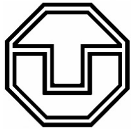


Fakultät Elektrotechnik und Informationstechnik

Institut für Grundlagen der Elektrotechnik und Elektronik - Lehrstuhl Hochparallele VLSI-Systeme und Neuromikroelektronik

Beleg

zur Lehrveranstaltung

Schaltkreis- und Systementwurf

Quadratwurzelberechnung mit Hilfe des Newtonschen Näherungsverfahrens

Andrej Olunczek
andrej@olunczek.de
Matrikel-Nr.: 3066276
ICPRO-Login: anol07

Betreuer: Prof. Dr.-Ing. habil. R. Schüffny
Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. R. Schüffny
vorgelegt am: 31. März 2010

Inhaltsverzeichnis

1 Einleitung **7**

2 Vorstellung des Algorithmus **9**
2.1 Das allgemeine Newtonverfahren 9
2.2 Spezialisierung zur Bestimmung von Quadratwurzeln 10
2.3 Optimierungen 13

3 Entwurf des Schaltkreises **15**
3.1 Der Datenpfad 15
3.2 Der Zustandsautomat allgemein 19
3.3 Gatterschaltung des Zustandautomaten 20

4 Simulationsergebnisse **25**
4.1 Gesamtschaltung, FSM als Verilog-Beschreibung 25
4.2 nur Zustandsautomat, Gatterschaltung 29
4.3 Gesamtschaltung, FSM als Gatterschaltung 35

5 Zusammenfassung **39**

A Quellcode **I**

Abbildungsverzeichnis

1 Beispiel Newton-Verfahren 9
2 Zwei Funktionen zur Wurzelberechnung 11
3 Beispiel für Nichtkonvergenz 12
4 Datenflussgraphen 15
5 Datenpfad 16
6 Register-Transfer-Folgen 17
7 Zustandsgraph 18
8 Übersicht Gesamtschaltung 19
9 Gatterschaltung des Zustandsautomaten 23
10 Simulation - Verilog-FSM - normal 26
11 Simulation - Verilog-FSM - 0 27
12 Simulation - Verilog-FSM - error 28
13 Simulation - nur FSM - Startphase 30
14 Simulation - nur FSM - Dividierphase 1 und 2 31
15 Simulation - nur FSM - Dividierphase 3 und 4 32
16 Simulation - nur FSM - Newtonverfahren, erster Durchlauf 33
17 Simulation - nur FSM - Newtonverfahren, letzter Durchlauf 34
18 Simulation - Gatter-FSM - normal 36
19 Simulation - Gatter-FSM - 0 37
20 Simulation - Gatter-FSM - error 38

Tabellenverzeichnis

1 Benötigte Variablen 15
2 Zustandskodierungen und -übergänge 20
3 Vergleich Logikaufwand 21

Quellcodeverzeichnis

1 Verilog-Code des Zustandsautomaten I
2 Verilog-Code der Steuerlogik IV
3 Verilog-Code der Gesamt-Testbench XI
4 Verilog-Code der FSM-Testbench XIII
5 Perl-Code des Websimulators XV

1 Einleitung

Ziel der Lehrveranstaltung Schaltkreis- und Systementwurf ist es, einen integrierten Schaltkreis zu entwickeln. Dazu soll ein frei gewählter Algorithmus so aufbereitet werden, dass er als Schaltkreis umsetzbar ist. Für diesen Algorithmus sollen in einzelnen Schritten ein Datenpfad und eine Steuerung in Form eines Zustandsautomaten (FSM, finite state machine) und der dazugehörigen Steuerlogik entwickelt werden.

Als Algorithmus setze ich das Newtonsche Näherungsverfahren um. Mit Hilfe dieses Verfahrens können näherungsweise Nullstellen von Funktionen bestimmt werden. Ich nutze hier ein Spezialfall des Verfahrens zur Berechnung der Quadratwurzel einer positiven reellen Zahl, sofern sich diese mit Hilfe des Fließkommazahlenformates nach IEEE Standard 754-1985 in der 64-Bit-Variante (double) darstellen lässt.

Im ersten Schritt erläutere ich den Algorithmus, und wie er für die Berechnung von Quadratwurzeln genutzt werden kann. Unter bestimmten Umständen kann es sehr viele Durchläufe benötigen, ehe ein verwertbares Ergebnis zu Stande kommt. Aus diesem Grund habe ich ein paar Modifikationen vorgenommen, um die Berechnungszeit zu verkürzen. Die Modifikationen werde ich anhand entsprechender Beispiele erklären.

Der Mittelteil enthält eine Beschreibung und Analyse des Aufbaues von Datenpfad, Zustandsautomat und Steuerlogik. Anhand der Register-Transfer-Folgen werde ich den genauen Ablauf des Algorithmus in der Hardware aufzeigen. Insbesondere auf die Optimierungen an dem Zustandsautomaten werde ich detailliert eingehen, was Art und Ansteuerung der nötigen Flipflops betrifft.

Im Abschnitt 4 werde ich anhand einiger Simulationsprotokolle die Funktionsfähigkeit der Schaltung darlegen. Insbesondere die Gatterschaltung des Zustandsautomaten wird ausführlich durch Simulationen überprüft, ob alle Zustandsübergänge in der richtigen Reihenfolge ablaufen.

Zu guter Letzt werde ich in einer Zusammenfassung die Arbeit noch einmal bewerten und einen kurzen Ausblick auf Erreichtes und weitere Möglichkeiten geben.

2 Vorstellung des Algorithmus

2.1 Das allgemeine Newtonverfahren

Der hier vorgestellte Algorithmus, das Newtonsche Näherungsverfahren, auch Newton-Raphsonsche Methode genannt, ist eine Methode zur näherungsweisen Bestimmung einer Nullstelle einer stetig differenzierbaren Funktion $f(x)$. Die Grundidee des Verfahrens ist es, die Funktion durch eine Tangente zu linearisieren, die durch einen, durch x_n definierten, vorgegebenen Punkt $P(x_n,\ f(x_n))$ der Funktion $f(x)$ geht. Die Nullstelle dieser Tangente ist die nächste Näherung der Nullstelle der Funktion $f(x)$, und als x_{n+1} gleichzeitig der Startwert für die nächste Iterationsstufe.

Um x_{n+1} zu berechnen, muss zu erst die Tangente bestimmt werden. Die Tangente ist eine einfache lineare Funktion der Form $t(x) = ax + b$. Der Anstieg der Tangente berechnet sich ganz einfach durch die Ableitung der Funktion $f(x)$ an der Stelle x_n. Demzufolge ist $a = f'(x_n)$. Um die vollständige Tangentengleichung zu bekommen, müssen wir die Tangente $t(x)$ mit der Funktion $f(x)$ in x_n gleichsetzten und nach b umstellen:

$$f(x_n) = f'(x_n)x_n + b \tag{1}$$

$$b = f(x_n) - f'(x_n)x_n \tag{2}$$

Daraus ergibt sich folgende Tangentengleichung:

$$t(x) = f'(x_n)x + f(x_n) - f'(x_n)x_n \tag{3}$$

Abbildung 1: Beispiel zum Newton-Verfahren: drei Schritte mit Startwert x_0=4

Von dieser Tangente benötigen wir nun die Nullstelle, da diese der nächste Näherungsschritt der Reihe ist. Die Nullstelle x_{n+1} berechnet sich wie folgt:

$$0 = f'(x_n)x_{n+1} + f(x_n) - f'(x_n)x_n \tag{4}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5}$$

Gleichung 5 ist damit die allgemeine Iterationsvorschrift des Newton-Verfahrens.

In Abbildung 1 ist ein Beispiel zu dem Verfahren gegeben. Die Funktion lautet $f(x) = (x/2)^2 - 1$, der Startwert $x_0 = 4$. Die ersten drei Iterationen berechnen sich wie folgt:

Tangente	**Nullstelle**
$t_0(x) = 2x - 5$	$x_1 = 2.5$
$t_1(x) = 1.25x - 2.5625$	$x_2 = 2.05$
$t_2(x) = 1.025x - 2.050625$	$x_3 = 2.00061$

Man sieht, das Verfahren konvergiert relativ schnell zur realen Nullstelle $x_{Nst} = 2$.

Die Konvergenz des Näherungsverfahrens ist aber nicht zwingend gegeben. Komplexere Funktionen, die z.B. Polstellen, Wendestellen oder ähnliches enthalten, können auch zu divergierenden Zahlenfolgen führen oder auch alternierendes oder zyklisches Verhalten haben. Ob die Folge konvergiert ist aber auch stark vom Startwert abhängig. Wenn die Folge konvergiert, dann quadratisch, die Anzahl gültiger Stellen verdoppelt sich also bei jedem Schritt[1].

2.2 Spezialisierung zur Bestimmung von Quadratwurzeln

Wir wollen hier aber nun mit Hilfe dieses Verfahrens die Quadratwurzel bestimmten. Zu diesem Zweck brauchen wir eine von a abhängige Funktion $f(x)$, die genau bei $\sqrt{a}$ ihre Nullstelle hat. Dazu bieten sich zwei Funktionen an:

$$f_1(x) = 1 - \frac{a}{x^2} \tag{6}$$

$$f_2(x) = x^2 - a \tag{7}$$

Mit Hilfe der allgemeinen Iterationsvorschrift aus Gleichung 5 ergeben sich folgende Iterationsformeln:

$$f_1(x) : x_{n+1} = \frac{x_n}{2}\left(3 - \frac{x_n^2}{a}\right) \tag{8}$$

$$f_2(x) : x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \tag{9}$$

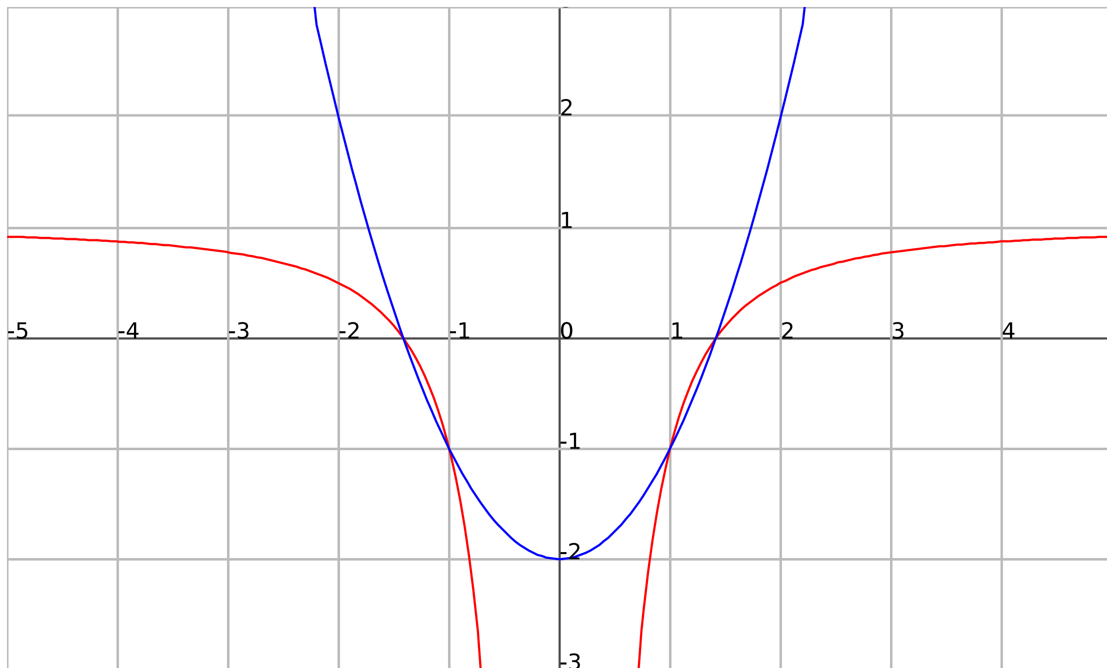


Abbildung 2: Mögliche Funktionen, die bei $\sqrt{a}$ (hier $a = 2$) eine Nullstelle haben

In Abbildung 2 sind die beiden Funktionen $f_1(x)$ (rot) und $f_2(x)$ (blau) dargestellt.

Welche der beiden Funktionen ist nun die Bessere? Da alle beide Funktionen alle drei Grundfunktionen, Addition, Multiplikation und Division, enthalten, ist der Hardwareaufwand ähnlich. Entscheidend ist also eher, welche Funktion schneller zum Ergebnis kommt. Der entscheidende Faktor ist dabei die Division, da sie sieben Takte benötigt. Formel 8 benötigt also zehn Takte, wenn man während der Division oder der Subtraktion schon die Halbierung von x_n vornimmt. Formel 9 ist etwas kürzer und benötigt nur neun Takte. In Formel 8 kann man aber die Division in der Iteration vermeiden, indem man den Kehrwert von a schon vorher berechnet. So benötigt man in der Schleife dann nur noch die Multiplikation mit diesem Kehrwert, im Idealfall ist das Ergebnis nach vier Takten berechnet.

Es gibt aber auch Unterschiede in der Konvergenz der beiden Funktionen. Da Funktion $f_2(x)$ keine Pol- oder Wendestellen enthält, konvergiert die Folge zu einer der Nullstelle. Wir betrachten hier nur den positiven Zahlenbereich, also alles rechts der y-Achse. Liegt der Startwert oberhalb der Nullstelle, nähert sich die Folge stetig von oben an die Nullstelle heran. Bei einem Startwert der unterhalb der Nullstelle liegt, ist der Wert nach dem ersten Iterationsschritt wieder oberhalb, und es gilt selbiges wie bei einem Startwert oberhalb der Nullstelle. Anders sieht es bei Funktion $f_1(x)$ aus. Dort kann es passieren, das die Folge mit wechselndem Vorzeichen Richtung unendlich wandert. Ein Beispiel dazu ist in Abbildung 3 zu sehen. Für den Fall dass der Startwert zwischen 0 und der Nullstelle liegt, konvergiert das Verfahren aber.

Ein weiterer Punkt, der wichtig ist, ist die Frage, ob zu jedem 64-Bit-Wert aus

der Menge der nach 'IEEE 754' möglichen Zahlen auch ein Ergebnis berechnet werden kann. Dazu ist zuerst einmal der mögliche Bereich zu definieren. Die 64-Bit-Fließkommazahl besteht aus einem Bit des Vorzeichen V, 52 Bit der Mantisse M und elf Bit des Exponenten E. Die Zahl berechnet sich dann zu

$$Z = (-1)^V \cdot (1.0 + \frac{M}{2^{52}}) \cdot 2^{E-1023}$$

Die Maximal- und Minimalwerte berechnen sich demzufolge zu

$$\begin{aligned} max &= (1 + \frac{2^{52}-1}{2^{52}}) \cdot 2^{1023} = 1.79769e + 308 \\ min &= (1 + \frac{0}{2^{52}}) \cdot 2^{-1022} = 2.22507e - 308 \end{aligned}$$

Allerdings sind für den Fall, dass der Exponent 0 ist, subnormale Zahlen definiert, die die Lücke zwischen 0 und dem zuvor berechneten Minimalwert füllen sollen. Diese definieren sich zu

$$Z = (-1)^V \cdot \frac{M}{2^{52}} \cdot 2^{-1022}$$

Damit ist der kleinste definierte Wert

$$min = \frac{1}{2^{52}} \cdot 2^{-1022} = 4.94066e - 324$$

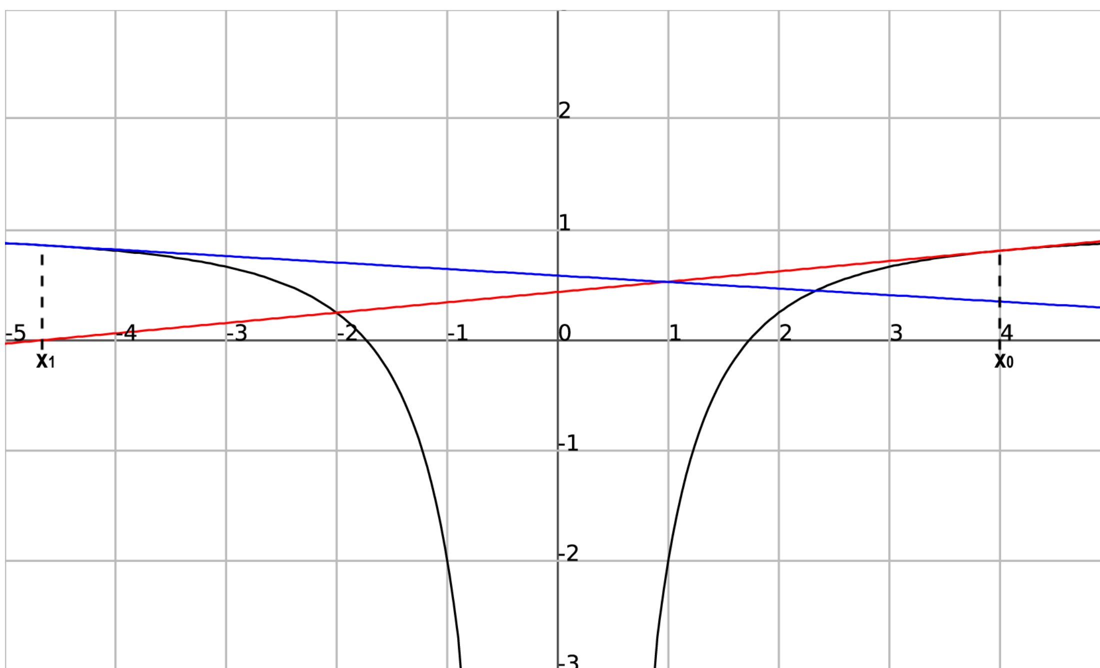


Abbildung 3: Für $a = 2$ und $x_0 = 4$ konvergiert das Beispiel nicht zur Nullstelle

Wenn wir jetzt Funktion $f_1(x)$ betrachten, ist der erste Arbeitsschritt die Kehrwertbildung von a. Schon für den Fall, dass die Quadratwurzel aus der kleinstmöglichen Fließkommazahl gebildet werden soll, gibt es ein Problem. $1/4.94066e-324 = 2.02402e+323$. Da dass größer als die maximal mögliche Zahl ist, würde ein Ergebnis von $1/a = \infty$, also einen Zahlenüberlauf verursachen. Damit lässt sich der Vorteil der Funktion $f_1(x)$ nicht nutzen. Da nun Funktion $f_2(x)$, ohne diesen Vorteil der Kehrwertbildung, einen Rechenschritt weniger benötigt, ist nun zu testen, ob diese für alle Eingangswerte ein gültiges Ergebnis bringt. Um das Verhalten vorherzusagen brauchen wir natürlich auch den Startwert für die Iterationsfolge. Für diesen nimmt man im Allgemeinen folgende Formel:

$$x_0 = 0.5 \cdot a + 0.5 \tag{10}$$

Dieser Wert ist immer größer oder gleich der zu berechnenden Nullstelle, der Quadratwurzel aus a. Damit kann sich die Folge von Begin an von oben an die Nullstelle annähern. Außerdem wird die Anzahl der Konstanten klein gehalten.

Um nun die Konvergenz von Formel 9 zu untersuchen, unterscheidet man am Besten in drei Fälle. Ist a kleiner, gleich oder größer als 1. Am einfachsten ist der Fall für $a = 1$, dann ist der Startwert gleich dem Ergebnis, weitere Berechnungen sind nicht nötig. Für den Fall $a < 1$ sind sowohl a als auch x_0 und folgende x_n kleiner als 1. Damit ergibt der Quotient ein sinnvolles, als Fließkommazahl darstellbares, Ergebnis, da bei der Division immer ein Wert größer als a heraus kommt. Geht man davon aus, dass x_n immer eine normale positive Fließkommazahl größer 0 ist, stellt der Wert innerhalb der Klammer eine Zahl dar, die mindestens dem Doppelten des kleinstmöglichen Wertes entspricht. Damit ist auch die abschließende Halbierung ohne Probleme möglich, ohne das auf 0 gerundet werden müsste. Für den Fall $a > 1$ sieht es ganz ähnlich aus. Da die x_n immer kleiner werden, ist die Gefahr des Zahlenbereichsüberlauf für die ersten Iterationsschritte am größten. Für sehr große a ist der Startwert $x_0 = 0.5a$. Dadurch kann x_n nicht größer als die Hälfte der größtmöglichen Zahl werden. Der Wert in der Klammer beträgt für sehr große x_n näherungsweise x_n, der Quotient a/x_n beginnt erst bei Annäherung an die Nullstelle größere Auswirkung zu haben. Daraus ist zu erkennen, das die Summe innerhalb der Klammer immer einen Wert kleiner der größtmöglichen Zahl haben muss. Da für den hier betrachteten Fall die x_n immer größer 1 sind, muss auch der Quotient eine, mit dem Fließkommaformat darstellbare, Zahl ergeben. Somit kommt man mit Formel 9 immer zu einem Ergebnis.

2.3 Optimierungen

Um jetzt einen Überblick über Verhalten, Laufzeiten und Konvergenz zu haben, habe ich einen kleinen Simulator unter `http://olunczek.de/ice-test/` online gestellt. In den linken drei Spalten läuft das Verfahren mit der Funktion$f_2(x)$, in der rechten Spalte mit der Funktion $f_1(x)$. In der linken läuft das Verfahren normal durch, in den mittleren beiden mit Optimierungsschritten, auf die ich im Folgenden noch eingehen werde.

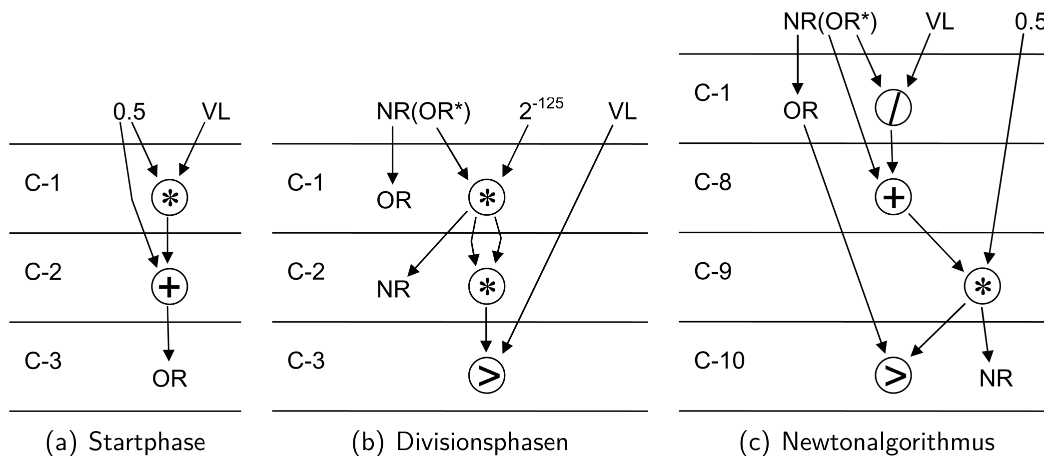
Man sieht, dass für sehr große Zahlen, z.B. $5e200$ sehr viele Iterationsschritte nötig sind, um zum finalen Ergebnis zu kommen. In diesem Beispiel sind das 338 Schritte, was am Ende über 3380 Takte benötigen würde. Da der Startwert in diesem Fall mehr als das $1e100$-fache des Ergebnisses beträgt, fragt man sich, ob man sich nicht schneller an das Ergebnis annähern kann. Da für Werte, die deutlich größer als das Ergebnis sind, die Newton-Iteration näherungsweise der Halbierung entspricht, halbiert man im einfachsten Fall solange den Startwert, wie er nicht kleiner als das Ergebnis wird. Da auch das noch sehr viele Schritte benötigen würde, kommen weitere Stufen davor, die mehrere Halbierungen auf einen Schlag erledigen. Für die Probe wird das Ergebnis der Halbierungen quadriert und mit dem Ausgangswert verglichen.

In der Simulation habe ich zwei Varianten, einmal mit drei Stufen mit jeweils 100, zehn und einer Halbierung pro Durchlauf. In der anderen Variante mit vier Stufen mit jeweils 125, 25, fünf und einer Halbierung. Bei einer Simulation mit dem Maximalwert von $1.79e+308$ sieht man, dass die Anzahl der Durchläufe in der ersten Stufe fünf, bzw. vier beträgt. Daraus kann man schlussfolgern, dass die längste Durchlaufzeit dann erreicht wird, wenn die Anzahl der Durchläufe vier, neun, neun bzw. bei vier Stufen drei, vier, vier, vier beträgt. Das wird bei einem Eingangswert von $4e301$ erreicht. In der Simulation sieht man nun, dass die Vier-Stufen- Variante, der mit drei Stufen vorzuziehen ist, da sie 17 Takte weniger benötigt. Es wird sich später auch zeigen, dass diese Variante genau in einem Zustandsautomaten mit fünf Bit, bzw. 32 Zuständen zu realisieren ist.

3 Entwurf des Schaltkreises

3.1 Der Datenpfad

Der ganze Algorithmus teilt sich im Wesentlichen in drei Teile, Startphase, Dividierphasen und der eigentliche Newton-Algorithmus. Dazu habe ich drei Datenflussgraphen aufgestellt:



(a) Startphase (b) Divisionsphasen (c) Newtonalgorithmus

Abbildung 4: Datenflussgraphen - **Im ersten Durchlauf wird OR benutzt, dann NR*

Da hier alle Operationen von der jeweils vorhergehenden abhängen, gibt es keine weiteren Möglichkeiten an der Reihenfolge einzelner Operationen herum zu schieben. Im Folgenden noch eine Übersicht über hier und im folgenden benutzte Variablennamen:

Variable	Beschreibung
VL	zu Beginn eingelesener Wert
OR	altes im vorherigen Schritt berechnetes Ergebnis
NR	neues aktuell berechnetes Ergebnis
TMP	temporärer Speicher, um zu frühes Überschreiben des alten Ergebnisses zu verhindern

Tabelle 1: Beschreibung der hier benutzten Variablen

Das temporäre Register wird nötig, da während der Sprungauswertung schon der nächste Schleifendurchlauf beginnt, ohne zu wissen, ob das Ergebnis noch benötigt wird. Damit kein falscher Wert in OR geschrieben wird, wird diesem TMP vorgeschaltet. So kann man nach dem Sprung entscheiden, ob der Wert in TMP nach OR übernommen wird, oder nicht.

Aus den Datenflussgraphen ist recht einfach zu erkennen, dass man jeweils einen Dividierer, eine Multipliziereinheit und eine ALU benötigt. Des Weiteren wird für jede Variable aus Tabelle 1 ein Register benötigt, sowie die Konstanten $2^{-125} =$

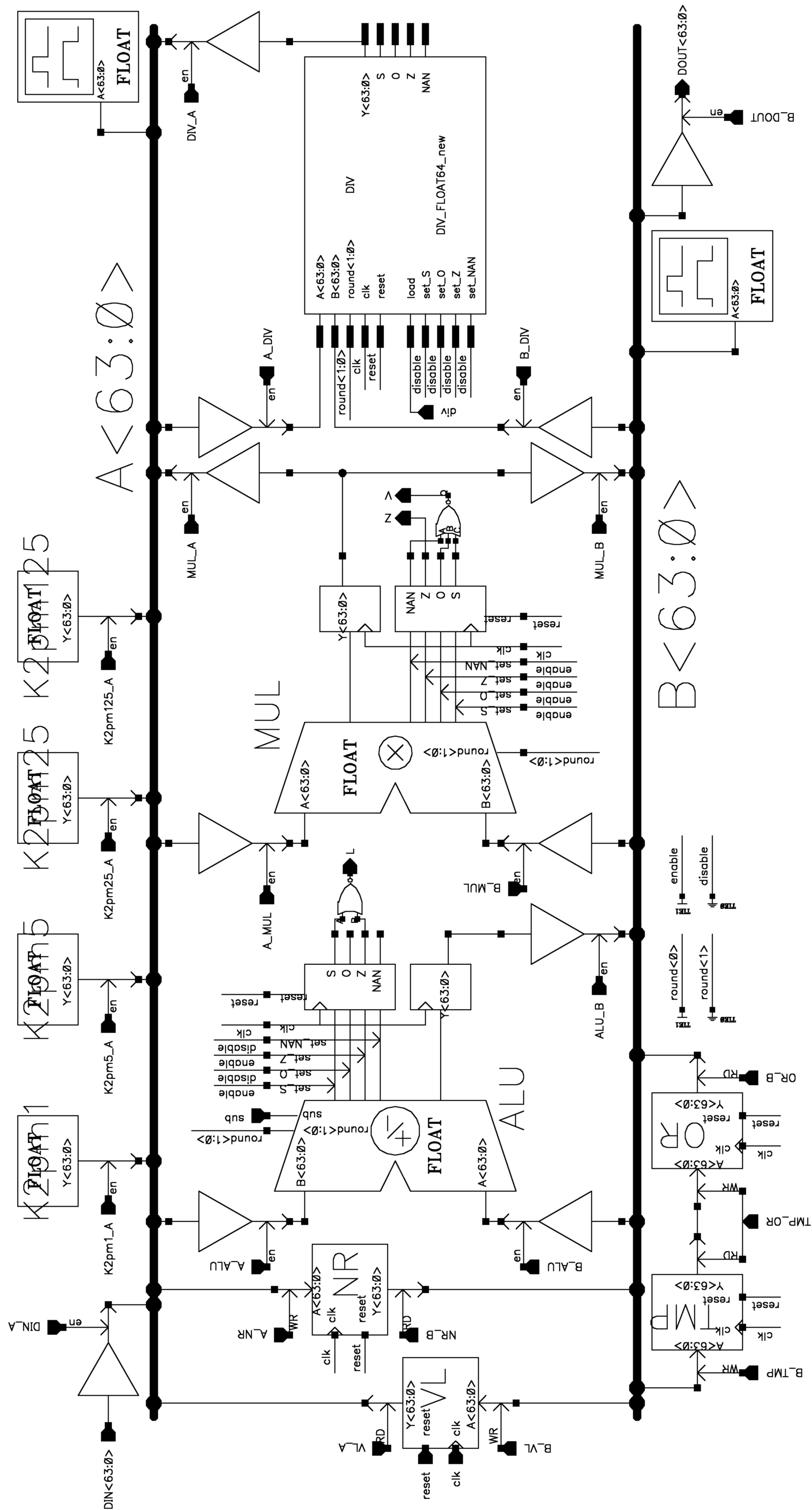


Abbildung 5: Der Datenpfad für den Algorithmus

$2.35099 \cdot 10^{-38}$, $2^{-25} = 2.98023 \cdot 10^{-08}$, $2^{-5} = 0.03125$ und $2^{-1} = 0.5$. Der Datenpfad (Abbildung 5) kann somit in einer einfachen 2-Bus-Architektur entworfen werden. Die Anordnung der Elemente ist so gewählt, dass einfache Register mit je einem Eingang und einem Ausgang genügen. Aus diesem Grund ist die ALU auch horizontal gespiegelt.

Der Rundungsmodus für die Rechenblöcke ist so voreingestellt, dass immer auf die nächste darstellbare Zahl gerundet wird. Da die Flags, wenn sie ausgewertet werden,

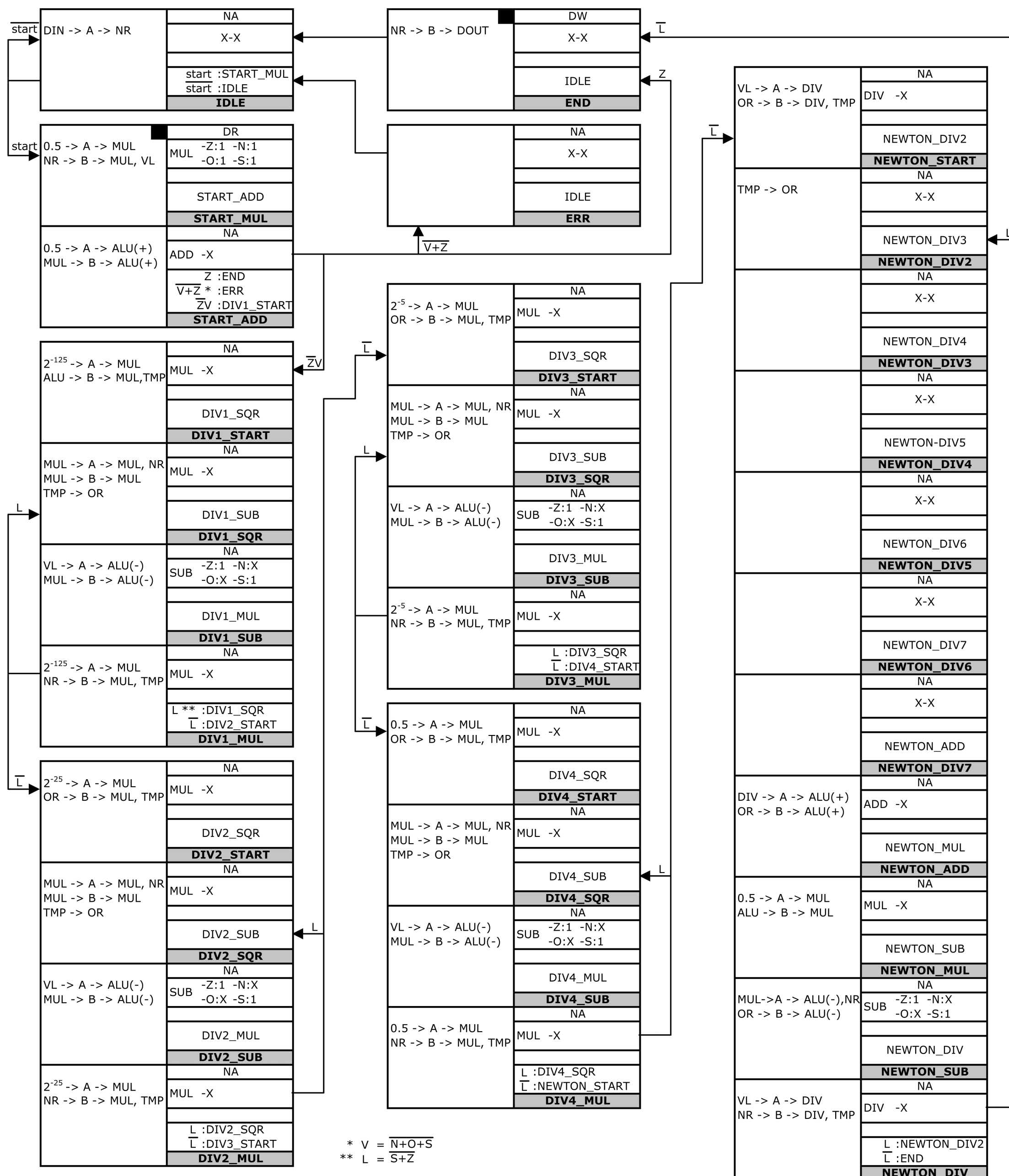


Abbildung 6: Register-Transfer-Folgen zur Ablaufsteuerung

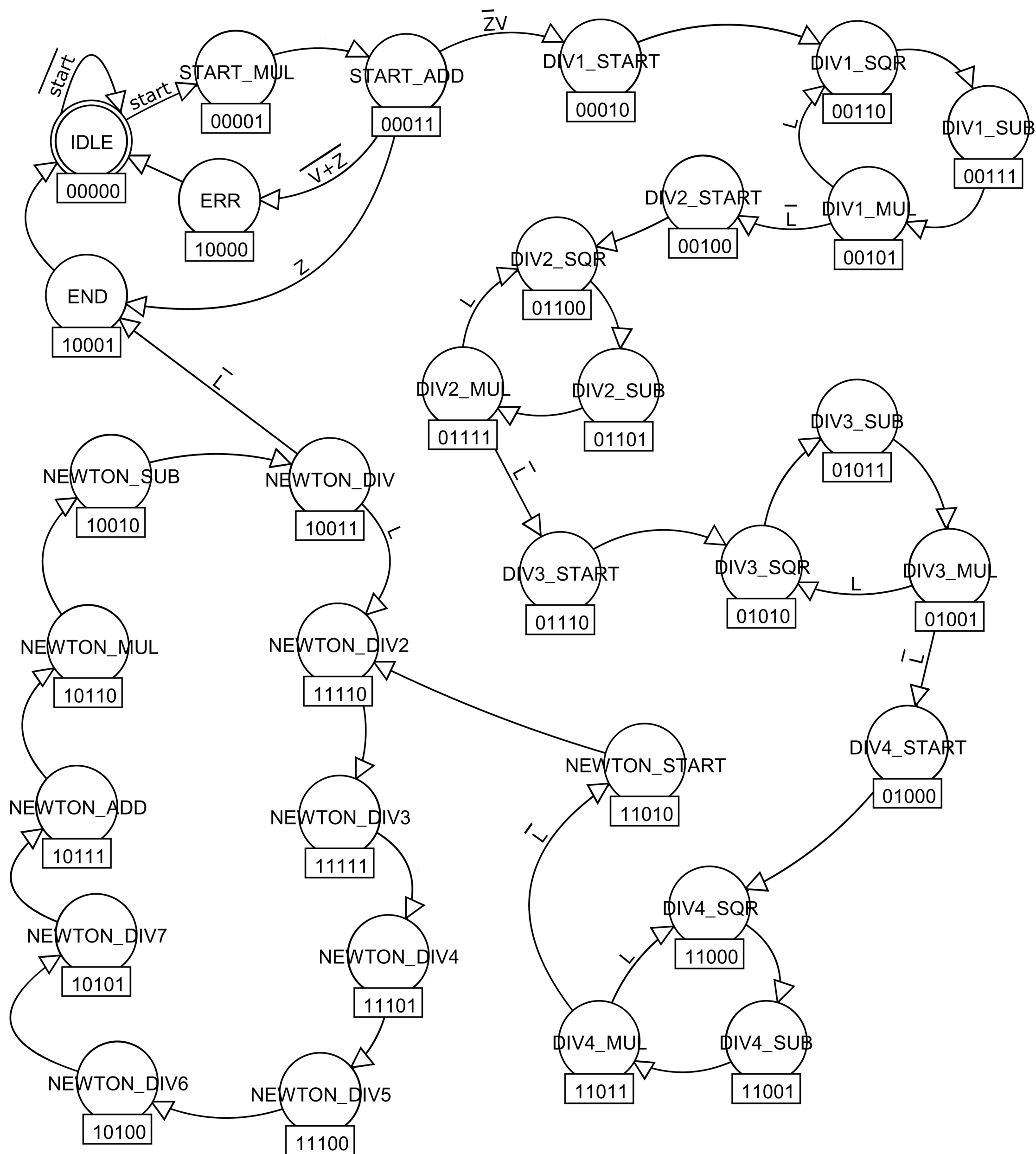


Abbildung 7: Zustandsgraph mit Binärkodierung

immer in der selber Art und Weise genutzt werden, sind sie schon vorkonfiguriert. So werden überzählige Eingänge gespart. Bei der ALU werden nur S und Z genutzt, bei dem Multiplizierer alle vier Flags. Um auch Ausgänge zu sparen werden einige Flags auch am Ausgang zusammengefasst. Im Folgenden verwende ich eine vereinfachte Logiknotation $A \wedge B = AB$ und $A \vee B = A + B$. So werden bei der ALU S und Z zu $L = \overline{S + Z}$ und beim Multiplizierer werden $N = NAN$, O und S zu $V = \overline{N + O + S}$.

Die beiden Float-Monitore an den beiden Bussen werden nur zur Simulation benö-

tigt, und haben keinen schaltungsrelevanten Nutzen.

3.2 Der Zustandsautomat allgemein

In Abbildung 6 sind die Register-Transfer-Folgen zu sehen. Es gibt prinzipiell zwei Möglichkeiten der Abarbeitung. Während der Auswertung von Flags für bedingte Sprünge, kann man schon Berechnungen vornehmen, ohne zu wissen, wohin der Sprung geht. Bei einer Schleife berechnet man also entweder dass, was nach dem Verlassen der Schleife passiert, oder man fängt schon mit dem nächsten Iterationsschritt an.

Für die vorgeschalteten Divisionsstufen hier heißt dass also, entweder man optimiert die Geschwindigkeit, wenn keine Schleifendurchläufe nötig sind. Dafür berechnet man während der Flagauswertung schon den ersten Schritt der nächsten Schleife. Dadurch dauert der erste Schleifendurchlauf nur drei, alle weiteren Durchläufe dafür vier Takte. Ich habe mich hier aber dafür entschieden die maximale Durchlaufzeit zu reduzieren, wenn die Schleifen mehrmals durchlaufen werden. Dafür wird während der Flagauswertung schon mit dem nächsten Durchlauf der Schleife begonnen. Somit werden für den ersten Durchlauf vier und für die weiteren Durchläufe nur drei Takte benötigt.

Während der Berechnung des Startwertes werden die Flags kontrolliert, um zu erkennen, ob die eingegebene Zahl zu einem gültigen Ergebnis verarbeitet werden kann. Ist dass nicht der Fall, wird ein Fehler ausgegeben. Liegt eine 0 am Eingang, wird diese sofort als Ergebnis ausgegeben.

Um nun daraus den Zustandsautomaten zu bauen, werden die Register-Transfer-

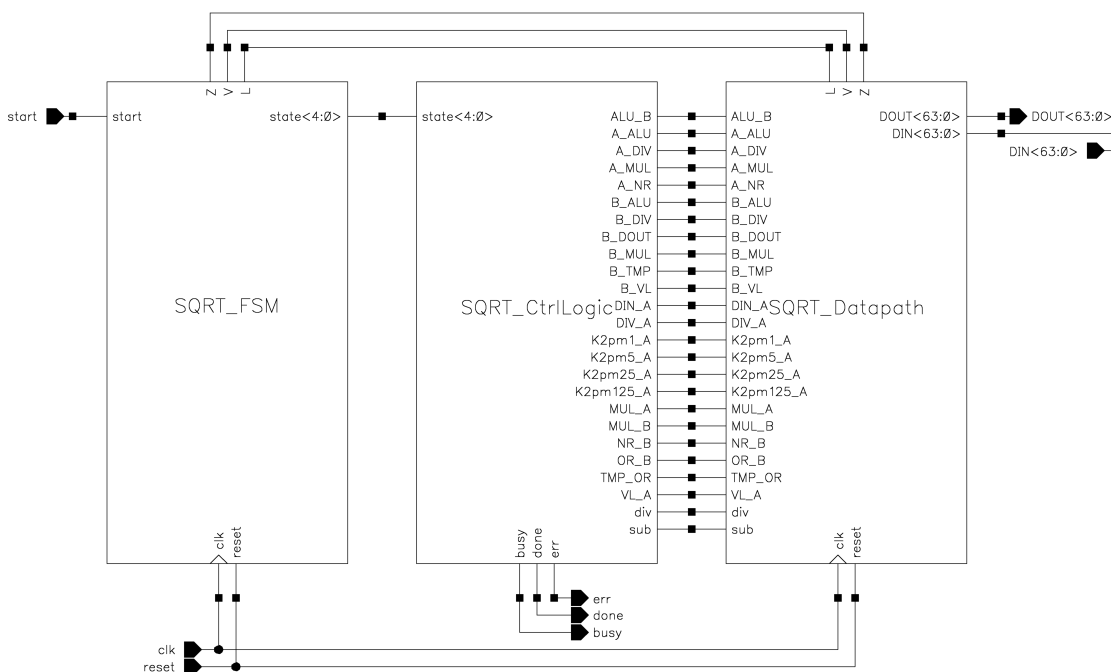


Abbildung 8: Gesamtübersicht der Schaltung mit Anordnung der einzelnen Einheiten

Folgen zuerst in einen Zustandsgraphen umgewandelt. In Abbildung 7 ist das Ergebnis zu sehen. Im gleichen Schritt wird die Zustandskodierung eingeführt. Ich habe hier den Gray-Code genutzt, in dem sich in jedem Schritt nur ein Bit ändert. Das spart übermäßiges hin- und herkippen der Flipflops in der FSM, und reduziert damit etwas die Leistungsaufnahme.

Zunächst wird der Zustandsautomat aus Abbildung 7 direkt eins zu eins in Verilog beschrieben, um die prinzipielle Funktionsweise des Automaten zu überprüfen. Der Verilog-Code für den Zustandsautomaten ist im Anhang im Quellcode 1 zu sehen. Damit die Schaltung funktioniert, wird noch eine Steuerlogik benötigt, die anhand des aktuellen Zustandes die Steuerleitungen des Datenpfades ansteuert. Der Verilog-Code für diese Steuerlogik ist im Anhang in Quellcode 2 dargestellt. In Abbildung 8 ist dargestellt, wie die einzelnen Elemente, Zustandsautomat(FSM), Steuerlogik und Datenpfad zur Gesamtschaltung zusammengeschlossen sind.

3.3 Gatterschaltung des Zustandautomaten

Teil der Aufgabe ist es aber auch, den Zustandsautomaten direkt als Schaltung zu realisieren. Dazu sind in Tabelle 2 noch einmal alle Zustandsübergänge aufgelistet.

state	s4	s3	s2	s1	s0	L	V	Z	start	next_state	D4	D3	D2	D1	D0	J4	K4	J3	K3	J2	K2	J1	K1	J0	K0	T4	T3	T2	T1	T0
IDLE	0	0	0	0	0	X	X	X	0	IDLE	0	0	0	0	0	0	X	0	X	0	X	0	X	0	X	0	0	0	0	0
IDLE	0	0	0	0	0	X	X	X	1	START_MUL	0	0	0	0	1	0	X	0	X	0	X	0	X	1	X	0	0	0	0	1
START_MUL	0	0	0	0	1	X	X	X	X	START_ADD	0	0	0	1	1	0	X	0	X	0	X	1	X	X	0	0	0	0	1	0
START_ADD	0	0	0	1	1	X	0	0	X	ERR	1	0	0	0	0	1	X	0	X	0	X	X	1	X	1	1	0	0	1	1
START_ADD	0	0	0	1	1	X	X	1	X	END	1	0	0	0	1	1	X	0	X	0	X	X	1	X	0	1	0	0	1	0
START_ADD	0	0	0	1	1	X	1	0	X	DIV1_START	0	0	0	1	0	0	X	0	X	0	X	X	0	X	1	0	0	0	0	1
DIV1_START	0	0	0	1	0	X	X	X	X	DIV1_SQR	0	0	1	1	0	0	X	0	X	1	X	X	0	0	X	0	0	1	0	0
DIV1_SQR	0	0	1	1	0	X	X	X	X	DIV1_SUB	0	0	1	1	1	0	X	0	X	X	0	X	0	1	X	0	0	0	0	1
DIV1_SUB	0	0	1	1	1	X	X	X	X	DIV1_MUL	0	0	1	0	1	0	X	0	X	X	0	X	1	X	0	0	0	0	1	0
DIV1_MUL	0	0	1	0	1	1	X	X	X	DIV1_SQR	0	0	1	1	0	0	X	0	X	X	0	1	X	X	1	0	0	0	1	1
DIV1_MUL	0	0	1	0	1	0	X	X	X	DIV2_START	0	0	1	0	0	0	X	0	X	X	0	0	X	X	1	0	0	0	0	1
DIV2_START	0	0	1	0	0	X	X	X	X	DIV2_SQR	0	1	1	0	0	0	X	1	X	X	0	0	X	0	X	0	1	0	0	0
DIV2_SQR	0	1	1	0	0	X	X	X	X	DIV2_SUB	0	1	1	0	1	0	X	X	0	X	0	0	X	1	X	0	0	0	0	1
DIV2_SUB	0	1	1	0	1	X	X	X	X	DIV2_MUL	0	1	1	1	1	0	X	X	0	X	0	1	X	X	0	0	0	0	1	0
DIV2_MUL	0	1	1	1	1	1	X	X	X	DIV2_SQR	0	1	1	0	0	0	X	X	0	X	0	X	1	X	1	0	0	0	1	1
DIV2_MUL	0	1	1	1	1	0	X	X	X	DIV3_START	0	1	1	1	0	0	X	X	0	X	0	X	0	X	1	0	0	0	0	1
DIV3_START	0	1	1	1	0	X	X	X	X	DIV3_SQR	0	1	0	1	0	0	X	X	0	X	1	X	0	0	X	0	0	1	0	0
DIV3_SQR	0	1	0	1	0	X	X	X	X	DIV3_SUB	0	1	0	1	1	0	X	X	0	0	X	X	0	1	X	0	0	0	0	1
DIV3_SUB	0	1	0	1	1	X	X	X	X	DIV3_MUL	0	1	0	0	1	0	X	X	0	0	X	X	1	X	0	0	0	0	1	0
DIV3_MUL	0	1	0	0	1	1	X	X	X	DIV3_SQR	0	1	0	1	0	0	X	X	0	0	X	1	X	X	1	0	0	0	1	1
DIV3_MUL	0	1	0	0	1	0	X	X	X	DIV4_START	0	1	0	0	0	0	X	X	0	0	X	0	X	X	1	0	0	0	0	1
DIV4_START	0	1	0	0	0	X	X	X	X	DIV4_SQR	1	1	0	0	0	1	X	X	0	0	X	0	X	0	X	1	0	0	0	0
DIV4_SQR	1	1	0	0	0	X	X	X	X	DIV4_SUB	1	1	0	0	1	X	0	X	0	0	X	0	X	1	X	0	0	0	0	1
DIV4_SUB	1	1	0	0	1	X	X	X	X	DIV4_MUL	1	1	0	1	1	X	0	X	0	0	X	1	X	X	0	0	0	0	1	0
DIV4_MUL	1	1	0	1	1	1	X	X	X	DIV4_SQR	1	1	0	0	0	X	0	X	0	0	X	X	1	X	1	0	0	0	1	1
DIV4_MUL	1	1	0	1	1	0	X	X	X	NEWTON_START	1	1	0	1	0	X	0	X	0	0	X	X	0	X	1	0	0	0	0	1
NEWTON_START	1	1	0	1	0	X	X	X	X	NEWTON_DIV2	1	1	1	1	0	X	0	X	0	1	X	X	0	0	X	0	0	1	0	0
NEWTON_DIV2	1	1	1	1	0	X	X	X	X	NEWTON_DIV3	1	1	1	1	1	X	0	X	0	X	0	X	0	1	X	0	0	0	0	1
NEWTON_DIV3	1	1	1	1	1	X	X	X	X	NEWTON_DIV4	1	1	1	0	1	X	0	X	0	X	0	X	1	X	0	0	0	0	1	0
NEWTON_DIV4	1	1	1	0	1	X	X	X	X	NEWTON_DIV5	1	1	1	0	0	X	0	X	0	X	0	0	X	X	1	0	0	0	0	1
NEWTON_DIV5	1	1	1	0	0	X	X	X	X	NEWTON_DIV6	1	0	1	0	0	X	0	X	1	X	0	0	X	0	X	0	1	0	0	0
NEWTON_DIV6	1	0	1	0	0	X	X	X	X	NEWTON_DIV7	1	0	1	0	1	X	0	0	X	X	0	0	X	1	X	0	0	0	0	1
NEWTON_DIV7	1	0	1	0	1	X	X	X	X	NEWTON_ADD	1	0	1	1	1	X	0	0	X	X	0	1	X	X	0	0	0	0	1	0
NEWTON_ADD	1	0	1	1	1	X	X	X	X	NEWTON_MUL	1	0	1	1	0	X	0	0	X	X	0	X	0	X	1	0	0	0	0	1
NEWTON_MUL	1	0	1	1	0	X	X	X	X	NEWTON_SUB	1	0	0	1	0	X	0	0	X	X	1	X	0	0	X	0	0	1	0	0
NEWTON_SUB	1	0	0	1	0	X	X	X	X	NEWTON_DIV	1	0	0	1	1	X	0	0	X	0	X	X	0	1	X	0	0	0	0	1
NEWTON_DIV	1	0	0	1	1	1	X	X	X	NEWTON_DIV2	1	1	1	1	0	X	0	1	X	1	X	X	0	X	1	0	1	1	0	1
NEWTON_DIV	1	0	0	1	1	0	X	X	X	END	1	0	0	0	1	X	0	0	X	0	X	X	1	X	0	0	0	0	1	0
END	1	0	0	0	1	X	X	X	X	IDLE	0	0	0	0	0	X	1	0	X	0	X	0	X	X	1	1	0	0	0	1
ERR	1	0	0	0	0	X	X	X	X	IDLE	0	0	0	0	0	X	1	0	X	0	X	0	X	0	X	1	0	0	0	0

Tabelle 2: Zustands- und Übergangskodierungen für D-, JK- und T-Flipflops

	D	D_0	J	J_0	K	K_0	T	T_0
state<4>	32	25	17	16	4	7	25	40
state<3>	20	37	11	28	5	9	19	44
state<2>	48	42	16	33	10	15	31	86
state<1>	62	70	30	28	43	37	86	64
state<0>	60	60	41	46	46	47	104	110

(a) Logikkosten

	D	D_0	J	J_0	K	K_0	T	T_0
state<4>	11	9	5	6	1	1	7	12
state<3>	6	11	4	9	1	1	7	16
state<2>	14	14	5	9	3	5	12	30
state<1>	20	19	9	8	13	12	27	20
state<0>	17	17	12	13	14	14	38	41

(b) Gatteranzahl

	D	D_0	J	J_0	K	K_0	T	T_0
state<4>	29	24	14	15	3	3	22	33
state<3>	18	36	12	25	4	4	22	43
state<2>	44	38	17	24	10	12	37	78
state<1>	62	58	29	25	41	37	82	63
state<0>	57	60	44	45	47	47	120	127

(c) Gatterkosten

Tabelle 3: Vergleich Logikaufwand für die FSM, die Minimalwerte sind gelb hinterlegt

Um zu entscheiden, welche Art von Flipflops die besten sind, habe ich in der Tabelle auch die Ansteuerung für die zu Verfügung stehenden D(ata)-, J(ump/)K(ill)- und T(oggle)-Flipflops erfasst. Mit Hilfe des Karnaugh-Skriptes habe ich für alle Typen jeweils die Gleichungstabellen für alle Eingänge in der normalen und der negierten Form erstellt. Die dabei errechneten theoretischen Logikkosten sind in Tabelle 3(a) aufgelistet. Für die negierten Eingangssignale (mit _0 gekennzeichnet) ist der Wert um 1 erhöht, da theoretisch noch ein Inverter für die korrekte Ansteuerung nötig ist.

Dieser theoretische Wert ist aber sehr ungenau, da er davon ausgeht, dass es AND- bzw. OR-Gatter mit beliebig vielen Eingängen gibt. Aus diesem Grund ist in Tabelle 3(b) die Anzahl der NAND-/NOR-Gatter aufgelistet, denn nur diese sind verfügbar. Die Ausgabe des Karnaugh-Skriptes gibt eine Formel der Form $O = (A\,B\,\cdots\,X) + (B\,D\,\cdots\,Y) + \cdots$ aus. Um daraus die Anzahl der NAND-/NOR-Gatter zu bestimmen sind folgende Umformungen nötig: $A\,B\,C = \overline{\overline{A}+\overline{B}+\overline{C}}$ und $A+B+C = \overline{\overline{A}\,\overline{B}\,\overline{C}}$. Da es nur Gatter mit maximal vier Eingängen gibt werden noch folgende Umrechnungen angewandt: $A\,B\,C\,D\,E\,F = \overline{\overline{\overline{A\,B\,C\,D}}+\overline{E\,F}}$. Um die Logik zu minimieren ist manchmal noch folgende Umwandlung möglich: $A\,B\,C\,D\,E + A\,B\,C\,D\,F = (A\,B\,C\,D)\,(E+F)$. Mit Hilfe dieser Formeln wurde die Anzahl der nötigen NAND-/NOR-Gatter ermittelt.

Zur endgültigen Ermittlung der günstigsten Flipflops habe ich in Tabelle 3(c) die Logikkosten der errechneten Gatter zusammengestellt. Dazu wurde einfach die Summe aller Eingänge aller Gatter zusammen addiert. Man sieht nun, dass für die beiden niederwertigsten Bits D-Flipflops nötig sind, und für die drei hochwertigsten JK-Flipflops. Diese kosten zwar etwas mehr, machen auf die Logikkosten der Gatter umgerechnet, aber nur etwa einen Wert von 2 aus[6].

Die finalen Gleichungen für die Ansteuerung der Flipflops sehen damit wie folgt aus:

$$\begin{aligned} D_0 \; = \; & \overline{\overline{\overline{\overline{\overline{\overline{Q_4}\,\overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}} + \overline{start+Q_0}}} + \overline{\overline{\overline{\overline{Q_4}\,\overline{Q_3}\,\overline{Q_2}}\,Q_0}+\overline{Z}} + \overline{Q_4+Q_3+\overline{Q_2}+\overline{Q_1}} + \cdots}} \\ & \overline{\overline{\cdots + \overline{Q_4+\overline{Q_3}+\overline{Q_2}+Q_1}\;\overline{Q_4+\overline{Q_3}+Q_2+\overline{Q_1}} + \overline{\overline{Q_4}+\overline{Q_3}+Q_2+Q_1} + \cdots}} \\ & \overline{\overline{\cdots + \overline{\overline{Q_4}+\overline{Q_3}+\overline{Q_2}+\overline{Q_1}} + \overline{\overline{Q_4}+Q_3+\overline{Q_2}+Q_1}}\;\overline{\overline{Q_4}+Q_3+Q_2+\overline{Q_1}}\;\overline{Q_0\,L}} \end{aligned}$$

$$D_1 = \overline{\overline{\overline{\overline{Q_4\,\overline{Q_3}\,\overline{Q_2}}\;\overline{Q_1+Q_0}}\;\overline{\overline{Q_4+Q_3+\overline{Q_1}+\overline{Q_0}}\;\overline{Q_2}\,V\,\overline{\overline{Z}}}\;\overline{\overline{Q_4+Q_3+\overline{Q_2}+\overline{Q_0}}\;\overline{L}}\;\cdots}}$$

$$\cdots\;\overline{\overline{Q_3\,Q_1\,Q_0\,L}} + \overline{\overline{\overline{Q_4+\overline{Q_3}+Q_2+\overline{Q_0}}\;\overline{\overline{Q_1}}\,L}\;\overline{Q_4\,Q_3\,Q_2\,Q_0}\;\cdots}$$

$$\cdots\;\overline{\overline{\overline{\overline{\overline{Q_4}+Q_3+Q_2+\overline{Q_0}}\;\overline{L}}\;\overline{Q_4\,\overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}}}}$$

$$J_2 = \overline{\overline{\overline{\overline{Q_4}\,\overline{Q_3}\,Q_1\,\overline{Q_0}}\;\overline{Q_4\,Q_3\,Q_1\,\overline{Q_0}}\;\overline{\overline{\overline{Q_4}+Q_3+\overline{Q_1}+\overline{Q_0}}\;L}}}$$

$$K_2 = \overline{\overline{\overline{Q_4}\,Q_3\,Q_1\,\overline{Q_0}}\;\overline{Q_4\,\overline{Q_3}\,Q_1\,\overline{Q_0}}}$$

$$J_3 = \overline{\overline{\overline{Q_4}\,Q_2\,\overline{Q_1}\,\overline{Q_0}}\;\overline{\overline{\overline{Q_4}+Q_2+\overline{Q_1}+\overline{Q_0}}\;L}}$$

$$K_3 = \overline{\overline{Q_4}+\overline{Q_2}+Q_1+Q_0}$$

$$J_4 = \overline{\overline{\overline{Q_3+Q_2+\overline{Q_1}+\overline{Q_0}}\;\overline{V\,\overline{Z}}}\;\overline{Q_3\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0}}}$$

$$K_4 = \overline{Q_3+Q_2+Q_1}$$

Mit Hilfe dieser Vorgaben kann man nun die Schaltung für den Zustandsautomaten zusammenstellen. Das Ergebnis ist in Abbildung 9 zu sehen. Durch Umformungen war es dabei noch möglich NAND-NAND- bzw. NOR-NOR-Folgen durch Komplexgatter der Form OR-NAND bzw. AND-NOR zu ersetzen. Die nötigen Umformungen dafür lauten: $\overline{\overline{A\,B}\,C} = \overline{(\overline{A}+\overline{B})\,C}$ bzw. $\overline{\overline{A+B}+C} = \overline{\overline{A}\,\overline{B}+C}$.

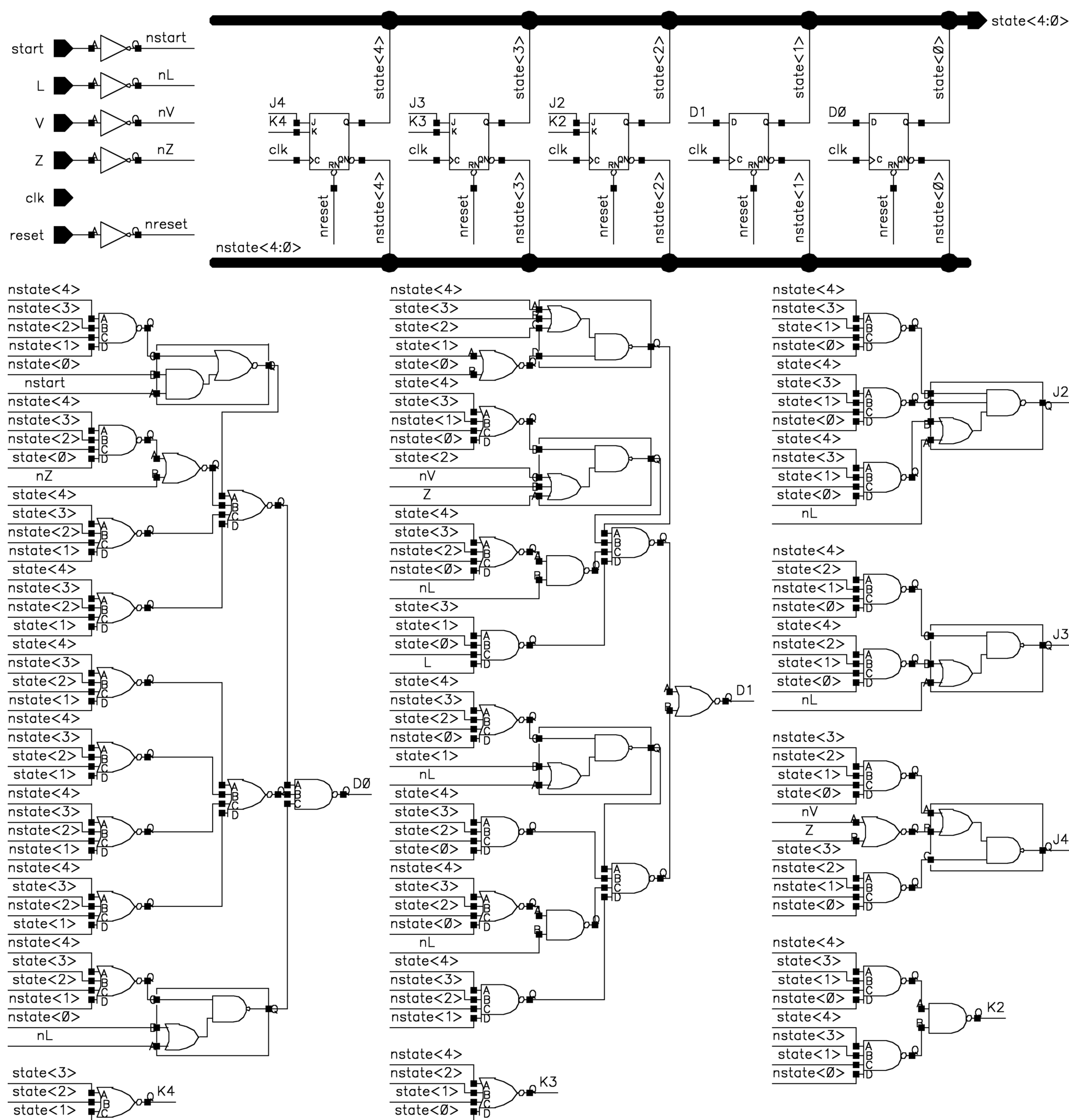


Abbildung 9: Gatterschaltung des Zustandsautomaten

4 Simulationsergebnisse

4.1 Gesamtschaltung, FSM als Verilog-Beschreibung

Die folgenden Simulationsprotokolle zeigen die prinzipielle Funktionstüchtigkeit der Schaltung mit dem in Verilog beschriebenen Zustandsautomaten. Im Anhang ist im Quelltext 3 der Verilog-Code der Testbench aufgeführt. Ich habe hier beispielhaft nur drei Testfälle aufgeführt. Den normalen Fall, also die Berechnung der Quadratwurzel aus einer positiven reellen Zahl, den Fall wenn der Startwert 0 beträgt und den Fall dass der Startwert ungültig ist.

Für den normalen Fall habe ich als Startwert $3.3374811e94$ genommen, da bei diesem Wert jede Schleife genau einmal wiederholt wird, und somit jede Sprungabzweigung im Laufe der Abarbeitung einmal genutzt wird. Für den Fall dass der Startwert 0 beträgt, wird die Abarbeitung sofort beendet und die 0 als Ergebnis ausgegeben. Für den Fall, dass der Startwert ungültig ist, wird kein Ergebnis ausgegeben, dafür ein aber ein Fehlerflag gesetzt. Ich habe für den Fehlerfall nur ein Simulationsergebnis angegeben, da die Abarbeitung in allen drei Fällen, negative Zahl, Zahlenüberlauf und 'Not-a-Number' exakt gleich aussieht.

Die Baseline wurde in den Simulationsprotokollen jeweils so platziert, dass sie im zweiten Takt der Startphase liegt, wo die Flags V und Z ausgewertet werden, um den weiteren Fortlauf des Algorithmus zu bestimmen. Der Cursor 'TimeA' liegt jeweils in dem Takt, in dem das Ergebnis ausgegeben wird.

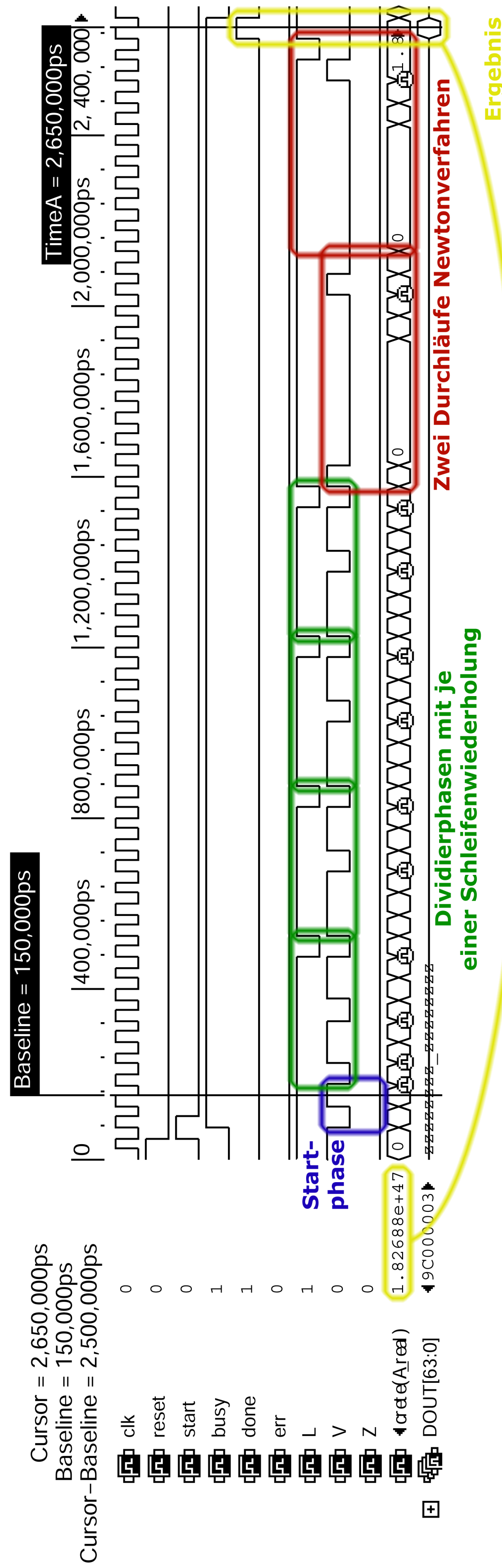


Abbildung 10: normaler Durchlauf des Algorithmus mit Startwert $3.3374811e94$

Abbildung 11: Verhalten der Schaltung bei Startwert 0

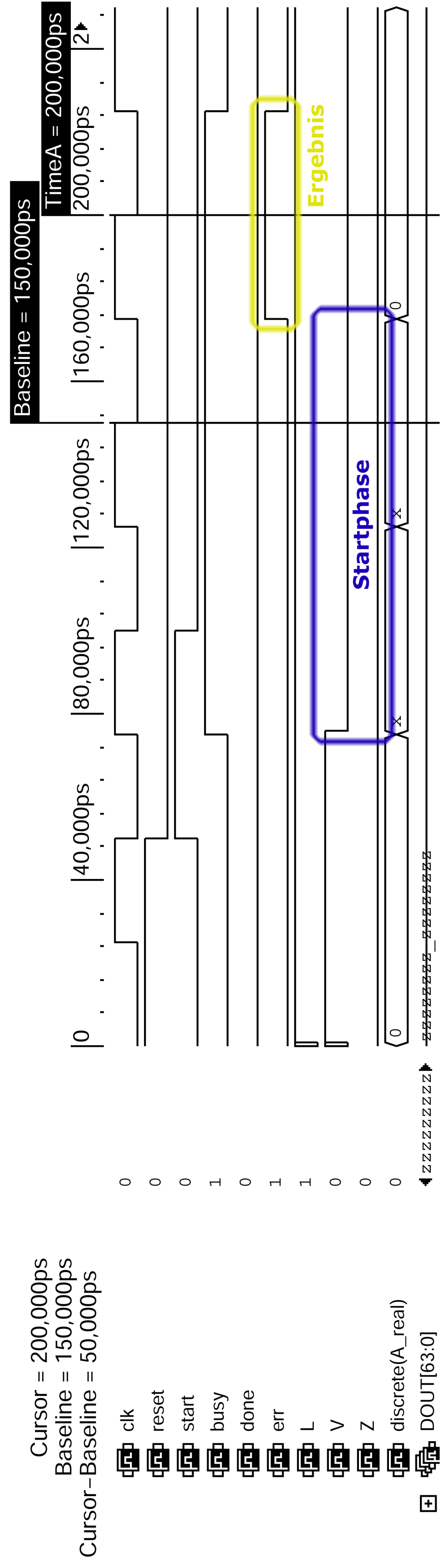


Abbildung 12: Verhalten der Schaltung bei fehlerhaftem Startwert

4.2 nur Zustandsautomat, Gatterschaltung

Die folgenden Simulationsprotokolle enthalten alle möglichen Zustandsübergänge des Zustandsautomaten. Simuliert wurde dabei nur die reine Gatterschaltung der FSM. Die Testbench ist dabei so gehalten, dass alle Zustände und Zustandsübergänge mindestens einmal durchlaufen werden. Der Verilog-Code der kompletten Testbench ist im Anhang in Quellcode 4 zu sehen.

Zu Begin werden erst alle Startphasen durchgelaufen, die zu einem vorzeitigen Ende der Abarbeitung führen. Das sind der Fehlerfall, also Eingabe von negativen oder zu großen Zahlen, oder Werte die keine gültige Fließkommazahl sind (Not-a-Number). Aber auch die Eingabe von 0 führt zur sofortigen Beendigung des Algorithmus, da das Ergebnis mit 0 schon feststeht, und eine weitere Bearbeitung durch den Algorithmus am Ende zu einem Division-durch-0-Fehler führen würde. Eine Besonderheit ist der Fall, bei dem die Flags eine ungültige Zahl und den Wert 0 gleichzeitig anzeigen. Dieser Fall tritt in der Praxis nie auf, da eine Zahl nicht gleichzeitig zu groß bzw. 'Not-a-Number' und 0 sein kann. Es gibt zwar den theoretischen Fall der -0, aber auch bei Eingabe von -0 wird vom Multiplizierer kein Vorzeichenflag gesetzt, sondern nur das Zero-Flag. Aufgrund dieses theoretischen Falles habe ich mich aber dazu entschieden in diesem Fall die 0 als Ergebnis auszugeben und keinen Fehler.

Im weiteren Verlauf der Simulation wird jede Schleife einmal wiederholt, um jeden Sprung in beide Richtungen einmal durchgeführt zu haben. An den Zustandsübergängen mit einer Auswertung der Flags L, V oder Z habe ich in den Simulationsprotokollen jeweils eine Markierung gesetzt. Die Funktionsfähigkeit des start-Flags ist am Anfang gegeben, da bei Setzung dieses Flags der Algorithmus korrekt startet. Im letzten Simulationsprotokoll in Abbildung 17 ist an der letzten steigenden Taktflanke auch zu erkennen, dass die FSM im IDLE-Zustand bleibt, wenn das start-Flag nicht gesetzt ist.

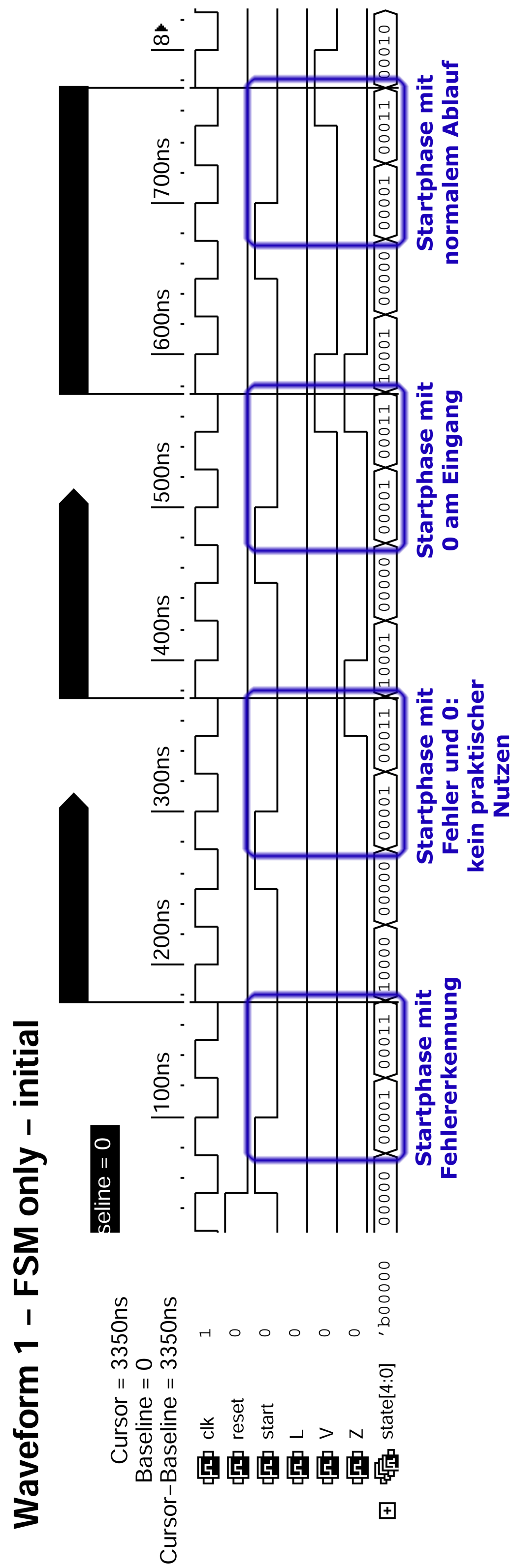


Abbildung 13: Die verschiedenen möglichen Startphasen des Algorithmus

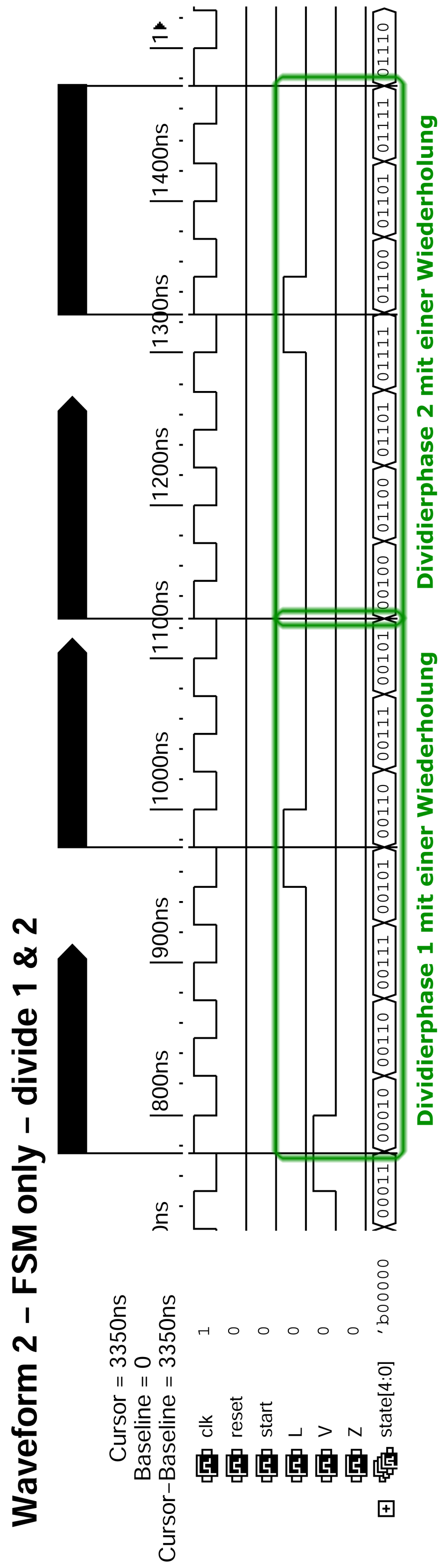


Abbildung 14: Divisionsphasen 1 und 2

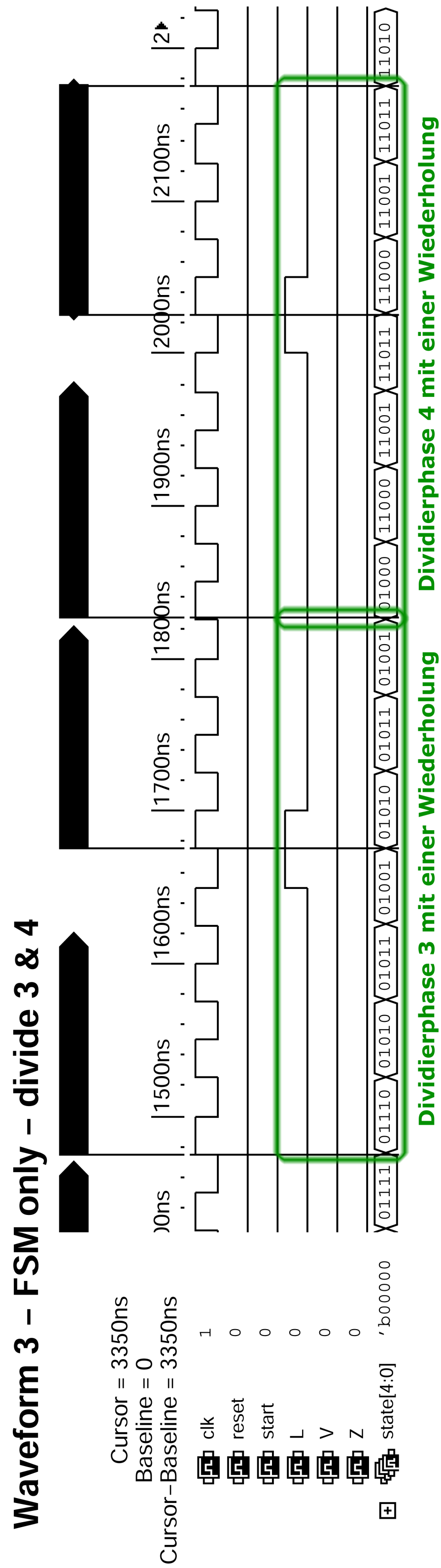


Abbildung 15: Divisionsphasen 3 und 4

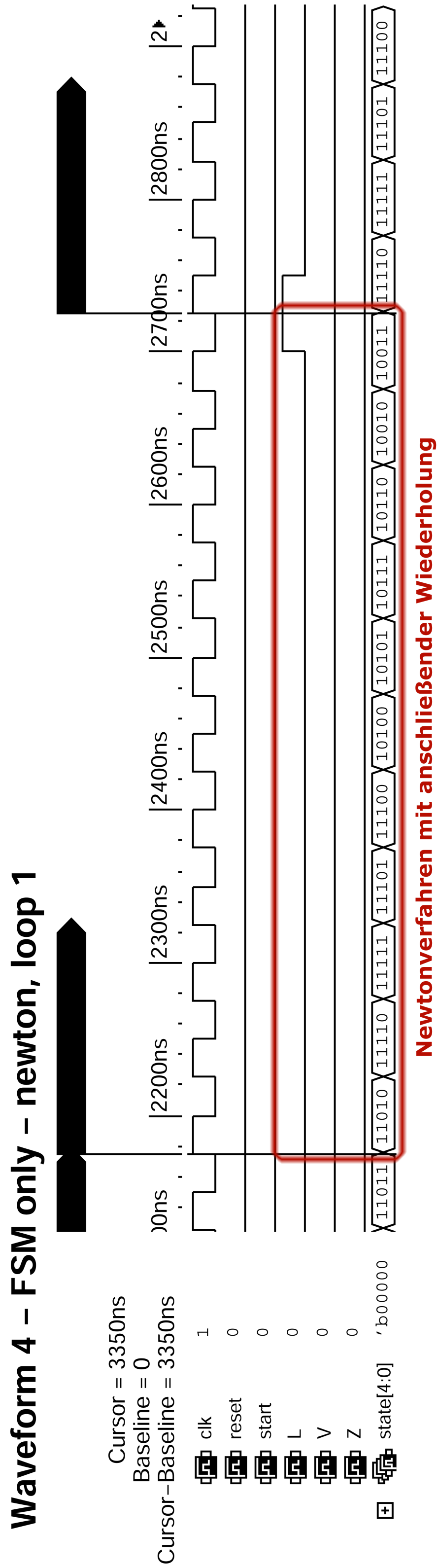


Abbildung 16: Erster Durchlauf des Newtonverfahrens mit Schleifenrücksprung

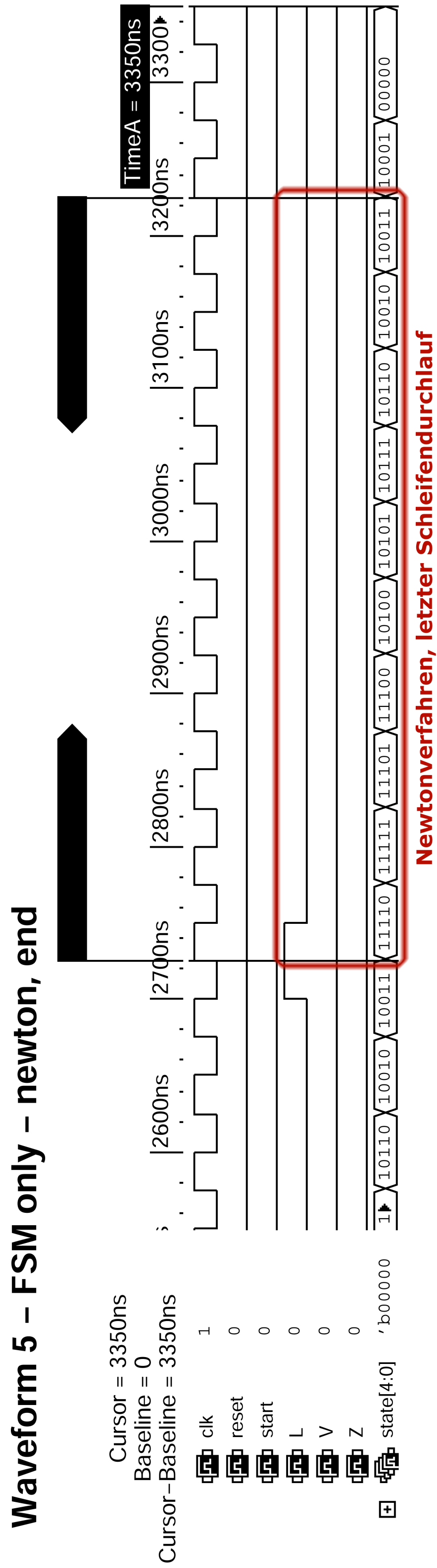


Abbildung 17: Letzter Durchlauf des Newtonverfahrens mit Verlassen der Schleife

4.3 Gesamtschaltung, FSM als Gatterschaltung

In diesem Abschnitt wurden lediglich die Simulationen aus Abschnitt 4.1 wiederholt. Diesmal wurde nur statt der Verilog-Beschreibung des Zustandsautomaten die Gatterschaltung benutzt. Da beide Simulationsreihen jeweils das gleiche Ergebnis zeigen, kann man davon ausgehen, dass sowohl die Verilog-Beschreibung als auch die Gatterschaltung dasselbe Verhalten zeigen.

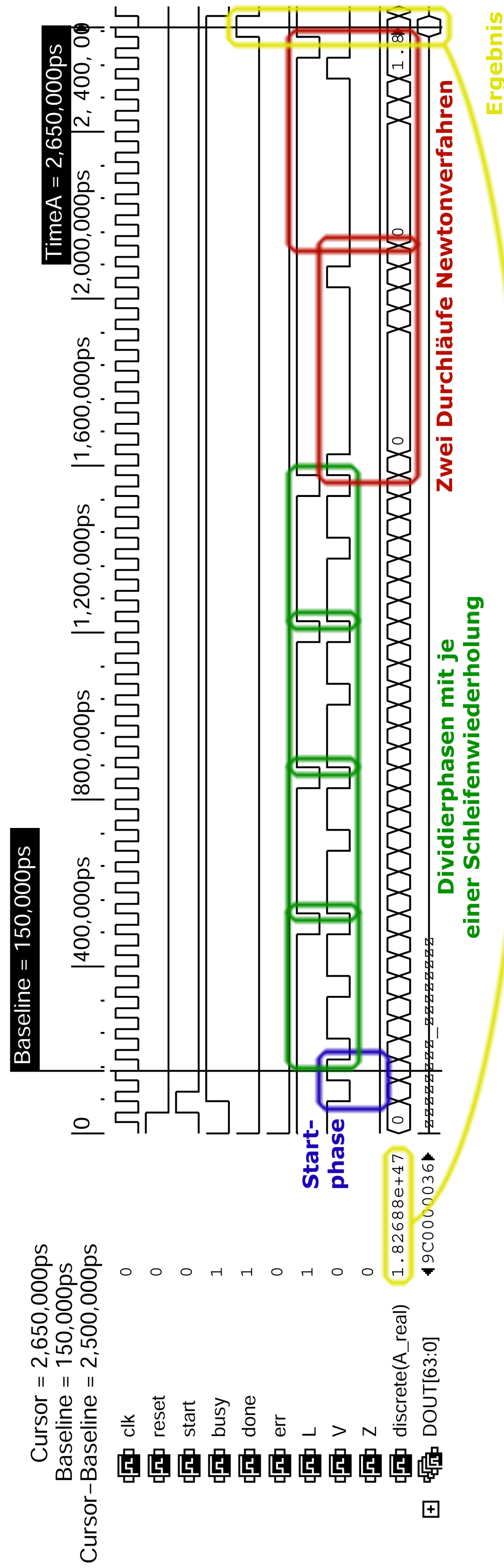


Abbildung 18: normaler Durchlauf des Algorithmus mit Startwert $3.3374811e94$

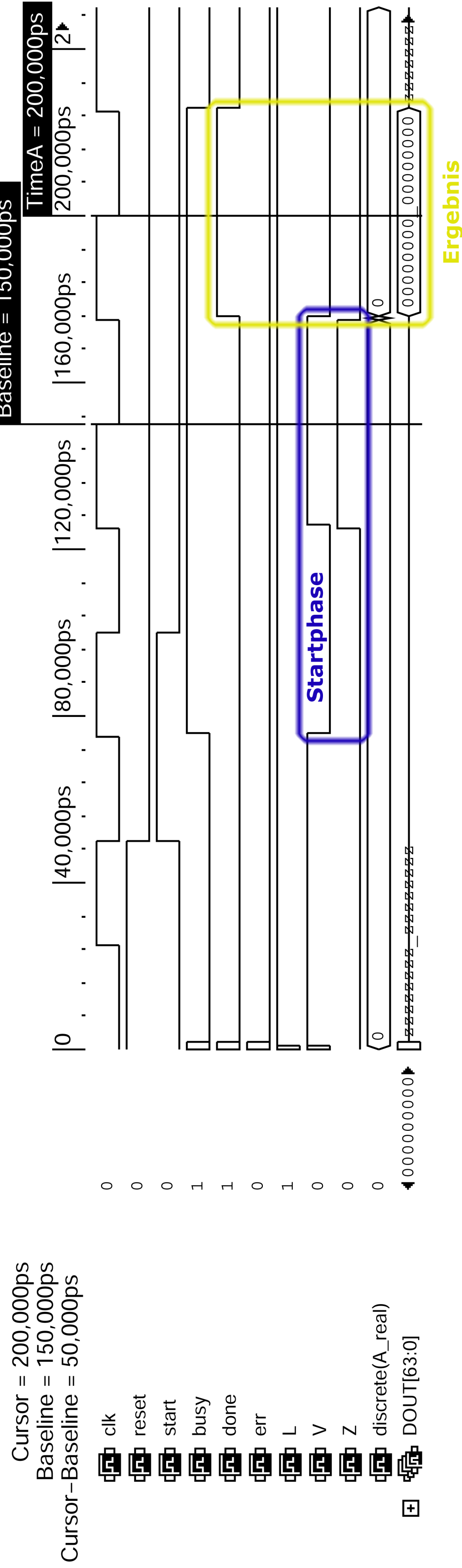


Abbildung 19: Verhalten der Schaltung bei Startwert 0

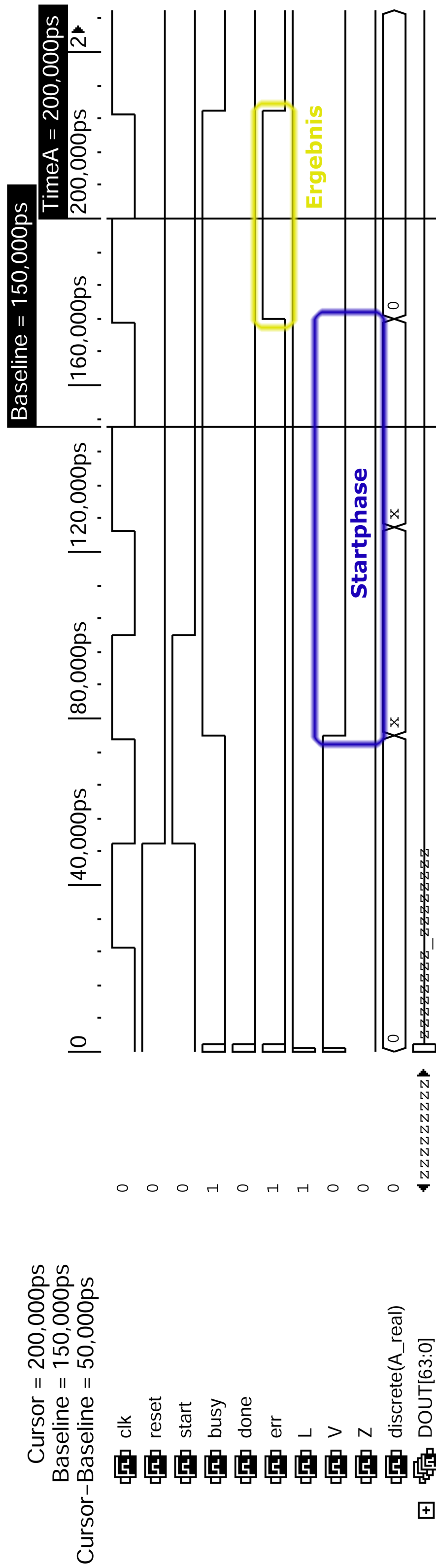


Abbildung 20: Verhalten der Schaltung bei fehlerhaftem Startwert

5 Zusammenfassung

In dieser Arbeit sollte an einem frei gewählten Algorithmus beispielhaft mit Hilfe gegebener Tools ein Schaltkreis entworfen werden. Ich habe hierfür eine spezielle Form des Newtonschen Näherungsverfahrens zur Berechnung der Quadratwurzel genommen. Mit verschiedensten Mitteln habe ich versucht den Algorithmus sowohl schneller zu machen, als auch möglichst sparsam mit Ressourcen wie Flächenbedarf und Stromaufnahme umzugehen.

Der Algorithmus ist allerdings nur sehr beispielhaft und die ganze Schaltung ohne praktischen Nutzen. Die gesamte Schaltung braucht trotz Optimierungen immer noch bis zu 135 Takte um zum Ergebnis zu kommen. Dieser Fall ist im Simulator unter `http://olunczek.de/ice-test/` für den Startwert $4.286e301$ nachzuvollziehen. Bei diesem Wert sind die meisten Durchläufe der Divisionsphase nötig. Das eigentliche Newtonverfahren braucht dank quadratischer Konvergenz und der Vorberechnung auf einen Wert von maximal dem doppelten des Zielwertes nie mehr als sieben Durchläufe. Die Berechnung der Quadratwurzel ist mit dem Vorwissen des Gleitkommaformates wesentlich einfacher zu bewerkstelligen. Die vereinfachte Form der Gleitkommazahlen lautet ja $z = a2^b$, wobei a zwischen 1 und 2 liegt. Die Wurzel berechnet sich demzufolge zu $\sqrt{z} = \sqrt{a2^b} = \sqrt{a}\sqrt{2^b}$. Die Quadratwurzel der Potenz lässt sich einfach berechnen. Und da a in einem sehr kleinen Bereich definiert ist, kann man dort die Wurzelfunktion durch eine einfache Gleichung annähern, so dass keine iterativen Berechnungen nötig sind.

Der Datenpfad wurde so konstruiert, dass er möglichst kleine einfache Register nutzt. Auch bei den Register-Transfer-Folgen wurden alle Zustände für die Optimierungsschritte genutzt, so dass es keine Leerzustände gibt. Dort ist also nur noch wenig Optimierungspotential. Beim Zustandsautomaten gibt es sicher noch viele Möglichkeiten zur Optimierung. Ich habe hier eine Zustandskodierung genutzt, die ich für gut halte. Es gibt aber sicher noch bessere, mit denen man die FSM kleiner und sparsamer gestalten kann. Die Anzahl von möglichen Kodierungen ist aber so groß und nur schwer überschaubar, dass es schwierig und aufwändig ist das Optimum zu finden.

Die vorliegende Arbeit zeigt nur Einblick in die wichtigsten Schritte des Schaltkreisentwurfs und gibt Einblicke zu verschiedenen Optimierungsmöglichkeiten. Das ganze ist aber nur beispielhaft und erhebt keinen Anspruch auf Vollständigkeit.

A Quellcode

Quellcode 1: Verilog-Code des Zustandsautomaten

```
1  module SQRT_FSM (L, V, Z, clk, reset, start, state);
2
3    parameter IDLE         = 5'b00000;
4    parameter START_MUL    = 5'b00001;
5    parameter START_ADD    = 5'b00011;
6    parameter DIV1_START   = 5'b00010;
7    parameter DIV1_SQR     = 5'b00110;
8    parameter DIV1_SUB     = 5'b00111;
9    parameter DIV1_MUL     = 5'b00101;
10   parameter DIV2_START   = 5'b00100;
11   parameter DIV2_SQR     = 5'b01100;
12   parameter DIV2_SUB     = 5'b01101;
13   parameter DIV2_MUL     = 5'b01111;
14   parameter DIV3_START   = 5'b01110;
15   parameter DIV3_SQR     = 5'b01010;
16   parameter DIV3_SUB     = 5'b01011;
17   parameter DIV3_MUL     = 5'b01001;
18   parameter DIV4_START   = 5'b01000;
19   parameter DIV4_SQR     = 5'b11000;
20   parameter DIV4_SUB     = 5'b11001;
21   parameter DIV4_MUL     = 5'b11011;
22   parameter NEWTON_START = 5'b11010;
23   parameter NEWTON_DIV2  = 5'b11110;
24   parameter NEWTON_DIV3  = 5'b11111;
25   parameter NEWTON_DIV4  = 5'b11101;
26   parameter NEWTON_DIV5  = 5'b11100;
27   parameter NEWTON_DIV6  = 5'b10100;
28   parameter NEWTON_DIV7  = 5'b10101;
29   parameter NEWTON_ADD   = 5'b10111;
30   parameter NEWTON_MUL   = 5'b10110;
31   parameter NEWTON_SUB   = 5'b10010;
32   parameter NEWTON_DIV   = 5'b10011;
33   parameter END          = 5'b10001;
34   parameter ERR          = 5'b10000;
35
36   input L;
37   input V;
38   input Z;
39   input clk;
40   input reset;
41   input start;
42
43   output state;
44
45   reg [4:0] state;
46   reg [4:0] next_state;
47
```

```
48   always @ (posedge clk or posedge reset)
49     if (reset) begin
50       state <= IDLE;
51     end
52     else begin
53       state <= next_state;
54     end
55
56   always @ (L or V or Z or start or state)
57     case (state)
58       IDLE          : if (start) begin
59                         next_state = START_MUL;
60                       end
61                       else begin
62                         next_state = IDLE;
63                       end
64       START_MUL     : next_state = START_ADD;
65       START_ADD     : if (Z) begin
66                         next_state = END;
67                       end
68                       else begin
69                         if (V) begin
70                           next_state = DIV1_START;
71                         end
72                         else begin
73                           next_state = ERR;
74                         end
75                       end
76       DIV1_START    : next_state = DIV1_SQR;
77       DIV1_SQR      : next_state = DIV1_SUB;
78       DIV1_SUB      : next_state = DIV1_MUL;
79       DIV1_MUL      : if (L) begin
80                         next_state = DIV1_SQR;
81                       end
82                       else begin
83                         next_state = DIV2_START;
84                       end
85       DIV2_START    : next_state = DIV2_SQR;
86       DIV2_SQR      : next_state = DIV2_SUB;
87       DIV2_SUB      : next_state = DIV2_MUL;
88       DIV2_MUL      : if (L) begin
89                         next_state = DIV2_SQR;
90                       end
91                       else begin
92                         next_state = DIV3_START;
93                       end
94       DIV3_START    : next_state = DIV3_SQR;
95       DIV3_SQR      : next_state = DIV3_SUB;
96       DIV3_SUB      : next_state = DIV3_MUL;
97       DIV3_MUL      : if (L) begin
98                         next_state = DIV3_SQR;
```

```
 99                         end
100                         else begin
101                            next_state = DIV4_START;
102                         end
103           DIV4_START    : next_state = DIV4_SQR;
104           DIV4_SQR      : next_state = DIV4_SUB;
105           DIV4_SUB      : next_state = DIV4_MUL;
106           DIV4_MUL      : if (L) begin
107                            next_state = DIV4_SQR;
108                         end
109                         else begin
110                            next_state = NEWTON_START;
111                         end
112           NEWTON_START  : next_state = NEWTON_DIV2;
113           NEWTON_DIV2   : next_state = NEWTON_DIV3;
114           NEWTON_DIV3   : next_state = NEWTON_DIV4;
115           NEWTON_DIV4   : next_state = NEWTON_DIV5;
116           NEWTON_DIV5   : next_state = NEWTON_DIV6;
117           NEWTON_DIV6   : next_state = NEWTON_DIV7;
118           NEWTON_DIV7   : next_state = NEWTON_ADD;
119           NEWTON_ADD    : next_state = NEWTON_MUL;
120           NEWTON_MUL    : next_state = NEWTON_SUB;
121           NEWTON_SUB    : next_state = NEWTON_DIV;
122           NEWTON_DIV    : if (L) begin
123                            next_state = NEWTON_DIV2;
124                         end
125                         else begin
126                            next_state = END;
127                         end
128           ERR           : next_state = IDLE;
129           END           : next_state = IDLE;
130           default       : next_state = IDLE;
131        endcase
132
133  endmodule
```

Quellcode 2: Verilog-Code der Steuerlogik

```
module SQRT_CtrlLogic (state, ALU_B, A_ALU, A_DIV, A_MUL, A_NR,
                       B_ALU, B_DIV, B_DOUT, B_MUL, B_TMP, B_VL,
                       DIN_A, DIV_A, K2pm1_A, K2pm5_A, K2pm25_A,
                       K2pm125_A, MUL_A, MUL_B, NR_B, OR_B,
                       TMP_OR, VL_A, busy, div, sub, done, err);

  parameter IDLE         = 5'b00000;
  parameter START_MUL    = 5'b00001;
  parameter START_ADD    = 5'b00011;
  parameter DIV1_START   = 5'b00010;
  parameter DIV1_SQR     = 5'b00110;
  parameter DIV1_SUB     = 5'b00111;
  parameter DIV1_MUL     = 5'b00101;
  parameter DIV2_START   = 5'b00100;
  parameter DIV2_SQR     = 5'b01100;
  parameter DIV2_SUB     = 5'b01101;
  parameter DIV2_MUL     = 5'b01111;
  parameter DIV3_START   = 5'b01110;
  parameter DIV3_SQR     = 5'b01010;
  parameter DIV3_SUB     = 5'b01011;
  parameter DIV3_MUL     = 5'b01001;
  parameter DIV4_START   = 5'b01000;
  parameter DIV4_SQR     = 5'b11000;
  parameter DIV4_SUB     = 5'b11001;
  parameter DIV4_MUL     = 5'b11011;
  parameter NEWTON_START = 5'b11010;
  parameter NEWTON_DIV2  = 5'b11110;
  parameter NEWTON_DIV3  = 5'b11111;
  parameter NEWTON_DIV4  = 5'b11101;
  parameter NEWTON_DIV5  = 5'b11100;
  parameter NEWTON_DIV6  = 5'b10100;
  parameter NEWTON_DIV7  = 5'b10101;
  parameter NEWTON_ADD   = 5'b10111;
  parameter NEWTON_MUL   = 5'b10110;
  parameter NEWTON_SUB   = 5'b10010;
  parameter NEWTON_DIV   = 5'b10011;
  parameter END          = 5'b10001;
  parameter ERR          = 5'b10000;

  input [4:0] state;

  output ALU_B;
  output A_ALU;
  output A_DIV;
  output A_MUL;
  output A_NR;
  output B_ALU;
  output B_DIV;
  output B_DOUT;
  output B_MUL;
```

```
51    output B_TMP;
52    output B_VL;
53    output DIN_A;
54    output DIV_A;
55    output K2pm1_A;
56    output K2pm5_A;
57    output K2pm25_A;
58    output K2pm125_A;
59    output MUL_A;
60    output MUL_B;
61    output NR_B;
62    output OR_B;
63    output TMP_OR;
64    output VL_A;
65    output busy;
66    output div;
67    output sub;
68    output done;
69    output err;
70
71    reg ALU_B;
72    reg A_ALU;
73    reg A_DIV;
74    reg A_MUL;
75    reg A_NR;
76    reg B_ALU;
77    reg B_DIV;
78    reg B_DOUT;
79    reg B_MUL;
80    reg B_TMP;
81    reg B_VL;
82    reg DIN_A;
83    reg DIV_A;
84    reg K2pm1_A;
85    reg K2pm5_A;
86    reg K2pm25_A;
87    reg K2pm125_A;
88    reg MUL_A;
89    reg MUL_B;
90    reg NR_B;
91    reg OR_B;
92    reg TMP_OR;
93    reg VL_A;
94    reg busy;
95    reg div;
96    reg sub;
97    reg done;
98    reg err;
99
100   always @ (state)
101     begin
```

```
102         ALU_B       = 1'b0;
103         A_ALU       = 1'b0;
104         A_DIV       = 1'b0;
105         A_MUL       = 1'b0;
106         A_NR        = 1'b0;
107         B_ALU       = 1'b0;
108         B_DIV       = 1'b0;
109         B_DOUT      = 1'b0;
110         B_MUL       = 1'b0;
111         B_TMP       = 1'b0;
112         B_VL        = 1'b0;
113         DIN_A       = 1'b0;
114         DIV_A       = 1'b0;
115         K2pm1_A     = 1'b0;
116         K2pm5_A     = 1'b0;
117         K2pm25_A    = 1'b0;
118         K2pm125_A   = 1'b0;
119         MUL_A       = 1'b0;
120         MUL_B       = 1'b0;
121         NR_B        = 1'b0;
122         OR_B        = 1'b0;
123         TMP_OR      = 1'b0;
124         VL_A        = 1'b0;
125         busy        = 1'b1;
126         div         = 1'b0;
127         sub         = 1'b0;
128         done        = 1'b0;
129         err         = 1'b0;
130
131         case (state)
132           IDLE          : begin
133                             DIN_A       = 1'b1;
134                             A_NR        = 1'b1;
135                             busy        = 1'b0;
136                           end
137           START_MUL     : begin
138                             K2pm1_A     = 1'b1;
139                             A_MUL       = 1'b1;
140                             NR_B        = 1'b1;
141                             B_MUL       = 1'b1;
142                             B_VL        = 1'b1;
143                           end
144           START_ADD     : begin
145                             K2pm1_A     = 1'b1;
146                             A_ALU       = 1'b1;
147                             MUL_B       = 1'b1;
148                             B_ALU       = 1'b1;
149                           end
150           DIV1_START    : begin
151                             K2pm125_A   = 1'b1;
152                             A_MUL       = 1'b1;
```

```
153                                 ALU_B       = 1'b1;
154                                 B_MUL       = 1'b1;
155                                 B_TMP       = 1'b1;
156                               end
157             DIV1_SQR        : begin
158                                 MUL_A       = 1'b1;
159                                 A_MUL       = 1'b1;
160                                 A_NR        = 1'b1;
161                                 MUL_B       = 1'b1;
162                                 B_MUL       = 1'b1;
163                                 TMP_OR      = 1'b1;
164                               end
165             DIV1_SUB        : begin
166                                 VL_A        = 1'b1;
167                                 A_ALU       = 1'b1;
168                                 MUL_B       = 1'b1;
169                                 B_ALU       = 1'b1;
170                                 sub         = 1'b1;
171                               end
172             DIV1_MUL        : begin
173                                 K2pm125_A   = 1'b1;
174                                 A_MUL       = 1'b1;
175                                 NR_B        = 1'b1;
176                                 B_MUL       = 1'b1;
177                                 B_TMP       = 1'b1;
178                               end
179             DIV2_START      : begin
180                                 K2pm25_A    = 1'b1;
181                                 A_MUL       = 1'b1;
182                                 OR_B        = 1'b1;
183                                 B_MUL       = 1'b1;
184                                 B_TMP       = 1'b1;
185                               end
186             DIV2_SQR        : begin
187                                 MUL_A       = 1'b1;
188                                 A_MUL       = 1'b1;
189                                 A_NR        = 1'b1;
190                                 MUL_B       = 1'b1;
191                                 B_MUL       = 1'b1;
192                                 TMP_OR      = 1'b1;
193                               end
194             DIV2_SUB        : begin
195                                 VL_A        = 1'b1;
196                                 A_ALU       = 1'b1;
197                                 MUL_B       = 1'b1;
198                                 B_ALU       = 1'b1;
199                                 sub         = 1'b1;
200                               end
201             DIV2_MUL        : begin
202                                 K2pm25_A    = 1'b1;
203                                 A_MUL       = 1'b1;
```

```
204                           NR_B        = 1'b1;
205                           B_MUL       = 1'b1;
206                           B_TMP       = 1'b1;
207                         end
208          DIV3_START   : begin
209                           K2pm5_A     = 1'b1;
210                           A_MUL       = 1'b1;
211                           OR_B        = 1'b1;
212                           B_MUL       = 1'b1;
213                           B_TMP       = 1'b1;
214                         end
215          DIV3_SQR     : begin
216                           MUL_A       = 1'b1;
217                           A_MUL       = 1'b1;
218                           A_NR        = 1'b1;
219                           MUL_B       = 1'b1;
220                           B_MUL       = 1'b1;
221                           TMP_OR      = 1'b1;
222                         end
223          DIV3_SUB     : begin
224                           VL_A        = 1'b1;
225                           A_ALU       = 1'b1;
226                           MUL_B       = 1'b1;
227                           B_ALU       = 1'b1;
228                           sub         = 1'b1;
229                         end
230          DIV3_MUL     : begin
231                           K2pm5_A     = 1'b1;
232                           A_MUL       = 1'b1;
233                           NR_B        = 1'b1;
234                           B_MUL       = 1'b1;
235                           B_TMP       = 1'b1;
236                         end
237          DIV4_START   : begin
238                           K2pm1_A     = 1'b1;
239                           A_MUL       = 1'b1;
240                           OR_B        = 1'b1;
241                           B_MUL       = 1'b1;
242                           B_TMP       = 1'b1;
243                         end
244          DIV4_SQR     : begin
245                           MUL_A       = 1'b1;
246                           A_MUL       = 1'b1;
247                           A_NR        = 1'b1;
248                           MUL_B       = 1'b1;
249                           B_MUL       = 1'b1;
250                           TMP_OR      = 1'b1;
251                         end
252          DIV4_SUB     : begin
253                           VL_A        = 1'b1;
254                           A_ALU       = 1'b1;
```

```
255                          MUL_B       = 1'b1;
256                          B_ALU       = 1'b1;
257                          sub         = 1'b1;
258                        end
259        DIV4_MUL      : begin
260                          K2pm1_A     = 1'b1;
261                          A_MUL       = 1'b1;
262                          NR_B        = 1'b1;
263                          B_MUL       = 1'b1;
264                          B_TMP       = 1'b1;
265                        end
266        NEWTON_START  : begin
267                          VL_A        = 1'b1;
268                          A_DIV       = 1'b1;
269                          OR_B        = 1'b1;
270                          B_DIV       = 1'b1;
271                          B_TMP       = 1'b1;
272                          div         = 1'b1;
273                        end
274        NEWTON_DIV2   : begin
275                          TMP_OR      = 1'b1;
276                        end
277        NEWTON_DIV3   : begin
278                        end
279        NEWTON_DIV4   : begin
280                        end
281        NEWTON_DIV5   : begin
282                        end
283        NEWTON_DIV6   : begin
284                        end
285        NEWTON_DIV7   : begin
286                        end
287        NEWTON_ADD    : begin
288                          DIV_A       = 1'b1;
289                          A_ALU       = 1'b1;
290                          OR_B        = 1'b1;
291                          B_ALU       = 1'b1;
292                        end
293        NEWTON_MUL    : begin
294                          K2pm1_A     = 1'b1;
295                          A_MUL       = 1'b1;
296                          ALU_B       = 1'b1;
297                          B_MUL       = 1'b1;
298                        end
299        NEWTON_SUB    : begin
300                          MUL_A       = 1'b1;
301                          A_ALU       = 1'b1;
302                          A_NR        = 1'b1;
303                          OR_B        = 1'b1;
304                          B_ALU       = 1'b1;
305                          sub         = 1'b1;
```

```
306                         end
307         NEWTON_DIV    : begin
308                           VL_A      = 1'b1;
309                           A_DIV     = 1'b1;
310                           NR_B      = 1'b1;
311                           B_DIV     = 1'b1;
312                           B_TMP     = 1'b1;
313                           div       = 1'b1;
314                         end
315         END           : begin
316                           NR_B      = 1'b1;
317                           B_DOUT    = 1'b1;
318                           done      = 1'b1;
319                         end
320         ERR           : begin
321                           err       = 1'b1;
322                         end
323       endcase
324     end
325
326 endmodule
```

Quellcode 3: Verilog-Code der Testbench der gesamten Schaltung

```verilog
module SQRT_top_tb;

  parameter CYCLE = 50;

  wire [63:0] DOUT;
  wire busy;
  wire done;
  wire err;

  reg  [63:0] DIN;
  reg  clk;
  reg  reset;
  reg  start;

  SQRT_top SQRT (
    .DOUT(DOUT),
    .busy(busy),
    .done(done),
    .err(err),
    .DIN(DIN),
    .clk(clk),
    .reset(reset),
    .start(start)
  );

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

  initial begin
    reset = 1'b1;
    #(CYCLE);
    reset = 1'b0;
  end

  initial begin
    start = 0;
    #(CYCLE);
    DIN = $realtobits(3.3374811e94);
    //DIN = $realtobits(-0.0);
    //DIN = $realtobits(0.0);
    //DIN = $realtobits(-1.1);
    //DIN = 64'b01111111111100000000000000000000000000
    //                              00000000000000000000000000; // +INF
    //DIN = 64'b11111111111100000000000000000000000000
    //                              00000000000000000000000000; // -INF
    //DIN = 64'b01111111111100000000000000000000000000
    //                              00000000010000000000000000; // NaN
    start = 1;
    #(CYCLE);
    start = 0;
```

```
51
52      while (busy == 1) begin
53         #(CYCLE);
54      end
55
56      $finish;
57   end
58
59 endmodule
```

Quellcode 4: Verilog-Code der Testbench des Zustandsautomaten

```
1  module SQRT_FSM_tb;
2
3    parameter CYCLE = 50;
4
5    wire [4:0] state;
6
7    reg  clk;
8    reg  reset;
9    reg  start;
10   reg  L;
11   reg  V;
12   reg  Z;
13
14   SQRT_FSM FSM (
15     .state(state),
16     .clk(clk),
17     .reset(reset),
18     .start(start),
19     .L(L),
20     .V(V),
21     .Z(Z)
22   );
23
24   initial clk = 1'b0;
25   always #(CYCLE/2) clk = ~clk;
26
27   initial begin
28     reset = 1'b1;
29     start = 0;
30     L     = 0;
31     V     = 0;
32     Z     = 0;
33     #(CYCLE);
34     reset = 1'b0;
35     start = 1;
36     #(CYCLE);      // start V=0 Z=0 => error
37     start = 0;
38     #(3*CYCLE);
39     start = 1;
40     #(CYCLE);      // start V=0 Z=1 => unused
41     start = 0;
42     #(CYCLE);
43     Z     = 1;
44     #(CYCLE);
45     Z     = 0;
46     #(CYCLE);
47     start = 1;
48     #(CYCLE);      // start V=1 Z=1 => zero
49     start = 0;
50     #(CYCLE);
```

```
51        Z       = 1;
52        V       = 1;
53        #(CYCLE);
54        Z       = 0;
55        V       = 0;
56        #(CYCLE);
57        start = 1;
58        #(CYCLE);        // start V=1 Z=0 => normal
59        start = 0;
60        #(CYCLE);
61        V       = 1;
62        #(CYCLE);
63        V       = 0;
64        #(3*CYCLE);      // div1
65        L       = 1;
66        #(CYCLE);        // loop back
67        L       = 0;
68        #(6*CYCLE);      // div2
69        L       = 1;
70        #(CYCLE);        // loop back
71        L       = 0;
72        #(6*CYCLE);      // div3
73        L       = 1;
74        #(CYCLE);        // loop back
75        L       = 0;
76        #(6*CYCLE);      // div4
77        L       = 1;
78        #(CYCLE);        // loop back
79        L       = 0;
80        #(13*CYCLE);     // newton
81        L       = 1;
82        #(CYCLE);        // loop back
83        L       = 0;
84        #(12*CYCLE);     // newton end
85        $finish;
86     end
87
88  endmodule
```

Quellcode 5: Perl-Code des Simulators (`http://olunczek.de/ice-test/`, 3. Spalte)

```
1  #!/usr/bin/perl -w
2
3  use strict;
4  use CGI::Carp qw(fatalsToBrowser);
5  use vars qw($i);
6  use vars qw($newresult);
7  use vars qw($clocks);
8
9  my $value = $ENV{'QUERY_STRING'};
10 my $oldresult;
11
12 $newresult = $value * 0.5 + 0.5;
13 $clocks = 3;
14
15 format STDOUT =
16 @<<<< @<<<<<<<<<<<<<<<<<<<<< @<<<<<<<<
17 $i, $newresult, $clocks
18 .
19
20 print "Content-type: text/html\n\n";
21 print "<html><head><title>Sqrt nach Newton-Raphson</title>";
22 print "</head><body>";
23 print "<pre><b><u>Startwert:</u></b> $newresult\n\n";
24
25 print "<u>Div 2<sup>125</sup>:</u>\n\n";
26 print "<b>Iter   x<sub>i</sub>                          Takte</b>\n";
27 $i = 0;
28 do {
29   if($i > 0) {
30     write;
31   }
32   if($i >= 9999) {
33     exit;
34   }
35   $oldresult = $newresult;
36   $newresult = 2.35099e-38 * $oldresult;
37   $i++;
38   $clocks = $clocks + 3;
39 } until(($newresult * $newresult) <= $value);
40 $newresult = $oldresult;
41 print "\nTakte: $clocks\n\n";
42 $clocks++;
43
44 print "\n<u>Div 2<sup>25</sup>:</u>\n\n";
45 print "<b>Iter   x<sub>i</sub>                          Takte</b>\n";
46 $i = 0;
47 do {
48   if($i > 0) {
49     write;
50   }
```

```
51     if($i >= 9999) {
52       exit;
53     }
54     $oldresult = $newresult;
55     $newresult = 2.98023e-08 * $oldresult;
56     $i++;
57     $clocks = $clocks + 3;
58   } until(($newresult * $newresult) <= $value);
59   $newresult = $oldresult;
60   print "\nTakte: $clocks\n\n";
61   $clocks++;
62
63   print "\n<u>Div 2<sup>5</sup>:</u>\n\n";
64   print "<b>Iter  x<sub>i</sub>                    Takte</b>\n";
65   $i = 0;
66   do {
67     if($i > 0) {
68       write;
69     }
70     if($i >= 9999) {
71       exit;
72     }
73     $oldresult = $newresult;
74     $newresult = 0.03125 * $oldresult;
75     $i++;
76     $clocks = $clocks + 3;
77   } until(($newresult * $newresult) <= $value);
78   $newresult = $oldresult;
79   print "\nTakte: $clocks\n\n";
80   $clocks++;
81
82   print "\n<u>Div 2:</u>\n\n";
83   print "<b>Iter  x<sub>i</sub>                    Takte</b>\n";
84   $i = 0;
85   do {
86     if($i > 0) {
87       write;
88     }
89     if($i >= 9999) {
90       exit;
91     }
92     $oldresult = $newresult;
93     $newresult = 0.5 * $oldresult;
94     $i++;
95     $clocks = $clocks + 3;
96   } until(($newresult * $newresult) <= $value);
97   $newresult = $oldresult;
98   print "\nTakte: $clocks\n\n";
99   $clocks++;
100
101  print "\n<u>Newton-Raphson:</u>\n\n";
```

```
102 print "<b>Iter   x<sub>i</sub>                         Takte</b>\n";
103 $i = 0;
104 do {
105   if($i >= 9999) {
106     exit;
107   }
108   $oldresult = $newresult;
109   $newresult = 0.5 * ($oldresult + $value / $oldresult);
110   $i++;
111   $clocks = $clocks + 10;
112   write;
113 } until($oldresult <= $newresult);
114 print "\n\n <b><u>Ergebnis:</u></b> $newresult\n";
115 $clocks++;
116 print "\n <b><u>Takte:</u></b> $clocks";
117
118 print "</pre></body></html>";
```

Literatur

[1] Wikipedia: *Newton-Verfahren*
http://de.wikipedia.org/wiki/Newton-Verfahren

[2] Wikipedia: *IEEE 754*
http://de.wikipedia.org/wiki/IEEE_754

[3] Wikipedia: *Gleitkommazahl*
http://de.wikipedia.org/wiki/Gleitkommazahl

[4] Wikipedia: *Heron-Verfahren*
http://de.wikipedia.org/wiki/Heron-Verfahren

[5] Achim Graupner, Daniel Matolin, Jens-Uwe Schlüßler: *Einführung in die Anwendung des Entwurfssystems CADENCE für die Lehrveranstaltung Schaltkreis- und Systementwurf* - 23. Oktober 2007
http://hpsn.et.tu-dresden.de/fileadmin/webfiles/lessons/vlsi_ice_07/Anleitung_ICE_2007.pdf

[6] austriamicrosystems AG: *c35 Cells*
http://asic.austriamicrosystems.com/databooks/c35/databook_c35_33/index.html