georg.von-der-brueggen [☺] tu-dortmund.de
harun.teper [☺] tu-dortmund.de

Hardwarepraktikum
Wintersemester 21/22

# Übungsblatt 6 – 25 Punkte

**(Block B2 – insgesamt 70 Punkte)**

**Bearbeiten ab Samstag, 27. November 2021.**
**Abgabe bis spätestens Freitag, 3. Dezember 2021, 23:59 Uhr.**

**Hinweis:** In dieser Übung müsst Ihr nicht alle Aufgaben bearbeiten. Bitte sucht Euch zwei Aufgaben aus 6.2, 6.3 und 6.4 aus. Es können also nur insgesamt 25 Punkte gesammelt werden.

**Hinweis:** Wir stellen für diese Übung keine Dateien bereit die zu verändern sind. Überlegt bitte selber, welche Dateien und Komponenten Ihr braucht und wie Ihr die Komponenten strukturieren wollt. Kommentiert dies in den Dateien entsprechend. Dabei können Komponenten die auf den vorherigen Übungszetteln erstellt wurden wiederverwendet werden. Denkt außerdem daran, eine Testbench Datei beizulegen mit der Eure Implementierung getestet werden kann und kommentiert auch diese entsprechend (was wird wie getestet).

## 6.1 Schieberegister in VHDL (5 Punkte)

Ein weiterer essentieller Baustein für die meisten sequenziellen Schaltungen ist das *Schieberegister*, welches für das Lagern und Bewegen von Daten zuständig ist, und deswegen gewöhnlicherweise für den Entwurf von Rechnern genutzt wird. Ein Schieberegister mit n Bits kann aus n Flip-Flops konstruiert werden, die über einen gemeinsamen Takt gesteuert werden. Bei jedem Takt sollen alle Bits des Registers gleichzeitig aktualisiert werden. Typischerweise gibt es vier Modi, in denen Schieberegister eingesetzt werden:

- Serial-in zu Parallel-out (SIPO): In diesem Modus werden die Eingabedaten seriell eingelesen und anschließend parallel ausgelesen.
- Serial-in to Serial-out (SISO): In diesem Modus werden die Einabedaten seriell eingelesen und anschließend seriell ausgelesen.
- Parallel-in to Serial-out (PISO): In diesem Modus werden die Einabedaten parallel eingelesen und anschließend seriell ausgelesen.
- Parallel-in to Parallel-out (PIPO): In diesem Modus werden die Einabedaten parallel eingelesen und anschließend parallel ausgelesen.

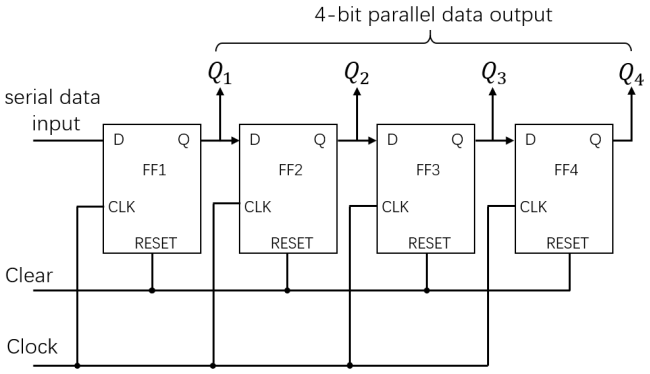


Abbildung 1: SIPO Schieberegister.

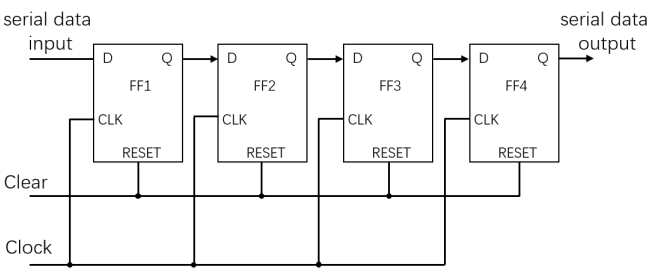


Abbildung 2: SISO Schieberegister.

**Aufgaben:**

a. (2 Punkte) Implementiert ein 8-Bit SIPO-Schieberegister in VHDL.

b. (2 Punkte) Implementiert ein 8-Bit PISO-Schieberegister in VHDL.

c. (1 Punkt) Testet die implementierten Schieberegister mit verschiedenen Eingabedaten.

## 6.2 Bitmustererkennung (10 Punkte)

In dieser Aufgabe soll das das Verhalten eines Roboters implementieren werden, der sich vor einem LED-Bildschirm von links nach rechts bewegt. Auf diesem LED-Bildschirm wird eine Folge von Nullen und Einsen angezeigt. Bei jeder steigenden Taktflanke bewegt sich der Roboter ein Bit nach rechts. Am Kopf des Roboters ist ein Sensor installiert, der die Zahlen (0 und 1) erkennt. Wenn der Roboter in einer beliebig langen Eingabe die Sequenz 1011 erkennt, dann gibt er einen Alarm aus. Der Ausgang Y ist 'TRUE', wenn der Roboter den Alarm auslöst.

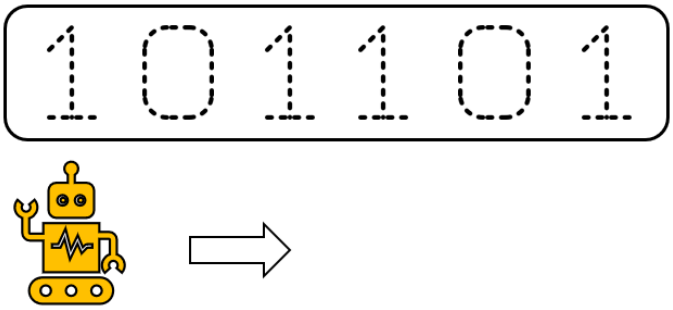


Abbildung 3: Roboter zur Bitmustererkennung.

**Aufgaben:**

a. (2 Punkte) Gebt die formale Beschreibung des Roboterverhaltens als Moore-Automaten und auch als Mealy-Automaten an. Zeichnet danach die beiden Automaten.

b. (1 Punkt) Notiert die Unterschiede und Vor- und Nachteile zwischen den beiden Automatentypen, auch bezogen auf digitale Schaltungen.

c. (4 Punkte) Implementiert das Verhalten des Roboters als Mealy-Automat in VHDL. Testet danach die Implementierung in einer Testbench.

d. (1 Punkt) Modifiziert eure Implementierung so, dass die zu erkennende Bitsequenz (mit Länge 4 Bits) per Parameter an den Automaten übergeben wird. Testet die Implementierung mit einigen Inputsequenzen in einer Testbench. (Es genügt wenn der Parameter einmalig zu Beginn übergeben werden kann.)

e. (2 Punkte) Beschreibt stichpunktartig in eigenen Worten, wie ihr die Implementierung erweitern würdet, wenn der Automat beliebig lange Bitsequenzen erkennen müsste.

**Hinweis:** Die genaue Ausgabe am Ausgang Y ist euch überlassen. Gebt aber bitte an was die Ausgabe bedeutet (z.B., eine '1' für 'TRUE', sonst eine '0'). Der Roboter soll nach einen 'TRUE' weiter fahren können (der Ausgang Y wird dann zunächst wieder auf 'FALSE' gesetzt) und auch wiederholt 'TRUE' ausgeben können.

## 6.3 Synchroner Vorwärts-Rückwärts-Zähler (10 Punkte)

In diesem Versuch soll ein synchroner Vorwärts–Rückwärts-Zähler entworfen werden. Der Zähler soll vier Eingänge und zwei Ausgänge haben:

- Einen Reseteingang (Reset).
- Einen Takteingang (Takt).
- Einen Steuereingang (Count).
- Einen Steuereingang (Down).
- Zwei Ausgänge (Q1 und Q0, Q1 ist der höherwertige).

Die Schaltung soll mit jeder positiven Taktflanke zählen und den Zählerstand als Binärzahl an den Ausgängen Q1 und Q0 ausgeben. Im Grundzustand, d.h. vor dem Einschalten des Taktes, soll die Schaltung sich im Zählerstand Q1Q0 = 00 befinden.

- Wenn am Steuereingang Count = 0 anliegt, soll die Schaltung nicht zählen.
- Wenn am Steuereingang Count = 1 anliegt, soll die Schaltung zählen.
- Wenn am Steuereingang Down = 0 anliegt, soll die Schaltung vorwärts zählen (Count=1).
- Wenn am Steuereingang Down = 1 anliegt, soll die Schaltung rückwärts zählen (Count=1).

Die Schaltung soll mit zwei JK-Flipflops realisiert werden. Die miteinander verbundenen Takteingänge der beiden Flipflops bilden den Takteingang der gesamten Schaltung.

Die Zustände des zu entwerfenden Automaten sind identisch mit den Ausgangszuständen, d.h. die Ausgabefunktion ist die identische Abbildung. [0,0], [0,1], [1,0], und [1,1] sind die vier möglichen Zählerzustände des Automaten. Es gilt: [0,0] = S0, [0,1] = S1, [1,0] = S2, [1,1] = S3.

**Hinweis:** Ihr könnt wahlweise einen Zähler mit Überlauf (wenn von 11 hochgezählt wird ist das Ergebnis 00) oder einen Zähler ohne Überlauf (wenn von 11 hochgezählt wird bleibt es bei 11) entwerfen. Gebt dies bitte in den entsprechenden Diagrammen und Dateien an.

**Aufgaben:**

a. (2 Punkte) Zeichnet den entsprechenden Mealy-Automaten auf, und gebt die formale Beschreibung an.

b. (1 Punkt) Beschreibt die Unterschiede zwischen einem JK-Flipflop und einem D-Flipflop.

c. (5 Punkte) Implementiert den Zähler in VHDL. Nutzt dazu zwei JK-Flipflops. Testet eure Implementierung der Komponenten und des gesamten Zählers in einer Testbench (hier kann der Reset ignoriert werden).

d. (2 Punkte) Ergänzt die Schaltung durch einen synchronen Reset (Schalter mit R), der es ermöglicht, den Grundzustand Q1 = Q2 = 0 jederzeit einzustellen und testet diesen.

## 6.4 Ampelsteuerung (10 Punkte)

Es soll eine Steuerung für eine Kreuzung mit vier Ampeln entworfen werden (siehe Abbildung 4). Ampel 1 und Ampel 3 sind synchronisiert, um den Verkehrsfluss in Nord- und Südrichtung zu steuern, während Ampel 2 und Ampel 4 synchronisiert sind, um den Verkehrsfluss in Ost- und Westrichtung zu steuern. Wenn die Ampel rot ist, muss das Fahrzeug anhalten. Wenn die Ampel grün ist, dürfen Fahrzeuge durchfahren. Gelb ist der Übergangszustand zwischen Rot und Grün. Abbildung 4 zeigt den Ablauf und alle Zustände dieser Ampeln.

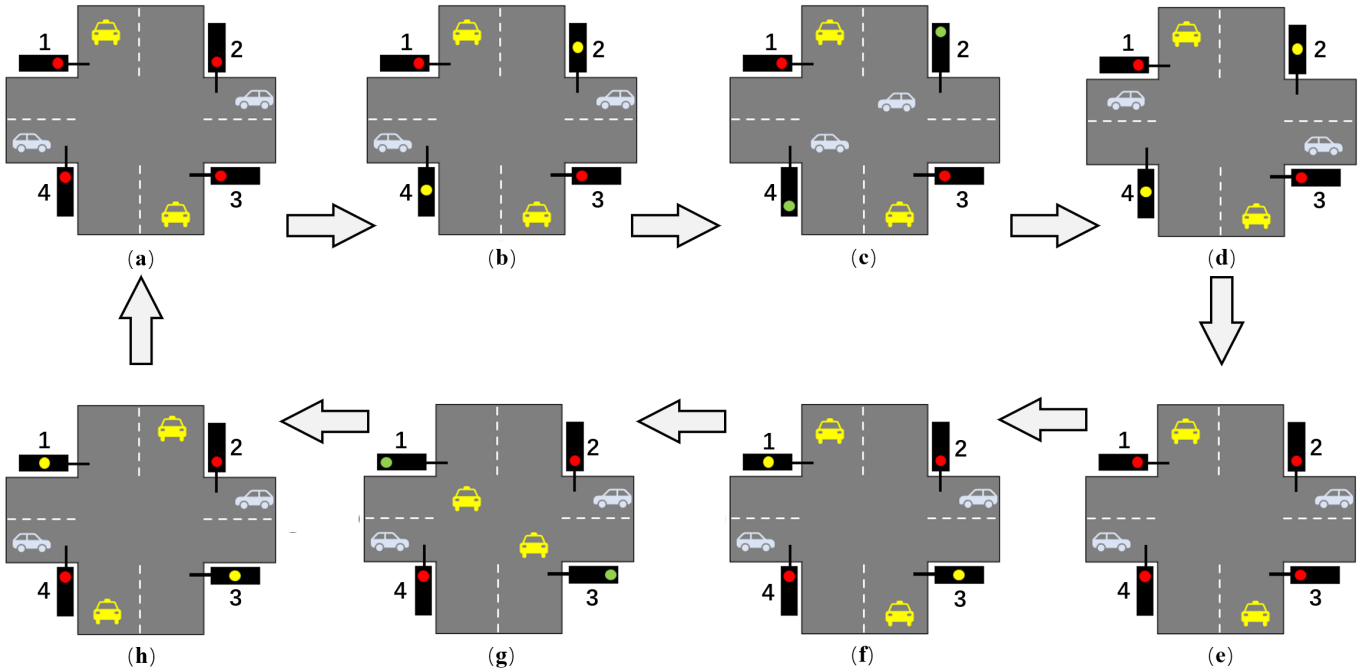


Abbildung 4: Alle Zustände und Zustandsübergänge bei der Ampelsteuerung.

Die Beschreibung der Ampelsteuerung:

- Am Anfang werden alle Ampeln auf Rot initialisiert und alle Autos müssen anhalten (siehe Abbildung 4(a)).
- Nach 10 Sekunden schalten die Ampeln 2 und 4 für 5 Sekunden auf Gelb um, wobei alle Autos noch stehen bleiben und warten müssen (siehe Abbildung 4(b)).
- Danach schalten die Ampeln 2 und 4 für 30 Sekunden auf Grün um, damit alle Autos auf der Ost- und Westrichtung durchfahren können (siehe Abbildung 4(c)).
- Dann schalten die Ampeln 2 und 4 wieder für 5 Sekunden auf Gelb um. Alle Fahrzeuge müssen stehen bleiben (siehe Abbildung 4(d)).
- Danach schalten die Ampeln 2 und 4 wieder für 10 Sekunden auf Rot und alle Autos müssen stoppen (siehe Abbildung 4(e)).
- Dann schalten die Ampeln 1 und 3 für 5 Sekunden auf Gelb um, und alle Autos müssen anhalten (siehe Abbildung 4(f)).
- Danach schalten die Ampel 1 und Ampel 3 für 30 Sekunden auf Grün um, und alle Autos auf der Nord- und Südrichtung können während dieser 30 Sekunden ungehindert durchfahren (siehe Abbildung 4(g)).
- Dann schalten die Ampeln 1 und Ampel 3 wieder für 5 Sekunden auf Gelb um und alle Fahrzeuge müssen stehen bleiben (siehe Abbildung 4(h)).
- Schließlich schalten die Ampel 1 und 3 wieder auf Rot, und alle Ampeln sind wieder im Originalzustand (siehe Abbildung 4(a)). Dieser bleibt ebenfalls für 10 Sekunden erhalten.

**Aufgaben:**

a. (2 Punkte) Gebt die formale Beschreibung der Ampelsteuerung als Moore-Automaten an und zeichnet diesen.

b. (6 Punkte) Implementiert die Ampelsteuerung in VHDL.

c. (2 Punkte) Testet eure Implementierung in einer Testbench.