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

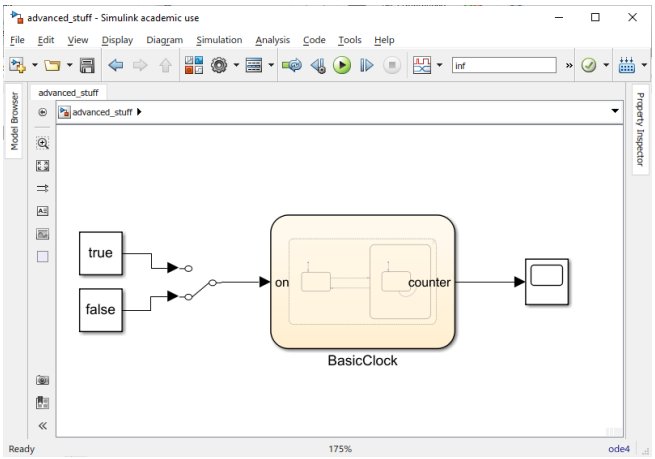
Freiburg, 29. April 2020

Praktikum
Embedded Systems Engineering

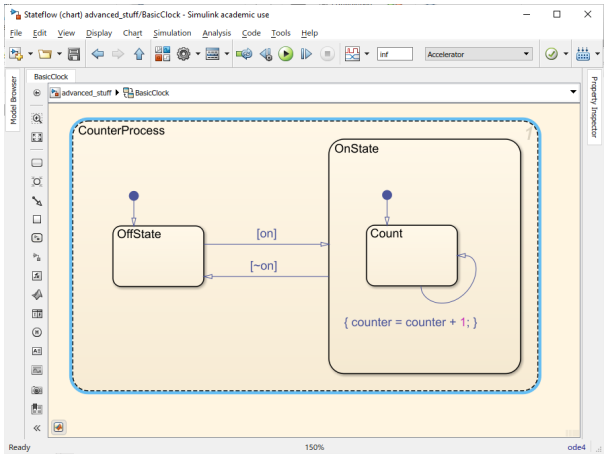
Teil 2.5
Erweiterte Simulink / Stateflow Funktionen

1 Modell

In diesem Dokument werden weitere, bisher noch nicht behandelte Funktionen von Simulink und Stateflow vorgestellt, welche für die Implementierung der automatischen Fahrrad-Gangschaltung hilfreich sein könnten. In Abbildung 1 ist die verwendete Testumgebung gezeigt. Diese besteht aus einem einzelnen Stateflow-Block, welcher exemplarisch einen Zähler mit einem Aktivierungs-Eingang und einem einzelnen Ausgang umsetzt.



(a) Simulink Modell



(b) Enthaltenes Stateflow Modell

Abbildung 1: Verwendetes Beispielprojekt

2 Initialisierung von Variablen / Konstanten

Wie in Abbildung 2 zu sehen kann mit einem Klick auf das Tabellen-Symbol der Model Explorer geöffnet werden. Dort können die Parameter des Stateflow Modells verwaltet, weitere Eingänge, Ausgänge, Variablen und Konstanten hinzugefügt und auch wieder entfernt werden.

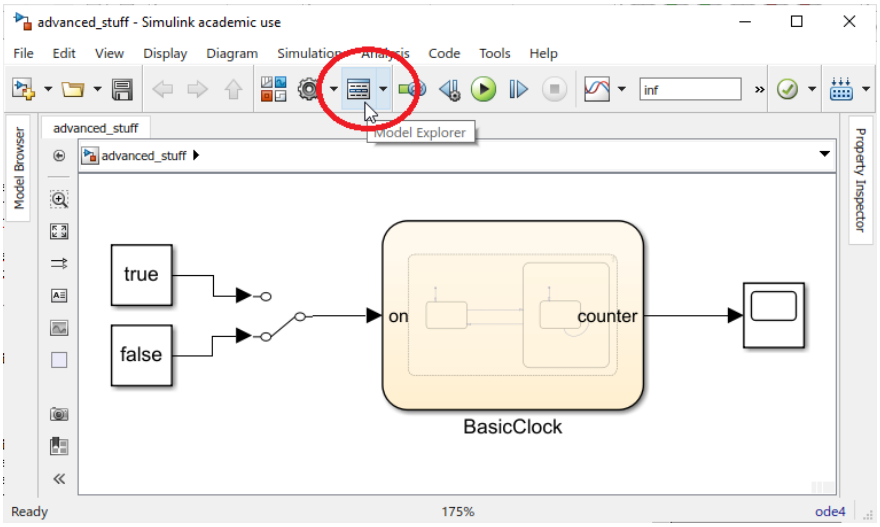


Abbildung 2: Öffnen des „Model Explorer“

Wie in Abbildung 3 dargestellt kann ein neuer Eingang / Ausgang oder eine neue Variable / Konstante mit einem Klick auf das „Daten“-Symbol (1) erstellt werden. Nach Festlegen des Namens wird unter Scope (2) der Typ festgelegt. Hierbei wird zwischen Input (Eingang), Output (Ausgang), Local (Variable) und Constant (Konstante) unterschieden. Unter Typ (3) sollte nun der genutzte Datentyp fest eingestellt werden um spätere Fehler durch Typverwechselungen und dadurch folgende „kryptische“ Fehlermeldungen zu vermeiden. Unter dem Feld Initial Value (Startwert) sollte ein Wert eingetragen werden, welcher zur Start der Simulation der Variablen oder dem Ausgang zugewiesen wird. Der Datenwert muss unter Umständen vor der Zuweisung konvertiert werden. Weitere Informationen dazu findest Du im Anhang.

Wie in Abbildung 4 gezeigt können Arrays und Look-Up Tabellen mit Hilfe des Size (1) Parameters erstellt werden. Dieser sollte auf die Anzahl an Elementen des Arrays oder der Tabelle eingestellt werden. Unter Constant value / Initial value (2) können in eckigen Klammern mit Kommata getrennte Werte gesetzt werden. In Stateflow kann mit eckigen Klammern hinter dem Variablen-/Konstantennamen auf einen beliebigen Eintrag zugegriffen werden - Beispiel: „events[3]“ ruft den Wert „5.0“ aus dem in der Abbildung gezeigten Array ab.

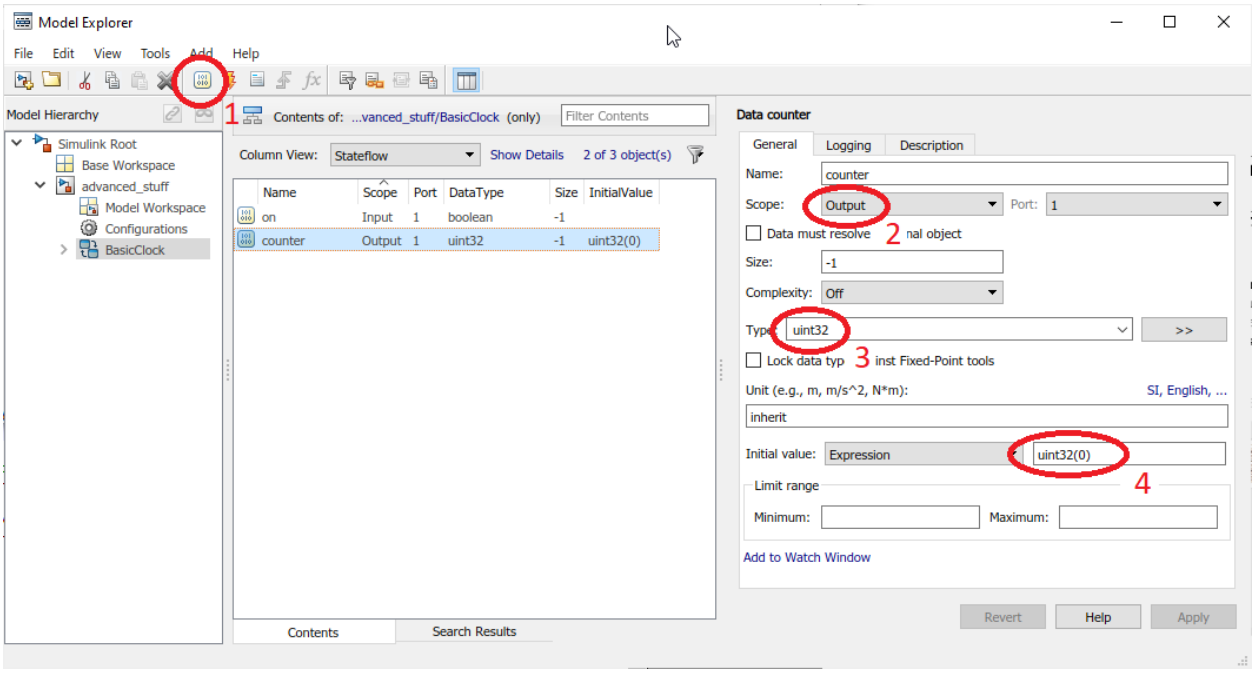


Abbildung 3: Erstellen / bearbeiten von Variablen mit Hilfe des „Model Explorer“

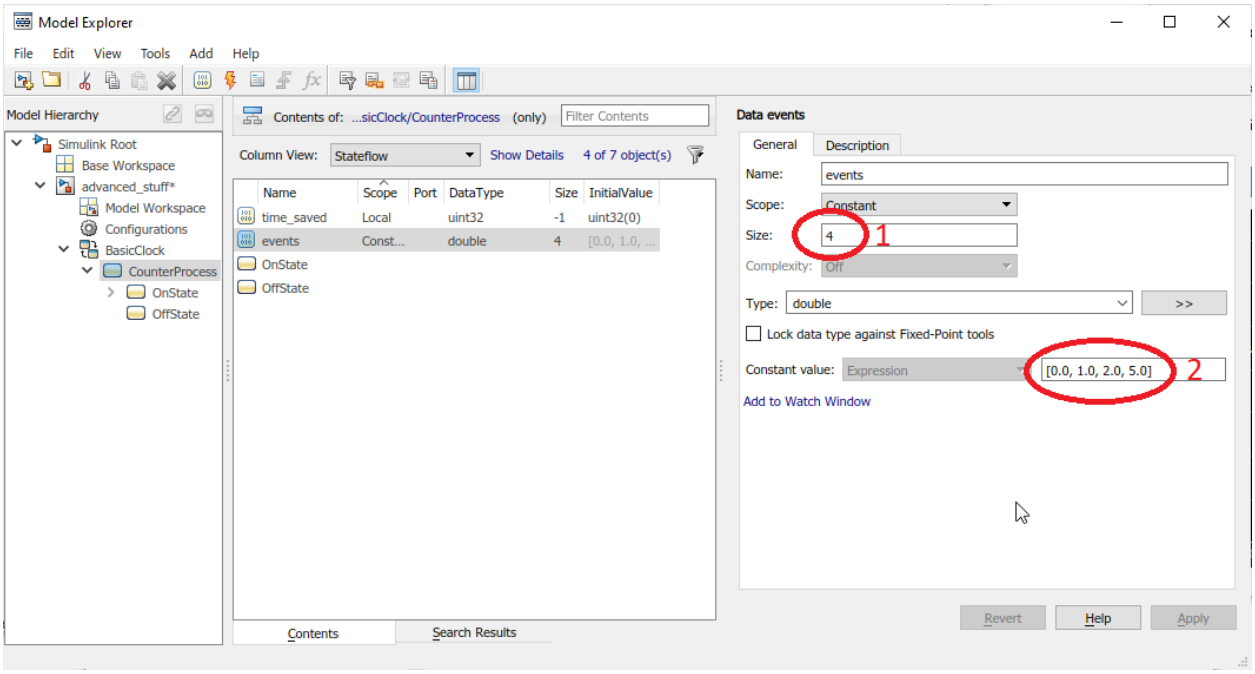


Abbildung 4: Look-Up Tabelle als Konstante im „Model Explorer“

3 Hierarchische Statecharts

Stateflow bietet die Funktion Statecharts zu verschachteln und somit in logische Abschnitte zu unterteilen. Dies schafft Ordnung und hat den Vorteil, dass Variablennamen in verschiedenen Kontexten mehrfach verwendet werden können, ohne dass diese im Konflikt stehen. So kann beispielsweise die Variable „counter“ zum Zählen von Clock-Pulsen in einem „ClockCounter“ Subchart genutzt werden, während die Variable „counter“ die Anzahl an Ein-/Ausschaltzyklen in einem „OnOffCounter“ Subchart zählt.

Um ein Subchart zu erstellen kann, wie in Abbildung 5 zu sehen ist, mit der rechten Maustaste ein State-Block angewählt und der Menüeintrag Group & Subchart gefolgt von Subchart benutzt werden. Auch die Verwendung der Tastenkombination Strg+Shift+G ist möglich. Um ein Subchart wieder in einen gewöhnlichen State-Block zu ändern kann die oben genannte Methode noch einmal verwendet werden.

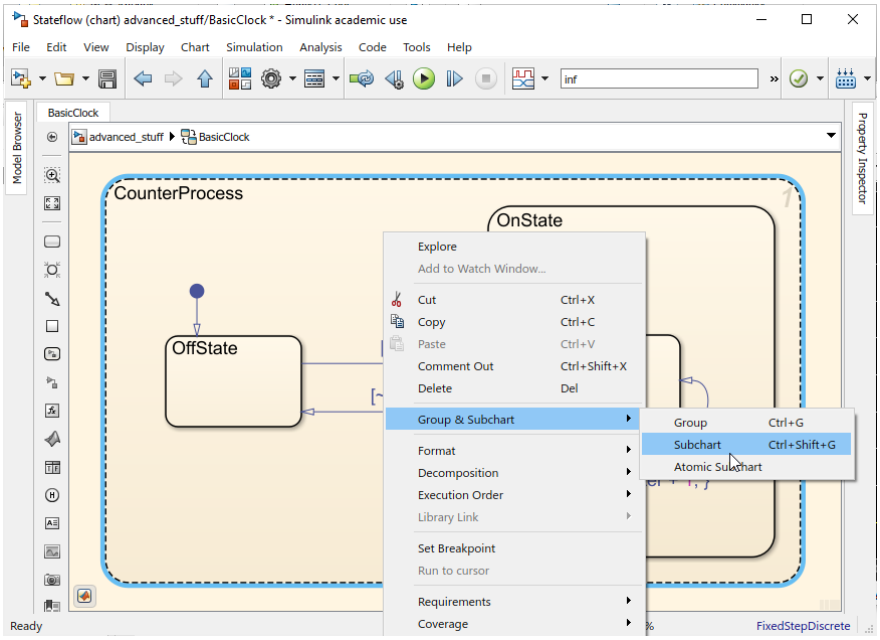
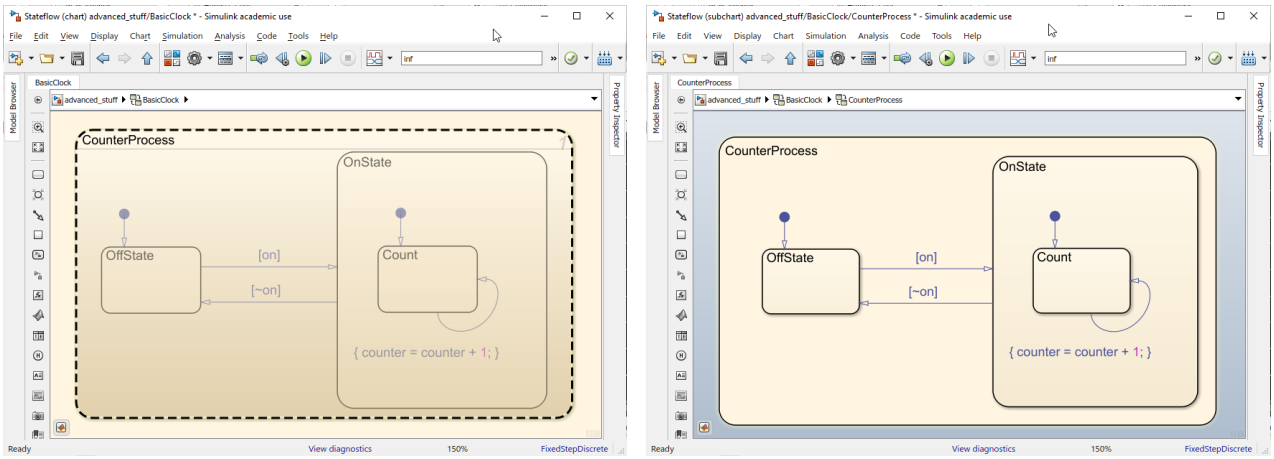


Abbildung 5: State-Block in Subchart konvertieren

In Abbildung 6a ist zu erkennen, dass das erzeugt Subchart nun wie ein einfacher State-Block dargestellt wird. Dieser kann nun auch wie ein normaler Block in der Größe verändert werden. Dessen Inhalt wird jedoch nicht durch Verschieben oder Größenänderungen beeinflusst. Durch einen Doppelklick auf den Block kann das Subchart, wie in Abbildung 6b gezeigt, geöffnet werden. Der äußere Rahmen kann, ohne Beeinflussung des Blockes im umgebenden State-Chart, in der Größe verändert werden.

Wie in Abbildung 7 gezeigt, können Variablen im Model Explorer innerhalb dieses Subcharts definiert werden. Diese sind nur in diesem Subchart und weiteren Unter-Subcharts sichtbar. Somit lässt sich ein versehentliches Überschreiben der Variable durch einen Funktional unzusammenhängenden State-Block vermeiden.



(a) Subchart in einem Stateflow Block (b) Innerhalb des Subcharts

Abbildung 6: Ansichten eines Subcharts

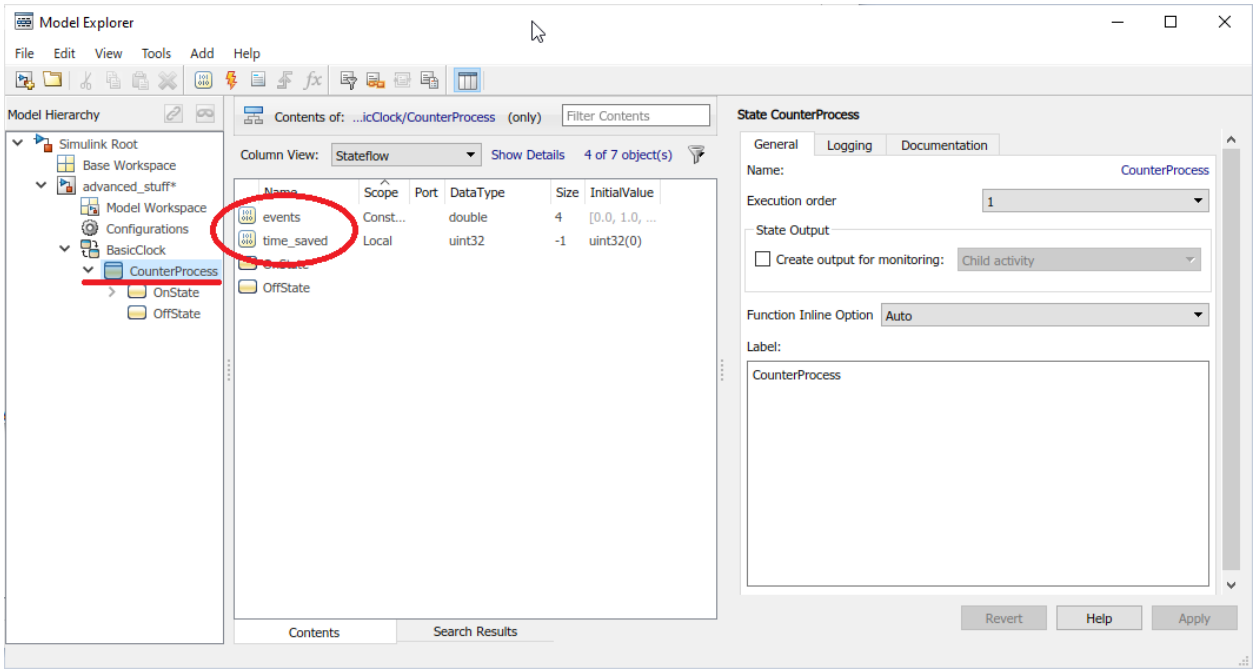


Abbildung 7: Lokale Variablen innerhalb eines Subcharts

4 Daten-Inspektor

Um die Fehlerfindung zu vereinfachen stellt Simulink den „Data Inspector“ bereit. Dieser kann genutzt werden um Datensignale und interne Variablen zu überwachen und somit den internen Zustand der Zustandsmaschine des Stateflow-Blocks anzuzeigen.

Durch Markieren von Simulink Signalen („Pfeil“ oder auch „Verbindung“) können diese, wie in Abbildung 8 gezeigt, zum Daten-Inspektor hinzugefügt werden. Dies erfolgt durch einen Klick auf das „Signal“-Symbol und die Auswahl der „Log Selected Signals“ (Ausgewählte Signale aufzeichnen) Menüoption, während die gewünschten Signale markiert sind.

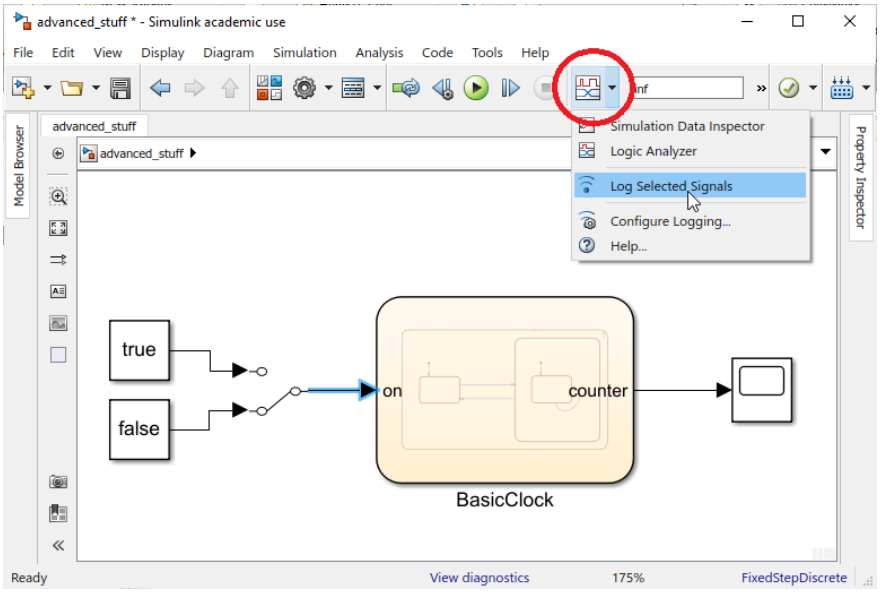


Abbildung 8: Signal zum Daten-Inspektor hinzufügen

Durch Markieren eines Stateflow-Blocks und Auswahl der Menüoption „Log Chart Signals“ (Diagramm Signale aufzeichnen) unter dem „Signal“-Symbol kann, wie in Abbildung 9 gezeigt, ein Dialog geöffnet werden. Dieser wird genutzt um Stateflow Signale an- und abzuwählen und diese in die Aufzeichnung miteinzubeziehen.

Der Stateflow-Signal Dialog ist in Abbildung 10 exemplarisch dargestellt. Es können Variablen, Eingänge und Ausgänge durch den Daten-Inspektor aufgezeigt werden. Aber auch der aktuelle Zustand der Stateflow Zustandsmaschine kann aufgezeichnet werden, indem vor den betreffenden Zuständen jeweils ein Haken gesetzt wird. Das aufgezeichnete Signal hat den Datenwert „1“, wenn der Zustand oder ein Unterzustand aktiv ist oder „0“, wenn sich die Zustandsmaschine in einem anderen Zustand befindet.

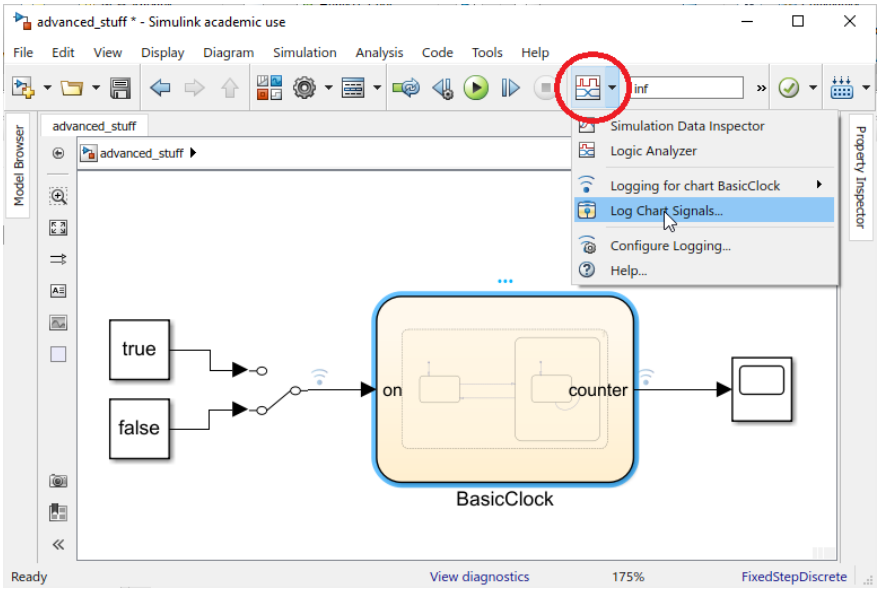


Abbildung 9: Stateflow-Signale zum Daten-Inspektor hinzufügen

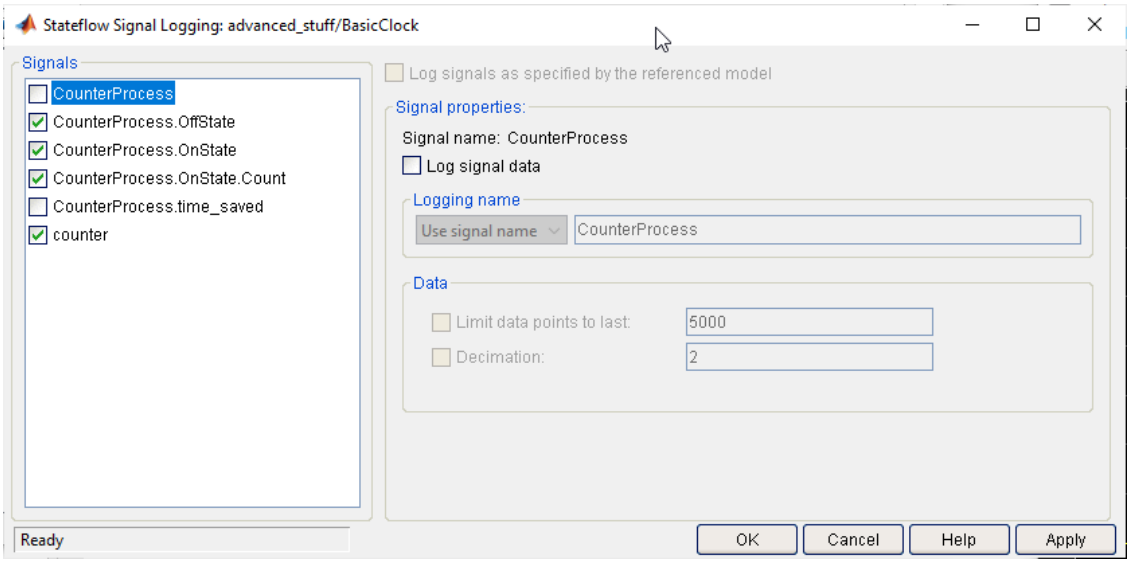


Abbildung 10: Stateflow-Signale Auswahldialog

Wie in Abbildung 11 gezeigt, kann der Daten-Inspektor nach erfolgter Simulation der Simulink Umgebung durch einen Klick auf das „Signal“-Symbol und Auswahl der Menüoption „Simulation Data Inspector“ (Dateninspektor) geöffnet werden. Wie in Abbildung 12 zu sehen, können einzelne Signale auf der linken Seite des Daten-Inspektors angewählt und zur aktuellen Ansicht hinzugefügt werden. Mit einem Klick auf das „Kachel“-Symbol (1) kann das angezeigte Layout geändert werden. Ein Klick auf das „Zeiger“-Symbol kann ein Cursor zur aktuellen Anzeige hinzugefügt werden. Durch Verschieben des Cursors kann das Datensignal zu einem beliebigen Zeitpunkt angezeigt und analysiert werden.

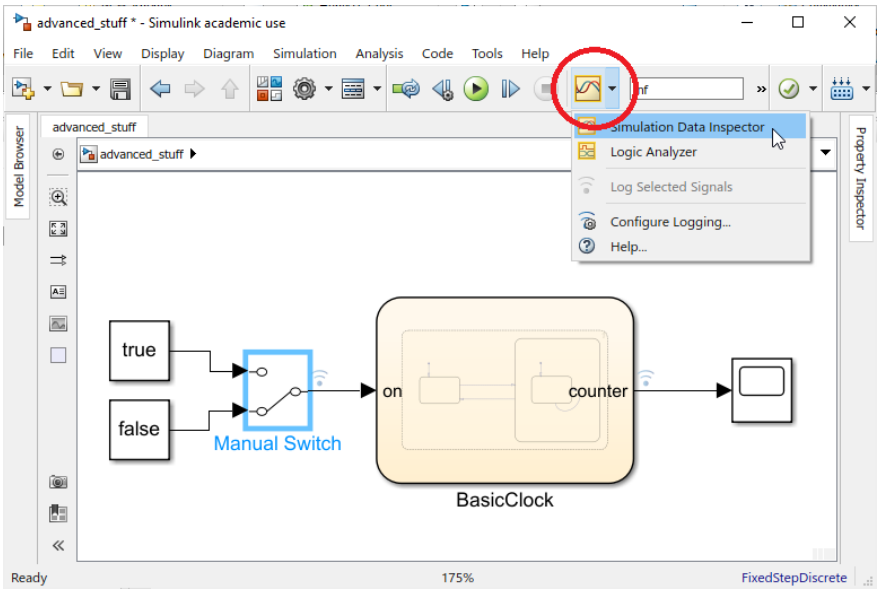


Abbildung 11: Daten-Inspektor öffnen

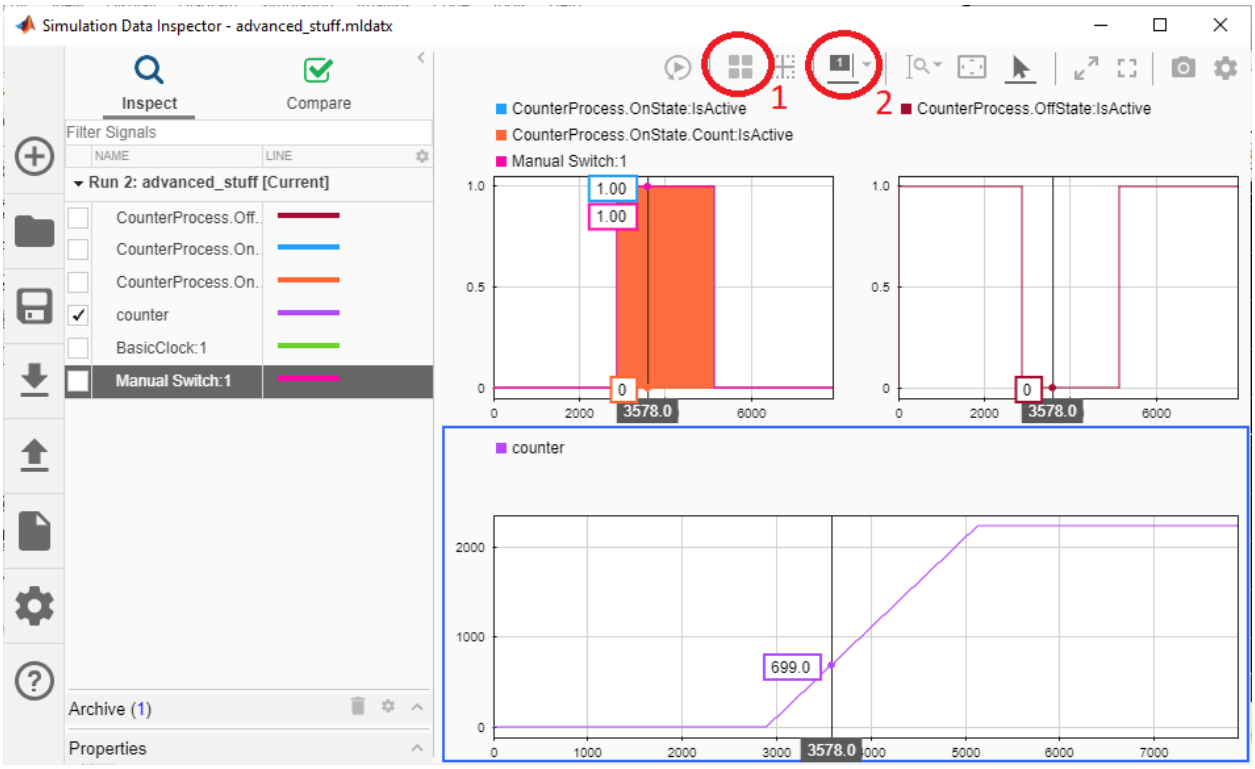


Abbildung 12: Daten-Inspektor mit Cursor und gewählten Signalen

5 Zusatz

5.1 Bedingungen

Bedingungen werden in Stateflow in rechteckige Klammern gefasst. Diese bestehen aus einem Boolschen Ausdruck, welcher aus mehreren Unterbedingungen oder Variablen bestehen kann. Bedingungen können mit den Operatoren `&&` (und), `||` (oder) und `^` (xor) zu einem Boolschen Ausdruck vereint werden.

Um zwei oder mehrere Bedingungen in Stateflow zu verknüpfen sollte immer der `&&`-Operator oder der `||`-Operator benutzt werden um Fehler zu vermeiden und Rechenzeit zu sparen. Die Funktionsweise kann unter Wikipedia: Kurzschlussauswertung nachgelesen werden. Zum Vergleich siehe Wikipedia: Logischer Operator

```
1 [ a == 42 ]
2 [ a == 1337 || a == 7353 ]
```

5.2 Boolsche Variablen - `boolean`

Der Typ `boolean` kann verwendet werden um genau zwei Werte, `true` und `false` zu speichern. Wie in (5) zu sehen kann mit dem ~-Operator der Ausdruck invertiert werden.

```
1 a = true
2 b = false
3
4 [ a ]       % Prüfe ob a wahr ist
5 [ ~b ]      % Prüfe ob a falsch ist
6 [ ~b && a ] % Prüfe ob b falsch und a wahr ist
```

5.3 Ganze Zahlen - `int32` und `uint32`

Die Typen `int32` und `uint32` stellen jeweils eine Vorzeichenbehaftete / -lose ganze Zahl dar. Sie lassen sich auf eingebetteten Plattformen meistens einfach verarbeiten und sind somit für den generischen Einsatz geeignet. Zahlen mit Typ `int32` und `uint32` haben jeweils einen Zahlenraum von 0 bis $2^{32} - 1$ und -2^{31} bis $2^{31} - 1$. Um in Matlab eine Zahl von Typ `int32` und `uint32` zu erzeugen muss diese wie in (1) und (2) gezeigt konvertiert werden. Vergleiche zwischen zwei `int32` oder `uint32` Zahlen können durch die einfachen Vergleichsoperatoren `==` , `~=`, `>`, `>=`, `<` und `<=` erfolgen.

```
1  a = uint32(42) % Konvertiere den Wert 42 zu uint32
2  b =  int32(a)  % Konvertiere Wert der Variable a zu int32
3
4  [ a == 1 ]    % Vergleiche ob beide ganzen Zahlen übereinstimmen
5  [ b ~= -1 ]   % Vergleiche ob beide ganzen Zahlen sich unterscheiden
```

5.4 Gleitkommazahlen - `single` und `double`

Gleitkommazahlen, also die Typen `double` und `single`, stellen die Möglichkeit zur Verfügung irrationale Zahlen mit begrenzter Genauigkeit (64, 32 und 16 Bit, je nach System) zu verarbeiten und speichern. Berechnungen mit Gleitkommazahlen erfordern auf eingebetteten Systemen eine Floating-Point-Unit (FPU) und sind rechenintensiver als gewöhnliche ganze Zahlen. Wenn kein Datentyp in Matlab / Simulink angegeben wird, wird meistens `double` als Datentyp angenommen.

Für Floating-Point Zahlen sollte nicht der `==`-Operator verwendet werden um auf einen bestimmten Wert zu prüfen. Dies hat den Ursprung, dass einige Zahlen in der Floating-Point Speicherweise nicht exakt wiedergegeben werden können. Somit könnte ein einfacher Vergleich mit dem `==`-Operator fehlschlagen. Aus diesem Grund sollte immer auf einen Wertebereich um die gewünschte Zahl herum wie in (6) geprüft werden.

```
1 a = 2.718
2 b = double(-0.5)
3 c = single(3.14159)
4
5 [ a < 2.0 ]
6 [ c >= 3.141 && c <= 3.142 ] % Prüfe ob equivalent zu π
```