FPGA tutorial

Lecture 4

18.11.2020

Jochen Steinmann

III. Physikalisches Institut B

RWTH AACHEN UNIVERSITY


Solution

Solution 03a

Counter with GENERIC

```
entity counter is
    generic(
        DATA_WIDTH    :  integer := 8
    );
    port(
        i_sl_CLK      :  in  std_logic;                                    -- clock
        i_sl_en       :  in  std_logic;                                    -- enable
        i_sl_dir      :  in  std_logic;                                    -- direction
        o_slv_counter :  out std_logic_vector(DATA_WIDTH-1  downto 0)    -- counter
    );
end counter;

---------------------------------------

architecture behavior of counter is
    signal cnt : unsigned(DATA_WIDTH-1 downto 0) := (others => '0');  -- define signal, because we cannot readback outputs
begin

    process(i_sl_CLK)
    begin
        if rising_edge(i_sl_CLK) then
            if (i_sl_en = '1') then
                if(i_sl_dir = '0') then
                    cnt <= cnt + 1;
                else
                    cnt <= cnt - 1;
                end if;
            end if;
            o_slv_counter <= std_logic_vector(cnt);
        end if;
    end process;

end behavior;
```

Solution 03c

Testbench

```
BEGIN
    -- Instance of unit under test.
    uut: counter
    PORT MAP (
        i_sl_CLK    => CLK,
        i_sl_en     => en,
        i_sl_dir    => dir,
        o_slv_counter => o_slv_counter
    );

    uut2: counter
    GENERIC MAP(
        DATA_WIDTH => 9
    )
    PORT MAP (
        i_sl_CLK    => CLK,
        i_sl_en     => en,
        i_sl_dir    => dir,
        o_slv_counter => o_slv_counter2
    );

-------------------------------------------
    -- test bench definition.
    tb_CLK :process
    begin
        CLK <= '0';
        wait for 1 ns;
        CLK <= '1';
        wait for 1 ns;
    end process;

END;
```

Solution 03 a

ads

Solution 03b

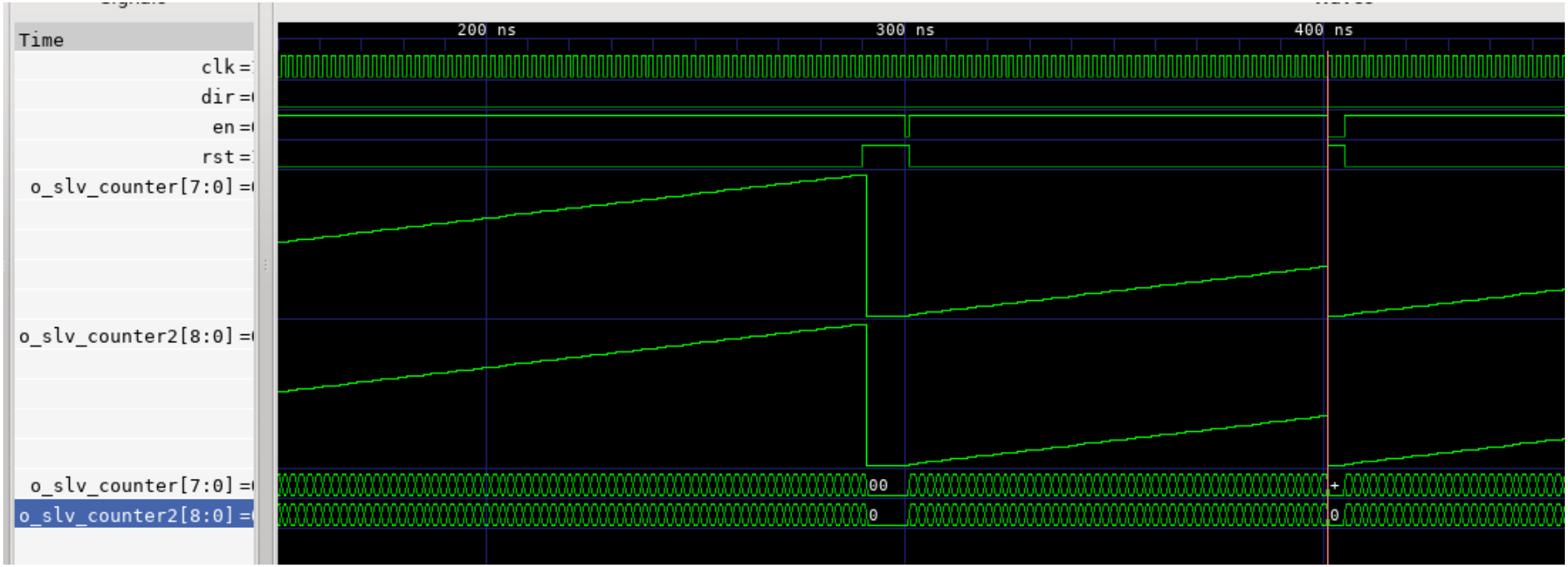
Counter with RESET

```
entity counter is
    generic(
        DATA_WIDTH    :  integer := 8
    );
    port(
        i_sl_CLK      :  in  std_logic;                                  -- clock
        i_sl_en       :  in  std_logic;                                  -- enable
        i_sl_dir      :  in  std_logic;                                  -- direction
        i_sl_rst      :  in  std_logic;                                  -- reset
        o_slv_counter :  out std_logic_vector(DATA_WIDTH-1  downto 0)    -- counter
    );
end counter;
```

```
architecture behavior of counter is
    signal cnt : unsigned(DATA_WIDTH-1 downto 0) := (others => '0');  -- define signal, because we cannot readback outputs
begin

    process(i_sl_CLK)
    begin
        if rising_edge(i_sl_CLK) then
            if (i_sl_en = '1') then
                if( i_sl_rst = '1' ) then       -- option 1 ( reset when enabled )
                    cnt <= (others => '0');
                else                            -- we have to make sure, that we
                                                -- are not resetting and counting
                                                -- at the same time
                    if(i_sl_dir = '0') then
                        cnt <= cnt + 1;
                    else
                        cnt <= cnt - 1;
                    end if;
                end if;
            elsif ( i_sl_rst = '1') then        -- option 2 ( reset when disabled )
                cnt <= (others => '0');
            end if;
--          o_slv_counter <= std_logic_vector(cnt); -- option 1a  ( o_slv_counter is always 1 clock cycle late related to cnt )
        end if;
        o_slv_counter <= std_logic_vector(cnt);     -- option 1b  ( o_slv_counter is always 1 clock cycle late related to cnt )
    end process;

    -- o_slv_counter <= std_logic_vector(cnt);     -- option 2 ( o_slv_counter and cnt have the same value at the same clock cycle)

end behavior;
```

Solution 03 b

asd

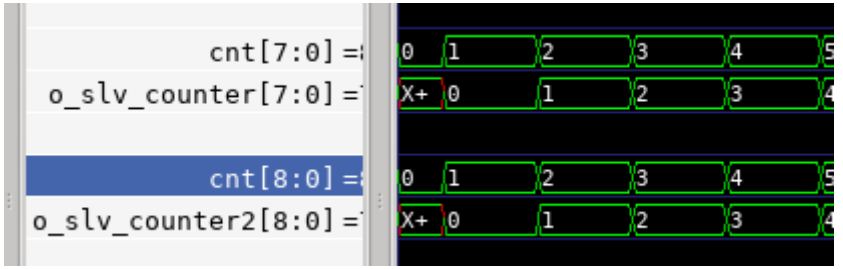


A short look, where to assign the outputs

Inside CLK process or outside

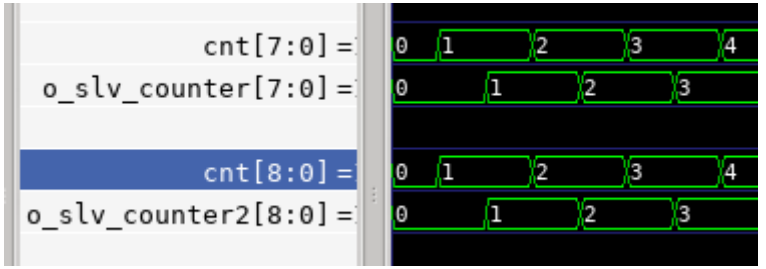
- Option 01 inside CLK process
  - Rising edge

Output delayed by 1 clock cycle



  - No rising egde

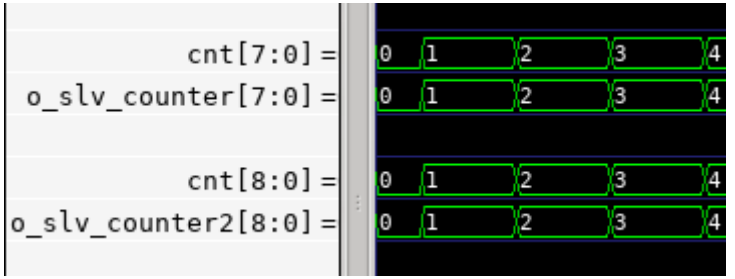
Output delayed by 0.5 clock cycle



Action on falling edge
– should be avoided

- Option 02 outside CLK process

Output delayed by 0 clock cycle



Solution 03c

```
entity pwm is
    generic(
        DATA_WIDTH    :  integer := 8
    );
    port(
        i_sl_CLK      :  in  std_logic;                                  -- clock
        i_slv_val     :  in  std_logic_vector(DATA_WIDTH-1 downto 0);    -- pwm value
        o_sl_pwm      :  out std_logic                                   -- pwm output
    );
end pwm;
```

```
    signal slv_pwm_cnt : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');

begin
    pwm_cnt: counter
    GENERIC MAP(
        DATA_WIDTH => DATA_WIDTH
    )
    PORT MAP (
        i_sl_CLK    => i_sl_clk,
        i_sl_en     => '1',
        i_sl_dir    => '0',
        i_sl_rst    => '0',
        o_slv_counter => slv_pwm_cnt
    );


    process(i_sl_CLK)
    begin
        if rising_edge(i_sl_CLK) then
            if unsigned(slv_pwm_cnt) >= unsigned(i_slv_val) then
                o_sl_pwm <= '0';
            else
                o_sl_pwm <= '1';
            end if;
        end if;
    end process;

end behavior;
```

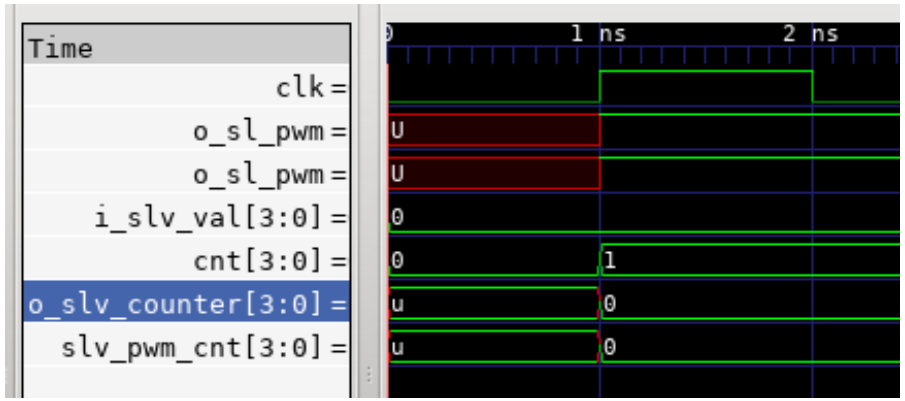
Solution 03 c

PWM

You might have seen some warning like this:

../../src/ieee2008/numeric_std-body.vhdl:1624:7:@1ns:(assertion warning): NUMERIC_STD.">=": metavalue detected, returning FALSE

At the beginning there is (depending on the implementation) a short time of undefined:




NEW

VHDL

for loop

For writing testbenches or even some algorithms, sometimes, it is useful to use loops

```
for I in 0 to 3 loop
      if (A = I) then
            Z(I) <= '1';
       end if;
end loop;
```

There is no need to define the loop parameter (e.g. I)

Calculation with GENERIC

- Sometimes, you have to give certain values, which rely on each other
- As an example, you have to constrain the vector width and the numbers
  - Let‘s assume BIT_WIDTH is 3
  - std_logic_vector(BIT_WIDTH -1 downto 0) 3 bit (2 – 0)
  - std_logic_vector((2** BIT_WIDTH) -1 downto 0) 2^3 = 8 bit (7 – 0)

VHDL – CASE

- Also case is quite useful, when we want to build some logic

```
C2: case y is
        when  "00"    => Out_2 <= 0;
        when  "01"    => Out_2 <= 1;
        when others   => Out_2 <= 3;   -- would be “10” and “11”
end case C2;
```

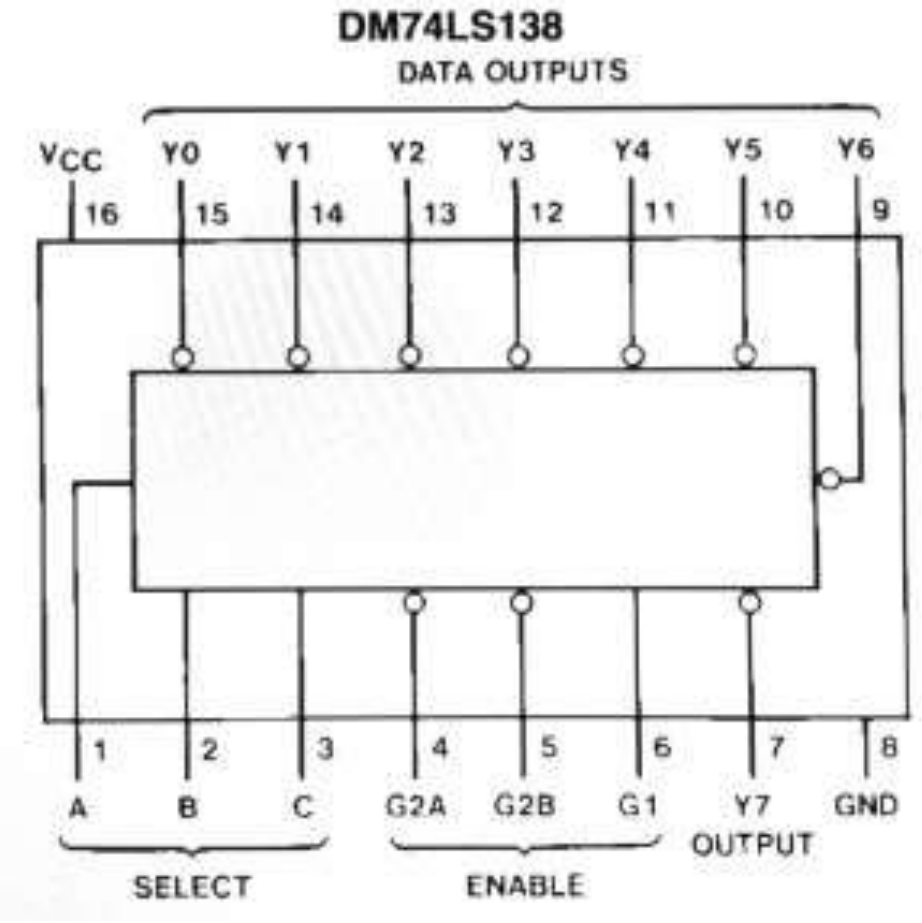
```
CASE input IS
        WHEN "00" =>
                output0 <= '1' ;
                output1 <= '0' ;
                output2 <= '0' ;
                output3 <= '0' ;
```

Multiple „actions“ are also possible

Excercises

Talking to many devices

- Address decoder
- selects only one device
  3 out of 8
- 2 different types
  - one hot (1)
  - one cold (0)
- all others inverted
  logic level



one famous type:
74 HC 138

Logic table of 74HC138

FUNCTION TABLE

ENABLE INPUTS			SELECT INPUTS			OUTPUTS							
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
			L	L	L	L	H	H	H	H	H	H	H
			L	L	H	H	L	H	H	H	H	H	H
			L	H	L	H	H	L	H	H	H	H	H
			L	H	H	H	H	H	L	H	H	H	H
			H	L	L	H	H	H	H	L	H	H	H
			H	L	H	H	H	H	H	H	L	H	H
			H	H	L	H	H	H	H	H	H	L	H
			H	H	H	H	H	H	H	H	H	H	L

disabled

Need of an address decoder?

- Independent:
  - outside FPGA
  - inside FPGA
- Different logic blocks (chips) have to be addressed.
  - ensure that only one is active and talks with the master
  - if multiple devices (slave) are active, may cause short circuits and lot of trouble

Excercise 12

- Address encoder
  - if talking to multiple chips, you have to ensure, that only one is selected at the same time!
  - Two options
    - one hot → selected chip 1, others 0
    - one cold → selected chip 0, others 1
  - Input
    - 2 bit (0-3)
  - Output
    - 4 bit

Hint:

you can use a bus to access elements of a bus
a[b] → will access the bit b of a

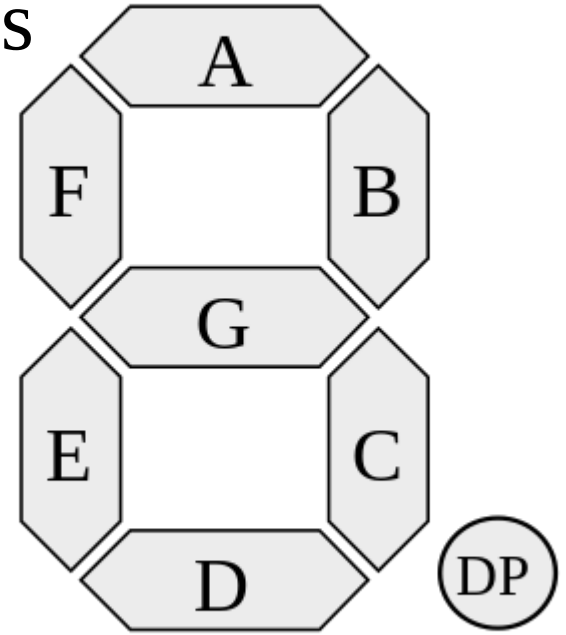
Output to humans

- There are many ways how an FPGA device can communicate with the “operator”
  - High energy physics
    - mostly remote access → e.g. Network etc.
  - Consumer devices
    - need graphical menues
      - LEDs → easy
      - **7-Segment displays → more complex, but still simple**
      - LCDs → too complex for this course

Difference

- Humans use **decimal** system
- FPGAs and computers use **binary** system

- Need to convert between different worlds
  - 1001 → 9

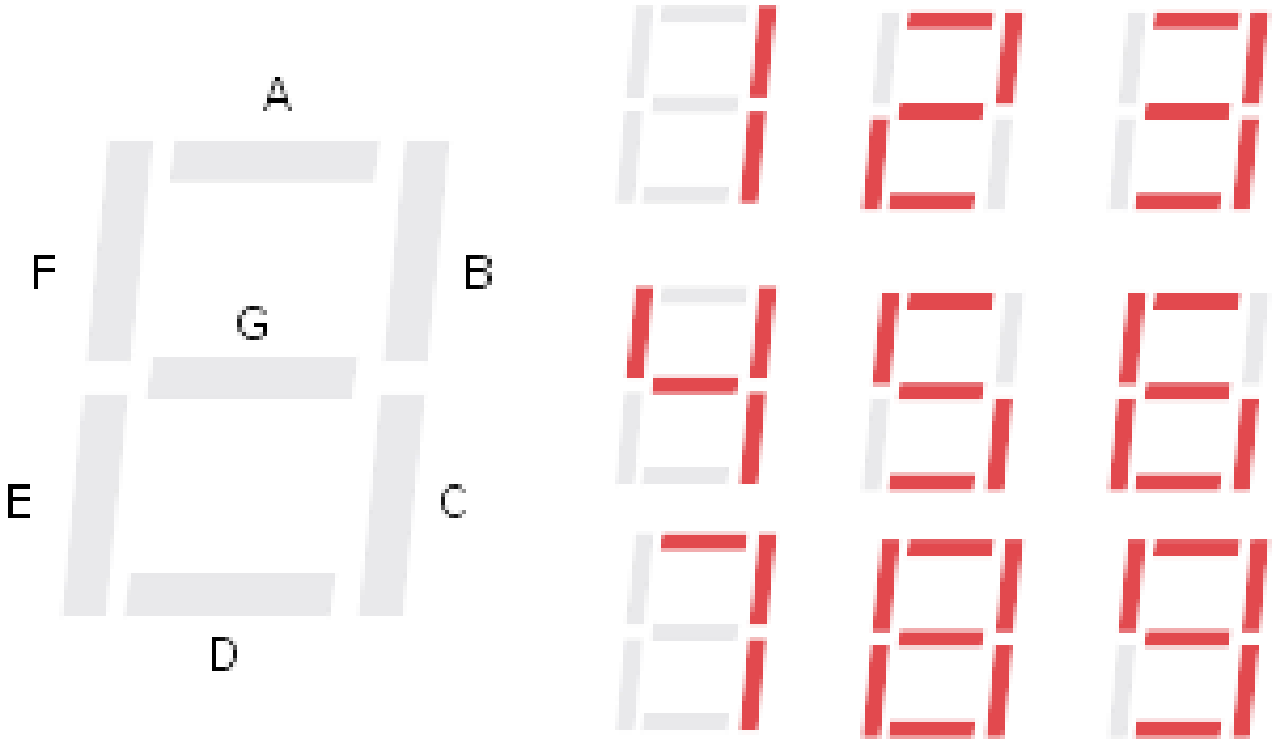


1st 7 segment lookup table

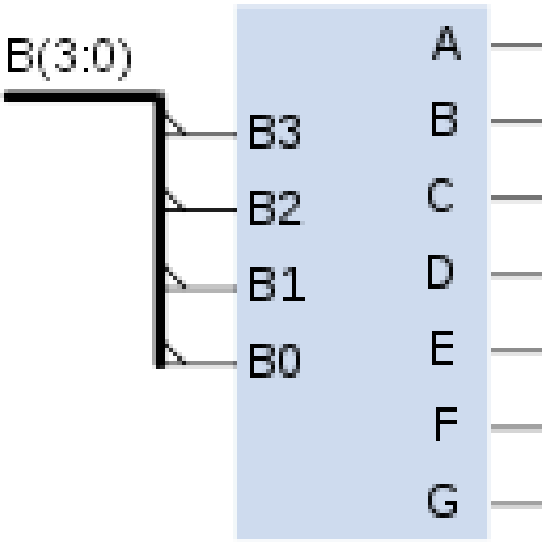
Excercise 13

- Heading forwards to display numbers on the 7-segement display
- Lookup table for 7 segment display
  - 7 segments can display 10 values 0 – 9
  - depending on the input different outputs have to be switched on
- Inputs 4 Bit → sw3 – sw0
- Outputs 8 Bit → 7 Segments

Excercise 13



An un-illuminated seven-segment display, and nine illumination patterns corresponding to decimal digits

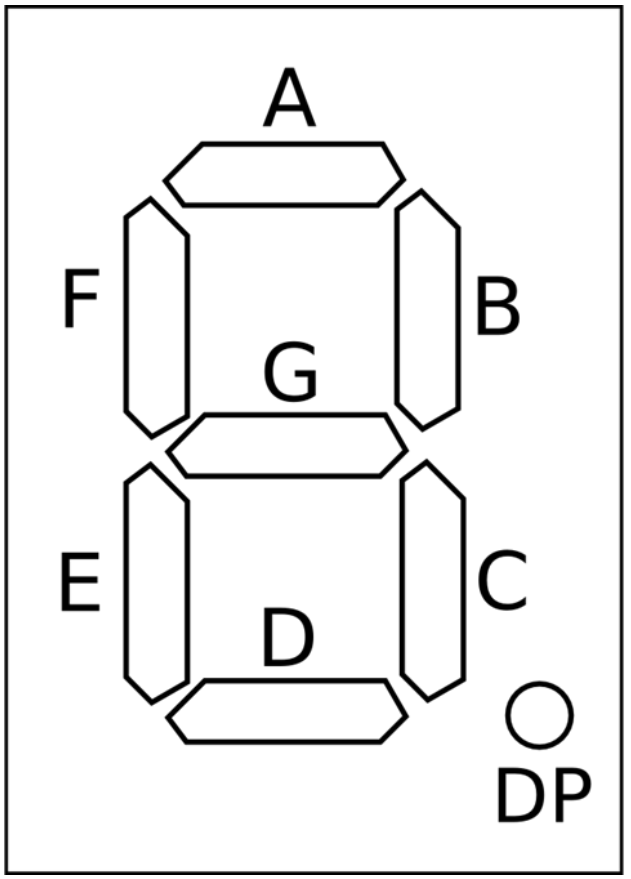


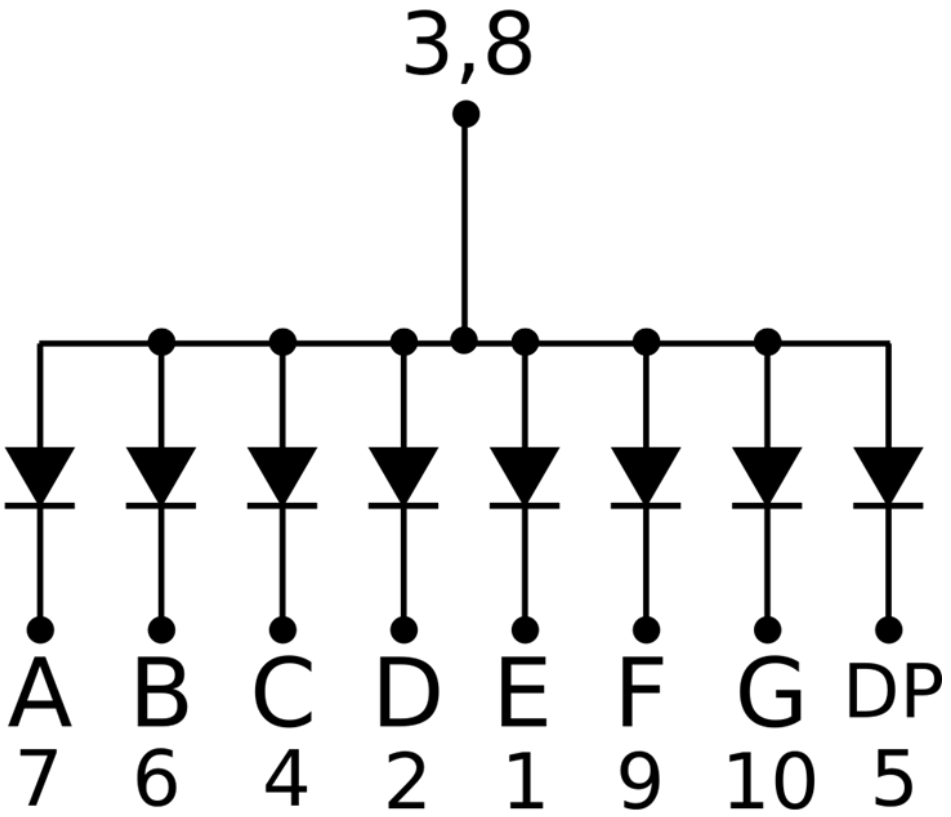
one Bus

segments = “0000011”;
A + B on

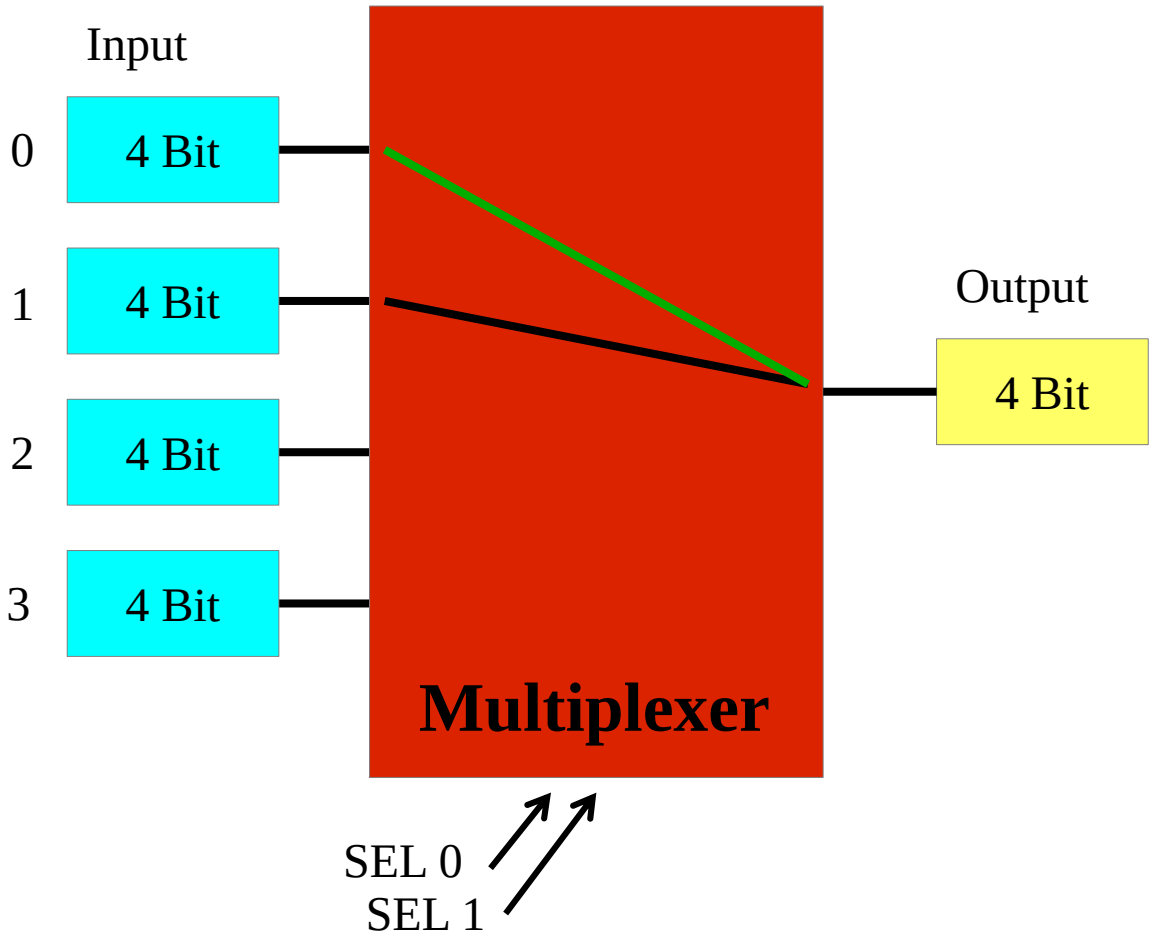
Excercise 13

Pin 1





Multiplexing



SEL1	SEL0	OUT
0	0	0
0	1	1
1	0	2
1	1	3

Excercise 14

- Create a 16 bit (= 4 x 4bit) to 4bit multiplexer
- Inputs
  - sel 2 bit, which select which part of the input is “connected” to the output
  - in 16 bit
  - out 4 bit

ToDo

04a address decoder

(de-) select one bit of 8

3 bit input

3 to 8 (one cold)

3 to 8 (one hot)

04b 7 segment lookup table

map 4 input bits to the 7 segments, which should light up

use case - structure

04c Multiplexer

Multiplex 4 x 4bit to 1 x 4 bit depending on the selected input

Have a look into the skeletons folder