64-041 Übung Rechnerstrukturen

Aufgabenblatt 10 Ausgabe: 09.01., Abgabe: 16.01. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

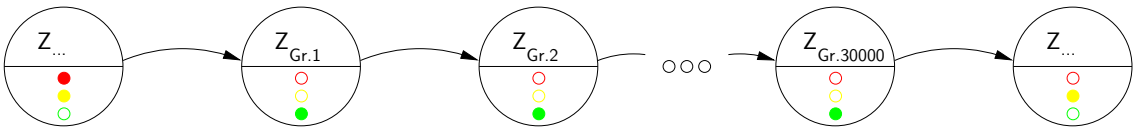
Aufgabe 10.1 (Punkte 5+10+10+10+10)

Gekoppelte Automaten: In der Vorlesung wurde eine Ampelschaltung (Folie 704ff.) vorgestellt. Wird das System jetzt aber mit einem konstanten Takt von beispielsweise 1 KHz betrieben, dann muss man für einen realistischen Betrieb die Zeiten der jeweiligen Ampelphasen deutlich verlängern. Soll beispielsweise die Grünphase 30 Sekunden dauern, dann entspricht das 30 000 Takten.

Wir betrachten jetzt eine Ampelschaltung mit einem zentralen 1 KHz Takt. Die Ampel soll zyklisch die vier Ausgaben {rot, rot-gelb, grün, gelb} erzeugen, wobei folgende Zeitbedingungen einzuhalten sind:

Zustand	Zeitdauer
rot	25 Sek.
rot-gelb	2 Sek.
grün	30 Sek.
gelb	4 Sek.

(a) Eine ad-hoc Lösung wäre natürlich, wie in der Grafik unten skizziert, entsprechend viele Zustände einzuführen, die nacheinander durchlaufen werden.



Dieser Ansatz hat jedoch mehrere Nachteile, weswegen man üblicherweise zwei (oder mehr) *gekoppelte Automaten* einsetzt: einen „Hauptautomaten" der die Zustandsübergänge realisiert und einen Zähler (trivialer Automat), der für die Wartezeiten sorgt. Zählen Sie kurz (jeweils ein Satz zur Begründung) auf, welche Nachteile dies sein könnten.

(b) Beschreiben Sie (textuell), wie die gekoppelten Automaten zusammenarbeiten. Welche Leitungen gehen vom Hauptautomaten zum Zähler, welche gehen zurück?

(c) Überlegen Sie sich was passiert, wenn beide Automaten mit identischem Taktsignal arbeiten: der Taktvorderflanke? Was ist dabei zu berücksichtigen? Beschreiben Sie, wie die

Ampel auf die Grünphase schaltet (Z_{Gr}), die Wartezeit von 30 Sekunden vergeht und anschließend dieser Zustand verlassen wird.

Tipp: bei den gekoppelten Automaten und ihren Taktschemata ist entscheidend, wann die einzelnen Zustandsübergänge stattfinden. Bei gegenseitigen Abhängigkeiten (Steuersignale, Zählerstände) kann es leicht vorkommen, dass man einen Takt „zu spät" ist.

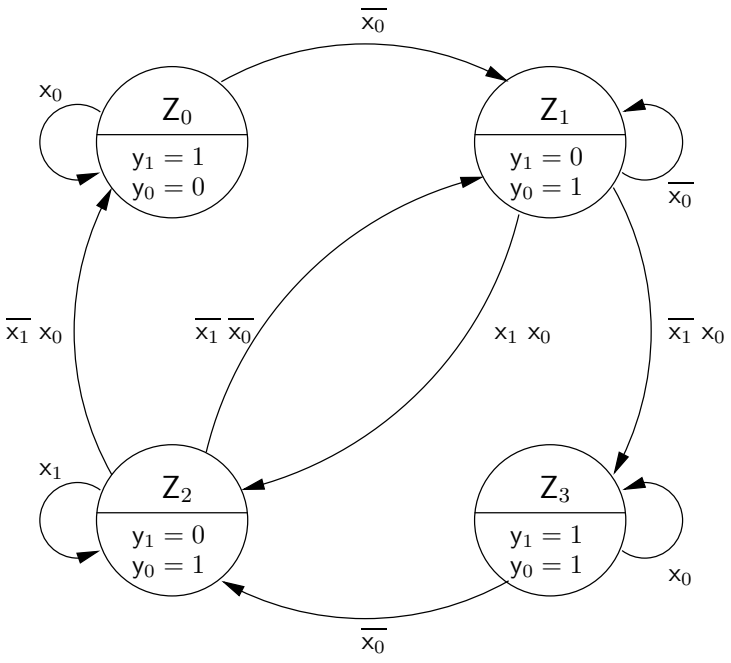
(d) Überlegen Sie sich was passiert, wenn die Automaten unterschiedlich getaktet sind, so dass der eine mit der Taktvorderflanke und der zweite Automat mit der Rückflanke arbeitet. Verfahren Sie wie in Aufgabenteil (c).

(e) Entwerfen Sie jetzt die zugehörige Ampelschaltung, wobei Sie als Taktschema eine der beiden Varianten aus Aufgabenteil (c) oder (d) wählen können.

Zeichnen Sie das Zustandsdiagramm für den Hauptautomaten als Moore-Modell und geben sie für jeden Zustand an, welche Werte die Ausgangsleitungen (die Lampen und alle Steuerleitungen für den/die Zähler) haben.

Aufgabe 10.2 (Punkte 10+10+10)

Schaltwerk-Analyse: Wir betrachten das Zustandsdiagramm eines Moore-Schaltwerks mit Eingängen $X = (x_1, x_0)$ und Ausgaben $Y = (y_1, y_0)$ sowie vier Zuständen Z_0, Z_1, Z_2, Z_3. Wir codieren die Zustände Z binär mit zwei Bits (z_1, z_0) und damit $Z_0 = (0,0)$, $Z_1 = (0,1)$, $Z_2 = (1,0)$ und $Z_3 = (1,1)$:



(a) Ermitteln Sie aus dem Zustandsdiagramm die zugehörigen Gleichungen für die Übergangsfunktion δ zur Berechnung des Folgezustands Z^+ aus aktuellem Zustand Z und den Eingabewerten X.

Eine Lösungsmöglichkeit ist das Aufstellen der Flusstafel, alternativ das Aufstellen der Übergangs- und Ausgangstabellen und dann die Logikminimierung.

(b) Ermitteln Sie die zugehörigen Gleichungen für die Ausgangsfunktion λ zur Berechnung des Ausgangswerts Y als Funktion des aktuellen Zustands Z.

(c) 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).

Aufgabe 10.3 (Punkte 10+10+5)

Installation und Test der GNU Toolchain: Ziel dieser Aufgabe ist es, dass Sie selbst Zugang zu einem C-Compiler und den zugehörigen Tools haben. Wir empfehlen die *GNU Toolchain* mit dem `gcc` C-Compiler und Werkzeugen. Diese ist auf den meisten Linux-Systemen bereits vorinstalliert, so dass Sie die Befehle direkt ausführen können. Am einfachsten geht dies, indem Sie die Linux-, bzw. Dual-boot Rechner in den PC-Poolräumen nutzen.

Ansonsten soll Sie die Aufgabe aber auch motivieren vielleicht auf dem eigenen Rechner mit den Werkzeugen zu „spielen“. Also installieren Sie die entsprechenden Programme (distributionsabhängig).

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. Alternativ können Sie auch einen anderen C-Compiler verwenden, Sie müssen sich dann aber die benötigten Befehle und Optionen selbst heraussuchen.

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_3.c` von der Webseite herunter. Passen Sie die Datei an, indem Sie dort ihren Namen und die Matrikelnummer eintragen. Anschließend können Sie das Programm übersetzen und sich den erzeugten Assembler- und Objektcode anschauen.

```
1  /* aufg10_3.c
2   * Einfaches Programm zum Test des gcc-Compilers und der zugehörigen Tools.
3   * Bitte setzen Sie in das Programm ihren Namen und die Matrikelnummer ein
4   * und probieren Sie alle der folgenden Operationen aus:
5   *
6   * Funktion          Befehl                                    erzeugt
7   * ----------------+-----------------------------------------+----------------
8   * C -> Assembler:  gcc -Og -S aufg10_3.c                    -> aufg10_3.s
9   * C -> Objektcode: gcc -Og -c aufg10_3.c                    -> aufg10_3.o
10  * C -> Programm:   gcc -Og -o aufg10_3.exe aufg10_3.c       -> aufg10_3.exe
11  * Disassembler:    objdump -d aufg10_3.o
12  *                  objdump -d aufg10_3.exe
13  * Ausführen:       aufg10_3.exe
14  */
15
```

```
16 #include <stdio.h>
17
18 int main( int argc, char** argv )
19 { int matrikelNr    = 123456;
20   char vorname[32]  = "John";
21   char nachname[32] = "Doe";
22 //char *vorname     = "John";
23 //char *nachname    = "Doe";
24
25   printf("Name: %s %s - Matrikelnr.: %d\n", vorname, nachname, matrikelNr);
26   return 0;
27 }
```

(a)  [1] Machen Sie sich mit dem Compiler und den Tools vertraut. Probieren Sie die vorgeschlagenen Befehle aus und sehen Sie sich die Ausgaben an.

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

```
./aufg10_3.exe          > loesung10_3.txt
echo "=========================================" >> loesung10_3.txt
objdump -d aufg10_3.o >> loesung10_3.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.

(c) In dem Code `aufg10_3.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.

[1] 10 Punkte als verspätetes Weihnachtsgeschenk ;-)