Digitaltechnik
Wintersemester 2023/2024
Hausaufgabe 07+08

TECHNISCHE UNIVERSITÄT DARMSTADT

ENCRYPTO CRYPTOGRAPHY AND PRIVACY ENGINEERING

M.Sc. Andreas Brüggemann, M.Sc. Nora Khayata, M.Sc. Daniel Günther Abgabefrist: 22.12.23 23:59

Allgemeine Hinweise

- Es gibt insgesamt 6 Hausaufgabenblätter mit je maximal 20 Punkten. Zum Erwerb der Studienleistung benötigen Sie (neben 50% der Quiz-Punkte) **mindestens 60 Punkte über alle Hausaufgabenblätter hinweg**.
- Die Hausaufgabenblätter sind in **Gruppen von drei Studierenden** zu bearbeiten und abzugeben. Schreiben Sie Namen und Matrikelnummer aller Mitglieder Ihrer Abgabegruppe in Ihre Abgabe. Ein Mitglied muss die Abgabe dann **als einzelne PDF-Datei** im Moodle hochladen und die anderen Mitglieder über die entsprechende Moodle-Funktion als Ko-Autor:innen angeben.
- Die Abgabe dieses Blatts in Moodle muss bis spätestens **22.12.23 23:59** erfolgen.
- Bewertet wird, falls nicht explizit anders gesagt, insbesondere der Lösungsweg, nicht nur das Ergebnis. Geben Sie alle nötigen Zwischenschritte an.
- Allgemein dürfen Sie in diesem Blatt einen Taschenrechner für Operationen wie $+, -, \cdot, /, \log$ etc. im Dezimalsystem verwenden. Für die Klausur werden die Zahlen so gewählt, dass sie ohne Hilfe eines Taschenrechners berechnet werden können. Es ist jedoch sehr hilfreich, die ersten zehn Zweierpotenzen (bis 2^{10}) zu beherrschen.

Hinweise zur Nutzung von SystemVerilog

- Für sämtliche SystemVerilog-Aufgaben finden Sie im Moodle entsprechende Code-Gerüste, Tests etc.
- Insofern entsprechende Code-Gerüste gegeben sind, ändern und schreiben Sie **keinen** Code außerhalb der dafür vorgesehenen und markierten Stellen.
- Nutzen Sie außerhalb von Testbenches ausschließlich synthetisierbaren Code, d.h., nutzen Sie keine Elemente von SystemVerilog, welche nur für Tests vorgesehen sind (z.B. `initial` Blöcke).
- Häufig geben wir für die Aufgaben auch entsprechende Testbenches an. Diese erkennen Sie am "`_tb`" am Ende des Dateinamens. Nutzen Sie diese Testbenches, um Ihren Code auf Fehler zu überprüfen.
- **Testen unter Windows:** `./sim.bat [modulname]` **Testen sonst:** `./sim.sh [modulname]`
- Falls unzureichende Rechte für `./sim.sh`: Folgendes Kommando eingeben: `chmod +x sim.sh`
- Warnungen von der Form `VCD warning: $dumpvars: ...` können Sie ignorieren.
- Unsere Testbenches testen Ihren Code nicht für alle möglichen Fälle. Sie liefern also einen Indikator für die Qualität Ihres Codes, aber keine Garantie, dass dieser fehlerfrei ist und entsprechend bewertet wird. Wir behalten uns vor, zusätzliche nicht veröffentlichte Testbenches zu nutzen, um Ihren Code zu testen.
- Kommentieren Sie Ihren Code ausführlich in deutsch oder englisch. **Unkommentierter Code wird nicht näher bewertet, wenn unsere Tests fehlschlagen.**
- Laden Sie Ihren SystemVerilog-Code (alle geänderten `.sv`-Dateien) in einem **einzelnen ZIP Archiv** zusammen mit der einzelnen PDF-Datei mit Ihren Lösungen zu anderen Aufgaben im Moodle als Abgabe hoch. Es wird nur Code in den entsprechend dafür vorgesehenen Dateien bewertet. Anderer Code, zum Beispiel wenn in der PDF-Datei eingebunden, wird nicht korrigiert und mit 0 Punkten bewertet.
- Im Moodle finden Sie eine SystemVerilog Anleitung zur Installation und Nutzung der notwendigen Tools.

Hausaufgabe 1 Realisierung neuer Speicherelemente [9 Punkte]

In dieser Aufgabe geht es darum, neue Latches und Flip-Flops aus bestehenden Speicherelementen aufzubauen.

Hausaufgabe 1.1 Rücksetzbarer Flip-Flops mit Taktfreigabe [2 Punkte]

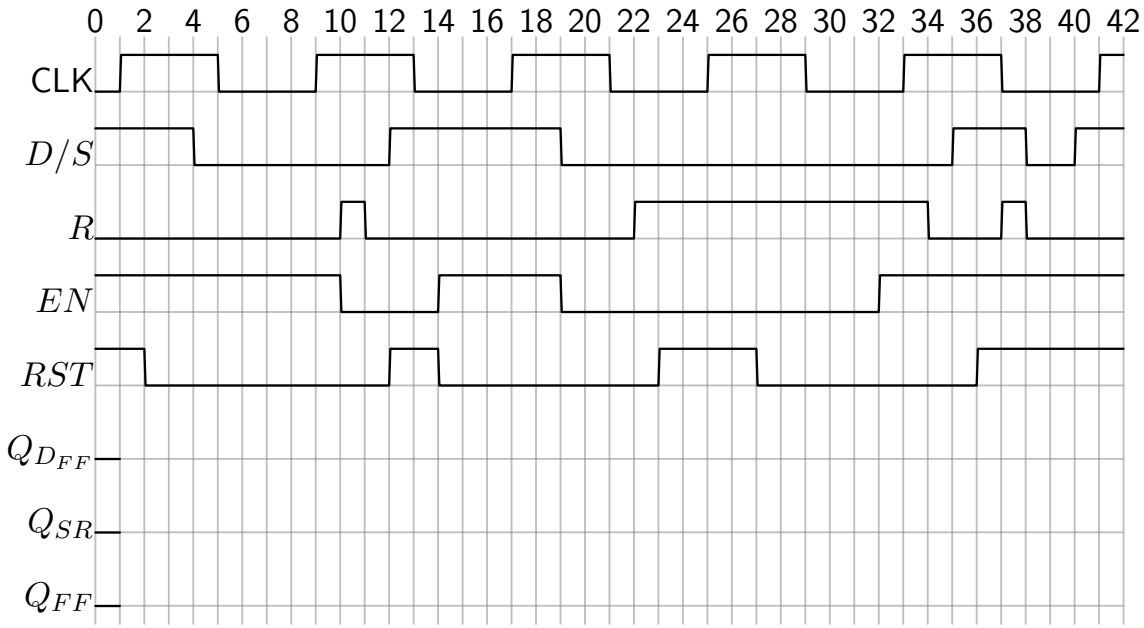
Realisieren Sie einen Flip-Flop mit den Eingängen CLK, D, EN und RST sowie den Ausgängen Q und Q' der sich bei positiven Taktflanken von CLK wie folgt verhält:

- RST = 0 und EN=0: Ausgang Q wird gehalten.
- RST = 0 und EN=1: Eingang D wird gespeichert und an Ausgang Q ausgegeben.
- RST = 1 und EN=0: Ausgang Q wird gehalten.
- RST = 1 und EN=1: Flip-Flop wird *synchron* zurückgesetzt. Der Wert '0' wird gespeichert und an Ausgang Q ausgegeben.

Verwenden Sie dazu einen D-Flip-Flop sowie beliebige kombinatorische Gatter.

Hausaufgabe 1.2 Zeitverhalten der Speicherelemente [3 Punkte]

Ergänzen Sie im folgenden Timing-Diagramm das Schaltverhalten eines D-Flip-Flops mit Ausgang $Q_{D_{FF}}$, eines SR-Latches mit Ausgang Q_{SR} und des rücksetzbaren Flip-Flops mit Taktfreigabe mit Ausgang Q_{FF} aus Übung 1.1.



Hausaufgabe 1.3 WG-Latch mittels SR-Latch realisieren [4 Punkte]

Ein neuer Latch-Typ namens "WG-Latch" soll mittels eines SR-Latches und Basisgattern (NOT, AND, OR, auch AND3, OR3, ...) realisiert werden. Dabei soll der Ausgang Q des WG-Latch der Ausgang Q des verbauten SR-Latches sein. Die Interpretation der freien Eingänge W und G ist durch folgende Zustandsübergangstabelle gegeben.

Hausaufgabe 1.3.1 [1 Punkt]

Vervollständigen Sie die Zustandsübergangstabelle mit den Werten für S und R, die gesetzt werden müssen, um den entsprechenden Ausgang Q zu erreichen.

W	G	Q_{prev}	Q	S	R	Interpretation
0	0	0	1			Zustand auf 1 setzen
		1	1			
0	1	0	1			Zustand invertieren
		1	0			
1	0	0	0			Zustand halten
		1	1			
1	1	0	0			Zustand auf 0 setzen
		1	0			

Hausaufgabe 1.3.2 [1 Punkt]

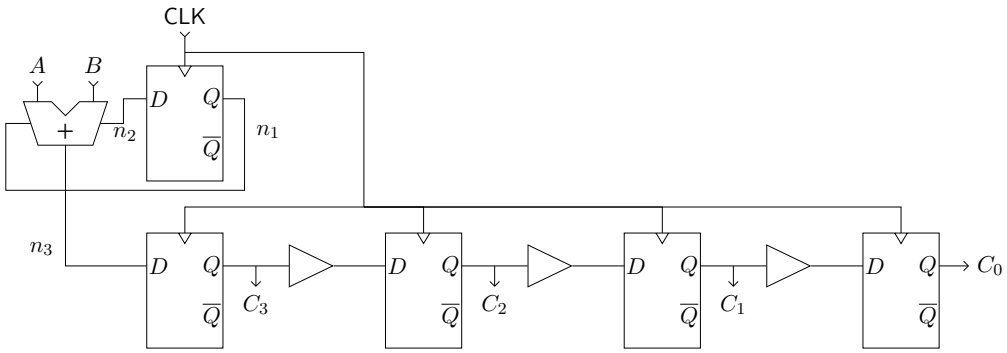
Geben Sie die resultierenden Funktionen für S und R als Summe von Produkten an.

Hausaufgabe 1.3.3 [2 Punkte]

Geben Sie das Schaltbild des WG-Latches an. Nutzen Sie dafür ein SR-Latch und Basisgatter.

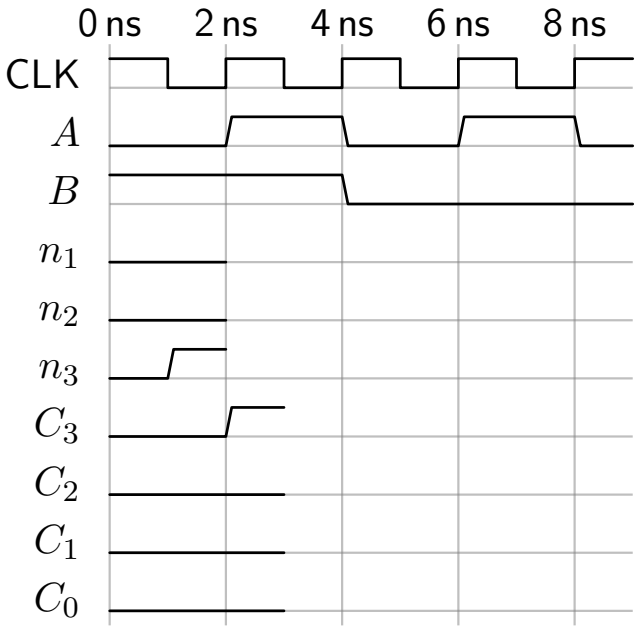
Hausaufgabe 2 Serielle Addition [3 Punkte]

Folgende Schaltung ist in der Lage, mit nur einem Volladdierer 4 bit Zahlen zu addieren:



In dieser Aufgabe möchten wir die Zahlen 1010_2 und 0011_2 addieren. Dazu geben wir diese Bit für Bit *umgekehrt seriell* ein, d.h., für z.B. 1010_2 geben wir im ersten Takt 0 ein, dann 1, dann 0, und dann wieder 1. Am Ende lesen wir das Ergebnis $C = C_3C_2C_1C_0$ ab.

Vervollständigen Sie das nachfolgende Timing-Diagramm. Nehmen Sie dabei an, dass $t_{\text{cd,FA}} = t_{\text{pd,FA}} = t_{\text{cd,BUF}} = t_{\text{pd,BUF}} = 1\,\text{ns}$ gilt, wobei FA der Volladdierer und BUF ein Buffer ist.



Welchen Binärwert hat am Ende (nach 9 ns) $C = C_3C_2C_1C_0$? Vergleichen Sie diesen zur Probe mit $1010_2 + 0011_2$.

Hausaufgabe 3 Lichtschranke [3 Punkte]

Entwerfen Sie eine digitale Schaltung für die Kontrolleinheit einer Lichtschranke. Sie erhalten den Eingang `Sensor`, der direkt an den Sensor angeschlossen ist. Der Sensor gibt eine logische 1 aus, wenn die Lichtschranke unterbrochen ist und eine 0, wenn dies nicht der Fall ist. Als weiteren Eingang dürfen Sie ein Taktsignal `CLK` nutzen. Der einzige Ausgang heißt `O`.

Zweck der Kontrolleinheit ist es, kurze Störungen des Sensors herauszufiltern und nur tatsächliche Unterbrechungen der Lichtschranke zu erkennen. Dazu soll `O` nur genau dann auf 1 gesetzt werden, wenn die Lichtschranke für mindestens 3 Takte am Stück unterbrochen ist (`Sensor` = 1) und direkt danach für genau 3 Take nicht unterbrochen ist (`Sensor` = 0). Es ist nicht wichtig, wie lange `O` auf 1 gesetzt bleibt, wenn eine Unterbrechung der Lichtschranke erkannt wurde.
Hinweis: Nutzen Sie für Ihre Lösung eine Kette von D-Flip-Flops. Sie dürfen annehmen, dass alle D-Flip-Flops am Anfang eine 0 gespeichert haben.

Hausaufgabe 4 Displaysteuerung [2 Punkte]

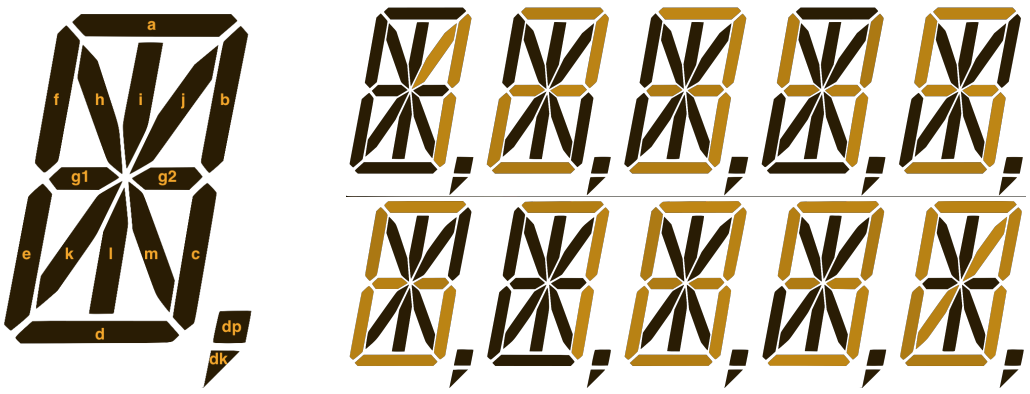


Figure 1: Schema einer 16-Segmentanzeige.

In dieser Aufgabe soll die Hardwareschnittstelle `display` für die Ansteuerung einer 16-Segmentanzeige implementiert werden, auf welcher wir Ziffern anzeigen wollen. Die Funktionsweise der 16-Segmentanzeige ist in Abbildung 1 illustriert. Für die Implementierung ist folgendes zu beachten:

- Am Eingang `DIGIT` liegt die anzuzeigende Ziffer an.
- Der 16-Bit Ausgang `SEGMENT` setzt diejenigen Segmente auf 1, die zur Anzeige der an `DIGIT` anliegenden Ziffer benötigt werden. Dabei sollen die Belegungen der einzelnen Segmente in der folgenden Reihenfolge in `SEGMENT` geschrieben werden: a, b, c, d, e, f, g1, g2, h, i, j, k, l, m, dp, dk.
- Wenn am Eingang `DIGIT` keine gültige Ziffer anliegt, soll ein großes X ausgegeben werden.

Hinweis: Sie können Ihren Code mit folgendem Kommando kompilieren und testen:
Windows: `./sim.bat display`, Sonst: `./sim.sh display`

display.sv

```
1 `timescale 1ns / 1ns
2
3 module display
4
```

```
    ( input  logic  [3: 0]  DIGIT,    // Binary encoding of the digit to display

      output logic  [15:0]  SEGMENT); // Encoding of all segments to enable

/* ===================================== INSERT CODE HERE ===================================== */



/* ============================================================================================== */

endmodule
```

Hausaufgabe 5 4-Bit-Multiplizierer [3+3* Punkte]

In dieser Aufgabe implementieren Sie einen kombinatorischen 4-Bit-Multiplizierer in SystemVerilog. Machen Sie sich dazu zunächst klar, wie die schriftliche Multiplikation zweier Zahlen im Dezimalsystem durchgeführt wird und wie sich dies auf das Binärsystem übertragen lässt. Die gesamte Aufgabe nutzt vorzeichenlose (unsigned) Zahlen.

In dieser Aufgabe ist es nicht erlaubt, arithmetische Operationen und Shifts wie z.B. `<<`, `*`, `+` zu nutzen, diese müssen stattdessen selbstständig implementiert werden falls notwendig. Weiterhin sind keine `always_comb`-Blöcke und kein ternärer Operator (`.?.:.`) erlaubt.

Hausaufgabe 5.1 1-Bit-Multiplexer [1 Punkt]

Implementieren Sie zunächst das SystemVerilog Modul `multiplexer1bit` Der Multiplexer erhält zwei 4 Bit breite Eingabewerte `m0` und `m1` und ein 1 Bit breites Steuersignal `s`. Ist der Input `s` = 0, so soll Eingabewert `m0` an den Ausgang angelegt werden, ist der Input `s` = 1, so soll Eingabewert `m1` an den Ausgang angelegt werden.
Hinweis: Diese und die nächste Teilaufgabe teilen sich eine Testbench. Kommandos zum Testen: Windows: `./sim.bat multiplexer`, Sonst: `./sim.sh multiplexer`

multiplexer.sv

```
module multiplexer1bit
            ( input  logic [7:0] m0, m1,  // data inputs
              input  logic       s,       // selection input
              output logic [7:0] out);    // output

/* ===================================== INSERT CODE HERE ===================================== */



/* ============================================================================================== */

endmodule
```

Hausaufgabe 5.2 2-Bit-Multiplexer [1 Punkt]

Implementieren Sie als nächstes das SystemVerilog Modul `multiplexer2bit`. Im Vergleich zum 1-Bit-Multiplexer erhält der 2-Bit-Multiplexer vier 4 Bit breite Eingabewerte `m0`, `m1`, `m2` und `m3` und ein 2 Bit breites Steuersignal `s`. Bei `s` = 2 soll zum Beispiel `m2` ausgegeben werden.
Hinweis: Sie können den zuvor implementierten 1 Bit Multiplexer nutzen.

multiplexer.sv

```
module multiplexer2bit
            ( input  logic [7:0] m0, m1, m2, m3,  // data inputs
              input  logic [1:0] s,               // selection inputs
              output logic [7:0] out);            // output

/* ===================================== INSERT CODE HERE ===================================== */



/* ============================================================================================== */

endmodule
```

Hausaufgabe 5.3 Shifter [1 Punkt]

Beim schriftlichen Multiplizieren müssen manche Zahlen um eine oder mehrere Stellen verschoben werden. Ein Shifter stellt diese Funktionalität bereit. Implementieren Sie das SystemVerilog Modul `shifter`. Der Shifter erhält eine 4 Bit Zahl `in` und eine Steuergröße `s` als Eingabe und gibt die um `s` Stellen arithmetisch nach links verschobene Zahl aus: `in <<< s`.
Hinweis: Nutzen Sie den 2 Bit Multiplexer für eine Fallunterscheidung, um wie viele Stellen die Zahl verschoben werden soll. Weiterhin dürfen Sie Konkatenation nutzen.
Hinweis: Denken Sie beim Testen daran, auch die Datei des Multiplexers einzubinden. Automatisch funktioniert das mit folgendem Kommando zum Kompilieren und Testen:
Windows: `./sim.bat shifter`, Sonst: `./sim.sh shifter`

shifter.sv

```
`timescale 1ns / 1ns

module shifter
          ( input   logic [3:0] in,   // input
            input   logic [1:0] s,    // shift
            output  logic [7:0] out); // set to input <<< shift

/* ===================================== INSERT CODE HERE ===================================== */



/* ============================================================================================== */

endmodule
```

Hausaufgabe 5.4 Multiplizierer [3* Punkte]

Bonusaufgabe: Diese Aufgabe ist optional und kann bis zu drei zusätzliche Punkte geben, um einen fehlenden Punkt zur Studienleistung auszugleichen.

Implementieren Sie unter Zuhilfenahme der zuvor implementierten SystemVerilog Module den Multiplizierer im Modul `multiplier`.
Hinweis: Die entsprechende Datei enthält auch ein Modul für die Addition von zwei Zahlen, welches Sie nutzen können.
Hinweis: Denken Sie beim Testen daran, alle notwendigen Dateien aus Voraufgaben einzubinden. Automatisch funktioniert das mit folgendem Kommando zum Kompilieren und Testen:
Windows: `./sim.bat multiplier`, Sonst: `./sim.sh multiplier`

multiplier.sv

```
`timescale 1ns / 1ns

module adder // you can just use this
        ( input   logic [7:0] a, b,
          output  logic [7:0] sum);
  assign sum = a + b;
endmodule

module multiplier
          ( input logic [3:0] a, b,
            output logic [7:0] product);

/* ===================================== INSERT CODE HERE ===================================== */



/* ============================================================================================== */

endmodule
```

Viel Spaß beim Bearbeiten! ☺