University Bremerhaven

INSTITUTE FOR AUTOMATION AND ELECTR. ENG.

Prof. Dr.-Ing. Kai Mueller

Sample Test Digital Systems / VHDL ST-1

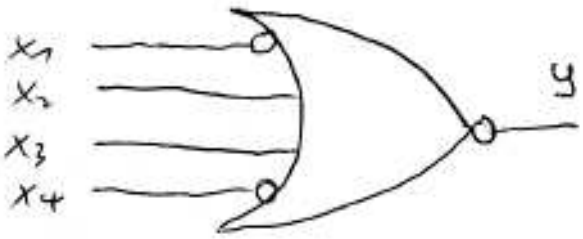
Time 120 minutes
– class documents allowed –

Solutions

(1) Boolean Logic

(1.1) DeMorgan Theorem

$$y = x_1 \cdot x_2' \cdot x_3' \cdot x_4 = y'' = \left(x_1' + x_2 + x_3 + x_4'\right)'$$



(1.2) Duality Theorem

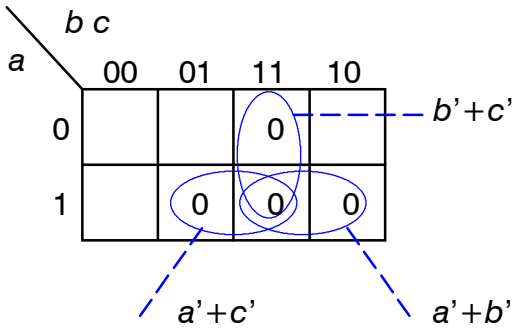
Apply the Duality theorem to $(x + 0) \cdot (x + 1) = x$

$x \cdot 1 + x \cdot 0 = x$ (no parentheses required)

$x \cdot 1 + x \cdot 0 = x + 0 = x$, q.e.d.

(1.3) Logic Minimization

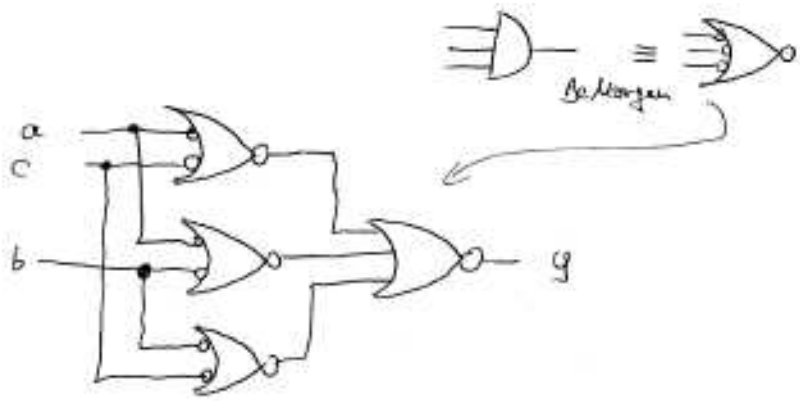
product of sums means “zeros”



$y = (a'+c') \cdot (a'+b') \cdot (b'+c')$

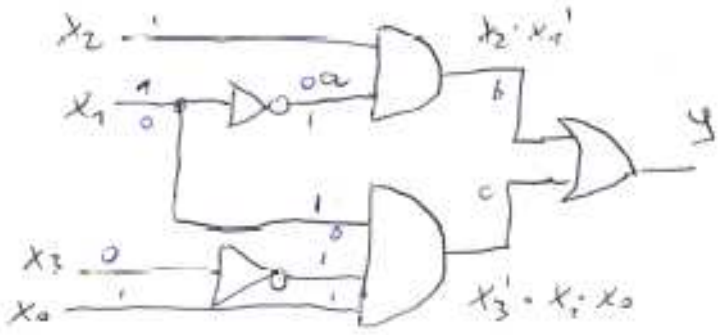
Complete circuit diagram with NOR gates (DeMorgan was applied here):

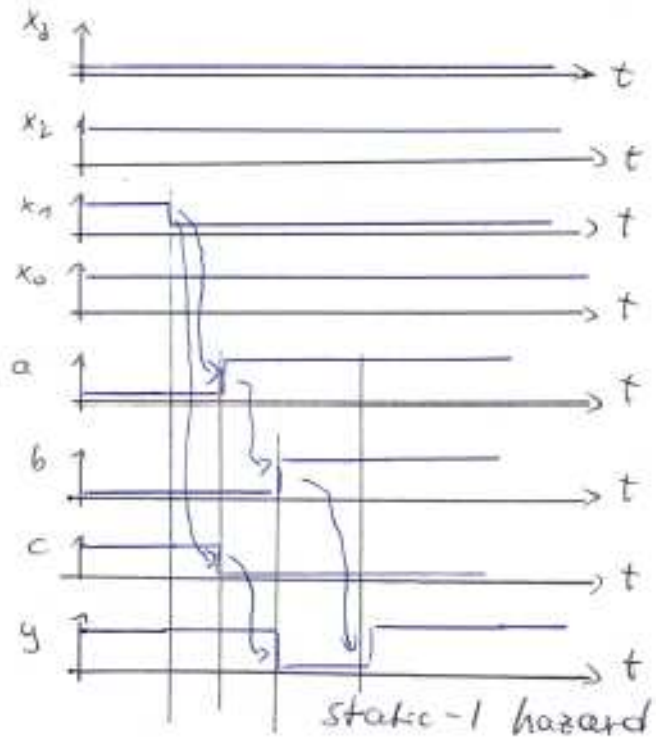
(1) = 6	(2) = 6	(3) = 6	(4) = 6	(5) = 6	Σ = 30
1.0 ≥ 29	1.3 ≥ 27	1.7 ≥ 26	2.0 ≥ 24	2.3 ≥ 23	2.7 ≥ 21	3.0 ≥ 20	3.3 ≥ 18	3.7 ≥ 17	4.0 ≥ 15

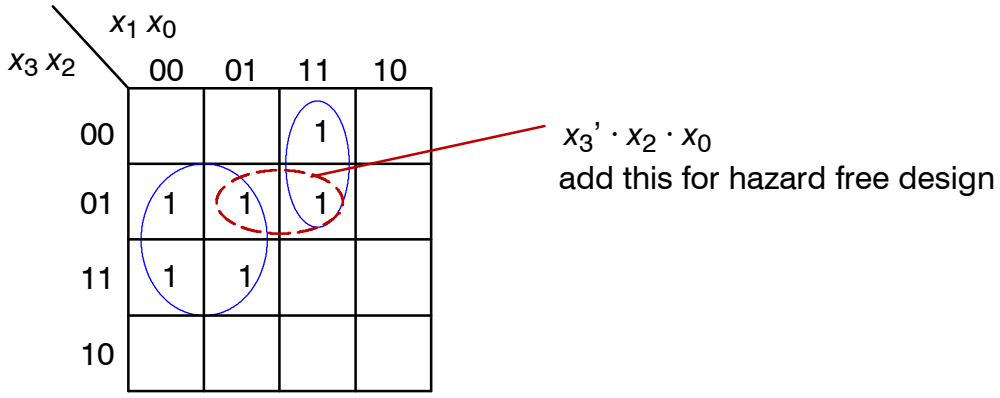


(1.4) Hazards

x3	x2	x1	x0	y	
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	1	
0	**1**	**0**	**1**	**1**	<-+
0	1	1	0	0	| (in both cases output is ’1’)
0	**1**	**1**	**1**	**1**	--+
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	0	
1	1	0	0	1	
1	1	0	1	1	
1	1	1	0	0	
1	1	1	1	0	







Hazard-free version:

$y = x_2 \cdot x_1' + x_3' \cdot x_1 \cdot x_0 + x_3' \cdot x_2 \cdot x_0$ (one additional AND).

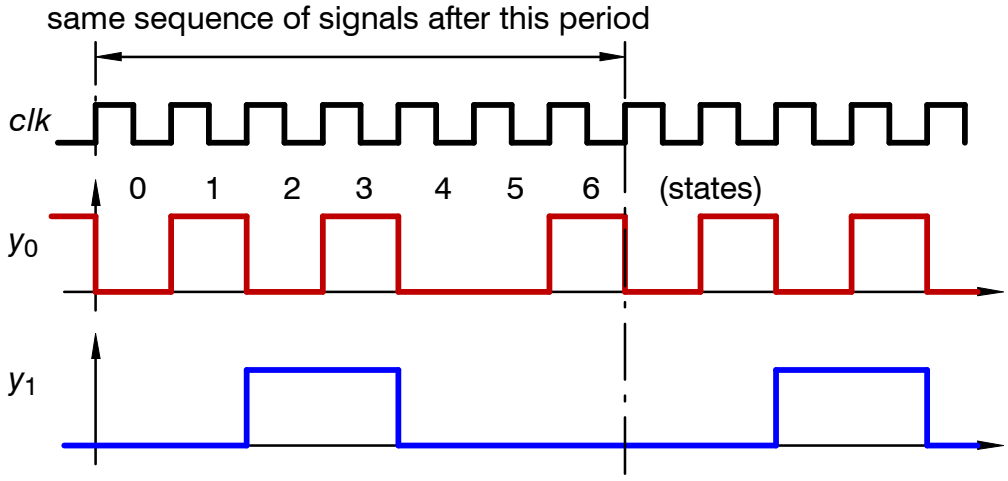
(2) Hardware State Machine (Moore Machine)

The inputs *reset* and *start* have a similar effect, so a new signal

operate = *reset* ' AND *start*

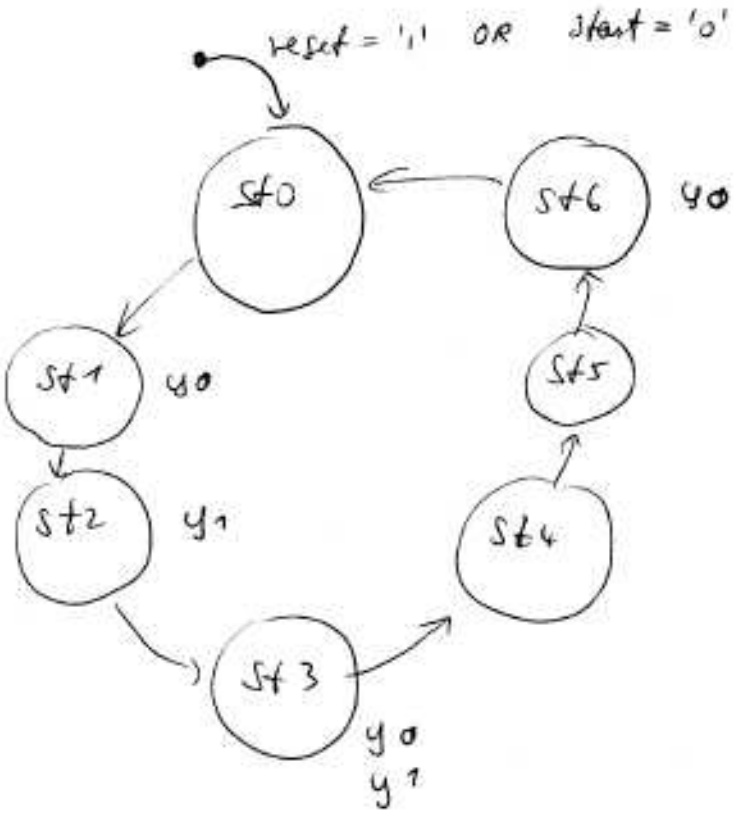
can be used to create a (simpler) next state logic.

operate	x2	x1	x0	x2n	x1n	x0n	(next states)
0	x	x	x	0	0	0	
1	0	0	0	0	0	1	
1	0	0	1	0	1	0	
1	0	1	0	0	1	1	
1	0	1	1	1	0	0	
1	1	0	0	1	0	1	
1	1	0	1	1	1	0	
1	1	1	0	0	0	0	
1	1	1	1	x	x	x	(don’t cares)



(2.1) State machine has 7 different states; 3 FFs are required (would allow 8 states).

(2.2) State diagram



(2.3) Next-state logic (from above truth table)

Op x_2 \ x_1 x_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	X	0
10	0	0	1	0

x2n

x2n = Op · (x1'+x0) · (x2+x1)

Op x_2 \ x_1 x_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	1	X	0
10	0	1	0	1

x1n

x1n = Op · x1’ · x0
+ Op · x2’ · x1 · x0’

Op x_2 \ x_1 x_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	0	X	0
10	1	0	0	1

x0n

x0n = Op · x0’ · (x2’ + x1’ + x0)

(2.4) Output logic

x2	x1	x0	y0	y1
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	x	x

x2 \ x1 x0	00	01	11	10
0	0	1	1	0
1	0	0	X	1

y0

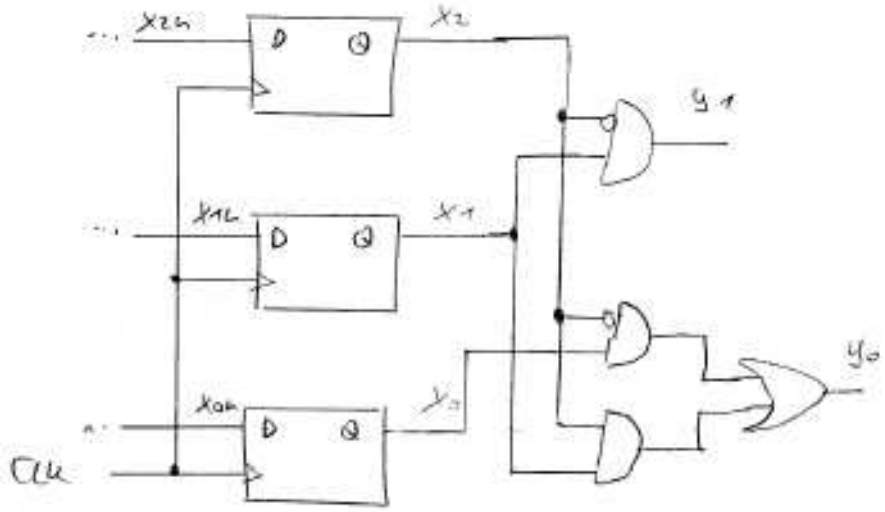
y0 = x2’ · x0 + x2 · x1

x2 \ x1 x0	00	01	11	10
0	0	0	1	1
1	0	0	X	0

y1

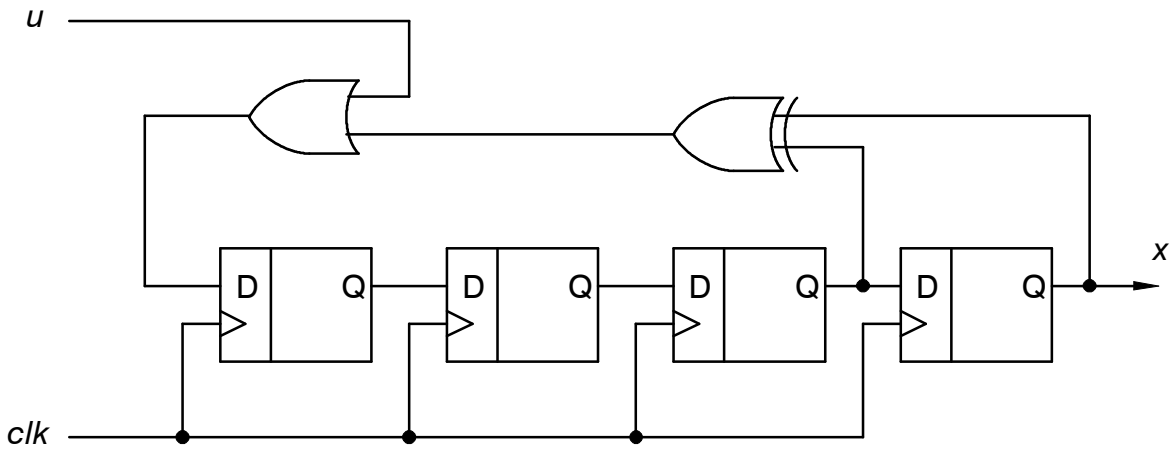
$y1 = x2' \cdot x1$

(2.5) Flip-flops and output logic hardware circuit



(3) Coupled Flip-Flops

The following circuit should be synthesized by VHDL.



(3.1) *Structural design* in VHDL

```
entity prbs_struct is
    Port ( u : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           x : out  STD_LOGIC);
end prbs_struct;

architecture Behavioral of prbs_struct is
SIGNAL  a, b, q0, q1, q2, q3: std_logic;
begin

      x <= q3;
      a <= q2 XOR q3;
      b <= u OR a;

      ffs: PROCESS(clk, u)
      BEGIN
            IF clk'event AND clk='1' THEN
                  q0 <= b;
                  q1 <= q0;
                  q2 <= q1;
                  q3 <= q2;
            END IF;
      END PROCESS ffs;

end Behavioral;
```

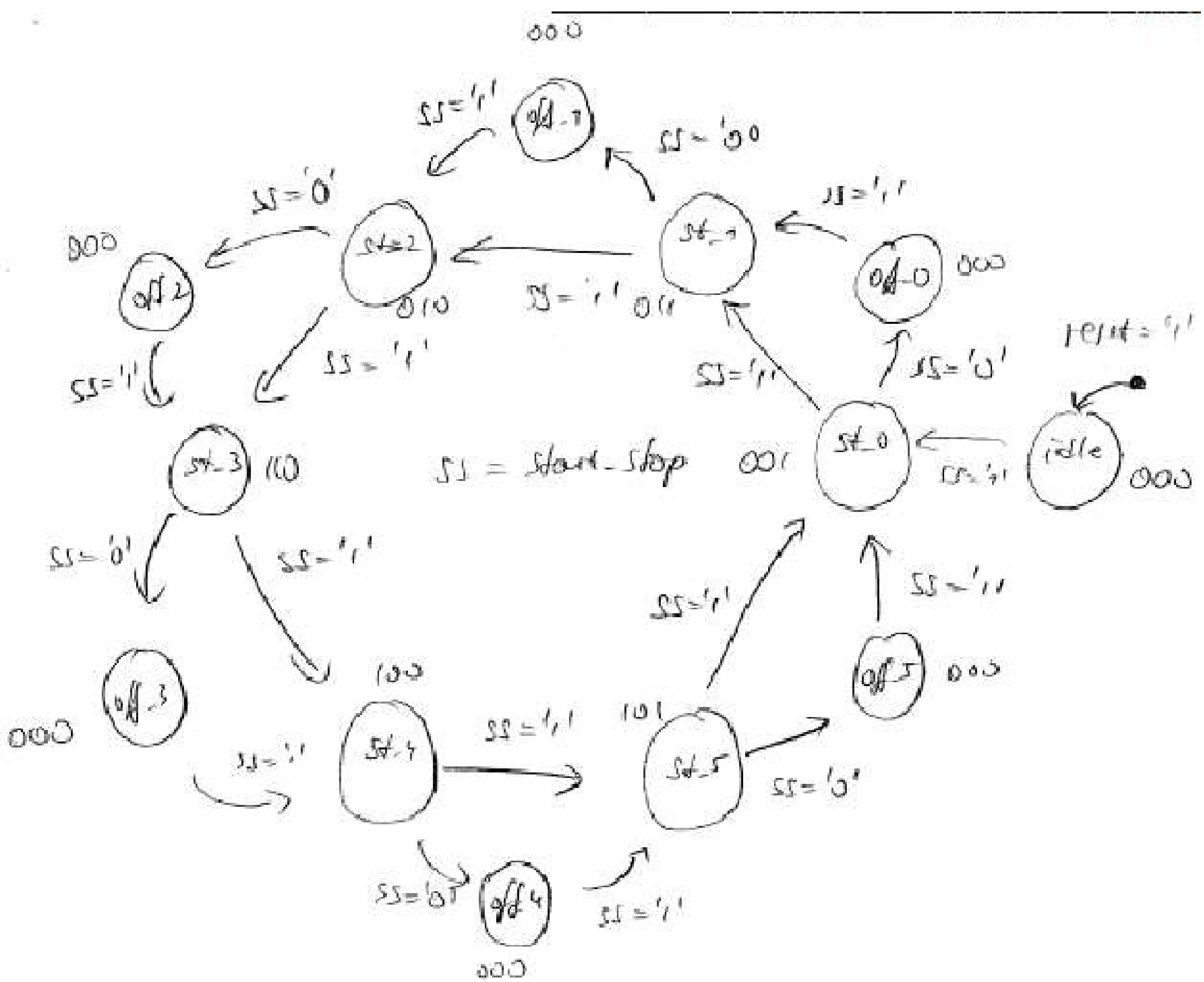
(3.2) Purpose: Signal generation (Pseudo random binary sequence = PRBS)

(3.3) Behavioral design

```
entity prbs_behav is
    Port ( u : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           x : out  STD_LOGIC);
end prbs_struct;

architecture Behavioral of prbs_behav is
SIGNAL  qff : std_logic_vector(3 downto 0);
begin

      x <= qff(3);

      ffs: PROCESS(clk, u)
      BEGIN
            IF clk'event AND clk='1' THEN
                  if u='1' THEN
                        qff <= qff(2 downto 0) & '1';
                  ELSE
                        qff <=
                              qff(2 downto 0) & (qff(2) XOR qff(3));
                  END IF;
            END IF;
      END PROCESS ffs;

end Behavioral;
```

(4) AC Machine Power Transistor Signals

(4.1) State diagram:



(4.2) VHDL module:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity inverter is
    Port ( reset : in  STD_LOGIC;
           start_stop : in  STD_LOGIC;
                 rl_time : in STD_LOGIC_VECTOR(9 DOWNTO 0);
           clk : in  STD_LOGIC;
           STrans : out  STD_LOGIC_VECTOR (2 downto 0));
end inverter;
```

```
architecture Behavioral of inverter is
TYPE state_type IS (idle, st_0, st_1, st_2, st_3, st_4, st_5,
                                    off_0, off_1, off_2, off_3,
off_4, off_5);
SIGNAL  state, next_state : state_type;
begin

      -- timed state machine
      seq : PROCESS(clk, reset, next_state, rl_time)
      VARIABLE  cnt : INTEGER RANGE 0 to 1023 := 0;
      BEGIN
            IF clk'event AND clk='1' THEN
                  IF reset='1' THEN
                        state <= idle;
                        cnt := 0;
                  ELSE
                        cnt := cnt + 1;
                        IF cnt=rl_time THEN
                              state <= next_state;
                              cnt := 0;
                        END IF;
                  END IF;
            END IF;
      END PROCESS seq;

      cmb: PROCESS(state, start_stop)
      BEGIN
            next_state <= state;
            STrans <= "000";        -- default
            CASE state IS
                  WHEN  idle =>
                        IF start_stop='1' THEN
                              next_state <= st_0;
                        END IF;
                  WHEN  st_0 =>
                        IF start_stop='1' THEN
                              next_state <= st_1;
                        ELSE
                              next_state <= off_0;
                        END IF;
                        STrans <= "001";
                  WHEN  st_1 =>
                        IF start_stop='1' THEN
```

```
                next_state <= st_2;
        ELSE
                next_state <= off_1;
        END IF;
        STrans <= "011";
WHEN    st_2 =>
        IF start_stop='1' THEN
                next_state <= st_3;
        ELSE
                next_state <= off_2;
        END IF;
        STrans <= "010";
WHEN    st_3 =>
        IF start_stop='1' THEN
                next_state <= st_4;
        ELSE
                next_state <= off_3;
        END IF;
        STrans <= "110";
WHEN    st_4 =>
        IF start_stop='1' THEN
                next_state <= st_5;
        ELSE
                next_state <= off_4;
        END IF;
        STrans <= "100";
WHEN    st_5 =>
        IF start_stop='1' THEN
                next_state <= st_0;
        ELSE
                next_state <= off_5;
        END IF;
        STrans <= "101";
WHEN off_0 =>
        IF start_stop='1' THEN
                next_state <= st_1;
        END IF;
WHEN off_1 =>
        IF start_stop='1' THEN
                next_state <= st_2;
        END IF;
WHEN off_2 =>
        IF start_stop='1' THEN
```

```
                        next_state <= st_3;
                    END IF;
                WHEN off_3 =>
                    IF start_stop='1' THEN
                        next_state <= st_4;
                    END IF;
                WHEN off_4 =>
                    IF start_stop='1' THEN
                        next_state <= st_5;
                    END IF;
                WHEN off_5 =>
                    IF start_stop='1' THEN
                        next_state <= st_0;
                    END IF;
            END CASE;
        END PROCESS cmb;

    end Behavioral;
```

(4.3) Reversing the sequence of the switches requires an addition input, i.e. “direction”. The IF clauses in the combinational PROCESS need to be modified, for instance:

```
                WHEN  st_2 =>
                    IF start_stop='1' THEN
                        IF direction='0' THEN
                            next_state <= st_3;
                        ELSE
                            next_state <= st_1;
                    ELSIF direction='0' THEN
                        next_state <= off_2;
                    ELSE
                        next_state <= off_1;
                    END IF;
                    STrans <= "010";
```