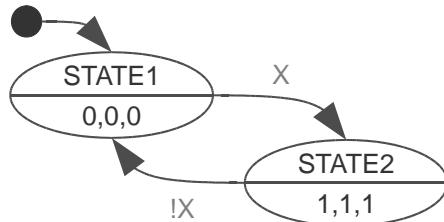


# **STDE**

## **State Transition Diagram Editor**



**Ein Werkzeug für Automaten-Erstellung und Codegenerierung**

**Autoren: Jan Montag, Andreas Schwenk, Georg Hartung**  
**Labor für Digitaltechnik/Technische Informatik**  
**Institut für Nachrichtentechnik**  
**Technische Hochschule Köln**

**Benutzerhandbuch**

**Inhaltsverzeichnis**

1.	Einführung.....	3
2.	Voraussetzungen und Installation.....	4
3.	Die Arbeitsoberfläche.....	5
4.	Dateioperationen und Einstellungen.....	6
4.1.	Anlegen eines neuen Projekts.....	6
4.2.	Öffnen eines bestehendes Projekts.....	6
4.3.	Speichern des geöffneten Projekts.....	7
4.4.	Das Dialogfeld: „Einstellungen“.....	7
5.	Erstellung eines Zustands-Übergangsgraphen.....	8
5.1.	Einfügen und Bearbeiten von Signalen.....	8
5.2.	Einfügen und Bearbeiten von Komponenten.....	9
5.2.1	Zustände (States).....	9
5.2.2	Übergänge (Transitions) und Startknoten.....	12
5.3.	Einfügen und Bearbeiten von Variablen.....	15
6.	Verifikation, Export (SCXML), Code-Generierung (C / VHDL).....	16
7.	Sonstiges.....	18
7.1.	Mehrfachselektion.....	18
7.2.	Export als Bilddatei.....	18
8.	Kontakt.....	18
9.	Revisionshistorie.....	19
	Anhang A: Syntaxregeln nach EBNF.....	20

## 1 Einführung

Bei dem vorliegenden Produkt „*STDE - State Transition Diagram Editor*“ handelt es sich um eine Software zur Erstellung erweiterter endlicher Zustandsautomaten mit Ausgaben. Gemäß der im Rahmen der Digitaltechnik verwendeten Varianten werden hier die beiden Typen „*Moore*“ und „*Mealy*“ unterstützt.

Neben der visuellen Darstellung ist es insbesondere möglich aus dem Automaten einen gültigen Source-Code zu erzeugen. Eine direkte Implementierung ermöglicht den Export nach „C“ (z. B. für die Programmierung von Mikroprozessoren) und / oder in die Hardwarebeschreibungssprache „VHDL“ (Programmierung von FPGAs). Alternativ kann auch ein Export in das Format SCXML erfolgen. Dies ermöglicht die Weiterverarbeitung mittels Programme Dritter.

Entstanden ist das Tool „*STDE*“ im Wintersemester 2011 / 2012 an der FH Köln im Rahmen des Moduls: „Softwarepraktikum 2“ des Bachelorstudiengangs Technische Informatik. Die Version 1.2 wurde von Georg Hartung erstellt.

## 2 Systemvoraussetzungen und Installation

Zum Ausführen der Software wird **Java SE 1.6 (oder neuer)** benötigt. Falls Java nicht auf Ihrem System installiert ist, folgen Sie bitte dem Link:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

*Hinweis für Apple-User:*

Ab **OS X „Lion“** wird Java standardgemäß nicht mehr vorinstalliert. Sie können die Installation wie folgt manuell vornehmen:

*Programme > Dienstprogramme > Java-Einstellungen.app  
(Applications > Utilities > Java Preferences.app)*

Zur **Installation von STDE** kopieren Sie „STDE.jar“ in ein beliebiges Verzeichnis. Unter den meisten Betriebssystem ist ein **Start** aus diesem Verzeichnis über Doppelklick möglich. Andernfalls führen Sie das Kommando „java -jar STDE.jar“ über die Konsole aus oder besuchen Sie:

<http://www.softpedia.com/get/Others/Miscellaneous/Jarfix.shtml>

Folgende Beispiel-Projekte sind im Lieferumfang enthalten:

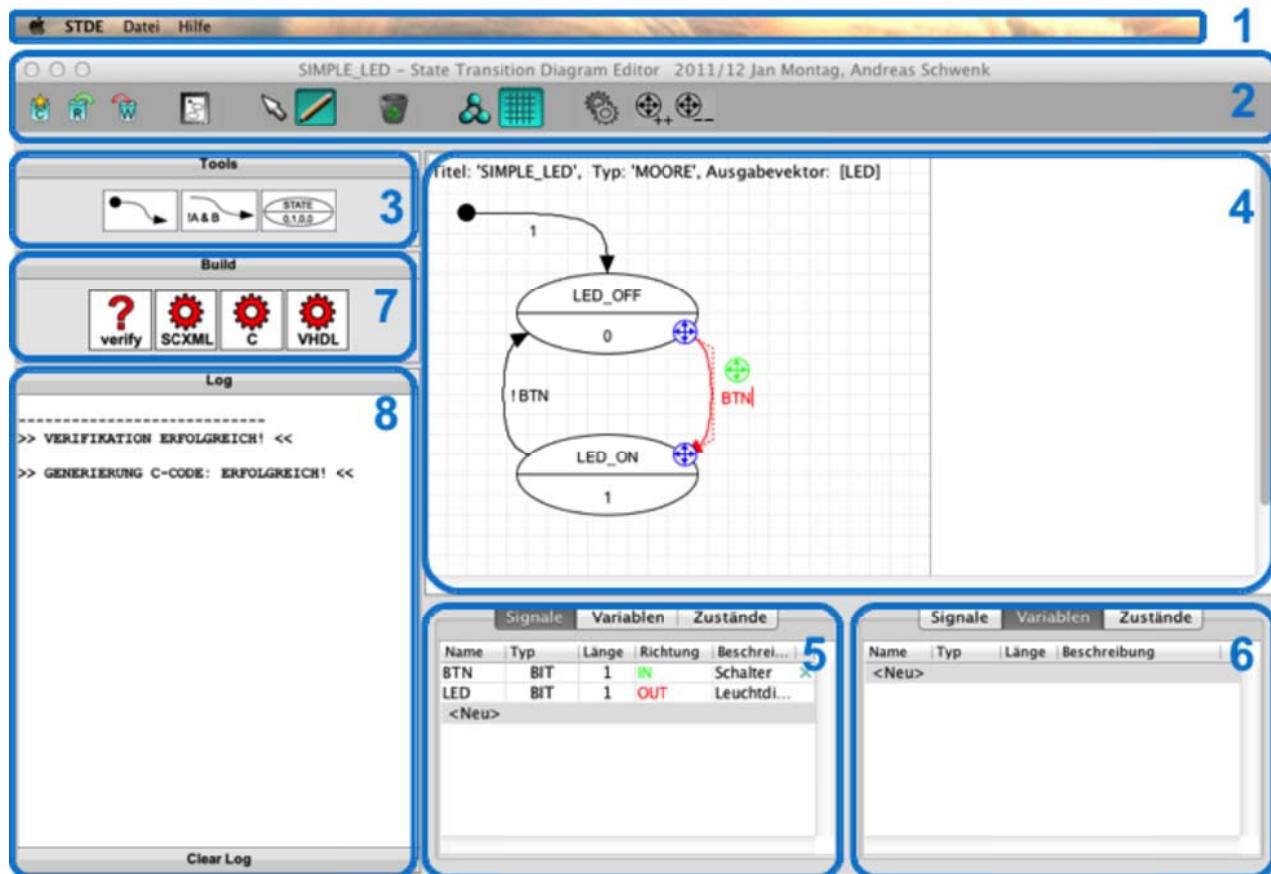
Projekt-/Dateiname	Beschreibung
bikelight_moore.stde	Moore-Beispiel aus der Veranstaltung GTI (FH Köln)
bikelight_mealy.stde	Mealy-Beispiel aus der Veranstaltung GTI (FH Köln)
simple_LED.stde	Sehr einfacher „Moore“-Automat. Eine LED wird über einen Schalter bedient.
simple_LED_mealy.stde	Sehr einfacher „Mealy“-Automat. Eine LED wird über einen Schalter bedient.
elevator.stde	Einfache Aufzugsteuerung
serial_transmitter.stde	Serieller Bitübertrager. Demonstriert die Nutzung von Variablen.
NIM-Spiel	Demonstriert das „Rechnen des Automaten“

### 3 Die Arbeitsoberfläche

Die in der folgenden Abbildung dargestellte Oberfläche zeigt den grundlegenden Aufbau von „STDE“. Die aufgeführten Punkte werden in den weiteren Kapiteln referenziert und näher erläutert.

*Hinweis: Die folgenden Screenshots wurden mit der ersten Version des STDE erzeugt.*

Bild 1: Arbeitsoberfläche



1. Menüleiste
2. Buttonleiste
3. Tools
4. Zeichenbereich
5. Signal-/Variablen-/Zustandstabelle 1
6. Signal-/Variablen-/Zustandstabelle 2
7. Verifikation / Export / Generierung
8. Fehler-Logging

## 4 Dateioperationen und Einstellungen

### 4.1 Anlegen eines neuen Projekts

Ein neues Projekt kann

- (a) beim Start von „STDE“ oder
- (b) über den Button „Neues Projekt anlegen“ (  ) in der Buttonleiste oder
- (c) über die Menüleiste: Datei > Neu...

erzeugt werden.



Das sich öffnende Dialogfenster ermöglicht die folgenden Einstellungen:

Feld / Option	Beschreibung
Name	Name des Automaten <i>Hinweis: Diese Option kann über das Einstellungsmenü jederzeit geändert werden.</i>
Breite / Höhe	Größe der Zeichenfläche <i>Hinweis: Diese Option kann über das Einstellungsmenü jederzeit geändert werden.</i>
Automaten-Modell	Typ des Automaten: „Moore“ oder „Mealy“ <i>Hinweis: Diese Option kann später <u>nicht</u> mehr geändert werden!</i>

### 4.2 Öffnen eines bestehenden Projekts

Ein bestehendes Projekt kann jederzeit wie folgt geöffnet werden:

- (a) über den Button „Projekt öffnen“ (  ) in der Buttonleiste oder
- (b) über die Menüleiste: Datei > öffnen...

Es wird empfohlen den „Standardpfad“ direkt nach dem Anlegen zu ändern, so dass beim Öffnen, Speichern, Generieren und Exportieren direkt zu diesen Verzeichnissen gewechselt wird.

Nähere Informationen finden Sie unter „4.4: Das Dialogfeld: 'Einstellungen'“.

### 4.3 Speichern des geöffneten Projekts

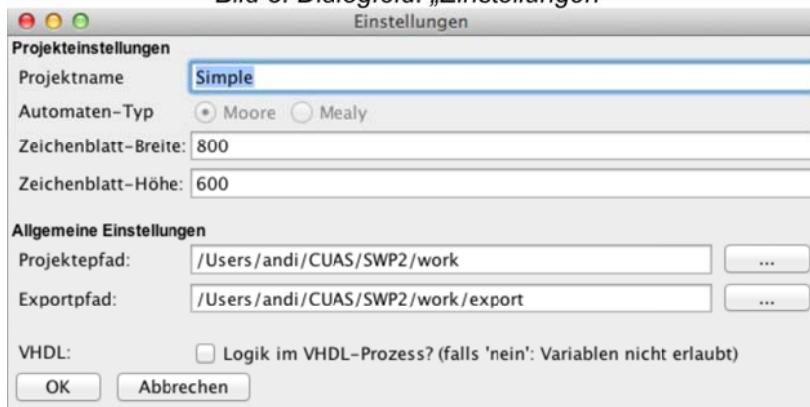
Ein geöffnetes Projekt kann gespeichert werden:

- (a) über den Button „Projekt speichern“ (  ) in der Buttonleiste oder
- (b) über die Menüleiste: Datei > speichern bzw. Datei > speichern unter...

### 4.4 Das Dialogfeld: „Einstellungen“

Sie erreichen das Dialogfeld „Einstellungen“ über das Symbol  in der Menüleiste.

Bild 3: Dialogfeld: „Einstellungen“



Die Optionen unter „Projekteinstellungen“ beziehen sich auf das aktuelle Projekt; die „Allgemeinen Einstellungen“ hingegen haben globalen Charakter (und werden in der Datei „preferences.txt“ gespeichert):

Feld / Option	Beschreibung	
Projektname	Name des aktuellen Projekts / des erstellten Automaten	
Automaten-Typ	Zeigt den Typ des Automaten in diesem Projekt. Eine Änderung ist nicht möglich.	
Zeichenblatt Breite / Höhe	Größe der Zeichenfläche. Bestimmt auch die Dimension exportierter Bilddateien	
Projektpfad	Gibt den Standarddateipfad für Projekte an. Beim Öffnen und Speichern eines Projekts wird automatisch zu diesem Pfad gewechselt. Erfolgt in den Textfeldern keine Eintragung, so ist der Standardpfad das „Home“-Verzeichnis des Benutzers (z. B. „Eigene Dateien“). Statt manuelles Füllen der Textfelder ist die Verzeichnisauswahl (nach Klick auf den entsprechenden rechten Button [...] ) über den Explorer möglich.	
Exportpfad	Analog zu „Projektpfad“, bezieht sich jedoch auf die exportierten / erzeugten Dateien (generierter C-/VHDL-Code, Export als SCXML)	
Logik im VHDL-Prozess	[X]	VHDL-Codegenerierung: die Zustandsüberführungs- sowie die Ausgabefunktion werden in einem VHDL-“Process“ implementiert
	[ ]	Implementierung der Überführungs- und Ausgabefunktion über bedingte Signalzuweisungen (als Multiplexer).

## 5 Erstellung eines Zustands-Übergangsgraphen

Ein Zustands-Übergangsdiagramm beschreibt (hier) einen endlichen Automaten mit Ausgaben. Unterschieden werden die beiden Typen „Moore“ und „Mealy“:

- (a) Moore: Die Ausgabe ist an den Zustand gebunden
- (b) Mealy: Die Ausgabe ist den Übergang gebunden

Beide Typen werden in „STDE“ unterstützt. Zunächst wird eine Beziehung zwischen der formalen Schreibweise aus der theoretischen Informatik mit den Bezeichnungen in „STDE“ hergestellt:

$$A = (Q, \Sigma, \Omega, \delta, \lambda, q_0, F)$$

<u>Formal</u>	<u>Beschreibung</u>	<u>Bezeichnung in „STDE“</u>
$Q$	Menge der Zustände	Zustände
$\Sigma$	endliches Eingabealphabet	Eingangssignale
$\Omega$	endliches Ausgabealphabet	Ausgabesignale
$\delta$	Übergangsfunktion	Übergangsbedingungen
$\lambda$	Ausgabefunktion	Ausgabevektor
$q_0$	Startzustand	Startknoten
$F$	Menge der Endzustände	(nicht vorhanden)

### 5.1 Einfügen und Bearbeiten von Signalen

Ein System ist ein Teil der realen Welt, das in eine Hülle eingebettet ist. Ein Signal ist eine bei einem **Namen** identifizierte Entität, die Daten überträgt. Die Schnittstellensignale des Systems kommunizieren mit der Außenwelt. Dabei können in „STDE“ die folgenden **Signal-Richtungen** unterschieden werden:

<u>Richtung</u>	<u>Beschreibung</u>
IN	Eingehendes Signal (von außen)
OUT	Ausgehendes Signal (nach außen)
INOUT	Signal, das sowohl für eingehende, als auch für ausgehenden Informationen genutzt werden kann

Neben der „Richtung“ ist der **Signaltyp** entscheidend: Grundsätzlich operiert ein digitales Signal auf Bitebene. Aus Gründen der Einfachheit können Signale zu einem Bus zusammengefasst werden. Folgende Typen stehen in „STDE“ zur Verfügung:

<u>Typ</u>	<u>Beschreibung</u>
BIT	Einzelnes BIT $\in \{0,1\}$
BIT_N	Bus: Konkatenation einzelner Bits
SIGNED	Interpretation als vorzeichenbehaftete Integer-Zahl
UNSIGNED	Interpretation als vorzeichenlose Integer-Zahl

Hinweis: BIT\_N und UNSIGNED unterscheiden sich hinsichtlich der Code-Generierung in VHDL (siehe „6. Verifikation, Export, Gode-Generierung“).

Obige Tabelle enthält jedoch keine Informationen über die Busbreite der Signale. Hierzu kann ein Signal bezüglich der **Länge** attribuiert werden. Die Länge gibt die Anzahl der verwendeten Bits an. Die Wahl ist nicht restriktiv, es empfiehlt sich jedoch zur Abbildung in die „C-Welt“ 2er Potenzen (1, 2, 4, 8, ...) zu verwenden. Im Rahmen der Generierung (siehe hierzu Kapitel 6) wird in „C“ auf den nächst größeren vorhanden Datentyp abgebildet. Zur Verwendung in FPGAs (hier: VHDL) kann es jedoch bei großen Projekten zur Ressourcenminimierung vorteilhafter sein, nicht auf eine 2er Potenzen „aufzurunden“.

Eine **Signal-Beschreibung** dient dem Verständnis des „Lesers“ eines erstellten Automaten. Diese wird bei der Codeerzeugung als Kommentar sichtbar.

Signale und deren Charakteristik können nun wie folgt in „STDE“ eingefügt werden:

- (1.) Wechseln Sie in einer der beiden „Signal-/Variablen-Zustandstabellen“ in den Reiter „Signale“:

Bild 4: Signal-Tabelle

Signale		Variablen	Zustände	
Name	Typ	Länge	Richtung	Beschrei...
BTN	BIT	1	IN	Schalter 
LED	BIT	1	OUT	Leuchtdi...
<Neu>				

- (2.) Klicken Sie auf „<Neu>“, um ein (weiteres) Signal einzufügen. Es erscheint eine leere Zeile.
- (3.) Beschreiben Sie Ihr Signal gemäß den oben beschriebenen Eigenschaften **Name, Typ, Länge, Richtung** und **ggf. Beschreibung**  
Hinweis: Die vorgeschlagene Länge nach Wahl des Typen ist nicht bindend
- (4.) Möchten Sie ein Signal wieder löschen, so fahren Sie mit der Maus über die entsprechende Tabellenzeile und klicken dann auf das Symbol  (ganz rechts).

Der Ausgabevektor – als Konkatenation einzelner Ausgabesignale – wird im Zeichenbereich dargestellt. Weitere Informationen hierzu werden im Abschnitt „5.1 Einfügen und Bearbeiten von Komponenten“ gegeben.

## 5.2 Einfügen und Bearbeiten von Komponenten

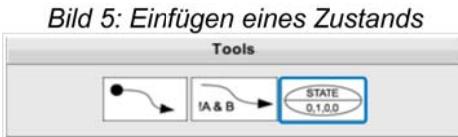
In „STDE“ gibt es zwei grundlegende Typen von Komponenten: **Zustände** und **Zustandsübergänge (= Transitionen)**.

### 5.2.1 Zustände (States)

Zustände beschreiben den aktuellen „Status“ des Automaten. Wie oben beschrieben hängt der Ausgabe-Vektor bei „Moore“ vom aktuellen Zustand ab; bei „Mealy“ vom Übergang, also vom Wechsel zwischen zwei Zuständen.

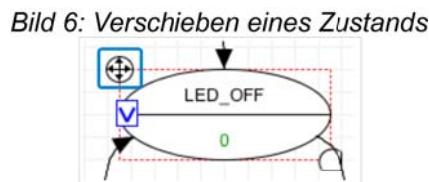
Zustände haben zur Identifikation einen eindeutigen **Namen**. Da Zustände im Zeichenbereich erstellt werden, haben sie auch eine **Position**. Die Beziehungen zwischen Zuständen werden durch Übergänge ausgedrückt.

- (1.) In „STDE“ fügen Sie einen Zustand durch Klick auf den im Bild markierten Button ein:



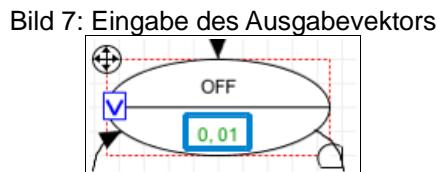
Beachten Sie, dass sich die Darstellung hier auf den Typ „Moore“ bezieht.

- (2.) Der „Einfügemodus“ ist nun aktiv. Wenn Sie mit der Maus über der Zeichenbereich fahren, wird der Zustand auf Höhe der Mausposition gezeigt. Falls die Zielposition nicht sichtbar ist, kann der Zeichenbereich über die Pfeiltasten (auf der Tastatur) verschoben werden.
- (3.) Fügen Sie den Zustand durch Linksklick an die gewünschte Stelle ein.
- (4.) Möchten Sie die Position jetzt (oder später) noch einmal **verschieben**, so ist dies über sogenannte „Anpacker“ möglich:
  - (a) Wechseln Sie -falls nötig- zunächst durch [ESC] oder in den „Selektionsmodus“
  - (b) Wählen Sie den Zustand durch Linksklick aus
  - (c) Ziehen Sie den Zustand mittels dem im Bild markierten „Anpacker“ per Drag'n'Drop an die gewünschte Position:



Tip: Die Zustands-Größe kann mit dem „Anpacker“ unten rechts justiert werden

- (5.) Legen Sie den **Namen** des Zustands fest, indem Sie auf diesen klicken und den Text über die Tastatur eingeben. Mit den Tasten [Tab] und [Enter] wechseln Sie zwischen verschiedenen Textfeldern einer Komponente (beispielsweise zwischen Zustandsbezeichner und Ausgabevektor bei „Moore“).  
Hinweis: Bei Texteingaben muss sich der Mauscursor im Zeichenbereich befinden.  
Das Hinzufügen einer **Beschreibung** ist tabellarisch möglich (Reiter: „Zustände“).
- (6.) Beim Automatentyp „Moore“ ist die Ausgabe an den Zustand gebunden. Ausgaben sind durch den **Ausgabevektor** definiert. Dies ist die Menge der Ausgangssignale. Sie sehen die „Reihenfolge“ der Ausgabesignale im oberen Zeichenbereich z. B: [LED, ENGINE]. Der Ausgabevektor wird (auf die gleiche Weise wie der „Name“ unter (5.)) in das entsprechende Textfeld eingegeben.



Die Syntax für die Ausgabevektoren ist (zunächst unter Vernachlässigung von Variablen) wie folgt definiert:

```
<Ausgabevektor> := <Konstante> { , ' <Konstante> }
```

Es handelt sich also um eine Folge von Konstanten, die durch Kommata getrennt sind. Eine einzelne Konstante ist wie folgt aufgebaut:

<u>Notation</u>	<u>Beschreibung</u>
binär	Beispiele: 0, 1, 01, 110, 1011 (kein Präfix)
hexadezimal	Beispiele: 0x0, 0x01, 0xA2, 0xF3F2 (Präfix: „0x“)
dezimal	Beispiele: #1, #56, #-37 (Achtung: Präfix „#“ erforderlich)

Es sei angemerkt, dass die Wertnotation *unabhängig* vom gewählten Signal-Typ vorgenommen werden kann. Es ist möglich in einem Zustand „binär“ zu codieren; in einem anderen hingegen dezimal. Binärzahlen müssen nicht mit „Nullen gefüllt werden“. Bereichsüberschreitungen werden im Rahmen der Verifikation anhand der Bitlänge erkannt; je nach Zielsprache {C, VHDL} werden nötige Konvertierungen automatisch vorgenommen.

Neben der Ausgabe von Konstanten ist die Ausgabe von Variableninhalten möglich (Variablen werden im Abschnitt „5.3“ behandelt).

**Beachten Sie, dass dies nur möglich ist, wenn unter „Einstellungen“ der Haken „Logik im VHDL-Prozess“ gesetzt ist.**

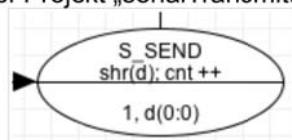
Damit erweitert sich die Syntax:

```
<Ausgabe> ::= <Konstante> | <Variable>
<Ausgabevektor> ::= <Ausgabe> { ' , ' <Ausgabe> }
```

Eine Variablenausgabe wird wie folgt notiert:

<u>Notation</u>	<u>Beschreibung</u>
Variablenname( ObereGrenze : UntereGrenze)	Die „Grenzen“ beschreiben das signifikanteste (oberste) verwendete Bit (MSB), sowie das nicht-signifikanteste (unterste) verwendete Bit (LSB). Das untere Nibble eines Bytes würde z. B. so angesprochen: „meineByteVariable(#3:0)“ Hinweise: In der aktuellen Version von „STDE“ können nur Variablen vom Typ BIT und BIT_N in Ausgaben verwendet werden Beachten Sie, dass die vorgestellte Raute (#) für Dezimalzahlen auch hier verwendet wird. Die Angabe der Grenzen ist <u>nicht</u> optional.

Bild 8: Variablenausgabe „d(0:0)“ aus dem Beispiel-Projekt „serialTransmitter.stde“



(7.) Ein Zustand kann nach Selektion durch den Button gelöscht werden.

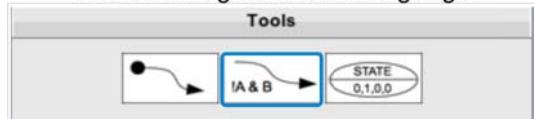
## 5.2.2 Übergänge (Transitions) und Startknoten

Übergänge setzen zwei Zustände miteinander in Relation. Tritt die Übergangsbedingung ein, so wird der Wechsel von Zustand A nach Zustand B vollzogen.

In „STDE“ können Zustandsübergänge wie folgt eingefügt werden:

- (1.) Klicken Sie auf den im Bild markierten Button „Zustand nach Moore einführen“ bzw. „Zustand nach Mealy einführen“ im Bereich „Tools“:

Bild 9: Einfügen eines Übergangs

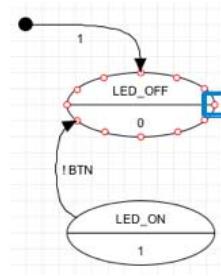


- (2.) Die beiden zugehörigen Zustände werden nun nacheinander ausgewählt:

- (a) Auswahl des „von“-Zustands: Sobald Sie mit dem Cursor über einen Zustand fahren, werden „Andock-Punkte“ in Form von kleinen roten Kreisen dargestellt. Dies sind die Stellen, an denen ein Übergang angeheftet werden kann. Es ist möglich, beliebig viele Übergänge an einen einzelnen „Andock-Punkt“ anzubringen.

Linksklick bestätigt das Setzen des Startpunkts.

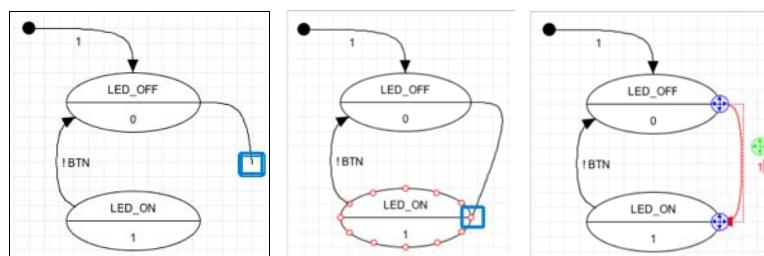
Bild 10: Die Markierung zeigt einen „Andock-Punkt“ im „von“-Zustand



- (b) Auswahl des „zu“-Zustands: Bewegen Sie die Maus auf einen der „Andock-Punkte“ des Zielzustands und bestätigen Sie auch hier wieder mit einem Linksklick.

Hinweis: Der Vorgang kann mit [ESC] unterbrochen werden.

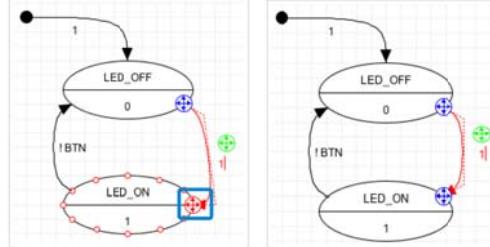
Bild 11: Links: das Mitführen der Kurve bei Mausbewegung, Mitte: „Andocken“ an den Zielzustand, Rechts: Ergebnis



- (3.) [Optional] Falls Sie mit der Auswahl der „Andock-punkte“ unzufrieden sind, oder gar eines der beiden „Enden“ an einen anderen Zustand verschieben wollen, ist dies wie folgt möglich: Per Drag'n'Drop können Sie die im Bild 9 dargestellten „Anpacker“ verschieben.

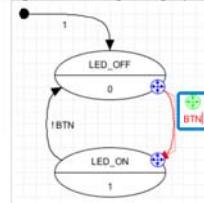
Hinweis: Das Verschieben auf einen anderen Zustand wird erst sichtbar, wenn sich der „neue“ Zustand unmittelbar unter der Maus befindet.

Bild 12: Links: „Andock-punkt“ des „zu“-Zustands, Rechts: Resultat



- (4.) Die **Übergangsbedingung** ist ein Ausdruck, der **Eingangssignale**, Konstanten, Vergleiche, Relationen und arithmetische Operatoren enthalten darf. Auch die Verwendung von Variablen ist zugelassen (näheres dazu in Abschnitt „5.3“). Die Erfassung erfolgt textuell; die Syntax ist in der nachfolgenden Tabelle definiert. Es findet eine Typ-Überprüfung statt (siehe Abschnitt „6. Verifikation, ...“). Die Position der Bedingung innerhalb es Zeichenbereiches kann mit dem grünen „Anpacker“ verändert werden. Ein Spontanübergang besteht aus einer eins („1“). Hinweis: Die Syntax ist unabhängig von der Zielsprache (C / VHDL)!

Bild 13: geänderte Übergangsbedingung („BTN“ ist hier vom Typ „BIT“)

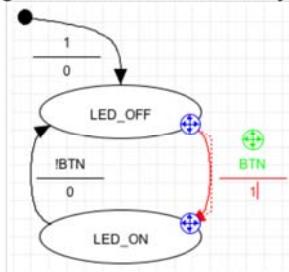


<u>Operation</u>	<u>Syntax</u>	<u>Beispiele (bitte ohne Anführungszeichen eingeben) (*1)</u>
logisches UND (AND)	<Ausdruck1> && <Ausdruck2>	Bsp.: „a && b“, „a && b && c“, ...
logisches ODER (OR)	<Ausdruck1>    <Ausdruck2>	Bsp.: „a    b“, „a    b    c    d“, ...
logisches NICHT (NOT)	! <Ausdruck1>	Bsp.: „!a“
Vergleich	<Ausdruck1> OP <Ausdruck2>	Bsp.: „a == b“, „a == b == c“, „a <= b“, ...
	OP := '=='   '!=   '<'   '>'   '<='   '>='	
Klammerung	(<Ausdruck>)	Bsp.: „a==b && (c==d    c==b)“, „!(a==b)“
Konstanten	(Siehe Beschreibungen in „5.2.1“)	Bsp.: „a==0x0A“, „a==1010“, „a==#10“

(\*1) a, b, c, d stehen hier für Signale bzw. Variablen

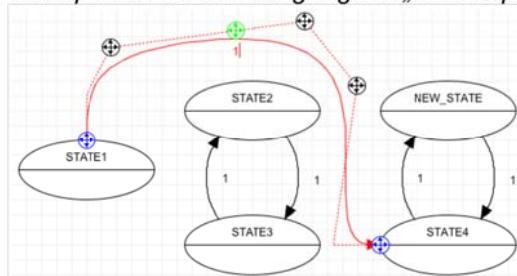
- (5.) Bei einem „Mealy“-Automaten ist die Ausgabe an den Übergang gebunden. Sie können von der Übergangsbedingung per [Tab] oder [Enter] in den Ausgabevektor wechseln – oder das Textfeld per Maus selektieren. Die Ausgabevektor-Syntax gleicht der eines Moore-Automaten; wir verweisen auf den Abschnitt „5.2.1“.

Bild 14: Ausgabevektor eines „Mealy“-Automaten



- (6.) Je nach erstelltem Automat kann es bei den Übergängen zu graphischen Überschneidungen kommen. Sie können einen Übergang mit nichttrivialem Verlauf (siehe Bild 15) einfügen, indem Sie „Kontrollpunkte“ definieren. Diese werden zwischen der Auswahl des „von“- und des „zu“-Zustands durch Klick in den leeren Zeichenbereich eingefügt. Ein nachträgliches Verschieben ist durch Drag and Drop der „Anpacker“ gegeben (nur bei Selektion sichtbar).

Bild 15: Beispiel für einen Übergang mit „Kontrollpunkten“

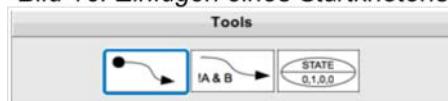


Ein Kontrollpunkt kann durch Klick auf hinzugefügt bzw. durch Klick auf gelöscht werden (Menüleiste).

- (7.) Ein Übergang kann nach Selektion durch den Button gelöscht werden.
- (8.) Für einen Automaten muss ein Startzustand definiert sein. Hierzu wird ein **Startknoten** eingefügt. Dies ist ein Übergang, der keinen „von“-, sondern nur einen „zu“-Zustand hat. Die Übergangsbedingung kann nicht geändert werden und ist immer 1 (Spontanübergang). Beim Typ „Mealy“ kann der Ausgabevektor gesetzt werden.

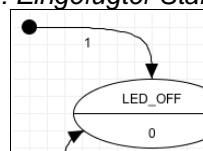
- (a) Wählen Sie das Tool „Startknoten einfügen“:

Bild 16: Einfügen eines Startknotens



- (b) Platzieren Sie den Knotenpunkt durch Linksklick. Wählen Sie dann entweder direkt einen Startzustand aus, oder fügen Sie wie im Punkt (6.) beschrieben „Kontrollpunkte“ ein.

Bild 17: Eingefügter Startknoten



### 5.3 Einfügen und Bearbeiten von Variablen

**Beachten Sie, dass die Verwendung von Variablen nur möglich ist, wenn unter „Einstellungen“ der Haken „Logik im VHDL-Prozess“ gesetzt ist.**

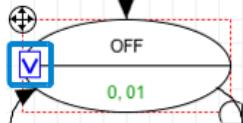
Variablen dienen dem repetitiven (wiederholten) Traversieren („Durchfahren“) von Zuständen. Sie agieren als Kontrollfluss und können beispielsweise als Zustands-Zähler eingesetzt werden. In beschränktem Maße ist auch Rechnen mit ihnen möglich. Eine Variable kann einen passenden Eingangswert erhalten und auch über Ausgangssignale ausgegeben werden.

Variablen weisen die gleichen Eigenschaften wie Signale auf. Ihnen fehlt jedoch eine **Richtung**. Aufgrund der Äquivalenz zu den Signalen hinsichtlich des Einfügens und Löschens verweisen wir auf den Abschnitt „5.1 Einfügen und Bearbeiten von Signalen“.

Variablenzuweisungen werden in den **Zuständen** des Automaten vorgenommen. Die Vorgehensweise ist wie folgt:

- (1.) Auswahl eines Zustands (wie unter 5.2.1 beschrieben)
- (2.) Aktivieren Sie die Variablen-Zuweisungen für diesen Zustand, indem Sie auf das „V“-Symbol klicken. Die Deaktivierung funktioniert analog.

Bild 18: Aktivierung der Variablen-Zuweisungen

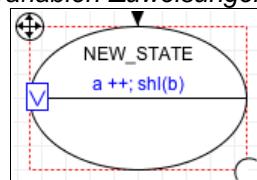


- (3.) Es erscheint ein weiters Textfeld, indem die Zuweisungen eingegeben werden können. Die folgende Operationen stehen Ihnen zur Verfügung:

Operation	Syntax	Beschreibung
Inkrement / Dekrement	<VAR> "++" <VAR> "--"	Erhöht / erniedrigt den Wert der Variablen um 1 Bsp.: „cnt ++“ oder „cnt --“
Shiften	"shl" " ( " <VAR> " ) " "shr" " ( " <VAR> " ) "	Shiftet den Wert einer Variablen nach links (shl) bzw. rechts (shr). Bsp.: „shl(myvar)“
Zuweisen einer Konstante	<VAR> "=" <NUMBER>	Weist einer Variablen eine Konstante zu. (Wertnotation so wie in Kap. 5.2.1) Bsp.: „myvar = #3“
Zuweisen e. Signals (IN)	<VAR> "=" <SIGNAL>	Weist einer Variablen ein Eingangssignal zu. Bsp.: „myvar = inputsignal_a“
Addition / Subtraktion	<VAR> "=" <VAR> ( "+"   "-" ) <NUMBER>	Variable1 = Variable2 <u>±</u> Konstante Bsp.: „myvar1 = myvar2 + 1337“

Es ist möglich mehr als eine Variablenzuweisung in einem Zustand vorzunehmen. Die Trennung folgt per Semikolon:

Bild 19: Multiple Variablen-Zuweisungen in einem Zustand



Das Beispiel-Projekt „serial\_transmitter.stde“ enthält zwei Variablen und verdeutlicht das Konzept.

## 6 Verifikation, Export (SCXML), Code-Generierung (C / VHDL)

Neben der visuellen Darstellung ist es insbesondere möglich zum Automaten zugehörigen Code zu erzeugen. Dies ist mit den Zielsprachen „C“ und „VHDL“ möglich. Der generierte Code hat den Charakter einer „Schablone“, d. h. zur Einsetzung in der Praxis kann es (vor allem in „C“) erforderlich sein weiteren Code manuell einzufügen.

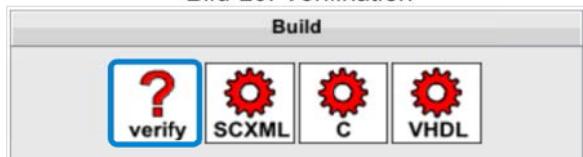
Es ist (a) möglich einen Export im Dateiformat SCXML als „Zwischenformat“ zu erzeugen oder (b) eine direkte Codeausgabe in die Zielsprachen {C, VHDL} vorzunehmen.

Von der „Design-Phase“ (hier: die Erstellung eines Automaten) wird eine Unabhängigkeit von der Zielsprache gefordert. Mit Hilfe der „neutralen“ Sprache in den Übergangsbedingungen (siehe „5.2.2“) wird dies gewährleistet.

Vor der Code-Generierung kann eine manuelle Verifikation vorgenommen werden (Dies spart Zeit, da für die reine Verifikation kein Speicherort bestimmt werden muss).

(1.) Starten Sie die Verifikation mit dem Button „verify“:

Bild 20: Verifikation



(2.) Aufgetretene Fehler werden im Fenster „Log“ dargestellt (Über den Button „clear-log“ kann die Auflistung jederzeit wieder gelöscht werden).

Im Erfolgsfall erscheint die Meldung: „VERIFIKATION ERFOLGREICH!“.

Falls Fehler vorhanden sind, werden diese – wenn möglich – im Zeichenbereich hervorgehoben (rot unterstrichen).

Die folgenden Validitätsprüfungen werden vorgenommen:

Prüfung	Beschreibung
Projektname korrekt?	Muss mit einem Buchstaben beginnen, darf keine Sonderzeichen enthalten (Grund: Verwendung Bezeichner Name der „Entity“ in VHDL)
Startzustand vorhanden?	Ein Startknoten muss vorhanden sein
Zustandsnamen korrekt?	Siehe Punkt: „Projektname“
Signalnamen korrekt?	Siehe Punkt: „Projektname“
Variablennamen korrekt?	Siehe Punkt: „Projektname“
Sind alle Identifier (Bezeichner) disjunkt?	Es darf kein Paar aus der Menge { Signale, Variablen, Zustände } geben, die den gleichen Namen tragen
Übergangsbedingung korrekt?	Überprüfung von Syntax und teilw. Semantik (Typen) Vgl. EBNF im Anhang A
Ausgabevektor korrekt?	Überprüfung von Syntax und Wertebereichen Vgl. EBNF im Anhang A
Variablenzuweisungen korrekt?	Überprüfung von Syntax und teilw. Semantik (Typen) Vgl. EBNF im Anhang A
Bezeichner „Reset“ oder „Clk“ verwendet?	Die VHDL Code-Generierung verwendet diese Signalnamen

- (3.) SCXML-Code wird über den Button  erzeugt. Wählen Sie den Speicherort. Eine Anpassung des Standardpfads ist wie in „4.4“ beschrieben möglich.

Typen werden wie folgt umgesetzt:

<u>STDE</u>	<u>Bitlänge</u>	<u>SCXML</u>
BIT	1	bit
BIT_N UNSIGNED	4	nibble
	8	byte
	sonst	vector mit Angabe von „size“ = Bitlänge
SIGNED	sonst	integer mit Angabe von „size“ = Bitlänge

- (4.) C-Code wird über den Button  erzeugt. Wählen Sie den Speicherort. Eine Anpassung des Standardpfads ist wie in „4.4“ beschrieben möglich.  
Ab Version 1.3 wird nicht mehr eine Main-Automatenschleife erzeugt, sondern:
- Eine Header-Datei <Modellname>.h (<Modellname> = STDE-Modellname)  
Sie enthält Vereinbarungen für die verwendeten Ein- und Ausgaben, die in Vektoren zusammengefasst sind, sowie die Prototypen einer Schritt- und einer Simulationsfunktion für den Automaten
  - Eine Schrittfunctionsdatei <Modellname>.c  
Sie enthält die Schrittfunction  

```
bool fsm_<Modellname>
    (bool reset,
     InVector_<Modellname> *inv,
     outVector_<Modellname> *outv)
```

 die entweder den Automaten zurücksetzt (`reset == true`) oder aus den aktuellen Eingaben (`inv`) den Folgezustand und die Ausgaben (`outv`) berechnet. (Die Zustandsvariable ist lokal in der Funktion vereinbart)
  - Eine Testfunktionsdatei <Modellname>\_exec.c  
Sie enthält Beispielcode zur Simulation des Automaten in c in einer Funktion  
`void exec_fsm_<Modellname>()`

Typen werden wie folgt umgesetzt:

<u>STDE</u>	<u>Bitlänge</u>	<u>C (benutzt werden MISRA-kompatible Typdefinitionen)</u>
BIT	1	bool
BIT_N UNSIGNED	1	bool
	2 – 8	uint8_t
	9 – 16	uint16_t
	17 – 32	uint32_t
	33 – 64	uint64_t
SIGNED	1	bool
	2 – 8	int8_t
	9 – 16	int16_t
	17 – 32	int32_t
	33 – 64	int64_t

- (5.) VHDL-Code wird über den Button  erzeugt. Wählen Sie den Speicherort. Eine Anpassung des Standardpfads ist wie in „4.4“ beschrieben möglich.

Typen werden wie folgt umgesetzt:

<u>STDE</u>	<u>Bitlänge</u>	<u>VHDL</u>
BIT	1	std_logic
BIT_N	1	std_logic
	N	std_logic_vector(N-1 downto 0)
SIGNED	N	signed(N-1 downto 0)
UNSIGNED	N	unsigned(N-1 downto 0)

Für eine korrekte Code-Generierung wird empfohlen, dass innerhalb von Übergangsbedingungen Vergleiche ('==', '!=', ...) zwischen Signalen (bzw. Variablen) und Konstanten immer in folgender Reihenfolge verwendet werden:

<Signal/Variable> == <Konstante>

Bsp.: „a == #5“ und nicht: „#5 == a“

Andernfalls kann die Umsetzung in VHDL fehlerhaft sein (VHDL ist streng typisiert; Konstanten in „STDE“ können hingegen beliebig codiert werden: Die implementierte Umsetzung verlangt obige Reihenfolge).

## 7 Sonstiges

### 7.1 Mehrfachselektion

Mehrfachselektion von Zuständen und Übergängen ist möglich durch:

- (a) Sukzessives Selektieren bei gedrückt gehaltener [Shift]-Taste
- (b) „Ziehen eines Rahmens“ um die Objekte (bei gedrückt gehaltener linker Maustaste)
- (c) den Button  : „Alle Komponenten selektieren“

Die Modi (a) und (b) verlangen, dass der „Selektionsmodus“ () aktiv ist.

### 7.2 Export als Bilddatei

Der Zeichenbereich kann als Bilddatei (Format: png – „portable network graphics“) exportiert werden. Der Button  öffnet ein entsprechendes Speichermenü. Es wird zunächst in das Verzeichnis gewechselt, das unter „4.4 Das Dialogfeld ‘Einstellungen‘“ als Projektverzeichnis definiert wurde.

## 8 Kontakt

Für Kritik, Vorschläge, Bug-Meldungen etc. sind wir sehr dankbar. Sie können uns wie folgt erreichen:

[georg.hartung@th-koeln.de](mailto:georg.hartung@th-koeln.de)

## 9 Revisionshistorie

- 0.2: 02.12.2011: Jan Montag, Andreas Schwenk: Alpha
- 1.0: 18.01.2012: Jan Montag, Andreas Schwenk: Final
- 1.1: 22.10.2012: Andreas Schwenk:  
Generierung von VHDL-Code gem. der Veranstaltung GTI ab dem WS  
2012 / 2013 (FH Köln). Verbesserung der graphischen  
Transitionsbearbeitung.
- 1.2: 13.02.2015: Georg Hartung  
Erweiterung der Syntax von Variablen-Zuweisungen und Ausgabe-  
Ausdrücken, dadurch einfache Berechnungen usw. ermöglicht
- 1.3: 6.6.2016: Georg Hartung  
Codeerzeugung für C umgestellt: Erzeugung der Schrittfunktion etc.

## Anhang A: Syntaxregeln nach EBNF („Erweiterte Backus-Naur Form“)

### Allgemein:

```

<BIT>      ::= 0 | 1
<DEC>      ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<HEX>      ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F
<LETTER>   ::= A | B | ... | Z | _
<NUMBER>   ::= BIT { BIT }
              | "0x" { HEX }
              | "#" [ "-" ] { DEC }
<SIGNAL>   ::= <LETTER> { <NUMBER> | <LETTER> }

```

### Übergangsbedingungen: (Prüfung durch den „ConditionParser“)

```

<EXP>       ::= <AND> { " | " <AND> }
<AND>      ::= <COMPARE> { "&&" <COMPARE> }
<COMPARE>  ::= <UNARY> { "==" <UNARY> } | <UNARY> { "<>" <UNARY> }
              | <UNARY> { "<=" <UNARY> } | <UNARY> { ">=" <UNARY> }
              | <UNARY> { "<" <UNARY> } | <UNARY> { ">" <UNARY> }
<UNARY>   ::= [ "!" ] "(" <EXP> ")" | [ "!" ] <SIGNAL>
              | [ "!" ] <VARIABLE> | <NUMBER>

```

### Variablenzuweisungen: (Prüfung durch den „VariableAssignmentParser“)

```

<ASSIGNLIST> ::= <ASSIGNMENT> { ";" <ASSIGNMENT> }
<ASSIGNMENT> ::= "shl" "(" <VARIABLE> ")"
                  | "shr" "(" <VARIABLE> ")"
                  | <VARIABLE> "=" <VARIABLE> ( "+" | "-" ) <NUMBER>
                  | <VARIABLE> "=" <VARIABLE> ( "+" | "-" ) <VARIABLE>
                  | <VARIABLE> "=" <NUMBER>
                  | <VARIABLE> "=" <SIGNAL>
                  | <VARIABLE> "++"
                  | <VARIABLE> "--"

```

### Ausgabevektoren: (Prüfung durch den „OutputVectorParser“)

```

<OUTPUT>     ::= <NUMBER>
                  | <VARIABLE> [ "(" <NUMBER> [ , <NUMBER> ] ")" ]
                  | <SIGNAL>    [ "(" <NUMBER> [ , <NUMBER> ] ")" ]
<OUTPUT_VEC> ::= <OUTPUT> { " , " <OUTPUT> }

```