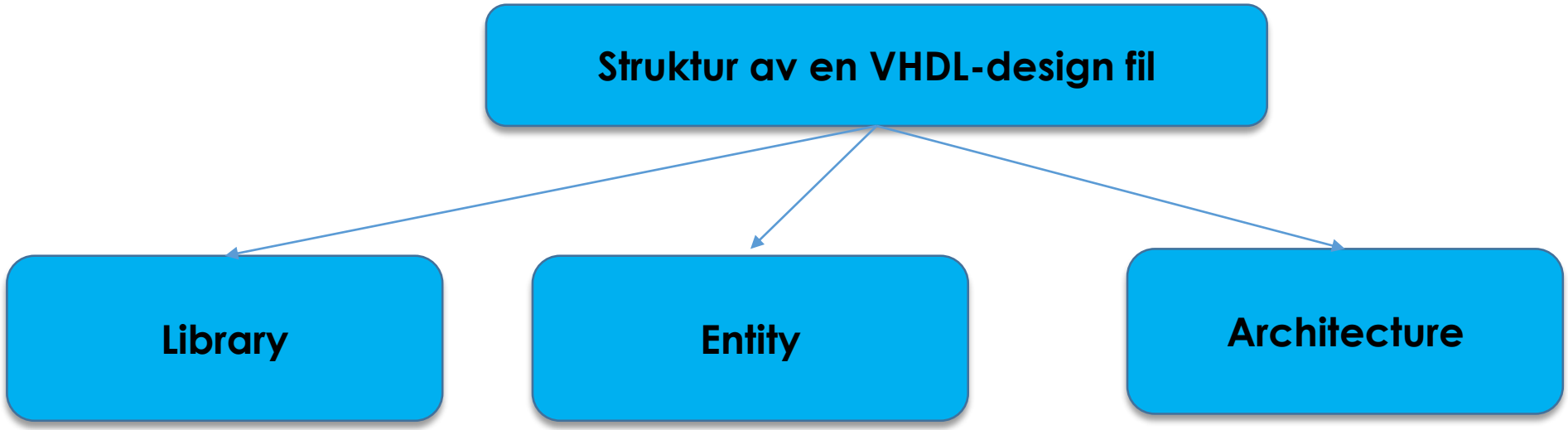
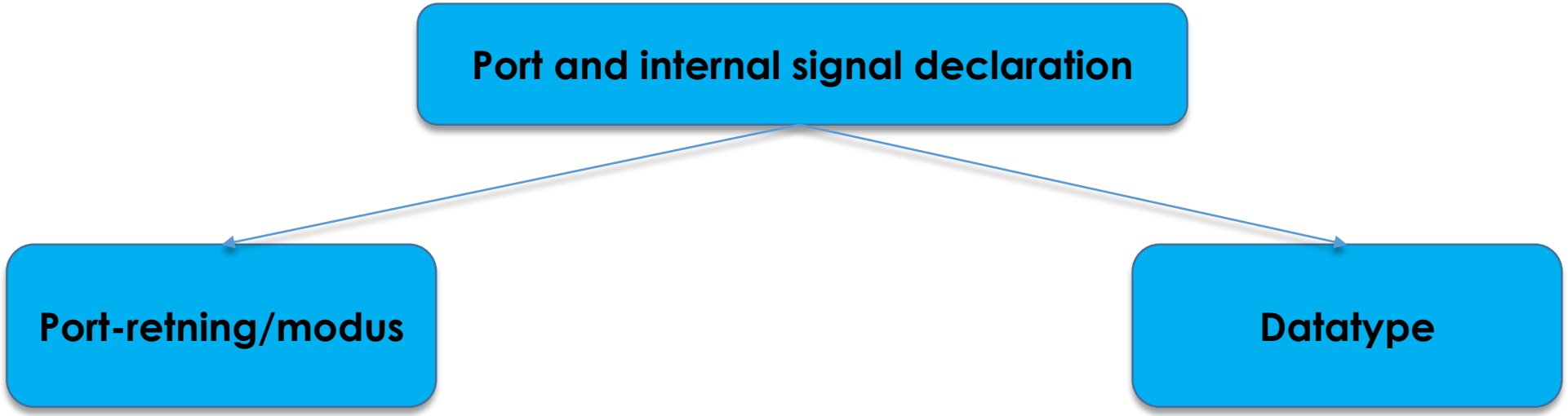
Forelesning 02

Kapittel 2: Oversikt over FPGA og EDA

Lecture 01+ Lab01+ Oppgave 01: Oppsummering



Lecture 01+ Lab01+ Oppgave 01: Oppsummering



Lecture 01+ Lab01+ Oppgave 01: Oppsummering

- Flow Design
- Truth Table/Sannhetstabell
- K-diagram
- Boolsk uttrykk

Lecture 01+ Lab01+ Oppgave 01: Oppsummering

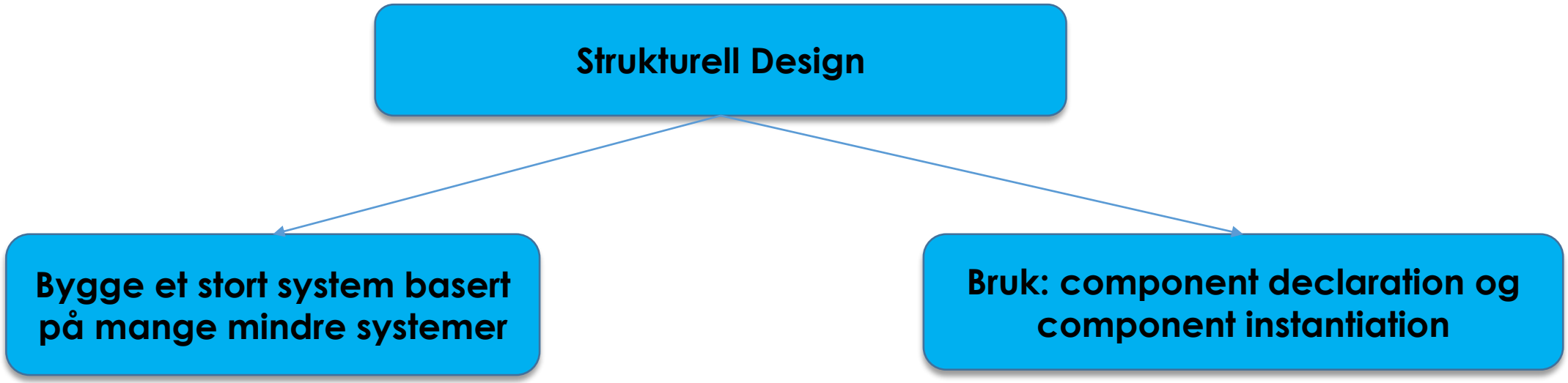
Setninger

Concurrence statement/Samtidig setninger

Alle setninger slutter med semicolon (;)

Internal signal declaration har ikke retning!!!!

Lecture 01+ Lab01+ Oppgave 01: Oppsummering



Lecture 01+ Lab01+ Oppgave 01: Oppsummering

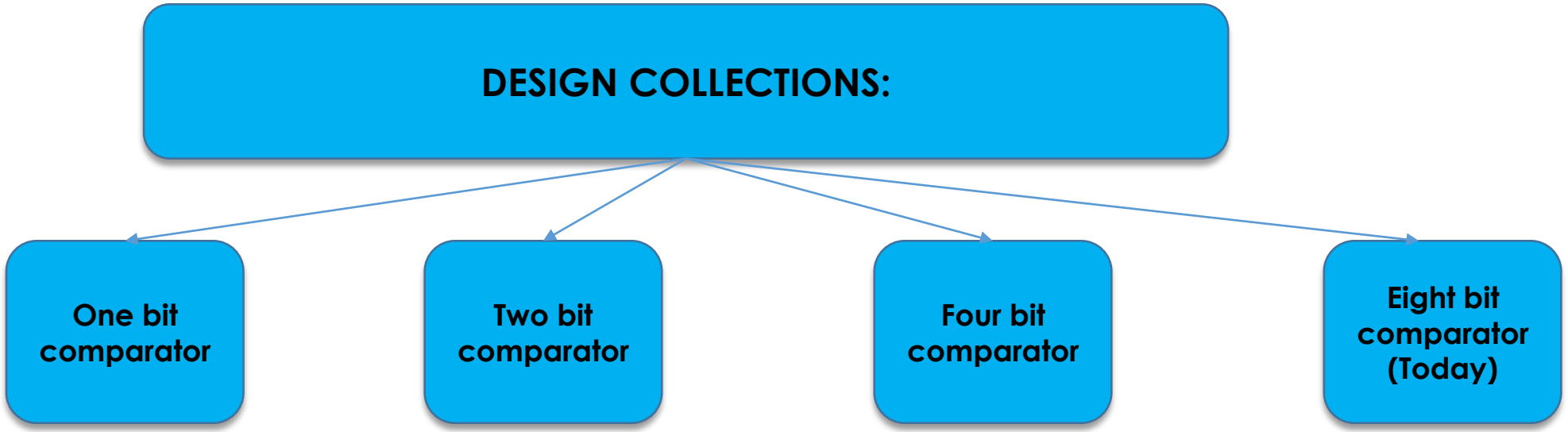
TestBench

Testbench har ofte ikke inngang og utgangssignaler

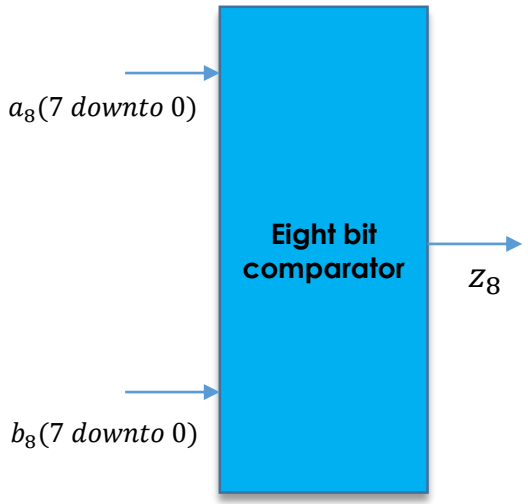
4 deler i en testbench

- **Component declaration for design vi ønsker å teste**
- **Internal signal declaration**
- **Component Instatiation: Unit Under Test**
- **Testvector generator**

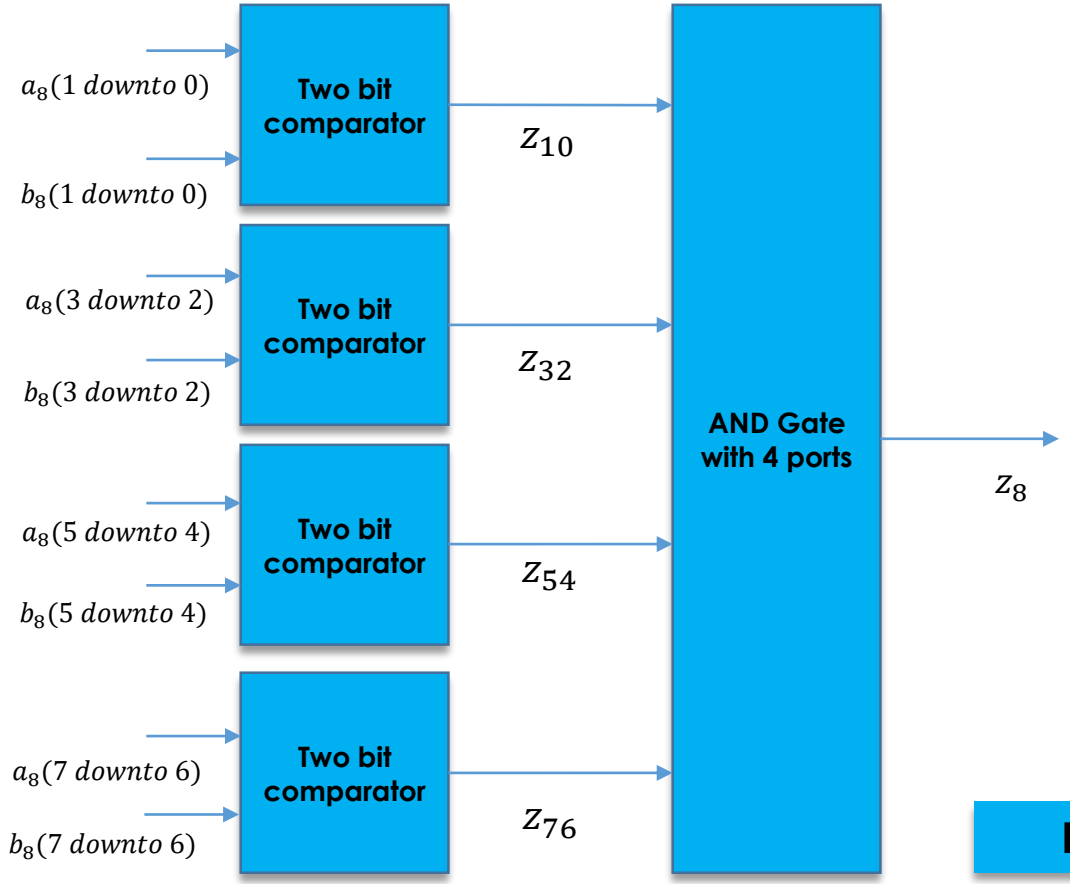
Lecture 01+ Lab01+ Oppgave 01: Oppsummering



Design 8 Bit Comparator Using Structural Design Methods



Top Block



Detailed Design

Design 8 Bit Comparator Using Structural Design Methods

- Create a folder on your desktop – Name: **lecture02**
- **Download** two bit comparator design from canvas
- Create a **new project** with Xilinx Vivado – Project Name: Lecture02_8bits_comp_struct_design
- **Add** the two-bit comparator design into the project
- **Create** eight-bit comparator design within your project
- **Implement** the structural design within the architecture of 8-bit comparator
- **Create** testbench to verify the functionality of the 8-bit comparator design

Design 8 Bit Comparator Using Structural Design Methods

- **Generate test vector using loop**

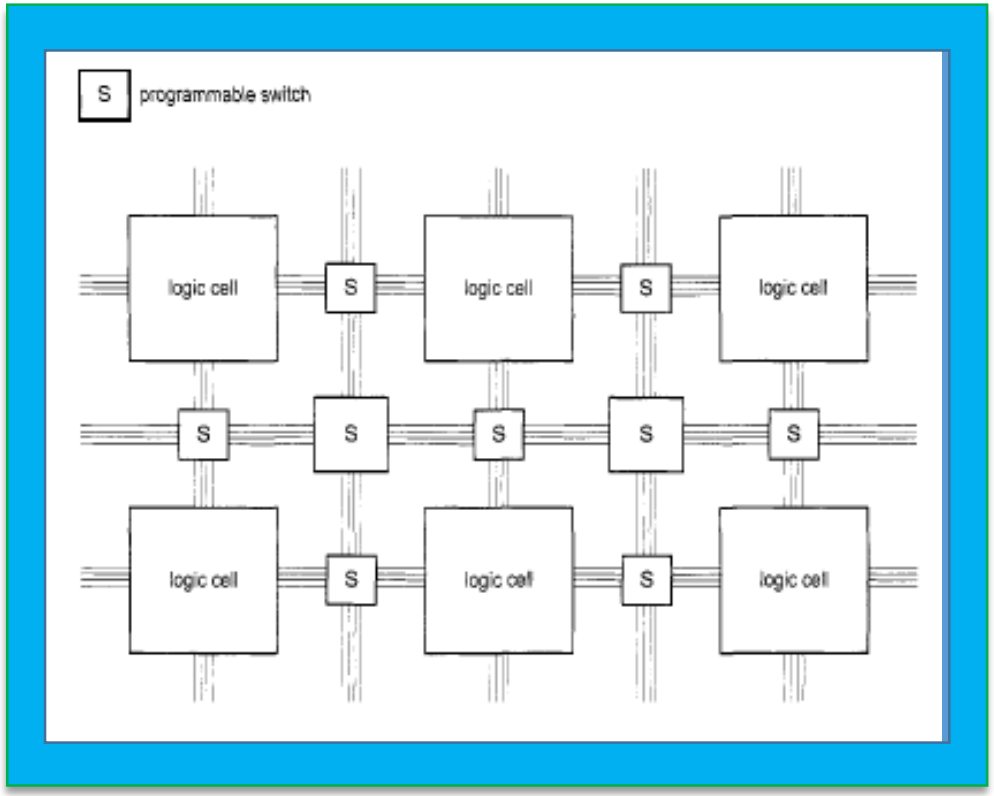
```
-- Stimulus process
stim_proc: process
begin
   for a_int in 0 to 3 LOOP
      for b_int in 0 to 3 LOOP
         a_tb <= std_logic_vector(to_unsigned(a_int,2));
         b_tb <= std_logic_vector(to_unsigned(b_int,2));
         wait for 100 ns;
      end loop;
   end LOOP;
```

Innledning

- FPGA: **F**ield **P**rogrammable **G**ate **A**rray
- EDA: Electronic Design Automation
- Det å utvikle et stort FPGA-system involvert mange komplekse utforminger og optimaliseringer. Vi trenger verktøy som hjelper å **automatisere** noen oppgaver
- Vi bruker Vivado-webversjon for å **syntetisere** og **implementere** våre design
- Vi bruker **Vivado simulator** for **simulering**
- Vi bruker **BASYS-3** (med ATRIX-7 Xilinx FPGA-brikke) kort til å utføre laber, oppgaver.

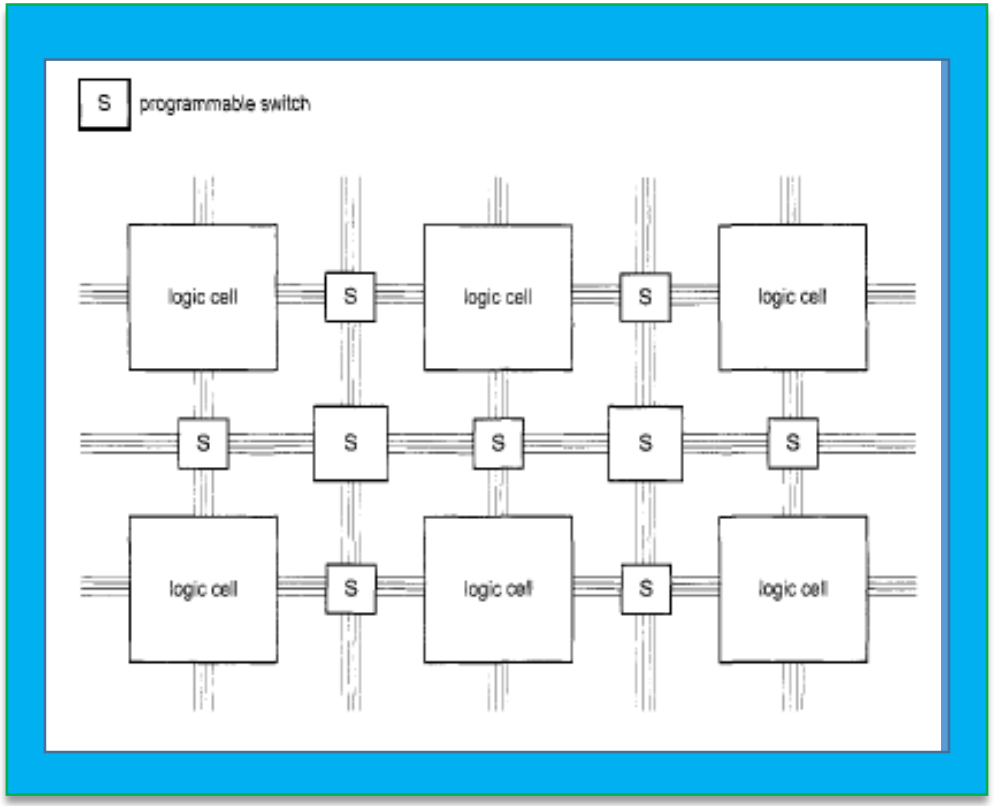
Generelt FPGA-Utstyr

- FPGA: field programmable gate array som er **et logisk utstyr** og det inneholder en to-dimensjonal array av generelle **logiske celler** og **programmerbar svitsjer**
- Struktur av et FPGA-utstyr er fremstilt



Generelt FPGA-Utstyr

- En logisk celle kan **konfigureres** eller **programmeres** til å utføre en viss funksjon
- En programmerbar svitsj kan **tilpasses** til å gi forbindelser blant de logiske cellene
- Et **egendefinert** design kan implementeres med å spesifisere den funksjonen på hver logisk celle og å sette selektivt forbindelsen av hver programmerbar svitsj
- Når et design blir **syntetisert** og **implementert**, bruker vi en enkel kabel til å nedlaste de logiske cellene og svitsj konfigurasjon til FPGA-utstyr og slik får vi en egendefinert krets.
- Siden denne prosessen kan **utføres** i «**feltet/the field**» ,ikke i fabrikasjonsfabrikk, blir utstyret kjent som «Field programmable»

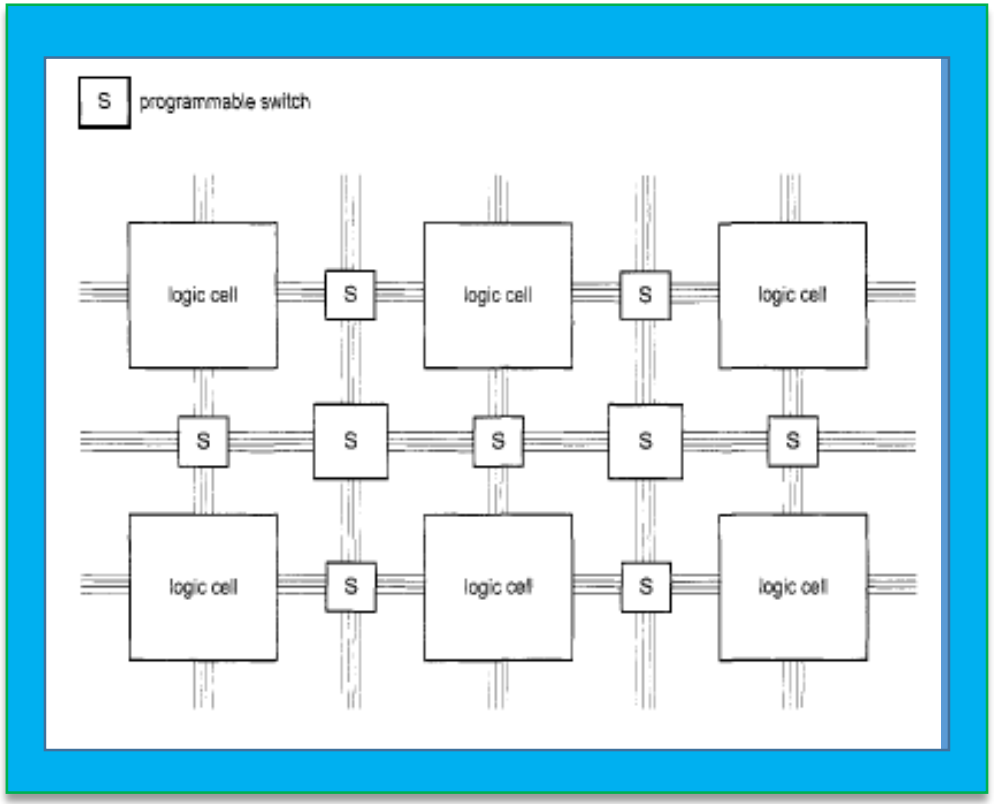


LUT-basert Logisk Celle

- En logisk celle inneholder ofte en liten konfigurerbar **kombinasjonskorts** med **D-type flip-flop**.
- Den vanligste metoden til å implementere en konfigurerbar kombinasjonskorts er **look-up table** (LUT)
- En **n-inngangs LUT** kan betraktes/ vurderes som et lite **2**n by 1 minne**
- Hvis vi er flink til å skrive innhold til minnet, kan vi bruke LUT til å implementere **hva som helst** n-inngang kombinasjonsfunksjon
- En eksempel or 3-inngangs LUT er vist

$$a \oplus b \oplus c$$

- Utgangssignal/utmatingssignal kan brukes **direkte** eller **lagrer** på D-FF

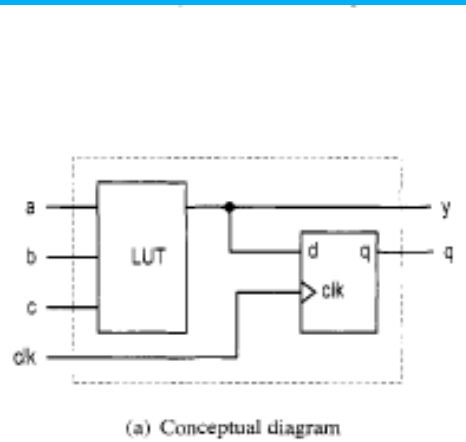


LUT-basert Logisk Celle

- En eksempel or **3-inngangs LUT** er vist

$$a \oplus b \oplus c$$

- Utgangssignal/utmatingssignal kan brukes direkte eller lagrer på D-FF



(a) Conceptual diagram

a b c	y
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

(b) Example table

Figure 2.2 Three-input LUT-based logic cell.

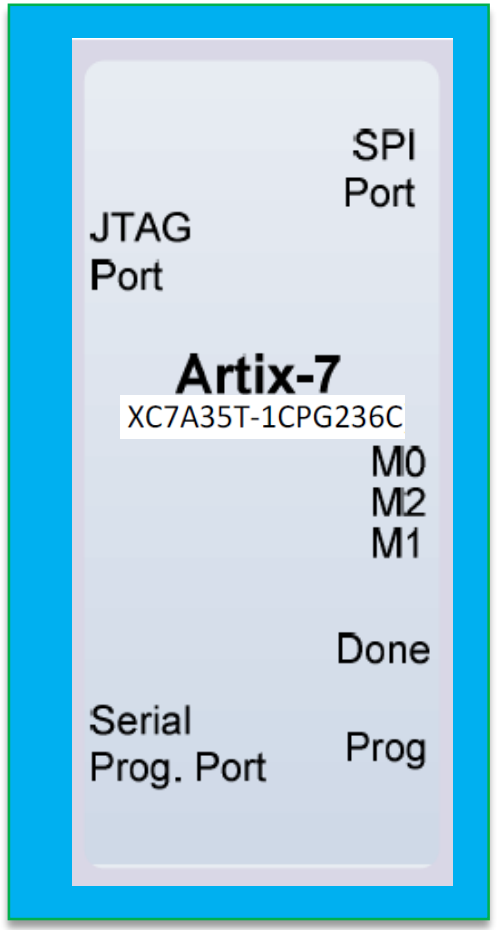
Macro Celle

- De meste FPGA utstyrene legger inn noen visse **macro cells eller macro block**
- De er designet og **fabrikert** på transistorsnivå og deres funksjonaliteter er **motsatt/komplement** de generelle logiske cellene
- Alminnelig brukt macro cells inkluderer: **memory blocks**/ minneblokk, **Kombinasjonal multiplier**, **klokkeforvaltningskrets**, og **I/O grensesnitts kretser**
- **Avansert** FPGA utstyr kan enda ha en eller flere forfabrikkerte **prosessorer**

Oversikt over ATRIX-7 utstyr

The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance, and more resources than earlier designs. Artix-7 35T features include:

- 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops);
- 1,800 Kbits of fast block RAM;
- Five clock management tiles, each with a phase-locked loop (PLL);
- 90 DSP slices;
- Internal clock speeds exceeding 450MHz;
- On-chip analog-to-digital converter (XADC).



BASYS-3 Kort

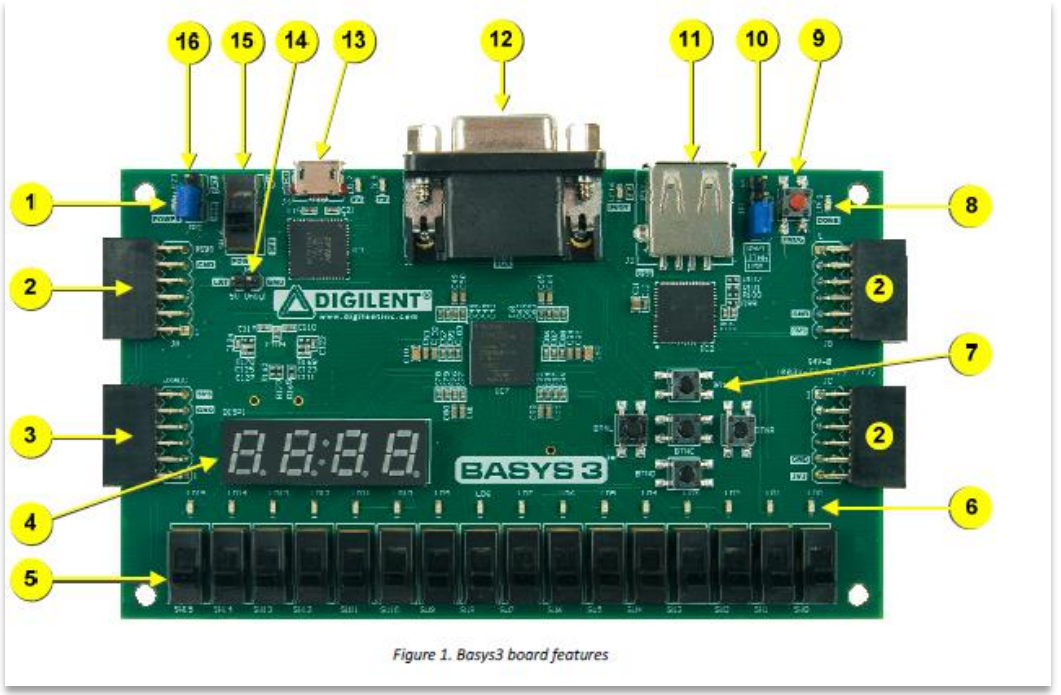


Figure 1. Basys3 board features

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

FPGA-Utviklingsflyt/Flow

- Prosessen på **venstre**
  - Spissfindighet/refinement og programmeringsprosess som omformer fra **abstrakt VHDL-kode** til **konfigurasjon** på utstyrsnivå og nedlaster designet til FPGA-utstyr
- Prosessen på **høyre**
  - **Valideringsprosess** som sjekker om systemets **funksjonaliteter** og **ytelsesmålet**

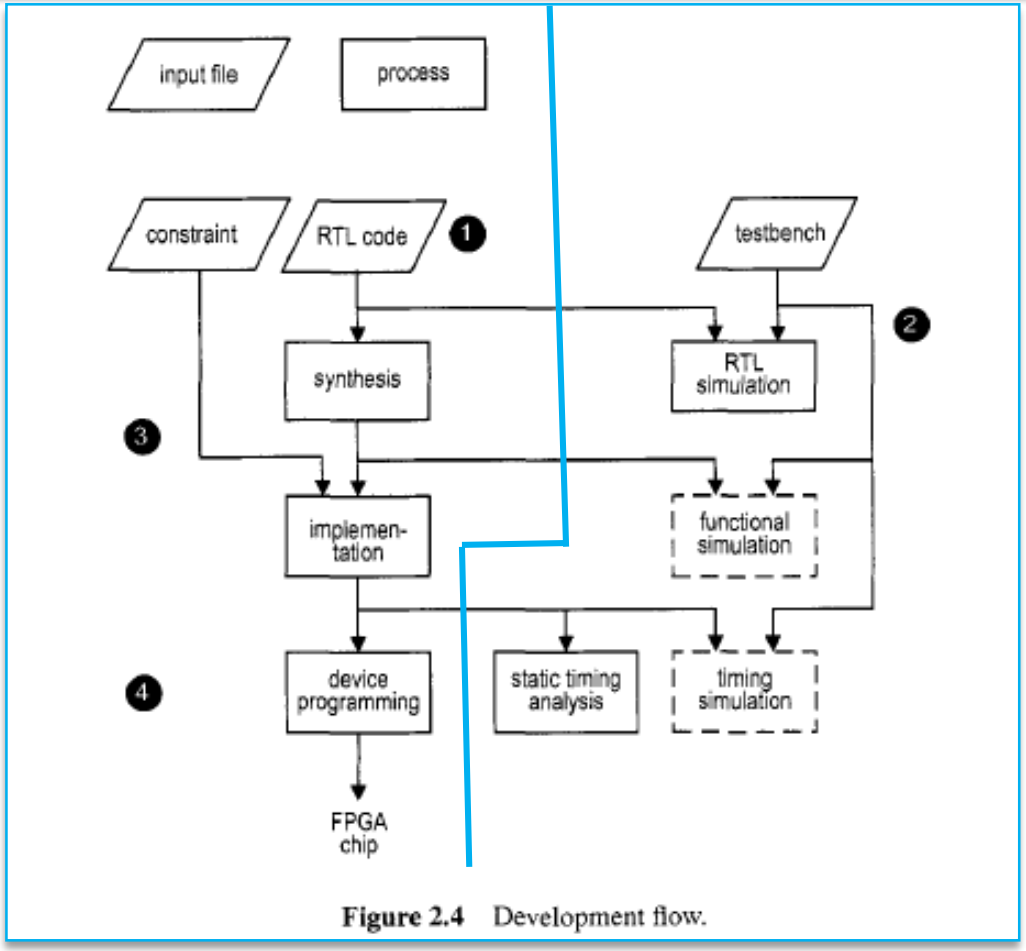


Figure 2.4 Development flow.

FPGA-Utviklingsflyt/Flow

Trinn 1: Konstruere systemet og produsere-VHDL-filerr. Vi trenger kanskje en «constraint » fil

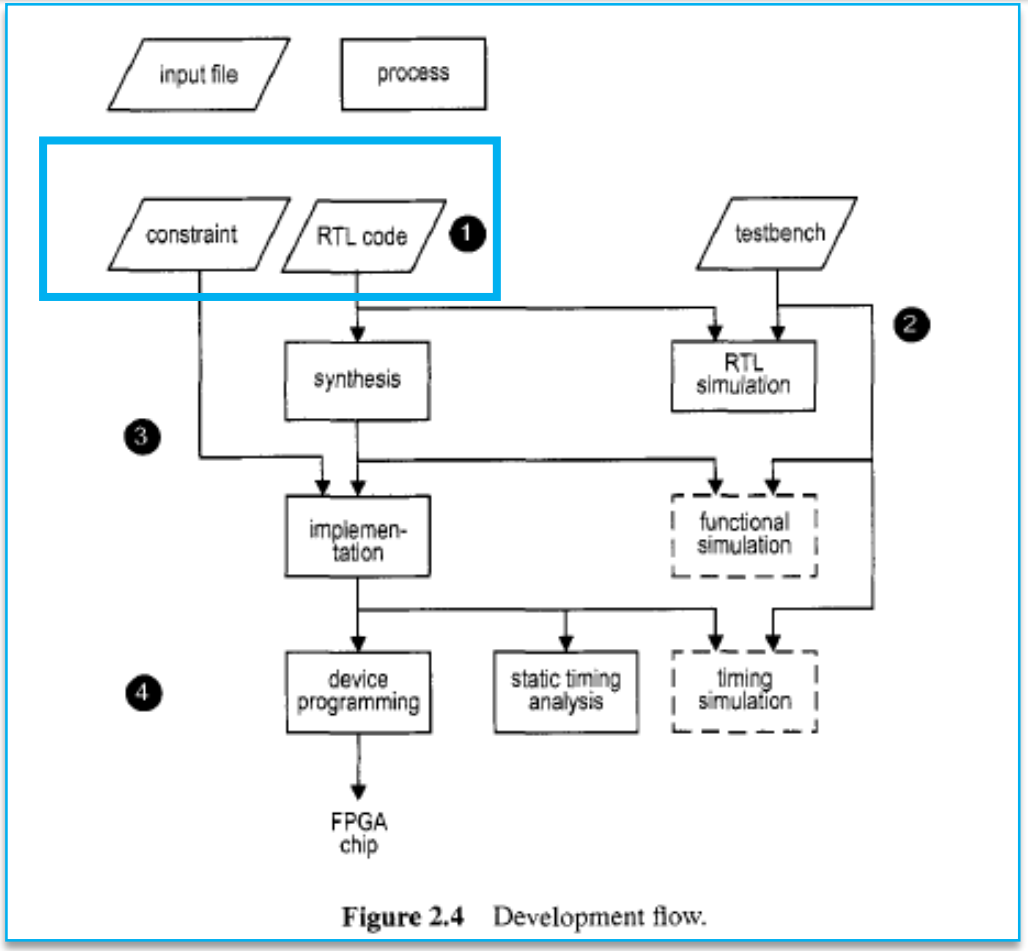


Figure 2.4 Development flow.

FPGA-Utviklingsflyt/Flow

- Trinn 2: Utvikle en testbenk i HDL og utføre RTL simulering. RTL betyr at HDL-kode blir utført på register-nivå

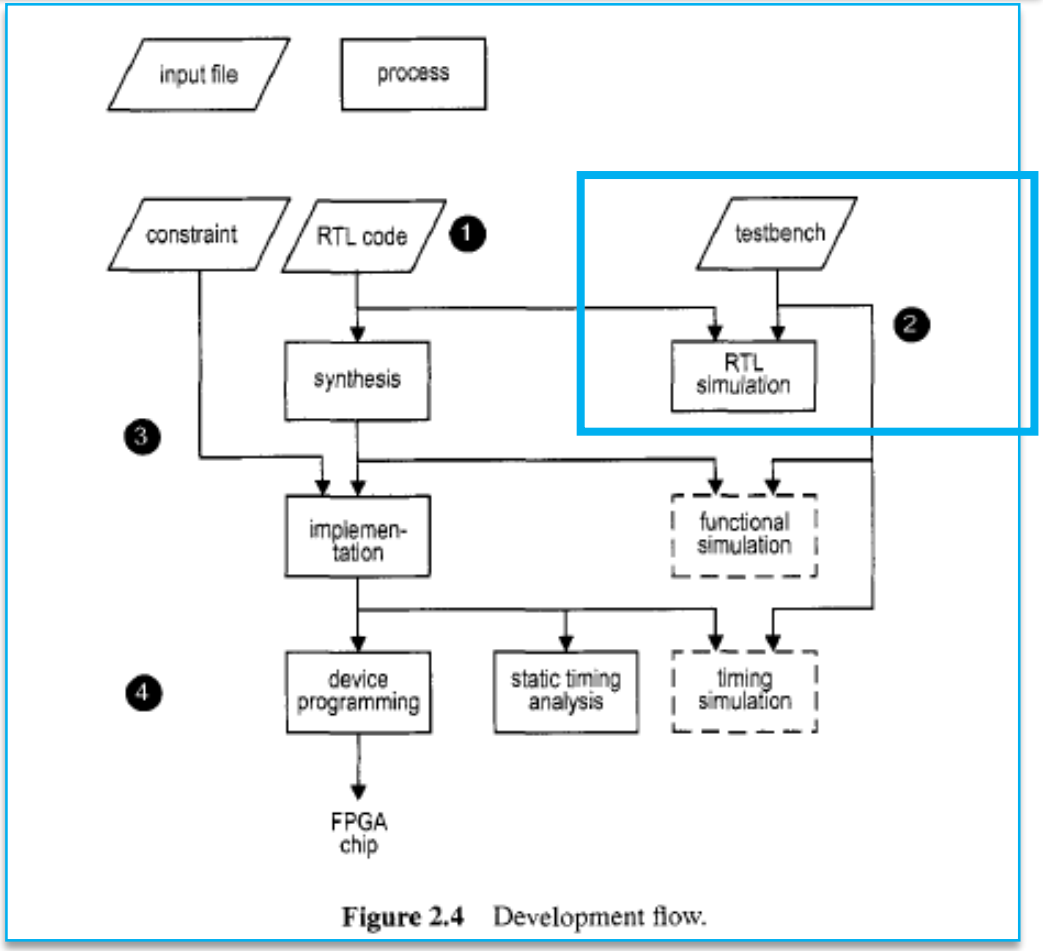


Figure 2.4 Development flow.

FPGA-Utviklingsflyt/Flow

- Trinn 3: Utføre **syntese** og **implementering**

  Trinn 3a) Syntese: Programvare omformer HDL-konstruksjoner til generelle port-nivåkomponenter slik som: AND, OR og FF

  Trinn 3b) Implementering **består** tre faser: **translate/tolkning**, **map/kartlegging**, og **route/rutering**

  - **Tolkning**: sammensette multiple designfiler til en nettliste

  - **Kartlegging**: teknologikartlegger fra generelle porter i nettliste til FPGA sine logiske celler og IOBs

  - **Plassering** og **Rutering**: Fysisk layout innenfor FPGA-brikken. Cellene blir plassert i fysiske lokasjoner og ruteringer å forbinde signaler blir bestemt

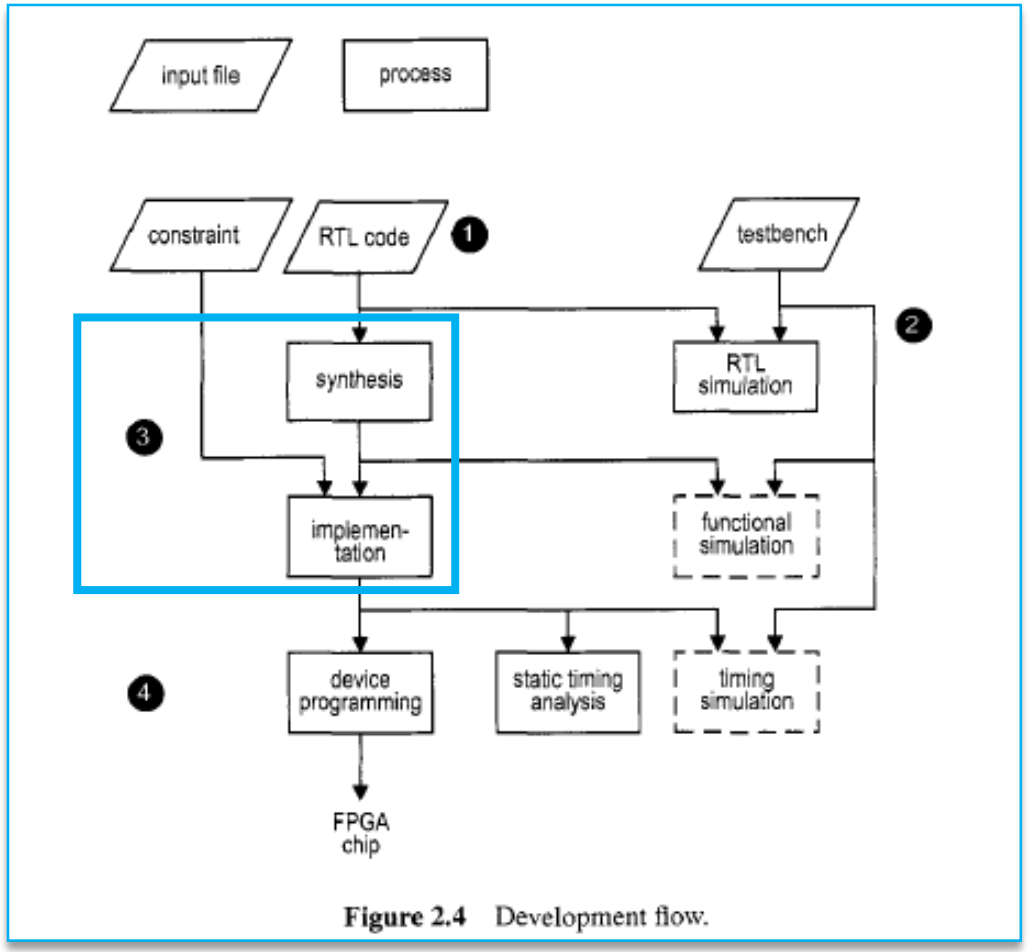


Figure 2.4 Development flow.

FPGA-Utviklingsflyt/Flow

- Trinn 4: Generere og nedlaste programmeringsfil (bit-fil)

  - Konfigurasjonsfil(**bit-fil**) blir generert i overstemmelse med den endelige **nettlisten**. Denne filen blir nedlastet til FPGA-utstyr serielt for å konfigurere de logiske cellene og svitsene. Den fysiske kretsen kan følgelig verifiseres

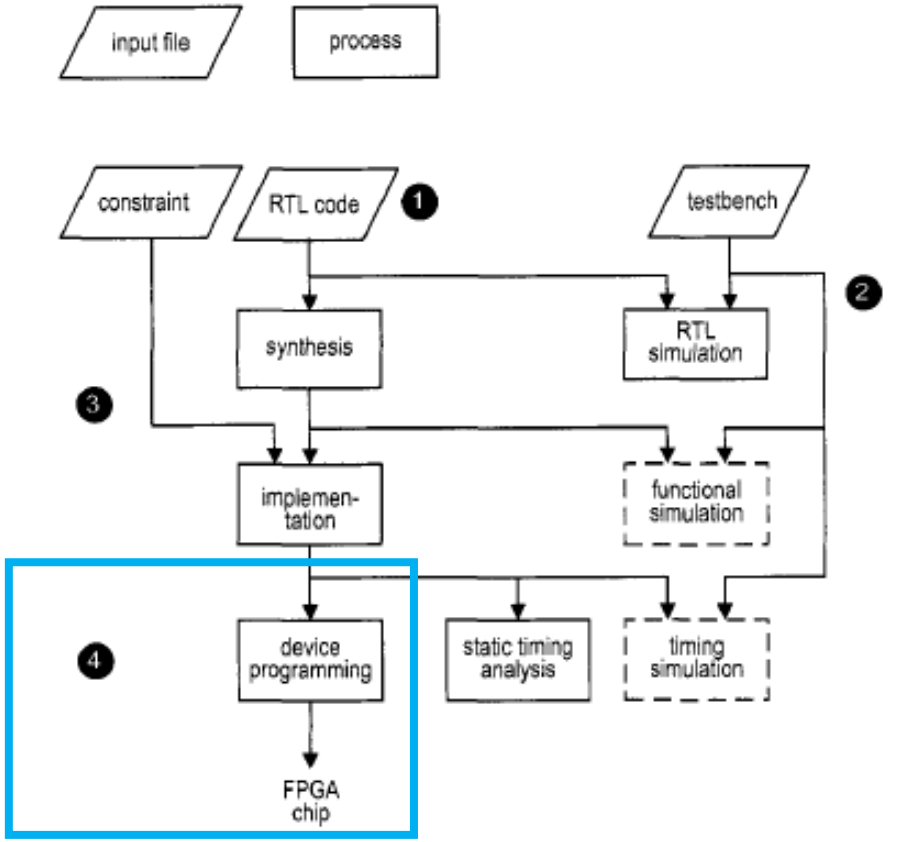


Figure 2.4 Development flow.

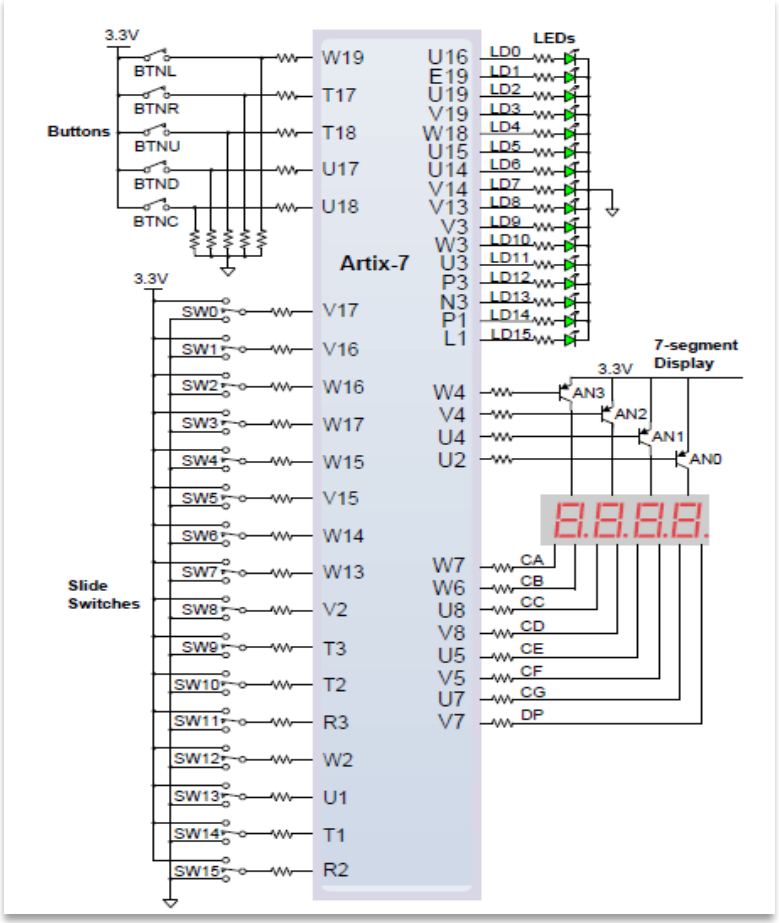
Constraint File

- **Den constraint filen for BASYS-3**

```
1  ## This file is a general .xdc for the Basys3 rev B board
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used ports (in each line, after get_ports) ac
5
6  ## Clock signal
7  #set_property PACKAGE_PIN W5 [get_ports clk]
8      #set_property IOSTANDARD LVCMOS33 [get_ports clk]
9      #create_clock -add -name sys_clk_pin -period 10.00 -wavef
10
11 ## Switches
12 #set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
13     #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
14 #set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
15     #set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
16 #set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
17     #set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
```

Pin Number

- I/O PINS in BASYS-3
- Se design manual: Basys3_RM.pdf



XDC File From Digilent

- I/O PINS in BASYS-3
- Bruk av xdc-fil levert av Digilent
  - Vi kopierer innholdet av xdc-filen og aktiverer PIN som vi vil bruker i vårt prosjekt

```
6   ## Clock signal
7   #set_property PACKAGE_PIN W5 [get_ports clk]
8       #set_property IOSTANDARD LVCMOS33 [get_ports clk]
9       #create_clock -add -name sys_clk_pin -period 10.00 -wavefo
10
11  ## Switches
12  #set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
13      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
14  #set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
15      #set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
```

Endrer bare signalnavn og indeks på 2 plasser

Merk: Case sensitive i XDC fil