Institutt for Elektronikk og telekommunikasjon
Department of Electronics and telecommunication

Eksamensoppgave i TFE4141 – Design av Digitale System 1

Examination paper for TFE4141 – Design of Digital Systems 1

Faglig kontakt under eksamen /

Academic contact during the examination: Øystein Gjermundnes

Tlf. / Phone: 41318536

Eksamensdato / Examination date: Torsdag / Thursday 08.12.2016

Eksamenstid (fra-til) / Examination time (from-to): 09:00 – 13:00

Hjelpemiddelkode/Tillatte hjelpemidler: A – Alle trykte og håndskrevne hjelpemidler tillatt. Alle kalkulatorer tillatt.

Permitted examination support material: A- All printed and hand-written support material is allowed. All calculators are allowed.

Informasjon om trykking av eksamensoppgave

Originalen er:

1-sidig X **2-sidig** □

sort/hvit □ **farger** X

skal ha flervalgskjema □

Annen informasjon/Other information:

Benytt figurer og tabeller ***i oppgaveteksten*** i størst mulig utstrekning ved løsning av oppgaven. Skriv inn svar på ***angitt plass***. Lever oppgaveteksten med dine svar som din besvarelse. Bruk ekstra ark om nødvendig.

Use figures and tables in the **exam set** as much as possible for your answers. Fill in your answers in the specified locations and submit the examination set. Continue on separate sheets if needed.

Maksimalt antall poeng for hver oppgave og hvert punkt er **gitt i parentes**.

Maximum points achievable for each problem is given in **parenthesis.**

Maksimalt antall poeng oppnåelig totalt: **100**.

Maximum points achievable in total: **100**.

Språk/Language: Norsk(Bokmål)/Norwegian og/and Engelsk/English

Antall sider / Number of pages: 23

Antall sider vedlegg / Number of pages enclosed:

Kontrollert av / Checked by:

Dato / Date Sign

Problem 1: To diagrammer (5 poeng)

Se på Figure 1. Den øverste delen av figuren viser en tilstandsmaskin og den nederste delen viser en nettliste. Begge diagrammene representerer en digital krets. **Representerer disse to diagrammene samme digitale krets? Forklar hvorfor/hvorfor ikke.**

Problem 1: To diagrams (5 points)

Have a look at Figure 1. The upper part of the figure shows a state machine and the bottom shows a netlist. Both diagrams represent a digital circuit. **Do these two diagrams represent the same digital design? Explain why/why not.**

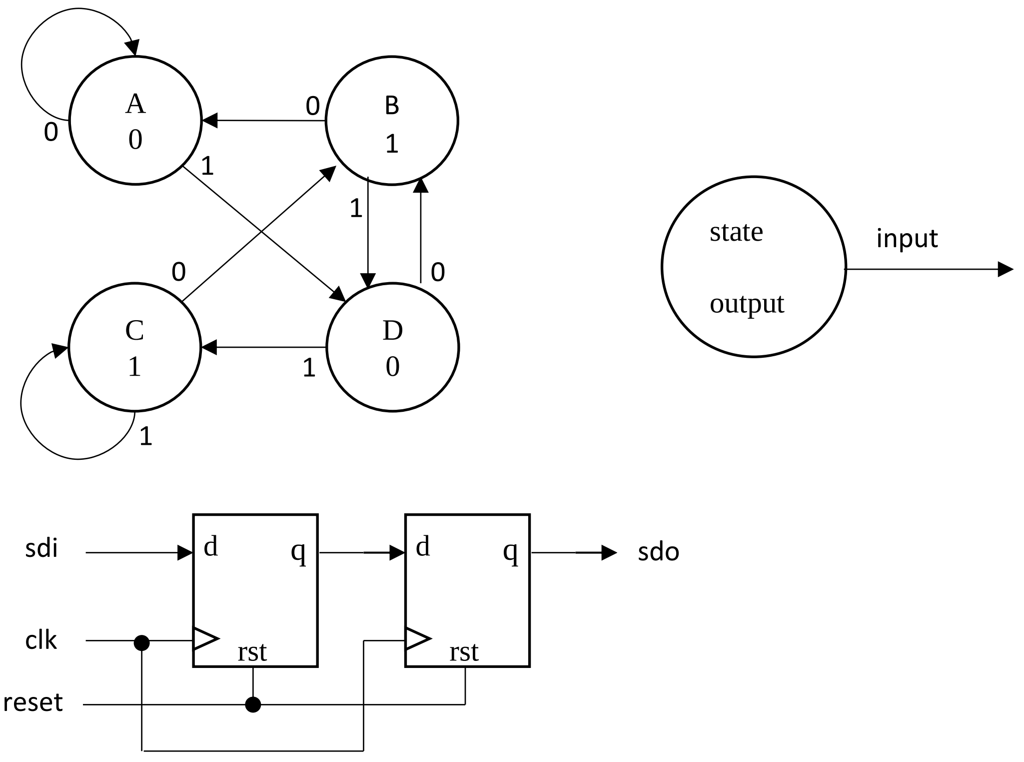


Figure 1. State machine and netlist.

Yes, they both represents a shift register provided that the:

- state machine is implemented using two flip flops for storing the state
- encoding for the state machine is as follows: A=00, B=01, C=11, D=10

Problem 2: Energieffektive kretser (15 poeng)

I denne oppgaven skal du finne frem til designvalg som leder frem til de mest energieffektive kretsene.

Problem 2: Energy efficient circuits (15 points)

In this problem, you will look for the design choices that leads to the most energy efficient circuits.

a) (5 poeng/points)

Figure 2 viser en FSM med to alternative tilstandstilordninger (A og B). **Hvilken tilstandstilordning bør man velge for å være sikker på at tilstansmaskinen bruker så lite energi som mulig. Forklar hvordan du tenker.**

Figure 2 shows an FSM with two alternative state encodings (A and B). **Which of the two encodings should you choose in order to ensure that your state machine consume as little energy as possible. Explain your reasoning.**

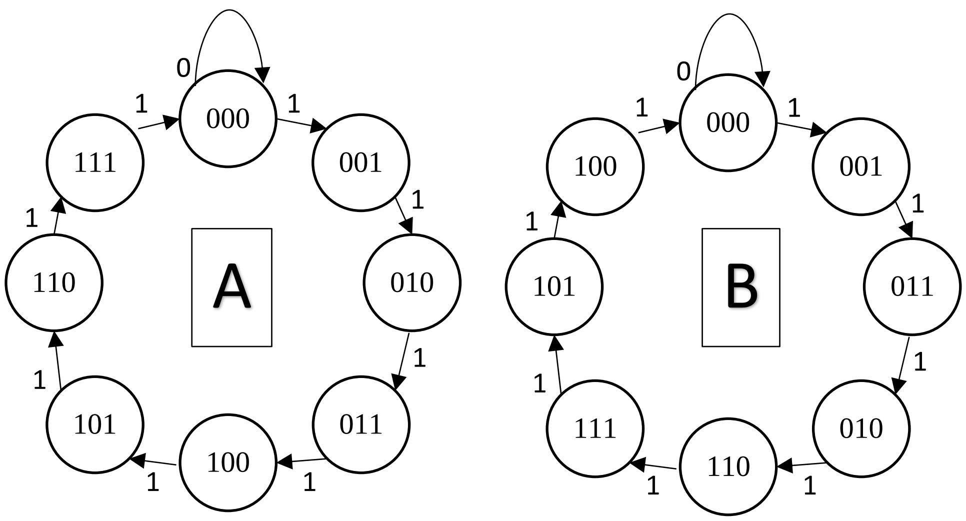


Figure 2. Two different state encodings for the same FSM.

Only one bit is toggling for each state transition of state diagram B. In state diagram A all of the bits toggle for some state transitions.

Dynamic energy is consumed when nodes in the design toggle. By ensuring as little toggling as possible in the design, the dynamic energy consumption is minimized.

The gray coded counter B is therefore the most energy efficient state encoding.

b) (5 poeng/points)

Figure 3 viser et lite design der de to inngangene `a` og `b` legges sammen for å produsere summen `y`. Det er registre på alle inn- og utganger. Signalet `di_valid` er aktivt når inngangene `a` og `b` har gyldige data. Signalet `do_valid` er aktivt når resultatet `y` er gyldig. Figure 4 viser RTL koden for designet.

Figure 3 shows a small design where the two inputs `a` and `b` are added and produces the sum `y`. All inputs and outputs are registered. The `di_valid` signal is high when `a` and `b` are valid and the `do_valid` signal is high when the output `y` is valid. Figure 4 contains the RTL code for the design.

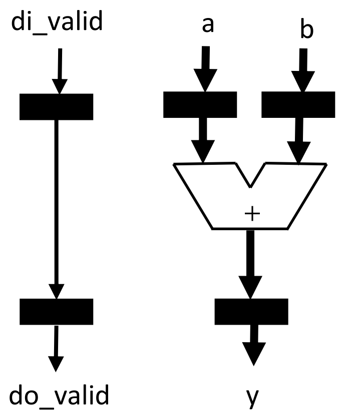


Figure 3. Adder with registers on inputs and outputs.

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

entity problem_2b is
  port ( clk, reset_n : in std_logic;
         di_valid     : in std_logic;
         a, b         : in std_logic_vector (15 downto 0);
         do_valid     : out std_logic;
         y            : out std_logic_vector (15 downto 0)
       );
end problem_2b;

architecture rtl of problem_2b is
  signal s1_valid, s2_valid          : std_logic;
  signal s1_a, s1_b, s1_sum, s2_sum  : std_logic_vector(15 downto 0);
begin


  ctrl: process(clk, reset_n)
  begin
    if(reset_n = '0') then
      s1_valid <= '0';
```

```
      s2_valid <= '0';
    elsif(clk'event and clk='1') then
      s1_valid <= di_valid;
      s2_valid <= s1_valid;
    end if;
  end process;
  do_valid <= s2_valid;

  input_reg: process(clk)
  begin
    if(clk'event and clk='1') then
      s1_a <= a;
      s1_b <= b;
    end if;
  end process;
  s1_sum <= std_logic_vector(unsigned(s1_a) +
                             unsigned(s1_b));

  output_reg: process(clk)
  begin
    if(clk'event and clk='1')  then
      s2_sum <= s1_sum;
    end if;
  end process;
  y <= s2_sum;

end rtl;
```

Figure 4. RTL code for the design in Figure 3.

Foreslå en endring av prosessene `input_reg` og `output_reg` slik at energiforbruket reduseres.

Suggest a change to the `input_reg` and `output_reg` processes so that the energy consumption is reduced.

```
input_reg: process(clk)
begin
  if(clk'event and clk='1') then
    if(di_valid='1') then
      s1_a <= a;
      s1_b <= b;
    end if;
  end if;
end process;

output_reg: process(clk)
begin
  if(clk'event and clk='1')  then
    if(s1_valid='1') then
      s2_sum <= s1_sum;
    end if;
  end if;
end process;
```

Figure 5. Use this textbox when suggesting improvements to the RTL code in problem 2b).

c) (5 poeng/points)

Forklar hvorfor endringen du gjorde i problem 2b) vil redusere energiforbruket til designet.

Explain why the change you did in Problem 2b) will reduce the energy consumption of the design.

The energy consumption will be reduced because of two reasons:

- The flip flops in the input and output registers are only updated when data is valid. Unnecessary toggling is therefore avoided and the dynamic energy consumption reduced.
- The synthesis tool will be able to infer clock gates for the flip flops in the input and output registers. This will remove unnecessary toggling of the clock pins of the flip flops.

Problem 3: Maksimer throughput (10 poeng)

Frank er en flink designer, men nå har han et problem han ikke finner ut av. Han har lyst til å lage en modul som tar inn en strøm av data pakker. Disse pakkene er av to typer **C**ompute(C) og **B**ypass(B). **C** pakkene skal det gjøres beregninger på før de sendes ut, mens **B** pakkene skal sendes rett gjennom modulen. Det er viktig at pakkene kommer ut i samme rekkefølge som de kommer inn og at en ny pakke mottas og sendes ut hver eneste klokkesyklus. Figure 6 viser strømmen av pakker inn og ut av modulen.

Frank har laget designet i Figure 7, men uansett hvordan han kontrollerer mux-en så ender han opp med å måtte stoppe pakkestrømmene innimellom.

Problem 3: Maximize throughput (10 points)

Frank is a capable designer, but now he has a problem he struggles with. He has been given the task of designing a module that accepts a stream of data packets. Two types of packets exist: **C**ompute (C) packets and **B**ypass(B) packets. The design will do computations on the C packets, whereas the B packets will be sent straight through the module. It is important that the packets are sent out of the module in the same order as they came in and that new packets are accepted and sent out on every clock cycle. Figure 6 shows the stream of packets in and out of the module.

Frank has made the design in Figure 7, but no matter how he controls the mux, he ends up in a situation where he has to stop the packet stream occasionally.

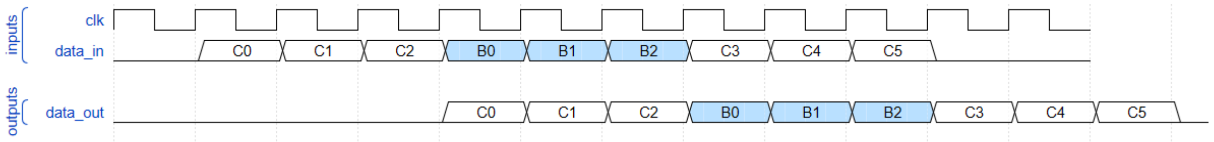


Figure 6. Stream of input packets and output packets.

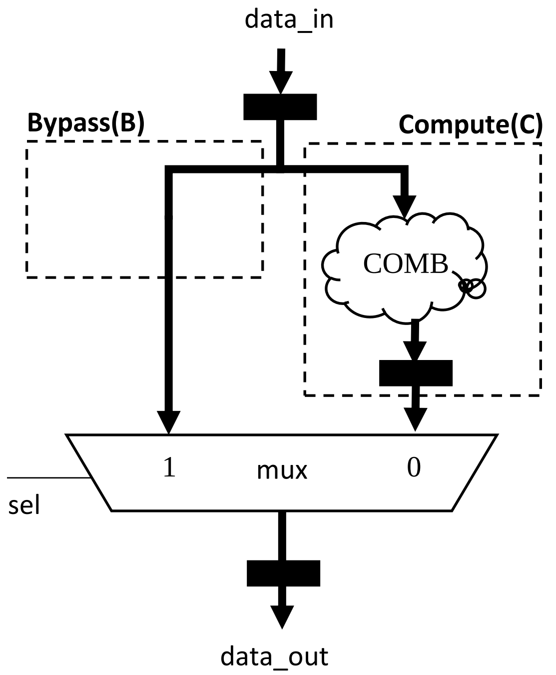
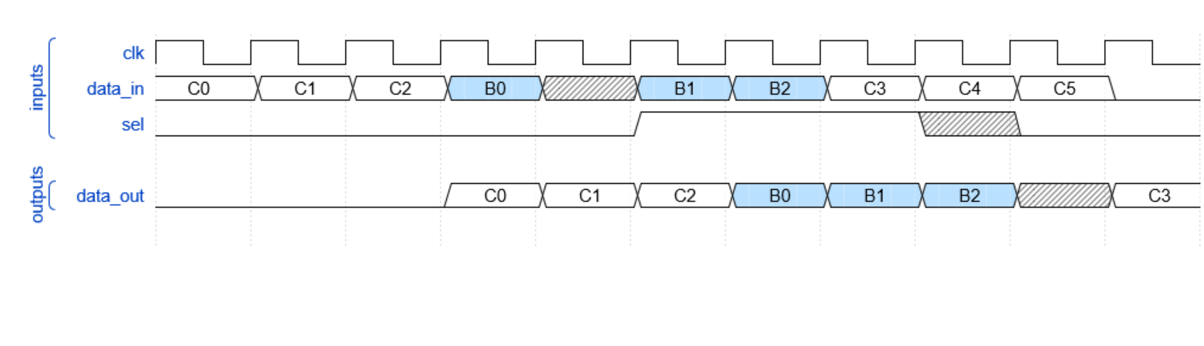


Figure 7. Packet processor in problem 3.

a) (5 poeng/points)

Lag et bølgediagram (waveform) som viser en situasjon der Frank's design vil medføre at pakkestrømmen må stoppes.

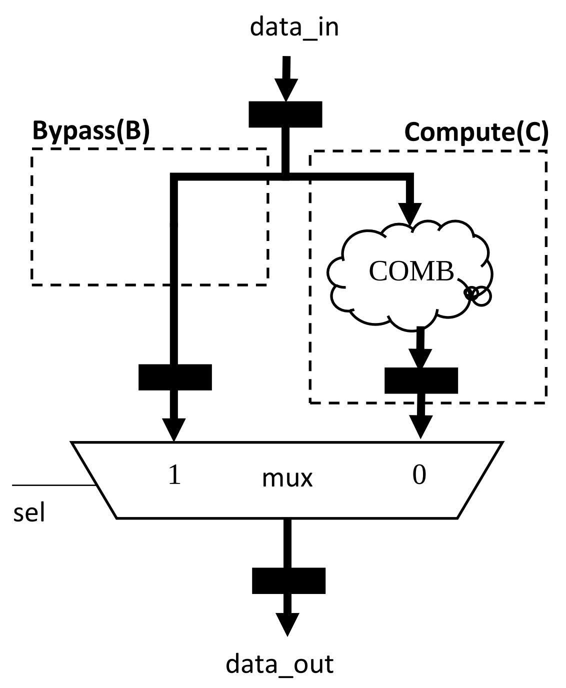
Create a waveform that shows a situation where Frank's design will have to stall the packet stream.



b) (5 poeng/points)

Foreslå en endring av designet til Frank slik at han unngår bobler i pakkestrømmene. Tegn et nytt blokkdiagram nedenfor.

Suggest a modification of Frank's design so that he avoids inserting bubbles in the packet stream. Draw a new block diagram below.



Problem 4: Design av krypterings krets (25 poeng)

Didrik og Jonas har designet en krets, `DJC_Core`, som kan kryptere og dekryptere meldinger. Overordnet blokkdiagram er vist i Figure 8 sammen med bølgediagrammer i Figure 9 som viser hvordan data flyter inn og ut av modulen.

For å forsterke krypteringen bestemmer de seg for å implementere Cipher Block Chaining (CBC) på toppen av `DJC_Core`. I CBC mode kryptering blir hver klartekst (PT_i) blokk XORet med den forrige siffertekstblokken (CT_{i-1}) før den krypteres og produserer en ny siffertekstblokk (CT_i). CBC mode kryptering er vist i Figure 10.

Dekryptering foregår ved at man dekrypterer siffertekst blokken (CT_i) og deretter XORer resultatet med forrige siffertekstblokk (CT_{i-1}) for å produsere klarteksten (PT_i). CBC mode dekryptering er vist i Figure 11.

XOR-operasjonene gjøres bitvis mellom bit j i en blokk og bit j i den andre blokka.

Problem 4: Design of an encryption circuit (25 points)

Didrik og Jonas have designed an encryption circuit, `DJC_Core`, that can encrypt and decrypt messages. A top-level diagram is shown in Figure 8 together with waveforms in Figure 9 that shows how data flows in and out of the module.

In order to strengthen the encryption, they decide to implement Cipher Block Chaining (CBC) on top of `DJC_Core`. In CBC mode encryption, every plaintext block (PT_i) is XORed with the previous ciphertext block (CT_{i-1}) before it is encrypted and produces a new ciphertext block (CT_i). CBC mode encryption is shown in Figure 10.

Decryption is done by first decrypting the ciphertext (CT_i) and then subsequently XOR the result with the previous ciphertext block (CT_{i-1}) in order to produce the plaintext (PT_i). CBC mode decryption is shown in Figure 11.

The XOR-operations are done bitwise between bit j in one block and bit j in the other block.

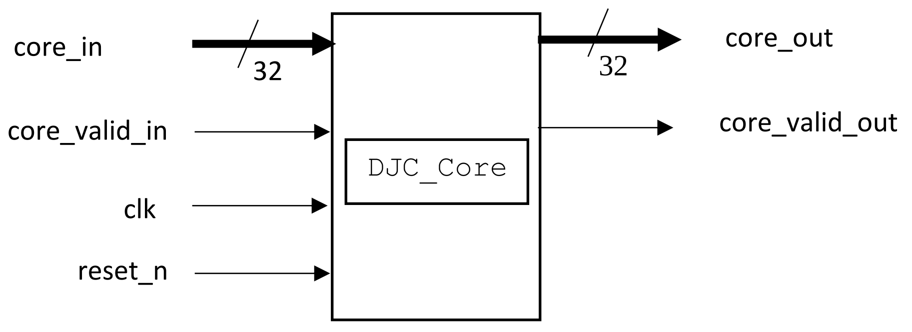


Figure 8. The `DJC_Core` core entity.

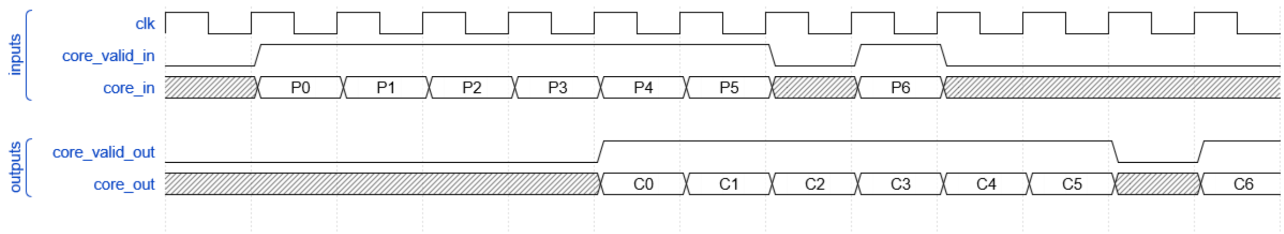
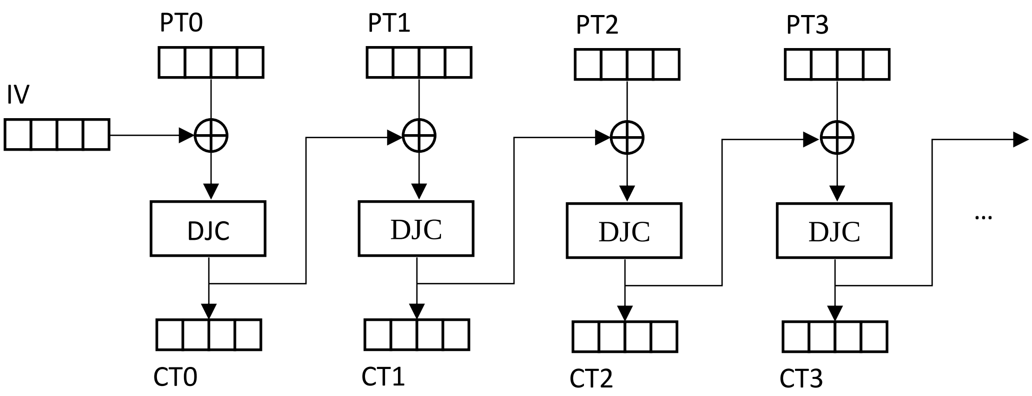


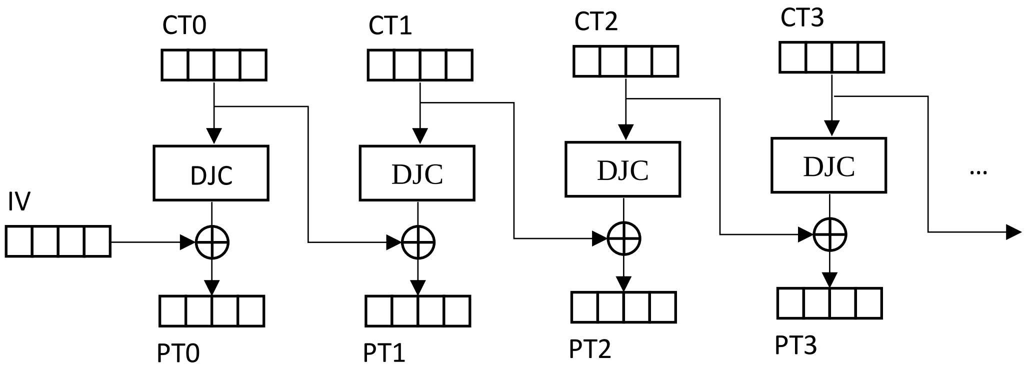
Figure 9. Waveform illustrating the relationship between interface signals of the `DJC_Core`.



CBC mode **encryption**

IV: Initialization vector, PT: Plain text, CT: Ciphertext

Figure 10. CBC mode encryption.



CBC mode **decryption**

IV: Initialization vector, PT: Plain text, CT: Ciphertext

Figure 11. CBC mode decryption.

a) (5 poeng/points)

Kan CBC mode kryptering parallelliseres? Begrunn svaret.

Can CBC mode encryption be parallelized? Explain your reasoning.

No
Encryption of block *i* cannot start before encryption of block *i-1* is completed.

b) (5 poeng/points)

Kan CBC mode dekryptering parallelliseres? Begrunn svaret.

Can CBC mode decryption be parallelized? Explain your reasoning.

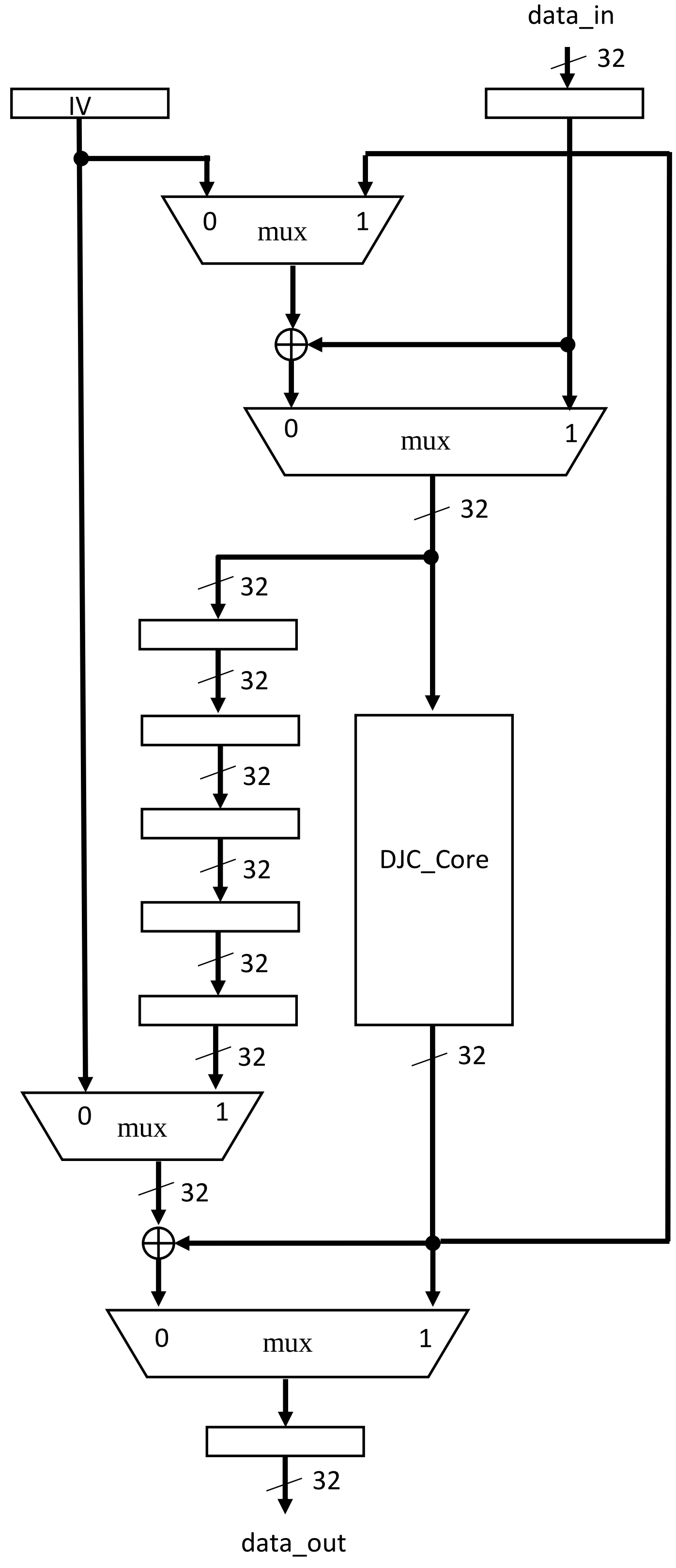
Yes
Decryption of block *i* can start immediately and before block *i-1* is decrypted.

c) (15 poeng/points)

Lag et design `CBC_Crypto` som kan utføre både CBC mode kryptering og dekryptering. **Tegn blokkdiagram for designet ditt. Fokuser på dataveien (datapath). Benytt `DJC_Core` som en submodul.**

Create a design `CBC_Crypto` that can perform both CBC mode encryption and decryption. **Draw a block diagram for your design. Focus on the datapath. Use `DJC_Core` as a submodule.**

CBC_Crypto BLOKK DIAGRAM / BLOCK DIAGRAM



Decryption performance: This design allows a new 32-bit message to be decrypted every cycle.
Encryption performance: The design allows a new 32-bit message to be encrypted every fourth clock cycle.

Problem 5: Kode evaluering (10 poeng)

En kollega av deg i firmaet TrafficLight Inc har laget en trafikklyskontroller for trafikklyset vist i Figure 12. RTL koden til trafikklyskontrolleren er vist i Figure 13. **Undersøk koden til kollegaen din og gjør en kode evaluering (code review). Beskriv hva som bør forbedres for at koden skal tilfredsstille bedriftens kvalitetskrav.**

Problem 5: Code review (10 points)

One of your colleagues in the company TrafficLight Inc has made a traffic light controller for the traffic light in Figure 12. The RTL code for the traffic light controller is listed in Figure 13. **Go through the code of your colleague and do a code review. Describe what needs to be improved so that the code satisfies the quality standards of the company.**

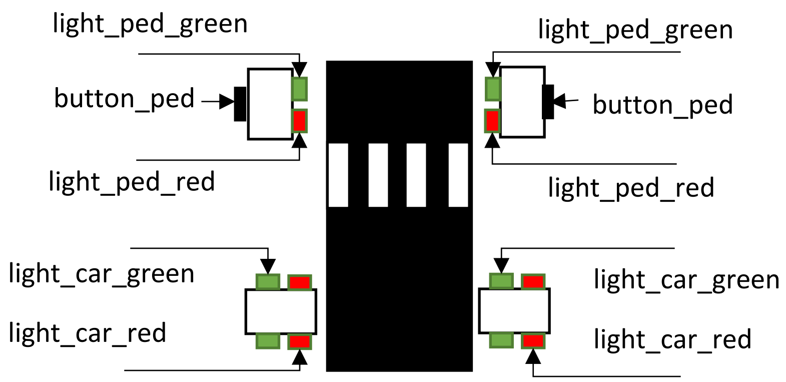


Figure 12. Traffic light controlled intersection.

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

-- TODO:
entity traffic_light_controller is
    port ( clk              : in   std_logic;
           reset_n          : in   std_logic;
           light_car_red    : out  bit;
           light_car_green  : out  bit;
           button_ped       : in   std_logic;
           light_ped_red    : out  std_logic;
           light_ped_green  : out  std_logic);
end traffic_light_controller;

architecture rtl of traffic_light_controller is
  type state_type is (e_GO_CAR, e_GO_PEDESTRIAN);
  signal curr_state, next_state : state_type;
  signal timer       : std_logic_vector(31 downto 0);
  signal timer_alarm : std_logic;
  signal start_timer : std_logic;
begin

  -- Clocked
  process(clk) begin
    if(clk'event and clk='1') then
      curr_state <= next_state;
      timer      <= timer + 1;
    end if;
  end process;
  timer_alarm <= and(timer);

  -- Compute next state
  process(curr_state, reset_n) begin
    if(reset_n='0') then
      light_car_red   <= '1';
      light_ped_red   <= '1';
      light_car_green <= '0';
      light_ped_green <= '0';
    end if;
    case curr_state is
      when e_go_car         =>
        if(button_ped = '1') then
          next_state <= e_GO_PEDESTRIAN;
          start_timer <= '1';
        end if;
        light_car_red   <= '0';
        light_car_green <= '1';

          when e_GO_pedestrian =>
if(timer_alarm = '1') then
next_state <= e_GO_CAR;
end if;
light_ped_red   <= '0';
light_ped_green <= '1';
end case;
end process;
end rtl;
```

- Missing description of purpose of the module and file header
- TODOs in the code must be removed/fixed

Inconsistent use of std_logic / bit. Should use std_logic for all outputs.

- start_timer not used.

- Missing reset of curr_state and timer. These are control signals and must be reset in order to put the design in a known when it is powered on.
- Missing explanation/documentation of process.
- Addition is not defined std_logic_vector (timer).

- Timer_alarm <= and(timer);
  This is actually OK. The and function is computing the and operation of all bits in timer (VHDL-2008).

- **This is a combinational process. It does not make sense to try to reset combinational logic. Only flops can be reset.**
- Missing explanation/documentation of process.
- **Timer_alarm is missing from the sensitivity list. This leads to latches.**

- Use capital letter for enums or at least be consistent throughout the code.

- Horrybly unaligned code. Impossible to read.
- Missing default state.
- **Signals are not given a value in all states. This leads to latches.**
- timer_alarm is not in the sensitivity list

Figure 13. RTL code to be reviewed in Problem 5.

KODE EVALUERING / CODE REVIEW

- See comments in Figure 13.
- The candidate should mention at least 10 issues. Among these must be the important issues highlighted with bold red.

Problem 6: Design prosess (10 poeng)

Du har fått i oppdrag å designe en RSA krypteringskrets. Denne krypteringskretsen skal implementeres som ASIC i en ny 10nm prosess i en av TSMC sine fabrikker. **Beskriv kort designprosessen fra prosjektet begynner til du har fått tilbake kretser fra fabrikken.**

Problem 6: Design process (10 points)

You have been given the task of designing a RSA encryption circuit. This encryption circuit will be implemented as ASIC in a new 10nm process in one of TSMCs fabrics. **Give a short description of the design process from the project starts until you have circuits back from the fabric.**

BESKRIVELSE AV DESIGN PROSESS / DESCRIPTION OF DESIGN PROCESS

- Establish the requirements for the design. Functional, interface, performance, power, energy, area requirements are important types of requirements.
- Start the design exploration phase. The goal is to come up with a suitable microarchitecture that is likely to satisfy all the requirements.
  - Suggest an initial microarchitecture
  - Model the design in a high level language. Check that the model produces a functionally correct result.
  - Run tests and extract performance metrics from the model. Estimate/measure also other attributes such as energy consumption and area.
  - Compare the measured/estimated metrics against the requirements. Adjust the microarchitecture and update the model if requirements are not met.
- Conduct reviews of the proposed microarchitecture. Get valuable feedback from your colleagues. Adjust the microarchitecture if needed.
- Create detailed block specifications and verification plans. Review the documents.
- Write the RTL code for the design.
- Create testbenches or formal verification environments. Use these to verify that the design is functionally correct.
- Synthesize the design.
- Measure area, frequency, performance, energy and power on the RTL implementation. Adjust the RTL code and the microarchitecture if requirements are not met.
- Create a FPGA prototype and run system tests on that.
- Insert scan flip-flops. Create test vectors for use during production testing.
- Run place & route and create layout.
- Generate GDSII files for the layout and send them to the fab.
- Create mask sets at the fab.
- Manufacture the design.

- Production testing of manufactured chips
- Insert fully working chips into different types of products. (E.g. smart watches, light bulbs and cell phones).
- Carefully planning is necessary through the whole project. Planning includes breaking the work down into manageable tasks, estimate the effort needed to do all tasks, set milestones. Tracking progress against the plan is also important.

Max points can be achieved if all the yellow terms are mentioned.

Problem 7: Design av addisjonskrets (25 points)

Problem 7: Design of an adder circuit (25 points)

a) (5 poeng/points)

Den kritiske stien til designet i Figure 14 er lang. **Foreslå en endring slik at den kritiske stien reduseres uten at oppførselen til kretsen endres. Tegn et blokkdiagram for den nye kretsen.**

The critical path for the design in Figure 14 is very long. **Suggest a change to the design so that the critical path is reduced without any change in the behavior of the design. Draw a block diagram for the new design.**

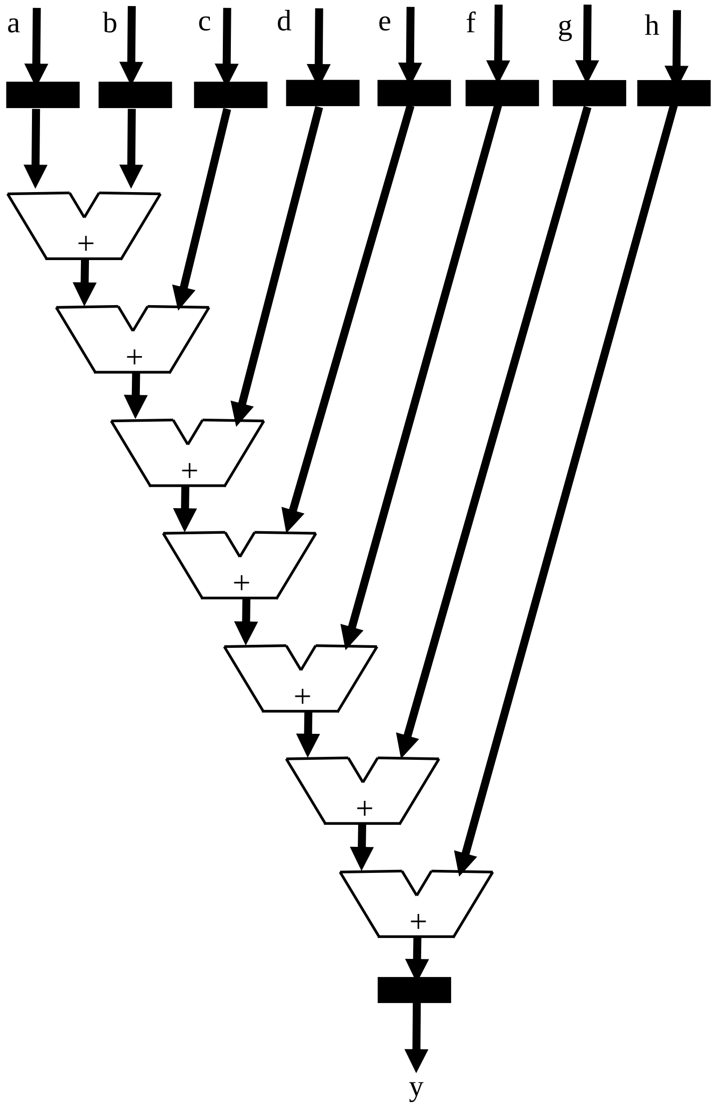
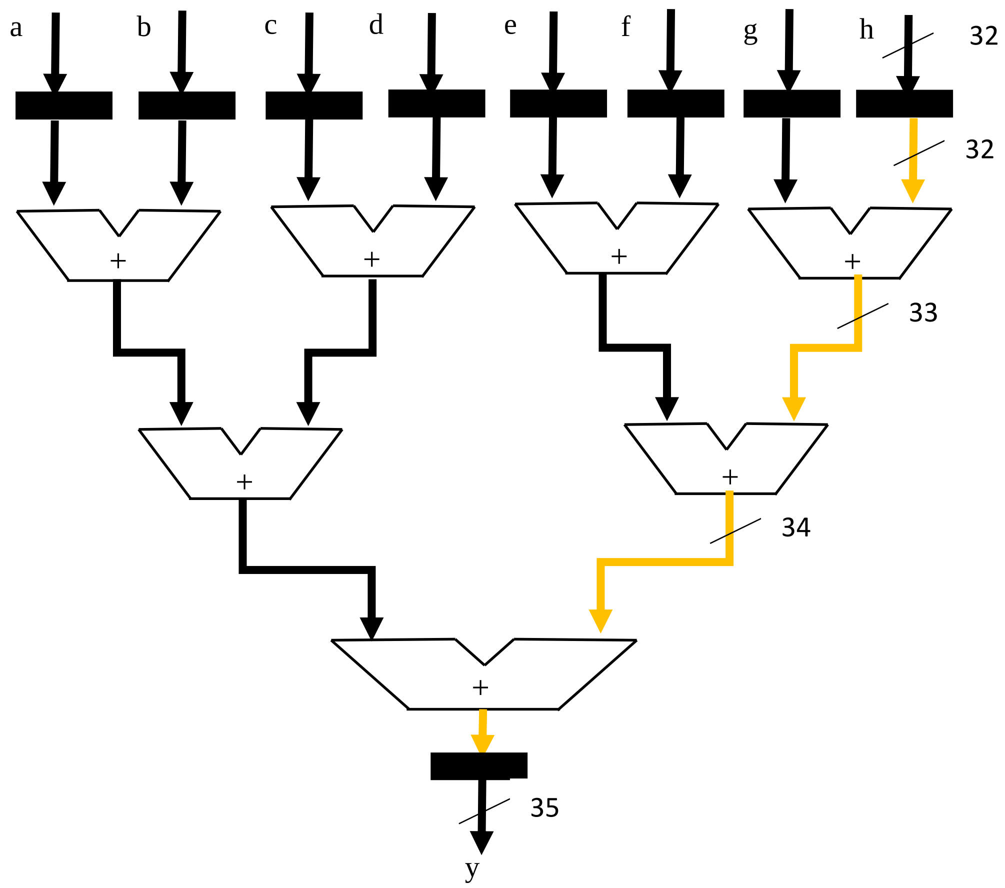


Figure 14. Design with long critical path.

BLOKKDIAGRAM FOR RASKERE KRETS / BLOCK DIAGRAM FOR FASTER CIRCUIT



b) (5 poeng/points)

Alle inngangene (a, b, c, d, e, f, g, h) til kretsen er 32-bit. **Hvor mange bit må addererne i designet ditt i Problem 7a) ha for at man skal unngå overflyt?**

All the inputs (a, b, c, d, e, f, g, h) to the design are 32-bit. **How many bits are needed for each adder in your design from Problem 7a) in order to avoid overflow?**

BITS PER ADDERER / BITS PER ADDER

Definition: An n-bit adder is an adder that is able to add two n-bit operands and produce a n+1 bit result. (n-sum bits and one carry out = n+1 bits).

The adders in the first stage must be 32 bit.
The adders in the second stage must be 33 bit.
The adders in the final stage must be 34 bits.

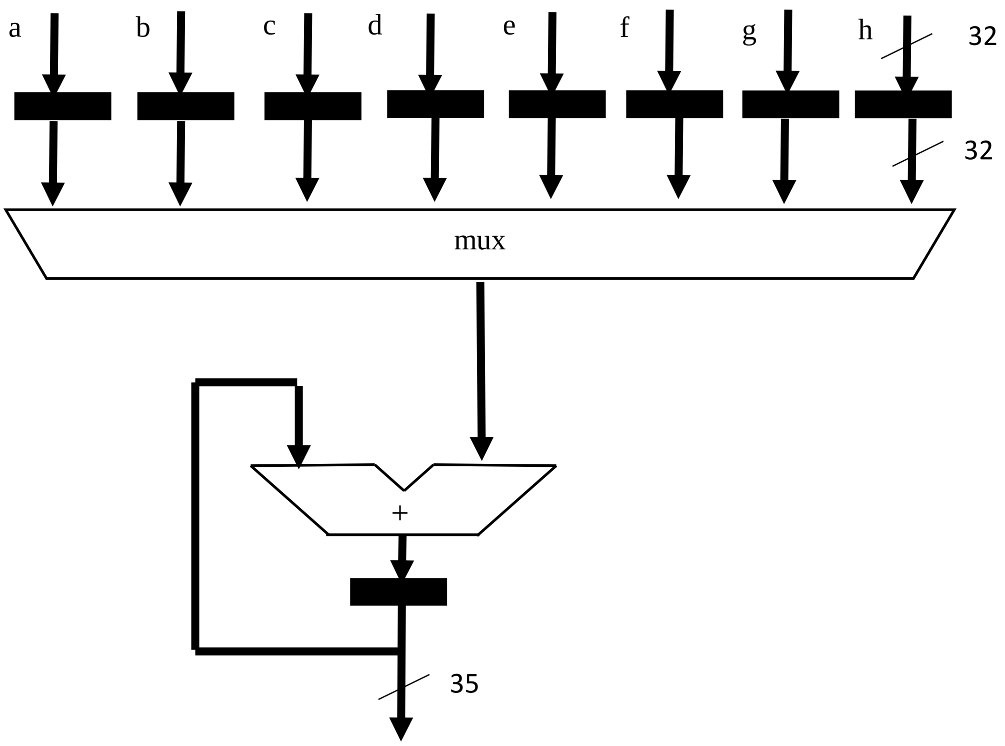
A 32 bit adder with carry out is often implemented in VHDL as a 33 bit adder with the most significant bit of the inputs tied off to 0.

c) (5 poeng/points)

Designet fra Problem 7a) er alt for stort. Det er nødvendig å redusere arealet selv om det vil medføre redusert ytelse. **Bruk bare en enkelt adderer og tegn et blokkdiagram for dataveien (the datapath) til det areal optimaliserte designet ditt.**

The design from Problem 7a) is too big. It is necessary to reduce the area even though this will result in reduced performance. **Use only a single adder, and draw a block diagram for the datapath of your area-optimized design.**

AREAL OPTIMALISERT DESIGN / AREA OPTIMIZED DESIGN



d) (10 poeng/points)

Skriv RTL kode for det areal-optimaliserte designet i Problem 7c. Skriv kun RTL kode for dataveien (datapath).

Write RTL code for the area-optimized design in Problem 7c. Only write RTL code for the datapath.

RTL KODE / RTL CODE

```
--------------------------------------------------------------------------------
-- Solution example of problem7d
-- Module that computes the sum of the inputs a,b,c,d,e,f,g,h.
-- Only processes that implements the datapath are included.
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity problem7d is
  port ( clk             : in std_logic;
         reset_n         : in std_logic;
         a,b,c,d,e,f,g,h : in std_logic_vector (31 downto 0);
         data_valid_in   : in std_logic;
         data_valid_out  : out std_logic;
         y               : out std_logic_vector (34 downto 0));
end problem7d;

architecture rtl of problem7d is
  -- Input registers
  signal a_r, b_r, c_r, d_r, e_r, f_r, g_r, h_r : std_logic_vector(31 downto 0);
  -- Signals used for selecting operand to the adder
  signal operand_a                              : std_logic_vector(31 downto 0);
  signal operand_sel                            : std_logic_vector(2 downto 0);
  -- Signals used for implementing the accumulator
  signal acc_nxt, acc_r                         : unsigned(34 downto 0);
  -- Control signals for initializing the accumulator and updating the
  -- accumulator
  signal init_acc, update_acc                   : std_logic;
begin

  -- Input registers
  process(clk, reset_n) begin
    if(reset_n='0') then
      a_r <= (others =>'0'); b_r <= (others =>'0');
      c_r <= (others =>'0'); d_r <= (others =>'0');
      e_r <= (others =>'0'); f_r <= (others =>'0');
      g_r <= (others =>'0'); h_r <= (others =>'0');
    elsif(clk'event and clk='1') then
      if(data_valid_in='1') then
        a_r <= a; b_r <= b; c_r <= c; d_r <= d;
        e_r <= e; f_r <= f; g_r <= g; h_r <= h;
      end if;
    end if;
  end process;

  -- Accumulator register
  process(clk, reset_n) begin
    if(reset_n='0') then
      acc_r <= (others => '0');
    elsif(clk'event and clk='1') then
      if(update_acc='1') then
        acc_r <= acc_nxt;
      end if;
    end if;
  end process;

  -- Mux for picking one of the operands from the input registers
  process(a_r, b_r, c_r, d_r, e_r, f_r, g_r, h_r, operand_sel) begin
    case(operand_sel) is
```

```
      when "000" =>
        operand_a <= a_r;
      when "001" =>
        operand_a <= b_r;
      when "010" =>
        operand_a <= c_r;
      when "011" =>
        operand_a <= d_r;
      when "100" =>
        operand_a <= e_r;
      when "101" =>
        operand_a <= f_r;
      when "110" =>
        operand_a <= g_r;
      when others => -- "111" =>
        operand_a <= h_r;
    end case;
  end process;

  -- Combinational process for the adder
  process(acc_r, operand_a, init_acc) begin
    if(init_acc = '1') then
      acc_nxt <= (others => '0');
    else
      acc_nxt <= unsigned("000" & operand_a) + acc_r;
    end if;
  end process;

  -- Drive the output signal
  y <= std_logic_vector(acc_r);

end rtl;
```