SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

Systemarchitektur SS 2021

Präsenzblatt 6 (Lösungsvorschläge)

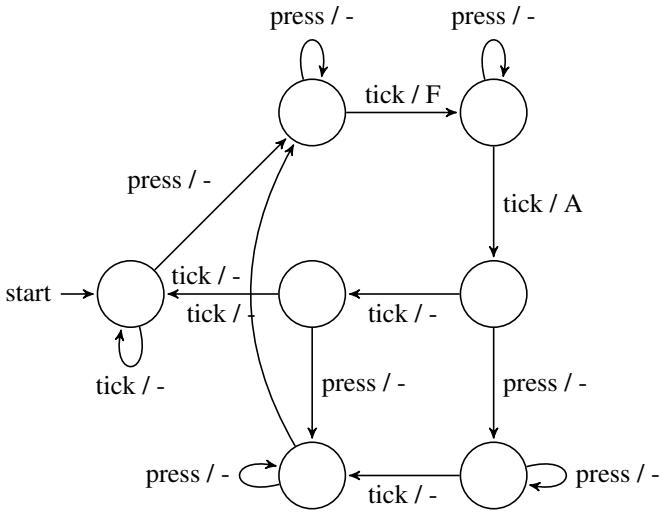
Hinweis: **Dieses Aufgabenblatt wurde von Tutoren erstellt. Die Aufgaben sind für die Klausur weder relevant noch irrelevant, die Lösungsvorschlage weder korrekt noch inkorrekt.**

Aufgabe 6.1: Endliche Automaten

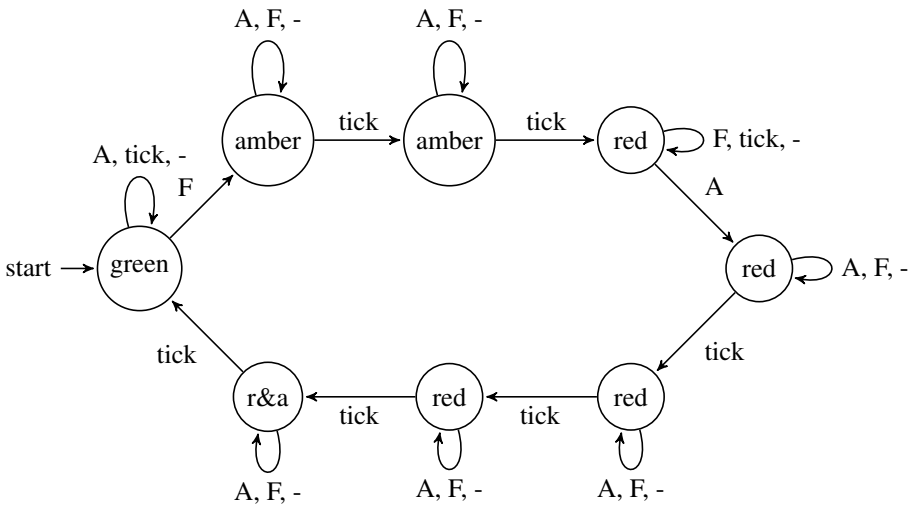
1. Konstruieren Sie einen Mealy-Automaten, der die grobe Steuerung einer einfachen Fußgänger-Ampel implementiert und den folgenden Spezifikationen genügt:
   - **Ausgaben**: A (Autos) und F (Fußgänger), je nachdem, wer gerade fahren bzw. gehen soll.
   - **Eingaben**: $press$ (ein Fußgänger drückt und möchte die Straße überqueren) und ein Signal $tick$ (alle 20 Sekunden von einer eingebauten Uhr).
   - **Verhalten**: Solange kein Fußgänger drückt, zeigt die Ampel Grün für die Autos. Drückt ein Fußgänger, so hat er ab dem nächsten $tick$ der eingebauten Uhr genau 20 Sekunden lang Grün. Im Anschluss daran müssen mindestens 60 Sekunden vergehen, ehe die Ampel erneut für Fußgänger auf Grün schalten kann.
2. Eine Ampel zeigt natürlich nicht durch ein F oder A an, wer gerade passieren darf.
   Konstruieren Sie also einen Moore-Automaten, der das Verhalten der Anlage **für Autos** realisiert:
   - Der Automat erhält die Eingaben F und A, sowie alle 2 Sekunden ein Signal $tick$ von der bereits für die grobe Steuerung verwendeten Uhr.
   - Es gibt 4 Ausgaben: Grün, Gelb, Rot, sowie Gelb und Rot gleichzeitig.
   - Die Ampel schaltet von Grün für 4 Sekunden auf Gelb, ehe sie zu Rot umspringt.
   - Von Rot ausgehend wartet die Ampel zunächst 6 Sekunden, schaltet dann für 2 Sekunden auf Rot und Gelb, ehe sie auf Grün umspringt.

Lösungsvorschlag: Wir haben der Übersichtlichkeit halber auf die Benennung der Zustände verzichtet. Sie dürfen den Zuständen natürlich noch tolle Namen verpassen.

1. Mealy-Automat mit 7 Zuständen (Bei den Transitionen steht zuerst die Eingabe, dann '/', dann die Ausgabe):



2. Die Ausgabe steht jeweils im Zustand. Wir nehmen an, dass die beiden Automaten durch die Uhr synchronisiert werden, also dass das Signal F bzw. A nur zu einem $tick$ erfolgt.



Aufgabe 6.2: Schaltungen und Verilog

Digitale Signal Prozessoren (DSP) sind spezielle Prozessoren zur Verarbeitung von Bild-, Ton- und Funksignalen. Eine häufige Aufgabe für DSPs ist es, die Summe über mehrere Multiplikationen zu berechnen. Um dies effizient tun zu können, bieten ihre Befehlssätze häufig einen speziellen Multiply and Acumulate (MLA) Befehl an. Dieser nimmt drei Zahlen als Eingabe und berechnet $(a \cdot b) + c$.

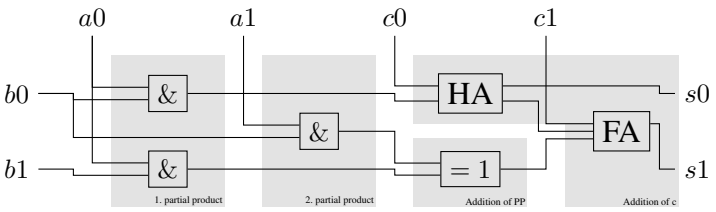
1. Realisieren Sie eine kombinatorische Schaltung, die $(a \cdot b) + c$ für 2-Bit-Binärzahlen berechnet. Verwenden Sie dazu Halbaddierer, Volladdierer und unsere Standardbibliothek.

2. Implementieren Sie eine kombinatorische Schaltung, die $(a \cdot b) + c$ für 8-Bit-Binärzahlen berechnet in Verilog.

3. Verwenden Sie das in 6.2.2 geschriebene Modul, um ein weiteres Modul zu schreiben, das:
   a) jeden Taktzyklus zwei 8-Bit-Zahlen a und b nimmt
   b) die Summe aller vorangegangenen Berechnungen $(a \cdot b)$ ausgibt
   c) einen reset-Eingang hat, um die Summe zurückzusetzen.

Lösungsvorschlag:

1.



2.

```
1  module MulSum(
2     input [7:0] a,b,c,
3     output [7:0] s
```

```
);

        assign s = (a*b)+c;

endmodule
```

3.

```
module MLA(
  input clk,
  input rst,
  input [7:0] a,b,
  output [7:0] out
);
        // wire for mulsum output
        wire [7:0] mulSumOut;
        // register to store sum of previous clock cycles
        reg [7:0] MLA_store = 8'b0;
        // instantiate module from 6.1.2 with a,b and the sum of the previous clock
        MulSum mulsum( .a(a), .b(b), .c(MLA_store), .s(mulSumOut));
        always @ (posedge clk) // for synchronized reset
        begin
          if(!rst)
                // stores the result of the current cycle at the end of the cycle
                MLA_store <= mulSumOut;
          else
                // resets to 0 if rst is 1
                MLA_store <= 8'b00000000;
        end
        //assign output
        assign out = MLA_store;
endmodule
```

Aufgabe 6.3: Verilog: 4-MUX

Schreiben Sie ein Verilog Modul, welches vier n-Bit Zahlen a_0, a_1, a_2, a_3 und zwei Selektionsbits s_0, s_1 als Eingaben erhält, und $a_{<s_1 s_0>}$ ausgibt. Verwenden Sie dabei nicht den ternären Auswahloperator. Geben Sie auch den zugehörigen Schaltkreis an (ohne normale Multiplexer zu nutzen).

Lösungsvorschlag:

```
module multiplexer #(parameter n)(
        input [n-1:0] a0,
        input [n-1:0] a1,
        input [n-1:0] a2,
        input [n-1:0] a3,
        input [1:0] s,
        output [n-1:0] erg
);
        reg [n-1:0] h1;
        reg [n-1:0] h2;
        assign h1 = ((a0 &(~s[0])) | (a1 & s[0]));
        assign h2 = ((a2 &(~s[0])) | (a3 & s[0]));
        assign erg = ((h1 &(~s[1])) | (h2 & s[1]));
endmodule
```

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

System Architecture SS 2021

Tutorial Sheet 6 (Suggested Solutions)

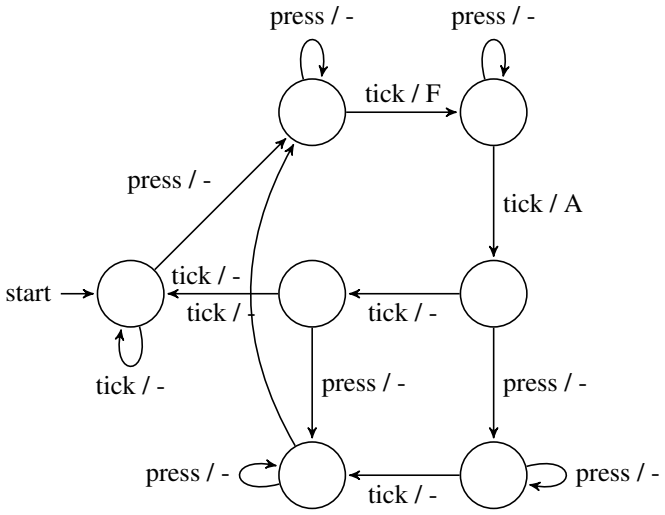
Note: **This task sheet was created by tutors. The tasks are neither relevant nor irrelevant for the exam, the suggested solutions are neither correct nor incorrect.**

Problem 6.1: Finite State Machines

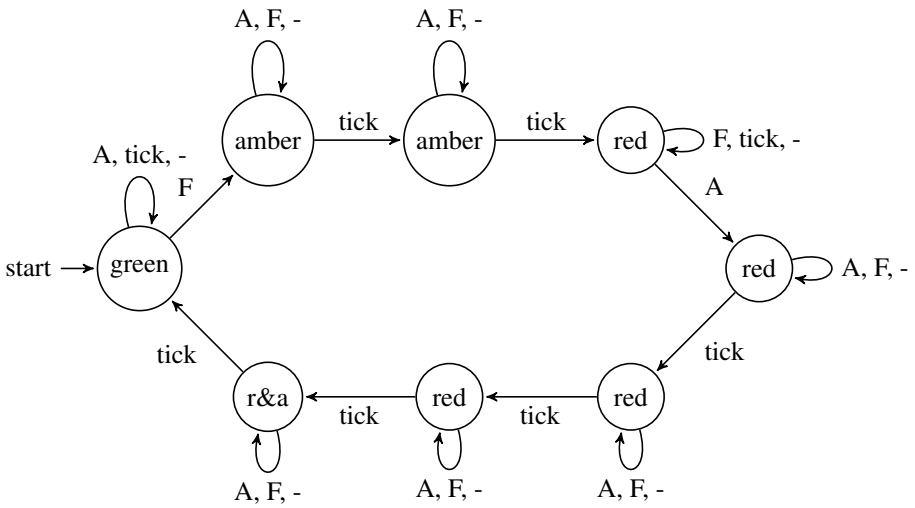
1. Construct a Mealy machine that implements the control of a simple pedestrian traffic light and follows the following spezifications:
   - **Output**: A (Cars) and F (Pedestrians), depending on who is allowed to drive/walk.
   - **Input**: $press$ (a pedestrian presses and wants to cross the street) and a signal $tick$ (every twenty seconds acording to an internal clock).
   - **Behaviour**: As long as no pedestrian presses, the traffic light shows green for cars. When a pedestrian presses, he has green for twenty seconds, starting with the next $tick$ of the internal clock. After this sixty seconds have to pass until the traffic light can once again show green for pedestrians.
2. A traffic light of course does not show who is allowed to pass by F or A.
   Construct a Moore machine that realizes the behaviour of the traffic light for cars:
   - The machine gets the inputs F and A and every two seconds a signal $tick$ from the internal clock.
   - There are four outputs: green, yellow, red, and yellow and red simultaniously.
   - The traffic light switches for four seconds from green to yellow before changing to red.
   - Starting with red the traffic light first waits 6 seconds, then switches for two seconds to red and yellow and than to green.

Suggested solution: For the sake of visibility we have not named the states.

1. Mealy machine with seven states:



2. Output is written in the state. We assume that the two machines are synchronised by the clock and the signals F and A only happen when a $tick$ occurs.



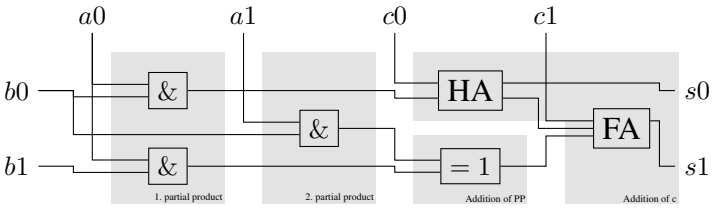
Problem 6.2: Circuits and Verilog

Digital signal processors (DSPs) are specialized processors for processing image, sound, and radio signals. A common task for DSPs is to calculate the sum over multiple multiplications. To do this efficiently, their instruction sets often provide a special Multiply and Acumulate (MLA) instruction. This takes three numbers as input and calculates $(a \cdot b) + c$.

1. Implement a combinatorial circuit that computes $(a \cdot b) + c$ for 2-bit binary numbers. Use half adders, full adders, and our standard library.
2. Implement a combinatorial circuit that computes $(a \cdot b) + c$ for 8-bit binary numbers in Verilog.
3. Use the module written in 6.2.2 to write another module that:
   a) takes two 8-bit numbers a and b every clock cycle.
   b) outputs the sum of all previous calculations $(a \cdot b)$
   c) has a reset input to reset the sum.

Suggested solution:

1.



2.

```
1  module MulSum(
2    input [7:0] a,b,c,
3    output [7:0] s
4  );
```

```
5
6           assign s = (a*b)+c;
7
8   endmodule
```

3.

```
1   module MLA(
2     input clk,
3     input rst,
4     input [7:0] a,b,
5     output [7:0] out
6   );
7           // wire for mulsum output
8           wire [7:0] mulSumOut;
9           // register to store sum of previous clock cycles
10          reg [7:0] MLA_store = 8'b0;
11          // instantiate module from 6.1.2 with a,b and the sum of the previous clock
12          MulSum mulsum( .a(a), .b(b), .c(MLA_store), .s(mulSumOut));
13          always @ (posedge clk) // for synchronized reset
14          begin
15            if(!rst)
16                  // stores the result of the current cycle at the end of the cycle
17                  MLA_store <= mulSumOut;
18            else
19                  // resets to 0 if rst is 1
20                  MLA_store <= 8'b00000000;
21          end
22          //assign output
23          assign out = MLA_store;
24  endmodule
```

Problem 6.3: Verilog: 4-MUX

Write a Verilog-module that has 4 n-bit numbers a_0, a_1, a_2, a_3 and two selection bits s_0, s_1 as inputs and $a_{<s_1 s_0>}$ as output. Do not use the ternary operator for the module. Also construct the circuit represented by the module (without using a normal multiplexer).

Suggested solution:

```
1   module multiplexer #(parameter n)(
2           input [n-1:0] a0,
3           input [n-1:0] a1,
4           input [n-1:0] a2,
5           input [n-1:0] a3,
6           input [1:0] s,
7           output [n-1:0] erg
8   );
9           reg [n-1:0] h1;
10          reg [n-1:0] h2;
11          assign h1 = ((a0 &(~s[0])) | (a1 & s[0]));
12          assign h2 = ((a2 &(~s[0])) | (a3 & s[0]));
13          assign erg = ((h1 &(~s[1])) | (h2 & s[1]));
14  endmodule
```