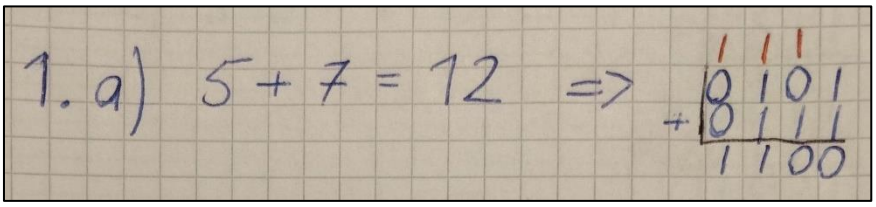
Lösningsförslag övningsuppgifter 2023-04-06

1. Genomför följande aritmetiska operationer på 4-bitars binär form:

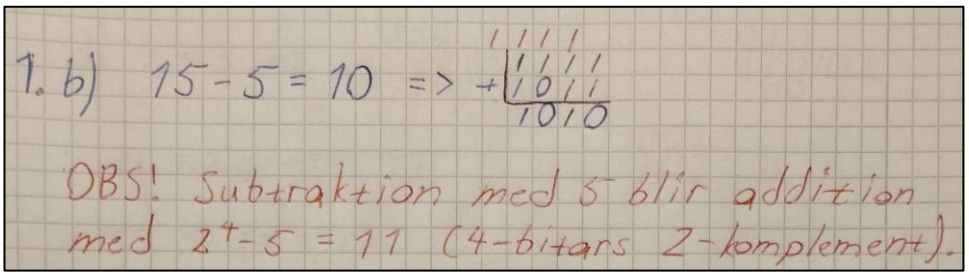
a) 5 + 7
b) 15 - 5
c) 5 * 3
d) 15 / 5

Lösningsförslag:

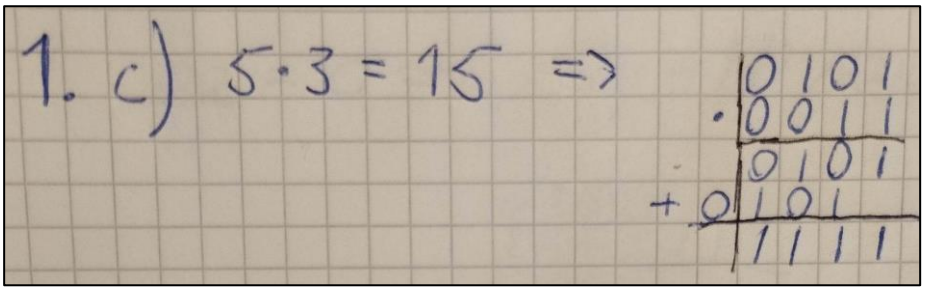
Binära operationer kan genomföras på samma sätt som för decimala tal. I detta fall används liggande stolen för division.



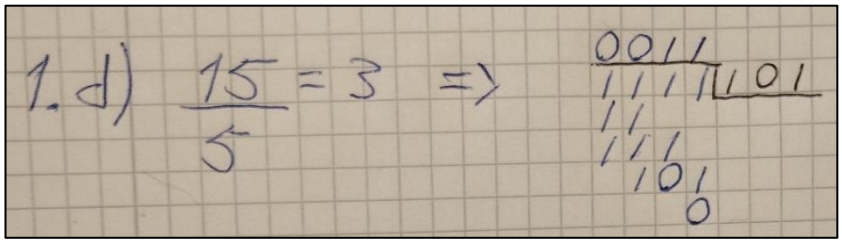
Figur 1: Binär addition.



Figur 2: Binär subtraktion med 4-bitars 2-komplement.



Figur 3: Binär multiplikation.



Figur 4: Binär division med liggande stolen.

Division med liggande stolen sker från vänster till höger, där kvot q samt rest r beräknas:

- $1 < 101 \Rightarrow q_3 = 0$, lägg till biten till höger => division med 11 i stället för 1
- $11 < 101 \Rightarrow q_2 = 0$, lägg till biten till höger => division med 111 i stället för 11
- $111 / 101 = 7 / 5 \Rightarrow q_1 = 1, r_0 = 10$ => lägg till biten till höger => nästa gång division med 101
- $101 / 101 = 5 / 5 \Rightarrow q_0 = 1, r_0 = 0$ => Inga fler bitar till höger, beräkningen är klar

Därmed gäller att kvoten q = 0011 => 15 / 3 = 1111 / 101 = 0011.

2. Rita upp en OR-grind med CMOS-transistorer och visa spänningsfallen i kretsen för samtliga kombinationer 00 – 11 av insignaler A och B.

Lösningsförslag:

För en OR-grind gäller att

- X = 1 om A = 1 eller B = 1, vilket skrivs som X = A + B.
- X = 0 om A = 0 och B = 0, vilket skrivs som X’ = A’ * B’.

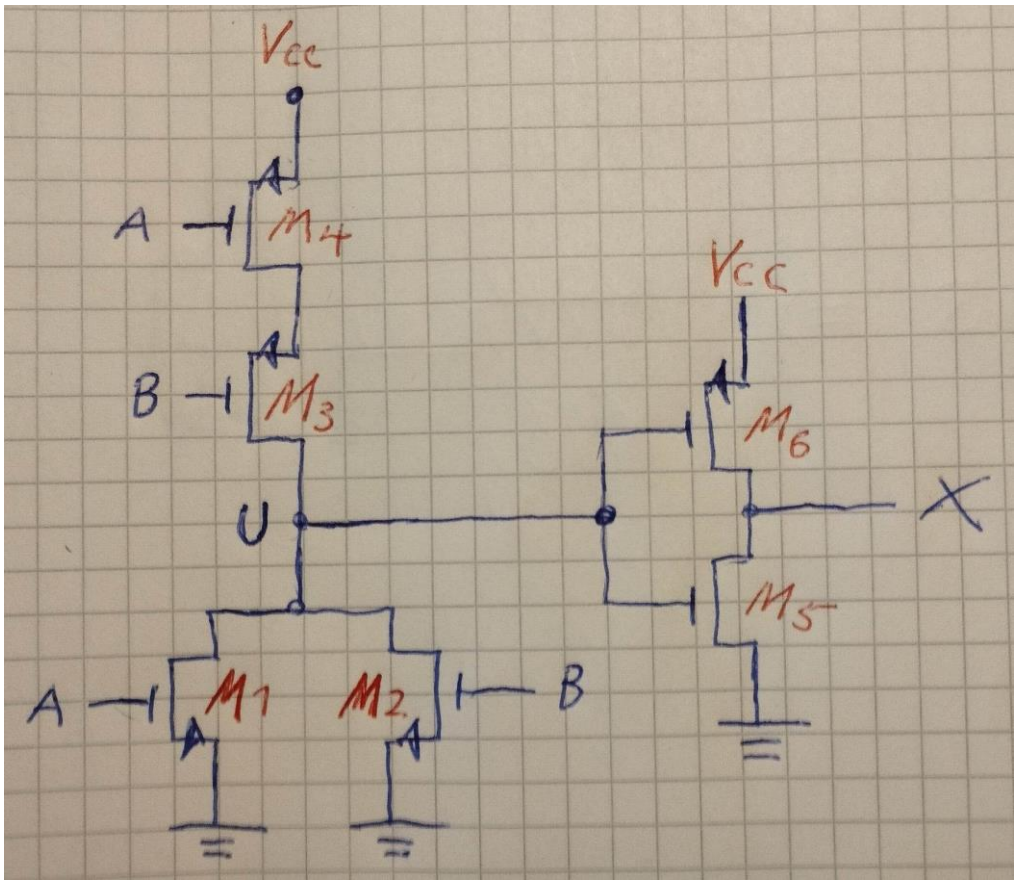
Det nedre och övre nätet ska ha motsatt funktion, så att endast en av delarna leder vid ett givet tillfälle. Vi kopplar det nedre nätet som X = A + B och det övre nätet som X’ = A’ * B’ och får då en NOR-grind (då transistorswitchar inverterar naturligt när de leder. Sedan kopplas en NOT-grind på NOR-grindens utgång för att realisera OR-grinden.

Se figuren nedan. Notera att signal X = U’, dvs. inversen till U:

- Koppla det nedre nätet som X = A + B => parallella transistorer med A och B som insignaler. För att det nedre nätet ska leda måste då antingen A eller B vara 1. Då blir signalen U = 0 och därmed X = 1.

- Koppla det övre nätet som X’ = A’ * B’ => seriella transistorer med A och B som insignaler (A’ respektive B’ kan läsas som att av respektive B måste vara 0 för att ansluten transistor ska leda). För att det övre nätet ska leda måste då både A och B vara 0. Då blir signalen U = 1 och därmed X = 0.

- Vi har nu skapat en NOR-grind. Vi ansluter en NOT-grind till utgången och realiserar då en OR-grind.



Figur 5: OR-grind realiserad med CMOS-transistorer.

3. Förklara skillnaden mellan deklaration av ett package samt en package body i VHDL.

Lösningsförslag:

Ett package består av en header (kallad package) samt en kropp (kallad package body):

- Package innehåller deklaration av konstanter, typer, submoduler och komponenter.
- Package body innehåller definition av submoduler (funktioner och procedurer).

Som exempel, se package *misc* i bilaga A.

4. Definiera en funktion i VHDL som indikerar ifall ett givet tal *number* är jämnt eller inte genom att returnera *true* eller *false.* Talet i fråga ska tillhöra datatypen *natural.*

Lösningsförslag:

```
--------------------------------------------------------------------------------
-- number_is_even: Indicates if specified number is even.
--
--                 - number: The number to check.
--
--                 - return: True if the number is even, else false.
--------------------------------------------------------------------------------
function number_is_even(constant number: natural)
return boolean is
constant bits: std_logic_vector(31 downto 0) := std_logic_vector(to_unsigned(number, 32));
begin
    if (bits(0) = '0') then
        return true;
    else
        return false;
    end if;
end function;
```

,

Bilaga A – Filen *misc.vhd* innehållande package *misc*

```
--------------------------------------------------------------------------------
-- misc.vhd: Contains miscellaneous functions and constants via package misc.
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package misc is

--------------------------------------------------------------------------------
-- Binary codes for hexadecimal digits 0x00 - 0x0F:
--------------------------------------------------------------------------------
constant OFF  : std_logic_vector(6 downto 0) := "1111111";
constant ZERO : std_logic_vector(6 downto 0) := "1000000";
constant ONE  : std_logic_vector(6 downto 0) := "1111001";
constant TWO  : std_logic_vector(6 downto 0) := "0100100";
constant THREE: std_logic_vector(6 downto 0) := "0110000";
constant FOUR : std_logic_vector(6 downto 0) := "0011001";
constant FIVE : std_logic_vector(6 downto 0) := "0010010";
constant SIX  : std_logic_vector(6 downto 0) := "0000010";
constant SEVEN: std_logic_vector(6 downto 0) := "1111000";
constant EIGHT: std_logic_vector(6 downto 0) := "0000000";
constant NINE : std_logic_vector(6 downto 0) := "0010000";
constant A    : std_logic_vector(6 downto 0) := "0001000";
constant B    : std_logic_vector(6 downto 0) := "0000011";
constant C    : std_logic_vector(6 downto 0) := "1000110";
constant D    : std_logic_vector(6 downto 0) := "0100001";
constant E    : std_logic_vector(6 downto 0) := "0000110";
constant F    : std_logic_vector(6 downto 0) := "0001110";

--------------------------------------------------------------------------------
-- to_natural: Converts a std_logic_vector to a natural (unsigned) number.
--
--             - vector: The vector whose content is to be converted.
--
--             - return: Corresponding natural number.
--------------------------------------------------------------------------------
function to_natural(constant vector: std_logic_vector)
return natural;

--------------------------------------------------------------------------------
-- to_std_logic_vector: Converts a natural (unsigned) number to std_logic_vector.
--
--                      - number     : The number to be converted.
--                      - vector_size: The size (number of bits) of the
--                                     returned vector (default = 32).
--
--                      - return: Corresponding std_logic_vector (i.e. bits).
--------------------------------------------------------------------------------
function to_std_logic_vector(constant number     : natural;
                             constant vector_size: natural := 32)
return std_logic_vector;

--------------------------------------------------------------------------------
-- add: Adds specified numbers and returns the sum as std_logic_vector.
--
--      - x          : Vector containing the first number.
--      - y          : Vector containing the second number.
--      - vector_size: The size of the returned vector (default = 32).
--
--      - return: The sum of the numbers as std_logic_vector.
--------------------------------------------------------------------------------
function add(constant x, y       : std_logic_vector;
             constant vector_size: natural := 32)
return std_logic_vector;
```

```
    ------------------------------------------------------------------------------
    -- subtract: Subtracts specified numbers with 2-complement and returns the
    --           difference as std_logic_vector.
    --
    --           - x          : Vector containing the first number.
    --           - y          : Vector containing the second number.
    --           - vector_size: The size of the returned vector (default = 32).
    --
    --           - return: The difference between the numbers as std_logic_vector.
    ------------------------------------------------------------------------------
    function subtract(constant x, y       : std_logic_vector;
                      constant vector_size: natural := 32)
    return std_logic_vector;

    ------------------------------------------------------------------------------
    -- get_binary_code: Returns the binary code of specified hexadecimal digit.
    --
    --                  - digit: Hexadecimal digit whose binary code is returned.
    --
    --                  - return: The binary code of specified digit.
    ------------------------------------------------------------------------------
    function get_binary_code(constant digit: std_logic_vector(3 downto 0))
    return std_logic_vector;

    end package;

    package body misc is

    ------------------------------------------------------------------------------
    -- to_natural: Converts a std_logic_vector to a natural (unsigned) number.
    --
    --             - vector: The vector whose content is to be converted.
    --
    --             - return: Corresponding natural number.
    ------------------------------------------------------------------------------
    function to_natural(constant vector: std_logic_vector)
    return natural is
    begin
        return to_integer(unsigned(vector));
    end function;

    ------------------------------------------------------------------------------
    -- to_std_logic_vector: Converts a natural (unsigned) number to std_logic_vector.
    --
    --                      - number     : The number to be converted.
    --                      - vector_size: The size (number of bits) of the
    --                                     returned vector (default = 32).
    --
    --                      - return: Corresponding std_logic_vector (i.e. bits).
    ------------------------------------------------------------------------------
    function to_std_logic_vector(constant number     : natural;
                                 constant vector_size: natural := 32)
    return std_logic_vector is
    begin
        return std_logic_vector(to_unsigned(number, vector_size));
    end function;
```

```
    --------------------------------------------------------------------------------
    -- add: Adds specified numbers and returns the sum as std_logic_vector.
    --
    --      - x           : Vector containing the first number.
    --      - y           : Vector containing the second number.
    --      - vector_size: The size of the returned vector (default = 32).
    --
    --      - return: The sum of the numbers as std_logic_vector.
    --------------------------------------------------------------------------------
    function add(constant x, y       : std_logic_vector;
                 constant vector_size: natural := 32)
    return std_logic_vector is
    constant sum: natural := to_natural(x) + to_natural(y);
    begin
       return to_std_logic_vector(sum, vector_size);
    end function;

    --------------------------------------------------------------------------------
    -- subtract: Subtracts specified numbers with 2-complement and returns the
    --           difference as std_logic_vector.
    --
    --           - x           : Vector containing the first number.
    --           - y           : Vector containing the second number.
    --           - vector_size: The size of the returned vector (default = 32).
    --
    --           - return: The difference between the numbers as std_logic_vector.
    --------------------------------------------------------------------------------
    function subtract(constant x, y       : std_logic_vector;
                      constant vector_size: natural := 32)
    return std_logic_vector is
    constant difference: natural := to_natural(x) + (2 ** vector_size - to_natural(y));
    begin
       return to_std_logic_vector(difference, vector_size);
    end function;

    --------------------------------------------------------------------------------
    -- get_binary_code: Returns the binary code of specified hexadecimal digit.
    --
    --                  - digit: Hexadecimal digit whose binary code is returned.
    --
    --                  - return: The binary code of specified digit.
    --------------------------------------------------------------------------------
    function get_binary_code(constant digit: std_logic_vector(3 downto 0))
    return std_logic_vector is
    begin
        case (digit) is
            when "0000" => return ZERO;
            when "0001" => return ONE;
            when "0010" => return TWO;
            when "0011" => return THREE;
            when "0100" => return FOUR;
            when "0101" => return FIVE;
            when "0110" => return SIX;
            when "0111" => return SEVEN;
            when "1000" => return EIGHT;
            when "1001" => return NINE;
            when "1010" => return A;
            when "1011" => return B;
            when "1100" => return C;
            when "1101" => return D;
            when "1110" => return E;
            when "1111" => return F;
            when others => return OFF;
        end case;
    end function;

    end package body;
```