Sensor guidelines for IN3160/4160: 2023

Rev 04

General principles

The exam is meant to show

- To what degree a candidate meets the minimum requirements stated in the "learning outcome" section of the course pages:
    - IN3160: https://www.uio.no/studier/emner/matnat/ifi/IN3160/index-eng.html#learning_outcomes
    - IN4160: https://www.uio.no/studier/emner/matnat/ifi/IN4160/index-eng.html#learning_outcomes

    IN3160: *"*
    Learning outcome

    After completion of the course you´ll:

    - *understand important principles for design and testing of digital systems*
    - *understand the relationship between behavior and different construction criteria*
    - *be able to describe advanced digital systems at different levels of detail*
    - *be able to perform simulation and synthesis of digital systems.*

    "

    IN4160 has an added last sentence compared to IN3160: *"*

    - *be able to perform advanced implementation and analysis techniques*

    "

    - The learning outcome is considered the *minimum requirement to pass* a course.
- How each candidate knowledge and skills related to the course content relates to other candidates taking or having taken the course earlier.

A sensor guideline cannot capture all possibilities for an exam answer, thus **a sensor must also decide based on the general principles**, rather than blindly using the detailed list provided in this document.

The bullet point scores must be seen as guidelines, not absolutes. The bullet text does not always cover all possible situations, *for example* if there is no DUT-input generated in data testing, *full score cannot be given for that bullet point,* even if the subpoints for assertions and data output is well covered.

The guidelines will be published for the course students along with the exam results, so they can also be used for reference when writing justification of the score given at a task in the exam.

Autocorrected part

The part being autocorrected should normally be corrected by the exam tool (inspera)

1: Design Flow (2)

Specification - VHDL design entry - RTL-simulation – Synthesis- Gate level simulation – Place and route – Static timing analysis – Device programming

2: Digital design Knowledge (30)

3: Diagrams (6)

- Hint:
    - Draw you own timing diagram or sequence of both named and unnamed signals in the block diagram.
        - Assume start is active at all times
        1. **+req+r1req+synchreq…**
        2. **+ack+r1ack+synchack…**
        3. **-req-r1req-synchreq…**
        4. **-ack-r1ack-synchack…**
        5. **=>s_idle**+req+r1req+synchreq… ->2

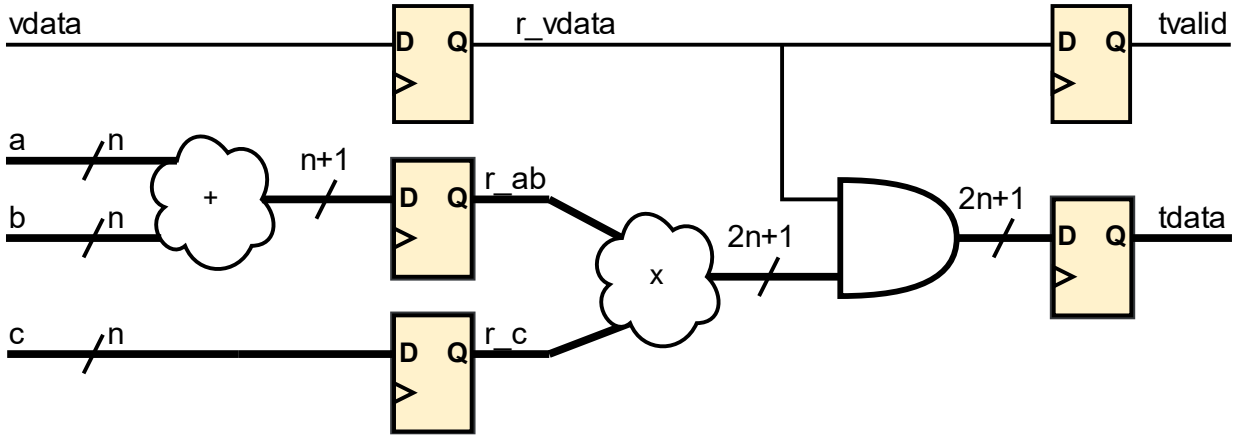
4: ASMD and datapath diagram (2)

- Hint: The clue here is noting how registers work in ASMD
    - That is we select
        - the a+b sum for the register once in the add state
        - the sum*c when in the multiply state
    - The three states requires at least two registers
    - Only y and sum has registers

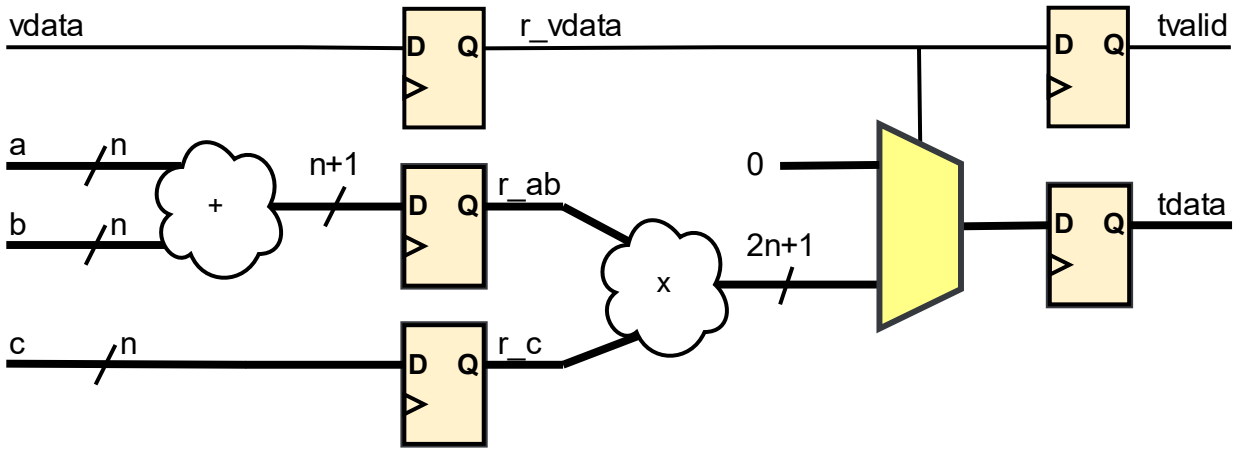
Manually corrected part:

5: Datapath diagram:

Suggested solution:



Alternate solution, using multiplexers:



5: Datapath diagram score (10)

- **all relevant signals present /1** (a, b, c, vdata, tvalid, tdata)
  - *All relevant, none irrelevant*
- **Readability: /2**
  - direction: (1)
    - consistent
    - from left to right or top to bottom
  - math operations visible as combinational logic: (1)
    - not looking like a register / clocked module
    - Identifiable as a math operation
- **Pipeline /5**
  - Registers (2)
    - Stage 1 (r_vdata, r_ab, r_c) : (1)
    - Stage 2 (tvalid, tdata): (1)
  - Control and data signals have equal number of pipeline stages (1)
  - Math operations (2)
    - performed in different stages (1)
    - Addition is performed first stage and multiplication in last (1)
- **Vector sizes in chart /1**
  - 10 (n) abc,
  - 11 (n+1) r_ab
  - 21 (2n+1) tdata
- **Result selector /1**
  - Implemented so data is zero when not valid (0,5)
  - Implemented before the last registers (0,5)

6: Pipeline implementation

Suggested solution:

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

entity pipelined is
 generic (
    N: integer := 10
  );
 port (
  clk, reset : in  std_logic;
  a,b,c      : in  std_logic_vector(N-1 downto 0);
  vdata      : in  std_logic;
  tvalid     : out std_logic;
  tdata      : out std_logic_vector(N*2 downto 0)
  );
end entity pipelined;

architecture RTL of pipelined is
  signal next_ab,r_ab  : signed(N downto 0);
  signal next_c, r_c   : signed(N-1 downto 0);
  signal next_tdata    : std_logic_vector(N*2 downto 0);
  signal r_vdata       : std_logic;

begin
  REGISTER_ASSIGNMENT: process(clk) is
  begin
    if rising_edge(clk) then
      if reset then
        r_ab    <= (others => '0');
        r_c     <= (others => '0');
        tdata   <= (others => '0');
        r_vdata <= '0';
        tvalid  <= '0';
      else
        r_ab    <= next_ab;
        r_c     <= next_c;
        tdata   <= next_tdata;
        r_vdata <= vdata;
        tvalid  <= r_vdata;
      end if;
    end if;
  end process;

  next_ab  <= signed(a(N-1) & a) + signed(b(N-1) & b);
  next_c   <= signed(c);
  next_tdata <=
    std_logic_vector((r_ab*r_c)) when r_vdata else
    (others => '0');

end architecture RTL;
```

6: Pipeline implementation score (15)

- **Library usage: /0.5**
  correct and consistent with use in the architecture
    - IEEE std_logic_1164 - when std_logic is used
    - IEEE numeric_std - when (un)signed arithmetics is used-
    - etc
- **Readability /1.5** -- proportional with the amount of meaningful VHDL
    - Sensible, consequent naming, ex:
        - R_ for registers that are not in the entity
        - _next for register input
    - Indentation
        - Normally Block layout, consequent.
    - Punctuation, parenthesis, and use of capital letters
        - Consequent/ in a readable manner
    - Etc.
- **Entity /2**
    - generic declaration: (1)
    - Port declaration (1)
        - Correct signals present (a,b,c,vdata, tdata, tvalid, clk, (reset)), none irrelevant
        - Correct direction and type
        - Corresponding sizes ,
        - using generics
- **Architecture: /11**
    - Type conversions (4)
        - using signed type conversion (unsigned is 0)
        - intermediate calculation size correct (ie add 1 bit for both a and b)
        - correct concatenation : (ie top bit inserted, not bottom)
        - extending with sign-bit : (ie extending with '0' or '1' is 0 point)
    - Syntax (0.5)
        - Generally correct syntax
            - Ex: correct use of assignment operator
            - Etc.
    - Declarations in the declarative region (0.5)
    - Descriptions RTL or SYNTH (0.5)
    - Correct use of sensitivity list (0.5)
        - (all), or all signals used listed for CL)
        - (clk) for synch reset register update, (clk, reset)
    - Signals for all internal registers (0.5)
    - Signals for internal register inputs (0.5)
        - at least those used in CL computation (next_ab, next_tdata)
    - Register assignment (and reset if any) in a single process (1)
    - CL as concurrent statements or in one or more processes (1)
    - Correct implementation of pipeline (2)
        - Implements a two stage pipeline
        - Selects result or 0 based on pipelined vdata (r_vdata or similar)

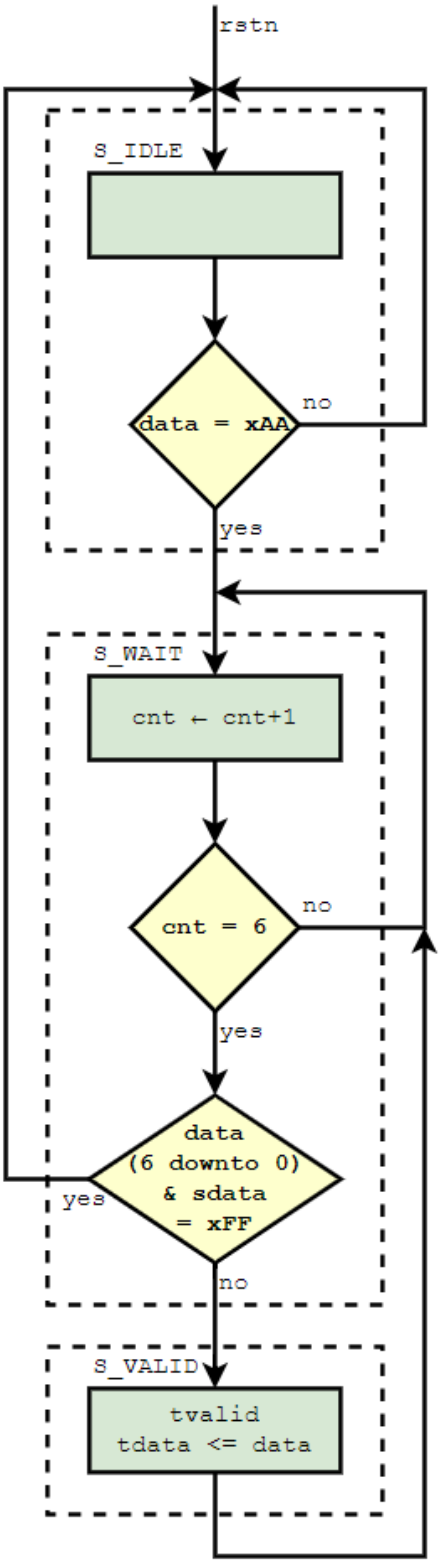
cocotb testbench for pipelined *(not required for this task)*

```
import cocotb
from cocotb import start_soon
from cocotb.clock import Clock
from cocotb.triggers import ClockCycles, RisingEdge,FallingEdge, Timer, ReadOnly

async def setabc(dut, aa, bb, cc, vdata):
    await FallingEdge(dut.clk)
    dut.a.value = aa
    dut.b.value = bb
    dut.c.value = cc
    dut.vdata.value = vdata
    await RisingEdge(dut.clk)

async def valid_check(dut):
    dut._log.info("valid-check started...")
    await ClockCycles(dut.clk, 2)
    while 1:
        await RisingEdge(dut.clk)
        await ReadOnly()
        if (dut.tvalid.value == 0):
          assert dut.tdata == 0, "Data present when tvalid is 0"

async def calc_check(dut):
    dut._log.info("Results-check started...")
    RAB = 0
    RC  = 0
    RES = 0
    await ClockCycles(dut.clk, 2)
    while 1:
        await RisingEdge(dut.clk)
        await ReadOnly()
        # Correct order is necessary to mimic shiftreg
        RES = RAB*RC
        RAB = int(dut.a.value.signed_integer) +int(dut.b.value.signed_integer)
        RC  = int(dut.c.value.signed_integer)
        if(dut.tvalid.value == 1):
            assert dut.tdata.value.signed_integer == RES, "Calculation incorrect"

async def stimuli_generator(dut):
    ''' Generates all data for this tesbench'''
    dut._log.info("Generating stimuli...")
    PERIOD_NS = 5
    start_soon(Clock(dut.clk, PERIOD_NS, 'ns').start())
    await setabc(dut, 0,0,0,0)
    dut.reset.value = 1
    await Timer(12, units= 'ns')
    dut.reset.value = 0
    await Timer(10, units= 'ns')
    await setabc(dut, 5,6,3,0)
    await setabc(dut, 6,6,2,1)
    await setabc(dut, 6,6,2,0)
    await setabc(dut, 7,8,9,1)
    await setabc(dut, -1,-2,-3,1)
    await setabc(dut, -2,3,4,1)
    await setabc(dut, 2,3,-4,1)
    await setabc(dut, 2,-3,4,1)
    await setabc(dut, 2,-3,4,0)
    await Timer(30, units= 'ns')
    dut._log.info("Stimuli ended...")

@cocotb.test()
async def main_test(dut):
    """ Starts comparator and stimuli """
    dut._log.info("Running tests...")
    start_soon(valid_check(dut))
    start_soon(calc_check(dut))
    await start_soon(stimuli_generator(dut))
    dut._log.info("... all tests done!")
```

7: ASM diagram

Suggested solution:

```
Defaults:

data ← data(6 downto 0) & sdata
tdata <= (others => '0')
tvalid <= '0'
cnt ← 0
```



7 ASM diagram score (10)

- **Diagram syntax /3**
  - All ASM blocks correct marked (0.5)
  - Box entry from above (0.5)
  - Valid exits (No double exit) (0.5
  - Square state boxes (0.5)
  - Assignments (0.5)
    - '←' for register assignments
    - <= used for non-register assignments
  - Decisions in proper decision boxes (0.5)
- **Implementation /7**
  - Reset enters idle/ correct start box (1)
  - Moore output only (1)
  - Correct number of states (idle, wait for byte, data_valid) (1)
  - All output values defined for all states (1)
    - *Default values stated unless obvious*
      - *(Boolean) Flag use is considered obvious if it is only true when used*
      - *Register assignment is always obvious*
        - *(values are kept unless re-assigned)*
    - *Using '<=' for vectors requires default clause or all state assignment.*
      - *Using <= once w/o declaring defaults reduces this score by .5.*
  - Correct decisions (2)
    - Correct counting
    - Check on data and sdata (1)
  - Transitions lands in correct state (1)
    - *A correct state has correct output and correct decisions.*

9 VHDL implementation (of FSM)

Solution suggestion:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
  port (
    clk       : in  std_logic;
    rstn      : in  std_logic;
    sdata     : in  std_logic;
    tdata     : out std_logic_vector(7 downto 0);
    tvalid    : out std_logic
    );
end fsm;

architecture rtl of fsm is

  type state_type is (S_IDLE, S_WAIT, S_VALID);
  signal present_state, next_state : state_type;

  signal cnt, cnt_next : integer range 0 to 8 := 0;
  signal data, next_data : std_logic_vector(7 downto 0);

begin
  -- The description starts here

  -- 1: present state assigment:
  sync : process(clk) is
  begin
    if rising_edge(clk) then
      if rstn = '0' then
        present_state <= S_IDLE;
        cnt  <= 0;
        data <= (others => '0');
      else
        present_state <= next_state;
        data <= next_data;
        cnt <= cnt_next;
      end if;
    end if;
  end process;

  -- 2: combinational next_state logic
  --next_state_CL : process(data, present_state, sdata, cnt) is
  next_state_CL : process(all) is
  begin
    case present_state is
      when S_IDLE =>
        next_state <=
          S_WAIT when data(7 downto 0) = x"AA" else
          S_IDLE;
      when S_WAIT =>
        next_state <=
          S_IDLE when cnt = 6 and data(6 downto 0) & sdata = x"FF" else
          S_VALID when cnt = 6 else
```

```
            S_WAIT;
        when S_VALID =>
          next_state <= S_WAIT;
        when others =>
          next_state <= S_IDLE;
      end case;
    end process next_state_CL;

    -- 3: combinatorial output logic
    output_CL : process(all) is
    begin
      --default output values
      tvalid  <= '0';
      tdata   <= (others => '0');
      cnt_next <= 0;

      case present_state is
        when S_IDLE =>
          null;
        when S_WAIT =>
          cnt_next <= cnt + 1;
        when S_VALID =>
          tvalid <= '1';
          tdata <= data;
        when others =>
          null; -- use default values
      end case;
    end process output_CL;

    -- shiftreg
    next_data <= data(6 downto 0) & sdata;

end architecture;
```

9: VHDL implementation (of FSM) score (13)

- **Library usage: /0.5**
  correct and consistent with use in the architecture
    - IEEE std_logic_1164 - when std_logic is used
    - IEEE numeric_std - when (un)signed arithmetics is used-
    - etc
- **Readability /1.5** -- proportional with the amount of meaningful VHDL
    - Sensible, consequent naming, ex:
        - R_ for registers that are not in the entity
        - _next for register input
    - Indentation
        - Normally Block layout, consequent.
    - Punctuation, parenthesis, and use of capital letters
        - Consequent/ in a readable manner
    - Etc.
- **Entity /2**
    - Port declaration
        - Correct signals present (rstn, clk, sdata, tdata, tvalid),
        - non irrelevant
        - Correct direction and type
        - Corresponding sizes,
    - Etc.
- **Architecture /7**
    - Code style description (RTL or ..?) (0.5)
    - Declarations in the declarative region (0.5)
    - Separate process for register update (1)
    - Separate process for state encoding (1)
    - Separate process (or statements) for output decoding (1)
    - Correct use of sensitivity in processes (1) Both need to be good
        - Clk for registers
        - All (or all listed) for CL
    - Syntax (1)
        - Understandable VHDL ()
    - Signals for all internal registers (0.5)
        - (cnt, data)
    - Signals for internal register inputs (0.5)
        - (next_cnt, next_data)
- **Implementation /2:**
    - Correct State assignment decisions (0.5)
    - Correct counting decisions (0.5)
    - Correct output (0.5) (based on state)
    - Correct shiftregister (0.5) (always on)

9 Test bench

Suggested solution:

```
-- libraries should be present
library IEEE;
use IEEE.std_logic_1164.all;

use std.env.finish;

-- work shall be present
use work.all;

-- entity shall be empty or have generic
entity tb_fsm is
end tb_fsm;

-- achitecture shall have meaninful name (not rtl, or structural)
architecture tb of tb_fsm is

  procedure write_sof(signal clk : in std_logic;
                      signal sdata : out std_logic) is
    variable sof : std_logic_vector(7 downto 0) := x"AA";
    begin
      for i in 0 to 7 loop
        wait until falling_edge(clk);
        sdata <= sof(7-i);
      end loop;
    end procedure;

  procedure write_eof(signal clk : in std_logic;
                      signal sdata : out std_logic) is
    begin
      for i in 0 to 7 loop
        wait until falling_edge(clk);
        sdata <= '1';
      end loop;
    end procedure;

  procedure write(signal clk : in std_logic;
                  signal sdata : out std_logic;
                  signal tdata : in std_logic_vector(7 downto 0);
                  signal tvalid : in std_logic;
                  data : in std_logic_vector(7 downto 0)) is
    begin
      for i in 0 to 7 loop
        wait until falling_edge(clk);
        sdata <= data(7-i);
      end loop;
      wait until rising_edge(tvalid);
      assert tdata = data report "assert violation";
    end procedure;

  signal clk    : std_logic := '0';
  signal rstn   : std_logic;
  signal sdata  : std_logic := '0';
  signal tdata  : std_logic_vector(7 downto 0);
```

```
  signal tvalid : std_logic;

  component fsm is
    port (
      clk    : in  std_logic;
      rstn   : in  std_logic;
      sdata  : in  std_logic;
      tdata  : out std_logic_vector(7 downto 0);
      tvalid : out std_logic);
  end component;


begin
  UUT : fsm port map (
    clk    => clk,
    rstn   => rstn,
    sdata  => sdata,
    tdata  => tdata,
    tvalid => tvalid);

  clk <= not clk after 10 ns;
  rstn <= '1';

  stim_proc : process
  begin
    report "Starting test";
    wait for 1 us;
    write_sof(clk,sdata);
    write(clk,sdata,tdata,tvalid,x"0F");
    write(clk,sdata,tdata,tvalid,x"F0");
    write(clk,sdata,tdata,tvalid,x"10");
    write(clk,sdata,tdata,tvalid,x"A0");
    write_eof(clk,sdata);
    wait for 5 us;
    finish;
  end process;
```

9: Test bench score (12)

- **Library usage: /0.5**
  correct and consistent with use in the architecture
    - IEEE std_logic_1164 - when std_logic is used
    - IEEE numeric_std - when (un)signed arithmetics is used-
    - etc
- **Readability /1.5** -- proportional with the amount of meaningful VHDL
    - Sensible, consequent naming, ex:
        - R_ for registers that are not in the entity
        - _next for register input
    - Indentation
        - Normally Block layout, consequent.
    - Punctuation, parenthesis, and use of capital letters
        - Consequent/ in a readable manner
    - Etc.
- **Empty Entity /1**
- **Architecture /9**
    - Syntax (0.5)
        - Understandable VHDL
    - Declarations in the declarative region (0.5)
    - Code style description (0.5)
        - *Behavioral or something descriptive as for simulation, not synthesis*
    - Correct use of sensitivity list (0.5)
    - Correct Component declaration (1)
    - Signal declarations for all signals into entity (1)
        - *Default values for component input only*
    - Correct component instantiation (1)
    - Correct Clock generation (process, continuous) (1)
    - Reasonable use of subroutines (procedures) (1)
    - Data testing (2)
        - using assertions (1)
        - all data (1)
            - *tvalid*
                - *0 when not appropriate*
                - *1 when appropriate*
            - *tdata*
                - *correct when tvalid*
                - *0 when not tvalid*