Universität Bremen – Fachbereich 3
Mathematik & Informatik

Dipl.-Inf. Daniel Große, grosse@informatik.uni-bremen.de, MZH 3460

Programmieraufgaben zur Vorlesung

Qualitätsorientierter Hardware-Entwurf

Als Voraussetzung zur Zulassung zum Fachgespräch muss eine der folgenden Aufgaben bearbeitet werden. Die Bearbeitung soll in Zweiergruppen erfolgen. Zum Vorlesungsende müssen die Programme vorgestellt werden. Dokumentation und geeignete Testfälle sind der Abgabe beizufügen.

Aufgabe 1

MorseCode-Generator in SystemC - In der Vorlesung wurde ein einfacher MorseCode-Generator mit VHDL modelliert. Als Programmieraufgabe soll ein MorseCode-Generator mit SystemC modelliert werden. Dieser soll allerdings keine 7-Segment-Anzeige ansteuern, sondern ganze Wörter (bis zu 20 Zeichen) ausgeben können. Es muss also zwischen kurzen Pausen (zwischen Buchstaben) und langen Pausen (zwischen Wörtern) unterschieden werden. Die Modellierung soll auf RT-Ebene statt finden und zyklengenau sein. Außerdem sollen die Zeichen A-Z und die Zahlen 0-9 des Morsealphabets unterstützt werden, welche hier aufgelistet sind:

a	.–	i	..	r	.–.
b	–...	j	.–––	s	...
c	–.–.	k	–.–	t	–
d	–..	l	.–..	u	..–
e	.	m	––	v	...–
e	..–..	n	–.	w	.––
f	..–.	o	–––	x	–..–
g	––.	p	.––.	y	–.––
h	....	q	––.–	z	––..

1	.––––	6	–....
2	..–––	7	––...
3	...––	8	––..
4	....–	9	––––.
5	.....	0	–––––

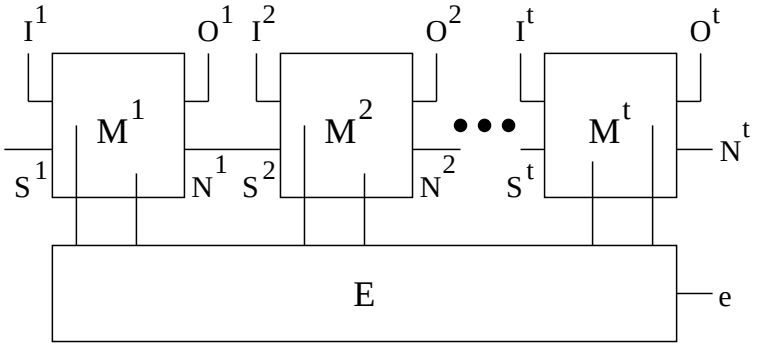
Aufgabe 2

SAT Beweiser - SAT Beweiser kommen in vielen Gebieten des Schaltkreisentwurfes bzw. der Verifikation zum Einsatz. Ein SAT Beweiser liest eine Boolesche Formel f in konjunktiver Normalform (CNF) im DIMACS-Format[1] ein und gibt als Ergebnis eine erfüllende Belegung von f zurück oder beweist, dass keine solche Belegung existiert. Der SAT-Beweiser soll als grundlegenden Algorithmus den DPLL-Algorithmus nutzen und ist in C++ zu implementieren. Da SAT Beweiser zum Teil sehr große Schaltungen verarbeiten müssen, soll die Implementierung möglichst speichereffizient arbeiten.

[1] http://www.cs.ubc.ca/ hoos/SATLIB/Benchmarks/SAT/satformat.ps

Aufgabe 3

Model Checker - Um Eigenschaften in Schaltungen zu prüfen, soll ein Model Checker entwickelt werden. Dieser soll sequentielle Schaltungen im blif-Format[2] (andere Formate sind in Absprache auch möglich) einlesen und in diesen selbst generierte temporale Eigenschaften (z.B. mit SAT oder BDDs) prüfen. Die Eigenschaften sollen als Boolescher Ausdruck vorliegen. Die Syntax der Ausdrücke kann frei gewählt werden. Folgendes Bild soll die Aufgabe verdeutlichen:



Um eine temporale Eigenschaft E über t Zeittakte prüfen zu können, muss der Schaltkreis M über t Zeittakte abgerollt werden. Der Zustand der Schaltung wird hier durch die Variablen $S^1 \dots S^t$ bzw. $N^1 \dots N^t$ repräsentiert. Die Eingänge und Ausgänge der Schaltung sind hingegen durch $I^1 \dots I^t$ bzw. $O^1 \dots O^t$ gegeben. Der Ausgang e stellt die Gültigkeit der Eigenschaft E dar.

Beispiel: Die Variable $x17$ soll in Zeitpunkt $t = 2$ immer 0 sein, falls im Zeitpunkt 1 $x16$ gleich $x15$ ist. Die folgende Formulierung wäre möglich:

```
.property x16@t1 = x15@t1 -> x17@t2 = 0
```

Gilt die Eigenschaft soll gelten $e = 1$, andernfalls $e = 0$. Der Model Checker soll in C++ implementiert werden. Für die symbolischen Operationen kann ein externe SAT-Engine oder ein BDD Paket eingebunden werden.

Abgabe der Implementierung mit Dokumentation und Testfällen: 04.07.2008

[2] http://embedded.eecs.berkeley.edu/research/vis/blif.ps