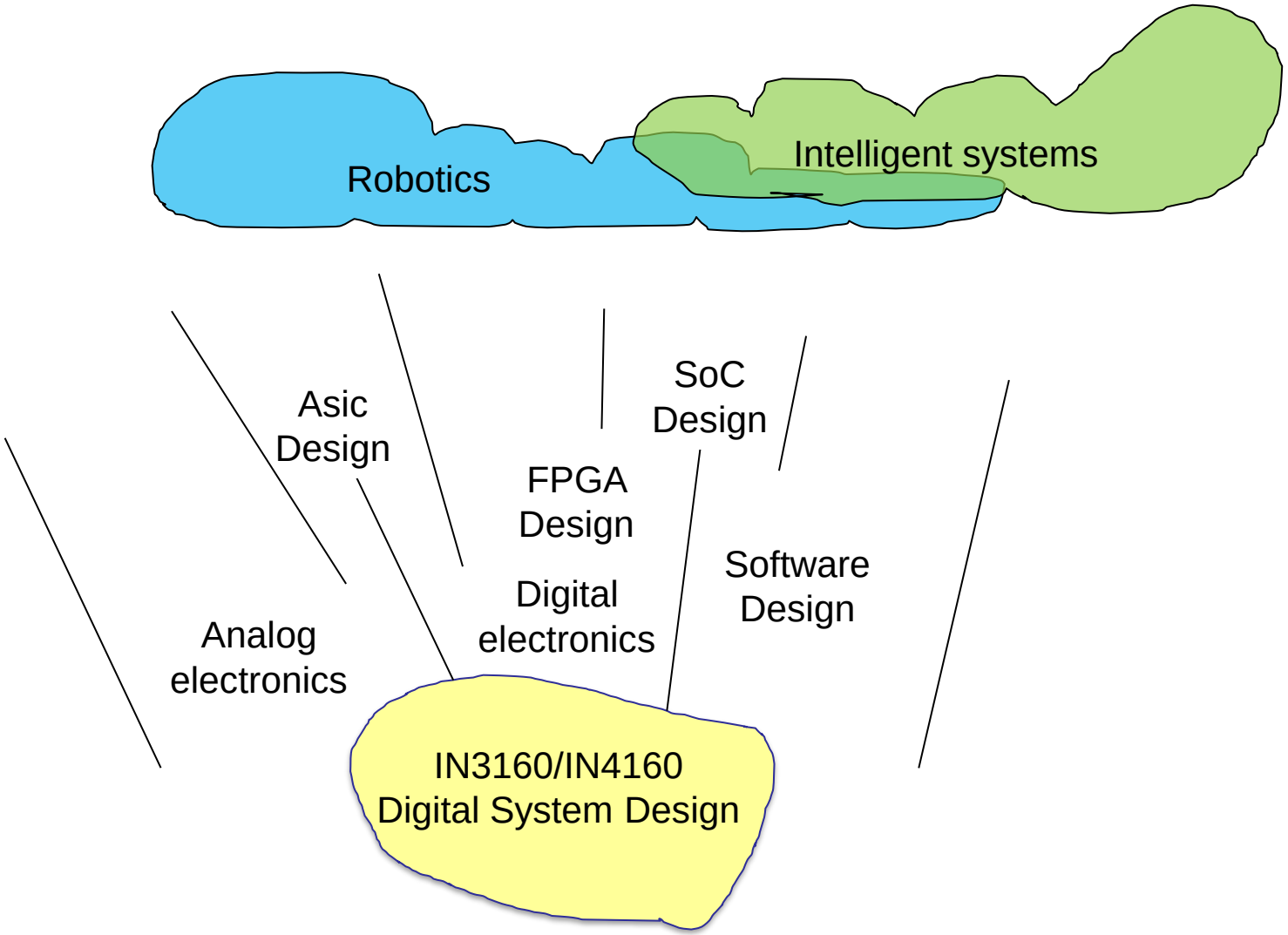
- www.menti.com
- Code 19654197

- Python testbenches will be mandatory in this course from 2024.
  - This year we will allow some testing amongst voulenteers.
  - The assignments will be the same design-wise, but test benches will be written using Python and tested using an open source framework (cocotb, GHDL and GTKWave)

# Study program connections



# Relevancy




Finishing this course you will be able to do work within the field of digital design.

# Course Goals and Learning Outcome

https://www.uio.no/studier/emner/matnat/ifi/IN3160/index-eng.html

- In this course you will learn about the design of advanced digital systems.
- This includes programmable logic circuits, a hardware design language and system-on-chip design (processor, memory and logic on a chip).
- Lab assignments provide practical experience in how real design can be made.

- After completion of the course you'll:...

# ... IN3160 vs IN4160 ...

IN3160

- After completion of the course you'll:
  - understand important principles for design and testing of digital systems
  - understand the relationship between behavior and different construction criteria
  - be able to describe advanced digital systems at different levels of detail
  - be able to perform simulation and synthesis of digital systems.

IN4160

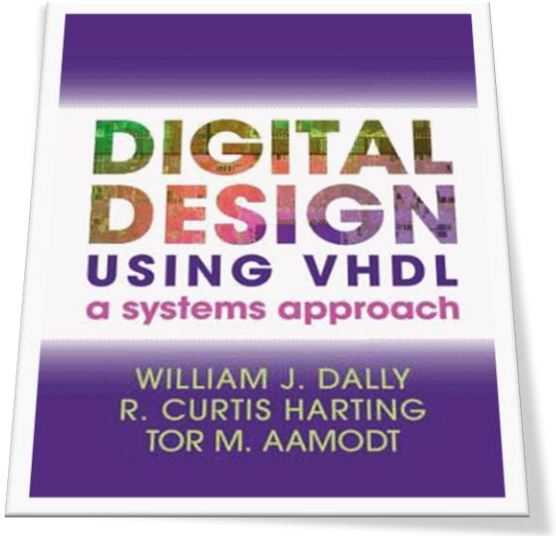
- After completion of the course you'll:
  - understand important principles for design and testing of digital systems
  - understand the relationship between behavior and different construction criteria
  - be able to describe advanced digital systems at different levels of detail
  - be able to perform **advanced** simulation and synthesis of digital systems
  - **be able to perform advanced implementation and analysis techniques**

**NOTE**: these are MINIMUM requirements for passing an exam.

- You will be given the same opportunities to learn, and the curriculum is the same.
- *Grading will be (slightly) stricter for IN4160 due to added minimum requirements*
- Otherwise, this course will be held as one.

# Syllabus

- Dally, William J. - Harting, R. Curtis - Aamodt, Tor M.
  **Digital Design Using VHDL A Systems Approach**
  Cambridge University Press 2016
  ISBN9781107098862




- Lectures and lecture slides
- Mandatory assignments
- Handouts – Will be made available digitally on semester page
  (Link from 2022 can be used until the 2023 link is ready)
  - Cookbook
  - Articles (Reset Circuits, Steve Kilts)

# Compulsory lab assignments

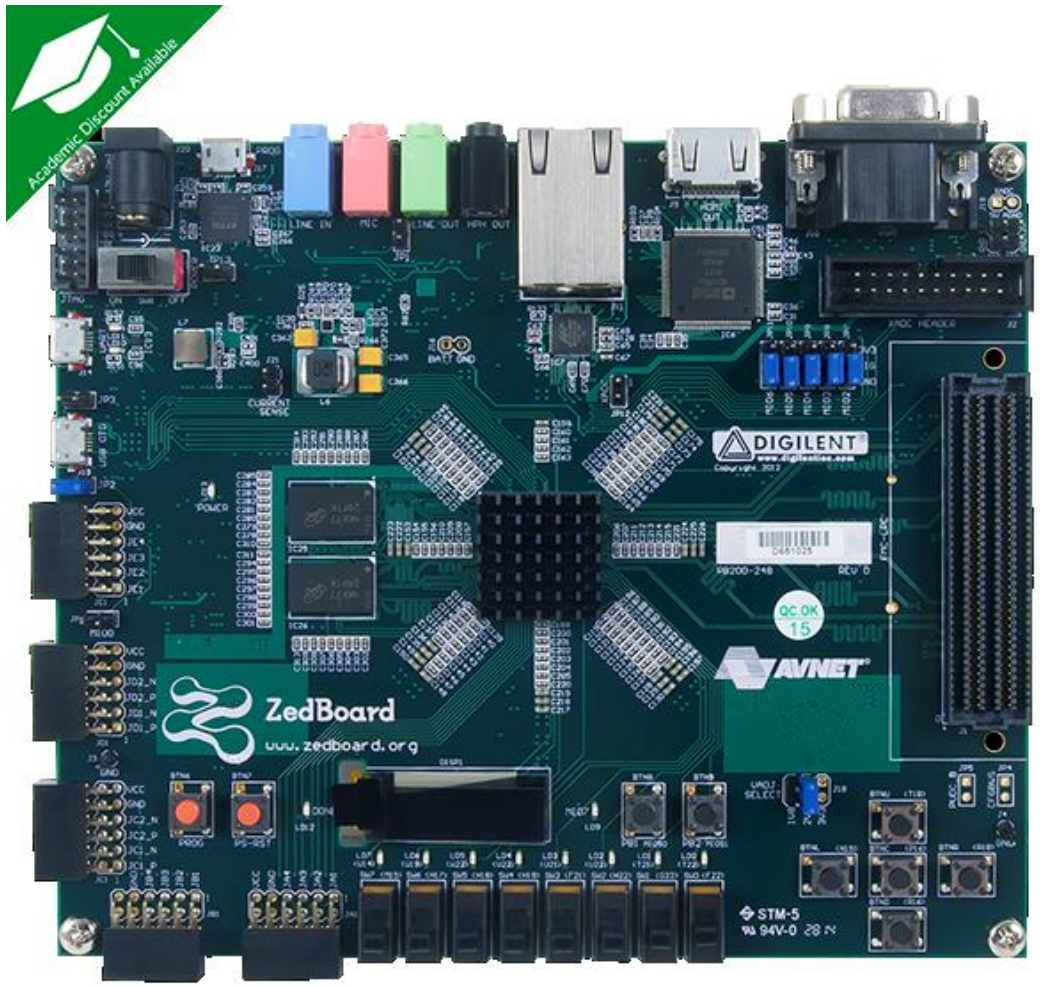
- There are 10 compulsory lab assignments.
  - *The book has chapter-exercises that can be used for self-study.*
- All assignments must be completed to take the exam.
  - *Lab workload and complexity increases* through the semester

- Lectures are prerequisite for some assignments
  - Lectures most intensive in the beginning

- The lab assignments utilises the digilent Zedboard, featuring a Xilinx Zynq 7020 device that includes both a hardcoded ARM processor and FPGA fabric.

- You will be introduced to tools and board first.

- By the end of this course you will design a system, using both processor and FPGA fabric, that will both regulate, read and display the speed of an electric motor connected to the board.




https://store.digilentinc.com/zedboard-zynq-7000-arm-fpga-soc-development-board/

# LAB

- Lab starts now!
  - **Assignments are available in Canvas!**
  - Assignments are individual.
    - WHAT ABOUT...
      - Collaboration?
      - Previously approved assignments?
      - ChatGPT?
  - There will be one assignment (3) using peer review only.
    - Your reviews are mandatory for your approval!
  - Some assignments may require that you show your setup to the lab supervisor.
    - ***Labs can be done entirely remote, but on-site is strongly adviced.***
- **LISP (2428) is the LAB.**
  - Both hardware and software will be available in LISP.
    - 4 boards with camera will be available online for those in quarantine/ isolation / specieal needs.
- Questions..?

# Software

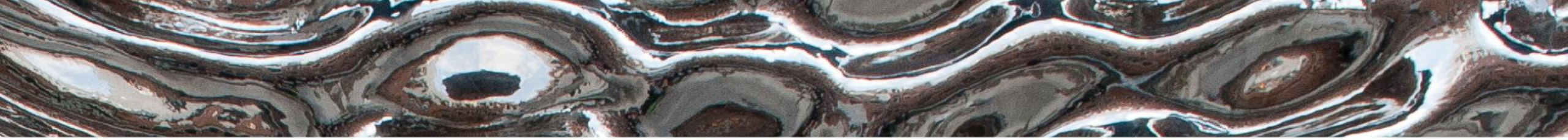
- Vivado, Vitis
  - Floorplanning and Programming FPGA boards
  - https://www.xilinx.com/products/design-tools/vivado/vivado-ml.html
    - Standard edition is free and should be sufficient up to assignment 9.
- Questa=Modelsim
  - Compilation, Simulation, waveform viewer
  - "industry standard"

- Tool chain for Python Testbenches = Vivado + these
  - GHDL
    - Open source VHDL simulation, used together with CocoTB below.
  - GTKWave
    - Open source waveform viewer
  - CocoTB
    - Cosimulation framework
    - This is invoked when using "make" when using python based testbenches.

- All software can be accessed from Linux machines on IFI, and IFI-digital-electronics

# Resources

- Semester page/ course web "Vortex"
  - Course information
    - Exam date, lecture schedule, etc.
- Canvas (link from semester page)
  - Assignment-files, delivery and -feedback
  - Some links to external content
- in3160-discourse
  - Communication and discussion:
    - Requires login, allowes anonymous posting.
    - Both students and staff can answer ☺
  - Note: Use the manned lab hours as much as possible for questions.



UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

**IN3160**
**Introduksjon, HDL og PL**

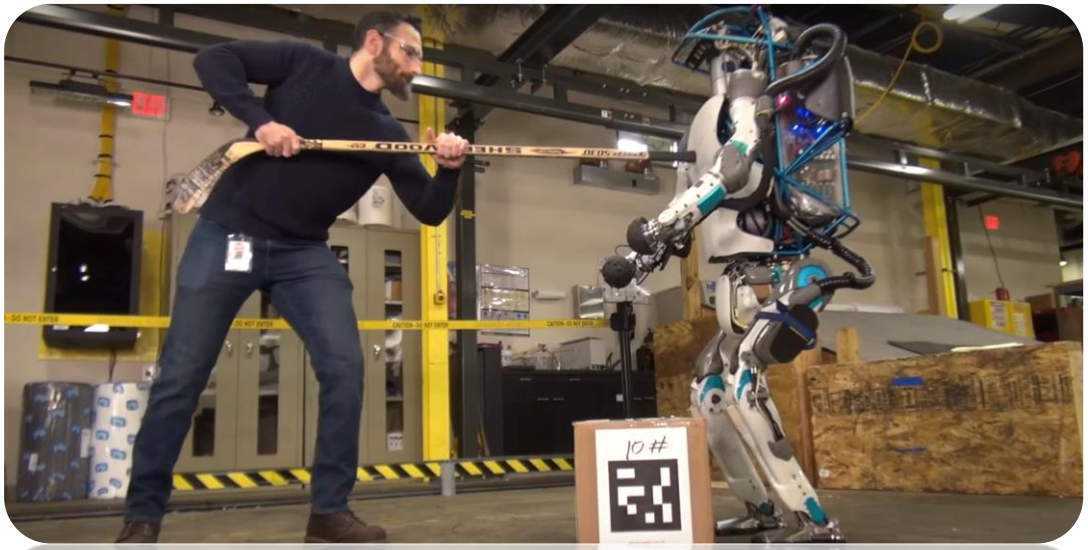
# **H**ardware **D**escription **L**anguage & **P**rogrammable **L**ogic

Yngve Hafting

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

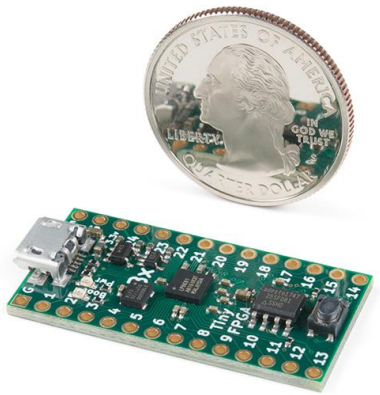
# Why learn Digital Systems Design?

http://www.aes-eu.com/10gbe-fpga-nic.php

https://www.youtube.com/watch?v=rVlhMGQgDkY

http://www.moisund.com/2014/04/08/opning-av-ny-basestasjon-i-aseral/

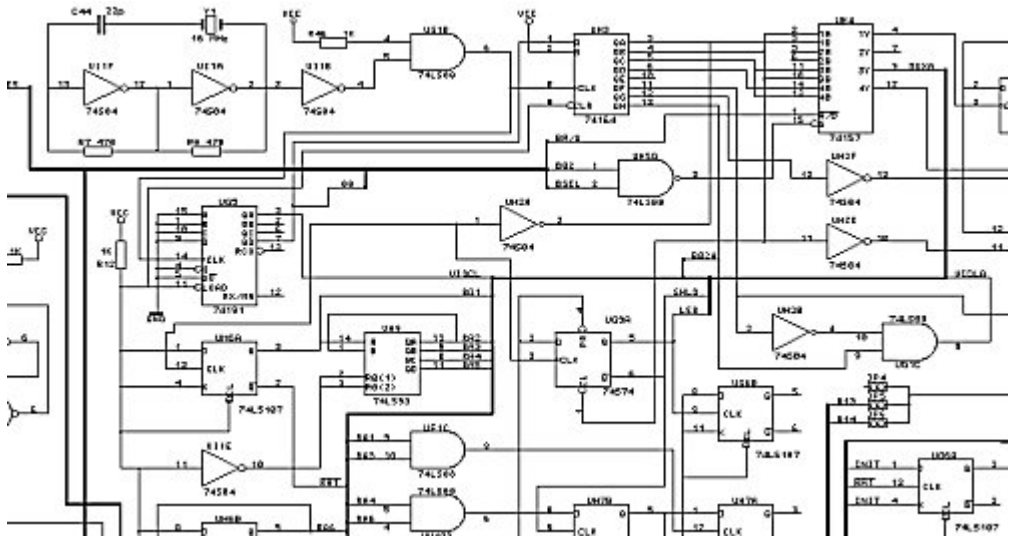


https://www.sparkfun.com/products/14829

Wikipedia: U.S. Navy photo by Photographer's Mate Airman Marvin E. Thompson Jr.

https://www.komplett.no/product/11257/pc-nettbrett/komplett-pc/komplett-gamer-xtreme/komplett-gamer-xtreme-i250?offerId=KOMPLETT-310-11257#

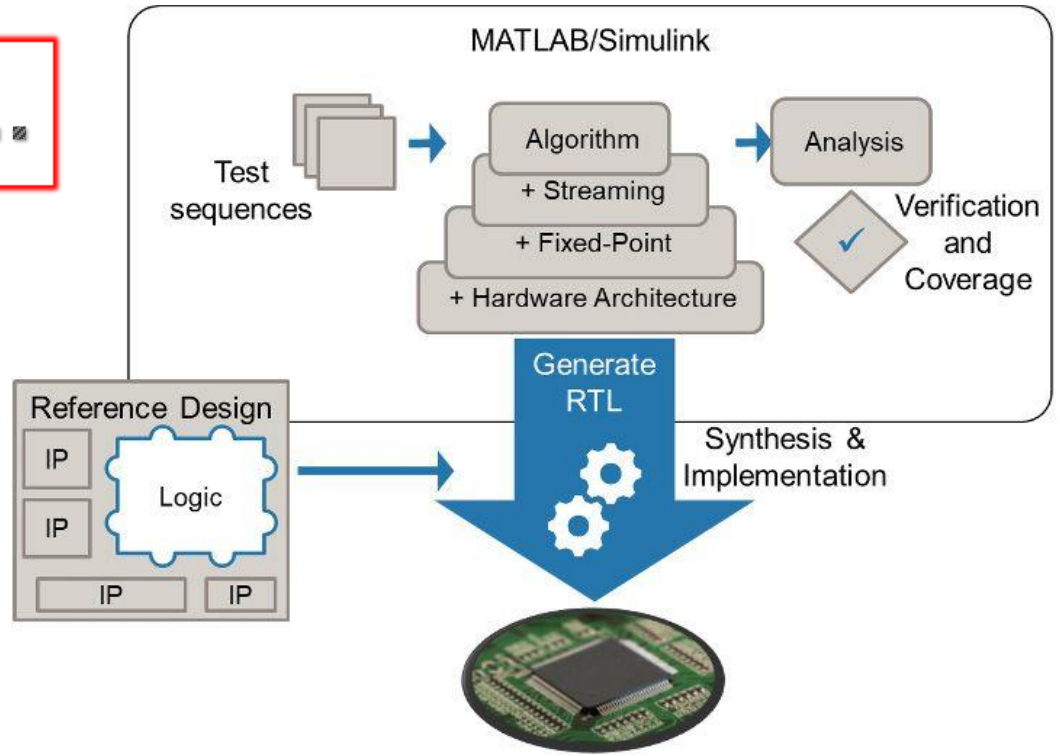
# How to implement digital designs...



- Low Level
  - Netlists
  - Schematic diagrams
  - Programming using hardware description languages (HDL programming)

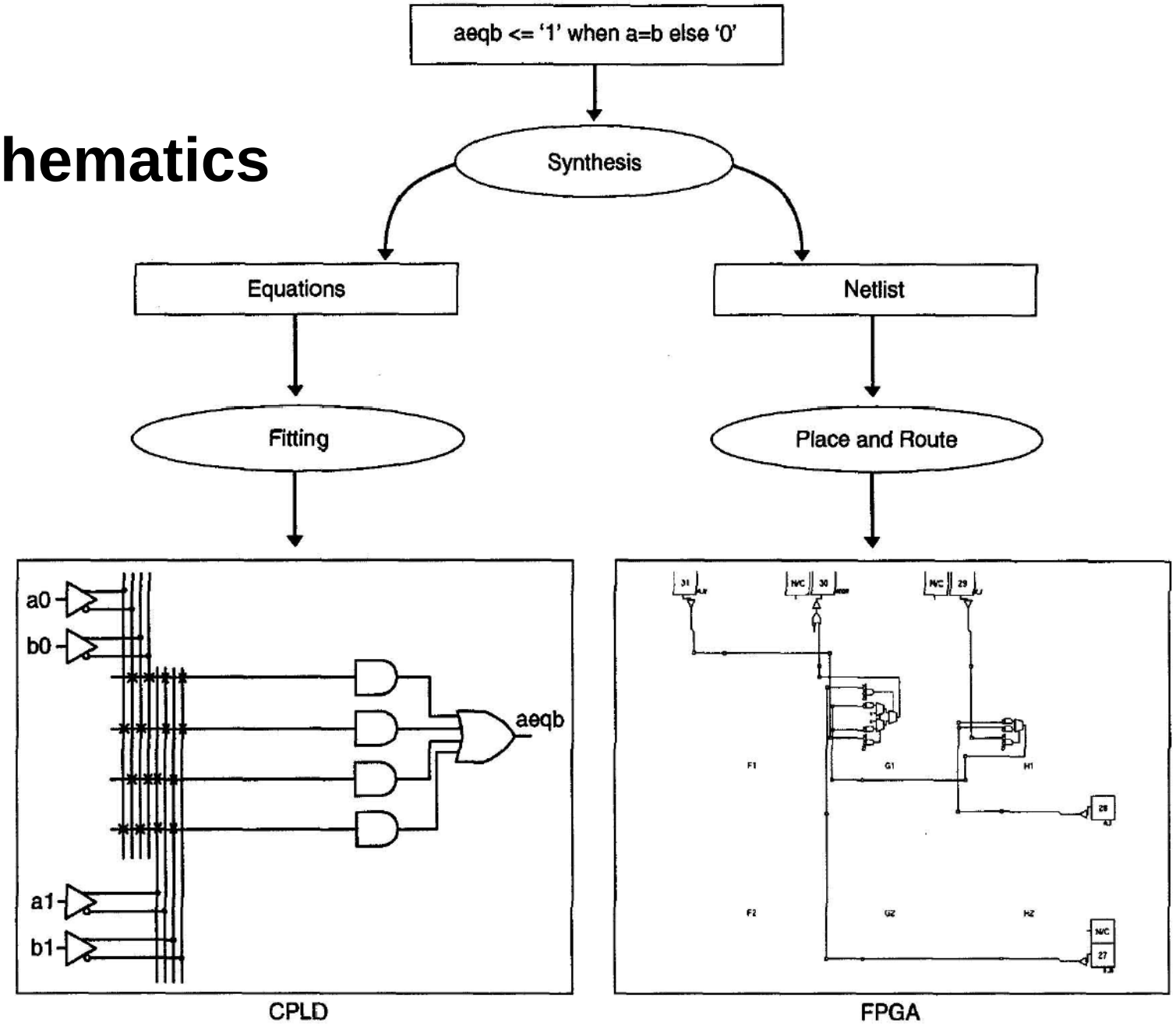
**IN3160...**

- High Level
  - HDL programming (RTL)
  - Block diagrams
    - Connecting premade models (IP's)
  - High level synthesis...
    - Code Generators (Matlab/Simulink)
      - Uses IP's
      - Generates HDL/ Netlists



# HDL vs netlists/ Schematics

- Syntesis enables
  - One design
    - Several physical implementations



# Why HDL?

- Technology independent code
- Different abstraction layers

**Netlist:**

```
U1: xor2 port map(a(0), b(0), x(0));
U2: xor2 port map(a(1), b(1), x(1));
U3: nor2 port map(x(0), x(1), aeqb);
```

**Boolean equations:**

```
aeqb <=      (a(0) xor b(0))
         nor (a(1) xor b(1));
```

**Concurrent statements:**

```
aeqb <= '1' when a=b else '0';
```

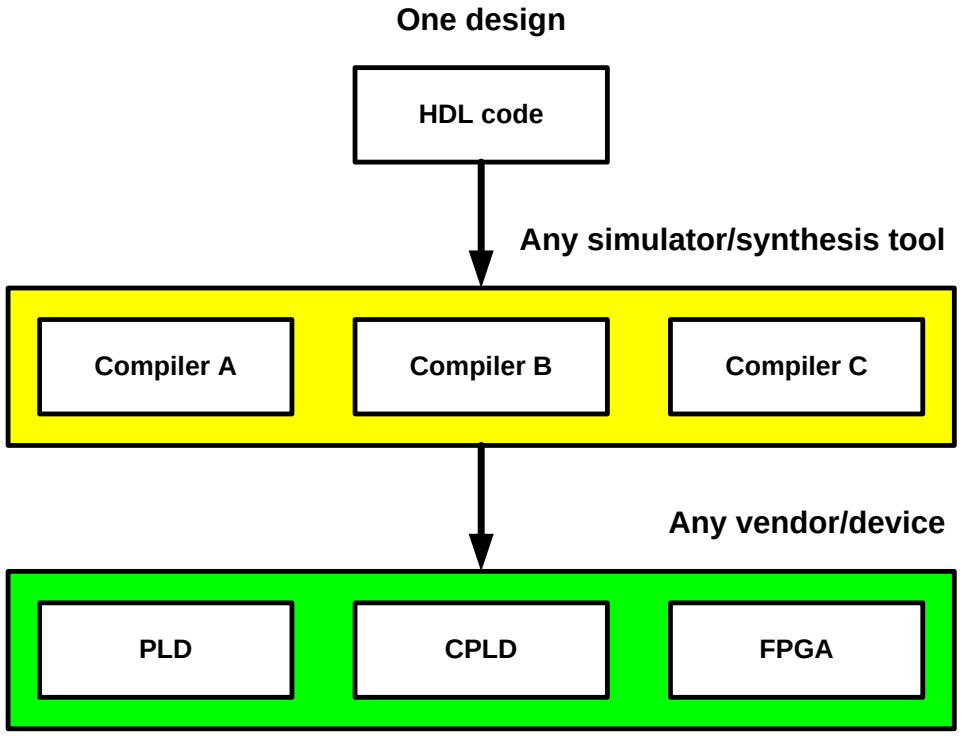
**Sequential statements:**

```
if a=b then
  aeqb <= '1';
else
  aeqb <= '0';
end if;
```

# Why HDL?

- Portability
  - Tool and device independency

- IEEE Standards
  - VHDL and System Verilog
    - Both IEEE standards
      (Institute of Electrical and Electronics Engineers)
    - VHDL - IEEE 1076
    - System Verilog - IEEE 1364



# Why HDL?

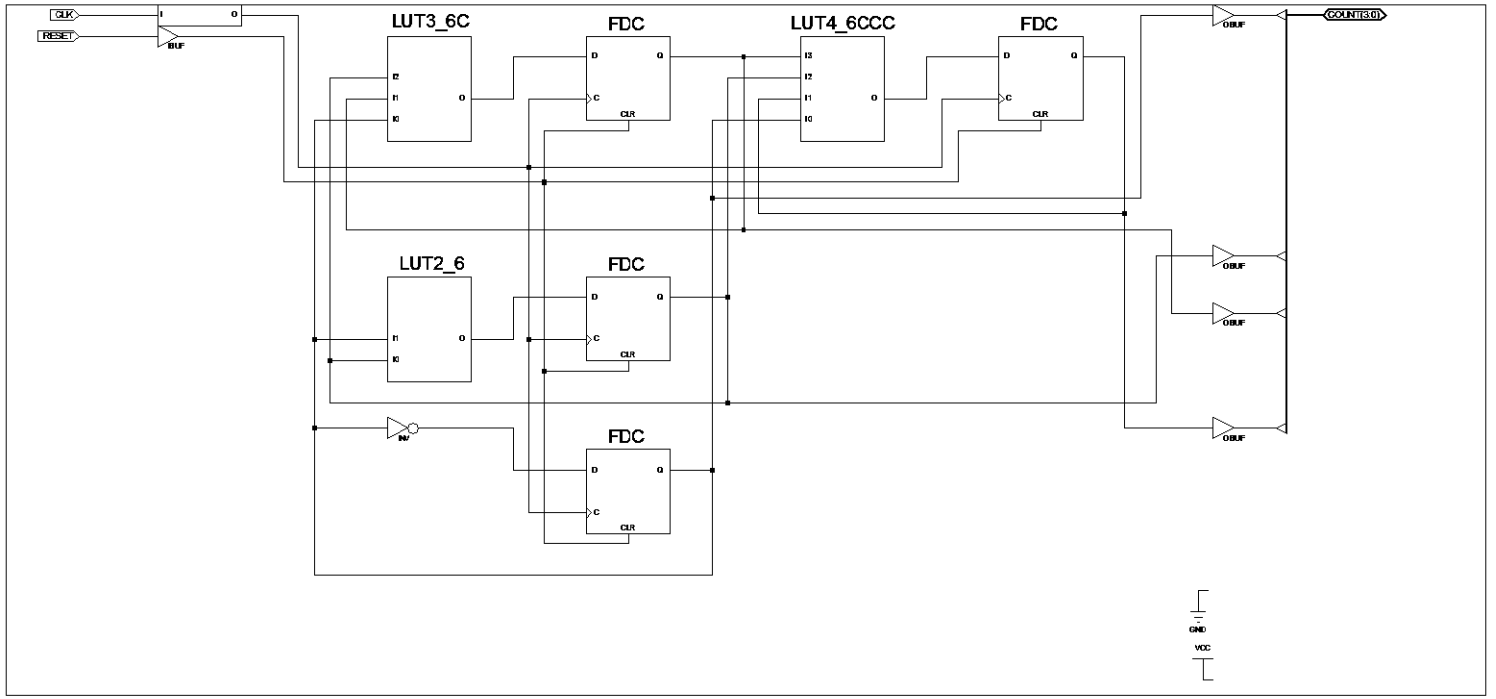
- Simple mainteenance/expansion of a design

```
COUNT : inout std_logic_vector(7 downto 0); -- Count value
---
COUNTER :
  process (RESET,CLK)
  begin
    if(RESET = '1') then
      COUNT <= (others => '0');
    elsif rising_edge(CLK) then
      COUNT <= COUNT + 1;
    end if;
  end process COUNTER;
```

# Why HDL?

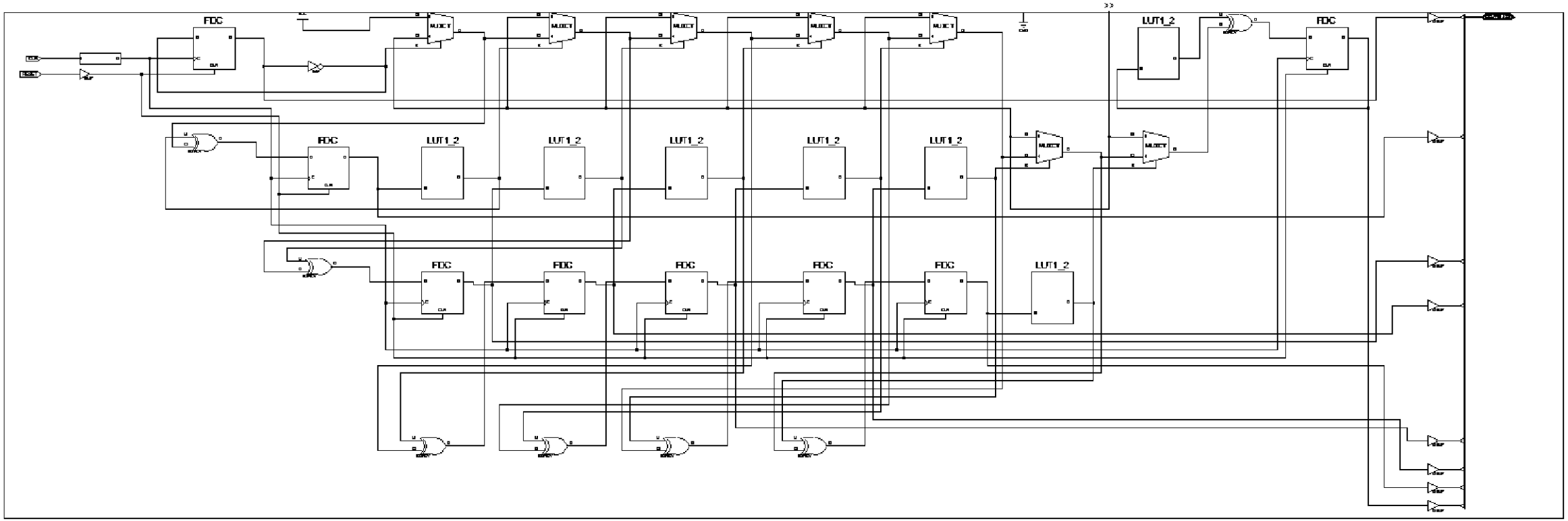
## 4 bit counter:

```
COUNT : inout std_logic_vector(3 downto 0); -- Count value
```



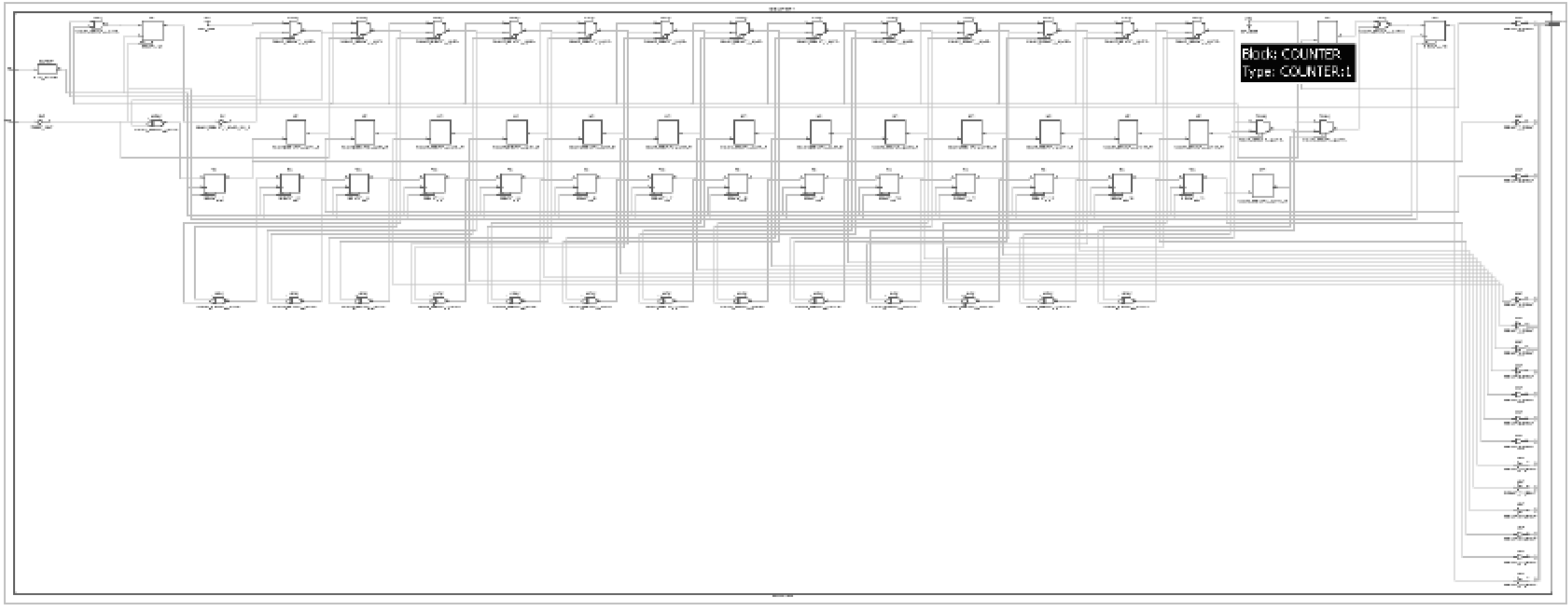
# 8 bit …

```
COUNT : inout std_logic_vector(7 downto 0); -- Count value
```



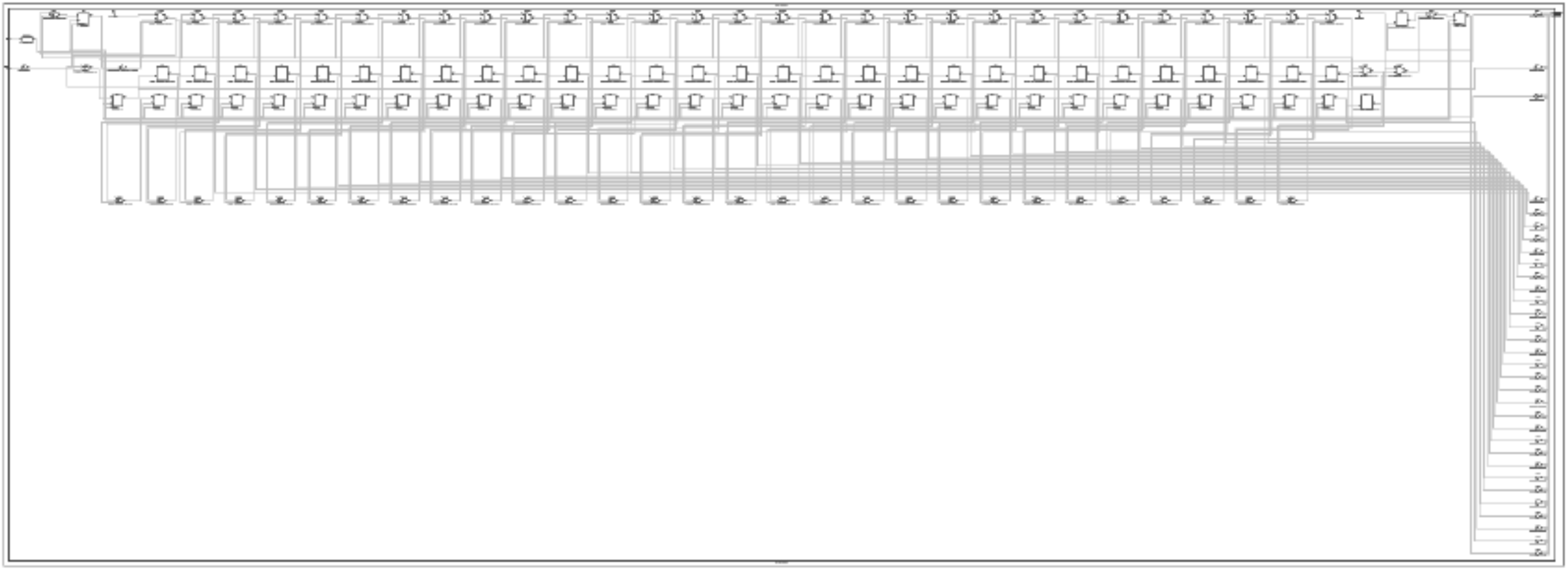
# 16 bit …

```
COUNT : inout std_logic_vector(15 downto 0); -- Count value
```



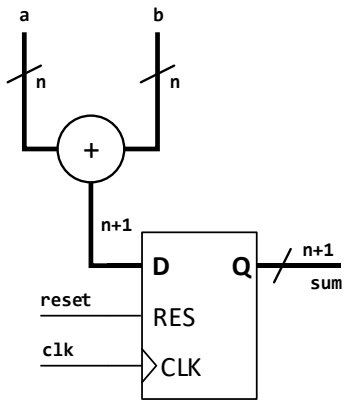
# 32 bit...

```
COUNT : inout std_logic_vector(31 downto 0); -- Count value
```



# HDL vs software

```
process(reset, clk)
begin
  if (reset = '1') then
    sum <= '0';
  elsif rising_edge(clk) then
    sum <= a + b;
  end if;
end process;
```




| HDL «Hardware description language» | Software programs |
|---|---|
| **Defines the logic function of a circuit** | **Defines the sequence of instructions and which data shall be used** for one or more processors or processor cores |
| **CAD tools *syntetizises* designs** to enable realization using physical gates. | ***A compiler translates program code to machine code instructions*** that the processor can read sequentially from memory |
| ***Implemented using programmable logic*** *(PL,* FPGA, CPLD, PLD, PAL, PLA, …) or *ASICs* (application specific circuits) (“ASICs”, processors, ..-chips,.. etc.) | **Is stored in computer memory** |
| **Verilog (SystemVerilog)**<br>**VHDL (VHDL 2008)**<br><br>(System C m. fl.) | C, C++, C#, Python, Java, assemblere (ARM, MIPS, x86, …) Fortran, LISP, Simula, Pascal, osv… |

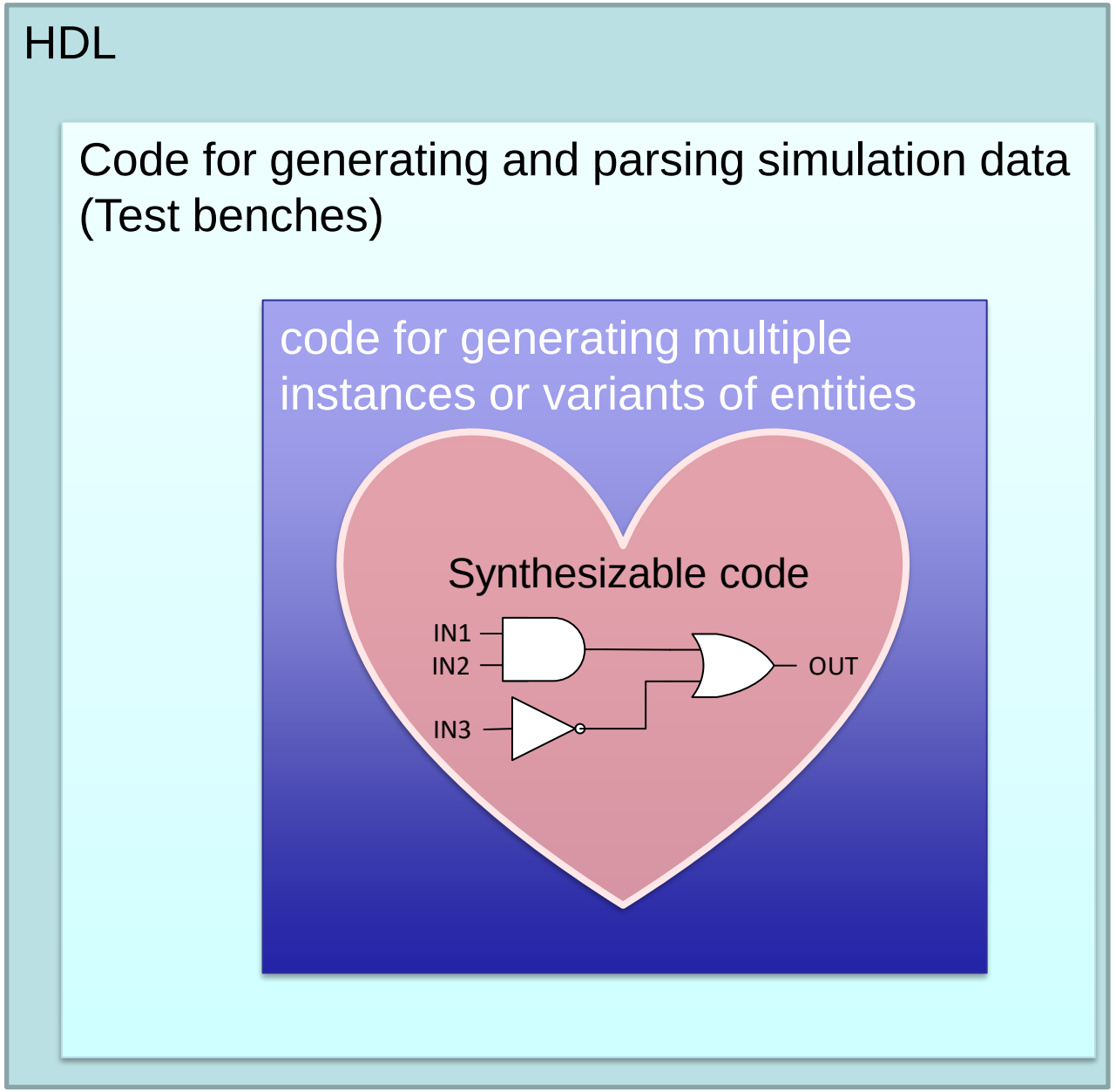
```
int sum(int a, int b){
  int s;
  s = a + b;
  return s;
}
```

```
MOV R5, #0       ;set base adr
LDR R7, [R5, #8] ;load reg R7
LDR R8, [R5, #12];load reg R8
ADD R0, R7, R8   ;R0=R7+R8
STR R0, [R5, #16];store R0
```

```
01001100 10011001 00100100 01001010
11001100 10111001 01100100 11110110
01001100 10011001 00111100 11101010
01001100 10011001 00100100 01011010
…
```
(binary code is random, for illustration only)

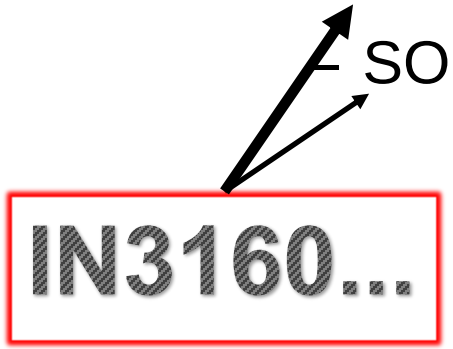
# What is HDL

- VHDL = VHSIC HDL:
  - Very High Speed Integrated Circuit **Hardware Description Language**
  - The purpose is to generate circuits, and verify their function through simulation.
  - **Synthesizable** (realizable) **code work concurrently** (in parallel, always on).
  - Code for simulation include things such as file I/O which cannot be synthesized.
  - Testbenches can use synthesizable elements, but will use sequential statements, and is only run as software.
    *This may be confusing at times...*



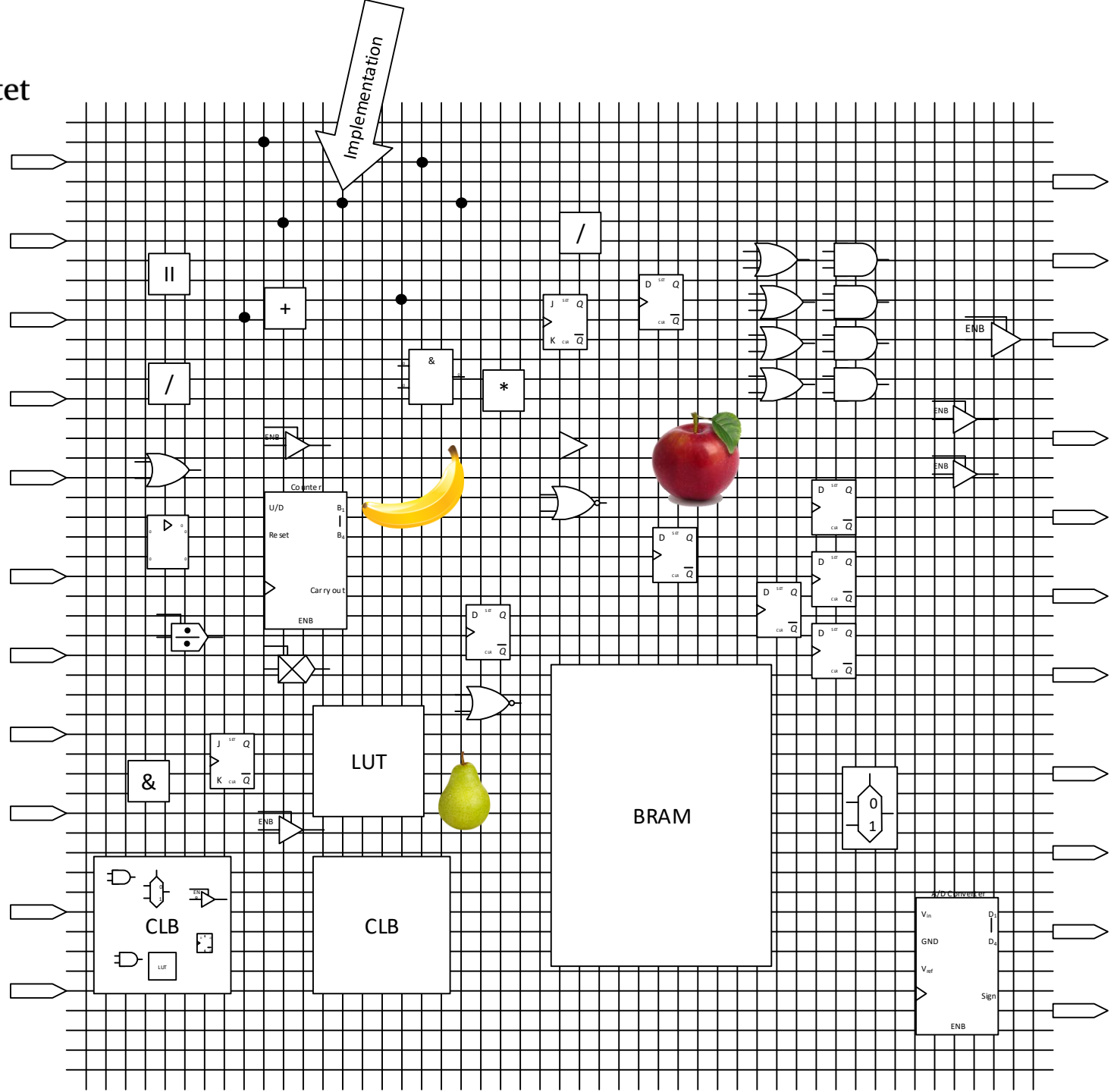
# What hardware..?

- ASIC : Application Specific Integrated Circuit
  - Processors, microcontrollers, GPUs (x86, ARM, GeForce)
  - Customized chips
- PL : Programmable logic
  - PLA, PAL "Programmable Logic Array" / .."Array Logic"
  - CPLD "Complex Programmable Logic Device" = Several PAL/PLAs, FFs
  - FPGA "Field Programmable Gate Array" = More complex array of primitives
  - SOC's "System On Chip" = FPGA + micro –prosessor/-controller



# What is PL and FPGA ? (Programmable Logic)

- PL = FPGA, CPLD, PLA…
  (Field Programmable Gate Array, Complex Programmable logic Device)
- PL vs processor (FPGA vs CPU, MCU) ?
- PL vs ASIC (Application Specific Integrated circuit)?

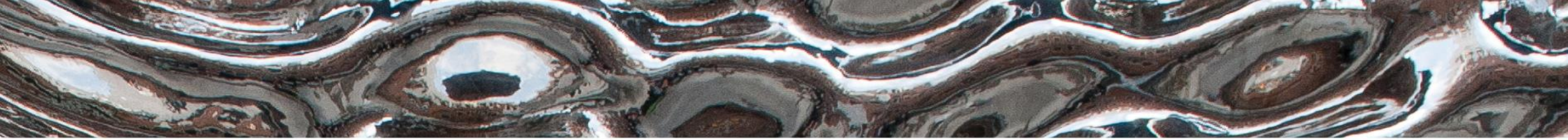


## When or why choose programmable logic?

- (Verify behavior of ASIC)
- Prototyping flexibility
  - Lots of multi purpose IO
  - Reprogrammable
- Small batch production
- Parallellism
- Custom / fast
- Runtime reconfigurability
- …

## When to avoid programmable logic?

- High Volume + low cost
- When dedicated HW is well suited.
- When extreme speed is required => ASIC
- …



UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

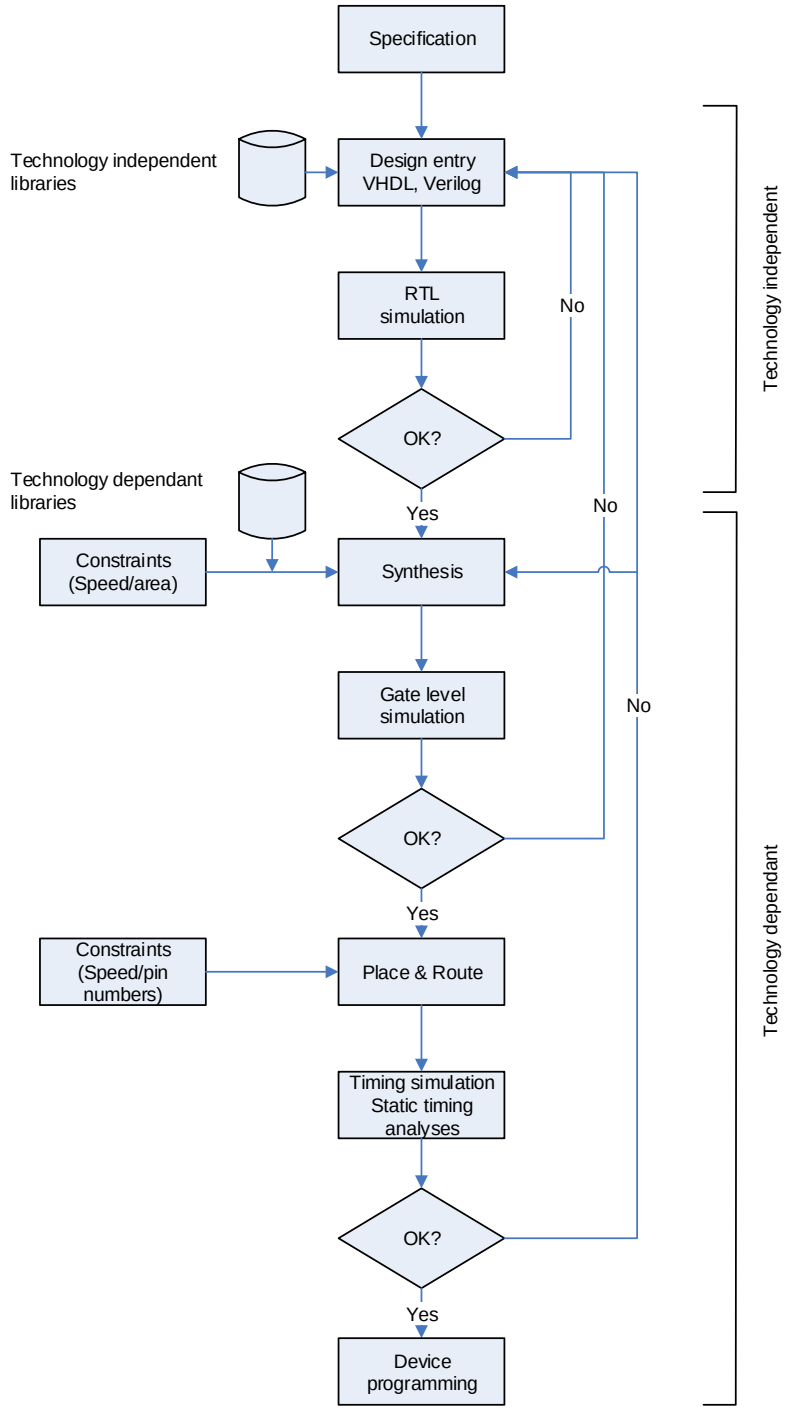
**IN3160**

# Digital Design Flow

Yngve Hafting

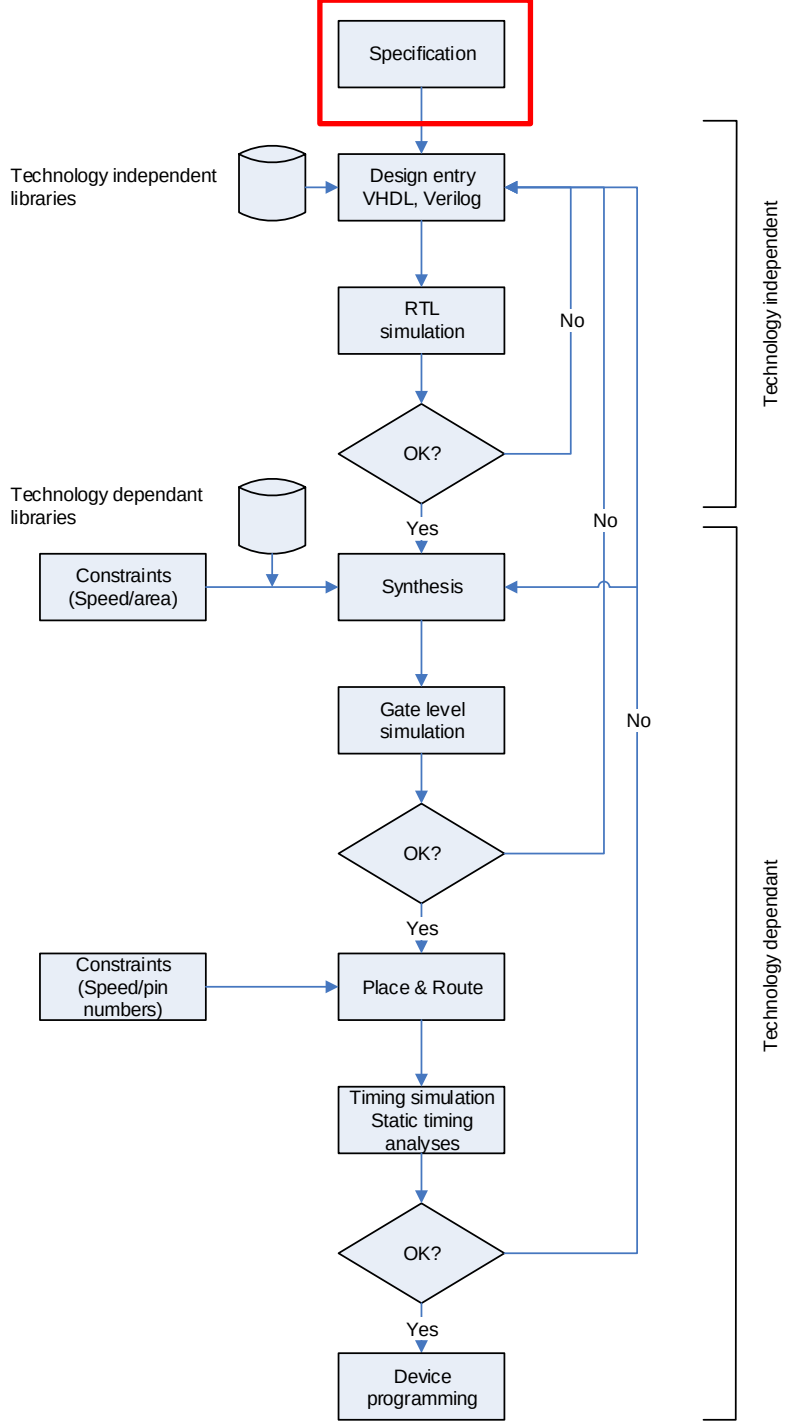
# Overview

- Digital design tools.
- Specification
- Design entry, synthesis and PAR
- Timing analysis
- Timing simulation
- Testing



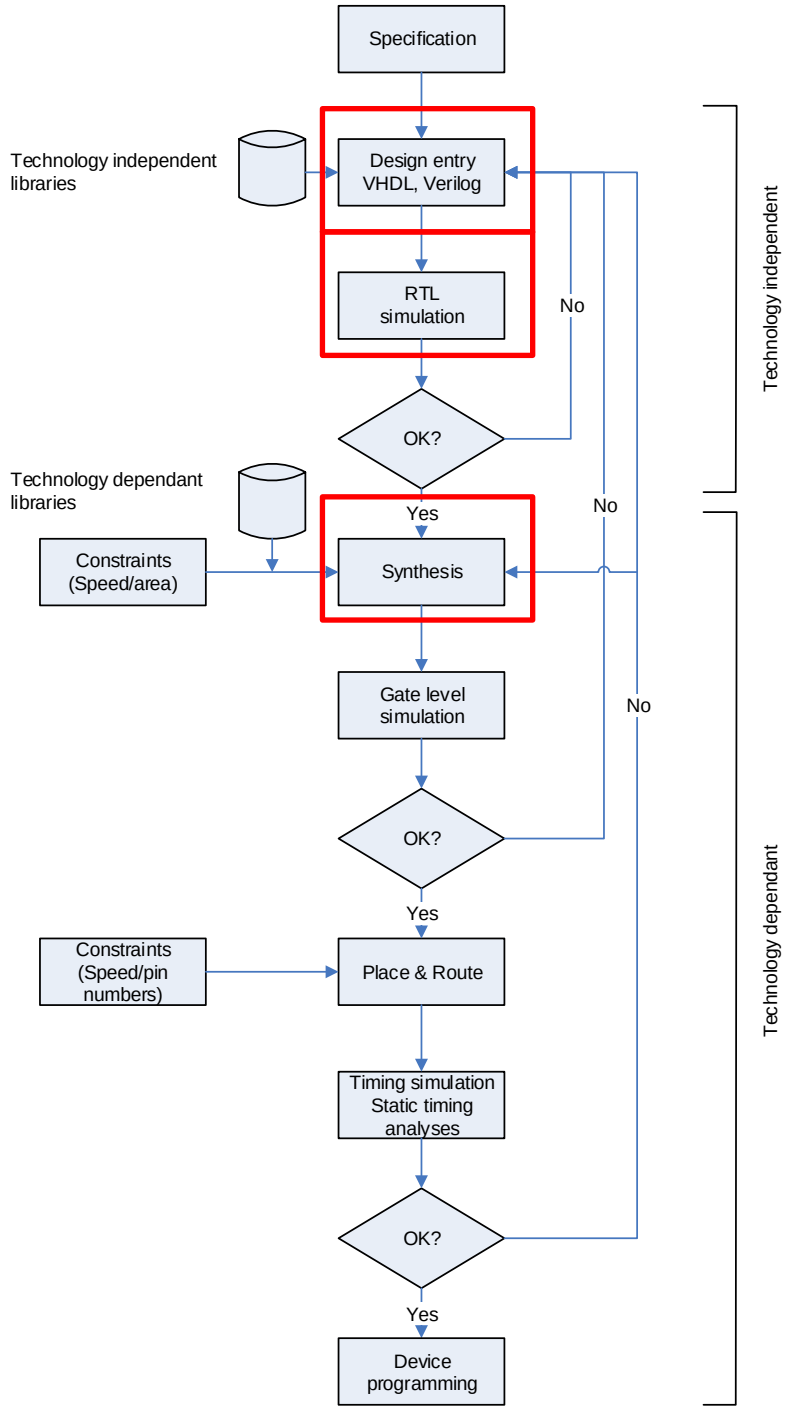
# Digital Design Flow: Specification

1. Define the problem
2. Draw a functional diagram
   - block diagram with major components and connections
3. Identify IO requirements
4. Identify necessary interface circuits
5. Decide on HDL (VHDL, Verilog, System C,…)
6. *Draw a program flowchart (ASM diagram)*
   - Defines how the design shall work logically.
   - By hand or using tools such as:
     - *Visio, Draw.io, Lucid chart, etc.*



# Digital Design tools…

- Design entry:
  - Use your favourite HDL text editor (Notepad++, Emacs, Vivado or Questa).
- Simulation (RTL, Gate Level, Timing)
  - Here: Typically using Questa (=Modelsim)
- Synthesis, Implementation, Programming
  - Vendor specific tools,
    - Here: Vivado by Xilinx
  - Also possible: Digilent tools for programming.

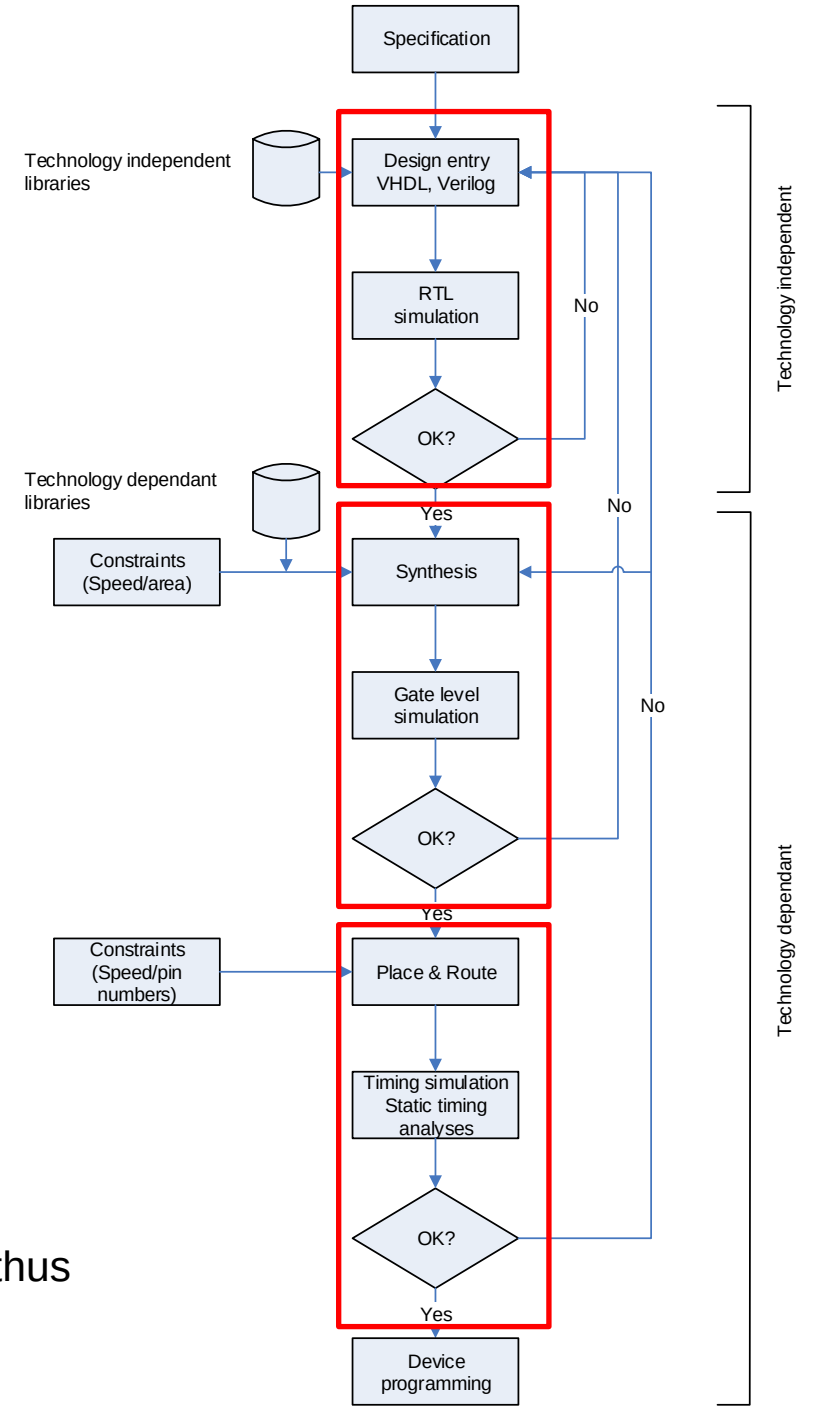


# Design entry, synthesis and PAR

- RTL = Register Transfer Level
  - RTL does not use specific gates or technology
  - Designs are *mostly* done in RTL
  - RTL  simulation can be used to verify logic function.

- Gate level synthesis
  - Technology specific gates are selected for all components in the design.
    - Typically a synthesizer will pick gates specific for the (FPGA) chip family we use.
  - Once we have a gate level design we can
    - calculate gate-, but not propagation delays
    - Simulate using gate delays.

- Place and route
  - After synthesis gates can be placed within a specific (FPGA) chip.
  - When place and route is performed propagation delays may also be simulated thus
  - We can do all timing simulation, including propagation delays.



# Static timing analysis

- Performed by EDA tools on synthesized or routed designs
- Will attempt to
  - find critical path(s) and
  - check if timing requirements (constraints) can be met.

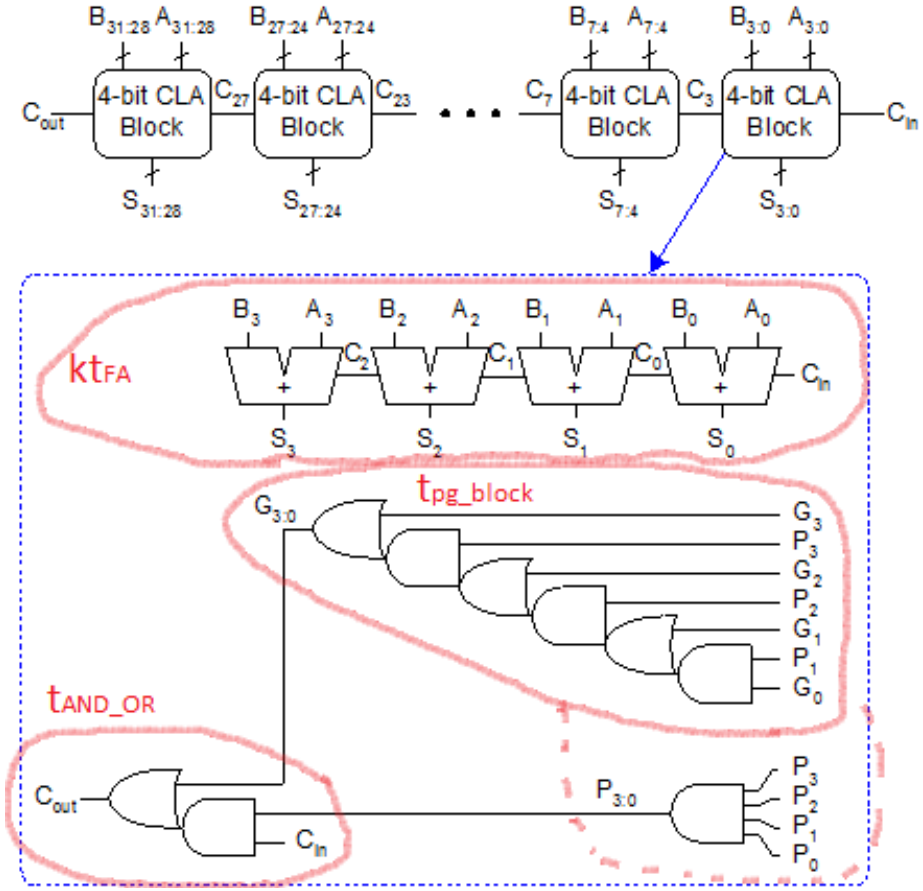
## IN2060: Carry-Lookahead Adder Delay

For *N*-bit CLA with *k*-bit blocks:

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{AND\_OR} + kt_{FA}$$

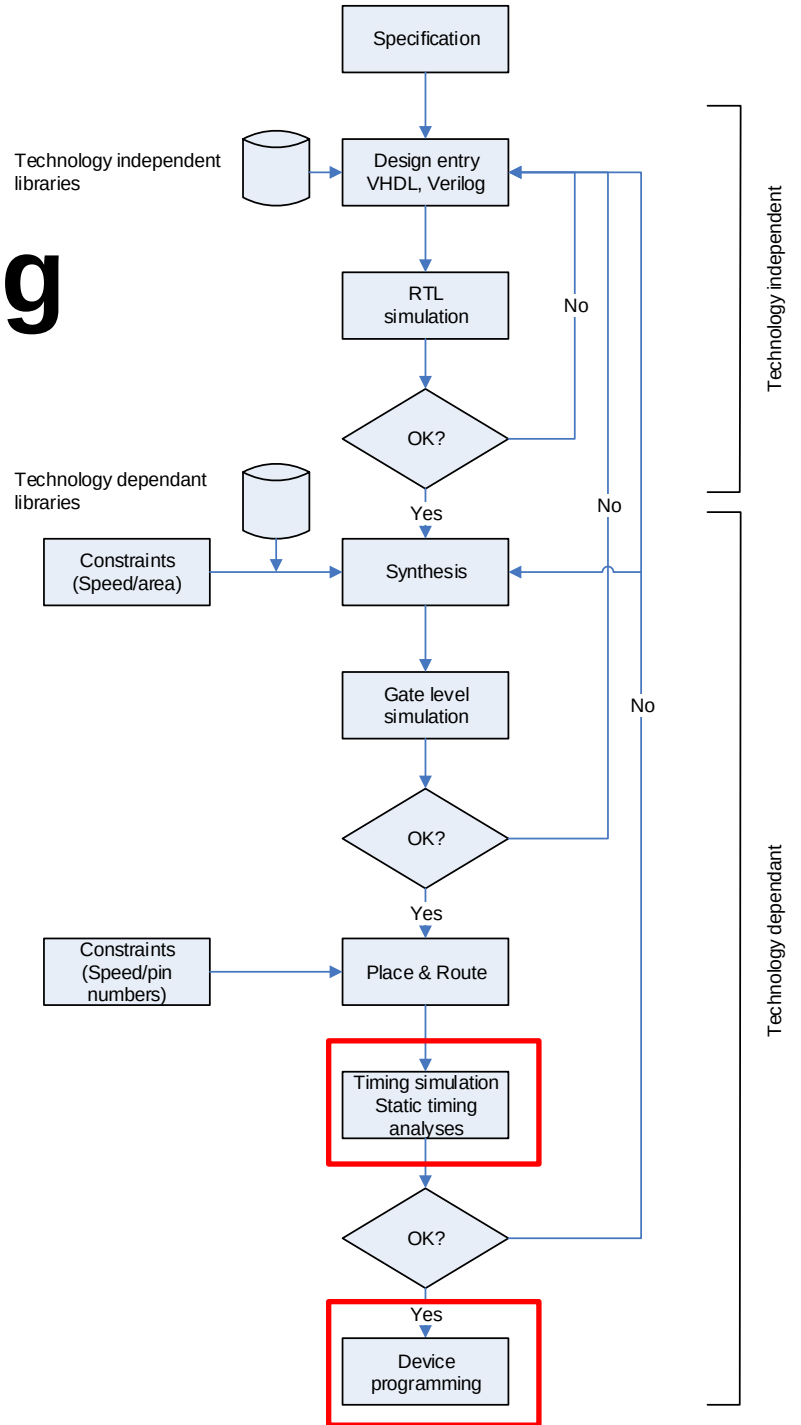
- $t_{pg}$ : delay to generate all $P_i$, $G_i$
- $t_{pg\_block}$ : delay to generate all $P_{i:j}$, $G_{i:j}$
- $t_{AND\_OR}$ : delay from $C_{in}$ to $C_{out}$ of final AND/OR gate in *k*-bit CLA block

An *N*-bit carry-lookahead adder is generally much faster than a ripple-carry adder for $N > 16$



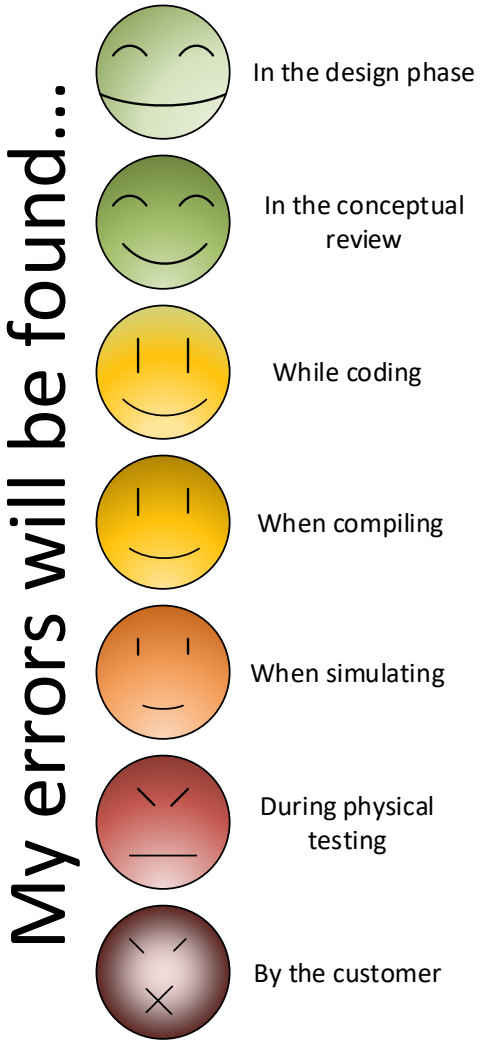
# Timing simulation, programming

- Simulating synthesized or routed designs
  - *Verification and test benches will be discussed further later…*
- Uses timing information for every component in use.
  - Requires much more resources than RTL simulation.
  - Can be slow for complex designs
    - Hence the option to simulate at gate level, before performing PAR.
- Device programming…
  - (Usually done from vivado, but third part tools *may* be used).



# Testing and verification

- «*Testing*» is to find *physical errors* in a *device*.
  - Built in self-tests
    - Ex: Memory tests in BIOS
  - Design for testability
    - Means that we design for physical testing.
    - We *may* touch this later in the course.
- «*Verification*» is to check the *design*
  - Simulation
    - Testbenches
      - HDL
      - Scripts
      - Co-simulation using normal programming languages
  - Analysis:
    - Compilation
    - Timing Analysis
    - Implementation reports
- *Spend more time in early phases!*
  - Avoid spending *much more* time fixing bugs later



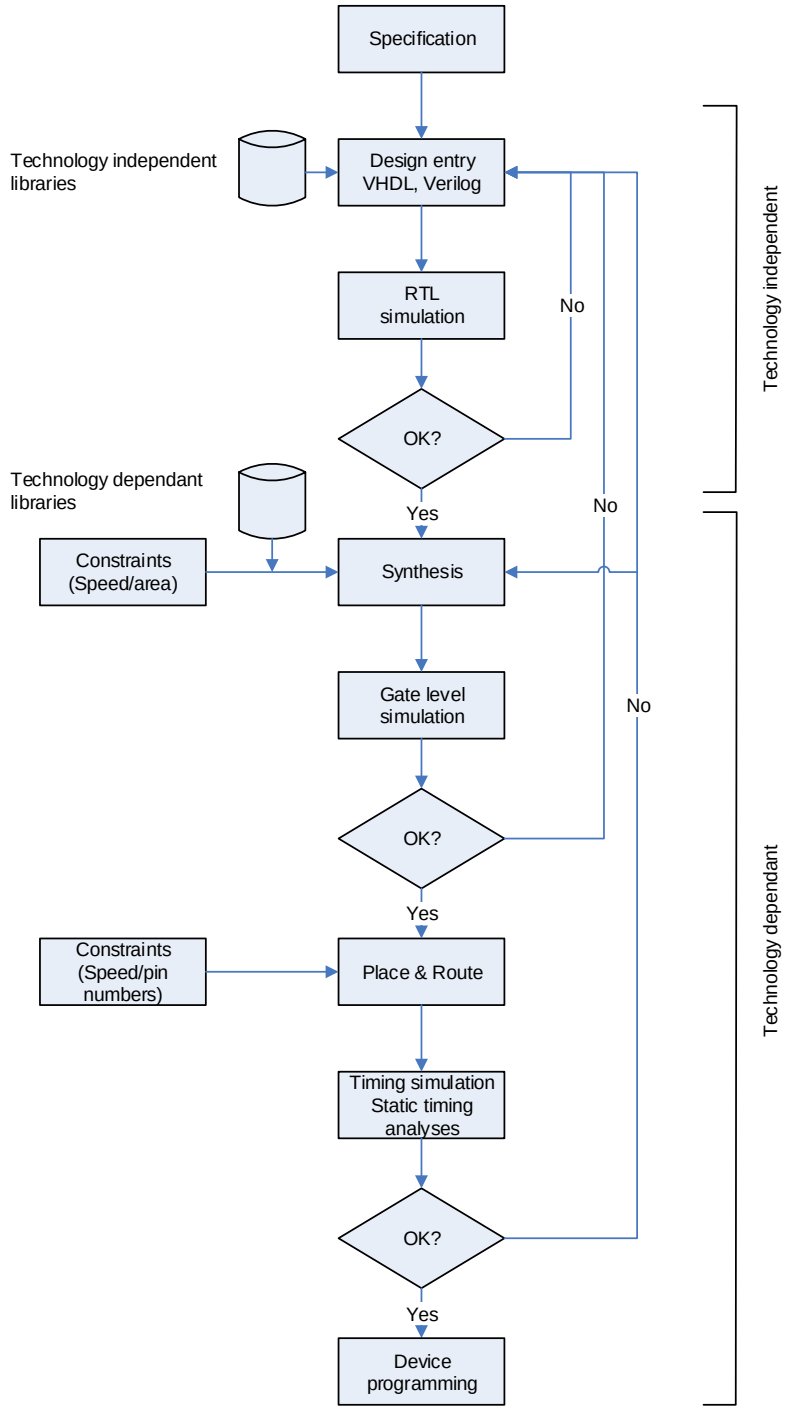
# Introduction to course hardware and software tools

- Zedboard
- Questa
- Vivado

- ROBIN wiki:
  https://robin.wiki.ifi.uio.no/Hovedside
  - Software
    - FPGA tools
      https://robin.wiki.ifi.uio.no/FPGA_tools
- Cook book and ZedBoard documentation
  - Canvas – IN3160
    - Cookbook_v3_5.pdf
    - ZedBoard HW UG vX_X.pdf
  - Zynq intro video:
    https://www.xilinx.com/video/soc/zedboard-overview-featuring-zynq.html

# Digital Design tools…

- Design entry:
  - Use your favourite HDL text editor (Notepad++, Emacs, Vivado or Questa).
- Simulation (RTL, Gate Level, Timing)
  - Questa (=Modelsim)
  - GHDL
- Synthesis, Implementation, Programming
  - Vendor specific tools…
    - Here: Vivado by Xilinx



# Simulation and test benches

Simulation can be run several ways:

1. Manually setting inputs and specifying time intervals in the GUI or console
   - Tedious and not really practical at all
   - *Normally this is only done only initially.*

2. To make scripts (tcl for Questa) in a separate (.do) file.
   - The *script commands will be added to the console during manual use, and can be copied as text into a .do file.*
   - setting up the simulation windows can be done reusing script commands.

3. Using test benches written in HDL (VHDL)
   - possible in combination with running scripts
   - VHDL can be used to generate code for applying test vectors sequentially to the inputs of an entity for simulating.
   - *Test bench code is SW even if it is written in an HDL* (not synthesizable)
   - VHDL has built in test-specific attributes
4. Using Co-Simulation (cocotb)
   - runs simulation and coroutines in parallell
     - Environment switches back and forth between coroutines and simulation
   - Test vectors are generated in software coroutines
   - Checks and reporting is done in coroutines
   - Python can be used for test-benches
     - not built for hardware testing initially
     - scale better with large design and complex testing
     - non-HDL-specific issues has (way) better support

# Suggested reading, Mandatory assignments

- D&H:
  - 1.4 p 11-13
  - 1.5 p 13-16
  - 1.6 p 16-17
  - 2.1 p 22-28
  - 2.2 p 28-30
  - 2.3 p 30-34
  - 3.1-3.5 p 43-51 = repetition (known from previous courses)

- Oblig 1: «Design Flow»
  - See canvas for further instruction.

Note: Some of this content will be covered in depth in later lectures.
- *Read this to familiarize yourself with content, form and language.*