Programmierbare Logik

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Name, Vorname	
Matrikelnummer	
Studiengang	
Unterschrift	**Tag der Prüfung:** 25. September 2019

Bitte beachten!

1. Prüfen Sie, ob Ihre Klausur vollständig ist. Sie muss aus den durchnummerierten Seiten von 1 bis 12 bestehen. Nehmen Sie die Klausur bitte nicht auseinander. Falls Sie ein unvollständiges Exemplar erhalten haben, lassen Sie sich bitte eine einwandfreie Klausur aushändigen.
2. Zum Bestehen der Klausur sind 50% der Punktzahl - Summe der Punkte aus der Laborübung plus erreichte Punkte der Klausur - erforderlich.
3. Die Bearbeitungszeit beträgt 90 Minuten.
4. Außer einfachen (nicht programmierbaren) Taschenrechnern sind keine Hilfsmittel zugelassen.
5. Das Betreiben von Mobiltelefonen und Computern ist im Prüfungsraum nicht erlaubt.
6. Schreiben Sie bitte gut leserlich und nicht mit Bleistift. Ihre Klausur wird ansonsten nicht gewertet. Lassen Sie einen Korrekturrand von mindestens 4 cm frei.
7. Mit der Unterschrift bestätigen Sie, dass Sie prüfungsfähig sind und zu Beginn der Klausur die vollständigen Unterlagen erhalten haben.

Anmerkung: Maximale Punktzahl= 110 Punkte, 100% = 100 Punkte

(Punkte/Note: 95/1,0; 90/1,3; 85/1,7; 80/2,0; 75/2,3; 70/2,7; 65/3,0; 60/3,3; 55/3,7; 50/4.0)

Aufgabe	1	2	3			
erreichbare Punkte	30	40	40			
erreichte Punkte						

Zusatzleistung: Punkte: Note:

Ort und Datum: Unterschrift:

Aufgabe 1 Multiplexer

Punkte: 30

Folgender VHDL-Quelltext beschreibt einen Multiplexer.

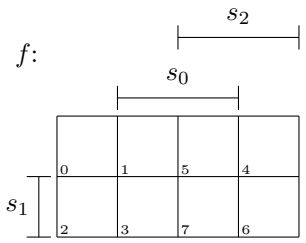
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MUX_6to1 IS
    PORT
    (
        x : IN std_logic_vector(5 DOWNTO 0);
        s : IN std_logic_vector(2 DOWNTO 0);
        f : OUT std_logic
    );
END ENTITY MUX_6to1;

ARCHITECTURE Behavioral OF MUX_6to1 IS
BEGIN
    PROCESS (x,s) is
    BEGIN
        CASE s is
            WHEN "000"   => f <= x(5);
            WHEN "001"   => f <= x(4);
            WHEN "010"   => f <= x(3);
            WHEN "011"   => f <= x(2);
            WHEN "100"   => f <= x(1);
            WHEN "101"|"110"|"111" => f <= x(0);
            WHEN OTHERS => f <= 'U';
        END CASE;
    END PROCESS;
END ARCHITECTURE Behavioral;
```

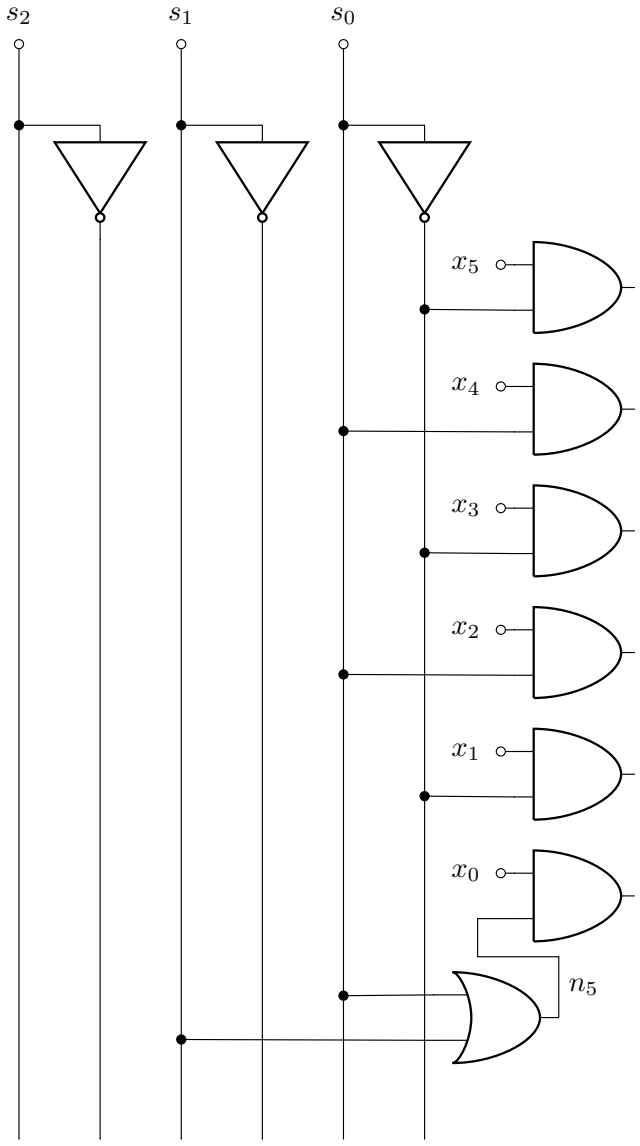
Aufgabenstellung:

a) Entwickeln Sie eine kombinatorische Schaltung mit einer geringsten Anzahl an Gattern. Erlaubt sind nur Grundgatter mit 2 Eingängen des Typs UND bzw. ODER und die Negation. Geben Sie die Boole'sche Funtion an. Hinweis zu Lösung: Klammern Sie geschickt aus! [15 Pkt.]

b) Ergänzen Sie das Schaltbild. [15 Pkt]

i_{10}	s_2	s_1	s_0	f
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	





$\multimap f$

Aufgabe 2 Einfache CRC-Codierung (7/4)

Punkte: **40**

Gegeben ist folgendes Generator-Polynom:

$$G(u) = 1 \cdot u^3 + 1 \cdot u^0$$

Aufgabenstellung:

a) Berechnen Sie alle fehlenden Codewörter in der Tabelle rechts. Tragen Sie ihre Ergebnisse in die gegebene Wahrheitstabelle ein. [10 Pkt.]

b) Entwicklen Sie eine kombinatorische Schaltung zur Erzeugung des Codewortes mit minimaler Gatteranzahl. Erlaubt sind Gatter mit zwei Eingängen (AND, NAND, OR, NOR, XOR, XNOR). [10 Pkt.]

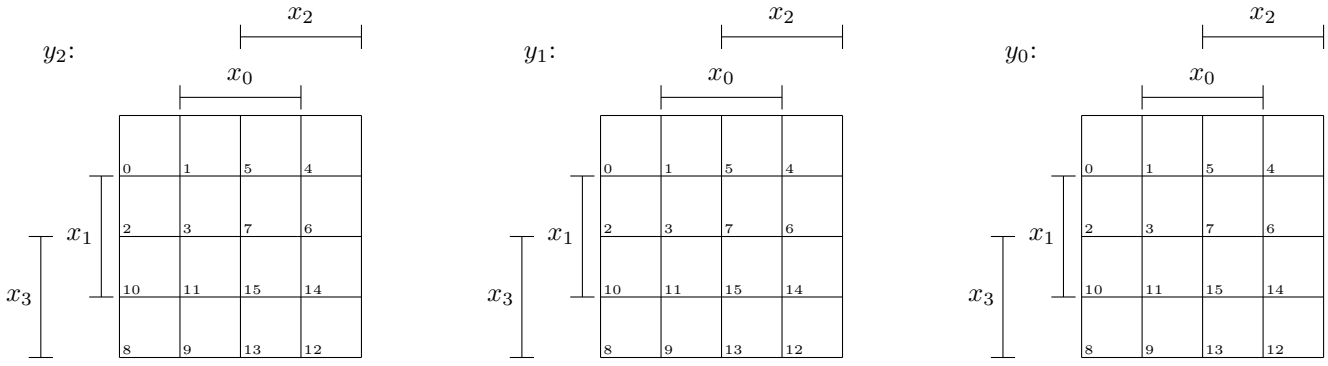
c) Geben Sie das Schaltbild mit Eingangs- und Ausgangsregister (Register-Kombinatorik-Register) an. Nutzen Sie dazu das gegebene Schaltbild. [10 Pkt.]

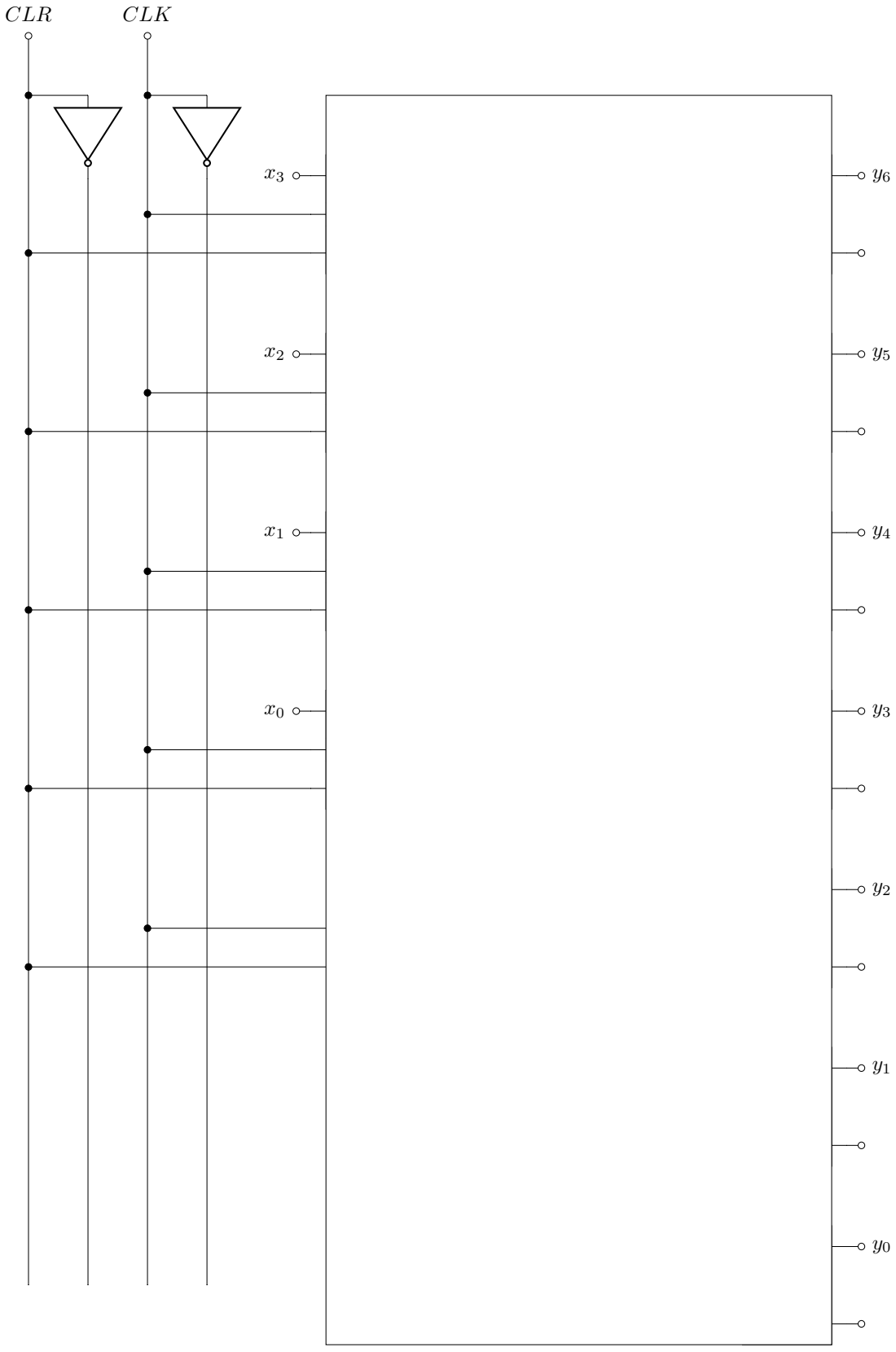
d) Entwicklen Sie eine VHDL-Beschreibung zur Realisierung der Codeworterzeugung. Für die Registerstufen sind Master-Slave D-FF zu verwenden, aktiv mit der steigenden Flanke. Die Register werden mit einem *High*-aktiven RESET-Signal zurückgesetzt. [10 Pkt.]

	n= 7 Codewort						
	m= 4				**k= 3**		
	x_3	x_2	x_1	x_0			
$(i)_{10}$	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1
2	0	0	1	0	0	1	0
3	0	0	1	1			
4	0	1	0	0	1	0	0
5	0	1	0	1	1	0	1
6	0	1	1	0	1	1	0
7	0	1	1	1			
8	1	0	0	0	0	0	1
9	1	0	0	1	0	0	0
10	1	0	1	0			
11	1	0	1	1	0	1	0
12	1	1	0	0	1	0	1
13	1	1	0	1			
14	1	1	1	0	1	1	1
15	1	1	1	1	1	1	0

Hinweis zur Berechnung des Restwertes:

y_6	y_5	y_4	y_3	y_2	y_1	y_0	$\div$	g_3	g_2	g_1	g_0
0	0	1	1	0	0	0	$\div$				





- Lückentext für das Eingangsregister

```
1  LIBRARY ieee;
   USE ieee.std_logic_1164.ALL;
3  USE ieee.std_logic_unsigned.all;
   USE ieee.numeric_std.ALL;
5
   ENTITY PIPO4Bit is
7      port(
           ....................................
9          ....................................
           ....................................
11       );
   end PIPO4Bit;
13
   ARCHITECTURE PIPO4Bit_arc OF PIPO4Bit IS
15 SIGNAL ..............................
   SIGNAL ..............................
17 BEGIN
       PIPO4Bit: PROCESS ........... IS
19     BEGIN
           .........
21         .........
           .........
23         .........
       END PROCESS PIPO4Bit;
25      ............
   END PIPO4Bit_arc;
```

- Lückentext für das Ausgangsregister

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY PIPO7Bit is
    port(
         ..........
         ..........
         ..........
        );
end PIPO7Bit;

ARCHITECTURE PIPO7Bit_arc OF PIPO7Bit IS
SIGNAL ................................
SIGNAL ................................
BEGIN
    PIPO7Bit : PROCESS ......... IS
    BEGIN
         ........................................
         ........................................
         ........................................
         ........................................
    END PROCESS PIPO7Bit;
     ................
END PIPO7Bit_arc;
```

- Lückentext für die Einheit zur CRC-Generierung

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY CRC74 is
    port(
         CLK, CLR  : IN  std_logic;
         x  : IN  std_logic_vector(3 downto 0);
         y  : OUT std_logic_vector(6 downto 0)
        );
end CRC74;

ARCHITECTURE CRC74_struc OF CRC74 IS

COMPONENT ..... IS
    .....
    .....
    .....
END COMPONENT;

COMPONENT ..... IS
    .....
    .....
    .....
END COMPONENT;

SIGNAL .....
SIGNAL .....
SIGNAL .....

BEGIN
   .....
   .....
   .....
   .....
   .....
   .....
 END CRC74_struc;
```

Aufgabe 3 Zustandsautomat

Punkte: 40

Gegeben ist folgender Quelltext eines Zustandsautomaten in einer 3-Prozess-Beschreibung.

a) Skizzieren Sie das Blockschaltbild des Automaten in der 3-Prozess-Beschreibungsform. Erklären Sie textuell die Funktion. Um welchen Typ von Automat handelt es sich (bitte ankreuzen)? [10 Pkt.]

- ☐ Medvedev-Automaten
- ☐ Moore-Automat
- ☐ Mealy-Automat

b) Skizzieren Sie das Zustandsdiagramm des Automaten und erstellen die Zustandsfolgetabelle. [10 Pkt.]

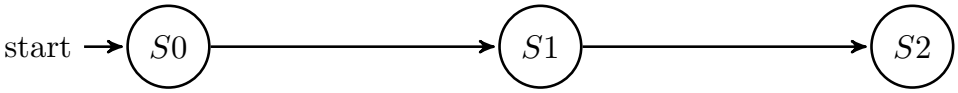
c) Überführen Sie die 3-Prozess-Beschreibung in einen 2-Prozessbeschreibung. Nutzen Sie dafür den Lückentext. [10 Pkt.]

d) Skizzieren Sie das Blockschaltbild des Automaten in der 2-Prozess-Beschreibungsform. [10 Pkt.]

```
1  --- Zustandsautomat ---------------------------------------
   LIBRARY ieee ;
3  USE ieee.std_logic_1164.all;

5  ------------------------------------------------------------
   ENTITY FSM_1 IS
7    PORT( X:          IN std_logic_vector(1 DOWNTO 0);
          Clk:         IN std_logic;
9         RESET:       IN std_logic;
          Y:           OUT std_logic);
11 END FSM_1;

13 ------------------------------------------------------------

15 ARCHITECTURE FSM OF FSM_1 IS
       TYPE    state_type IS (S0, S1, S2);
17     SIGNAL  next_state, current_state: state_type;
   BEGIN
19 ------------------------------------------------------------
   update_state_register: PROCESS(Clk, Reset)  -- Process #1
21   BEGIN
       IF (Reset='1') THEN  current_state <= S0;
23     ELSIF rising_edge(Clk) THEN current_state <= next_state;
       END IF;
25 END PROCESS update_state_register;

27 ------------------------------------------------------------
```

```
30 logic_next_state: PROCESS(X, current_state) -- Process #2
   BEGIN
32       CASE current_state IS
             WHEN S0 =>  IF      X = "01" THEN  next_state <= S1;
34                       ELSE    next_state <= S0;
                         end IF;
36           WHEN S1 =>  IF      X = "11" THEN  next_state <= S2;
                         ELSIF   X = "01" THEN  next_state <= S1;
38                       ELSE  next_state <= S0;
                         END IF;
40           WHEN S2 =>  IF      X = "01" THEN  next_state <= S1;
                         ELSE  next_state <= S0;
42                       END IF;
             WHEN OTHERS => next_state <= S0;
44       END CASE;
END PROCESS logic_next_state;
46

48
logic_out: PROCESS (X, current_state)       -- Process #3
50     BEGIN
           IF ((current_state = S2) and (X = "10")) THEN   Y <= '1';
52         ELSE                  Y <= '0';
           END IF;
54     END PROCESS logic_out;
END FSM;
56
```

$\boxed{\underline{X}/Y}$



i_{10}	**RESET**	**Eingang** $\underline{X}$		**Zustand** $\underline{S}$	**Zustand** $\underline{S^+}$	**Ausgang** $\underline{Y}$
		x_1	x_0			
0	0	0	0	S_0		
1	0	0	1	S_0		
2	0	1	0	S_0		
3	0	1	1	S_0		
4	0	0	0	S_1		
5	0	0	1	S_1		
6	0	1	0	S_1		
7	0	1	1	S_1		
8	0	0	0	S_2		
9	0	0	1	S_2		
10	0	1	0	S_2		
11	0	1	1	S_2		
12	0	0	0	S_3		
13	0	0	1	S_3		
14	0	1	0	S_3		
15	0	1	1	S_3		
16	1	1	1	S_x	S_0	0

```
12
14
ARCHITECTURE FSM OF FSM_1a IS
16      TYPE    state_type IS (S0, S1, S2);
        SIGNAL  next_state, current_state: state_type;
18  BEGIN
20  update_state_register: PROCESS(Clk, Reset)  -- Process #1
      BEGIN
22      IF (Reset='1') THEN  current_state <= S0;
        ELSIF rising_edge(Clk) THEN current_state <= next_state;
24      END IF;
    END PROCESS update_state_register;
26
28  logic_next_state_out: PROCESS(X, current_state) -- Process #2
      BEGIN
30        CASE current_state IS
              WHEN S0 =>  ....................................
32                        ....................................
                          ....................................
34                        ....................................
              WHEN S1 =>  ....................................
36                        ....................................
                          ....................................
38                        ....................................
                          ....................................
40            WHEN S2 =>  ....................................
                          ....................................
42                        ....................................
                          ....................................
44                        ....................................
              WHEN OTHERS => next_state <= S0;
46      END CASE;
    END PROCESS logic_next_state_out;
48  END FSM;
```