



Technische Universität Dresden
Fakultät Elektrotechnik
Institut für Grundlagen der Elektrotechnik und Elektronik

Beleg zur Vorlesung Schaltkreis- und Systementwurf

Entwurf eines diskreten n-fachen Differenzier- und Integrier-ASICs

Thomas Liske

Dresden, 2005

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. René Schüffny

Inhaltsverzeichnis

1	Einleitung	4
2	Algorithmus	5
2.1	Speicherlayout	5
2.2	Blockdiagramm	6
3	Datenpfad	7
4	Register-Transfer-Folge	8
5	Zustandsgraph	9
6	Logikverknüpfungen	10
6.1	Automatenlogik	10
6.2	Steuerlogik	12
7	Entwurf	16
7.1	Datenpfad	16
7.1.1	Arithmetic and Logic Unit	16
7.2	Steuerwerk	16
7.2.1	Automatenlogik	17
7.2.2	Steuerlogik	17
7.3	Top-Zelle	17
7.4	Pad-Zelle	17
8	Simulation	18
8.1	Arithmetic and Logic Unit	18
8.2	Steuerwerk	18
8.2.1	Automatenlogik	18
8.2.2	Steuerlogik	18
8.3	Gesamtsimulation	18
9	Zusammenfassung	19
A	Quellcode	20
A.1	fsm.pla	20

Inhaltsverzeichnis

A.2	ctrl.pla	21
A.3	Steuerlogik (functional)	21
A.4	Test Bench: ALU	26
A.5	Test Bench: Steuerlogik	27
B	Schematics	30
B.1	Arithmetic and Logic Unit	30
B.1.1	ADD32	31
B.1.2	ADD16	31
B.1.3	PCL2	32
B.1.4	ADD4	32
B.1.5	PCL4	32
B.1.6	FA	33
B.2	Datenpfad	33
B.3	Steuerwerk	34
B.4	Automatenlogik	35
B.5	Steuerlogik	36
B.6	Top-Zelle	37
B.7	Pad-Zelle	37
C	Simulationsdiagramme	38
C.1	Arithmetic and Logic Unit	38
C.2	Steuerlogik	39
D	Abbildungsverzeichnis	41
E	Tabellenverzeichnis	42
F	Literaturverzeichnis	43

1 Einleitung

Für Studenten des Studienganges *Informationssystemtechnik* ist die Pflichtvorlesung *Schaltkreis- und Systementwurf* im Hauptstudium zu besuchen.

Die Vorlesung behandelt mögliche Entwurfsverfahren für Integrierte Schaltungen (ASICs¹). Parallel zur Vorlesung findet ein Praktikum statt, bei dem die Studenten für selbst gewählte Algorithmen eigene ASICs entwerfen und simulieren sollen.

Als CMOS-Technologie kommt *Austria Microsystems CSI 0.35um* mit den Softwarepaketen *AMS Design-Kit*, *Cadence* und *Synopsys* zum Einsatz. Zusätzlich steht eine ausführliche Praktikumsanleitung [Aps03] zur Verfügung.

¹Application Specific Integrated Circuit

2 Algorithmus

Als Funktion des hier zu realisierenden ASICs wurde ein n -facher Differentierer und Integrierer für eine diskrete Folge von ganzen Zahlen gewählt. Die Daten sollen dabei aus einem externen Speicher ausgelesen und das Ergebnis in selbigen wieder zurückgeschrieben werden.

2.1 Speicherlayout

In dem vom ASIC erwarteten Speicherlayout (Abb. 2.1) gibt der Betrag von R die Anzahl der Differentiations- bzw. Integrationsdurchläufe an. Für positive R arbeitet der ASIC als Integrierer, für negative R als Differentierer. Das Datum N gibt die Anzahl der für die Differentiation bzw. Integration zu verwendenden Daten $x_i (i = 1..N)$ an.

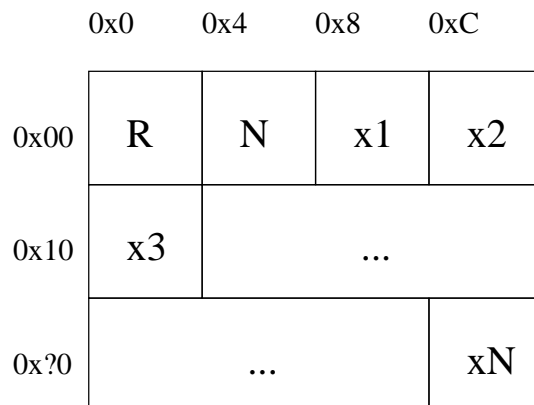


Abbildung 2.1: Speicherlayout

2.2 Blockdiagramm

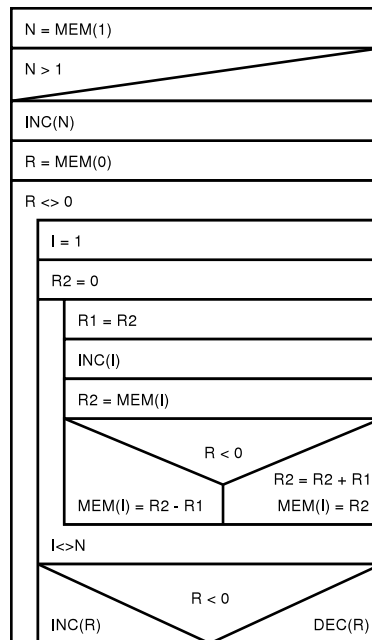


Abbildung 2.2: Blockdiagramm

Der Algorithmus wurde auf hinsicht geringer Nutzung der ALU¹ insbesondere durch Flag-Erhaltung konzipiert. So muß der Vergleich $R < 0$ nur in der inneren Schleife durchgeführt werden, denn das Sign-Flag der ALU bleibt bis zum Ende der äußeren Schleife erhalten.

Im Blockdiagramm (Abb. 2.2) außerdem nicht enthalten ist der Fall das beim Berechnen des Differentials oder Integrals ein Integer-Überlauf auftreten kann. In diesem Fall wird das Abarbeiten des Algorithmuses beendet und der Ausgabe-PIN *error_o* des ASICs wird auf high gesetzt. Der Speicherinhalt befindet sich dann in einem nicht definierten Zustand.

¹Arithmetic and Logic Unit

3 Datenpfad

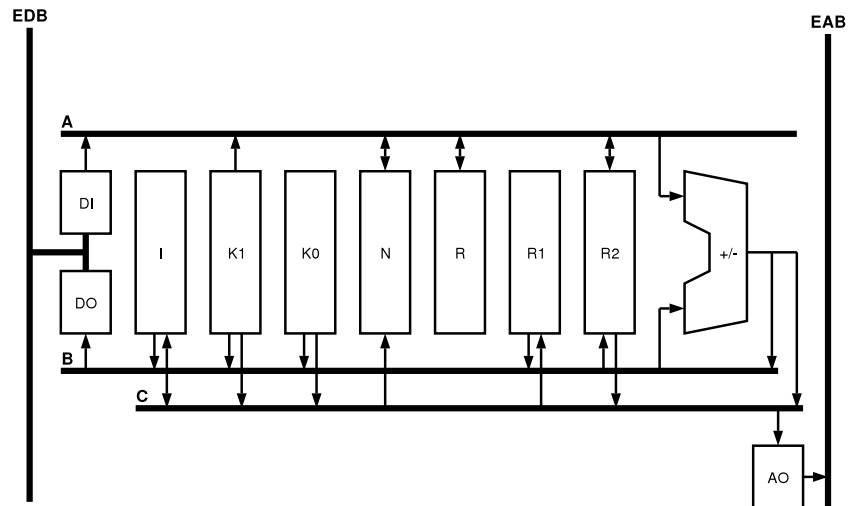


Abbildung 3.1: Datenpfad

Der Datenpfad (Abb. 3.1) für diesen ASIC enthält drei interne Datenbusse. Der externe Adreßbus (*EAB*) und externe Datenbus (*EDB*) wird über Ein- und Ausgabepuffer angeschlossen. Hinzu kommen noch Register für die Variablen und Konstanten sowie die ALU.

4 Register-Transfer-Folge

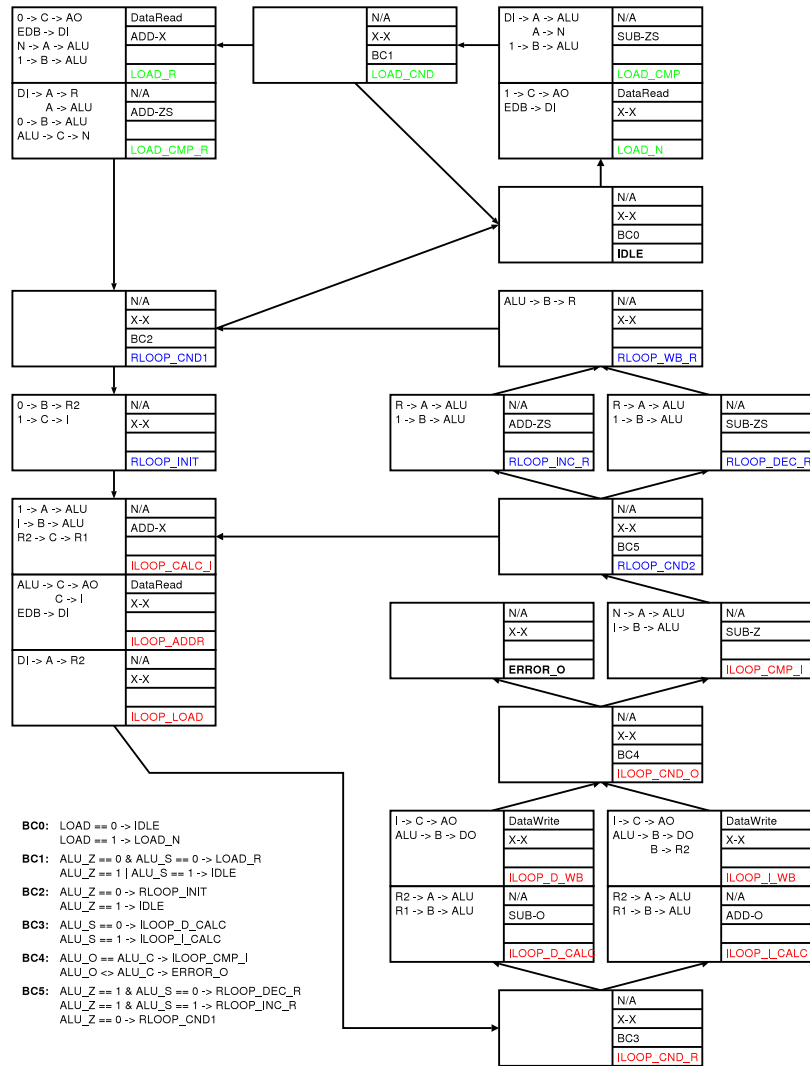


Abbildung 4.1: RT-Folge

5 Zustandsgraph

Die RT-Folge setzt sich aus 23 Zuständen zusammen. Für eine binäre Zustandskodierung mit Flip-Flops folgt daraus das fünf Flip-Flops benötigt werden. Damit lassen sich $2^5 = 32$ Zustände darstellen, die restlichen 9 nicht verwendeten Zustände (*DUMMY0* - *DUMMY8*) sollen immer in den Anfangszustand (*IDLE*) führen (siehe Zustandsgraph (Abb. 5.1)).

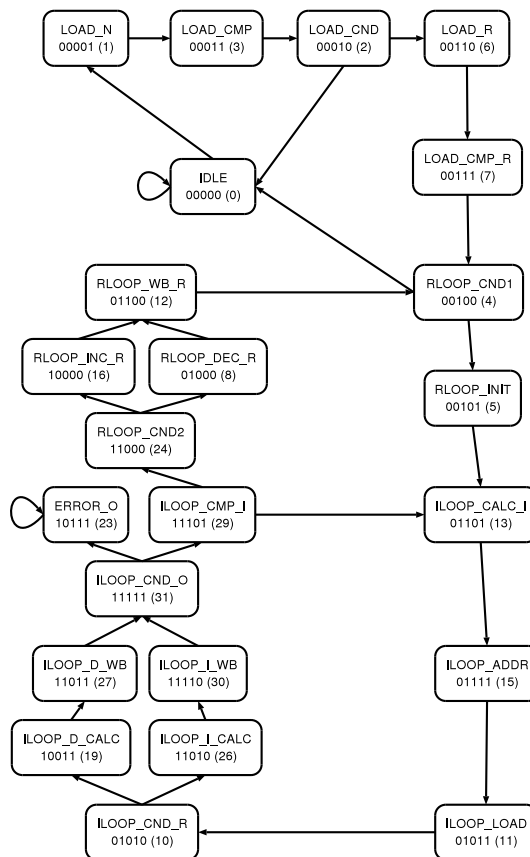


Abbildung 5.1: Zustandsgraph

Die Zustandskodierung wurde so umgesetzt das zwischen zwei Zuständen sich möglichst nur ein Bit ändert. Aufgrund der *DUMMY*-Zustände und da der Zustandsgraph (Abb. 5.1) mehrere Zyklen mit einer geraden Anzahl von Knoten besitzt läßt sich dies nicht für alle Übergänge realisieren.

6 Logikverknüpfungen

6.1 Automatenlogik

Aus Zustandsgraph (Abb. 5.1) und RT-Folge (Abb. 4.1) ergibt sich die Zustandsübergangstabelle (6.1).

Von dieser läßt sich die Tabelle zur Ansteuerung der Flip-Flop-Eingänge (Tab. 6.2) ableiten.

Zustand _i	Zustand _{i+1}	j ₄	k ₄	j ₃	k ₃	j ₂	k ₂	j ₁	k ₁	j ₀	k ₀
IDLE	IDLE	0	-	0	-	0	-	0	-	0	-
IDLE	LOAD_N	0	-	0	-	0	-	0	-	1	-
LOAD_N	LOAD_CMP	0	-	0	-	0	-	1	-	-	0
LOAD_CMP	LOAD_CND	0	-	0	-	0	-	-	0	-	1
LOAD_CND	LOAD_R	0	-	0	-	1	-	-	0	0	-
LOAD_CND	IDLE	0	-	0	-	0	-	-	1	0	-
LOAD_CND	IDLE	0	-	0	-	0	-	-	1	0	-
LOAD_R	LOAD_CMP_R	0	-	0	-	-	0	-	0	1	-
LOAD_CMP_R	RLOOP_CND1	0	-	0	-	-	0	-	1	-	1
RLOOP_CND1	RLOOP_INIT	0	-	0	-	-	0	0	-	1	-
RLOOP_CND1	IDLE	0	-	0	-	-	1	0	-	0	-
RLOOP_INIT	ILOOP_CALC_I	0	-	1	-	-	0	0	-	-	0
ILOOP_CALC_I	ILOOP_ADDR	0	-	-	0	-	0	1	-	-	0
ILOOP_ADDR	ILOOP_LOAD	0	-	-	0	-	1	-	0	-	0
ILOOP_LOAD	ILOOP_CND_R	0	-	-	0	0	-	-	0	-	1
ILOOP_CND_R	ILOOP_D_CALC	1	-	-	1	0	-	-	0	1	-
ILOOP_CND_R	ILOOP_I_CALC	1	-	-	0	0	-	-	0	0	-
ILOOP_D_CALC	ILOOP_D_WB	-	0	1	-	0	-	-	0	-	0
ILOOP_D_WB	ILOOP_CND_O	-	0	-	0	1	-	-	0	-	0
ILOOP_I_CALC	ILOOP_I_WB	-	0	-	0	1	-	-	0	0	-
ILOOP_I_WB	ILOOP_CND_O	-	0	-	0	-	0	-	0	1	-
ILOOP_CND_O	ILOOP_CMP_I	-	0	-	0	-	0	-	1	-	0
ILOOP_CND_O	ERROR_O	-	0	-	1	-	0	-	0	-	0
ERROR_O	ERROR_O	-	0	0	-	-	0	-	0	-	0
ILOOP_CMP_I	RLOOP_CND2	-	0	-	0	-	1	0	-	-	1
RLOOP_CND2	ILOOP_CALC_I	-	1	-	0	1	-	0	-	1	-
RLOOP_CND2	RLOOP_DEC_R	-	1	-	0	0	-	0	-	0	-
RLOOP_CND2	RLOOP_INC_R	-	0	-	1	0	-	0	-	0	-
RLOOP_DEC_R	RLOOP_WB_R	0	-	-	0	1	-	0	-	0	-
RLOOP_INC_R	RLOOP_WB_R	-	1	1	-	1	-	0	-	0	-
RLOOP_WB_R	RLOOP_CND1	0	-	-	1	-	0	0	-	0	-
DUMMY0	IDLE	0	-	-	1	0	-	0	-	-	1
DUMMY1	IDLE	0	-	-	1	-	1	-	1	0	-
DUMMY2	IDLE	-	1	0	-	0	-	0	-	-	1
DUMMY3	IDLE	-	1	0	-	0	-	-	1	0	-
DUMMY4	IDLE	-	1	0	-	-	1	0	-	0	-
DUMMY5	IDLE	-	1	0	-	-	1	0	-	-	1
DUMMY6	IDLE	-	1	0	-	-	1	-	1	0	-
DUMMY7	IDLE	-	1	-	1	0	-	0	-	0	1
DUMMY8	IDLE	-	1	-	1	-	1	0	-	0	-

Tabelle 6.2: Ansteuerung der Flip-Flop-Eingänge

Die aus der Zustandsübergangstabelle und der Tabelle zur Ansteuerung der Flip-Flop-Eingänge sich ableitenden Logikverknüpfungen wurden mittels dem Softwarepaketes *SIS* vereinfacht.

Dazu wurde die beiden Tabellen zusammengefaßt und in eine Beschreibung für ein

Zustand _i	Q _{4i}	Q _{3i}	Q _{2i}	Q _{1i}	Q _{0i}	Z _i	S _i	O _i	load _i	Zustand _{i+1}	Q _{4i+1}	Q _{3i+1}	Q _{2i+1}	Q _{1i+1}	Q _{0i+1}
IDLE	0	0	0	0	0	-	-	-	0	IDLE	0	0	0	0	0
IDLE	0	0	0	0	0	-	-	-	1	LOAD_N	0	0	0	0	1
LOAD_N	0	0	0	0	1	-	-	-	-	LOAD_CMP	0	0	0	1	1
LOAD_CMP	0	0	0	1	1	-	-	-	-	LOAD_CND	0	0	0	1	0
LOAD_CND	0	0	0	1	0	0	0	-	-	LOAD_R	0	0	1	1	0
LOAD_CND	0	0	0	1	0	1	-	-	-	IDLE	0	0	0	0	0
LOAD_R	0	0	0	1	0	-	1	-	-	IDLE	0	0	0	0	0
LOAD_CMP_R	0	0	1	1	0	-	-	-	-	LOAD_CMP_R	0	0	1	1	1
RLOOP_CND1	0	0	1	1	1	-	-	-	-	RLOOP_CND1	0	0	1	0	0
RLOOP_CND1	0	0	1	0	0	0	-	-	-	RLOOP_INIT	0	0	1	0	1
RLOOP_INIT	0	0	1	0	0	1	-	-	-	IDLE	0	0	0	0	0
ILOOP_CALC_I	0	1	1	0	1	-	-	-	-	ILOOP_CALC_I	0	1	1	0	1
ILOOP_ADDR_I	0	1	1	1	1	-	-	-	-	ILOOP_ADDR_I	0	1	1	1	1
ILOOP_LOAD	0	1	0	1	1	-	-	-	-	ILOOP_LOAD	0	1	0	1	1
ILOOP_CND_R	0	1	0	1	0	-	0	-	-	ILOOP_CND_R	0	1	0	1	0
ILOOP_CND_R	0	1	0	1	0	-	1	-	-	ILOOP_D_CALC	1	0	0	1	1
ILOOP_D_CALC	1	0	0	1	1	-	-	-	-	ILOOP_I_CALC	1	1	0	1	0
ILOOP_D_WB	1	1	0	1	1	-	-	-	-	ILOOP_D_WB	1	1	0	1	1
ILOOP_I_CALC	1	1	0	1	0	-	-	-	-	ILOOP_CND_O	1	1	1	1	1
ILOOP_I_WB	1	1	1	1	0	-	-	-	-	ILOOP_I_WB	1	1	1	1	0
ILOOP_CND_O	1	1	1	1	1	-	-	0	-	ILOOP_CND_O	1	1	1	1	1
ILOOP_CND_O	1	1	1	1	1	-	-	1	-	ILOOP_CMP_I	1	1	1	0	1
ERROR_O	1	0	1	1	1	-	-	-	-	ERROR_O	1	0	1	1	1
ERROR_O	1	0	1	1	1	-	-	-	-	ERROR_O	1	0	1	1	1
ILOOP_CMP_I	1	1	1	0	1	-	-	-	-	RLOOP_CND2	1	1	0	0	0
RLOOP_CND2	1	1	0	0	0	0	-	-	-	ILOOP_CALC_I	0	1	1	0	1
RLOOP_CND2	1	1	0	0	0	1	0	-	-	RLOOP_DEC_R	0	1	0	0	0
RLOOP_CND2	1	1	0	0	0	1	1	-	-	RLOOP_INC_R	1	0	0	0	0
RLOOP_DEC_R	0	1	0	0	0	-	-	-	-	RLOOP_WB_R	0	1	1	0	0
RLOOP_INC_R	1	0	0	0	0	-	-	-	-	RLOOP_WB_R	0	1	1	0	0
RLOOP_WB_R	0	1	1	0	0	-	-	-	-	RLOOP_WB_R	0	1	1	0	0
DUMMY0	0	1	0	0	1	-	-	-	-	RLOOP_CND1	0	0	1	0	0
DUMMY1	0	1	1	1	0	-	-	-	-	IDLE	0	0	0	0	0
DUMMY2	1	0	0	0	1	-	-	-	-	IDLE	0	0	0	0	0
DUMMY3	1	0	0	1	0	-	-	-	-	IDLE	0	0	0	0	0
DUMMY4	1	0	1	0	0	-	-	-	-	IDLE	0	0	0	0	0
DUMMY5	1	0	1	0	1	-	-	-	-	IDLE	0	0	0	0	0
DUMMY6	1	0	1	1	0	-	-	-	-	IDLE	0	0	0	0	0
DUMMY7	1	1	0	0	1	-	-	-	-	IDLE	0	0	0	0	0
DUMMY8	1	1	1	0	0	-	-	-	-	IDLE	0	0	0	0	0

Tabelle 6.1: Zustandsübergangstabelle

6 Logikverknüpfungen

PLA¹ (siehe A.1) überführt und mittels *read_pla* in *SIS* geladen. Anschließend wurden die *don't care* Ein- und Ausgänge mittels *full_simplify* eliminiert und gemeinsame Ausdrücke in den Termen mittels *gcx* herausgelöst. Dies führte zu folgendem Ergebnis:

```
UC Berkeley, SIS 1.3 (compiled 20-Mar-00 at 1:56 PM)
sis> read_pla fsm.pla
sis> full_simplify
sis> gcx -b
sis> print
j4 = [26] q0' q3
k4 = [21] + j2 q1' + k2 q0' + k2 q3' + k0 q2' + q0' q1' s'
j3 = [22] q0' q2' + [25] q0 + [26] q0 q4
k3 = j4 j2' s' + [25] j2' + [27] j2' s + j2' k2' k1' q0 q4 + k2 q0'
j2 = [22] k0' q3' + [22] q0' z' + [23] [27] k0' + [24] [26] + [28] k0' s' z'
k2 = j4' [23] q1 + [21] q1' z + [21] q4 + [22] q2
j1 = [20] q0 q2' + [23] q0 q2
k1 = [21] q2' s + [21] q2' z + [21] q4 + [23] q0' q2 + j1' v' q0 q2 q3 + k0 q2
j0 = j4 q4' s' + j4' [24] k0' q2' z' + j4' [24] q1 + [20] [27] load +
    [20] q2 z' + [28] q2
k0 = [22] q0 + [23] j1' q0 + [28] q0

[20] = q3' q4'
[21] = q0' q3'
[22] = q1' q4
[23] = q3 q4'
[24] = q3 q4
[25] = j1' q1' q4'
[26] = q1 q2'
[27] = q1' q2'
[28] = [20] q1
```

6.2 Steuerlogik

Aus Zustandsgraph (Abb. 5.1) und RT-Folge (Abb. 4.1) ergibt sich die Tabelle zur Ansteuerung des Datenpfades (Tab. 6.3) und die Ansteuerung der Ausgabe-PINs (Tab. 6.4).

¹Programmable Logic Array

Zustand	atu_mode	atu_to_b	atu_to_c	atu_wro	atu_wrs	atu_wrz	a_to_n	a_to_r	a_to_r2	b_to_r	b_to_r2	c_to_eab	c_to_i	c_to_n	c_to_r1	di_to_a	edb_to_di	i_to_b	i_to_c	k0_to_b	k0_to_c	k1_to_a	k1_to_b	k1_to_c	n_to_a	r1_to_b	r2_to_a	r2_to_c	r_to_a
IDLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LOAD_N	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LOAD_CND	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LOAD_CMP	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
RLOOP_CND1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RLOOP_INIT	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
LOAD_R	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
LOAD_CMP_R	0	0	1	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
RLOOP_DEC_R	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
DUMMY0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ILOOP_CND_R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ILOOP_LOAD	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
RLOOP_WB_R	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ILOOP_CALC_I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0
DUMMY1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ILOOP_ADDR	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
RLOOP_INC_R	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
DUMMY2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DUMMY3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ILOOP_D_CALC	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
DUMMY4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DUMMY5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DUMMY6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ERROR_O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RLOOP_CND2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DUMMY7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ILOOP_I_CALC	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
ILOOP_D_WB	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
DUMMY8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ILOOP_CMP_I	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
ILOOP_I_WB	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
ILOOP_CND_O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabelle 6.3: Ansteuerung des Datenpfades

6 Logikverknüpfungen

Zustand	ram wr en	ram rd en	ready	error o
IDLE	0	0	1	0
LOAD_N	0	1	0	0
LOAD_CND	0	0	0	0
LOAD_CMP	0	0	0	0
RLOOP_CND1	0	0	0	0
RLOOP_INIT	0	0	0	0
LOAD_R	0	1	0	0
LOAD_CMP_R	0	0	0	0
RLOOP_DEC_R	0	0	0	0
DUMMY0	0	0	0	0
ILOOP_CND_R	0	0	0	0
ILOOP_LOAD	0	0	0	0
RLOOP_WB_R	0	0	0	0
ILOOP_CALC_I	0	0	0	0
DUMMY1	0	0	0	0
ILOOP_ADDR	0	1	0	0
RLOOP_INC_R	0	0	0	0
DUMMY2	0	0	0	0
DUMMY3	0	0	0	0
ILOOP_D_CALC	0	0	0	0
DUMMY4	0	0	0	0
DUMMY5	0	0	0	0
DUMMY6	0	0	0	0
ERROR_O	0	0	0	1
RLOOP_CND2	0	0	0	0
DUMMY7	0	0	0	0
ILOOP_I_CALC	0	0	0	0
ILOOP_D_WB	1	0	0	0
DUMMY8	0	0	0	0
ILOOP_CMP_I	0	0	0	0
ILOOP_I_WB	1	0	0	0
ILOOP_CND_O	0	0	0	0

Tabelle 6.4: Ansteuerung der Ausgabe-PINs

Die sich daraus ableitenden Logikverknüpfungen wurden auch hier wieder in eine Beschreibung für ein PLA (siehe A.2) überführt und mittels *SIS* vereinfacht:

```
UC Berkeley, SIS 1.3 (compiled 20-Mar-00 at 1:56 PM)
sis> read_pla ctrl.pla
sis> full_simplify
sis> gcx -b
sis> print
alu_mode = [41] q1 q3' + [47] + [51]
alu_to_b = [36] q2 + [39] + [49]
alu_to_c = [40] [54]
alu_wro = [46] + [48]
alu_wrs = [42] + [43] + [47]
alu_wrz = [42] + [43] + [47] + [51]
a_to_n = [42] q2'
a_to_r = [42] q2
a_to_r2 = [40] [41] q3
b_to_r = [36] q2
b_to_r2 = [39] + [45] q2
c_to_eab = [37] + [39] + [49] + [50] + [55]
c_to_i = [45] q2 + [50]
c_to_n = [42] q2
c_to_r1 = [53]
di_to_a = [40] [41] + [42]
edb_to_di = [37] + [50] + [55]
i_to_b = [44]
i_to_c = [39] + [49]
k0_to_b = [33] q2
k0_to_c = [37]
k1_to_a = [53]
k1_to_b = [37] + [42] q2' + [43] + [47]
k1_to_c = [45]
n_to_a = [37] + [51]
r1_to_b = [46] + [48]
r2_to_a = [46] + [48]
```

6 Logikverknüpfungen

```
r2_to_c = [53]
r_to_a  = [43] + [47]
ram_wr_en = [39] + [49]
ram_rd_en = [37] + [50] + [55]
ready    = [52] q2' q3'
error_o   = [54] q1 q3' q4
```

```
[33] = q0 q3' q4'
[34] = q1 q3 q4
[35] = [54] q3
[36] = [52] q3
[37] = [40] q0' q2 q3'
[38] = q2' q3' q4
[39] = [34] q0' q2
[40] = q1 q4'
[41] = q0 q2'
[42] = [33] q1
[43] = [38] q0' q1'
[44] = [35] q1'
[45] = [33] q1'
[46] = [34] q0' q2'
[47] = [36] q2'
[48] = [38] q0 q1
[49] = [34] [41]
[50] = [35] [40]
[51] = [44] q4
[52] = q0' q1' q4'
[53] = [44] q4'
[54] = q0 q2
[55] = [45] q2'
```

7 Entwurf

Zum Entwurf wurde der Automat in eine Zellenhierarchie (Abb. 7.1) aufgeteilt. Durch diesen Ansatz lassen sich die einzelnen Zellen isoliert simulieren ohne Fehlereinflüsse durch andere Zellen. Die einzelnen Zellen sollen hier kurz vorgestellt werden.

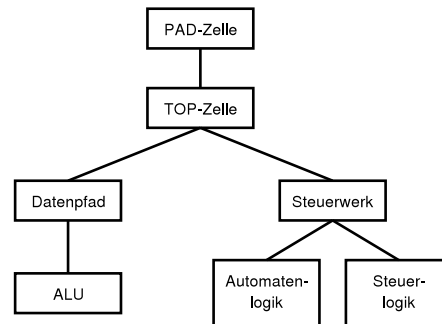


Abbildung 7.1: Zellenhierarchie

7.1 Datenpfad

Der Datenpfad (Abb. B.2) enthält die Register und die ALU welche mittels dreier Busse verbunden sind (vergl. 3).

7.1.1 Arithmetic and Logic Unit

Die zu entwerfende ALU muß Addition, Subtraktion und die Generierung der Flags *Overflow*, *Sign* und *Zero* unterstützen. Um die nötige Berechnungszeit zu minimieren wird die Wahl auf einen Addierer mit paralleler Übertragungslogik [Hst02] (*Carry look-ahead Addierer*).

Über den Eingang *alu_mode* kann die ALU von Addition (*alu_mode* = 0) auf Subtraktion (*alu_mode* = 1) umgeschaltet werden. Über die Eingänge *alu_wrc*, *alu_wro*, *alu_wrs* und *alu_wrz* kann das Erhalten (= 0) oder Setzen (= 1) der Flags *Carry*, *Overflow*, *Sign* und *Zero* festgelegt werden.

Die 32bit ALU (Abb. B.1) ist aus zwei 16bit Addierern (Abb. B.1.2) aufgebaut die mittels der Zelle PCL2 (Abb. B.1.3) parallel angesteuert werden. Die 16bit Addierer bestehen je aus vier 4bit Addierern (Abb. B.1.4) die mittels der Zelle PCL4 (Abb. B.1.5) parallel angesteuert werden. In den 4bit Addierern kommen wiederum je eine PCL4 Zelle und vier Volladdierer (Abb. B.1.6) mit *Propagate* und *Generate* Ausgängen zum Einsatz.

Das *Sign* Flag wird in der ALU Zelle direkt aus dem Ergebnis abgeleitet. Für das *Zero* Flag führen alle Addierer Zellen den Ausgang *ZN* heraus der den Wert logisch Wahr annimmt, wenn mindestens ein Bit im Ergebnis gesetzt ist - dieser wird dann negiert und als *Zero* Flag ausgegeben.

7.2 Steuerwerk

Das Steuerwerk (Abb. B.3) enthält den Zustandsspeicher und übernimmt die Ansteuerung des Datenpfades in Abhängigkeit des aktuellen Zustandes, der ALU-Flags und der Eingangs-PINs *load* und *reset*.

7.2.1 Automatenlogik

Die Automatenlogik (Abb. B.4) enthält die JK-Flip-Flops sowie die nötigen Logikgatter zur Ansteuerung der Eingänge der Flip-Flops zum Erreichen des nächsten Zustandes (vergl. 6.1).

Die Flip-Flops sind taktgesteuert und Schalten bei der positiven Flanke des Taktsignales. Beim Anlegen von logisch 1 auf den *reset* Eingangs-PIN werden die Flip-Flop-Speicher auf den Wert logisch 0 gesetzt (Zustand *IDLE*).

7.2.2 Steuerlogik

Die Steuerlogik (Abb. B.5) schaltet in Abhängigkeit des aktuellen Zustands die Tristate-Treiber sowie die ALU des Datenpfades. Dafür besteht die Schaltung nur aus logischen Gattern (vergl. 6.2).

7.3 Top-Zelle

Die Top-Zelle (Abb. B.6) verbindet das Steuerwerk mit dem Datenpfad und fast den Eingang sowie den Ausgang des externen Datenbusses zu einem bidirektionalen PIN zusammen.

7.4 Pad-Zelle

Die Pad-Zelle (Abb. B.7) enthält neben der Top-Zelle zusätzlich die Pad-Zellen und ist ansonsten PIN kompatibel zur Top-Zelle.

8 Simulation

8.1 Arithmetic and Logic Unit

Als Test Bench (siehe A.4) für die ALU (vergl. 7.1.1) wurde hauptsächlich die Flag-Generierung geprüft (siehe C.1).

8.2 Steuerwerk

8.2.1 Automatenlogik

Eine isolierte Simulation der Automatenlogik wurde nur begrenzt durchgeführt, da es von außen nicht vorgesehen ist den Automaten in einen definierten Zustand (außer *IDLE*) zu versetzen. So wurde nur das korrekte sequentielle abarbeiten der Zustandsübergänge kontrolliert.

8.2.2 Steuerlogik

Die Simulation (siehe A.5) der Steuerlogik (vergl. 6.2) erfolgte in der *functional* und *schematic* Beschreibung. Wenn man die Resultate (siehe C.2) vergleicht, stellt man fest, das in der *schematic* Beschreibung mehrere Spikes auftreten. Dies wird verursacht durch die unterschiedlich langen Signallaufzeiten durch die Logikkaskade zwischen dem Eingangs-PIN *state* und den Ausgangs-PINs. Da der Automat getaktet ist und ein Takt wesentlich länger dauert als das Auftreten der Spikes stellt dies kein Problem dar.

8.3 Gesamtsimulation

Nach den erfolgreichen Einzeltests wurde der komplette Automat mittels der Top-Zelle und verschiedenen Inhalten des Speichers simuliert. Dabei konnten keine Fehler entdeckt werden.

9 Zusammenfassung

Die Beschreibung durch *Verilog* gibt einem ein mächtiges Werkzeug in die Hand, welches jederzeit ermöglicht seine aufgebauten funktionalen Zellen einzeln oder zusammengesetzt zu testen bzw. mit einer formalen Beschreibung zu vergleichen.

Durch das Einführungsbeispiel der Lehrveranstaltung wurde ein hinreichender Einblick in die Funktionsweise der verwendeten Tools gegeben auf dessen Basis der eigene Entwurf ohne größeren Probleme durchgeführt werden konnte.

A Quellcode

A.1 fsm.pla

```
.i 9
.o 10
.ilb q4 q3 q2 q1 q0 z s o load
.olb j4 k4 j3 k3 j2 k2 j1 k1 j0 k0
.type fdr
00000---0 0-0-0-0-0-
00000---1 0-0-0-0-1-
00001---- 0-0-0-1--0
00011---- 0-0-0--0-1
0001000-- 0-0-1--00-
000101--- 0-0-0--10-
00010-1-- 0-0-0--10-
00110---- 0-0--0-01-
00111---- 0-0--0-1-1
001000--- 0-0--00-1-
001001--- 0-0--10-0-
00101---- 0-1--00--0
01101---- 0--0-01--0
01111---- 0--0-1-0-0
01011---- 0--00--0-1
01010-0-- 1--10--01-
01010-1-- 1--00--00-
10011---- -01-0--0-0
11011---- -0-01--0-0
11010---- -0-01--00-
11110---- -0-0-0-01-
11111--0- -0-0-0-1-0
11111--1- -0-1-0-0-0
10111---- -00--0-0-0
11101---- -0-0-10--1
110000--- -1-01-0-1-
1100010-- -1-00-0-0-
1100011-- -0-10-0-0-
01000---- 0--01-0-0-
10000---- -11-1-0-0-
01100---- 0--1-00-0-
01001---- 0--10-0--1
01110---- 0--1-1-10-
10001---- -10-0-0--1
10010---- -10-0--10-
10100---- -10--10-0-
10101---- -10--10--1
10110---- -10--1-10-
11001---- -1-10-0-01
11100---- -1-1-10-0-
```


A Quellcode

```
    ram_wr_en, ram_rd_en,
    ready, error_o,
    state
);

parameter OFF      = 1'b0;
parameter ON       = 1'b1;

parameter IDLE     = 5'b00000;
parameter LOAD_N   = 5'b00001;
parameter LOAD_CMP = 5'b00011;
parameter LOAD_CND = 5'b00010;
parameter LOAD_R   = 5'b00110;
parameter LOAD_CMP_R = 5'b00111;
parameter RLOOP_CND1 = 5'b00100;
parameter RLOOP_INIT = 5'b00101;
parameter ILOOP_CALC_I = 5'b01101;
parameter ILOOP_ADDR = 5'b01111;
parameter ILOOP_LOAD = 5'b01011;
parameter ILOOP_CMP_R = 5'b01010;
parameter ILOOP_CND_R = 5'b01010;
parameter ILOOP_D_CALC = 5'b10011;
parameter ILOOP_D_WB = 5'b11011;
parameter ILOOP_I_CALC = 5'b11010;
parameter ILOOP_I_WB = 5'b11110;
parameter ILOOP_CND_O = 5'b11111;
parameter ERROR_O = 5'b10111;
parameter ILOOP_CMP_I = 5'b11101;
parameter RLOOP_CND2 = 5'b11000;
parameter RLOOP_INC_R = 5'b10000;
parameter RLOOP_DEC_R = 5'b01000;
parameter RLOOP_WB_R = 5'b01100;

input [4:0] state;

output alu_mode;
output alu_to_b;
output alu_to_c;
output alu_wro;
output alu_wrs;
output alu_wrz;
output a_to_n;
output a_to_r;
output a_to_r2;
output b_to_r;
output b_to_r2;
output c_to_eab;
output c_to_i;
output c_to_n;
output c_to_r1;
output di_to_a;
output edb_to_di;
output i_to_b;
output i_to_c;
output k0_to_b;
output k0_to_c;
output k1_to_a;
output k1_to_b;
output k1_to_c;
output n_to_a;
output r1_to_b;
output r2_to_a;
```

A Quellcode

```
output r2_to_c;
output r_to_a;

output ram_wr_en;
output ram_rd_en;
output ready;
output error_o;

reg alu_mode;
reg alu_to_b;
reg alu_to_c;
reg alu_wro;
reg alu_wrs;
reg alu_wrz;
reg a_to_n;
reg a_to_r;
reg a_to_r2;
reg b_to_r;
reg b_to_r2;
reg c_to_eab;
reg c_to_i;
reg c_to_n;
reg c_to_r1;
reg di_to_a;
reg edb_to_di;
reg i_to_b;
reg i_to_c;
reg k0_to_b;
reg k0_to_c;
reg k1_to_a;
reg k1_to_b;
reg k1_to_c;
reg n_to_a;
reg r1_to_b;
reg r2_to_a;
reg r2_to_c;
reg r_to_a;

reg ram_wr_en;
reg ram_rd_en;
reg ready;
reg error_o;

always @ (state)
begin
    alu_mode      = OFF;
    alu_to_b     = OFF;
    alu_to_c     = OFF;
    alu_wro      = OFF;
    alu_wrs      = OFF;
    alu_wrz      = OFF;
    a_to_n       = OFF;
    a_to_r       = OFF;
    a_to_r2      = OFF;
    b_to_r       = OFF;
    b_to_r2      = OFF;
    c_to_eab     = OFF;
    c_to_i       = OFF;
    c_to_n       = OFF;
    c_to_r1      = OFF;
    di_to_a      = OFF;
```

A Quellcode

```
edb_to_di    = OFF;
i_to_b       = OFF;
i_to_c       = OFF;
k0_to_b      = OFF;
k0_to_c      = OFF;
k1_to_a      = OFF;
k1_to_b      = OFF;
k1_to_c      = OFF;
n_to_a       = OFF;
r1_to_b      = OFF;
r2_to_a      = OFF;
r2_to_c      = OFF;
r_to_a       = OFF;

ram_wr_en    = OFF;
ram_rd_en    = OFF;
ready        = OFF;
error_o      = OFF;

case (state)
  IDLE       : ready = ON;
  LOAD_N     : begin
                k1_to_c = ON;
                c_to_eab = ON;
                edb_to_di = ON;
                ram_rd_en = ON;
              end
  LOAD_CMP   : begin
                di_to_a = ON;
                a_to_n = ON;
                k1_to_b = ON;
                alu_wrs = ON;
                alu_wrz = ON;
                alu_mode = ON;
              end
  LOAD_R     : begin
                k0_to_c = ON;
                c_to_eab = ON;
                n_to_a = ON;
                k1_to_b = ON;
                edb_to_di = ON;
                ram_rd_en = ON;
              end
  LOAD_CMP_R : begin
                di_to_a = ON;
                a_to_r = ON;
                k0_to_b = ON;
                alu_to_c = ON;
                c_to_n = ON;
                alu_wrz = ON;
                alu_wrs = ON;
              end
  RLOOP_INIT : begin
                k0_to_b = ON;
                b_to_r2 = ON;
                k1_to_c = ON;
                c_to_i = ON;
              end
  ILOOP_CALC_I : begin
                k1_to_a = ON;
                i_to_b = ON;
                r2_to_c = ON;
              end
endcase
```

A Quellcode

```

        c_to_r1 = ON;
    end
ILOOP_ADDR      :   begin
        alu_to_c = ON;
        c_to_eab = ON;
        c_to_i   = ON;
        edb_to_di = ON;
        ram_rd_en = ON;
    end
ILOOP_LOAD      :   begin
        di_to_a = ON;
        a_to_r2 = ON;
    end
ILOOP_I_CALC:   begin
        r2_to_a = ON;
        r1_to_b = ON;
        alu_wro = ON;
    end
ILOOP_I_WB      :   begin
        i_to_c = ON;
        c_to_eab = ON;
        alu_to_b = ON;
        b_to_r2 = ON;
        ram_wr_en = ON;
    end
ILOOP_D_CALC:   begin
        r2_to_a = ON;
        r1_to_b = ON;
        alu_mode = ON;
        alu_wro = ON;
    end
ILOOP_D_WB      :   begin
        i_to_c = ON;
        c_to_eab = ON;
        alu_to_b = ON;
        ram_wr_en = ON;
    end
ERROR_0         :   error_o = ON;
ILOOP_CMP_I     :   begin
        n_to_a = ON;
        i_to_b = ON;
        alu_mode = ON;
        alu_wrz = ON;
    end
RLOOP_INC_R     :   begin
        r_to_a = ON;
        k1_to_b = ON;
        alu_wrz = ON;
        alu_wrs = ON;
    end
RLOOP_DEC_R     :   begin
        r_to_a = ON;
        k1_to_b = ON;
        alu_mode = ON;
        alu_wrz = ON;
        alu_wrs = ON;
    end
RLOOP_WB_R      :   begin
        alu_to_b = ON;
        b_to_r   = ON;
    end
end
endcase
```

A Quellcode

```
end
endmodule
```

A.4 Test Bench: ALU

```
// Verilog HDL for "Beispiel", "ALU_tb" "functional"

module ALU_tb ;
    parameter CYCLE = 60;

    reg clk, reset;
    reg [31:0] A, B;
    reg sub;
    reg set_C, set_0, set_S, set_Z;

    wire [31:0] func_Y;
    wire func_C, func_0, func_S, func_Z;
    wire [31:0] schem_Y;
    wire schem_C, schem_0, schem_S, schem_Z;

    ALU_patch alu_functional (.A(A), .B(B), .sub(sub), .clk(clk), .reset(reset),
        .set_C(set_C), .set_0(set_0), .set_S(set_S), .set_Z(set_Z),
        .Y(func_Y), .C(func_C), .0(func_0), .S(func_S), .Z(func_Z));
    ALU32    alu_schematic (.A(A), .B(B), .sub(sub), .clk(clk), .reset(reset),
        .set_C(set_C), .set_0(set_0), .set_S(set_S), .set_Z(set_Z),
        .Y(schem_Y), .C(schem_C), .0(schem_0), .S(schem_S), .Z(schem_Z));

    initial clk = 1'b0;
    always #(CYCLE/2) clk = ~clk;

    initial begin
        reset = 1'b1;
        sub = 0;
        A = 0;
        B = 0;
        set_C = 1'b1;
        set_0 = 1'b1;
        set_S = 1'b1;
        set_Z = 1'b1;

        #(2*CYCLE) reset = 1'b0;

        #(2*CYCLE) sub = 1'b1; // 0+0
        #(2*CYCLE) sub = 1'b1; // 0-0

        #(2*CYCLE) B = 1; sub = 0; // 0+1
        #(2*CYCLE) B = -1; // 0+(-1)

        #(2*CYCLE) A = 32'h7fff_ffff; B = 0; // 2147483647+0
        #(2*CYCLE) B = 1; // 2147483647+1
        #(2*CYCLE) B = -1; // 2147483647+(-1)

        #(2*CYCLE) A = 32'h8000_0000; B = 0; // -2147483648+0
        #(2*CYCLE) B = 1; // -2147483648+1
        #(2*CYCLE) B = -1; // -2147483648+(-1)

        #(2*CYCLE) A = -1; // -1-1
        #(2*CYCLE) A = 2; B=-1; // 2+(-1)
        #(2*CYCLE) A = 2; B=1; sub=1; // 2-1
    end
endmodule
```

A Quellcode

```
        #(2*CYCLE) $finish;
    end
endmodule
```

A.5 Test Bench: Steuerlogik

```
// Verilog HDL for "n_diff_int", "n_diff_int_steuerwerk_ctrl_tb" "functional"
```

```
module n_diff_int_steuerwerk_ctrl_tb ;
    parameter CYCLE = 30;

    wire alu_mode;
    wire alu_to_b;
    wire alu_to_c;
    wire alu_wro;
    wire alu_wrs;
    wire alu_wrz;
    wire a_to_n;
    wire a_to_r;
    wire a_to_r2;
    wire b_to_r;
    wire b_to_r2;
    wire c_to_eab;
    wire c_to_i;
    wire c_to_n;
    wire c_to_r1;
    wire di_to_a;
    wire edb_to_di;
    wire i_to_b;
    wire i_to_c;
    wire k0_to_b;
    wire k0_to_c;
    wire k1_to_a;
    wire k1_to_b;
    wire k1_to_c;
    wire n_to_a;
    wire r1_to_b;
    wire r2_to_a;
    wire r2_to_c;
    wire r_to_a;

    wire ram_wr_en;
    wire ram_rd_en;
    wire ready;
    wire error_o;

    reg [4:0] state;

    n_diff_int_steuerwerk_ctrl n_diff_int_steuerwerk_ctrl_i(
        .state(state),
        .alu_mode(alu_mode),
        .alu_to_b(alu_to_b),
        .alu_to_c(alu_to_c),
        .alu_wro(alu_wro),
        .alu_wrs(alu_wrs),
        .alu_wrz(alu_wrz),
        .a_to_n(a_to_n),
        .a_to_r(a_to_r),
        .a_to_r2(a_to_r2),
```

A Quellcode

```
.b_to_r(b_to_r),
.b_to_r2(b_to_r2),
.c_to_eab(c_to_eab),
.c_to_i(c_to_i),
.c_to_n(c_to_n),
.c_to_r1(c_to_r1),
.di_to_a(di_to_a),
.edb_to_di(edb_to_di),
.i_to_b(i_to_b),
.i_to_c(i_to_c),
.k0_to_b(k0_to_b),
.k0_to_c(k0_to_c),
.k1_to_a(k1_to_a),
.k1_to_b(k1_to_b),
.k1_to_c(k1_to_c),
.n_to_a(n_to_a),
.r1_to_b(r1_to_b),
.r2_to_a(r2_to_a),
.r2_to_c(r2_to_c),
.r_to_a(r_to_a),
.ram_wr_en(ram_wr_en),
.ram_rd_en(ram_rd_en),
.ready(ready),
.error_o(error_o)
);

parameter IDLE          = 5'b00000;
parameter LOAD_N       = 5'b00001;
parameter LOAD_CMP     = 5'b00011;
parameter LOAD_CND     = 5'b00010;
parameter LOAD_R       = 5'b00110;
parameter LOAD_CMP_R   = 5'b00111;
parameter RLOOP_CND1   = 5'b00100;
parameter RLOOP_INIT   = 5'b00101;
parameter ILOOP_CALC_I = 5'b01101;
parameter ILOOP_ADDDR  = 5'b01111;
parameter ILOOP_LOAD   = 5'b01011;
parameter ILOOP_CND_R  = 5'b01010;
parameter ILOOP_D_CALC = 5'b10011;
parameter ILOOP_D_WB   = 5'b11011;
parameter ILOOP_I_CALC = 5'b11010;
parameter ILOOP_I_WB   = 5'b11110;
parameter ILOOP_CND_0  = 5'b11111;
parameter ERROR_0      = 5'b10111;
parameter ILOOP_CMP_I  = 5'b11101;
parameter RLOOP_CND2   = 5'b11000;
parameter RLOOP_INC_R  = 5'b10000;
parameter RLOOP_DEC_R  = 5'b01000;
parameter RLOOP_WB_R   = 5'b01100;

initial begin
    state = IDLE;
    #(CYCLE)

    state = LOAD_N;
    #(CYCLE)

    state = LOAD_CMP;
    #(CYCLE)

    state = LOAD_CND;
    #(CYCLE)
end
```

A Quellcode

```
state = LOAD_R;
#(CYCLE)

state = LOAD_CMP_R;
#(CYCLE)

state = RLOOP_CND1;
#(CYCLE)

state = RLOOP_INIT;
#(CYCLE)

state = ILOOP_CALC_I;
#(CYCLE)

state = ILOOP_ADDR;
#(CYCLE)

state = ILOOP_LOAD;
#(CYCLE)

state = ILOOP_CND_R;
#(CYCLE)

state = ILOOP_D_CALC;
#(CYCLE)

state = ILOOP_D_WB;
#(CYCLE)

state = ILOOP_I_CALC;
#(CYCLE)

state = ILOOP_I_WB;
#(CYCLE)

state = ILOOP_CND_0;
#(CYCLE)

state = ERROR_0;
#(CYCLE)

state = ILOOP_CMP_I;
#(CYCLE)

state = RLOOP_CND2;
#(CYCLE)

state = RLOOP_INC_R;
#(CYCLE)

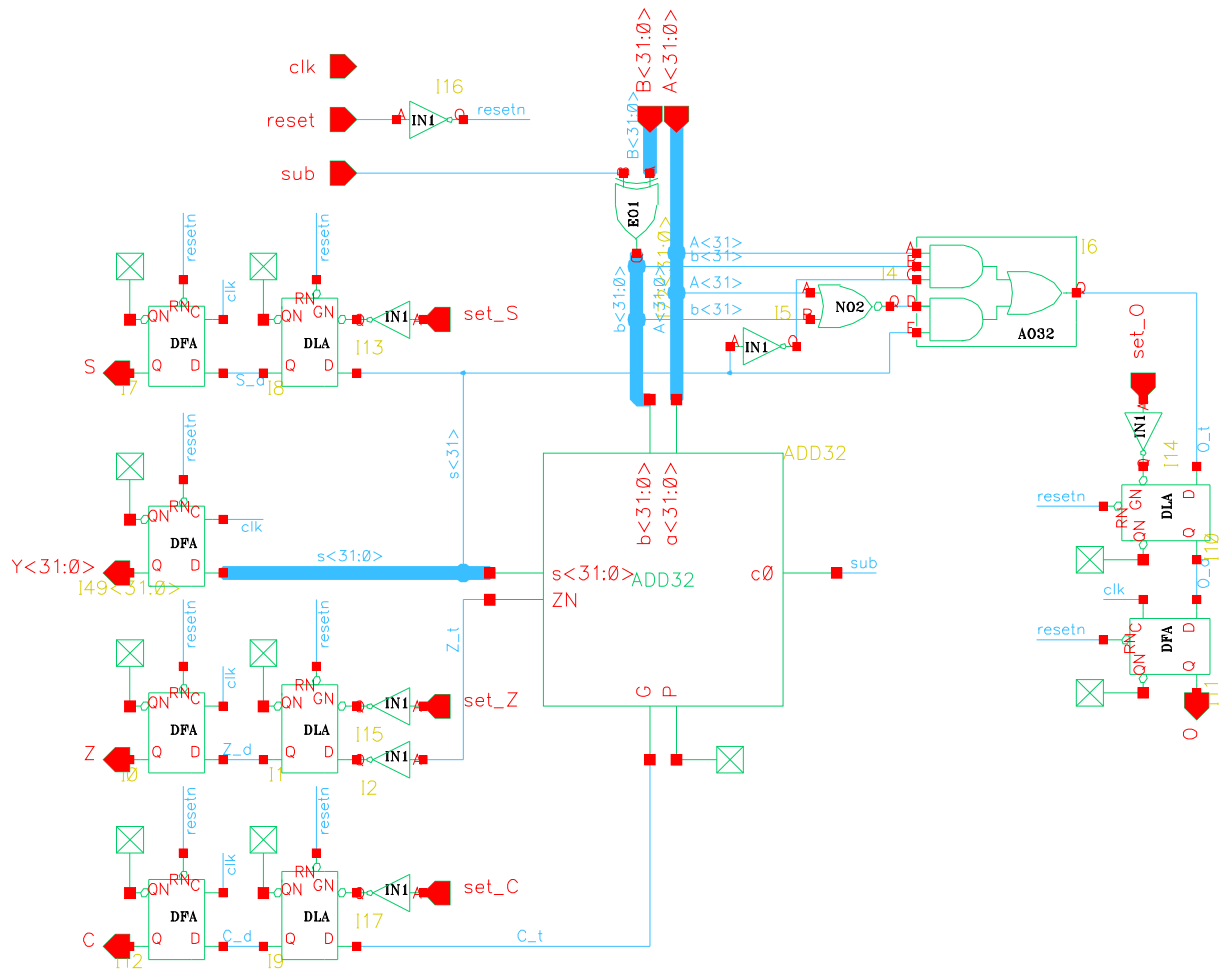
state = RLOOP_DEC_R;
#(CYCLE)

state = RLOOP_WB_R;
#(CYCLE)

$finish;
end
endmodule
```

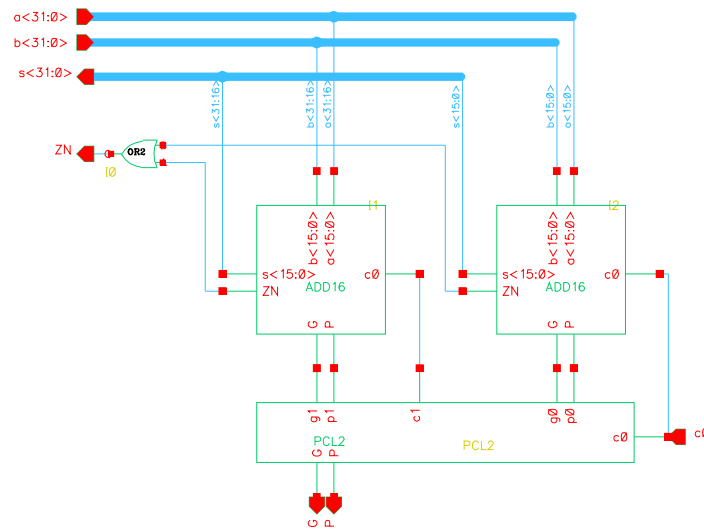
B Schematics

B.1 Arithmetic and Logic Unit

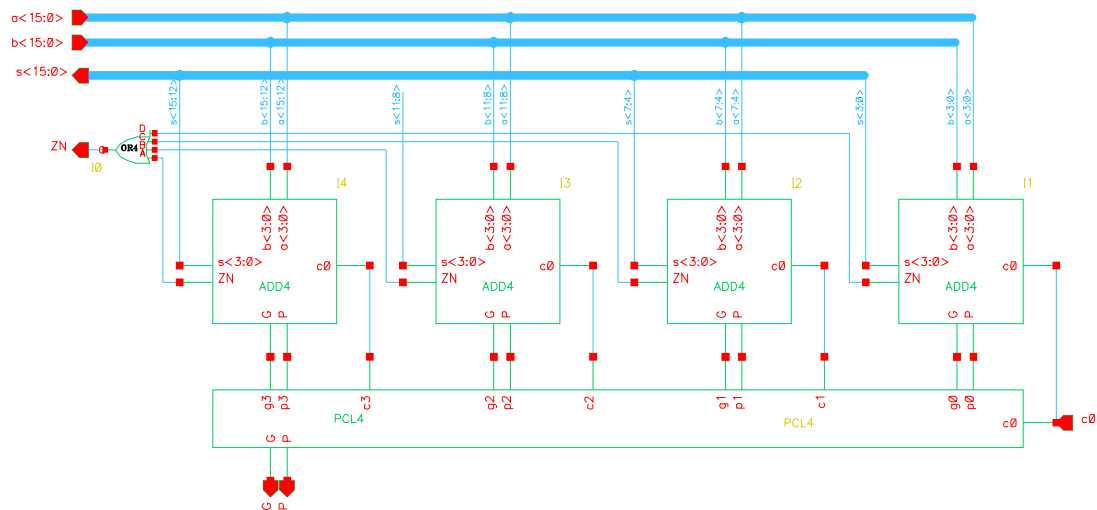


B Schematics

B.1.1 ADD32

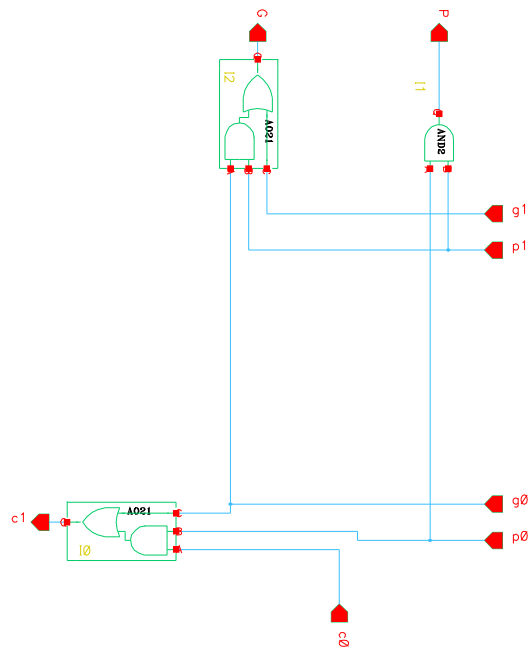


B.1.2 ADD16

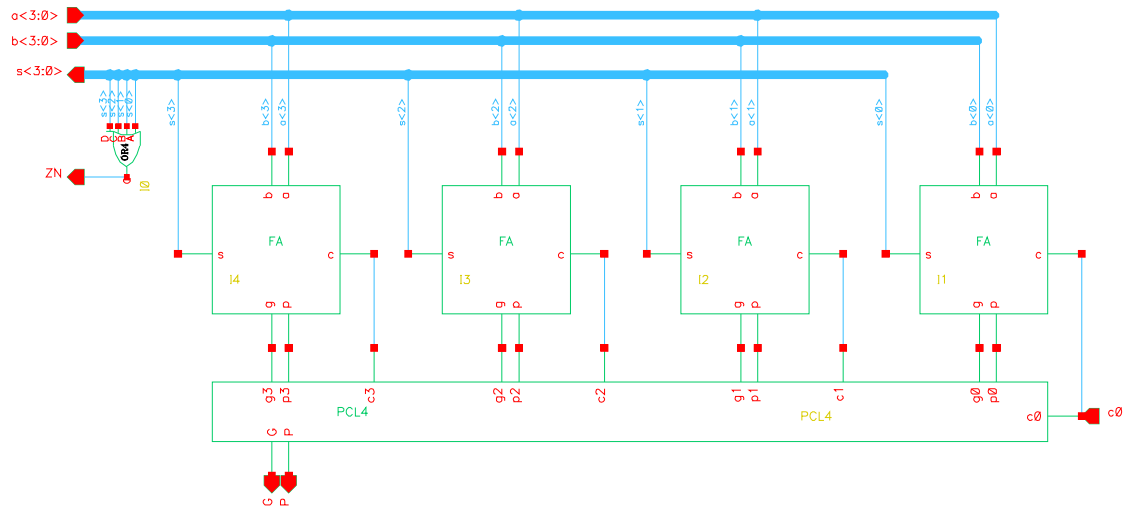


B Schematics

B.1.3 PCL2

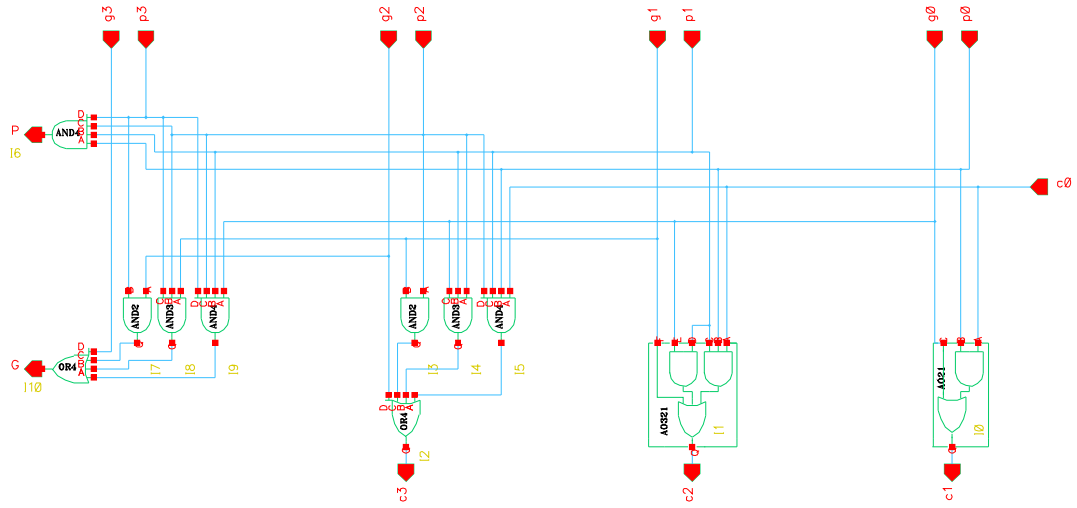


B.1.4 ADD4

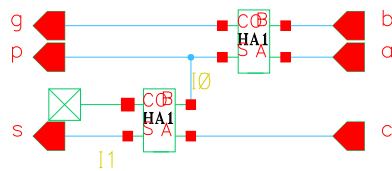


B.1.5 PCL4

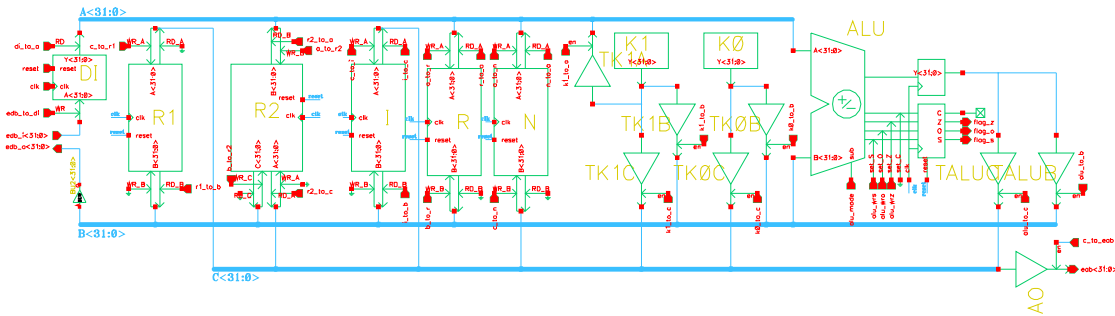
B Schematics



B.1.6 FA



B.2 Datenfad

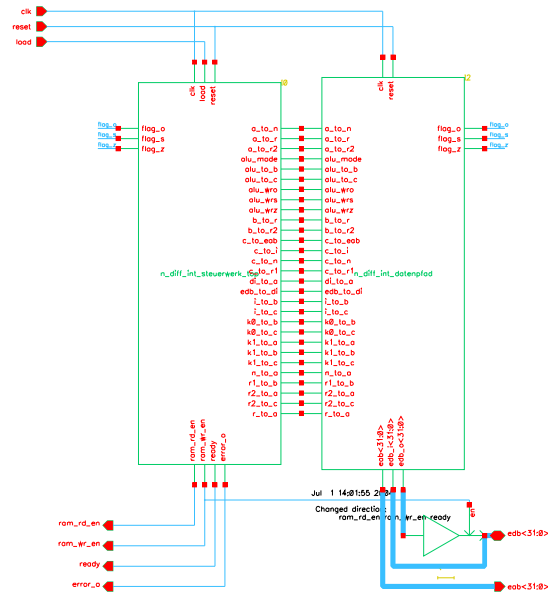


B.3 Steuerwerk

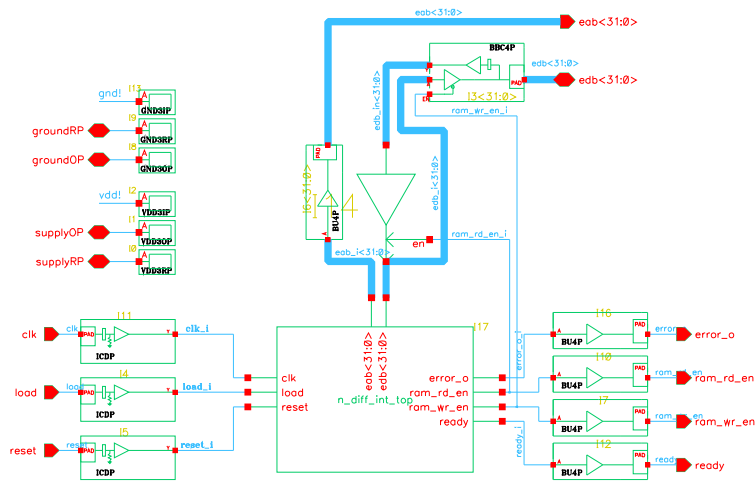


B Schematics

B.6 Top-Zelle

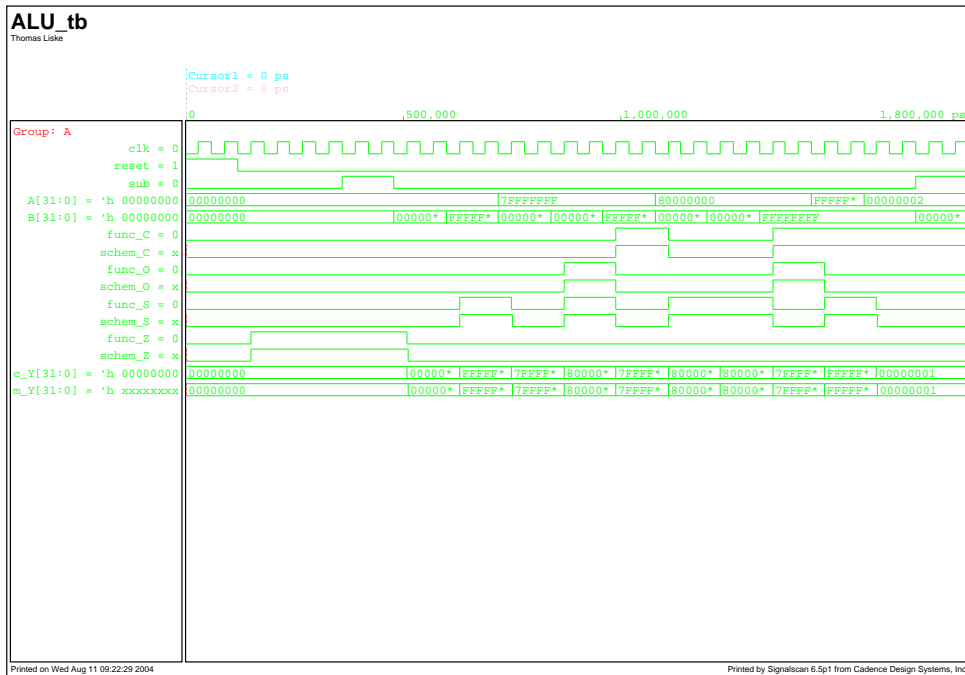


B.7 Pad-Zelle

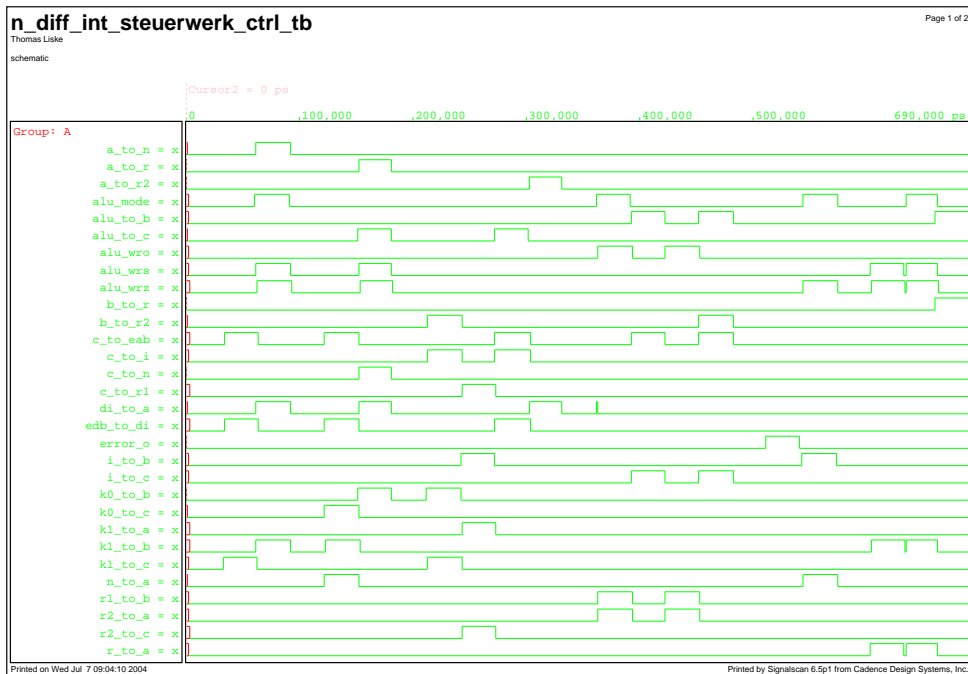
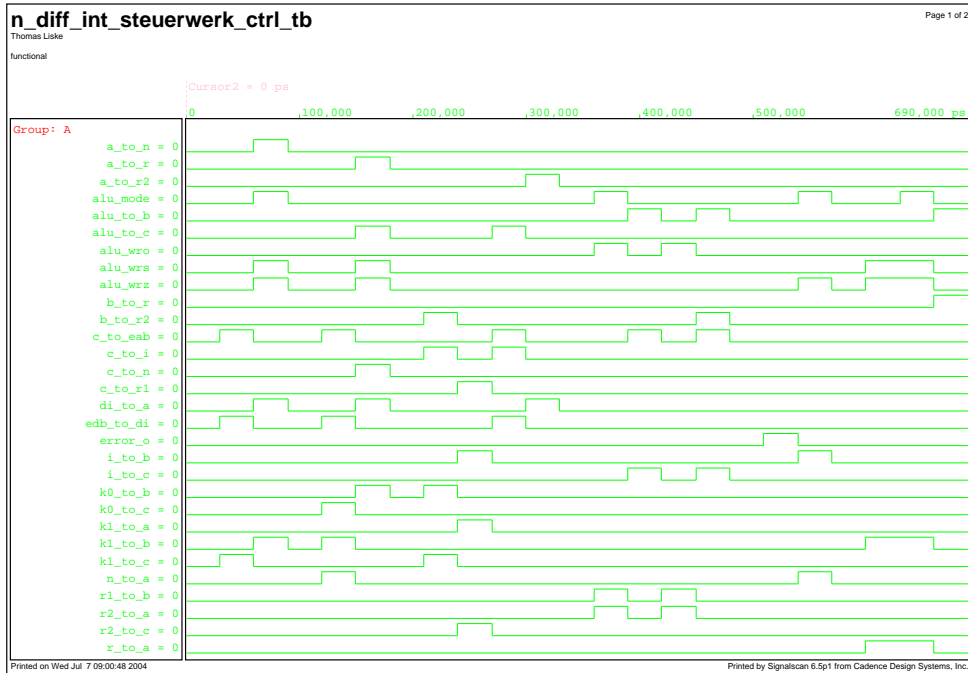


C Simulationsdiagramme

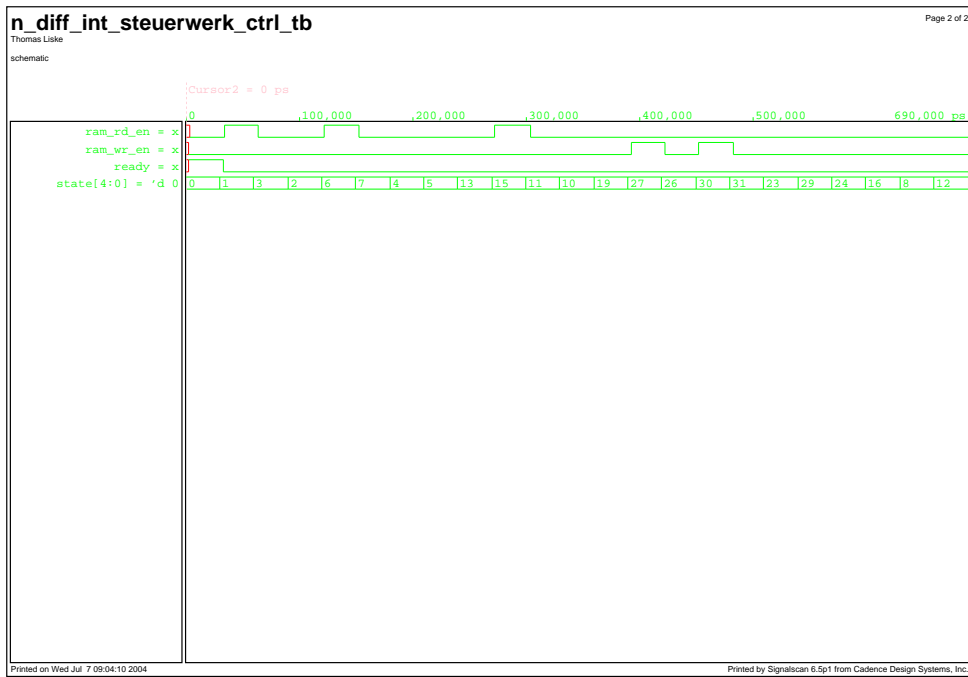
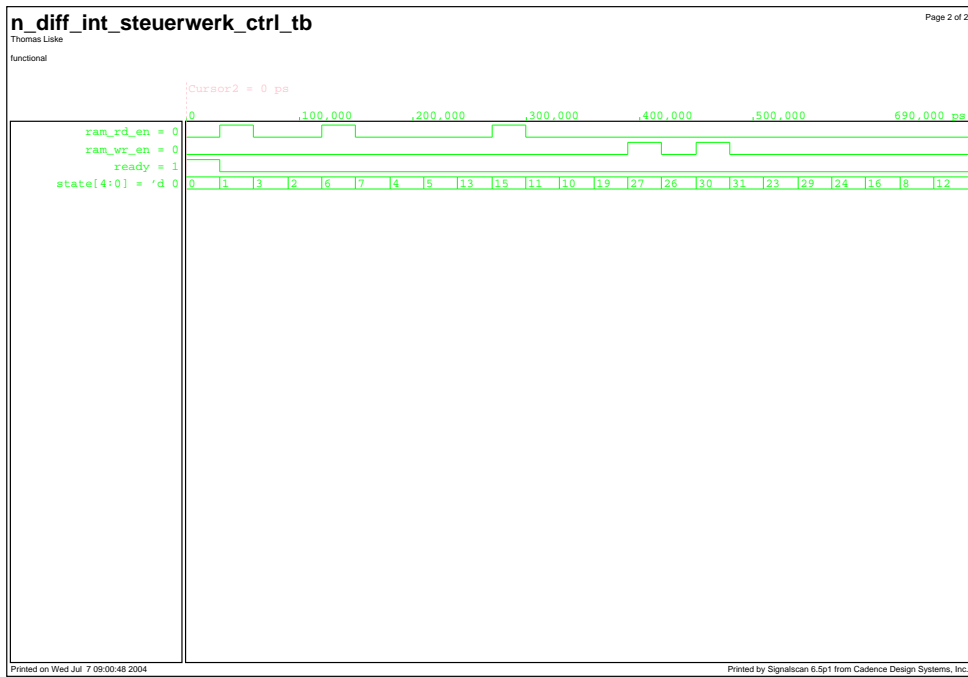
C.1 Arithmetic and Logic Unit



C.2 Steuerlogik



C Simulationsdiagramme



D Abbildungsverzeichnis

2.1	Speicherlayout	5
2.2	Blockdiagramm	6
3.1	Datenpfad	7
4.1	RT-Folge	8
5.1	Zustandsgraph	9
7.1	Zellenhierarchie	16

E Tabellenverzeichnis

6.2	Ansteuerung der Flip-Flop-Eingänge	10
6.1	Zustandsübergangstabelle	11
6.3	Ansteuerung des Datenpfades	13
6.4	Ansteuerung der Ausgabe-PINs	14

F Literaturverzeichnis

- [Aps03] A. Graupner, S. Getzlaff: Anleitung zu den Praktika "VLSI-Prozessorwurf", "Schaltkreiswurf", "Schaltkreis- und Systemwurf", 28. Oktober 2003
- [Hst02] U. Tietze, Ch. Schenk: Halbleiter-Schaltungstechnik, 12. Auflage, Springer Verlag, 2002, ISBN 3-540-42849-6