NTNU
Kunnskap for en bedre verden

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: Lørdag / Saturday 19.12.2015

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

Hjelpemiddelkode/Tillatte hjelpemidler: C – Spesifiserte trykte og håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.

Spesifisert liste over godkjente hjelpemidler:

- Trykte og håndskrevne hjelpemidler: Alle.
- Kalkulator: Casio fx-82ES PLUS, Citizen SR-270X og Citizen SR-270X College, Hewlett Packard HP30S

Permitted examination support material: C- Specified printed and hand-written support material is allowed. A specific basic calculator is allowed.

Specified list of permitted support material:

- Printed and hand-written material: All.
- Calculator: Casio fx-82ES PLUS, Citizen SR-270X and Citizen SR-270X College, Hewlett Packard HP30S

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: 26

Antall sider vedlegg / Number of pages enclosed:

Kontrollert av / Checked by:

Dato / Date Sign

Problem 1: Beskriv vanlige digitale byggeblokker (5 poeng)

Tabell 1 inneholder en oversikt over vanlige digitale byggeblokker. Fyll ut tabellen med følgende informasjon:

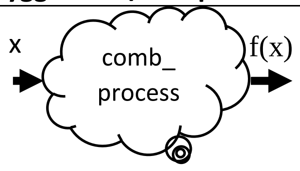
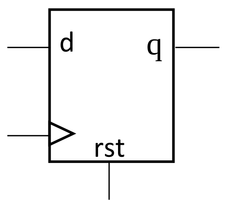
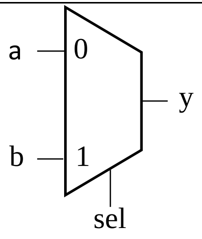
- Navn på byggeblokk
- Er dette en Sekvensiell (**S**) eller Kombinatorisk(**C**) krets? (Indiker dette med bokstavene **S** og **C** i **S/C** kolonnen).
- Beskriv funksjonaliteten (Max 20 ord)

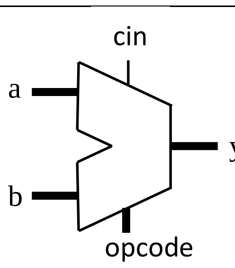
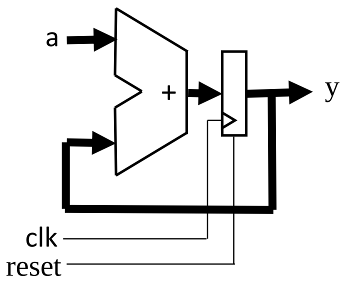
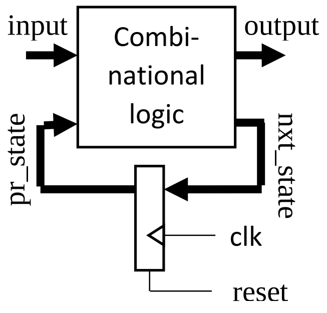
Problem 1: Describe common components (5 points)

For the digital components in Tabell 1 Vanlige digitale byggeblokker /Table 1 below, fill out the following:

- Name of component
- Is it a **S**equential or **C**ombinational circuit? (Indicate this with **S** and/or **C** in the **S/C** column).
- Describe the functionality (Max 20 words)

Tabell 1 Vanlige digitale byggeblokker /Table 1. Common digital components.

	Byggeblokk/Component	**Navn/Name**	**S/C**	**Funksjonalitet/Functionality**
Eksempel/ Example		General combinational block	C	Representation of combinational block/process with *n* inputs and *m* outputs that implements some function *f*.
a)		D - flip-flop	S	Value of d stored and available for reading on the output q after positive edge of clock.
b)		mux	C	Route either *a* or *b* out to the output *y* depending on the select signal *sel*.

c)		Arithmetic logic unit (ALU)	C	Unit that can typically add/subtract/multiply/compare two numbers.
d)		Accumulator	C+S	Unit that is able to serially add a stream of numbers: $y_i = y_{i-1} + a_i$
e)	 input Combi-national logic output pr_state nxt_state clk reset	State machine	C+S	Hardware representation of a finite state machine. Useful for implementing controllers. The next state and the output are functions of the present state and the input.

Problem 2: Tolking av VHDL kode (20 poeng)

Gjør følgende for alle kodesnuttene nedenfor:

- Lag blokkdiagram som illustrerer kretsen VHDL koden representerer.
- Beskriv funksjonen til kretsen (maks 20 ord).
- Tell antallet vipper som blir inferert ved syntese.
- Fullfør bølgeformen.

Problem 2: VHDL code interpretation (20 points)

For all code samples below, do the following:

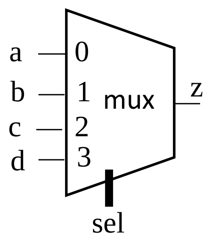
- Create small block diagrams/netlists that illustrates the circuit that the code represents.
- Describe the functionality (max 20 words).
- Count the number of flip-flops that will be inferred during synthesis
- Complete the waveform.

a)

```
library ieee;
use ieee.std_logic_1164.all;
entity problem_2a is
  port ( sel :  in std_logic_vector(1
downto 0);
         a   :  in std_logic;
         b   :  in std_logic;
         c   :  in std_logic;
         d   :  in std_logic;
         z   : out std_logic);
end problem_2a;
```

```
architecture rtl of problem_2a is
begin
  process(a, b, c, d, sel)
  begin
    case sel is
      when "00"      =>  z <= a;
      when "01"      =>  z <= b;
      when "10"      =>  z <= c;
      when "11"      =>  z <= d;
      when others    =>  z <= 'X';
    end case;
  end process;
end rtl;
```

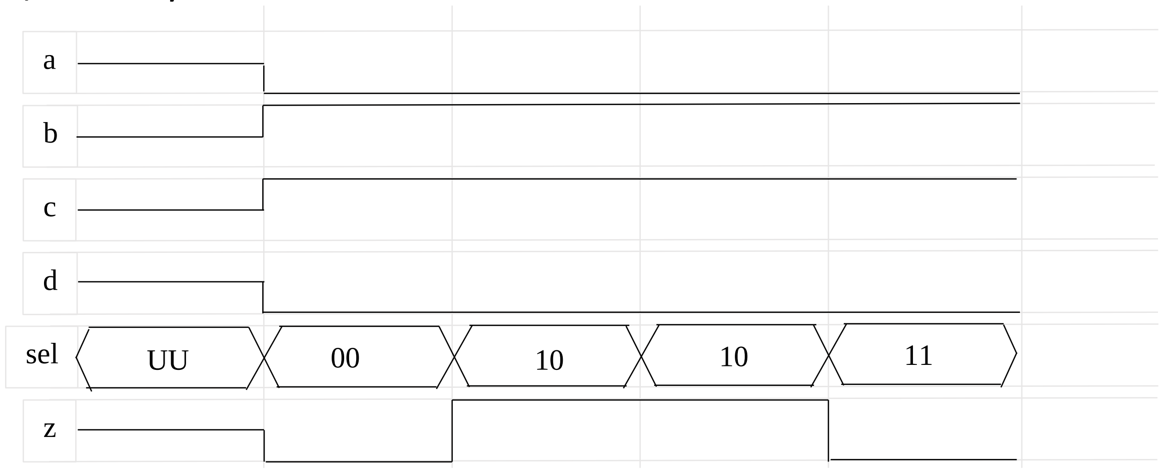
BLOKKDIAGRAM / BLOCK DIAGRAM:



FUNKSJON / FUNCTIONALITY:
4x1 mux

ANTALL VIPPER INFERERT VED SYNTESE / NUMBER OF FLIP-FLOPS INFERRED DURING SYNTHESIS:
0

BØLGEFORM / WAVEFORM:

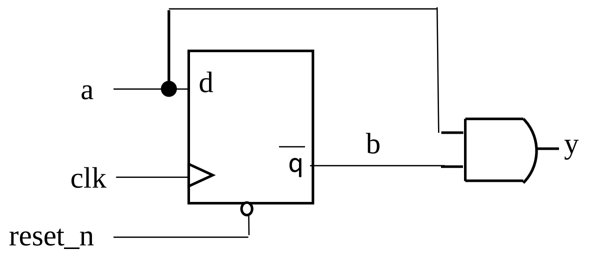


b)

```
library ieee;
use ieee.std_logic_1164.all;
entity problem_2b is
  port ( clk      :  in  std_logic;
         reset_n  :  in  std_logic;
         a        :  in  std_logic;
         y        :  out std_logic);
end problem_2b;
```

```
architecture rtl of problem_2b is
  signal b : std_logic;
begin
  process(clk, reset_n)
  begin
    if(reset_n = '0') then
      b <= '0';
    elsif (clk'event and clk='1') then
      b <= a;
    end if;
   end process;
   y <= (not b) and a;
 end rtl;
```

BLOKKDIAGRAM / BLOCK DIAGRAM:



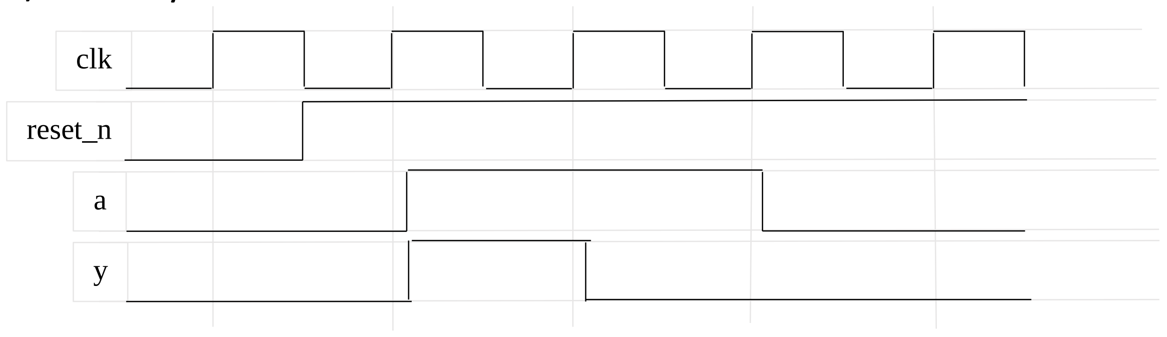
FUNKSJON / FUNCTIONALITY:

Edge detector for detecting transitions from ‘0’ to ‘1’.

ANTALL VIPPER INFERERT VED SYNTESE / NUMBER OF FLIP-FLOPS INFERRED DURING SYNTHESIS:

1

BØLGEFORM / WAVEFORM:

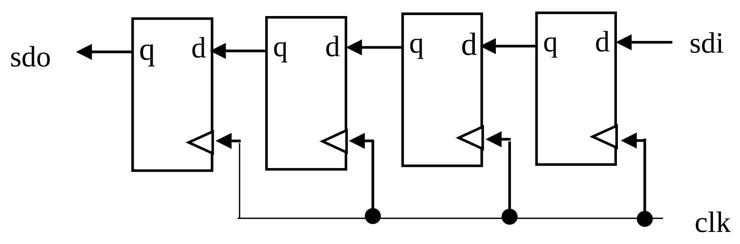


c)

```
library ieee;
use ieee.std_logic_1164.all;
entity problem_2c is
  port ( clk :  in  std_logic;
         sdi :  in  std_logic;
         sdo :  out std_logic);
end problem_2c;
```

```
architecture rtl of problem_2c is
  signal data_nxt: std_logic_vector(3 downto 0);
  signal data_r  : std_logic_vector(3 downto 0);
begin

  process(clk)
  begin
    if(clk'event and clk='1') then
      data_r <= data_nxt;
    end if;
  end process;

  process(sdi, data_r)
  begin
    data_nxt <= data_r(2 downto 0) & sdi;
  end process;

  sdo <= data_r(3);

end rtl;
```

BLOKKDIAGRAM / BLOCK DIAGRAM:



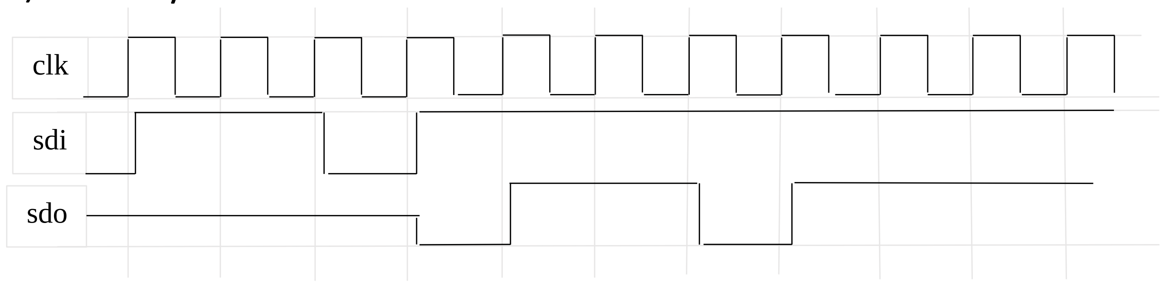
FUNKSJON / FUNCTIONALITY:

Four bit shift-register.

ANTALL VIPPER INFERERT VED SYNTESE / NUMBER OF FLIP-FLOPS INFERRED DURING SYNTHESIS:

4

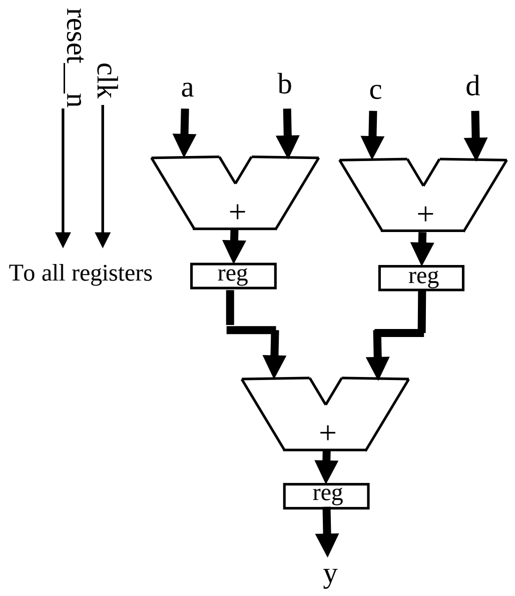
BØLGEFORM / WAVEFORM:



d)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity problem_2d is
  Port ( clk        :  in  std_logic;
         reset_n    :  in  std_logic;
         a, b, c, d :  in  std_logic_vector(15 downto 0);
         y          : out  std_logic_vector(15 downto 0));
end problem_2d;
architecture rtl of problem_2d is
  signal ab_sum_nxt, ab_sum_r    : unsigned(15 downto 0);
  signal cd_sum_nxt, cd_sum_r    : unsigned(15 downto 0);
  signal abcd_sum_nxt, abcd_sum_r: unsigned(15 downto 0);
begin
  process(clk, reset_n)
  begin
    if(reset_n = '0') then
      ab_sum_r   <= (others => '0');
      cd_sum_r   <= (others => '0');
      abcd_sum_r <= (others => '0');
    elsif(clk'event and clk='1') then
      ab_sum_r   <= ab_sum_nxt;
      cd_sum_r   <= cd_sum_nxt;
      abcd_sum_r <= abcd_sum_nxt;
    end if;
  end process;
  ab_sum_nxt   <= unsigned(a) + unsigned(b);
  cd_sum_nxt   <= unsigned(c) + unsigned(d);
  abcd_sum_nxt <= (ab_sum_r) + (cd_sum_r);
  y            <= std_logic_vector(abcd_sum_r);
end rtl;
```

BLOKKDIAGRAM / BLOCK DIAGRAM:

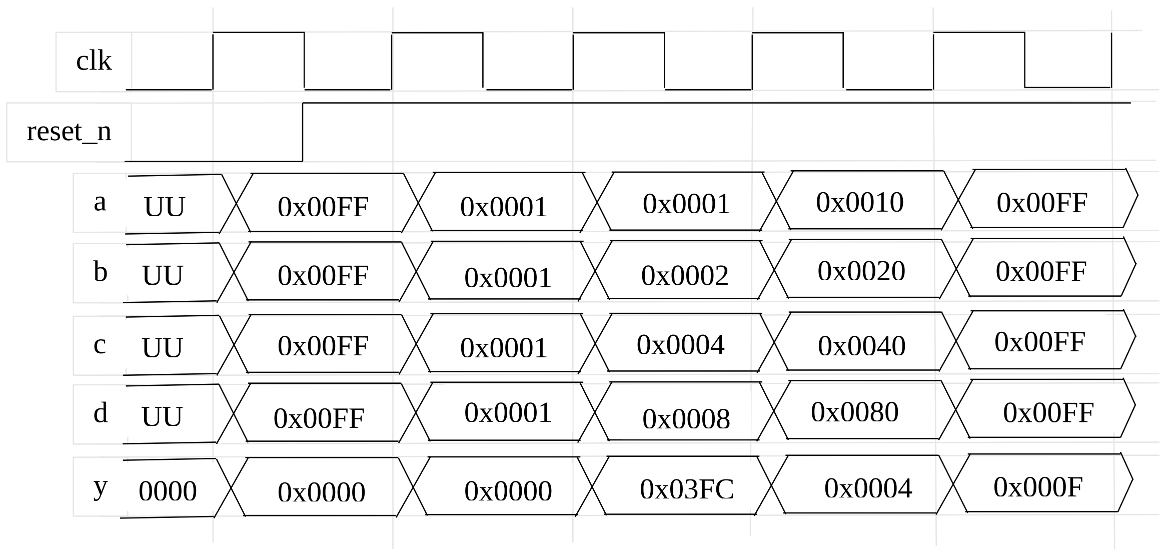


FUNKSJON / FUNCTIONALITY:

Pipelined adder tree.

ANTALL VIPPER INFERERT VED SYNTESE / NUMBER OF FLIP-FLOPS INFERRED DURING SYNTHESIS:
48

BØLGEFORM / WAVEFORM:



Problem 3: Optimalisering av kode (16 poeng)
Problem 3: Code optimization (16 points)

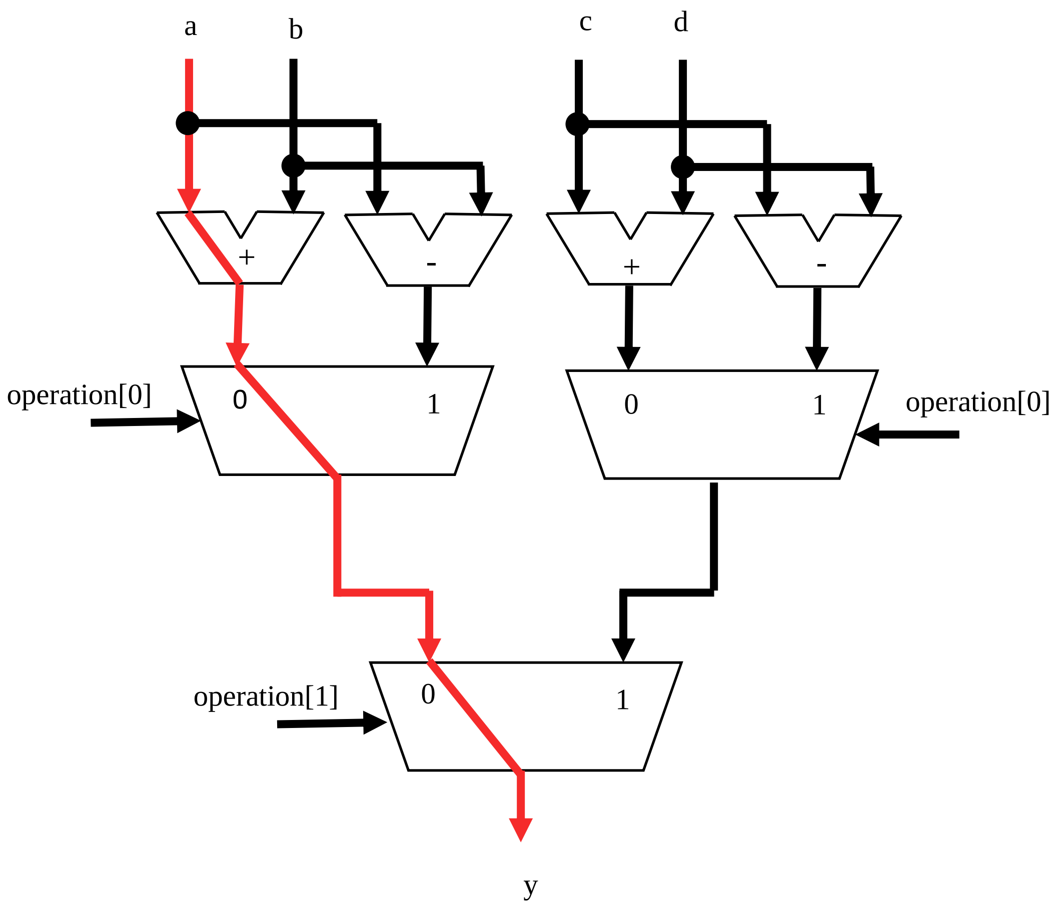
a)

Tegn et blokkdiagram som representerer koden nedenfor. Bruk følgende primitiv i tegningen din: “2x1 mux”, “add” og “sub”. Uthev kritisk (lengste) sti i kretsen.

Draw a block diagram representing the code below. Use the following primitives in your drawing: “2x1 mux”, “add” and “sub”. Highlight the critical path (i.e. the longest path).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity problem_3a is
  Port ( a, b, c, d :  in  std_logic_vector(15 downto 0);
         operation  :  in  std_logic_vector( 1 downto 0);
         y          : out  std_logic_vector(15 downto 0)
       );
end problem_3a;
architecture rtl of problem_3a is
  constant A_PLUS_B  : std_logic_vector(1 downto 0):= "00";
  constant A_MINUS_B : std_logic_vector(1 downto 0):= "01";
  constant C_PLUS_D  : std_logic_vector(1 downto 0):= "10";
  constant C_MINUS_D : std_logic_vector(1 downto 0):= "11";
  signal   result    : signed(15 downto 0);
begin
  process(a,b,c,d,operation)
  begin
    case(operation) is
      when A_PLUS_B  =>
        result <= signed(a) + signed(b);
      when A_MINUS_B =>
        result <= signed(a) - signed(b);
      when C_PLUS_D  =>
        result <= signed(c) + signed(d);
      when others    =>
        result <= signed(c) - signed(d);
    end case;
  end process;
  y <= std_logic_vector(result);
end rtl;
```

BLOKKDIAGRAM (bruk symbol for "2x1 mux", "add" og "sub"). Marker kritisk sti.
BLOCK DIAGRAM (use symbols for "2x1 mux", "add" and "sub"). Highlight the critical path.



One of the longest paths is marked with red color.

b)

Optimaliser RTL koden til **problem_3a** videre slik at det er garantert at kun en adderer blir inferert ved syntese. (Hint: Menteinngangen til addereren kan brukes til å legge til 1. Det kan være nyttig om man trenger å finne 2's komplement av et tall).

Optimize the RTL code of **problem_3a** further in order to ensure that only one adder is inferred during synthesis. (Tip: The carry-in signal of the adder can be used for adding 1. This can be useful if one needs to compute 2's complement of a number).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity problem_3b is
  Port ( a, b, c, d :  in  std_logic_vector(15 downto 0);
         operation  :  in  std_logic_vector( 1 downto 0);
         y          : out  std_logic_vector(15 downto 0)
       );
end problem_3b;
architecture rtl of problem_3b is
  constant A_ADD_B : std_logic_vector(1 downto 0):= "00";
  constant A_SUB_B : std_logic_vector(1 downto 0):= "01";
  constant C_ADD_D : std_logic_vector(1 downto 0):= "10";
  constant C_SUB_D : std_logic_vector(1 downto 0):= "11";
  signal   result  : unsigned(16 downto 0);
  signal   ac      : std_logic_vector(15 downto 0);
  signal   bd      : std_logic_vector(15 downto 0);
  signal   bd_inv  : std_logic_vector(15 downto 0);
begin

 ac     <= a when operation(1)='0' else c;
 bd     <= b when operation(1)='0' else d;
 bd_inv <= bd xor (bd'range => operation(0));
 -- Alternative: bd_inv <= bd when operation(0)='0' else not bd;

 -- operation(0)='1' when we subtract. Subtraction is achieved by computing the
 -- 2's complement of bd. 2's complement is calculated by inverting all bits and
 -- then add 1.

 -- It is possible to implement the addition of 1 efficiently
 -- by using the carry input of an adder.
 -- We can model this in VHDL by using the following formula:
 -- y = a + b + 1,
 -- 2y = 2a +2b +2
 -- 2y = 2a +1 + 2b +1
 -- 2y = (2a + 1) + (2b + 1)
 -- y = ((2a + 1) + (2b + 1))/2
 -- y = ((a<<1 + 1) + (b<<1 + 1)) >> 1
 -- In other words, extend the operands with one bit and insert the
 -- carry bit to the LSB of both operands. Add the operands and remove the
 -- LSB in the sum (which will always be 0).
 --
 -- Above we have also used the fact that multiplication by 2 is equivalent to
 -- shifting the bits one position left and that division by 2 is equivalent to
 -- shifting one bit to the right.
 -- Shifting is free (as in "requires no logic") in HW.
```

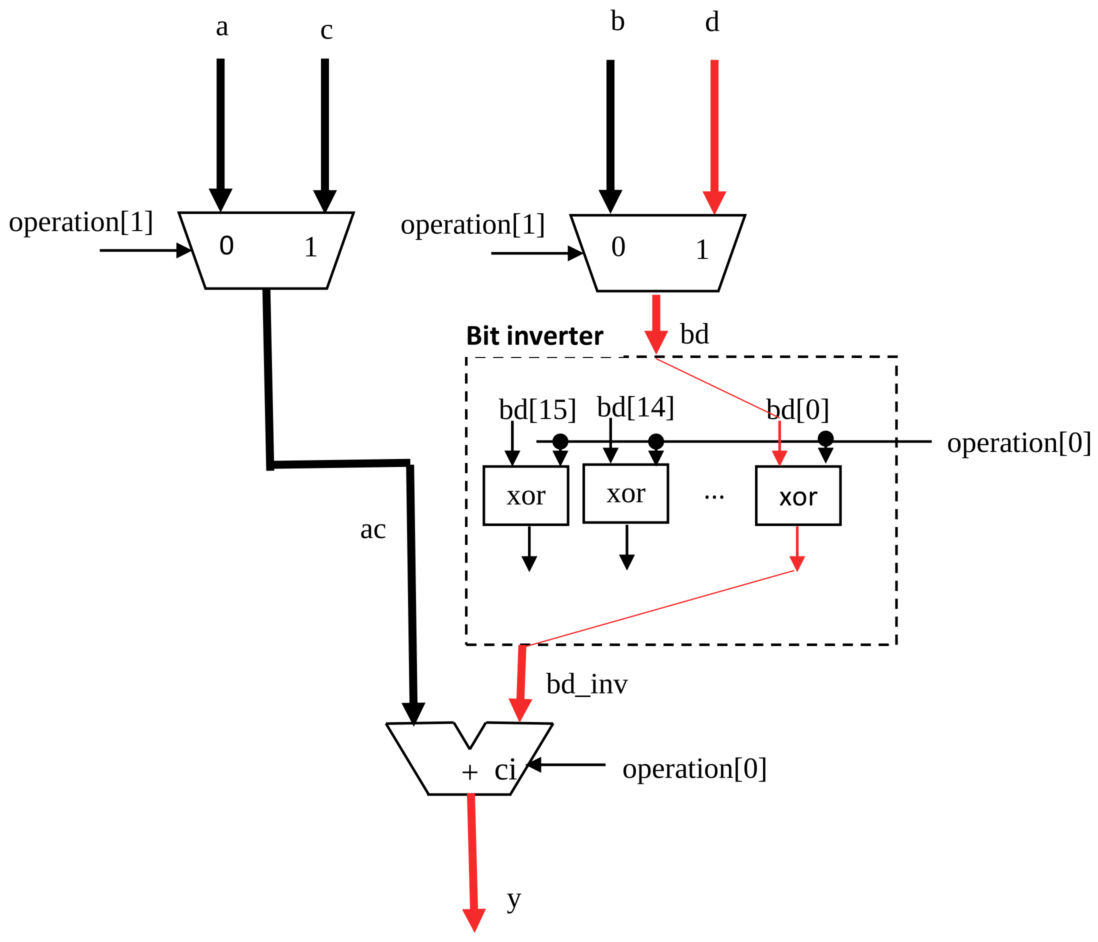
```
  -- It's just a matter of routing the wires correctly.
  result <= unsigned(ac & operation(0)) + unsigned(bd & operation(0));
  y <= std_logic_vector(result(16 downto 1));
end rtl;
```

c)

Tegn et blokkdiagram som representerer koden fra **problem_3b**. Alle symbol er tillatt. Marker kritisk sti.

Draw a block diagram representing the code of **problem_3b**. All symbols are allowed. Highlight the critical path.

BLOKKDIAGRAM for problem_3b (alle symbol er tillatt). Uthev kritisk sti.
BLOCK DIAGRAM for problem_3b (use any symbols). Highlight the critical path.



Longest path is marked with red.
One way to invert bits is to xor the bits with ‘1’. This is done in the bit inverter logic above.

d)
Hvilket design, **problem_3a** eller **problem_3b**, vil være raskest? Forklar hvorfor.
Which of the designs in **problem_3a** or **problem_3b** will be fastest? Explain why.

Critical path through problem_3a: 1 adder and two 2x1mux'es
Critical path through problem_3b: 1 adder, one 2x1 mux and one xor-gate.

A xor-gate has the same delay as a mux, the designs are therefore likely to have comparable speed.

Problem 4: Identifiser «latch»-ene (12 poeng)
Problem 4: Identify the latches (12 points)

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity problem_4 is
  Port ( a, b, c, d :  in  std_logic_vector(15 downto 0);
         operation  :  in  std_logic_vector( 1 downto 0);
         z          : out  std_logic_vector(15 downto 0);
         y          : out  std_logic_vector(15 downto 0)
       );
end problem_4;
architecture rtl of problem_4 is
  constant A_ADD_B : std_logic_vector(1 downto 0):= "00";
  constant A_SUB_B : std_logic_vector(1 downto 0):= "01";
  constant C_ADD_D : std_logic_vector(1 downto 0):= "10";
  constant C_SUB_D : std_logic_vector(1 downto 0):= "11";
  signal   result1    : signed(15 downto 0);
  signal   result2    : signed(15 downto 0);
begin
  process(a,b,c,d, operation)
  begin
    case(operation) is
      when A_ADD_B =>
        result1 <= signed(a) + signed(b);
      when A_SUB_B =>
        result2 <= signed(a) - signed(b);
      when C_ADD_D =>
        result1 <= signed(c) + signed(d);
      when others  =>  -- C_SUB_D
        result2 <= signed(c) - signed(d);
    end case;
  end process;
  y <= std_logic_vector(result1);
  z <= std_logic_vector(result2);
end rtl;
```

a)

Koden i **problem_4** vil gi “latch”-er dersom RTL koden syntetiseres. Hvorfor er det et problem?

The code in **problem_4** will infer latches. Why is that a problem?

In the code above we have tried to implement some combinatinal logic. This logic is supposed to implement a Boolean function where the outputs are a function of the current inputs and no old values.

The code above will however result in the inference of latches. These are sequential elements that will store old values and will result our circuit to produce a result that is dependent on the current inputs as well as on some old state from these unintended latches.

The value stored in the latch may often be random and the resulting circuit will therefore not only produce a wrong result. It can also be impossible to predict what this wrong value will be. It will often be a timing dependent random wrong value.

b)

Identifiser “latch”-ene i koden til **problem_4** og forklar hva som er galt med koden.

Identify the latches in the code of **problem_4** and explain what the problem with the code is.

We will get one latch per bit in result1 and result2. In total 32 latches. The latches will be inferred because we have not specified what the value for result1 and result2 will be for all input combinations. Here are some missing conditions:

- No value specified for result1 when operation=A_SUB_B or operation=C_SUB_D
- No value specified for result2 when operation=A_ADD_B or operation=C_ADD_D

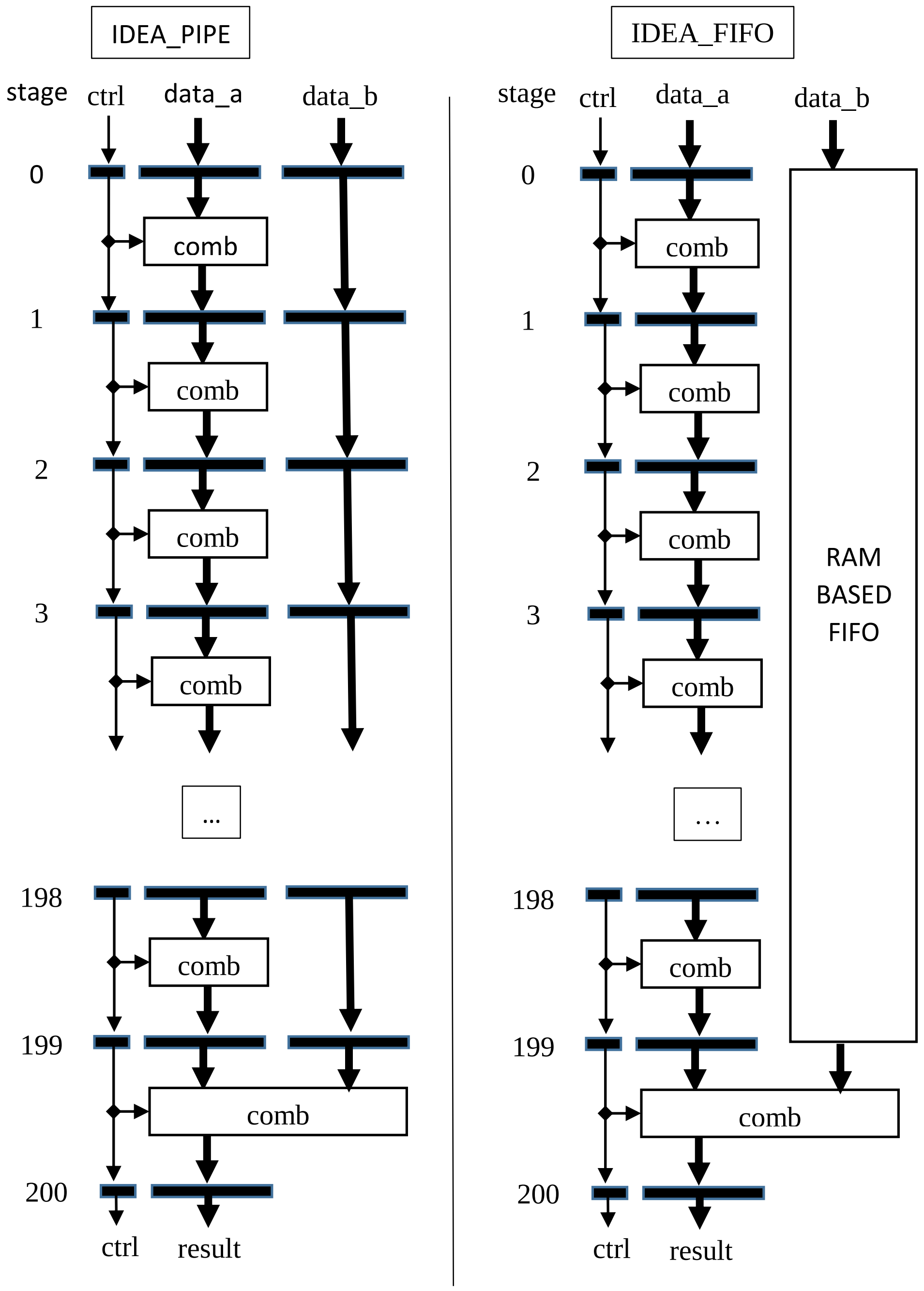
c)
Foreslå en måte å få bort "latch"-ene i koden.

Suggest a way to get rid of the latches in the code.

Make sure all outputs of the process are given a value for all input combinations. This can be done for example as indicated below.

```
process(a,b,c,d, operation)
begin
  case(operation) is
    when A_ADD_B =>
      result1 <= signed(a) + signed(b);
      result2 <= (others => ‘0’);
    when A_SUB_B =>
      result1 <= (others => ‘0’);
      result2 <= signed(a) - signed(b);
    when C_ADD_D =>
      result1 <= signed(c) + signed(d);
      result2 <= (others => ‘0’);
    when others  =>  -- C_SUB_D
      result1 <= (others => ‘0’);
      result2 <= signed(c) - signed(d);
  end case;
end process;
```

Problem 5: Laveffekt design teknikker (10 poeng)
Problem 5: Low power design (10 points)



Ken er i ferd med å lage et «pipelined» design. Inn til kretsen sendes et kontrollsignal (**ctrl**) og et par brede 512-bit signal med data (**data_a** og **data_b**). For hvert steg i Kens «pipeline» gjøres noe prosessering på **data_a**, mens **data_b** blir først brukt helt på slutten Kens «pipeline».

Ken har to ideer for hvordan han skal lage denne kretsen:

- **Idea_PIPE**: «Pipe» **data_b** sammen med **data_a.** (Til venstre i blokk diagrammet).
- **Idea_FIFO**: Skriv **data_b** til en RAM basert FIFO i starten på «pipelin-en» og les data når de trengs i bunnen på «pipelin-en». (Til høyre i blokkdiagrammet).

Ken is in the process of designing a very long pipe line. The input to the pipeline is some control bits (**ctrl**) and a couple of large 512-bit chunks of data (**data_a** and **data_b**). For every pipeline stage, some processing is done on **data_a**, but **data_b** is first needed at the very end of the pipeline.

Ken has two ideas for how to design this pipeline:

- **Idea_PIPE**: Pipe **data_b** along with **data_a.** (Illustrated at the left side of the diagram).
- **Idea_FIFO**: Push **data_b** to a RAM based FIFO at the beginning of the pipeline and pop the data when it is needed at the bottom of the pipeline. (Illustrated at the right side in the diagram).

a)

Diskuter hvilken av disse to ideene som vil være mest energieffektiv. Gjør rede for eventuelle antagelser du finner det nødvendig å gjøre.

Discuss which of the two ideas that are likely to be most energy efficient. Describe assumptions you deem necessary.

Dynamic power consumption can be reduced by reducing the activity or toggling in the design. Propagating a wide 512-bit chunk of data down through 200 pipeline steps is a very bad idea if we want to avoid unnecessary toggling.

In a pipeline all the data is moved every clock cycle. In a FIFO only the data that is written or read to the FIFO contributes to toggling. It is therefore very likely that a RAM based FIFO will be a more energy efficient solution than the long pipeline.

The **Idea_FIFO** is likely to be more energy efficient.

b)

Ken bestemmer seg for å implementere løsningen du anbefaler i **a)**. Er der andre ting Ken kan gjøre for å gjøre løsningen ytterligere energieffektiv?

Ken decides to go with your advice from **a)**. Are there any additional things Ken can do in order to reduce the energy consumption of his solution further?

Key techniques for reduction of power consumption:

- Use clock gating for each pipeline step so that the pipeline register is not updated unless the data about to enter the pipeline stage is actually valid.
- Turn off the clock (clock_gating) to the whole module if no valid data has been sent to the module.
- Turn off the power of the module (power gating) if it is not used at all for a longer period of time. This will reduce both dynamic and static power consumption to 0 as long as the power is off.

Additional techniques for reducing power consumption that might be applicable:

- If the critical path is much shorter than what is needed for the current clock frequency, then reducing dynamic power consumption through reducing the voltage (voltage scaling) might be an option.
- Try to reduce the number of pipeline stages if possible. That will reduce the latency in the design and might help software running on the design complete earlier if this software process is latency sensitive. If the process completes earlier, it might enter a low power state earlier. Fewer pipeline stages will result in fewer toggling registers.
- If a wide signal in a combinatorial block has a fan-out where only one of the paths are taken, then it can pay off to try to use AND gates on the inactive fan-out-paths in order to try to reduce the toggling on this inactive path.
- Gray coding of counters will ensure that the toggling of bits in counters are kept low.
- Compression and decompression of data will cost energy, but so does transporting large chunks of data around. Additional investigations are needed in order to decide if that is worth attempting.

Problem 6: Production testing (12 points)

a)

Hva er formålet med produksjonstesting?

What is the purpose of production testing / manufacturing testing?

There is always a chance that some chips are not working due to defects introduced during the manufacturing process. The purpose of production testing is to test the manufactured chips and sort the working chips-from the non-working chips.

This production testing process must be robust enough so that it reveals both defects that manifest themselves at slow speed, at the rated speed of the design, immediately after production or that would manifest itself after some time of use.

b)

Forklar hva en feilmodell er og gi ett eksempel på en feilmodell.

Explain what a fault model is and give an example of such a fault model.

During the production process many things can go wrong. It could be a spec of dust in the wrong place causing a short circuit or a open wire. (Typical defect types: Gate oxide shorts, electromigration, shorting defects, open defects).

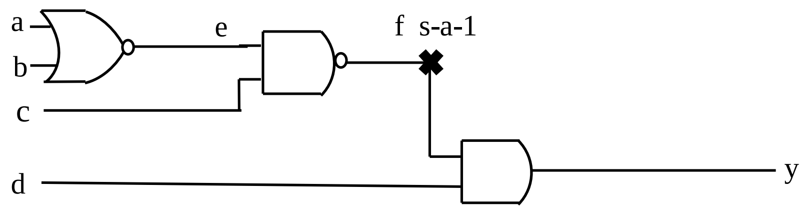
A fault model seek to model these physical defects. One example of such a model is the single stuck at fault model.

Many physical defects will yield the same behavior as if a single node in the design was wired permanently to ‘1’ or ‘0’. This is exactly how faults are modeled using the single-stuck-at fault model.

c)

Finn en test for node f stuck-at-1 (f s-a-1). Forklar fremgangsmåten.

Find a test for node *f* stuck-at-1 (*f* s-a-1). Explain your reasoning.



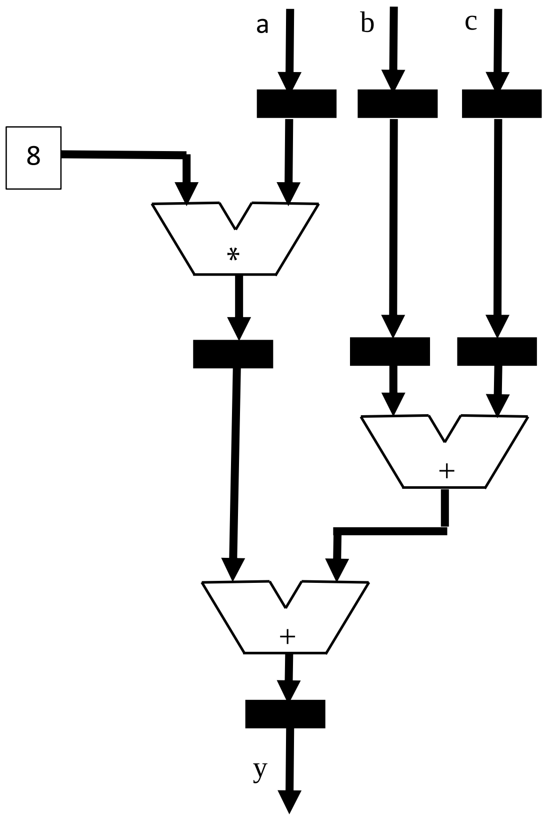
In order to detect f s-a-1 we need to control node f to the opposite value, namely 0.
f=0 => e=1 and c=1
e=1 => a=0 and b=0

In order to observe the fault effect on node y we need to set the input to the AND gate to the non-controlling value d=1.

Our test vector for f s-a-1:
(a,b,c,d)=(0,0,1,1)

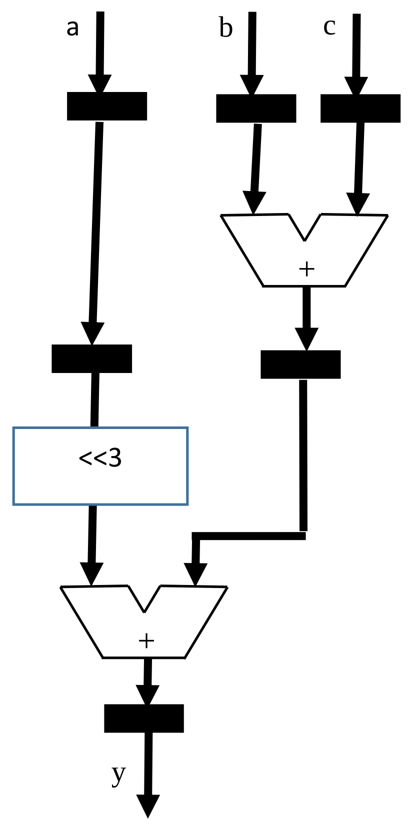
Problem 7 Øk klokkefrekvensen og reduser arealet (5 poeng)

Problem 7 Increase the clock frequency and reduce the area (5 points)



Forklar hvordan man kan endre kretsen over slik at det er mulig å samtidig doble klokkefrekvensen og redusere arealet.

Explain how it is possible to modify the design above so that the clock frequency is doubled and the area is reduced.



- Move the adder up one stage. This will remove several flip-flops.
- Realize that multiplying with 8 is 100% free in HW. It is just a left-shift by 3 operation, and that is just a matter of appending '000' to the LSBs of a. This wiring/shifting has been moved one pipeline register down so that it is obvious that we are not spending any flip-flops for the 3 static LSB's.

- The critical path has now been cut in half and we can therefore double the clock frequency.

Problem 8 RSA kryptering (20 poeng)
Problem 8 RSA encryption (20 points)

I semesteroppgaven så lagde vi en RSA krypteringskrets. Denne RSA krypterings regner ut:

$$C = M^e \bmod n, M < n.$$

In the term project, we have created an RSA encryption circuit. This RSA encryption circuit is able to compute:

$$C = M^e \bmod n, M < n.$$

a)
Tegn et blokkdiagram for dataveien i RSA kretsen din. Dette blokkdiagrammet bør vise alle registre, mux-er, adderere og skift-kretser i dataveien i designet ditt. Du trenger ikke ta med dataveien assosiert med serialisering av inn og ut data. Du trenger heller ikke ta med kontroll modulen. (Bruk tilleggsark om du trenger.)

Draw a block diagram for the *datapath* of your design. This block diagram should show all registers, mux'es, adders and shifters in the datapath of your design. You can omit the part of the datapath associated with serializing inputs and outputs. You can also omit everything related to the control. (Use additional sheets of paper if needed).

b)

Bruk blokkdiagrammet fra **a)** og forklar hvor mange klokkesykler ditt design trenger for å kryptere/dekryptere en 128 bit melding med en 128 bit nøkkel.

Use the block diagram from **a)** and explain how many clock cycles your design will need in order to encrypt/decrypt a 128 bit message using a 128 bit key.