1. (3 Punkte) Entscheiden Sie anhand der folgenden Aufgabenstellung, ob eine Softwareimplementierung auf einer Multi-Purpose CPU oder ein dedizierter Hardwareentwurf besser geeignet ist! Begründen Sie ihre Entscheidung!

   Es soll eine komplexe Recheneinheit implementiert werden, die mehrere verschieden Aufgaben erledigen soll. Laufzeit ist weniger wichtig als geringe Entwicklungskosten. Die Recheneinheit wir vermutlich nur sporadisch zum Einsatz kommen.

   Multi Purpose CPU: günstigere schnellere Entwicklung. Hardware kann wenn nicht für Aufgabe verwendet anderweitig genutzt werden.

   + Begründungen

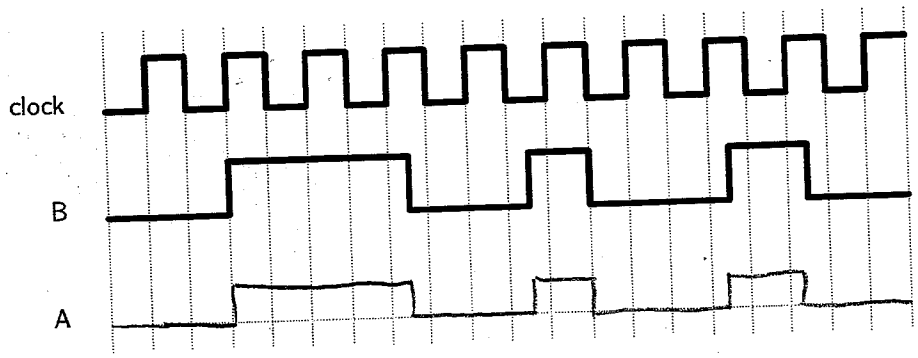
2. (1 Punkt) In VHDL wird das Verhalten einer Einheit in der architecture definiert. ✓

3. (2 Punkte) Zeichnen Sie den zeitlichen Verlauf von Signal A für **architecture beh** in die Zeitleiste ein!

```
architecture beh of test is
begin

  A <= B;

end architecture;
```



✓ in echt gerade :)

4. **wait**

(a) (2 Punkte) Füllen Sie **architecture beh** mit den passenden **wait** Befehlen!

```
architecture beh of test is
  signal input : STD_LOGIC_VECTOR (3 downto 0);
  signal enable : STD_LOGIC;
begin

  process
  begin

    input <= "0001";          -- 20 ns pausieren
    wait for 20ns ;
    input <= "1001";          -- pausieren bis enable low
    wait until enable='0';

  end process;

end architecture;
```

(b) (1 Punkt) Wie werden die eingefügten Befehle synthetisiert?

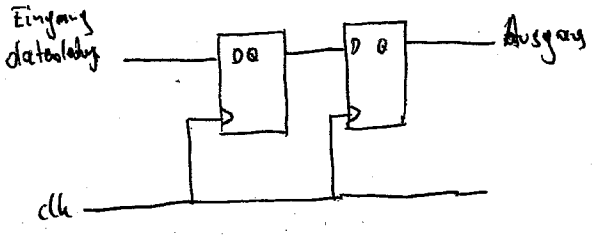
Sie werden nicht synthetisiert. Sie dienen nur zur Simulation

5. Synchronizer

(a) (1 Punkt) Was ist die Aufgabe eines Synchronizers und wo wird er eingesetzt?

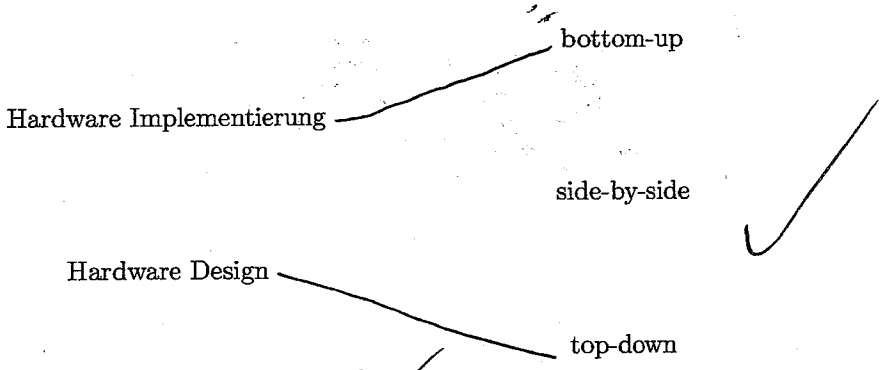
Ein asynchrones Signal soll auf ein clock Signal synchronisiert werden.
Vermeidung von Metastabilität.
Wird z.B. bei einem Button eingesetzt, da dieses natürlich asynchron ist.

(b) (1 Punkt) Zeichnen Sie schematisch den inneren Aufbau eines 2-stage Synchronizers!



6. Design Methoden

(a) (1 Punkt) Verbinden Sie Hardware Design und Implementierung mit der zugehörigen Methode!



(b) (2 Punkte) Erklären Sie die Vorteile der benutzten Methoden für Design und Implementierung!

Bei der Top-down Methode hat der Designer einen guten Überblick, und er kann sich auf das Gesamtbild konzentrieren, bevor er sich in Details verläuft.

Bei der bottom-up Methode werden die hoffentlich gut spezifizierten auf eine bestimmte Funktion hervorgehobenen Teile zuerst umgesetzt. Das sorgt dafür, dass das Spezifizierte möglichst genau umgesetzt wird.

7. (3 Punkte) Erklären Sie den Optimierungsschritt `common subexpression elimination` anhand eines Beispiels!

Expression ①: $A \wedge B \wedge C$ ②: $A \wedge B \wedge D$

gemeinsam ⇒ common subexpression elimination $N = A \wedge B$

① kann dargestellt werden als $N \wedge C$

② als $N \wedge D$

Die Expressions können sich die Gates teilen. Es werden weniger Gates benötigt.

common subexpression elimination erfolgt automatisch, ohne dass ein Eingreifen des Entwicklers

8. (4 Punkte) Implementieren Sie **entity** und **architecture** eines 4-zu-1 Multiplexers! Beschreiben Sie das erwartete Verhalten der benutzten Signale / Input Ports sowie die daraus resultierenden Ausgabewerte.

```
entity mux is
  port (
  E0,E1,E2,E3 : in  std_logic;
    sel       : in  std_logic_vector (1 downto 0);
    ausgang   : out std_logic
  );
end entity mux;

architecture bh of mux is
begin
 asdf: process (all)
  case sel is
    when "00" => ausgang <= E0;
    when "01" => ausgang <= E1;
    when "10" => ausgang <= E2;
    when "11" => ausgang <= E3;
  end case;

 end process;
end architecture;
```

9. (3 Punkte) Überprüfen Sie ob der gegebene Code Syntaxfehler enthält. Benutzen Sie dazu die gegebene Spezifikation.

Spezifikation:

```
[ pure | impure ] function identifier
[ [ parameter ] ( formal_parameter_list ) ] return type_name ;
```

Code:

```
function my_function_name param ( signal X : STD_LOGIC; ) INTEGER;
```

10. Gegeben ist folgende **entity** (alle Signale sind high-aktiv):

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

entity statemachine is
  port
  (
    button_down   : in std_logic;
    button_up     : in std_logic;
    up            : in std_logic;
    down          : in std_logic;
    wind          : in std_logic;
    motor_down    : out std_logic;
    motor_up      : out std_logic
  );
end entity statemachine;
```

Designen Sie eine Moore State Machine die eine Jalousiensteuerung nach folgender Beschreibung implementiert.

Wird `button_down` gedrückt, soll der Motor laufen (`motor_down`), bis der Endanschlag `down` erreicht ist. Ähnlich für `button_up`, `motor_up` und up. Auch während der Motor läuft, soll die Richtung geändert werden können, nicht aber angehalten werden können. Wenn `wind` ist, soll die Jalousie hoch fahren und sich auch nicht mit `button_down` runter fahren lassen.

(a) (4 Punkte) Füllen Sie die Zustandsübergangs- und die Ausgangstabelle:

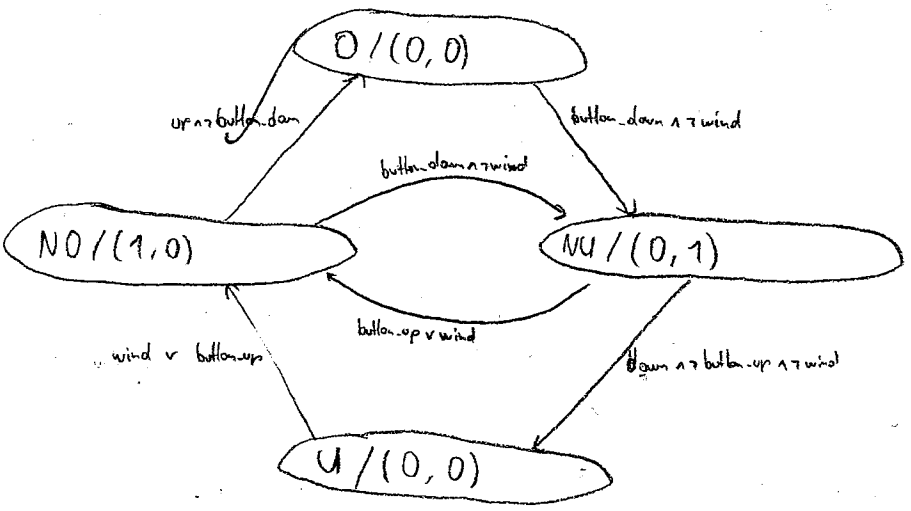
wenn wind wird Motor die abgeschalten

Alter Zustand	Bedingung	Neuer Zustand	
0	button_down∧¬wind	NU	✓
U	button_up∨wind	NO	✓
NU	button_up	NO	✗ wind
NO	button_down∧¬wind	NU	✓
NU	down∧¬button_up	U	✓
NO	up∧¬button_down	0	
	∧wind		

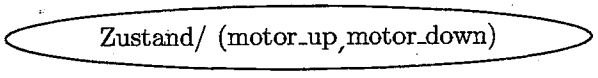
Zustand	motor_up	motor_down	
0	'0'	'0'	✓
U	'0'	'0'	✓
NO	'1'	'0'	✓
NU	'0'	'1'	✓

(b) (2 Punkte) Zeichnen Sie das Zustandsdiagramm inkl. Übergangsbedingungen, Ausgangswerten und Initialzustand (doppelt umrandet):

+ Anfangszustand



Notation:



Bedingung ⟶