64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 10 Ausgabe: 18.12., Abgabe: 08.01. 24:00

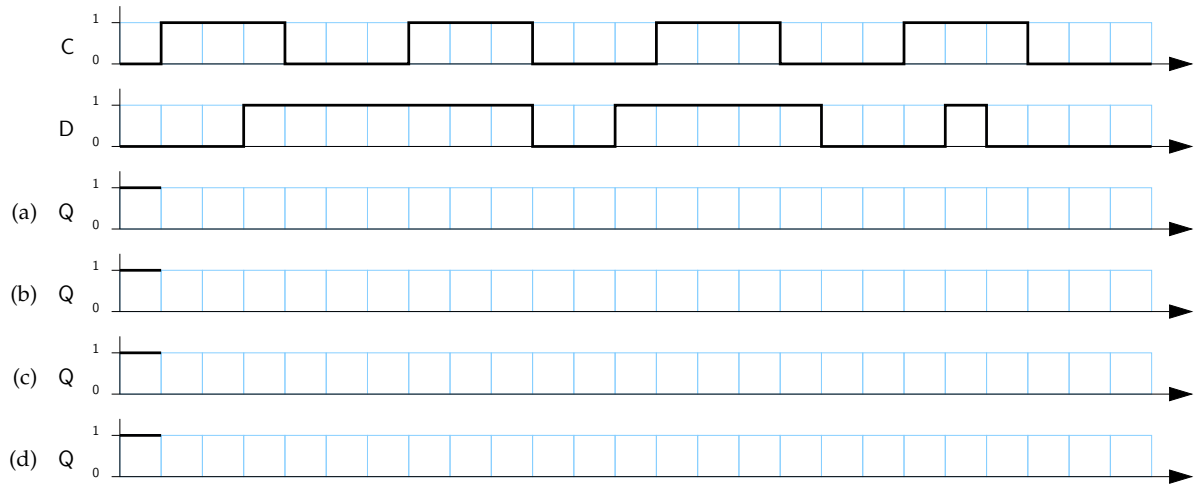
Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 10.1 (Punkte 4·5 (+1))

Flipflop Typen: Vervollständigen Sie für jedes Flipflop das folgende Impulsdiagramm (mit einer Zeiteinheit Verzögerung je Flipflop) und geben Sie an, um was für einen Typ es sich dabei handelt (jeweils einen Zusatzpunkt).

Die Flipflops sind hier in VHDL-Syntax beschrieben: `entity FF` definiert die Ein- und Ausgänge der Schaltung, während `architecture` ⟨*name*⟩ dann die jeweilige Implementation ist. Das Verhalten jedes der Flipflops ergibt sich aus den Anweisungen im Code; `rising_edge(C)`, bzw. `falling_edge(C)` sind boole'sche Funktionen die nur wahr werden, wenn auf dem Signal `C` eine Vorder-/Rückflanke auftritt. Wird die jeweilige Bedingung für die Zuweisung (Operator <=) nicht erfüllt, dann ändert sich der Ausgang `Q` nicht.

```
entity FF is
port(   C       : in  std_logic;        -- Clock Eingang
        D       : in  std_logic;        -- Daten, bzw. Enable Eingang
        Q       : out std_logic);       -- Ausgang
end entity FF;
```



(a)
```
architecture AUFG10_1a of FF is
begin
  process(C) is
  begin
    if (C = '1') then
        Q <= D;
    end if;
  end process;
end architecture AUFG10_1a;
```

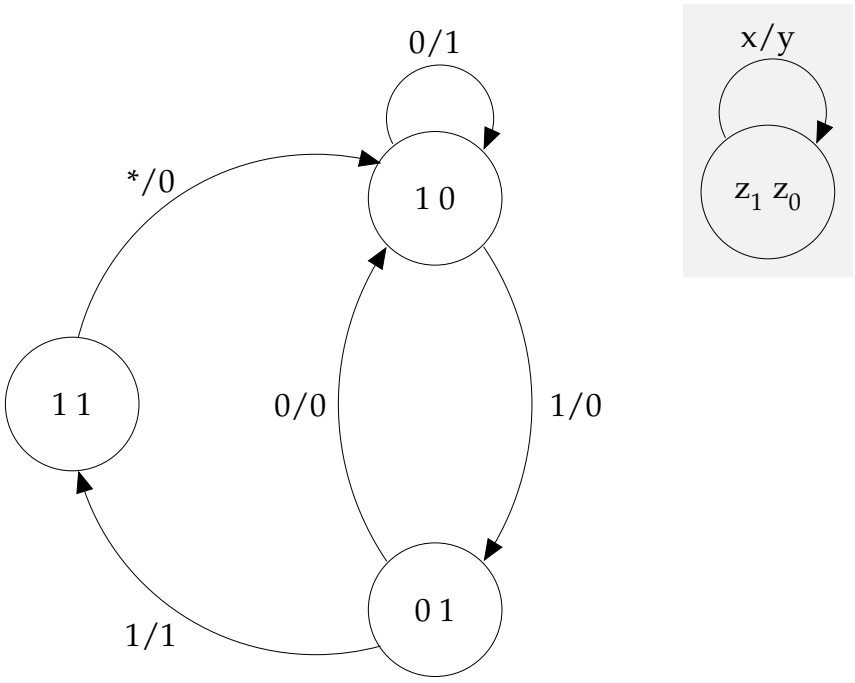
(b)
```
architecture AUFG10_1b of FF is
begin
  process(C) is
  begin
    if rising_edge(C) then
        Q <= D;
    end if;
  end process;
end architecture AUFG10_1b;
```

(c)
```
architecture AUFG10_1b of FF is
begin
  process(C) is
  begin
    if falling_edge(C) then
        Q <= D;
    end if;
  end process;
end architecture AUFG10_1b;
```

(d)
```
architecture AUFG10_1d of FF is
begin
  process(C) is
    variable L : std_logic;             -- lokale Variable
  begin
    if rising_edge(C) then
        L := D;                         -- := ist Variablenzuweisung
    elsif falling_edge(C) then          -- Kurzform 'else if'
        Q <= L;
    end if;
  end process;
end architecture AUFG10_1d;
```

Aufgabe 10.2 (Punkte 10+10+5+5+5)

Schaltwerk-Analyse: Wir betrachten das Zustandsdiagramm eines Automaten mit einem Eingang x, einer Ausgabe y und den drei Zuständen $Z = (z_1, z_0) = \{(1,0), (0,1), (1,1)\}$:



(a) Ermitteln Sie aus dem Zustandsdiagramm die zugehörigen Gleichungen für die Übergangsfunktion δ sowie die Ausgangsfunktion λ in disjunktiver Form.

Zur Lösung sollen dabei Übergangs- und Ausgangstabellen erstellt werden, die dann in KV-Diagramme übertragen werden. Daraus sind dann minimierte Schaltfunktionen abzuleiten.

(b) Überprüfen Sie den Automaten auf Vollständigkeit (in jedem Zustand ist für jede Eingangsbelegung mindestens ein Übergang aktiv) und Widerspruchsfreiheit (in jedem Zustand ist für jede Eingangsbelegung höchstens ein Übergang aktiv).

(c) Handelt es sich um einen Mealy- oder einen Moore-Automaten? Begründen Sie Ihre Antwort kurz.

(d) Erstellen Sie einen Schaltplan mit HADES.

(e) Ist der Automat in der vorgegeben Form praktisch einsetzbar?

Wenn ja: Warum?
Wenn nicht: Wie kann man das Problem lösen?

Aufgabe 10.3 (Punkte 10+10+10)

Entwurf eines Schaltwerks: Wir betrachten ein Schaltwerk mit sechs Zuständen $Z_0, \ldots Z_5$, einem Eingang x und vier Ausgängen y_3, y_2, y_1, y_0. Die Zustandsübergänge und die Ausgabe sind in der Tabelle angegeben:

x	Z	Z^+	y_3	y_2	y_1	y_0
0	Z_0	Z_1	0	1	0	0
0	Z_1	Z_2	0	0	0	0
0	Z_2	Z_3	1	0	0	0
0	Z_3	Z_5	0	1	1	1
0	Z_4	Z_0	0	0	1	0
0	Z_5	Z_4	1	1	0	1
1	Z_0	Z_2	0	1	0	0
1	Z_1	Z_4	0	0	0	0
1	Z_2	Z_0	1	0	0	0
1	Z_3	Z_4	0	1	1	1
1	Z_4	Z_3	0	0	1	0
1	Z_5	Z_2	1	1	0	1

(a) Zeichnen Sie das Zustandsdiagramm des Schaltwerks.

(b) Um das Schaltwerk zu realisieren, wählt man jetzt eine Codierung der Zustände. Wir betrachten zwei (von vielen) Möglichkeiten.
Bestimmen Sie für beide Codierungen die Funktionen des Zustandsübergangsschaltnetzes (das δ-Schaltnetz). Beachten Sie dabei die Möglichkeit von *Don't-Cares*. Die Tabellen und KV-Diagramme sollen mit abgegeben werden.

Z_i	Codierung 1 ($z_2\, z_1\, z_0$)	Codierung 2 ($z_2\, z_1\, z_0$)
Z_0	(100)	(001)
Z_1	(000)	(110)
Z_2	(011)	(000)
Z_3	(001)	(101)
Z_4	(010)	(100)
Z_5	(101)	(010)

(c) Offenbar lässt sich durch eine geeignete Codierung der Zustände eine erhebliche Vereinfachung der Schaltfunktionen erreichen. Das Problem ist nur, dass es alles andere als einfach ist, eine bestmögliche Codierung zu finden, wobei man dann auch noch die Funktionen für die Ausgabe (das λ-Schaltnetz) mit berücksichtigen müsste.

Geben Sie eine Codierung für die sechs Zustände an, die zumindest das λ-Schaltnetz des Automaten minimiert. Es sind dabei auch mehr als drei Bits für die Zustandscodes erlaubt. Erläutern Sie ihre Vorgehensweise.

Aufgabe 10.4 (Punkte 10+5)

Installation und Test eines C-Compilers (GNU Toolchain): In Vorbereitung auf Kapitel 13 zum x86-Assembler und analog zu den Beispielen in Kapitel 2 ab Folie 95, sollen Sie selbst Zugang zu einem C-Compiler und den zugehörigen Tools haben. Wir empfehlen die *GNU Toolchain* mit dem `gcc` C-Compiler und Werkzeugen. Die Programme sind auf Linux-Systemen in der Regel vorinstalliert, so dass Sie die Befehle direkt ausführen können.

Im Informatikum können Sie die Dual-Boot Rechner in den Poolräumen unter Linux nutzen. Wenn Sie zu Hause lernen und keinen Linux PC haben, bzw. die Cygwin-Umgebung (s.u.) nicht installieren wollen, können sie auch „Remote“ auf den Informatik Rechnern arbeiten: aus einem Linux-Terminal / der Windows-Eingabeaufforderung einloggen:

```
ssh rzssh1.informatik.uni-hamburg.de
```

. . . dort dann Start der Programme. **Achtung**: ihr Linux HOME-Verzeichnis ist `infhome`.

Für Windows-Systeme könnten Sie die Cygwin-Umgebung von cygwin.com herunterladen und installieren. Im Setup von Cygwin dann bitte den gcc-Compiler und die Entwickler-Tools auswählen und installieren. Natürlich können Sie auch jeden anderen C-Compiler verwenden, müssen sich dann aber die benötigten Befehle und Optionen selbst heraussuchen.

Hauptsächlich soll Sie die Aufgabe dazu motivieren vielleicht auf dem eigenen Rechner mit den Werkzeugen zu „spielen“ und so auch selbst zu sehen was aus programmiertem Code auf niedrigeren Abstraktionsebenen wird. Also installieren Sie die entsprechenden Programme (distributionsabhängig).

Anmerkung: keine Angst, die Aufgabe soll zeigen, wie Assemblercode aussieht und Ihnen helfen erste Einblicke zu gewinnen, wie Betriebssystem, (Programm-) Binär-Code und die Hardware zusammenspielen. **Es geht nicht darum Assemblerprogrammierung zu lernen!**

Für einen ersten Test tippen Sie bitte das folgenden Programm ab oder laden Sie sich die Datei `aufg10_4.c` herunter. Passen Sie die Datei an, indem Sie dort ihren Namen und die Matrikelnummer eintragen. Anschließend sollen Sie das Programm übersetzen und sich den erzeugten Assembler- und Objektcode analysieren.

```c
/* aufg10_4.c
 * Einfaches Programm zum Test des C-Compilers und der zugehörigen Tools.
 * Bitte setzen Sie in das Programm ihren Namen und die Matrikelnummer ein
 * und probieren Sie alle der folgenden Operationen aus:
 *
 * Funktion         Befehl                                    erzeugt
 * ----------------+-----------------------------------------+----------------
 * C -> Assembler:  gcc -O2 -S aufg10_4.c                    -> aufg10_4.s
 * C -> Objektcode: gcc -O2 -c aufg10_4.c                    -> aufg10_4.o
 * C -> Programm:   gcc -O2 -o aufg10_4.exe aufg10_4.c       -> aufg10_4.exe
 * Disassembler:    objdump -d aufg10_4.o
 *                  objdump -d aufg10_4.exe
 * Ausführen:       aufg10_4.exe
 */

#include <stdio.h>

int main( int argc, char** argv )
{ int matrikelNr    = 456789;
  char vorname[32]  = "Studi";
  char nachname[32] = "Informaticus";
//char *vorname     = "Studi";
//char *nachname    = "Informaticus";

  printf("Name: %s %s - Matrikelnr.: %d\n", vorname, nachname, matrikelNr);
  return 0;
}
```

(a) Machen Sie sich mit dem Compiler und den Tools vertraut. Probieren Sie die vorgeschlagenen Befehle aus und sehen Sie sich die Ausgaben an.

Erzeugen Sie eine Textdatei, die die Ausgabe des Programms und ein Listing des Disassemblers enthält. Dies geschieht am einfachsten mit den folgenden Befehlen:

```
./aufg10_4.exe          > loesung10_4.txt
echo "=========================================" >> loesung10_4.txt
objdump -d aufg10_4.o >> loesung10_4.txt
```

Markieren Sie in der Datei an welcher Stelle des Codes: Vorname, Nachname und Matrikelnummer stehen (mit kurzer Begründung). Diese Datei ist als Lösung des Aufgabenteils abzugeben.

(b) In dem Code `aufg10_4.c` sind die Zeilen 22 und 23 auskommentiert. Ändern Sie die Variablen für Vor- und Nachnamen in die zweite Version (Zeiger auf den String, statt `char`-Array).

Was ändert sich in dem Assembler-Code? Es genügt, die Änderungen (inhaltlich) zu beschreiben, es müssen keine Listings abgegeben werden.