Oppgave 1

En 4-input Xilinx LUT med innhold **”9009”** (hex) realiserer en:	A	xor-xor-or	
	B	xor-xor-nand	
	C	xor-xor-nor	**X**
	D	xor-xor-and	
	E	xor-xor-xor	

Oppgave 2

FPGA-teknologi	A	Forbindelseslinjer mellom LUT’er har vanligvis større tidsforsinkelse enn tidsforsinkelsen gjennom LUT’er i SRAM-teknologi.	**X**
	B	En FPGA krets basert på Flash er umiddelbart aktiv etter strømtilkobling.	**X**
	C	SRAM-teknologi er vel så motstandsdyktig mot stråling som antifuse teknologi.	
	D	Konfigurasjonsfiler er alltid så små at det er raskt å bytte til en ny konfigurasjon.	
	E	JTAG porten kan brukes både til konfigurasjon og til debugging.	**X**

Oppgave 3

Design 1	A	Block RAM som ikke brukes kan fjernes fra FPGA’en.	
	B	En Xilinx Block RAM har to uavhengige porter som begge kan leses fra og skrives til samtidig.	**X**
	C	Tilbakekoblingssløyfer med flip-flop’er kan brukes i en FPGA.	**X**
	D	Asynkront design anbefales ikke i en FPGA.	**X**
	E	En BUFG modul kan bare brukes til klokkesignaler.	

Oppgave 4

Design 2	A	En hard IP kjerne tar vanligvis mere plass enn en tilsvarende myk IP kjerne.	
	B	Med en Digital Clock Manager (DCM) modul kan man øke klokkesignalet til det firedobbelte og det genererte klokkesignalet vil være i fase med inngangsklokken.	**X**
	C	I en Xilinx FPGA har set inngangen til en flip-flop lavere prioritet enn reset inngangen.	**X**
	D	Initialverdien etter deklarasjon av et signal av typen std_logic vil være ’X’.	
	E	To signaler av typen std_logic med verdiene ’Z’ og ’0’ som driver samme signal får verdien ’X’.	

Oppgave 5 for INF3430

Høyhastighets serielinker	A	Differensielle signaler brukes for å redusere støy problemer.	**X**
	B	Høyhastighetslinker har i tillegg til de differensielle datalinjene også differensielle klokkelinjer.	
	C	De differensielle linjene fra en sender kan gå til opptil 4 mottagere.	
	D	8B/10B signal koding brukes for å unngå flere enn 8 påfølgende like bit.	
	E	For PCIe gen. 1 er faktisk datarate 2.0 Gbit/s med 8B/10B koding som gjør at linjens baudrate blir 2.5 Gbit/s.	**X**

Oppgave 6 (Oppgave 5 for INF4431)

Det er ikke krav om testbenk i besvarelsen. Testbenken er kun tatt med for enklere kunne simulere besvarelsen.

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

entity counter is
  port (
    rst     : in  std_logic;
    mclk    : in  std_logic;
    ena     : in  std_logic;
    load    : in  std_logic;
    value   : in  std_logic_vector(7 downto 0);
    up      : in  std_logic;
    zero    : out std_logic;
    max     : out std_logic;
    cnt     : out std_logic_vector(7 downto 0));
end counter;

architecture rtl_answ of counter is
begin

  process (rst, mclk) is
    variable cnt_i : unsigned(7 downto 0);
  begin
    if (rst = '1') then
      zero    <= '1';
      max     <= '0';
      cnt     <= (others => '0');
      cnt_i   := (others => '0');

    elsif rising_edge(mclk) then

      if load='1' then
        cnt_i := unsigned(value);
      else
        if ena='1' and up='1' and cnt_i/=x"FF" then
          cnt_i := cnt_i + 1;
        elsif ena='1' and up='0' and cnt_i/=x"00" then
          cnt_i := cnt_i - 1;
        end if;
      end if;

      zero <= '0';
      if cnt_i=x"00" then
        zero <= '1';
      end if;

      max <= '0';
      if cnt_i=x"FF" then
        max <= '1';
      end if;

      cnt <= std_logic_vector(cnt_i);

    end if;
  end process;

end rtl_answ;


library ieee;
use ieee.std_logic_1164.all;

entity tb_counter is
  -- empty;
end tb_counter;

architecture beh of tb_counter is

  component counter is
    port (rst   : in  std_logic;
          mclk  : in  std_logic;
          ena   : in  std_logic;
          load  : in  std_logic;
          value : in  std_logic_vector(7 downto 0);
          up    : in  std_logic;
          zero  : out std_logic;
          max   : out std_logic;
          cnt   : out std_logic_vector(7 downto 0));
  end component counter;

  signal rst   : std_logic;
  signal mclk  : std_logic:= '0';
  signal ena   : std_logic;
  signal load  : std_logic;
  signal value : std_logic_vector(7 downto 0);
  signal up    : std_logic;
  signal zero  : std_logic;
  signal max   : std_logic;
  signal cnt   : std_logic_vector(7 downto 0);

begin

  counter_0: counter
    port map (rst   => rst,
              mclk  => mclk,
              ena   => ena,
              load  => load,
              value => value,
              up    => up,
              zero  => zero,
              max   => max,
            cnt   => cnt);

  P_clock: process is
  begin
      mclk <= '0';
      wait for 50 ns;
      mclk <= '1';
      wait for 50 ns;
  end process P_clock;

  rst  <= '1', '0' after 100 ns;
  ena  <= '0',
          '1' after 400 ns,
          '0' after 3200 ns;
  up   <= '1',
          '0' after 1600 ns;
  load <= '0',
          '1' after 800 ns,
          '0' after 900 ns,
          '1' after 2400 ns,
          '0' after 2500 ns;
  value <= x"FA",
           x"05" after 2300 ns;

end beh;
```

Oppgave 6 for INF4431

```
module counter(output logic zero, max, [7:0] cnt,
               input  logic rst, mclk, ena, load,
               input  logic [7:0] value,
               input  logic up);

   logic [7:0] cnt_i;

   always_ff @(posedge mclk, negedge rst)
     begin
       if (rst)
         begin
            zero  <= '1;
            max   <= '0;
            cnt_i = '0;
         end else begin
            if (load)
              cnt_i = value;
            else begin
              if (ena && up && cnt_i!=8'hFF)
                cnt_i = cnt_i + 1;
              else if (ena && !up && cnt_i!=8'h00)
                cnt_i = cnt_i - 1;

              if (cnt_i==8'h00)
                zero <= '1;
              else
                zero <= '0;

              if (cnt_i==8'hFF)
                max <= '1;
              else
                max <= '0;
            end;
         end;
         cnt <= cnt_i;
     end
endmodule
```

Oppgave 7

Det er ikke krav om testbenk i besvarelsen. Testbenken er kun tatt med for enklere kunne simulere besvarelsen.

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

entity compute is
  port (
    rst     : in  std_logic;
    mclk    : in  std_logic;
    a       : in  unsigned(7 downto 0);
    b       : in  unsigned(7 downto 0);
    c       : in  unsigned(6 downto 0);
    d       : in  unsigned(6 downto 0);
    e       : in  unsigned(6 downto 0);
    f       : in  unsigned(6 downto 0);
    x       : out unsigned(15 downto 0);
    y       : out unsigned(15 downto 0));
end compute;

architecture rtl_answ of compute is
    signal preg1 : unsigned(15 downto 0);
    signal preg2 : unsigned(15 downto 0);
begin
  process (rst, mclk) is
    variable tmp1 : unsigned(15 downto 0);
    variable tmp2 : unsigned(15 downto 0);
  begin
    if (rst = '1') then
      tmp1  := (others => '0');
      tmp2  := (others => '0');
      preg1 <= (others => '0');
      preg2 <= (others => '0');
      x     <= (others => '0');
      y     <= (others => '0');
    elsif rising_edge(mclk) then

      tmp1  := a*b;
      preg1 <= tmp1 + ('0' & c);
      preg2 <= tmp1 - ('0' & c);

      tmp2:= (("000000000" & d) + ("000000000" & e) +
              ("000000000" & f));
      x <= preg1 + tmp2;
      y <= preg2 - tmp2;

    end if;
  end process;
end rtl_answ;


library ieee;
```

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

entity tb_compute is
  -- empty;
end tb_compute;

architecture beh of tb_compute is

  component compute is
    port (rst  : in  std_logic;
          mclk : in  std_logic;
          a    : in  unsigned(7 downto 0);
          b    : in  unsigned(7 downto 0);
          c    : in  unsigned(6 downto 0);
          d    : in  unsigned(6 downto 0);
          e    : in  unsigned(6 downto 0);
          f    : in  unsigned(6 downto 0);
          x    : out unsigned(15 downto 0);
          y    : out unsigned(15 downto 0));
  end component compute;

  signal rst  : std_logic;
  signal mclk : std_logic := '0';
  signal a    : unsigned(7 downto 0);
  signal b    : unsigned(7 downto 0);
  signal c    : unsigned(6 downto 0);
  signal d    : unsigned(6 downto 0);
  signal e    : unsigned(6 downto 0);
  signal f    : unsigned(6 downto 0);
  signal x    : unsigned(15 downto 0);
  signal y    : unsigned(15 downto 0);

begin

  compute_0: compute
    port map (rst  => rst,
              mclk => mclk,
              a    => a,
              b    => b,
              c    => c,
              d    => d,
              e    => e,
              f    => f,
              x    => x,
              y    => y);

  P_clock: process is
  begin
      mclk <= '0';
      wait for 50 ns;
      mclk <= '1';
      wait for 50 ns;
  end process P_clock;

  rst  <= '1', '0' after 100 ns;

  a <= x"01",
       x"10" after 400 ns,
       x"FE" after 600 ns,
       x"FF" after 700 ns;
  b <= x"02",
       x"11" after 400 ns,
       x"FF" after 600 ns;
  c <= "0000011",
       "0010000" after 400 ns,
       "1111111" after 600 ns;
  d <= "0000100",
       "0010001" after 400 ns,
       "1111111" after 600 ns;
  e <= "0000101",
       "0010010" after 400 ns,
       "1111111" after 600 ns;
  f <= "0000110",
       "0010011" after 400 ns,
       "1111111" after 600 ns;

end beh;
```

Oppgave 8

Det er ikke krav om testbenk i besvarelsen. Testbenken er kun tatt med for enklere kunne simulere besvarelsen.

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

entity databus is
  port (
    in0  : in  std_logic_vector(7 downto 0);
    in1  : in  std_logic_vector(7 downto 0);
    in2  : in  std_logic_vector(7 downto 0);
    in3  : in  std_logic_vector(7 downto 0);
    ena  : in  std_logic;
    sel  : in  std_logic_vector(1 downto 0);
    data : out std_logic_vector(7 downto 0));
end databus;

architecture rtl_answ of databus is
begin
  data <= in0 when (ena='1' and sel="00") else (others => 'Z');
  data <= in1 when (ena='1' and sel="01") else (others => 'Z');
  data <= in2 when (ena='1' and sel="10") else (others => 'Z');
  data <= in3 when (ena='1' and sel="11") else (others => 'Z');
end rtl_answ;


library ieee;
use ieee.std_logic_1164.all;

entity tb_databus is
  -- empty;
end tb_databus;

architecture beh of tb_databus is

  component databus is
    port (in0  : in  std_logic_vector(7 downto 0);
          in1  : in  std_logic_vector(7 downto 0);
          in2  : in  std_logic_vector(7 downto 0);
          in3  : in  std_logic_vector(7 downto 0);
          ena  : in  std_logic;
          sel  : in  std_logic_vector(1 downto 0);
```

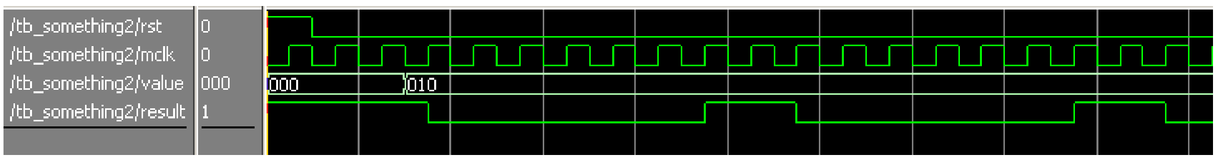
```
            data : out std_logic_vector(7 downto 0));
  end component databus;

  signal in0  : std_logic_vector(7 downto 0);
  signal in1  : std_logic_vector(7 downto 0);
  signal in2  : std_logic_vector(7 downto 0);
  signal in3  : std_logic_vector(7 downto 0);
  signal ena  : std_logic;
  signal sel  : std_logic_vector(1 downto 0);
  signal data : std_logic_vector(7 downto 0);

begin

  databus_0: databus
    port map (in0  => in0,
              in1  => in1,
              in2  => in2,
              in3  => in3,
              ena  => ena,
              sel  => sel,
              data => data);

  in0 <= x"01";
  in1 <= x"33";
  in2 <= x"05";
  in3 <= x"77";

  ena <= '0',
         '1' after 200 ns,
         '0' after 300 ns,
         '1' after 400 ns,
         '0' after 500 ns,
         '1' after 600 ns,
         '0' after 700 ns,
         '1' after 800 ns,
         '0' after 900 ns;

  sel <= "00",
         "01" after 400 ns,
         "10" after 600 ns,
         "11" after 800 ns;

end beh;
```

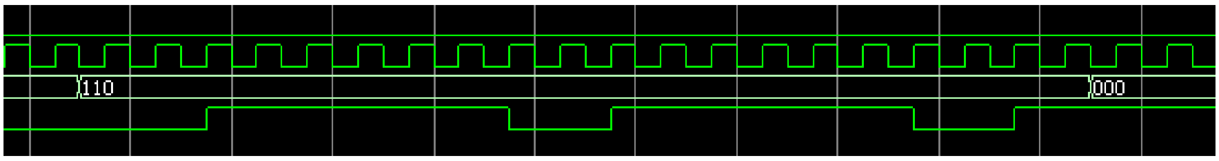
Oppgave 9

VHDL koden something utfører en såkalt puls bredde modulasjon (Puls Width Modulation, PWM).

Når value er lik null er result alltid lik ‘1’, men ellers bestemmer signalet value antall påfølgende klokkeperioder result skal være lik ‘1’. Dette gjentar seg med en periodelengde på 8 klokkeperioder.

Under vises simulering av modulen med value lik “010” (2 desimalt) og “110” (6 desimalt).

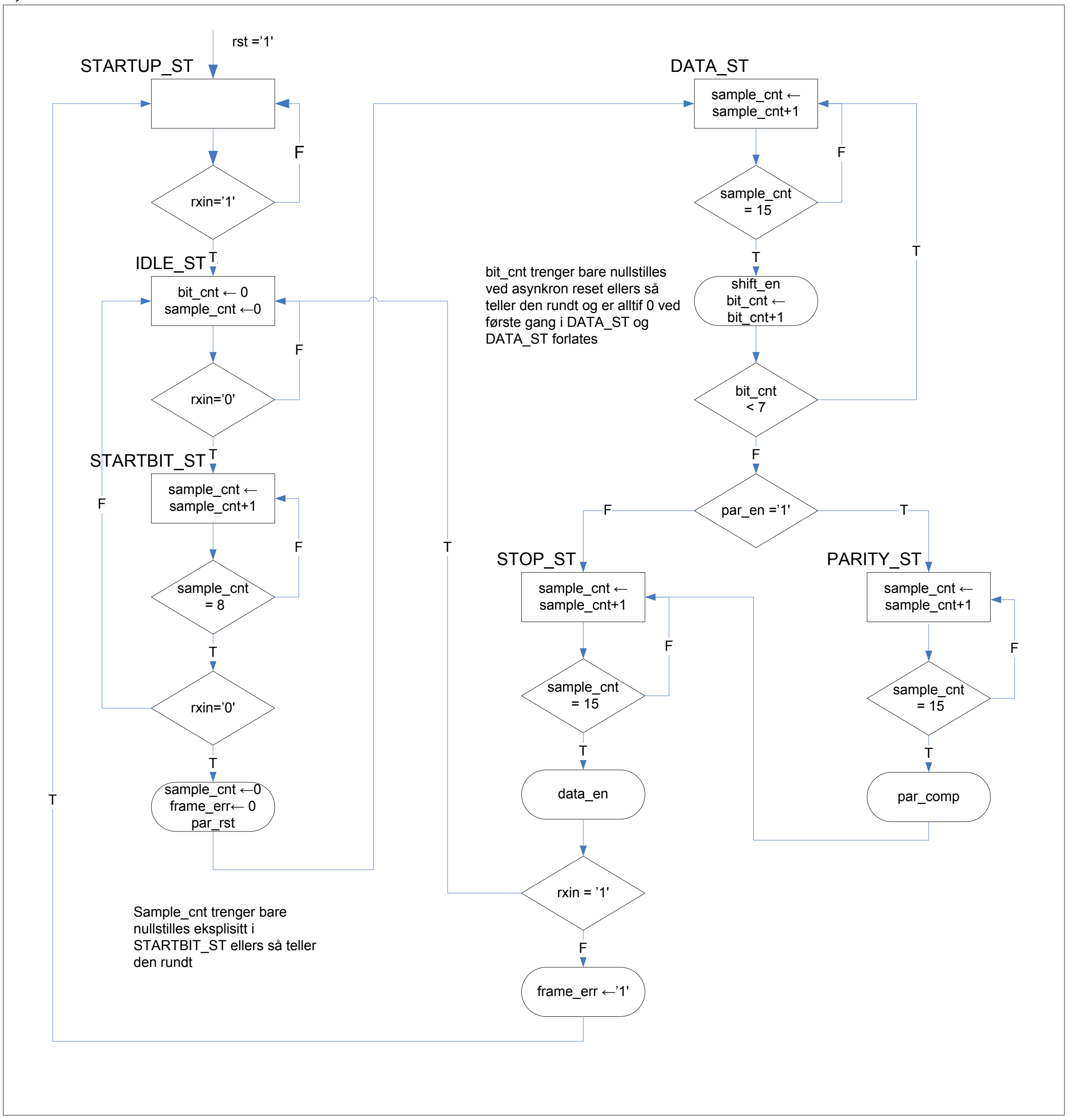




110
000

Oppgave 10

b)



rst ='1'
STARTUP_ST
rxin='1'
F
T
IDLE_ST
bit_cnt ← 0
sample_cnt ←0
rxin='0'
STARTBIT_ST
sample_cnt ←
sample_cnt+1
sample_cnt
= 8
rxin='0'
sample_cnt ←0
frame_err← 0
par_rst
Sample_cnt trenger bare nullstilles eksplisitt i STARTBIT_ST ellers så teller den rundt
DATA_ST
sample_cnt ←
sample_cnt+1
sample_cnt
= 15
shift_en
bit_cnt ←
bit_cnt+1
bit_cnt
< 7
bit_cnt trenger bare nullstilles ved asynkron reset ellers så teller den rundt og er alltif 0 ved første gang i DATA_ST og DATA_ST forlates
par_en ='1'
STOP_ST
sample_cnt ←
sample_cnt+1
sample_cnt
= 15
data_en
rxin = '1'
frame_err ←'1'
PARITY_ST
sample_cnt ←
sample_cnt+1
sample_cnt
= 15
par_comp

a), c, og d)

```
--Oppgave 10 a).
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity PARITY_CHECK is
  port
  (
    rst       : in std_logic; --asynkron reset
    bclk      : in std_logic; --klokke (16x bitfrekvensen)
    shift_en  : in std_logic; --enabler skifte inn serielle data
    rxin      : in std_logic; --serielle data in
    par_rst   : in std_logic; --resetter registre i paritetsjekkeren
    par_comp  : in std_logic; --sammenligner mottatt og beregnet paritet
    par_err   : out std_logic --paritets error
  );
end ;

architecture RTL_PARITY_CHECK of PARITY_CHECK is

begin
  process(rst,bclk)
    variable parity : std_logic;
  begin
    if rst = '1' then
      par_err <= '0';
      parity := '1'; --'1'= even, '0'=odd parity
                     --feil i Zwolinski side 74
    elsif rising_edge(bclk) then
      if par_rst = '1' then
        parity := '1';
        par_err <= '0';
      elsif shift_en = '1' then
        if rxin = '1' then
          parity := not parity;
          par_err <= '0';
        end if;
      elsif par_comp = '1' then
        if rxin /= parity then
          par_err <= '1';
        end if;
      end if;
    end if;
  end process;
end architecture RTL_PARITY_CHECK;

--Oppgave 10 c)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity UART_RX_CTRL is
  port
  (
    rst       : in std_logic;   --asynkron reset
    bclk      : in std_logic;   --klokke (16x bittiden)
    par_en    : in std_logic;   --Enabler paritetsbruk
    par_err   : in std_logic;   --aktivt høyt dersom paritetsfeil
    rxin      : in std_logic;   --serielle data input
    shift_en  : out std_logic;  --enabler skifte inn serielle data
    data_en   : out std_logic;  --overfører data til mottaksbuffer(FIFO)
    par_rst   : out std_logic;  --resetter registre i paritetssjekker
    par_comp  : out std_logic;  --sammenligner mottatt og beregnet paritet
    frame_err : out std_logic   --aktivt dersom stoppbit ikke detekteres
  );
end ;

architecture RTL_UART_RX_CTRL of UART_RX_CTRL is

  signal sample_cnt    : unsigned(3 downto 0);
  signal sample_cnt_rst: std_logic;
  signal sample_cnt_en : std_logic;

  signal bit_cnt      : unsigned(2 downto 0);
  signal bit_cnt_rst  : std_logic;
  signal bit_cnt_en   : std_logic;
  signal frame_err_i    : std_logic;
  signal frame_err_i_en : std_logic;
  signal frame_err_i_rst: std_logic;


  type UART_RX_STATES is (STARTUP_ST, IDLE_ST, START_ST, DATA_ST, PARITY_ST, STOP_ST);
  signal curr_st, next_st : UART_RX_STATES;

begin

UART_RX_STATE_COMB:
  process(curr_st,par_en,par_err,sample_cnt, bit_cnt,rxin)
  begin
    sample_cnt_en   <= '0';
    sample_cnt_rst  <= '0';
    bit_cnt_en      <= '0';
    bit_cnt_rst     <= '0';
    par_rst         <= '0';
    shift_en        <= '0';
    data_en         <= '0';
    par_comp        <= '0';
    frame_err_i     <= '0';
    frame_err_i_en  <= '0';
    next_st         <= IDLE_ST;

    case curr_st is

      when STARTUP_ST =>
        if rxin = '0' then
          next_st <= STARTUP_ST;
        else
          next_st <= IDLE_ST;
        end if;

      when IDLE_ST =>
        sample_cnt_rst <= '1';
        --bit_cnt_rst    <= '1';
        if rxin = '0' then
          next_st <= START_ST;
        else
          next_st <= IDLE_ST;
        end if;

      when START_ST =>
        sample_cnt_en <= '1';
        --Mere robust variant
--        if sample_cnt < 8 then
--          if rxin = '1' then
--            next_st <= IDLE_ST;
--          else
--            next_st <= START_ST;
--          end if;
--        elsif sample_cnt = 8 then
        if sample_cnt = 8 then
          if rxin = '0' then
            bit_cnt_rst <= '1';
            sample_cnt_rst <= '1'; --Mealy output
            frame_err_i_en <= '1';
            par_rst <= '1';
            next_st <= DATA_ST;
          else
            next_st <= IDLE_ST;
          end if;
        else
          next_st <= START_ST;
        end if;

      when DATA_ST =>
        sample_cnt_en <= '1';
        if  sample_cnt = 15 then
          shift_en <= '1';
          bit_cnt_en <= '1';
          if bit_cnt < 7 then
            next_st <= DATA_ST;
          else
            if par_en = '1' then
              next_st <= PARITY_ST;
            else
              next_st <= STOP_ST;
```

```vhdl
            end if;
          end if;
        else
          next_st <= DATA_ST;
        end if;

      when PARITY_ST =>
        sample_cnt_en <= '1';
        if sample_cnt = 15 then
          par_comp <= '1';
          next_st <= STOP_ST;
        else
          next_st <= PARITY_ST;
        end if;

      when STOP_ST =>
        sample_cnt_en <= '1';
        if sample_cnt = 15 then
          data_en <= '1';
          if rxin = '1'then
            next_st <= IDLE_ST;
          else
            frame_err_i_en <= '1';
            frame_err_i    <= '1';
            next_st <= STARTUP_ST;
          end if;
        else
          next_st <= STOP_ST;
        end if;

    end case;
  end process;

UART_RX_STATE_REG:
  process(rst,bclk)
  begin
    if rst = '1' then
      curr_st <= IDLE_ST;
    elsif rising_edge(bclk) then
      curr_st <= next_st;
    end if;
  end process;

FRAME_ERROR:
  process(rst,bclk)
  begin
    if rst = '1' then
      frame_err <= '0';
    elsif rising_edge(bclk) then
      if frame_err_i_rst = '1' then
        frame_err <= '0';
      elsif frame_err_i_en = '1' then
        frame_err <= frame_err_i;
      end if;
    end if;
  end process;

--Oppgitt i oppgave teksten
BIT_COUNTER:
  process(rst,bclk)
  begin
    if rst = '1' then
      bit_cnt <= (others => '0');
    elsif rising_edge(bclk) then
      if bit_cnt_rst = '1' then
        bit_cnt <= (others => '0');
      elsif bit_cnt_en = '1' then
        bit_cnt <= bit_cnt + 1;
      end if;
    end if;
  end process;

--Oppgitt i oppgave teksten
SAMPLE_COUNTER:
  process(rst,bclk)
  begin
    if rst = '1' then
      sample_cnt <= (others => '0');
    elsif rising_edge(bclk) then
```

```vhdl
      if sample_cnt_rst = '1' then
        sample_cnt <= (others => '0');
      elsif sample_cnt_en = '1' then
        sample_cnt <= sample_cnt + 1;
      end if;
    end if;
  end process;
end architecture RTL_UART_RX_CTRL;

--Oppgave 10d)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity TEST_UART_RX_CTRL is
end;

architecture TB_UART_RX_CTRL of TEST_UART_RX_CTRL is

component PARITY_CHECK
  port
  (
    rst       : in std_logic;
    bclk      : in std_logic;
    shift_en  : in std_logic;
    rxin      : in std_logic;
    par_rst   : in std_logic;
    par_comp  : in std_logic;
    par_err   : out std_logic
  );
end component;

component UART_RX_CTRL is
  port
  (
    rst       : in std_logic;
    bclk      : in std_logic;
    par_en    : in std_logic;
    par_err   : in std_logic;
    rxin      : in std_logic;
    shift_en  : out std_logic;
    data_en   : out std_logic;
    par_rst   : out std_logic;
    par_comp  : out std_logic;
    frame_err : out std_logic
  );
end component;

signal rst       : std_logic := '0';
signal bclk      : std_logic := '0';
signal par_en    : std_logic := '0';
signal par_err   : std_logic;
signal rxin      : std_logic := '1';
signal shift_en  : std_logic;
signal data_en   : std_logic;
signal par_rst   : std_logic;
signal par_comp  : std_logic;
signal frame_err : std_logic;
signal start     : std_logic := '0';
signal dout      : unsigned(7 downto 0);
signal dbus      : unsigned(7 downto 0);
constant Tbit : time := 160 ns;
constant Tb   : time := 10 ns;

begin


UUT: entity work.UART_RX_CTRL(RTL_UART_RX_CTRL)
port map
(
  rst          =>  rst,
  bclk         =>  bclk,
  par_en       =>  par_en,
  par_err      =>  par_err,
  rxin         =>  rxin,
  shift_en     =>  shift_en,
  data_en      =>  data_en,
  par_rst      =>  par_rst,
  par_comp     =>  par_comp,
  frame_err   =>  frame_err
);

--Bruker implementasjonen fra oppgave a) for å lage
--stimuli til UUT
--Ikke krav til selvtestende testbenk
PARITY: PARITY_CHECK
port map
(
  rst         => rst,
  bclk        => bclk,
  shift_en    => shift_en,
  rxin        => rxin,
  par_rst     => par_rst,
  par_comp    => par_comp,
  par_err     => par_err
);

bclk <= not bclk after Tb/2;

STIMULI:
process
  variable dcomp   : unsigned(7 downto 0); --For å kunne se signal i Waveformviewer

  procedure SEND_BYTES(signal   par_en           : in std_logic;
                       signal   par_err          : in std_logic;
                       signal   do               : in unsigned(7 downto 0);
                       variable dcomp            : inout unsigned(7 downto 0);
                       constant startbit_err_sim : in integer;
                       constant frame_err_sim    : in integer;
                       constant par_err_sim      : in integer;
                       constant test_descr       : in string;
                       signal   bclk             : in std_logic;
                       signal   rxd              : inout std_logic)is

    variable parity : std_logic := '1';

    type datainput is array (0 to 255) of unsigned(7 downto 0);
    variable din  : datainput;
    variable err  : integer := 0;

  begin
    err := 0;
    wait for Tbit*5;
    for j in 0 to 255  loop
      parity := '1';
      wait until rising_edge(bclk);
      rxd  <= '0';
      if startbit_err_sim = 1 then
        for i in 0 to 3  loop
          rxd <= not rxd;
          wait until rising_edge(bclk);
        end loop;
        rxd <= '1';
        wait for Tbit;
        rxd <= '0';
      end if;
      din(j) := to_unsigned(j,8);
      dcomp  := din(j);
      wait for Tbit;
      for i in 0 to 7  loop
        rxd <= std_logic(din(j)(i));
        if din(j)(i) = '1' then
          parity := not parity;
        end if;
        wait for Tbit;
      end loop;
      if par_en = '1' then
        if par_err_sim = 1 then
          rxd <= not parity;
        else
          rxd <= parity;
        end if;
        wait for Tbit;
      end if;
      if frame_err_sim = 1 then
        rxd <= '0';
      else
        rxd <= '1';
      end if;
      wait for Tbit*2;
      rxd <= '1';
      wait for Tbit*3;
      if par_err_sim = 1 then
        if par_err = '0' then
          err := err+1;
        end if;
      else
        if din(j) /= dbus then
          err:=err+1;
        end if;
      end if;
    end loop;
    if err = 0 then
      report "Passed: " & test_descr;
    else
      report "NOT Passed: " & test_descr;
    end if;
  end procedure SEND_BYTES;

begin
  rst <= '1', '0' after Tbit;
  rxin <= '1';
  par_en <= '0';
  SEND_BYTES(par_en,par_err,dbus,dcomp,0,0,0,"No parity error or frame errors",bclk,rxin);
  par_en <= '1';
  SEND_BYTES(par_en,par_err,dbus,dcomp,0,0,0,"With parity check enabled",bclk,rxin);
  SEND_BYTES(par_en,par_err,dbus,dcomp,0,0,1,"With parity error inserted",bclk,rxin);
  SEND_BYTES(par_en,par_err,dbus,dcomp,0,1,0,"With missing stoppbit",bclk,rxin);
  SEND_BYTES(par_en,par_err,dbus,dcomp,1,0,0,"With startbit noise",bclk,rxin);
  wait;
end process;


--Modell for shiftregisteret og mottaksbuffer i figur 1
--ikke krav om dette i testbenk
SHIFT_REGISTER:
process(rst,bclk)
begin
  if rst = '1' then
    dout <= (others => '0');
  elsif rising_edge(bclk) then
    if shift_en = '1' then
      dout(7) <= rxin;
      for i in 6 downto 0 loop
        dout(i) <= dout(i+1);
      end loop;
    end if;
    if data_en = '1' then
      dbus <= dout;
    end if;
  end if;
end process;

end architecture TB_UART_RX_CTRL;
```