# IN2060, Høst 2026

## Kapittel 4: HDL

Philipp Häfliger <hafliger@ifi.uio.no>

10. september 2026

# Så langt

- Kombinatorisk logikk
- Sekvensiell logikk

Med skjematikk, sannhetstabeller, state transition graphs, tilstandstabeller.
I dag: beskrive med HDL

# Innhold

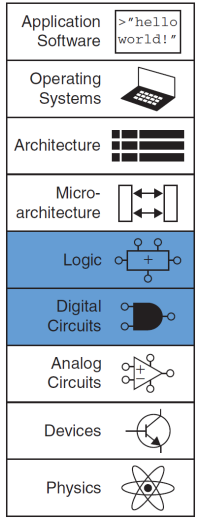
1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# Innhold


1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# Kapittel 4 Delmål

- Kjenne til forskjell mellom HDL og programmeringsspråk
- Kunne modifisere HDL-kode og forstå virkemåten til enkle digitale kretser bygget med HDL
- Kunne utføre enkle simuleringer med simuleringsverktøy for HDL
- Få øvelse i å benytte verktøy for digital design



# Ressursser for delmålet

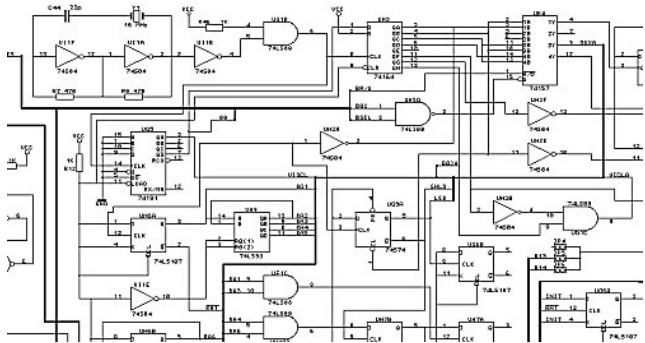
- Forelesning & selvstudium (lærebok)
  - Grunnlegende VHDL
  - Syntaks
  - Abstarksjonsnivåer for å beskrive ’architecture’ av en ’entity’: ’structural’, ’RTL’, ’behavioral’
  - Testbenker
- Ukeoppgaver
  - Øvelse i å skrive og forstå (V)HDL
- Oblig
  - Bruke verktøyene, simulere, teste
- Gruppetimer
  - Gjenomgang, forklaringer, flere eksempler

# Innhold


1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# Hardware Definition Language (HDL)

- beskriver forbindelser og beregninger av signaler
- alt skjer i parallel[a] selv om det beskrives som en sekvens av 'bygge-instruksjoner' (≠programmeringsspråk)



[a] unntak: testbenker

# HDL Workflow

1. HDL beskrivelse av kretsfunksjon
2. CAD verktøy generere 'netlist' (byggeblokker og forbindelser)
3. Funksjonell simulering/verifisering
4. Konfigurasjon til spesifikk programmerbar logikk (PL) eller utlegg for applikasjons-spesifikk krets (ASIC)
5. Mer realistisk simulering med portforsinkelser, timing analyse
6. Testing på fysisk krets

PL (FPGA, CPLD, PLD, PAL, PLA, ...): Her konfigureres det forbindelses-svitsjer mellom eksisternede logikk porter fra netliste
ASIC: Her genereres det utlegg med standard porter og plassering og routing fra netliste

# HDL eksempler

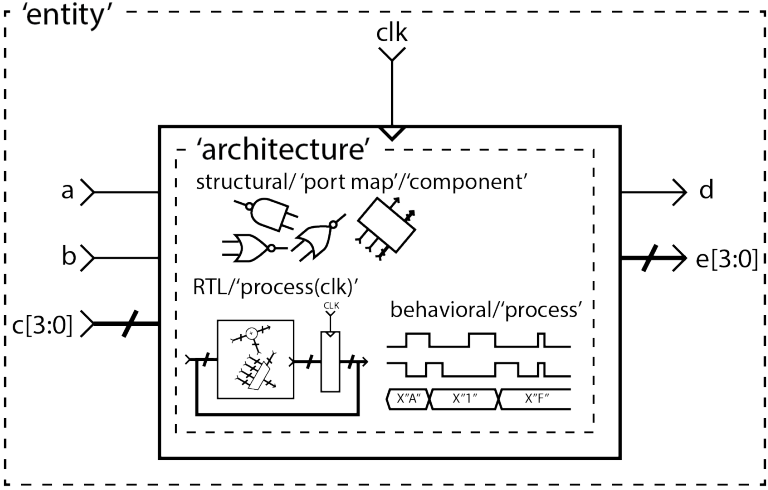
- **VHDL** (VHDL 2008): brukes mest i akademia, eldre en Verilog.
- **Verilog** (SystemVerilog): brukes mest i industrien.

Vi skal introdusere **VHDL i IN2060** (brukes i andre kurs på IFI (IN3160)).
Prinsipene skal være likt mellom forskjellige HDL.

# Innhold

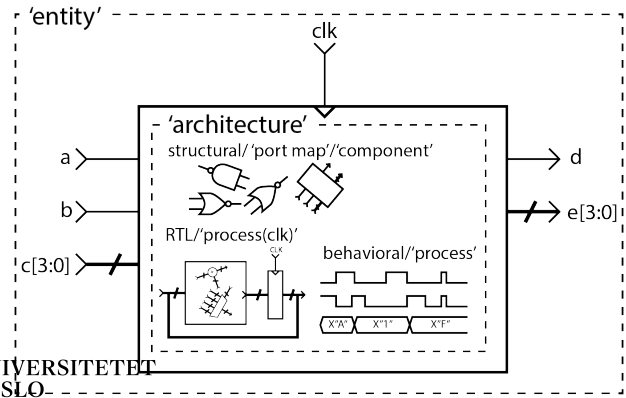
1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
   - entity
   - architecture, structural
   - architecture, register transfer language
   - architecture, behavioural
   - logic and arithmetic operands
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# VHDL modul illustrasjon



# VHDL modul syntax

**Se oblig 1:** Hver modul legges i egen fil og kompileres med `ghdl -a -std=08 <vhd-fil>`. Testbenker simuleres med `ghdl -r -std=08 <tb modul> -wave=wave.ghw`. Resultater kan ses på med `gtkwave wave.ghw`




```
library ieee;
use ieee.std_logic_1164.all; -- lo
-- use ieee.numeric_std.all; -- ar

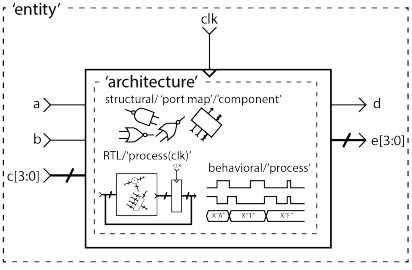
entity <e_name> is
    <byggeparameter & ports>
end entity <e_name>;

architecture <a_name> of <e_name>
    <signaler & moduler>
begin
    <structural/RTL/behavioral>
end architecture <a_name>;
```

# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
   - entity
   - architecture, structural
   - architecture, register transfer language
   - architecture, behavioural
   - logic and arithmetic operands
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# VHDL ’entity’ port definisjon




```
entity example is
    generic( N : integer := 4); -- parameter and
    -- ... its default value
    port(
        a, b, clk : in std_ulogic; -- bit in
        c : in std_ulogic_vector(N-1 downto 0); --
        d : out std_ulogic; -- bit out
        e : out std_ulogic_vector(N-1 downto 0) --
    );
end entity example;
```

# ’architecture’ stiler

’stiler’ kalles også ’levels of abstraction’ i literaturen. De er brukt litt ukonsistent. Vi fraviker boka her:

- **structural**
  for å definere forbindelser mellom lav nivå gates (full kontroll over innmaten. OBS: boka kaller disse for ’behavioral’.) eller hirarkisk mellom høy nivå entities. Bra for kombinatorisk logikk, men også sette sammen høy nivå systemer.
- **register transfer language (RTL)** for å definere sekvensiell logikk blokker med tilstandstransisjoner styrt av signaler (typisk så er det et CLK-signal). Bra for synkrone FSM-er.
- **behavioral** for å definere signalsekvenser for å simulere og teste kretsenes funksjonalitet. Bra for testbenker (ikke syntetiserbar!).

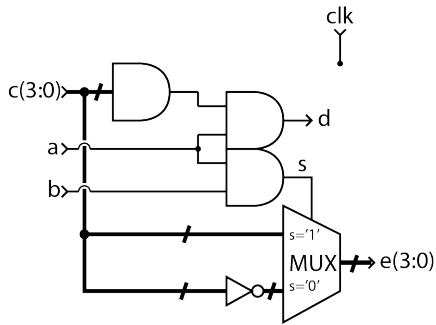
# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
   - entity
   - architecture, structural
   - architecture, register transfer language
   - architecture, behavioural
   - logic and arithmetic operands
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# VHDL ’architecture’ i ’structural’ stil

Eksempel 1, på port-level (for kombinatorisk logikk):

```
architecture structural of example is
    signal s: std_ulogic; -- extra node
begin
    d <= a and (and c);
    s <= a and b;
    e <= c when s else (not c);
end architecture structural;
```

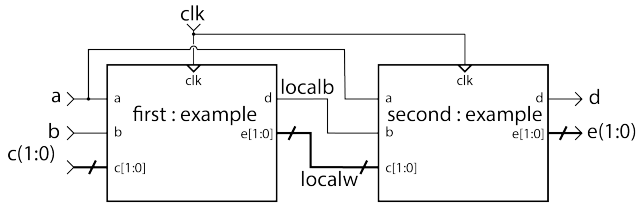



Ekvivalent:

```
...
    e <= c when s else (not c);
    s <= a and b;
...
```

# VHDL ’architecture’ i structural stil, hirarkisk (1/4)

Eksempel 2, hirarkisk (høy level kombinatorisk eller også sekvensielle blokker):




Vi lager en ny entity med (tilfeldigvis) samme grensesnitt, men den skal bygges med to ’example’ entities definert fra før. Det er ikke egentlig anbefalt å bruke de samme portnavnene som kan forvirre … følg godt med nå!

# VHDL ’architecture’ i structural stil, hirarkisk (2/4)

Samme grensesnitt som ’example’. Vi setter en annen ’default’ på parameter N og vil bruke den senere til å overskrive default-en i ’example’.

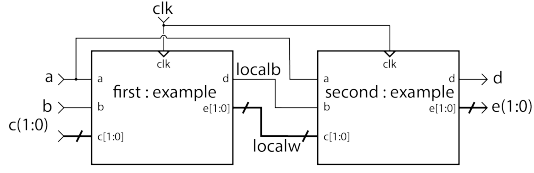
```
entity hierarchy_example is
    generic( N : integer :=2);
    port(
        a, b, clk : in std_ulogic; -- bit in
        c : in std_ulogic_vector(N-1 downto 0); -- bus in
        d : out std_ulogic; -- bit out
        e : out std_ulogic_vector(N-1 downto 0) -- bus out
    );
end entity hierarchy_example;
```

# VHDL 'architecture' i structural stil, hirarkisk (3/4)

```
architecture structural of hierarchy_example is
    component example is
        generic( N : integer);
        port(
            a, b, clk : in std_ulogic;
            c : in std_ulogic_vector(N-1 downto 0);
            d : out std_ulogic;
            e : out std_ulogic_vector(N-1 downto 0)
        );
    end component;
    signal localb : std_ulogic;
    signal localw : std_ulogic_vector(N-1 downto 0);
begin
    <structure declaration (i.e. connections) on next slide>
end architecture structural;
```

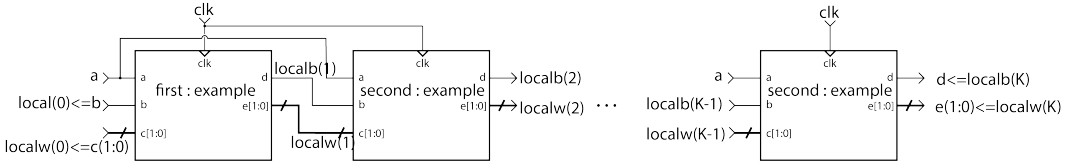
# VHDL 'architecture' i structural stil, hirarkisk (4/4)

```
architecture structural of hierarchy_example is
    <components and signal declarations on previous slide>
begin
    first : example
        generic map (N => N)
        -- <name in instance> => <name in architecture>
        port map (a => a, b => b, clk => clk, c => c, d => localb, e => localw);
    second : example
        generic map (N => N)
        port map (a => a, b => localb , clk => clk, c => localw, d => d, e => e);
end architecture structural;
```



# VHDL ’architecture’ i structural stil, instans array (1/4)

Eksempel 3, hirarkisk og en array av instanser:




Nok en ny entity med samme grensesnitt, men den skal bygges med K ’example’ entities definert fra før koblet i seri.

# VHDL ’architecture’ i structural stil, instans array (2/4)

```
library ieee;
use ieee.std_logic_1164.all; -- logic
-- use ieee.numeric_std.all; -- arithmetic

entity array_example is
    generic( N : integer := 2; -- bus width for ports c and e
        K : integer := 3); -- number of sequentially connected ’example’ component
    port(
        a, b, clk : in std_ulogic; -- bit in
        c : in std_ulogic_vector(N-1 downto 0); -- bus in
        d : out std_ulogic; -- bit out
        e : out std_ulogic_vector(N-1 downto 0) -- bus out
    );
end entity array_example;
```

# VHDL 'architecture' i structural stil, instans array (3/4)

```
architecture structural of array_example is
    component example is
        generic( N : integer);
        port(
            a, b, clk : in std_ulogic;
            c : in std_ulogic_vector(N-1 downto 0);
            d : out std_ulogic;
            e : out std_ulogic_vector(N-1 downto 0)
        );
    end component;
    signal localb : std_ulogic_vector(K downto 0);
    type intN_array is array (K downto 0) of std_logic_vector(N-1 downto 0);
    signal localw : intN_array;
begin
    <structure declaration>
end architecture structural;
```

# VHDL 'architecture' i structural stil, instans array (4/4)

```
architecture structural of array_example is
    <component and signal declarations>
begin
    localw(0) <= c;
    localb(0) <= b;
    d <= localb(K);
    e <= localw(K);
    element_array: for i in K-1 downto 0 generate
        element: example
            generic map (N => N)
            port map (a => a, b => localb(i), clk => clk, c => localw(i),
                      d => localb(i+1), e => localw(i+1));
    end generate element_array;
end architecture structural;
```

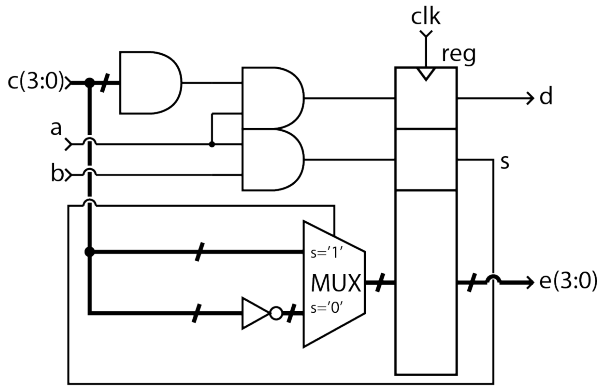
# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
   - entity
   - architecture, structural
   - architecture, register transfer language
   - architecture, behavioural
   - logic and arithmetic operands
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# VHDL 'architecture' RTL style

For å implementere FSMs

```
architecture RTL of example is
 signal s: std_ulogic;
begin
 process(clk)
 begin
  if rising_edge(clk) then
   d <= a and (and c);
   s <= a and b;
   e <= c when s else (not c);
  end if;
 end process;
end architecture RTL;
```

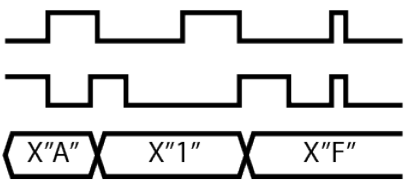


# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
   - entity
   - architecture, structural
   - architecture, register transfer language
   - architecture, behavioural
   - logic and arithmetic operands
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# VHDL 'architecture' behavioral style

For å programmere (!) en sekvens av inngangssignaler for å teste




```
architecture behavioral of example is
 signal s: std_ulogic;
begin
 process
 begin
  d <= a and (and c);
  wait for 10 ns;
  s <= a and b;
  wait for 10 ns;
  e <= c when s else (not c);
  wait for 10 ns;
  std.env.stop;
 end process;
end architecture behavioral;
```

# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
   - entity
   - architecture, structural
   - architecture, register transfer language
   - architecture, behavioural
   - logic and arithmetic operands
4. eksempel på høy level kombinatorisk logikk: ripple carry adder

# VHDL basic operators

**Table 4.2 VHDL operator precedence**

| | Op | Meaning |
|---|---|---|
| Highest | `not` | NOT |
| | `*`, `/`, `mod`, `rem` | MUL, DIV, MOD, REM |
| | `+`, `-` | PLUS, MINUS |
| | `rol`, `ror`, `srl`, `sll` | Rotate, Shift logical |
| | `<`, `<=`, `>`, `>=` | Relative Comparison |
| | `=`, `/=` | Equality Comparison |
| Lowest | `and`, `or`, `nand`, `nor`, `xor`, `xnor` | Logical Operations |

**Obs:**
`and` og `or` har samme prioritet, so man må *alltid* bruke klammer om man vil være sikker på rekkefølgen av beregningene. VHDL følger ikke `and` før `or` regelen!

# VHDL multi-bit numbers

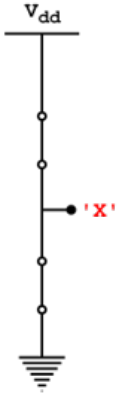
**Table 4.4 VHDL numbers**

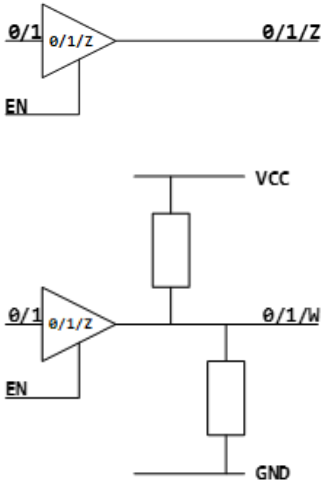
| Numbers | Bits | Base | Val | Stored |
|---|---|---|---|---|
| 3B"101" | 3 | 2 | 5 | 101 |
| B"11" | 2 | 2 | 3 | 11 |
| 8B"11" | 8 | 2 | 3 | 00000011 |
| 8B"1010_1011" | 8 | 2 | 171 | 10101011 |
| 3D"6" | 3 | 10 | 6 | 110 |
| 6O"42" | 6 | 8 | 34 | 100010 |
| 8X"AB" | 8 | 16 | 171 | 10101011 |
| "101" | 3 | 2 | 5 | 101 |
| B"101" | 3 | 2 | 5 | 101 |
| X"AB" | 8 | 16 | 171 | 10101011 |

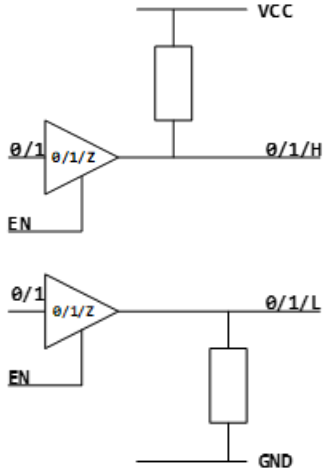
# std_(u)logic states

| Verdi | Betydning | Bruk |
|---|---|---|
| ‘U’ | Uninitialized | Startverdi *for alle signaler* i simulering |
| ‘X’ | Forcing Unknown | Kortslutning, MAO: feil som må fikses |
| ‘0’ | Forcing zero | Signal drevet lavt, jordet |
| ‘1’ | Forcing one | Signal drevet høyt |
| ‘Z’ | High impedance | Signal ikke drevet, tristate |
| ‘W’ | Weak unknown | Bussterminering (simulering) |
| ‘L’ | Weak zero | Pull-down motstand (simulering) |
| ‘H’ | Weak one | Pull-up motstand (simulering) |
| ‘-’ | Don’t care | Likegyldig om vi får ‘1’ eller ‘0’ |

# std_(u)logic states other than ’0’ and ’1’ explained



# std_(u)logic difference

**std_ulogic** throws an **error** when two outputs are connected together. Good!!! Use everywhere except for shared buses!

**std_logic** ‘resloves’ signals with two driving ports. Use (with care!) only for shared buses!!! **Errors might remain hidden**!

Resolution of shorted std_logic signals:

| shorted A/B | ‘U’ | ‘X’ | ‘0’ | ‘1’ | ‘Z’ | ‘W’ | ‘L’ | ‘H’ | ‘-’ |
|---|---|---|---|---|---|---|---|---|---|
| ‘U’ | ‘U’ | ‘U’ | ‘U’ | ‘U’ | ‘U’ | ‘U’ | ‘U’ | ‘U’ | ‘U’ |
| ‘X’ | ‘U’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ |
| ‘0’ | ‘U’ | ‘X’ | ‘0’ | ‘X’ | ‘0’ | ‘0’ | ‘0’ | ‘0’ | ‘X’ |
| ‘1’ | ‘U’ | ‘X’ | ‘X’ | ‘1’ | ‘1’ | ‘1’ | ‘1’ | ‘1’ | ‘X’ |
| ‘Z’ | ‘U’ | ‘X’ | ‘0’ | ‘1’ | ‘Z’ | ‘W’ | ‘L’ | ‘H’ | ‘X’ |
| ‘W’ | ‘U’ | ‘X’ | ‘0’ | ‘1’ | ‘W’ | ‘W’ | ‘W’ | ‘W’ | ‘X’ |
| ‘L’ | ‘U’ | ‘X’ | ‘0’ | ‘1’ | ‘L’ | ‘W’ | ‘L’ | ‘W’ | ‘X’ |
| ‘H’ | ‘U’ | ‘X’ | ‘0’ | ‘1’ | ‘H’ | ‘W’ | ‘W’ | ‘H’ | ‘X’ |
| ‘-’ | ‘U’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ | ‘X’ |

# std_ulogic foretrukket for lettere feildeteksjon

```
entity u_or_no_u is
    port( a,b: in std_logic;
        c: out std_logic;
        d,e: in std_ulogic;
        f: out std_ulogic
    );
end entity u_or_no_u;
```

```
architecture structural of u_or_no_u is
begin
    c <= a or b;
    c <= a and b;
    f <= d or e;
    f <= d and e;
end architecture structural;
```

```
bash-5.1$ ghdl -a --std=08 tb_u_vs_no_u_logic.vhd
bash-5.1$ ghdl -r --std=08 tb_u_or_no_u --wave=wave.ghw
u_vs_no_u_logic.vhd:10:9:error: too many drivers for port "f"
        f: out std_ulogic
        ^
bash-5.1$
```

# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
4. eksempel på høy level kombinatorisk logikk: ripple carry adder
   - 32-bit ripple carry adder
   - full adder
   - testbenk 32-bit adder

# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
4. eksempel på høy level kombinatorisk logikk: ripple carry adder
   - 32-bit ripple carry adder
   - full adder
   - testbenk 32-bit adder

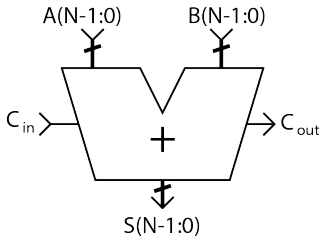
# 'entity' for 32 bit ripple carry adder

```
library ieee;
use ieee.std_logic_1164.all;
entity ripple_carry_adder_Nb is
  <bygge-parametere?>
  <porter?>
end entity ripple_carry_adder_Nb;
```

<symbol?>

# 'entity' for 32 bit ripple carry adder

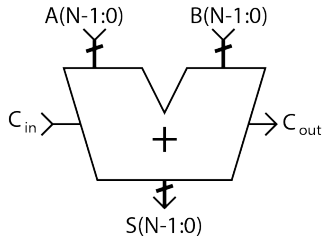
```
library ieee;
use ieee.std_logic_1164.all;
entity ripple_carry_adder_Nb is
  <bygge-parametere?>
  <porter?>
end entity ripple_carry_adder_Nb;
```




n-bit adder symbol.

# 'entity' for 32 bit ripple carry adder

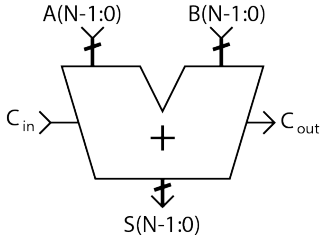
```
library ieee;
use ieee.std_logic_1164.all;
entity ripple_carry_adder_Nb is
  generic(
   N : integer := 32
  );
  <porter?>
end entity ripple_carry_adder_Nb;
```




n-bit adder symbol.

# 'entity' for 32 bit ripple carry adder

```
library ieee;
use ieee.std_logic_1164.all;
entity ripple_carry_adder_Nb is
  generic(
   N : integer := 32
  );
  port(
   a,b : in std_ulogic_vector(N-1 downto 0);
   cin : in std_ulogic;
   s : out std_ulogic_vector(N-1 downto 0);
   cout : out std_ulogic
  );
end entity ripple_carry_adder_Nb;
```




n-bit adder symbol.

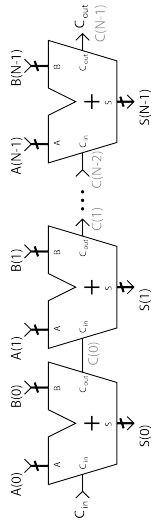
# strukturell 'architecture' for 32 bit ripple carry adder

```
architecture structural of ripple_carry_adder_Nb is
  <komponenter?>
  <signaler?>
begin
  <struktur>
end architecture structural;
```

<skjematikk?>

# strukturell 'architecture' for 32 bit ripple carry adder

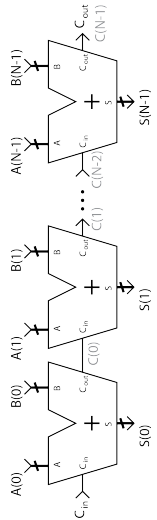
```
architecture structural of ripple_carry_adder_Nb is
  <komponenter?>
  <signaler?>
begin
  <struktur>
end architecture structural;
```




n-bit adder skjematikk.

# strukturell 'architecture' for 32 bit ripple carry adder

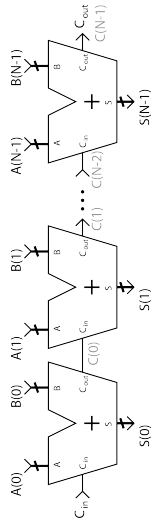
```
architecture structural of ripple_carry_adder_Nb is
  component full_adder_1b is
   port(
    a,b,cin: in std_ulogic;
    s,cout: out std_ulogic
   );
  end component full_adder_1b;
  signal c : std_ulogic_vector(N-1 downto 0);
begin
  <struktur>
end architecture structural;
```




n-bit adder skjematikk.

# strukturell 'architecture' for 32 bit ripple carry adder

```
architecture structural of ripple_carry_adder_Nb is
  <komponenter?>
  <signaler?>
begin
  c(0)<=((a(0) xor b(0)) and cin) or (a(0) and b(0));
  s(0) <= (a(0) xor b(0)) xor cin;
  full_adder_array: for i in 1 to N-1 generate
   full_adder_element: full_adder_1b
   port map (a => a(i),
    b => b(i),
    cin => c(i-1),
    s => s(i),
    cout => c(i)
   );
  end generate;
  cout <= c(N-1);
end architecture structural;
```



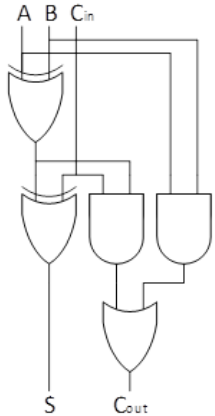

n-bit adder skjematikk.

# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
4. eksempel på høy level kombinatorisk logikk: ripple carry adder
   - 32-bit ripple carry adder
   - full adder
   - testbenk 32-bit adder

# fulladder

Implementasjon skal dere lage som del av oblig 2!




Skjematikk av en full-adder.

# Innhold

1. Kapittel 4 Delmål
2. HDL intro
3. VHDL Syntaks
4. eksempel på høy level kombinatorisk logikk: ripple carry adder
   - 32-bit ripple carry adder
   - full adder
   - testbenk 32-bit adder

# testbenk adder (1/5)

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

entity ripple_carry_adder_tb is
  generic(
    N : integer := 32
  );
  -- no ports, i.e. I/O for testbenches
end entity ripple_carry_adder_tb;

architecture behavioural of ripple_carry_adder_tb is
  <architecture>
end architecture behavioural;
```

# testbenk adder (2/5)

```
architecture behavioural of ripple_carry_adder_tb is
  component ripple_carry_adder_Nb is
    generic( N : integer := 32 );
    port(
      a,b : in std_ulogic_vector(N-1 downto 0);
      cin : in std_ulogic;
      s : out std_ulogic_vector(N-1 downto 0);
      cout : out std_ulogic
    );
  end component ripple_carry_adder_Nb;
  <signals>
begin
  <behavioural description>
end architecture behavioural;
```

# testbenk adder (3/5)

```
architecture behavioural of ripple_carry_adder_tb is
  <components>
  signal cin, cout: std_ulogic;
  signal a,b,s: std_ulogic_vector(N-1 downto 0) := (others => '0');
  signal success : boolean;
begin
  <behavioural description>
end architecture behavioural;
```

# testbenk adder (4/5)

```
architecture behavioural of ripple_carry_adder_tb is
  <components & signals>
begin
  DUT: ripple_carry_adder_Nb
    generic map (N => N)
    port map( a => a, b => b, s => s, cin => cin, cout => cout
    );
  process begin
    <testsignal time evolution & assertion/reporting>
  end process;
end architecture behavioural;
```

# testbenk adder (5/5)

```
architecture behavioural of ripple_carry_adder_tb is
  <components & signals>
begin
  <component instantiation, particularly the DUT>
  process begin
    a <= std_logic_vector(to_unsigned(2635,N));
    b <= std_logic_vector(to_unsigned(70000,N));
    cin <= '1';
    wait 10 ns;
    succsess <= (to_integer(unsigned(s)) = 72636) and (cout='0');
    wait 10ns
    assert(success)
      report ("Oh no :-( s=" & integer'image(to_integer(unsigned(s))))
      severity error;
  end process;
end architecture behavioural;
```

# Neste forelesning: digitale byggeblokker

Kjapt: Von Neumann arkitektur (se IN1020!)

Og så noen digiatale byggeblokker

- multipleksere
- skiftere, multiplisere/dele med $2^N$
- addere, subtraktere
- komparatorer
- En enkel ALU
- og nevne noen flere ...