64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 10 Ausgabe: 20.01., Abgabe: 27.01. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

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

Gekoppelte Automaten: In der Praxis müssen Automaten oft für eine bestimmte Zeit in Zuständen bleiben. Ein Beispiel wäre die Ampelschaltung der aus der letzten Übungsaufgabe 9.4. Dort war der Automat so beschrieben, dass die Ampel mit jedem Takt weiterschaltet, außer in dem Zustand Z_2, der der Hauptstraße eine Grünphase gibt, solange die Kontaktschleife der Nebenstraße nicht auslöst.

Jetzt soll die Grünphase der Nebenstraße aber auch deutlich länger andauern, als das „normale Umschalten“ (= 1 Takt). In unseren Beispiel soll die Grünphase 200 Takte dauern.

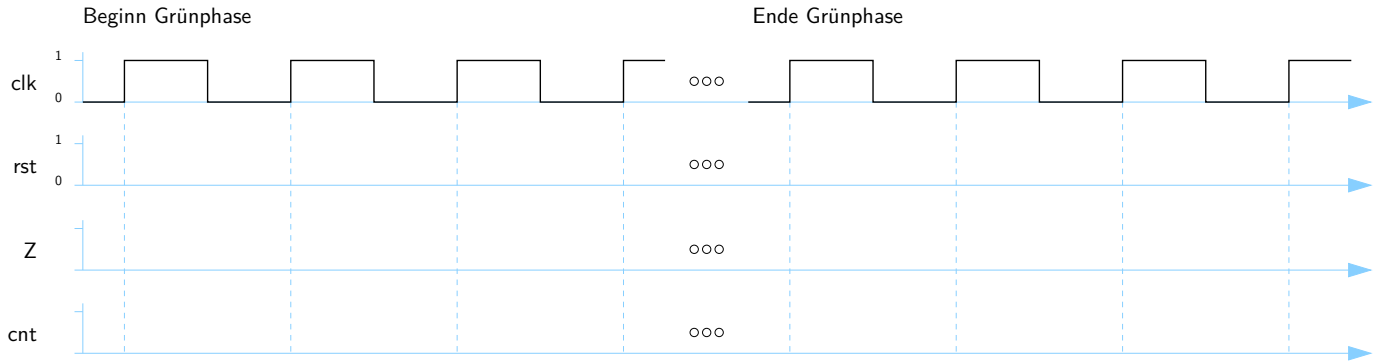
(a) So wäre es natürlich möglich einfach die Anzahl der Zustände zu erhöhen. Wie viele Zustände müsste der Automat dann haben? Was spricht gegen solche Lösungen?

In der Praxis nutzt man in solchen Situationen zwei (oder mehr) *gekoppelte Automaten*: einen „Hauptautomaten“, der die eigentliche Funktionalität realisiert, und einen Zähler (trivialer Automat), der für die Wartezeiten sorgt.

(b) Dazu soll es einen 8-bit Zähler geben der auf 0 gesetzt werden kann und sonst mit der Taktflanke raufzählt. Was müsste bei der Realisierung so einer Ampelschaltung zusätzlich implementiert werden?
Stellen Sie sich dazu vor, dass der Zähler in die HADES Schaltung aus Aufgabe 9.4 (d) eingebaut werden soll: was fehlt noch? (Es soll kein HADES Modell erstellt werden!)

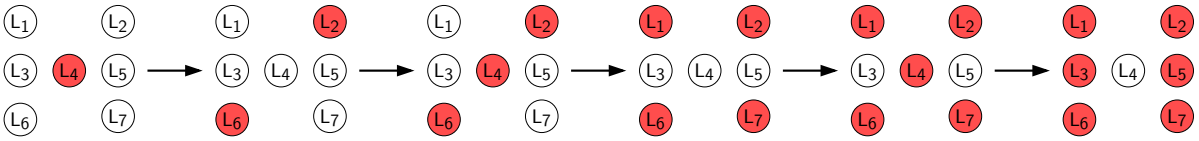
(c) Angenommen die externe Schaltung kann durch ein Signal *rst* des Hauptautomaten asynchron auf 0 gesetzt werden und liefert den Zählerstand zurück. Zeichnen Sie das Zustandsdiagramm des für die Ampelschaltung (analog zu 9.4 (a)).

(d) Beide Automaten sollen mit der Taktvorderflanke umschalten. Skizzieren Sie für den Fall der Grünphase-Nebenstraße, vergleichbar einem Impulsdiagramm, wie sich Zustandswechsel des Hauptautomaten Z_i, das Reset-Signal *rst* und der Zählerstand *cnt* zueinander verhalten.



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

Entwurf eines Automaten: In dieser Aufgabe soll das Schaltwerk für einen digitalen Würfel entworfen werden, d.h. eine Schaltung, die auf einem Feld von sieben Leuchtdioden zyklisch folgende Ausgaben erzeugt:



Z_1	0	0	0
Z_2	0	0	1
Z_3	0	1	0
Z_4	0	1	1
Z_5	1	0	0
Z_6	1	0	1

Der Automat habe weiter einen Eingang S. Wenn $S = 0$ ist, wird die Zustandsfolge $1 \ldots 6$ zyklisch durchlaufen; bei $S = 1$ bleibt der Automat im aktuellen Zustand hängen. Hier die Zustandscodierung:

(a) Zeichnen Sie das Zustandsübergangsdiagramm.

(b) Erstellen Sie Zustands- und Ausgangstabelle des Automaten. Die Tabelle enthält links den Eingangswert S und den aktuellen Zustand Z in 3-bit Binärcodierung (z_2, z_1, z_0). Angegeben sind dann der Folgezustand Z^+ und die Ausgangswerte zum Ansteuern der LEDs l_1, l_2, l_3, l_4.

(c) Warum müssen die anderen LEDs nicht beachtet werden?

(d) Übertragen Sie die boole'schen Funktionen in KV-Diagramme und minimieren Sie sie. Markieren Sie mögliche Schleifen und geben Sie die zugehörigen Ausdrücke an.

(e) Was würde geschehen, wenn die Schaltung beim Einschalten in einen der beiden Zustände $Z = (110)$ oder (111) gerät?

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

(g) Geben Sie eine Codierung für die sechs Zustände an, die das δ-Schaltnetz des Automaten minimiert. Auch hier sollen mehr als drei Bits für die Codierung der Zustände erlaubt sein. Wie würde bei dieser Zustandscodierung das λ-Schaltnetz aussehen?

(h) Welches Problem könnte auftreten, wenn man den Automaten wirklich in etwas naiver Weise mit einem solchen minimalen δ- bzw. λ-Schaltnetz realisiert? Was lässt sich gegen dieses Problem tun?

Aufgabe 10.3 (Punkte 10+5)

Installation und Test der GNU Toolchain: In Vorbereitung auf Kapitel 13 zum x86-Assembler und analog zu den Beispielen in Kapitel 2 ab Folie 90, 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. Diese ist auf den meisten Linux-Systemen bereits vorinstalliert, so dass Sie die Befehle direkt ausführen können.

Normalerweise könnten Sie dazu die Dual-boot Rechner in den PC-Poolräumen nutzen... Wenn Sie keinen Linux Rechner haben, bzw. nicht die Cygwin-Umgebung (s.u.) installieren wollen, können sie sich auch per `ssh` auf der `rzssh1.informatik.uni-hamburg.de` einloggen und die dort installierten Programme nutzen.

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).

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.

```c
/* aufg10_3.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 -Og -S aufg10_3.c                  -> aufg10_3.s
 * C -> Objektcode: gcc -Og -c aufg10_3.c                  -> aufg10_3.o
 * C -> Programm:   gcc -Og -o aufg10_3.exe aufg10_3.c     -> aufg10_3.exe
 * Disassembler:    objdump -d aufg10_3.o
 *                  objdump -d aufg10_3.exe
 * Ausführen:       aufg10_3.exe
 */

#include <stdio.h>

int main( int argc, char** argv )
{ int matrikelNr    = 123456;
  char vorname[32]  = "John";
  char nachname[32] = "Doe";
//char *vorname     = "John";
//char *nachname    = "Doe";

  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_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.

(b) 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.