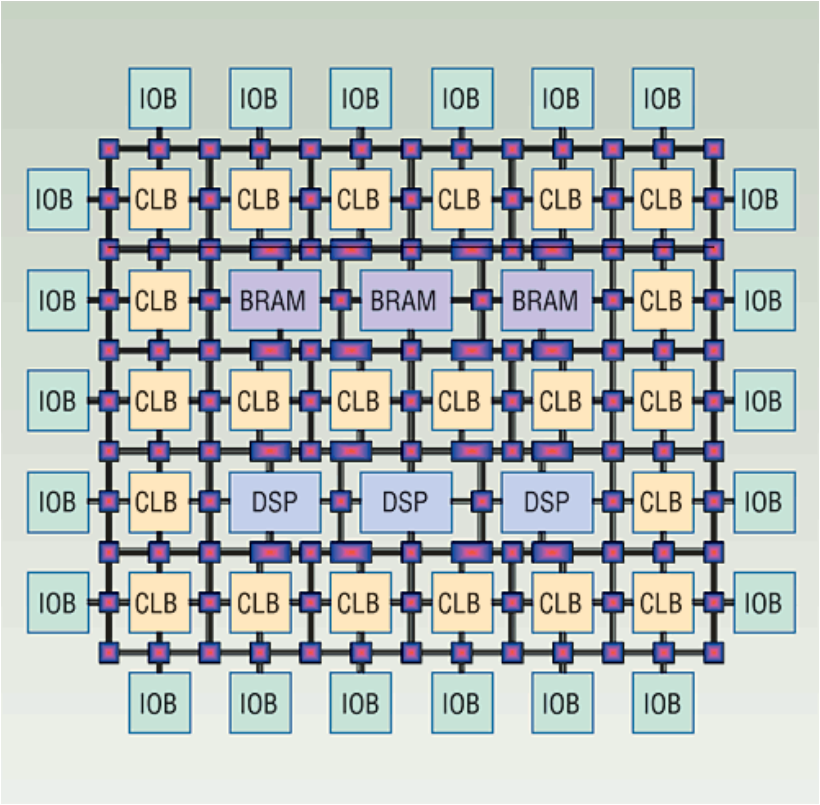
Laboraufgaben - Teil 3



1 Lehrinhalt und Lernziele

Die vorliegende Laborübung(en) ergänzt die Vorlesung Programmierbare Logik. Empfohlene Voraussetzungen für diese Vorlesung und die dazugehörigen Laborübungen sind die Kenntnisse der Informatik, der Rechnerarchitektur, und der Digitaltechnik (und nicht zuletzt Ihr Interesse an der Thematik).

► Aufbau der Laborübung

Nach der Lehrveranstaltung sollten Sie in der Lage sein, einfache digitale Funktionen mit VHDL und/oder Verilog HDL beschreiben zu können und das Verhalten zu simulieren.

Die Laborübungen sind in mehreren Teilaufgaben unterteilt. Des weiteren wird vermittelt, wie einfache Ein- und Ausgabegeräte (hier Schalter und LEDS und Anzeigen) mit dem FPGA Chip verbunden werden.

Nr.	Inhalt	Punkte
L-1	Basiskomponenten	10
L-2	Takte, Zähler und Zeitgeber	10
L-3	Erkennung einer Bit-Sequenz	10
L-4	Booth-Algorithmus mit Datenpfad und Steuerwerk	10

► Laboraufgabe: Bearbeitung, Protokoll, Bewertung, Testat

Zum Ende der Vorlesung: Alle Laborübungen sind fehlerfrei zu bearbeiten und mittels Laborbericht (in Latex) zu dokumentieren. Es müssen alle 4 Laborübungen auf der Hardware in ihrer Funktion und Richtigkeit nachgewiesen werden.

Wenn Sie glauben alle Aufgaben erfolgreich abgeschlossen zu haben, wenden Sie sich an den Übungsleiter und bitten Sie um die Rücksprache. Der Übungsleiter wird Ihre Abgabe dann anhand der folgenden Faktoren bewerten:

- Korrekte Bearbeitung der Aufgaben
- Verständnis des eigenen Codes
- Codingstyle (ist der Code übersichtlich und der Stil koherent?)
- Verständnis der generierten Netzliste
- Korrekte Beantwortung der Rückfragen des Übungsleiters

Ihre Aufgabe und der Laborbericht muss nach der Rücksprache als ZIP-Datei in das entsprechende Abgabefeld / Ordner in Moodle hochgeladen werden. Die erreichten Punkte werden Ihnen **erst nach upload** in Moodle eingetragen.

Hinweis: Zur Umsetzung der Aufgabenstellung lesen Sie bitte das *User Manual* `https://www.terasic.com.tw/attachment/archive/502/DE2_115_User_manual.pdf` durch.

Hinweis: Legen Sie für jede Laborübung ein neues Quartus II Projekt an. Achten Sie darauf, dass der richtige FPGA-Typ als Zielbaustein ausgewählt ist (Cyclone IV EP4CE115F29C8).

2 Aufgabenstellungen L-3

In vielen Bereichen der Datenkommunikation werden sogenannte Datenrahmen übertragen. Den Bit-Mustern an definierten Positionen innerhalb des Datenrahmens können spezielle Funktionen zugeordnet werden. So kann eine definierte Bit-Sequenz den Start eines Datenpaketes oder das Ende einen Datenpaketes bedeuten.

Aufgabe 1 Bit-Sequenzerkennung 10 Pkt.

In dieser Übung soll ein Zustandsautomat entwickelt werden, der das Bit-Muster von vier aufeinander folgenden gleichförmigen Bits erkennt. Der getaktete sequentielle Eingangsvektor $\mathbf{x}$ wird derart überwacht, daß der Ausgang auf $y = 1$ gesetzt wird, wenn für den Eingangsvektor gilt:

- $\mathbf{x} = [0000]$ oder
- $\mathbf{x} = [1111]$

Zudem gilt, daß für eine Sequenz von mehr als vier gleichförmigen Bits der Ausgang auf dem gesetzten Zustand bleibt. Die Abbildung 1 zeigt das Impulsdiagramm für den zu entwickelnden Zustandsautomaten.

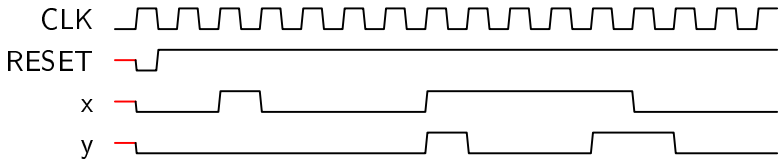


Abbildung 1: Impulsdiagramm des Zustandsautomaten zur Bit-Sequenzerkennung.

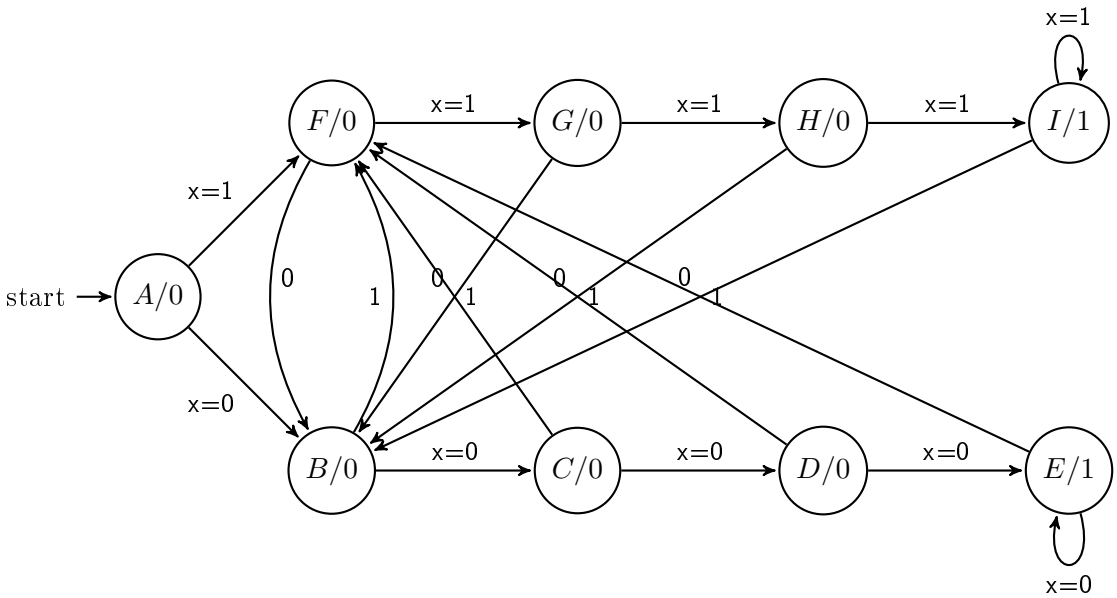


Abbildung 2: Zustandsdiagramm zur Erkennung einer gleichförmigen Bit-Sequenz.

Für die Implementierung des Zustandsautomaten mit dem Zustandsdiagramm nach Abbildung 2 werden neun Flip-Flops benötigt.

► **Aufgabenteil 1** (3 Pkt)

Schreiben Sie eine Statemachine. Die States sollen dabei wie in der folgenden Tabelle codiert werden. Verwenden Sie **nicht** die VHDL übliche typ definition ihrer States. Bauen Sie ihre Statemachine stattdessen klassisch aus Flip-Flops mit kombinatorischen Übergangsfunktionen.

Zustand	**Ausgänge der Flip-Flops**								
S	Q_8	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
A	0	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	1	0
C	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	1	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

1. Legen Sie ein neues Quartus Projekt für die Bit-Sequenzerkennung an. Achten Sie darauf, dass der richtige FPGA-Typ als Zielbaustein ausgewählt ist (Cyclone IV EP4CE115F29C8).
2. Schreiben Sie ein VHDL-Code, der die benötigten 9 D-Flip-Flops instanziiert. Nutzen Sie die Zuweisungen um die Eingänge der Flip-Flops zustandsabhängig zu definieren. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu.
3. Der Kippschalter SW_0 wird als synchroner RESET verwendet (*active low*) für den Zustandsautomaten verwendet. Der Kippschalter SW_1 dient als Eingang für die Variable x. Der Taster KEY_0 wird als Takteingang CLK manuell verwendet. Die grüne LED $LEDG_0$ zeigt den Ausgang y an. Die Ausgänge der Flip-Flops werden mit den roten LEDs $LEDR_8 \ldots LEDR_0$ angezeigt.
4. Fügen Sie den VHDL-Code ihrem Projekt hinzu. Compilieren Sie den Quellcode. Überprüfen Sie die Schaltung auf Gatter-Ebene mit dem *Quartus RTL Viewer*.
5. Simulieren Sie das Verhalten ihrer Schaltung.
6. Laden Sie ihren Entwurf auf das FPGA und überprüfen Sie die Funktion.
7. Verifizieren Sie ihren Entwurf mit dem Technology Viewer bezüglich der Implementierung.

► **Aufgabenteil 2** (3 Pkt)

Bei der Verwendung eines aktiven RESET ist es die Implementierung günstiger, den Ausgangszustand A derart zu kodieren, dass alle Ausgänge der Flip-Flops auf $y_n = 0$ kodiert sind. Für die Implementierung des modifizierten Zustandsautomaten mit dem Zustandsdiagramm nach Abbildung 2 werden neun Flip-Flops benötigt. Es ist somit nun die folgende Wahrheitstabelle gültig.

1. Legen Sie ein neues Quartus Projekt für die modifizierte Bit-Sequenzerkennung an. Achten Sie darauf, dass der richtige FPGA-Typ als Zielbaustein ausgewählt ist (Cyclone IV EP4CE115F29C8).
2. Schreiben Sie ein VHDL-Code, der die benötigten 9 D-Flip-Flops instanziiert. gemäß dem gegebenen VHDL-Code. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu. Hinweis: Mit wenigen Modifikation kann der VHDL-Code aus Aufgabenstellung A verwendet werden.

Zustand	**Ausgänge der Flip-Flops**								
S	z_8	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0
A	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	1	1
C	0	0	0	0	0	0	1	0	1
D	0	0	0	0	0	1	0	0	1
E	0	0	0	0	1	0	0	0	1
F	0	0	0	1	0	0	0	0	1
G	0	0	1	0	0	0	0	0	1
H	0	1	0	0	0	0	0	0	1
I	1	0	0	0	0	0	0	0	1

3. Der Kippschalter SW_0 wird als synchronen RESET verwendet (*active low*) für den Zustandsautomaten verwendet. Der Kippschalter SW_1 dient als Eingang für die Variable x. Der Taster KEY_0 wird als Takteingang CLK manuell verwendet. Die grüne LED $LEDG_0$ zeigt den Ausgang y an. Die Ausgänge der Flip-Flops werden mit den roten LEDs $LEDR_8 \ldots LEDR_0$ angezeigt.

4. Fügen Sie den VHDL-Code ihrem Projekt hinzu. Compilieren Sie den Quellcode. Überprüfen Sie die Schaltung auf Gatter-Ebene mit dem *Quartus RTL Viewer*.

5. Simulieren Sie das Verhalten ihrer Schaltung.

6. Laden Sie ihren Entwurf auf das FPGA und überprüfen Sie die Funktion.

7. Verifizieren Sie ihren Entwurf mit dem Technology Viewer bezüglich der Implementierung.

► **Aufgabenteil 3** (4 Pkt)

Im vorherigen Teil der Laborübung wurde über eine Zuweisung die Funktion des Zustandsautomaten definiert. Effizienter ist die Umsetzung mit der *CASE* Anweisung innerhalb eines *PROCESS*-Blocks. Ein Programmrumpf ist wie folgt gegeben:

```
1  LIBRARY ieee;
   USE ieee.std_logic_1164.all;
3
   ENTITY part2 IS
5  PORT ( .... Definition der Ein- und Ausgaenge
   ....);
7  END part2;

9  ARCHITECTURE Behavior OF part2 IS
   .... Signale definieren
11 TYPE State_type IS (A, B, C, D, E, F, G, H, I);
   SIGNAL z_Q, z_D : State_type; -- z_Q is present state, z_D is next state
13 BEGIN
   ....
15 PROCESS (x, z_Q) -- state table
   BEGIN
17 case z_Q IS
   WHEN A IF (x = '0') THEN z_D <= B;
19 ELSE z_D <= F;
   END IF;
21 .... Zustaende
   END CASE;
23 END PROCESS; -- state table
   PROCESS (Clock) -- state flip-flops
25 BEGIN
   ....
27 END PROCESS;
   .....assignments for output y and the LEDs
29 END Behavior;
```

1. Legen Sie ein neues Quartus Projekt für die alternative Umsetzung der Bit-Sequenzerkennung an. Achten Sie darauf, dass der richtige FPGA-Typ als Zielbaustein ausgewählt ist (Cyclone IV EP4CE115F29C8).

2. Schreiben Sie ein VHDL-Code, der strukturell dem obigen Programmrumpf entspricht. Fügen Sie diesen VHDL-Code ihrem Projekt hinzu.

3. Der Kippschalter SW_0 wird als synchronen RESET verwendet (*active low*) für den Zustandsautomaten verwendet. Der Kippschalter SW_1 dient als Eingang für die Variable x. Der Taster KEY_0 wird als

Takteingang CLK manuell verwendet. Die grüne LED $LEDG_0$ zeigt den Ausgang y an. Die Ausgänge der Flip-Flops werden mit den roten LEDs $LEDR_8 \ldots LEDR_0$ angezeigt.

4. Die LEDs sollen nun den aktuellen Zustand der Statemachine wie in **Aufgabenteil 1** darstellen.
5. Fügen Sie den VHDL-Code ihrem Projekt hinzu. Compilieren Sie den Quellcode. Überprüfen Sie die Schaltung auf Gatter-Ebene mit dem *Quartus RTL Viewer*.
6. Simulieren Sie das Verhalten ihrer Schaltung.
7. Laden Sie ihren Entwurf auf das FPGA und überprüfen Sie die Funktion.
8. Öffnen Sie Ihr Design im RTL-Viewer (Tools → Netlist Viewers → RTL Viewer).
9. Betrachten Sie die **Statemachine** (im RTL-Viewer in gelb dargestellt) per Doppelklick im Statemachine Viewer:
   - Was fällt Ihnen bezüglich des **Encoding** der Statemachine auf? (auswählbar am unteren Bildschirmrand)
10. Verifizieren Sie ihren Entwurf mit dem Technology Viewer bezüglich der Implementierung.

3 Hier ist Schluss!

Quelle: https://www.flaticon.es/