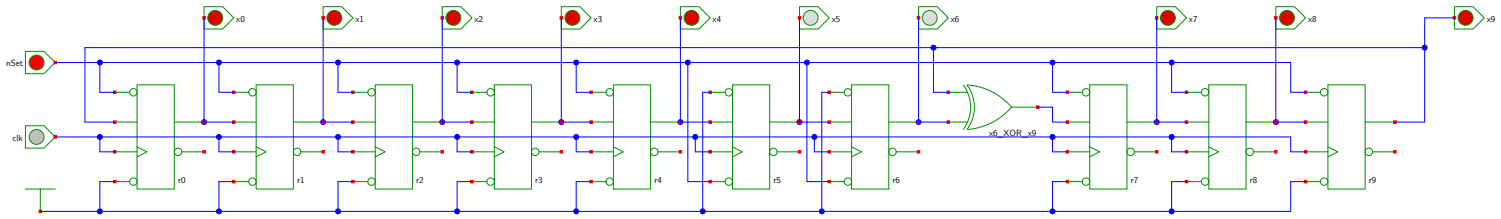
64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 10 Ausgabe: 20.12., Abgabe: 10.01. 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 10.1 (Punkte 20+5)

Pseudozufallszahlen: Linear rückgekoppelte Schieberegister (*LFSR*: Linear-Feedback Shift Register) werden außer in der Prüfsummenberechnung (RSB-Vorlesung: Kapitel 7, Folie 437ff) auch gerne zu Erzeugung von Zahlensequenzen als Pseudozufallszahlen benutzt. Dazu soll die unten skizzierte HADES Schaltung genutzt werden, in Datei: *LFSR.zip*



Die 10 Flipflops bilden ein Schieberegister in dem eine Zahl $X = x_9 \ldots x_0$ gespeichert ist, die mit jeder Taktflanke nach links (in der Grafik rechts!) geschoben wird. Bit x_9 wird in x_0 rotiert (Rückkopplung), außerdem wird $x_7 = x_9 \oplus x_6$ (Linearkombination).[1] Mit dem asynchronen Set-Eingang *nSet* wird das Register zu Beginn mit $x = 1110011111$ initialisiert. Der Inhalt des Registers, als Zahl interpretiert oder einzelne Bits x_i können dann als pseudozufällige Werte benutzt werden.

Der Begriff „Pseudozufall" bedeutet hier, dass diese Werte zwar systematisch erzeugt werden und auch immer die gleiche Sequenz von Zahlen erzeugen, diese aber nicht wie beim Zählen der Ordnung folgt.

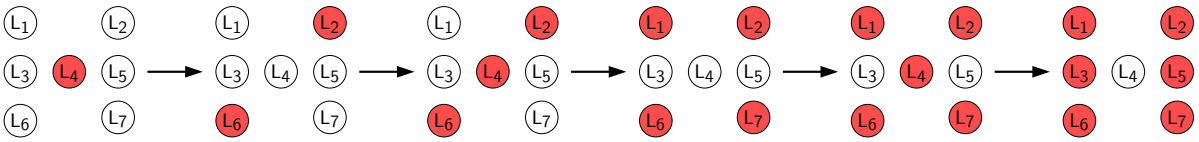
(a) Schreiben Sie ein Programm, das die Zahlenfolge des LFSRs erzeugt und als Text ausgibt. Dazu können Sie jede beliebige Programmiersprache nutzen. Das Programmlisting und die ersten 15 Zahlen sind abzugeben.

(b) Nach wie vielen Schritten wiederholt sich die Sequenz der Zahlen?

[1] Mathematisch entspricht das dem Polynom: $X^{10} + X^7 + 1$, für eine CRC-Berechnung hätten man noch einen zusätzlichen Dateneingang und die Berechnung von $x_0 = d_{in} \oplus x_9$.

Aufgabe 10.2 (Punkte 10+10+20+5+10)

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 die Ausgaben eines Würfels erzeugt, wie beispielsweise:



Die 6 Zustände z_2, z_1, z_0 sollen einfach als Dualzahl codiert werden. Der Automat hat jetzt noch zwei Eingänge a (Aktiv), r (Rückwärts):

$a = 0, r = *$: der Automat bleibt in seinem Zustand, eine Zahl wird also zweimal hintereinander gewürfelt.

$a = 1, r = 0$: mit jedem Takt (Vorderflanke) schaltet der Automat einen Zustand weiter, wie oben angedeutet.

$a = 1, r = 1$: bei der Eingabe $r = 1$ soll der Automat zwei Schritte zurück würfeln, also mit der Folge $1 \to 5 \to 3 \to 1 \ldots$, bzw. $2 \to 6 \to 4 \to 2 \ldots$.

(a) Zeichnen Sie das Zustandsübergangsdiagramm für diesen Automaten.

(b) Aus dem Diagramm von Aufgabenteil (a) sollen jetzt Zustandsübergangs- und Ausgangstabelle entwickelt werden. Benutzen Sie dabei folgende Variablenordnung: Eingangswerte a und r, dann aktueller Zustand Z in 3-bit Binärcodierung (z_2, z_1, z_0):

a	r	z_2	z_1	z_0	z_2^+	z_1^+	z_0^+	...
0	0	0	0	0	...			
...								

Tipps

- Sie können die Tabellen auch verkürzt mit Don't-Care Werten codieren.
- Es müssen nicht alle sieben LEDs als Ausgänge berücksichtigt werden, siehe RSB-Vorlesung: Kapitel 9, Folie 544f.

Dieser „würfelnde Automat" soll jetzt über die Pseudozufallszahlen aus Aufgabe 10.1 gesteuert werden...

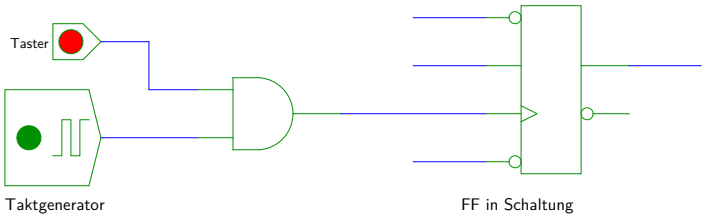
(c) Minimieren Sie die Schaltfunktionen aus Aufgabenteil (b) und geben Sie die Schaltung mit HADES ein. Dabei wird Eingang a mit x_8 des LFSR und r mit x_1 verbunden. Das LFSR ist fertig vorgegeben in: *LFSR.zip* – entpacken und README lesen!

Tipps

- Verwenden Sie *DFFRS* Flipflops für ihren Automaten, damit die Schaltung über ein 0-aktives Set-Signal in den Startzustand $Z = (0, 0, 1)$ versetzt werden kann. Als Beispiel dazu können Sie sich die LFSR-Schaltung ansehen.
- Beide Automaten erhalten identischen Takt (*clk*) und Initialisierungssignal (*nSet*).

(d) Wenn die Schaltung initialisiert worden ist, was sind die ersten 10 gewürfelten Werte?

(e) In der Praxis soll der Automat dauernd würfeln, bis der Benutzer eine Taste drückt, um seine Zahl abzulesen. Dies könnte man dadurch implementieren, dass der Takt, der ja dauernd „durchläuft“, mit der Taste gestoppt wird, wie hier skizziert:



Überlegen Sie sich, was an dieser Idee der Taktausblendung problematisch sein kann. Welche Alternative würde sich hier anbieten?

Aufgabe 10.3 (Punkte 15+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 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_3.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_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 -O2 -S aufg10_3.c                  -> aufg10_3.s
 * C -> Objektcode: gcc -O2 -c aufg10_3.c                  -> aufg10_3.o
 * C -> Programm:   gcc -O2 -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    = 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_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.