Ex1 Libraries and type conversion (10p)

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

entity calc is
  generic(size: integer := 16);
  port(
       A, B:    in std_logic_vector(size-1 downto 0);
       result:  out std_logic_vector(size downto 0)
       );
end entity calc;

architecture RTL of calc is
  signal i_result : signed(size downto 0);
begin
  i_result <= signed( ("0" & unsigned(A)) - ("0" & unsigned(B)));
  result <= std_logic_vector(i_result);
end architecture RTL;
```

sensor guide :

```
-- libraries
--  ieee                    1p
--   std_logic_1164.all : 1p
--   numeric_std.all :    1p
--
-- arkitektur
--   i_result declaration:
--     correct place       0,5p
--     signal              0,5p
--     signed              0,5p
--    correct size          0,5p      (size downto 0) eller (16 downto 0)
--  i_result  assignment
--     correct assignment operator <=          0,5p
--    signed conversion i_result'type (signed) 1p
--    concatenation of "0" for A og B:         1p
--    unsigned conversion for  A og B          1p
-- result assignment:
--    correct assignment operator <=              0,5p
--    conversion of i_result to std_logic_vector: 1p
-
```

improved architecture:

```
architecture RTL of calc is
  signal i_result : signed(size downto 0);
begin
  i_result <= ( signed("0" & a) - signed("0" & b));
  result <= std_logic_vector(i_result);
end architecture RTL;
```

The solution above is better, since the arithmetic subtraction is performed signed. The first solution actually subtracts unsigned, then converts to signed, which (should) lead to errors in case b > a. Both the original and the improved architecture should be accepted.

libraries /3

i_result declaration /2

i_result assignment /3,5

result assignment /1,5

Ex2 Process and subprogram (7p)

-- a)

```
architecture process_arch of subtract is
begin
  subtracting: process(all) is
    variable i_result: signed(size downto 0);
  begin
    i_result := signed( ("0" & unsigned(A)) - ("0" & unsigned(B)));
    result <= std_logic_vector(i_result);
  end process;
end architecture process_arch;
```

```
-----------------------------------------------------------------
-- guidelines: a)
-- correct process
--   sensitivity list (all) or (A,B)            1p
--   locale variables correctly declared        1p
--     correct place
--     correct type (variable)
--   correct assignment of result and i_result   1p
--      result shall be last in the process
--      correct assignment operator for both signal and variable
-----------------------------------------------------------------
```

-- b)

```
architecture subprog_arch of subtract is
  function sub( A, B : std_logic_vector) return signed is
      variable i_result: signed(A'left + 1 downto 0);
    begin
      i_result := signed( ("0" & unsigned(A)) - ("0" & unsigned(B)));
      return i_result;
    end function;
begin
  result <= std_logic_vector(sub(A,B));
end architecture subprog_arch;
```

```
-----------------------------------------------------------------------
-- guidelines b)
-- function chosen  (procedure is a poor choice here)  1p
-- Correct subprogram declaration
--    parameter usage                      1p
--      input vectors
--      return signed // use output type if procedure is used
--    correct declaration (variable) of i_result    1p
--      correct place
--      correct type
-- Riktig bruk av subprogram i arkitektur          1p
-----------------------------------------------------------------------
```

General remark for a) og b):

-- It is not intended to draw points for errors propagated from the previous assignment (ie errors in type conversions shall not be deducted points here)

a)

declaration /sensitivity list /1

local variable /1

assignment /1

b)

Subprogram choice /1

parameter usage /1

correct variable declaration /1

usage in architecture /1

CONSIDER: Is omitting local variable 2 point deduction in a)?

Since: we can not check assignment of result must be inside process.

Ex3 State diagrams (4p)

This exercise is auto corrected

It tests the ability to identify state diagrams and ASM diagrams.

Discern what the correct states are

Know how output is shown (moore and mealy)

Errors present in wrong choices:

- Mealy box presented as state
- Errors in input and output
- Wrong state transitions

Ex4 State tables (4p)

This exercise is auto corrected

It tests the ability to read state tables and compare with ASM diagram

Errors in state tables

- Wrong number of states (Mealy box as a state)
- Transition errors
- Output errors

Ex5 Multiple choice (20 points)

This exercise is autocorrected.

FSM = Finite state machine, ROM = Read Only Memory, RAM = Random access memory

A microcoded FSM can implement a Mealy Machine	T	1
A mealy machine does use registers for all outputs		2
A mealy machine does not use state registers		3
RAM is synchronous		4
ROM can be asynchronous	T	5
A microcoded FSM with a sequencer can be seen as a microcontroller or microprocessor.	T	6
All states are legal in a microcoded FSM		7
A look-up-table is digitally equivalent to a ROM or RAM device	T	8
A crossbar switch has at least two baffles		9
SRAM has interleaved memory		10
DRAM cells must be rewritten after read	T	1
Bus interfaces does not use control signals		2
Flow control is only used for clock domain crossing		3
Double buffering reduces delays		4
A FIFO does not use double buffering		5
A brute force synchronizer uses double buffering	T	6
Pipelining is typically implemented using RAM		7
Periodically valid signals is better than always valid signals		8
Divide and conquer is bad practice in digital design.		9
A brute force conqueror uses hack and slash for partitioning designs		20
A built-in self-test is synonymous with a self-testing testbench		1
A self-testing test bench can be used for verification	T	2
A behavioral model shall be synthesizable		3
A behavioral model shall not be synthesizable		4
Formal verification cannot be achieved		5
A shifter contains a shift register		6
Flip-flop inputs should switch during setup time to avoid metastability		7
It is generally bad practice to use brute force synchronizers for multiple bits	T	8
Sequential circuits should not be used as FSMs		9
Combinational logic does not contain flip-flops	T	30
VHDL means Verilog Hardware Description Service Language		31
Seven segment displays contain Linear Feedback shift registers		32
Synthesizeable VHDL is executed sequentially		33
Hazards occur in every digital design		34
Hazards occur when output changes more than once after input has changed	T	35
Moore Machines never create hazards		36
If we use “if” in VHDL we will avoid creating latches		37
If we use “if” in VHDL we will avoid creating hazards		38
The selected statement in VHDL is always the best option		39
Configurable logic blocks has four types of AD-converters		40

Ex6 Pipelining and type conversion (10p)

result = a+b+c computation and control signals shall be pipelined.

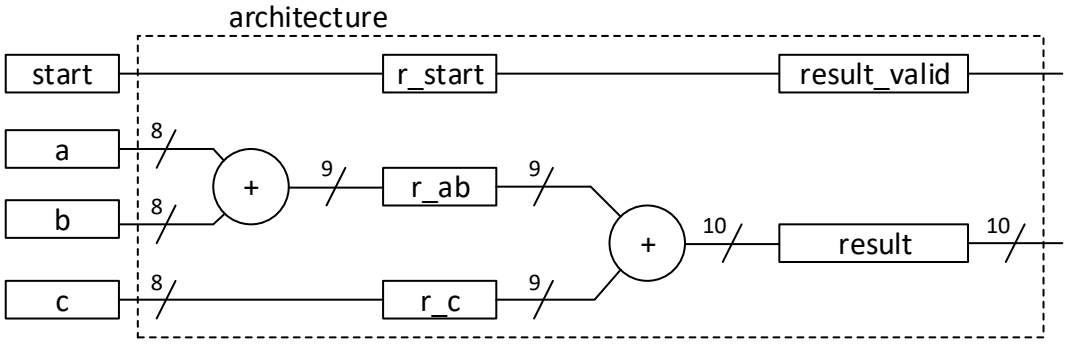
See solution for example.

4p for correct pipelining of a+b+c.

4p for correct pipelining of control signals.

2p for type conversion, reset and output registers.

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

entity pipelined is
 port (
  clk : in std_logic;
  rst : in std_logic;
  a   : in std_logic_vector(7 downto 0);
  b   : in std_logic_vector(7 downto 0);
  c   : in std_logic_vector(7 downto 0);
  result : out std_logic_vector(9 downto 0);
  start : in std_logic;
  result_valid : out std_logic
  );
end entity pipelined;

architecture rtl of pipelined is
  signal i : integer;
  signal start_tmp : std_logic;

begin

  process(clk)
    begin
    if rising_edge(clk) then
      if rst = '1' then
        result <= (others => '0');
        result_valid <= '0';
        i <= 0;
      else
        i <= to_integer(unsigned(a))+ to_integer(unsigned(b));
        result <=  std_logic_vector(
                     to_unsigned(i+to_integer( unsigned(c)),10));
        start_tmp <= start;
        result_valid <= start_tmp;
      end if;
    end if;
  end process;

end architecture rtl;
```



The figure above is not a part of the solution, it is just a visualisation of the datapath.

```
-- an alternate solution directly related to the datapath

architecture datapath of pipelined is
  signal r_start     : std_logic;
  signal cl_ab, r_ab : unsigned(8 downto 0);
  signal r_c         : unsigned(7 downto 0);
  signal cl_result   : unsigned(9 downto 0);
begin
  -- Combinational logic
  cl_ab <= (unsigned(a) + unsigned(b));
  cl_result  <=  r_ab + r_c;
  -- Register stage 1
  r_ab <=(others => '0') when rst else cl_ab       when rising_edge(clk);
  r_c  <=(others => '0') when rst else unsigned(c) when rising_edge(clk);
  r_start        <= '0'  when rst else start       when rising_edge(clk);
  -- Register stage 2
  result_valid   <= '0'     when rst else r_start                     when rising_edge(clk);
  result <= (others => '0') when rst else std_logic_vector(cl_result) when rising_edge(clk);
end architecture datapath;
```

Pipeline a+b+c /4

- ab_reg /1
- c_reg /1
- result_reg /1
- result_reg when start_reg /1

Pipeline control signals /4

- pipelined control /1
- 2 stage pipeline /1
- unconditional control pipeline /1
- no double drivers /1

Type conversion, reset and output registers /2

- all registers reset /0,5
- no CL reset /0,5
- size conversion /0,5
- type conversion /0,5

Ex7 ASM diagram (15p)

4p shows understanding of state box and decision box

3p signal assignment in state boxes, question in decision box

2p correct test of counter values (not one off)

1p reset arrow to first state

5p ASM in accordance with task

shows understanding of state box and decision box /4
ASM diagram rules
not testing the same twice

reset arrow to first state /1

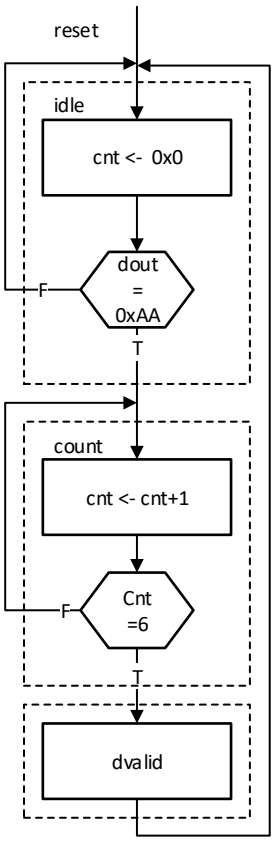
signal assignment in state boxes, question in decision box /3
generally /0,5
reset_count /0,5 counter <- (others=>'0')
run_count /0,5 counter <- counter +1;
dvalid /0,5
0xAA (decision) /0,5
counter_value (decsion) /0,5

correct test of counter values (not one off) /2
shall be from 0 to 7-1, we get the last bit entering the dvalid-state
(0 to 7, yields 1p, being a single period too long)

ASM in accordance with task /5
3 States:
wait until 0xAA /1
wait on counter /1
d_valid (=Moore) /1
2 decisions (AA and counter) /1
correct transitions /1

-> => = :=

Suggestion for ASM diagram



Note: always shift in so that *dout <- dout(7 downto 0) & sdata;*

Ex8 State machine implementation (15p)

Entity matches task description /1

in clk, rst, sdata,
out dout, dvalid

Layout /1

Architecture name /.5 (RTL or something sensible, not beh/sim..)
Indentation and general neatness /.5

Definiton of state type and curr_st/next_st-signals /2.

current state register with reset and sensitivity list /2

register assignment of signals (sig<=next_sig) /3

NOTE: CL by reg.state is allowed, CL "by self" not OK
output /1
output synch reset /.5
signals (counters/shiftreg) /1
signals synch reset /.5

nextstate/next_output logic process with sensitivity list /3

"all" is OK with CL, as is a list that makes sense.

in accordance with ASM diagram /3

next_state match /1
output match /1
signals match /1

Ex9 Testbench (15p)

Library references /1

Shall have empty entity /1

Arch shall not be rtl/structural. /1

Declaration of component /2

Signal declarations /1

Shall have port map/UUT instantiation. /2

Clock /1p

Shall have procedure/function for test vectors /2

Shall have stimuli process which:

- **Checks the test vectors:**
  0x0f, 0x4a, 0x6c (0xaa0f, 0xaa4a, 0xaa6c) /2
- **Shall verify the result, use of assert /1**
- **Shall report the results (e.g. use of report notice/failure). /1**