Prof. Dr. Christoph Scholl
M. Sc. Tobias Seufert
Tobias Faller, Timo Fritsch, Stefan Wittemer

Freiburg, 10. Mai 2020

Praktikum Embedded Systems Engineering

Teil 1 Einführung[1]

Stateflow ist ein Tool, welches zur Matlab/Simulink Familie[2] gehört. Es erweitert Simulinks rudimentäre Fähigkeiten zur Modellierung von diskreten Zustandsmaschinen. Hierzu wird eine Variante der Statecharts, wie sie von Harel[3] definiert wurden, verwendet. Ein Stateflow-Modell besteht grundsätzlich aus syntaktisch angereicherten Zustandsdiagrammen, die Event-getriggerte Prozesse modellieren. Verglichen mit einfachen Zustandsdiagrammen wurden diese um Hierarchie (Zustände können wiederum eingebettete Zustandsdiagramme enthalten), Parallelismus und symbolischen Variablen, wie man sie von Programmiersprachen kennt, erweitert. Sie sind damit mit den Zustandsdiagrammen, wie man sie von UML[4] oder dem Tool IBM Rational Statemate[5] kennt, verwandt. Sollten Sie mit dem Konzept der Statecharts noch nicht vertraut sein, dann sehen Sie sich unbedingt die Materialien hierzu auf der Homepage der Veranstaltung durch.

In Stateflow gibt es jedoch einige semantische Unterschiede. So gibt es beispielsweise keine wirkliche parallele Ausführung von Zuständen. Parallele Zustände (AND-States) haben implizit eine Ordnung, welche durch eine kleine Zahl in der grafischen Darstellung angezeigt wird. Die Simulation arbeitet die Zustände entsprechend dieser Ordnung ab. Dies ist bei Write-Write-/Write-Read-Abhängigkeiten zu berücksichtigen. Zusätzlich gibt es bei Transitionen keinen Nicht-Determinismus. Auch hier wird wieder implizit eine Ordnung auf den Transitionen vergeben. Da Transitionen, welche gleichzeitig aktiv sein können, selten erwünscht sind und deshalb meistens einen Hinweis auf eine fehlerhafte Modellierung darstellen, kann man sich dies von Stateflow als Warnung oder Fehler anzeigen lassen. Es ist i.d.R. schlechter Stil, wenn das (korrekte) Verhalten eines Modells auf diesen impliziten Ordnungen beruht, da schon kleine Änderungen (z.B. Verschieben eines Zustands) die Ordnung ändern können. Deshalb sollte in diesen Fällen die Ordnung auf den Zuständen bzw. Transitionen explizit vergeben und durch Kommentare erläutern werden. Ein weiterer Unterschied zu den Statecharts nach Harel betrifft die Behandlung von Events (siehe hierzu Abschnitt 1.2).

[1] Wir bedanken uns bei Prof. Dr. Martin Fränzle, Universität Oldenburg, für diverse Materialien aus seinen Lehrveranstaltungen, die an vielen Stellen die Grundlagen für die Konzeption dieses Praktikums lieferten.

[2] `http://www.mathworks.com`

[3] D. Harel and M. Politi, *Modeling Reactive Systems with Statecharts*, McGraw-Hill, 1998

[4] `http://www.uml.org/`

[5] `http://www.ibm.com/software/awdtools/statemate/`

Die Stärke von Stateflow folgt aus der nahtlosen Integration in Simulink (und Matlab). Dies ermöglicht die Kombination von komplexen zeitkontinuierlichen Modellen (z.B. physikalisches Modell einer Umgebung) und zeitdiskreten Modellen (z.B. getaktete Steuerlogik eines Mikrocontroller) und deren gemeinsame Simulation. Die Kombination Simulink/Stateflow geht dann weit über die Beschreibungsmöglichkeiten von Statecharts nach Harel hinaus. Im Folgenden wird der Schwerpunkt allerdings auf dem Entwurf von Statecharts mit Stateflow liegen.

Hinweis:

Grüne Boxen kennzeichnen Hinweise/Anmerkungen mit wichtigen Hintergrundinformationen.
Blaue Boxen enthalten Aufgaben, die Sie bearbeiten müssen.
Die roten Boxen markieren so genannte *Checkpoints*. Wenn Sie eine solche Stelle erreichen, sollen Sie den aktuellen Stand mit einem Betreuer durchsprechen, bevor Sie weitermachen. Dies ist typischerweise immer am Ende eines Praktikumteils der Fall.

Schriftliche Ausarbeitung:

Im Laufe der Einführung werden Sie **blaue Aufgabenboxen** entdecken. Diese sollen während Ihrer Arbeit mit *Matlab* schriftlich ausgearbeitet werden. Bevor Sie mit dem zweiten Teil fortfahren können müssen Sie sowohl Ihre Ausarbeitungen als auch Ihre Implementierung einem Tutor vorführen. Diese Ausarbeitung wird **nicht** eingesammelt.

Teil 1
Einführung in Stateflow

1 Eine einfache Stoppuhr

Der erste Teil soll Sie in Form eines kleinen Tutoriums mit der grundlegenden Benutzung von Stateflow bekannt machen.

Bitte beachten Sie:

Auch wenn wir Sie zunächst mit einer Schritt-für-Schritt-Anleitung an Simulink/Stateflow heranführen, so ist es durchaus gewünscht und später auch unbedingt notwendig, dass Sie die Möglichkeiten von Simulink und Stateflow selbständig erkunden. Es ist deshalb sehr empfehlenswert, sich parallel zu diesem Tutorium mit der umfangreichen Hilfefunktion vertraut zu machen. So sollte man sich z.B. immer die Dokumentation der jeweils verwendeten Elemente ansehen.

Aufgabe:

Erstellen Sie mit Hilfe der nachfolgenden Anleitung eine einfache Stoppuhr und simulieren Sie diese.

1.1 Start von Stateflow

Starten Sie nun die Matlab Anwendung (*Windows Icon* → *Alle Programme* → *MATLAB* → *R2020a* → *Matlab R2020a*[6]). Es sollte sich ein Fenster ähnlich zu Abb. 1 öffnen.

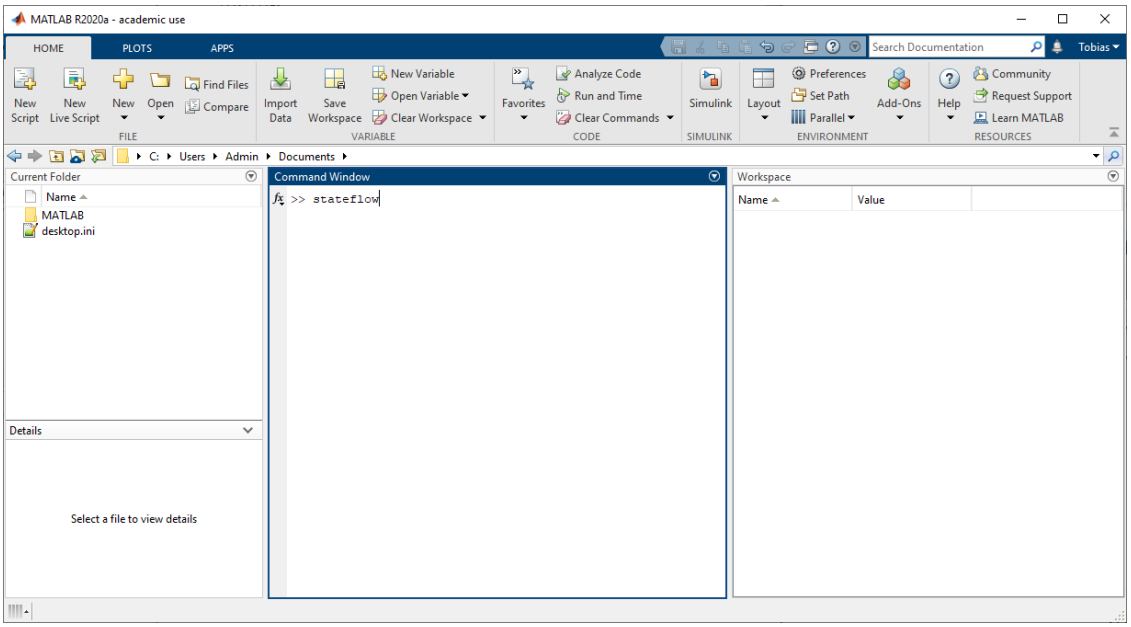


Abbildung 1: Matlab Startfenster

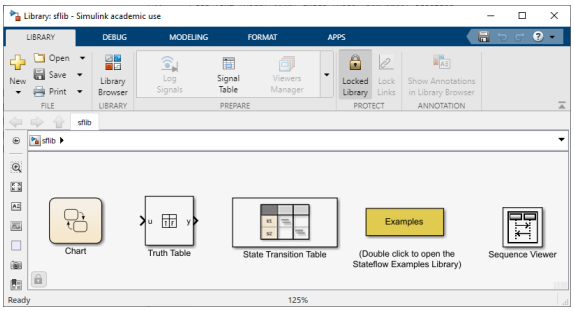
Geben Sie nun in die Matlab-Kommandozeile folgendes Kommando ein:

```
>> stateflow ↵
```

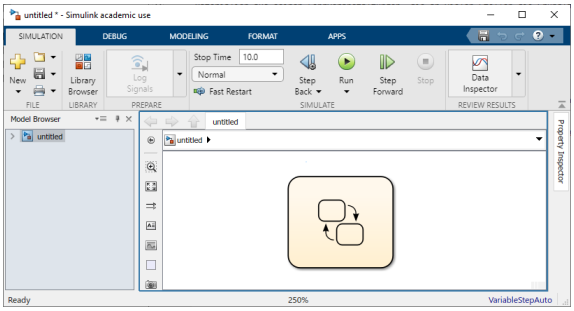
Dadurch öffnen sich zwei Fenster. Eines ist die Stateflow-Block-Library (Fenstername `sflib`), welches die folgenden Elemente – so genannte Blöcke – enthält: *Chart*, *Truth-Table*, *State Transition Tables*, *Examples* und *Message Viewer* (siehe auch Abb. 2(a)). Wenn Sie mit einem Rechtsklick das Kontextmenü eines der Blöcke öffnen, dann können Sie durch Auswahl von *Help* die zugehörige Hilfe aufrufen.

Das andere Fenster mit dem Namen `untitled*` ist ein neues Simulink-Modell, welches aus einem leeren Stateflow-Block besteht (Abb. 2(b)). Das Zeichen `*` im Namen des Fensters zeigt an, dass dieses Modell noch nicht gespeichert wurde. Sollte man weitere Stateflow-Blöcke benötigen, kann man diese durch Drag & Drop aus dem Bibliotheks-Fenster (`sflib`) hinzufügen. Zusätzlich könnte man auf diese Weise auch andere Simulink-Blöcke hinzufügen, um z.B. ein Modell der Umgebung zu bauen. Für unsere Stoppuhr reicht ein Stateflow-Block mit einer recht einfachen Simulink-Umgebung aus, diese wird allerdings erst später in diesem Tutorium erzeugt.

[6] R2020a bezeichnet die Matlab-Version (Jahr 2020, Frühjahrausgabe). Eine Anleitung zur Installation der aktuellen Matlabversion finden sie auf der Fachschaftsseite (https://fachschaft.tf.uni-freiburg.de/informationen/software/matlab-1). Bitte verwenden sie für ihren Heimrechner die selbe Version wie die Praktikumsraumrechner (R2020a).



(a) Stateflow Bibliothek



(b) Neues Simulink Fenster mit einem Stateflow-Block

Abbildung 2: Stateflow Start

Ändern Sie nun den Namen des Stateflow-Blocks, indem Sie auf den Block klicken und den Text `Chart` durch `StopWatch` ersetzen. Speichern Sie nun das Simulink-Modell (*File* → *Save as...*) und geben Sie dem Modell einen sinnvollen Namen. Es ist empfehlenswert, dass Sie für jedes Modell einen eigenen Unterordner anlegen. Speichern Sie alle Modelle generell mit der Dateiendung `.mdl`. Nach dem Abspeichern sollte Ihr Fenster in etwa wie Abb. 3 aussehen.

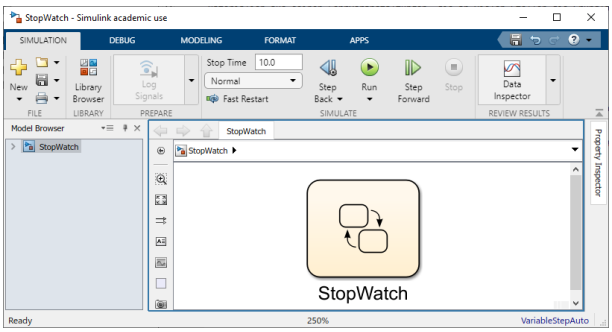


Abbildung 3: Das Fenster nach dem Abspeichern

1.2 Erzeugen eines Zustandsdiagramms

Im Moment ist der Stateflow-Block noch leer. Um ein entsprechendes Zustandsdiagramm zu erzeugen, müssen Sie den Stateflow-Editor durch Doppelklick auf den Stateflow-Block öffnen (Abb. 4).

Auf der linken Seite befinden sich einige quadratische Knöpfe. Wenn man den Mauszeiger über einen dieser Knöpfe bewegt, erscheint unten in der Statusleiste eine kurze Funktionsbeschreibung.

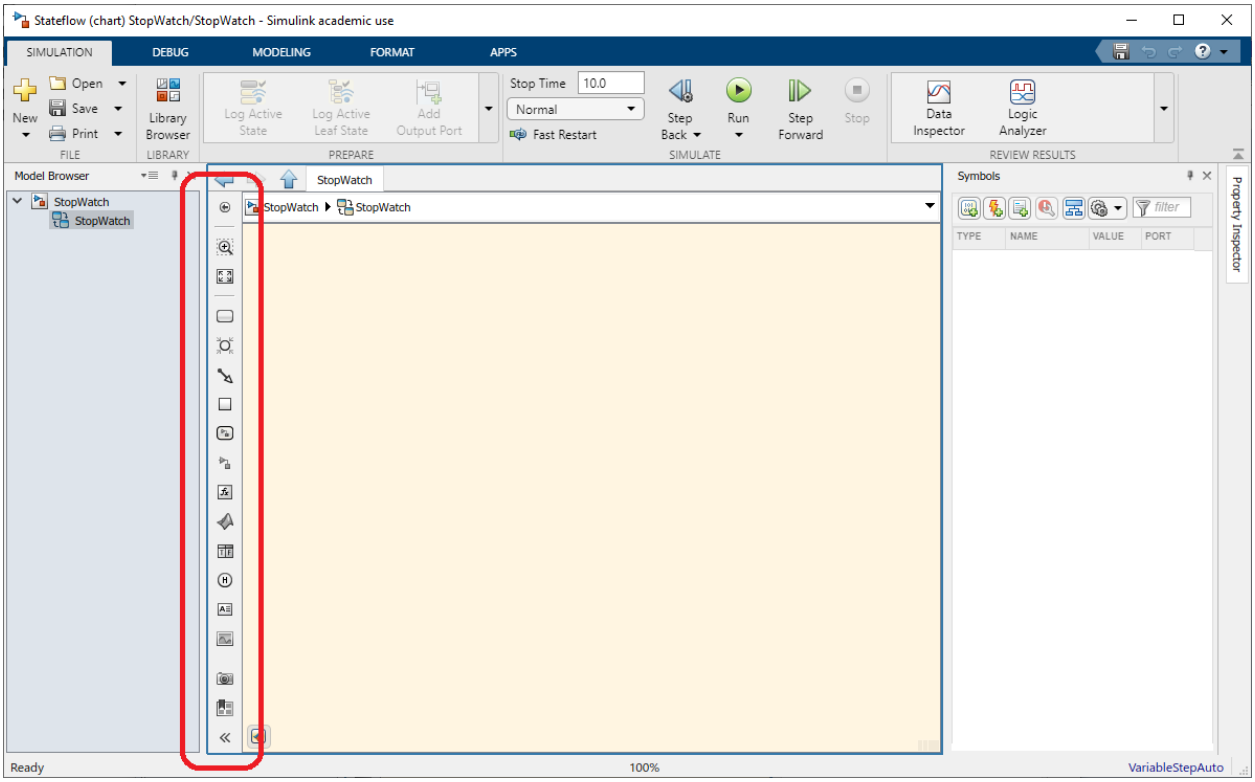


Abbildung 4: Der leere Stateflow-Editor

Mit dem Knopf *State* aktiviert man das Zustandswerkzeug. Aktivieren Sie das Zustandswerkzeug durch einen Linksklick und platzieren Sie mit der Maus einen neuen Zustand auf der Arbeitsfläche (erneuter Linksklick). Nach dem Platzieren blinkt ein Cursor innerhalb des neuen Zustands. Sie sollten nun dem Zustand einen Namen geben (`Stopped`). Den Namen eines Zustands können Sie jederzeit wieder ändern. Die Größe des Zustands ändern Sie durch Ziehen mit der Maus an den Ecken. Es ist natürlich auch möglich, einen Zustand zu verschieben. Klicken Sie dafür mit der Maus in den Zustand und halten Sie die linke Maustaste gedrückt, bis die gewünschte Position erreicht ist.

Erzeugen Sie nun noch einen zweiten Zustand mit dem Namen `Started`.

Wenn Sie sich mit dem Mauszeiger auf dem Rand eines Zustands befinden, verändert sich der Mauszeiger zu einem Kreuz. Drücken Sie nun die linke Maustaste und ziehen sie den Mauszeiger auf den Rand des anderen Zustands und lassen Sie die Maustaste los. Sie haben soeben eine Transition (Übergangskante) erzeugt.

Jede Transition kann mit einer Bedingung und Aktionen beschriftet werden. Klicken Sie hierfür auf die Transition. Es erscheint ein Fragezeichen. Klicken Sie auf dieses und geben Sie ein sogenanntes Label ein. Die Syntax ist wie folgt:

```
event [condition] / action
```

Alle drei Teile sind dabei optional.[7] Das `event` wird nur benötigt, wenn die Transition durch ein bestimmtes Event ausgelöst (getriggert) werden soll. Die `condition` benötigt man nur, wenn die Transition an eine Bedingung geknüpft ist und die `action` wird nur benutzt, wenn eine Aktion (z.B. Variablenzuweisung) ausgeführt werden soll. Die Aktion wird nur ausgeführt, wenn sowohl das Event als auch die Bedingung zutreffen. In diesem Fall wird auch die Transition ausgeführt und man verlässt den Quellen-Zustand und aktiviert den Ziel-Zustand. Weitere Informationen finden Sie in der Hilfe unter dem Stichwort *Transition Label*.

Hinweis:

An dieser Stelle gibt es einen wesentlichen Unterschied zwischen den Statecharts nach Harel und den Statecharts, wie sie in Stateflow verwendet werden. Während in Statecharts nach Harel beliebige Boolesche Ausdrücke mit Events erlaubt sind, können hier nur Disjunktionen von Events ($event_1|\ldots|event_n$) spezifiziert werden. Fehlt bei einer Transition die (explizite) Angabe eines Events, dann wird implizit eine Disjunktion über alle existierenden Events des Statecharts angenommen. Die Semantik der Stateflow-Statecharts entspricht in der Standardeinstellung im Prinzip dem synchronen Zeitmodell der Statecharts nach Harel. Es ist auch möglich, einem Stateflow-Statechart eine Super-Step-Semantik zuzuweisen (asynchrones Zeitmodell bei Harel). Allerdings ist zu beachten, dass sich durch die erwähnte implizite Ergänzung von Disjunktionen über Events ein Unterschied ergibt.
Ein weiterer Unterschied zu Statecharts sind so genannte *lokale Events*. Diese sind nicht global sichtbar, sondern nur in einem Teil der Zustandshierarchie.

Der initiale Zustand wird markiert durch eine *Default Transition*. Dies ist eine Transition ohne Quellen-Zustand. Benutzen Sie den Knopf *Default Transition*, um das entsprechende Werkzeug zu aktivieren.

Erstellen Sie nun mit den bisher beschriebenen Werkzeugen eine einfaches Zustandsdiagramm einer Stoppuhr, wie es in Abb. 5 gezeigt wird. Beachten Sie, dass `Started` ein weiteres Zustandsdiagramm enthält (Hierarchie). Machen Sie sich klar, was dies bedeutet.

Hinweis:

- Jede Variablenänderung durch eine Action wird standardmäßig auf der Matlab-Konsole ausgegeben. Das dies die Simulationsgeschwindigkeit erheblich beeinträchtigen kann, wird empfohlen Befehle durch ein Semikolon abzuschließen. Bsp: $i = 3;$
- Es ist manchmal nötig, Variablen innerhalb des Statecharts zwischen Typen zu konvertieren. Bsp: $uint8(x)$ konvertiert x in einen uint8.

[7] Es gibt noch die Möglichkeit, eine Aktion direkt bei der Bedingung anzugeben. Dies wird hier nicht besprochen und kann in der Hilfe nachgeschlagen werden. Achtung: Semantische Unterschiede beachten!

Aufgabe:

Machen Sie sich mit der Dokumentation zum Thema *Transitionen* vertraut. Fassen Sie schriftlich mit eigenen Worten zusammen wie das Format/Notation der Transitions-Labels in Stateflow definiert ist.

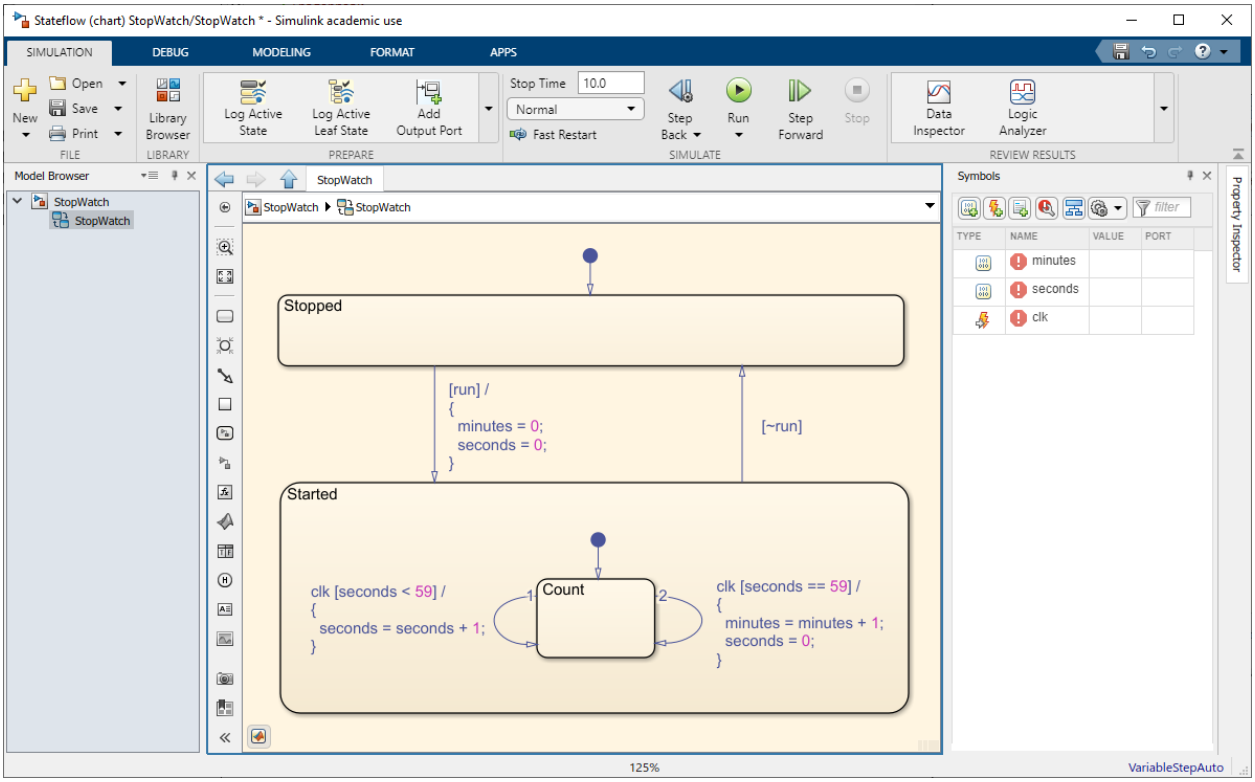


Abbildung 5: Eine einfaches Stateflow-Modell einer Stoppuhr

1.3 Definition der Eingänge und Ausgänge

Damit dieses Zustandsdiagramm mit einer (Simulink-)Umgebung kommunizieren kann, müssen Eingänge und Ausgänge definiert werden. In diesem Beispiel haben Sie bisher die global sichtbaren Datenelemente (data items) *clk*, *run*, *minutes* und *seconds* benutzt, aber noch nicht definiert.[8]

Sie müssen ein Eingangsevent (*clk*), einen Eingang von Simulink (*run*) und zwei Ausgänge zu Simulink (*minutes* und *seconds*) definieren. Öffnen Sie hierzu das Menü `Modeling` → `Model Explorer` und benutzen Sie die in Abb. 6 gezeigten Knöpfe um entsprechende Ein-/Ausgänge zu erzeugen. Verwenden Sie die gleichen Namen wie im Diagramm.

In der nachfolgenden Tabelle sind alle Datenelemente noch einmal aufgelistet. Beachten Sie auch den Typ und den Trigger bzw. den Wertebereich. Die einzelnen Einstellungen sind in der Abb. 7 dargestellt. Sie können mit dem `Model Explorer` alle Elemente nochmals überprüfen und ggf. korrigieren.

Name	Kategorie	Scope	Typ	Sonstiges
clk	Event	Input		Trigger: Rising
run	Data	Input	boolean	
minutes	Data	Output	uint32	
seconds	Data	Output	uint8	Range: $0, \ldots, 59$

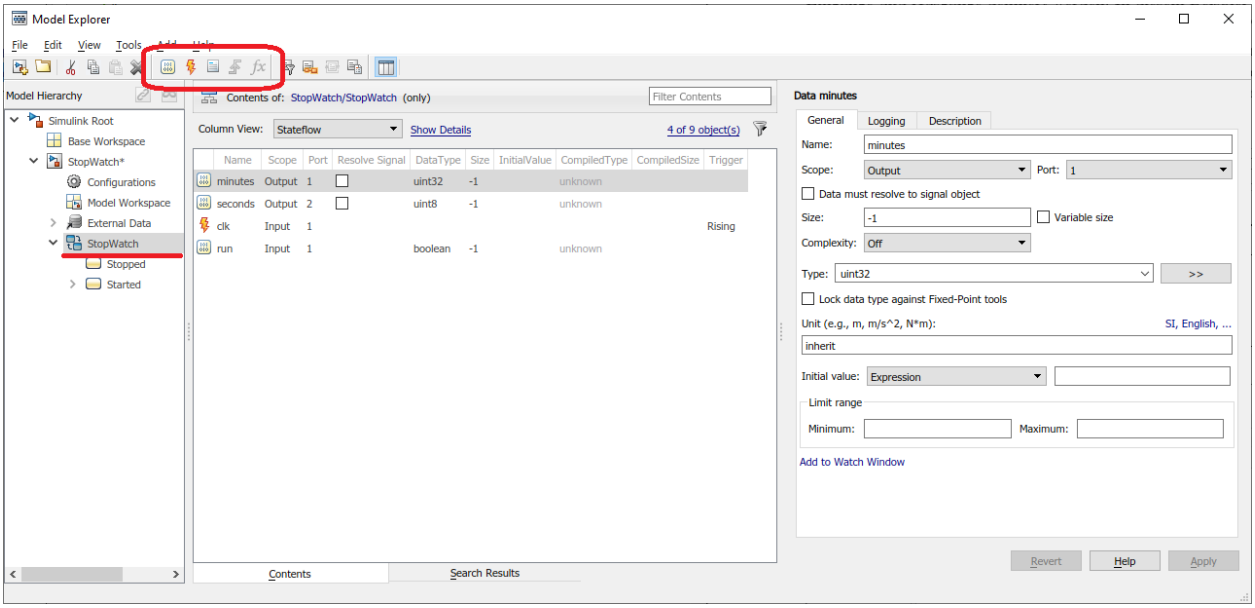
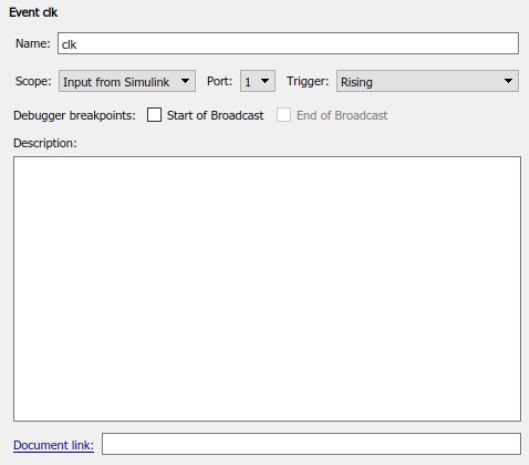
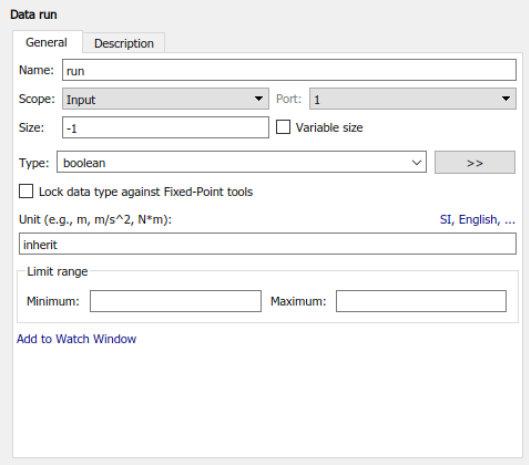


Abbildung 6: Der Model Explorer

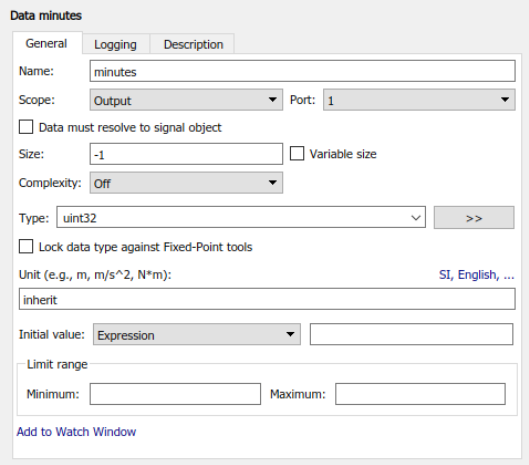
[8] Es gibt auch lokale Datenelemente, die nur innerhalb des Zustandsdiagrammes sichtbar sind, z.B. lokale Variablen. Es ist ratsam, ein Datenelement nur nach außen sichtbar zu machen, wenn dieses außerhalb des Diagramms benutzt werden soll.



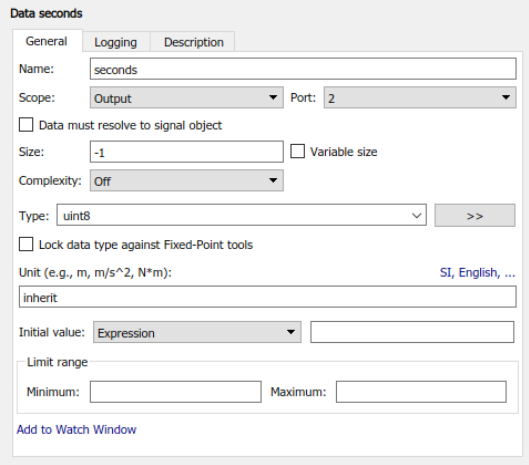
(a) Ereignis `clk`



(b) Eingang `run`



(c) Ausgang `minutes`



(d) Ausgang `seconds`

Abbildung 7: Definition der Datenelemente

1.4 Hinzufügen der Simulink Umgebung

Wechseln Sie nun wieder zurück zum Simulink-Fenster[9]. Es enthält immer noch nur einen Stateflow-Block. Man kann allerdings erkennen, dass dieser Block jetzt Ein- und Ausgänge besitzt. Es handelt sich um die Datenelemente, die zuletzt definiert wurden. Da diese nicht als lokal definiert wurden sind sie nun von außen sichtbar.

Sie müssen nun diese Ein- und Ausgänge mit den passenden Elementen verbinden, damit die Stoppuhr funktionieren kann. Öffnen Sie den *Library Browser* (Abb. 8) durch einen Klick auf `Simulation` → `Library Browser`.

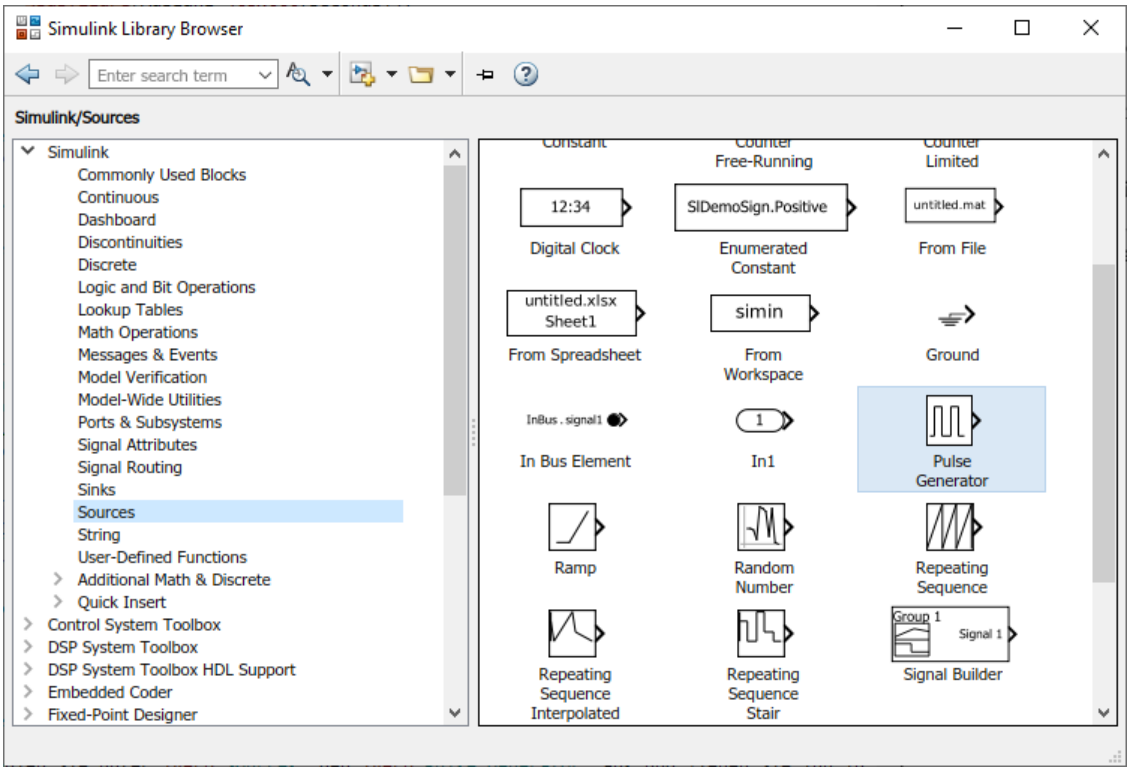


Abbildung 8: Der Library Browser

Wählen Sie unter `Simulink` → `Sources` den `Pulse Generator` aus und ziehen Sie ihn in das Simulink Fenster. Verbinden Sie nun den Ausgang des Generators mit dem *clk*-Eingang der Stoppuhr (angedeutet durch das Symbol einer steigenden Clock-Flanke). Doppelklicken Sie auf den Pulse Generator und ändern Sie `Period` auf 1 Sekunde. Für spätere Anwendungen ist zu beachten, dass es bei Stateflow-Blöcken nur einen Trigger-Eingang zur Versorgung externer Events gibt. Sollen mehrere Eingangsevents verwendet werden, so muss man über Multiplexer (Simulink Block *mux*) Vektoren von Inputs erzeugen. Die Zuordnung von Eingansevents zu den gewünschten Signalen muss durch die korrekte Vergabe der Port-Nummer erfolgen. Näheres ist im Hilfetext zu finden.

[9] Sie können mit den Pfeil-Symbolen im Modell navigieren. Mit einem Rechtsklick können Sie auswählen ob ein Block in einem separaten Fenster oder in einem neuen Tab geöffnet werden soll.

Fügen Sie zwei Blöcke vom Typ `Constant` hinzu. Sie können dies auch ohne den `Library Browser` erreichen, indem Sie auf eine freie Stelle der Simulink-Fläche klicken, den Namen `Constant` eingeben und die Auswahl mit der Enter-Taste bestätigen.

Ändern Sie die Konstanten durch Doppelklick auf das Symbol einmal auf `0` und im zweiten Fall auf `1` (Abb. 9). Ändern Sie unter `Signal Attributes` zusätzlich den Typ auf `boolean`.

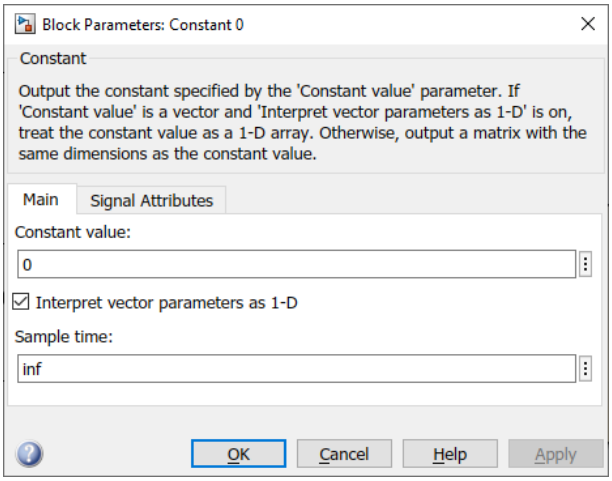


Abbildung 9: Definition eines konstanten Wertes

Verbinden Sie beide konstanten Werte mit den Eingängen eines `Manual Switch` (zu finden unter `Signal Routing`). Den Ausgang des Schalters verbinden Sie mit dem Eingang *run*.

Nun müssen noch die beiden Ausgänge des Stateflow-Blocks verbunden werden. Nutzen Sie hierfür zwei Blöcke vom Typ `Display` (zu finden unter `Sinks`).

Nun sollte Ihr Modell ungefähr wie in Abbildung 10 aussehen.

Hinweis:

In obigem Beispiel wird der Stateflow Chart durch eine Clock „getriggert“. Das heißt, der Chart wird immer dann aktiviert, wenn das clk-Event eine steigende Flanke aufweist (Trigger-Type: Rising). Man kann natürlich auch mehrere Input-Events definieren.
Es ist auch möglich gar kein Input-Event zu verwenden. Die eingestellte *Sample-Time* definiert dann, zu welchen Zeitpunkten ein Chart ausgeführt wird.
Im *Model Explorer* kann man in den Eigenschaften für einen Stateflow Chart angeben, mit welcher Update-Methode der Chart ausgeführt werden soll. Entweder erbt der Stateflow-Block die Sample-Time von der übergeordneten Hierarchieebene (standard) oder es ist explizit eine diskrete Sample-Time angegeben, evtl. mit Super-Step-Semantik.
Das Konzept der *Continuous Time* ist bei den Harel Statecharts nicht bekannt und spielt in diesem Praktikum keine Rolle. Machen Sie sich immer klar, zu welchen Zeitpunkten ein Statechart ausgeführt wird.

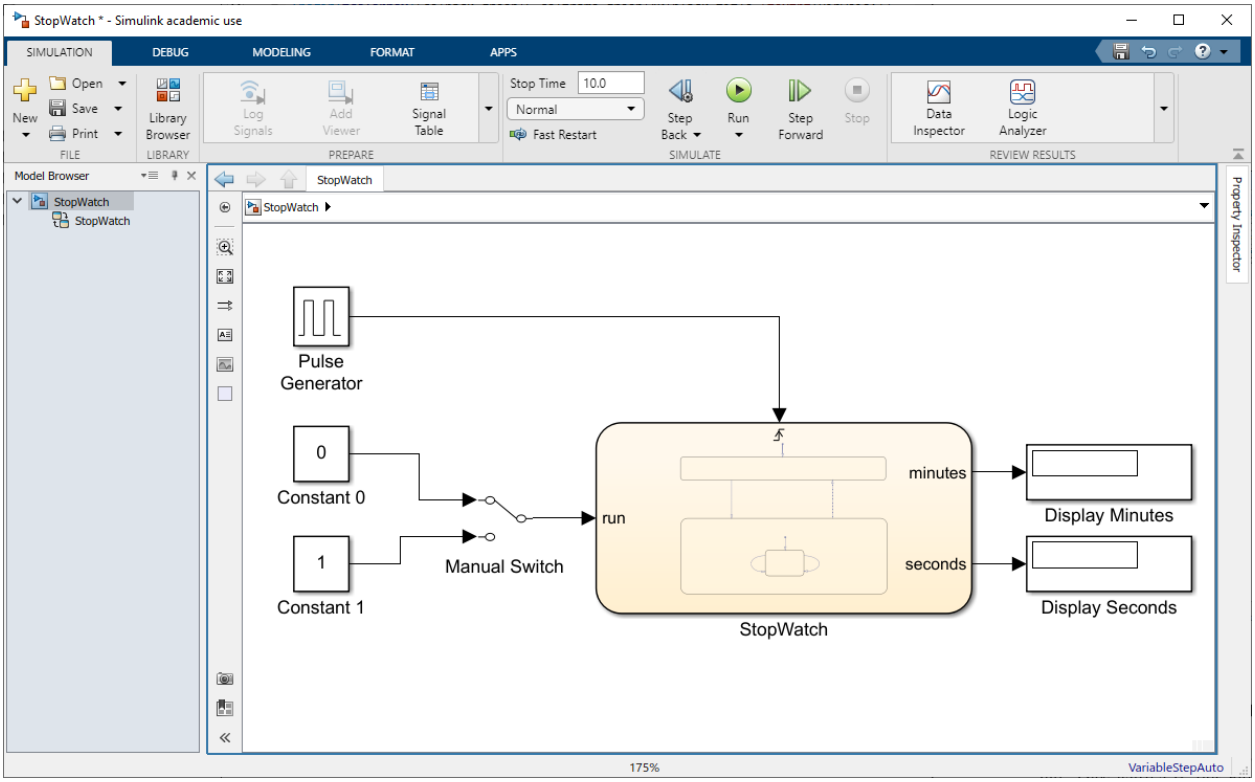


Abbildung 10: Das fertige Simulink-Modell

1.5 Simulation des Systems

Der Standard-Simulationsalgorithmus hält die Simulation an, wenn er einen stabilen Zustand erreicht hat. Dies wäre z.B. der Fall, wenn $run = 0$ gilt. Um dies zu verhindern öffnen Sie bitte `Modeling` → `Model Settings`.

In dem neuen Fenster ändern Sie im Abschnitt `Solver` den `Type` auf `Fixed-step`.
Die `Fixed-step size` ändern Sie von `auto` auf `1e-3`, eine Millisekunde, ab.
Ändern Sie die `Stop time` auf `1000.0` damit die Simulation ausreichend lange laufen kann.

Bitte beachten Sie, dass die Clock für die Stoppuhr durch den Generator auf 1 Sekunde eingestellt ist. Daher bekommt der Stateflow-Block jede Sekunde eine steigende Flanke. Dies löst (triggert) wiederum die Transitionen im Stateflow-Block aus. Jedoch heißt dies nicht unbedingt, dass die Simulation in Echtzeit abläuft. Je nach Simulationsaufwand, kann die Simulation schneller oder langsamer ablaufen. Sie können die Simulationsgeschwindigkeit anpassen indem Sie im Menü unter `Simulation` → `Run` das Untermenü ausklappen und die Option `Simulation Pacing` wählen. Hier können Sie die Simulationsgeschwindigkeit auf Echtzeit verlangsamen in dem Sie die Option „Enable pacing to slow down simulation“ mit einem Verhältnis von 1 Sekunde Simulationszeit pro Sekunde Echtzeit aktivieren. Dies ist in Abb. 11 gezeigt.

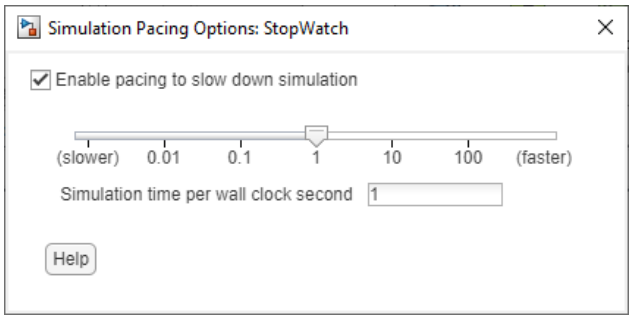


Abbildung 11: Simulationsgeschwindigkeit festlegen

Starten Sie nun die Simulation durch einen Klick auf `Simulation` → `Run`.

Während der Simulation können Sie den Schalter durch Doppelklick betätigen und so die Stoppuhr anhalten und fortsetzen. Wenn Sie das Stateflow Modell öffnen, können Sie durch farbliche Änderungen der Zustände und Transitionen erkennen, was gerade passiert (dies verlangsamt allerdings die Simulationsgeschwindigkeit). Sollten Sie beim Start der Simulation Fehlermeldungen oder Warnungen erhalten, dann versuchen Sie diese zu verstehen und gegebenenfalls zu beheben.

Mit `Simulation` → `Stop` können Sie die Simulation wieder beenden.

Aufgabe:

Prüfen sie, was passiert, wenn Sie den Wertebereich von `seconds` auf $0, \ldots, 58$ ändern. Sie können die Änderung im Model-Explorer vornehmen.

2 Erweiterung der Stoppuhr

Aufgabe:

Um sich besser mit Simulink und Stateflow vertraut zu machen, werden Sie nun die Uhr schrittweise erweitern.

a) Fügen Sie einen *lap*-Schalter für die Anzeige der Zwischenzeit hinzu, d.h. die Anzeige bleibt stehen, während die Stoppuhr intern weiterläuft. Man könnte dies sehr einfach erreichen, indem man im Simulink-Modell die Anzeigen von der Stoppuhr trennt. Hier soll aber eine Lösung gefunden werden, die die Anzeigen durch das Stateflow-Modell entsprechend steuert.

b) Fügen Sie einen *store/recall*-Schalter hinzu, d.h. der aktuelle Wert wird zwischengespeichert und ggf. später wiederhergestellt. Dies bedeutet genauer:

- *Schalter ein:* Die aktuelle Zeit wird intern gespeichert, egal ob die Stoppuhr gerade läuft oder nicht.
- *Schalter aus:* Die Stoppuhr wird auf den zuvor gespeicherten Wert zurückgesetzt, egal ob die Stoppuhr gerade läuft oder nicht.

Die Aktionen werden immer bei einer Änderung der Schalterstellung ausgeführt (steigende/fallende Flanke).

Stellen Sie sicher, dass Sie keine Updates des internen Zählers der Stoppuhr verpassen, d.h. benutzen Sie parallel ausgeführte Zustände (Rechtsklick `Decomposition` → `Parallel (AND)`).

c) Momentan wird die Uhr immer auf 0 zurückgesetzt, wenn man diese startet. Implementieren sie einen alternativen Resetmechanismus:
Die Zähler werde genau dann zurückgesetzt, wenn ein vollständiger *store/recall*-Zyklus durchlaufen wurde, wenn die Stoppuhr gleichzeitig angehalten ist. Dies bedeutet, die Stoppuhr wird nur dann auf 0 zurückgesetzt, wenn die Uhr angehalten ist und dann erst ein *store* und danach ein *recall* erfolgt. In der Zwischenzeit darf die Uhr nicht wieder aktiviert werden, ansonsten erfolgt kein Reset.

d) Überlegen Sie sich eine weitere Verbesserung der Stoppuhr und implementieren Sie diese.

e) Läuft Ihre Stoppuhr in Echtzeit? Wieso bzw. wieso nicht?

3 Zustandsaktionen

Selbstschleifen sind Transitionen, die denselben Zustand als Quelle und Ziel haben. Wie Sie bereits bei der Stoppuhr beobachten konnten kommen diese recht häufig vor und erhöhen nicht unbedingt die Übersichtlichkeit.

Aus diesem Grund gibt es Zustandsaktionen (engl: *State Actions*). Es gibt zwei Typen[10], die ausgeführt werden, während man sich in einem Zustand befindet:

- Die *during*-Aktion wird immer ausgeführt, wenn der Zustand aktiviert (getriggert wird). Die Syntax lautet:

  `during:` *action*

  Durch eine *during*-Action kann man eine Selbstschleife ohne Event oder Bedingung ersetzen.

- Zusätzlich gibt es noch Event-getriggerte Aktionen:

  `on` *event*`:` *action*

  Hiermit kann man nun unbedingte Selbstschleifen ausdrücken, welche an ein Event gebunden sind.

Um diese Zustandsaktionen einem Zustand zuzuweisen müssen Sie den Zustandsnamen mit einem Schrägstrich `/` erweitern. Dann können Sie in zusätzlichen Zeilen die Aktionen des Zustands eintragen:

```
Adams_adventurous_state /
on God_speaking: utter_wish_for_Eve
on Eve_whispering: eaten_apples++
```

Dadurch ist es nicht mehr nötig, für diese Übergänge extra Kanten einzuführen.

Achtung:

Auf den ersten Blick scheint es so, als wären die Zustandsaktionen nur eine abkürzende Schreibweise (*syntactic sugar*), allerdings gibt es kleine aber feine semantische Unterschiede:

- Zustandsaktionen werden nur ausgeführt, wenn es keine Transition gibt, welche aktiviert werden kann. Sollte eine Transition ausführbar sein, werden die obigen Zustandsaktionen *nicht* ausgeführt! Sie haben also eine niedrigere Priorität als Transitionen.
- Transitionen (auch Selbstschleifen!) verlassen einen Zustand. Dies macht einen Unterschied bei der Verwendung von temporaler Event-Logik (siehe nachfolgenden Abschnitt), da beim Verlassen eines Zustands die internen Event-Zähler zurückgesetzt werden. Zustandsaktionen verlassen einen Zustand nicht.

[10] Es gibt eigentlich noch mehr Zustandsaktionen, Sie finden diese in der Hilfe unter dem Stichwort *state actions*.

Aufgabe:

Falls Ihre Stoppuhr Selbstschleifen enthält, dann vereinfachen Sie sie durch die Verwendung von Zustandsaktionen. Speichern Sie vorher Ihr Modell unter einem neuen Namen (z.B. `stopwatch-extended`).

4 Temporaler Event-Operator

Gelegentlich möchte man das Auftreten eines Events zählen, um dann bei bestimmten Werten eine Aktion auszuführen. Durch die Einführung von lokalen Zählern könnte man dies erreichen, es ist aber nicht sehr elegant und macht das Modell unübersichtlich, da man zusätzliche Übergänge für die internen Zähler benötigt. Stateflow bietet dafür die temporalen Event-Operatoren an[11]. Die Syntax lautet:

`after(`*number* `,` *event* `)`

Diese kann man im Bedingungsteil von Transitionen anstatt gewöhnlicher Events verwenden, z.B.:

```
after(4, Calls_from_God ) [eaten_apples > 0] / leave_paradise
```

Die Transition wird aber nur aktiviert, wenn der Quellen-Zustand während der 4 Events nicht zwischenzeitlich verlassen wurde, da jedes Verlassen des Zustands die Event-Zähler zurücksetzt. Dies muss insbesondere bei Selbstschleifen beachtet werden, da auch diese den Zähler zurücksetzen.

Es gibt noch mehr als nur den `after`-Operator. Werfen sie einen Blick in die Hilfe (Stichwort: *Temporal Logic*).

Aufgabe:

Falls Sie Ihre Stoppuhr durch Verwendung des `after`-Operators vereinfachen können, dann tun Sie dies.

5 Varianten von Statecharts

Aufgabe:

Beschreiben Sie schriftlich die wichtigsten Unterschiede zwischen den Statecharts wie sie hier benutzt werden und den Statecharts wie sie von Harel definiert wurden. (Dies kann auch tabellarisch erfolgen.)

[11] Dies wird von Mathworks etwas irreführend *temporal logic events* genannt.

6 Die Messung von Frequenzen

Die Messung von Frequenzen ist eine grundlegende Operation im Bereich *Embedded Control*. Stellen Sie sich z.B. einen Sensor an einer Achse eines Fahrzeugs vor. Bei jeder Umdrehung aktiviert der Sensor für einen kurzen Moment ein Signal. Würde nun ein Controller dieses Signal auswerten und bestimmen wie oft es aktiviert wird (Frequenz), könnte man daraus die Geschwindigkeit ableiten.

Grundsätzlich gibt es zwei Möglichkeiten eine Frequenz zu bestimmen:

a) Das Zählen von Ereignissen innerhalb eines bestimmten Zeitfensters.

b) Die Messung der Zeit, die zwischen zwei einzelnen Ereignissen vergeht.

Beide Varianten haben ihre Vor- und Nachteile. Deshalb wird man in der Praxis oft auf Kombinationen oder Weiterentwicklungen dieser Grundprinzipien treffen. In Systemen, die ihre Eingänge in festen Zeitintervallen (Polling) abfragen, haben die beiden Methoden ihre größte Ungenauigkeit in entgegengesetzten Bereichen der Frequenzskala.

Aufgabe:

Auf der Vorlesungsseite finden Sie die Datei `FrequencyCount.mdl`. Öffnen Sie diese mit Matlab.
Das Modell enthält:

a) einen Frequenzgenerator, den man auf eine Frequenz zwischen 1Hz und 500Hz einstellen kann (Doppelklick auf den Slider).

b) einen leeren Stateflow-Block *Frequency Counter*, mit

   1) einem Ausgang *Measurement*. Hier soll das Messergebnis ausgegeben werden.

   2) einem Eingang *Signal*.

   3) einem Trigger *clk*, der den Block mit 1kHz ansteuert, d.h. das Zustandsdiagramm wird jede Millisekunde einmal ausgeführt.

c) ein Oszilloskop *Freq*, welches das Messergebnis und den Sollwert anzeigt.

Implementieren Sie nun beide obigen Formen der Frequenzmessung im Stateflow-Block.
Vergleichen Sie beide Möglichkeiten miteinander bei unterschiedlichen Frequenzen. Fügen Sie hierzu einen weiteren Ausgang hinzu, um beide Varianten auf dem Oszilloskop zu vergleichen.
Was ist ein vernünftiges Zeitfenster für die Variante a)?
Versuchen Sie anschließend ein Messverfahren zu entwickeln, dass für den ganzen Frequenzbereich gut funktioniert (z.B. durch geschickte Kombination beider Methoden).

Checkpoint:

Bevor Sie mit Teil 2 fortfahren dürfen, führen Sie bitte einem Betreuer die bislang erstellten Statecharts vor und demonstrieren Sie Ihre schriftlichen Ausarbeitungen (**blaue Aufgabenboxen**).