Realize Your Product Promise®

Model-Based Design
with
SCADE Suite®

Training Resources

Day 2

ANSYS

The SCADE Suite workflow

Exercise

Objective:

Remember the basic SCADE Suite workflow

Time: 10 min

Requirements:

Organize the following steps in the correct order:

- Simulate a model
- Check a model
- Generate a report
- Organize the workspace
- Create a new package
- Use a library operator
- Create a new operator
- Generate code
- Create a new project

© 2016 ANSYS, Inc.

AGENDA

Type declaration

Constant declaration

Operator and interface declaration

Control flow declaration

© 2016 ANSYS, Inc.

How to use common types such as floats, integers, or booleans?

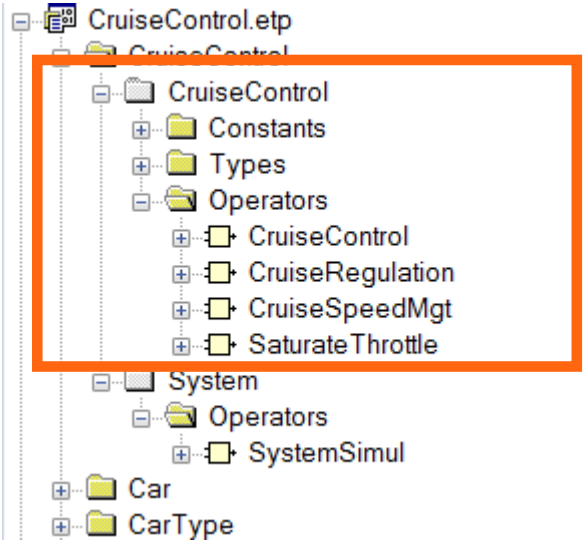
How to implement user-defined types to declare the system interface?

New Package Creation

A package allows to define a namespace and group elements together

Select the project / package where the package must be created:

- Create a package by clicking on: 
- Give it a name



© 2016 ANSYS, Inc.

Software Engineering Package Rules (Ada)

Ada

Find below the best practices to implement a Scade design for Ada code generation:

- Each declaration is encapsulated into a package
- Sensors, imported and non-imported elements are not mixed in the same package
- No top-down dependency (see next slides)
- No package circular dependency (see next slides)

ANSYS

© 2016 ANSYS, Inc.

Description of Package Dependencies (Ada)

Ada

P1 and P2 are packages:

Element of P1 depends on P2 declarations

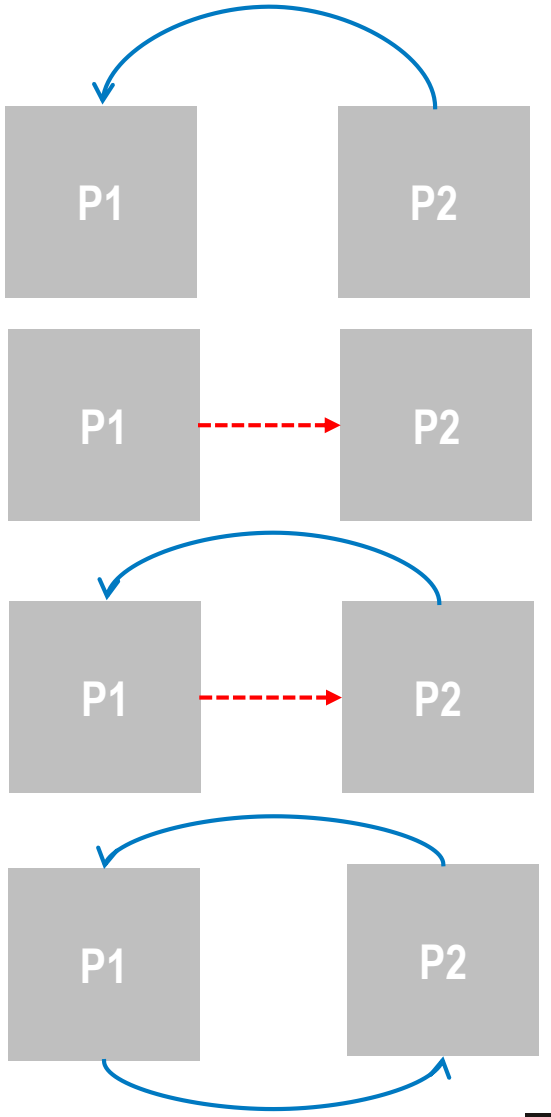
P2 is sub-package of P1

Top-down dependency:

- Element of P1 depends top-down on P2 sub-package declarations

Circular dependency between P1 and P2:

- Element of P1 depends on P2 declarations
- Element of P2 depends on P1 declarations



ANSYS

© 2016 ANSYS, Inc.

Conditions for Direct Translation into Ada Code

Ada

All global Scade declarations belong to package(s)

A package containing imported elements (including sensors) does not contain other non-imported elements at the same time

No package contains a global declaration that depends on one of its sub-packages

- But referring to a sibling package is allowed

There are no circular dependencies between packages

ANSYS

© 2016 ANSYS, Inc.

Ada Code Generation

Ada

When following constraints are not fulfilled, additional packages ensure the generated code remains SPARK-compliant

A warning message systematically informs users when this occurs:

- Non encapsulated declarations are available:

➔ *Kcg_Top* package is generated in the code to gather declarations

- A package P contains imported elements and / or sensors mixed with non-imported elements:

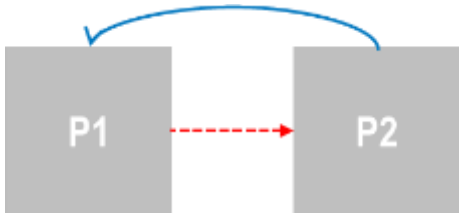
➔ *Kcg_Imported* and / or *Kcg_Sensor,* sub-packages of P*,* are generated in the code to separate declarations

ANSYS

© 2016 ANSYS, Inc.

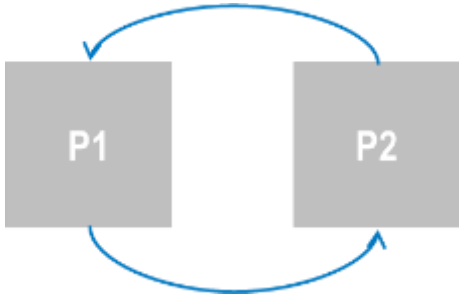
Ada Code Generation

Ada



Top-down dependency exists:

➔ Declarations of P1 are generated into *Kcg_sub,* sub-package of P1



Circular dependencies exist:

➔ Each declaration belonging to the circular dependencies is generated into a new package named *Kcg_<package>_<declaration>*

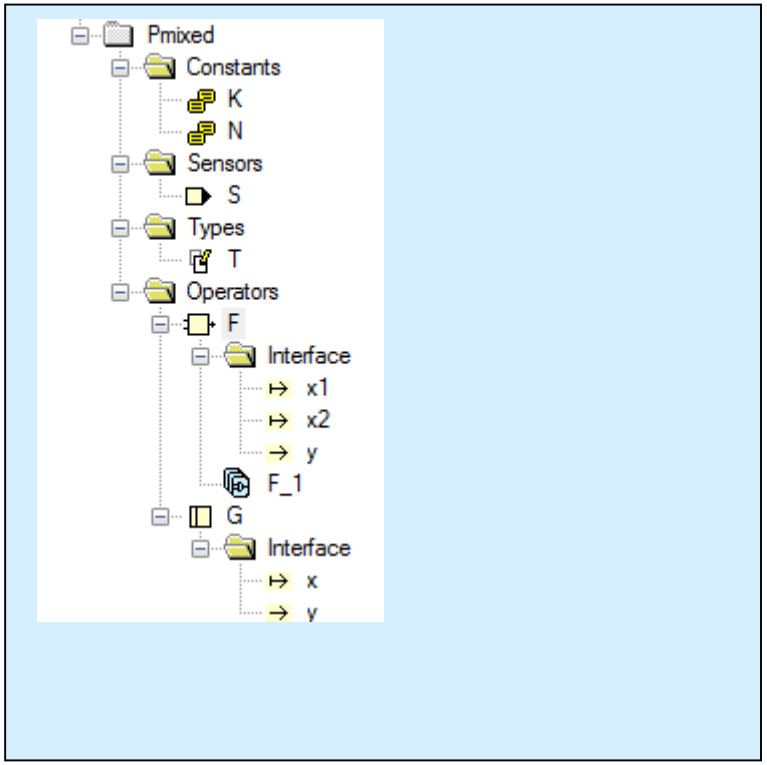
ANSYS

© 2016 ANSYS, Inc.

Mixed Elements (Ada)

Ada

Example




```
package Pmixed
 const K : int16 = 42;
 const imported N : int16;

 sensor S : float64 ;

 type imported T;

 function F (x1 , x2 : bool) returns (y : bool) ...
 function imported G (x : T) returns (y : T);

end;
```

Code generation: following message is raised to warn that additional packages are produced in the Ada code

```
Pmixed::: Introducing additional package
Pmixed::: new package "Kcg_Imported" to gather imported declarations
Pmixed::: Introducing additional package
Pmixed::: new package "Kcg_Sensor" to gather sensor declarations
```

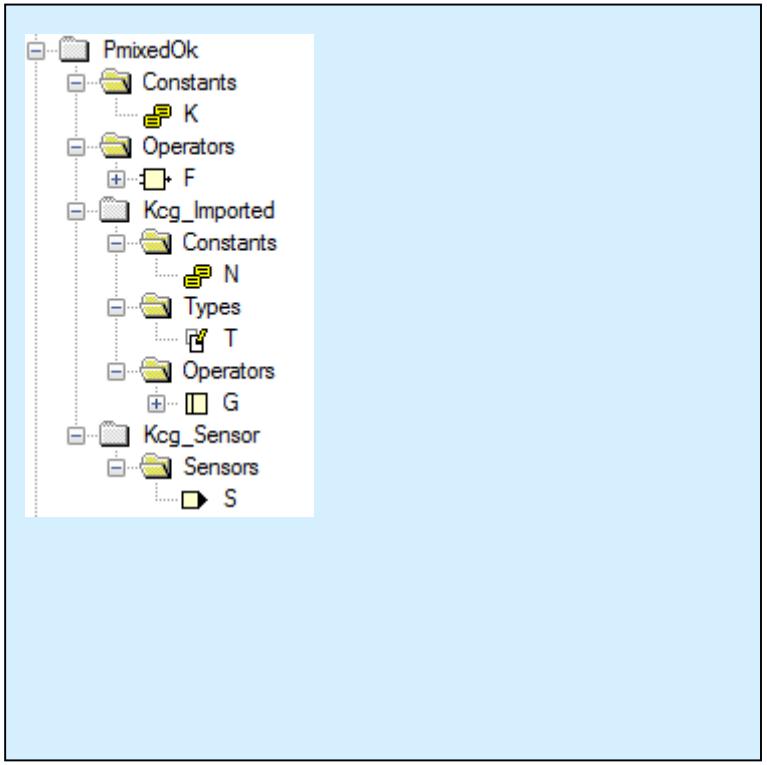
ANSYS

© 2016 ANSYS, Inc.

Mixed Elements (Ada)

Ada

Example




```
package PmixedOk
  const K : int16 = 42;
  function F (x1 , x2 : bool) returns (y : bool) ...

 package Kcg_Imported
  const imported N : int16;
  type imported T;
  function imported G (x : T) returns (y : T);
 end;

 package Kcg_Sensor
   sensor S : float64;
 end;

end;
```

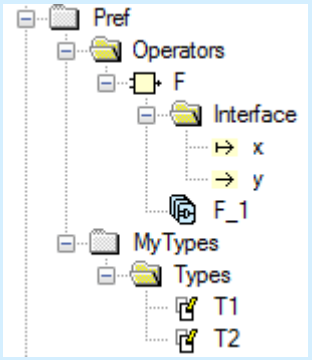
Direct code generation without additional packages

© 2016 ANSYS, Inc.

References to Sub-Packages (Ada)

Ada

Example




```
package Pref
 function F (x : MyTypes :: T1)
  returns (y :  MyTypes :: T2) ..

 package MyTypes
  type T1 = int32;
  type T2 = T1 ^ 10;
 end;

end;
```

Code generation: following message is raised to warn that one additional package is produced in the Ada code

```
Pref::: Introducing additional package
Pref::: new package "Kcg_Sub" to handle references to sub-packages
```

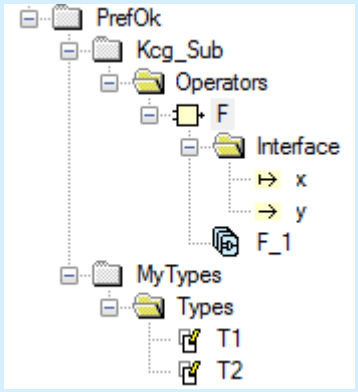
ANSYS

© 2016 ANSYS, Inc.

References to Sub-Packages (Ada)

Ada

Example




```
package PrefOk
 package Kcg_Sub
  function F (x : MyTypes :: T1)
   returns (y :  MyTypes :: T2) ...
 end;

 package MyTypes
  type T1 = int32
  type T2 = T1 ^ 10;
 end;

end;
-- sibling reference is ok
```

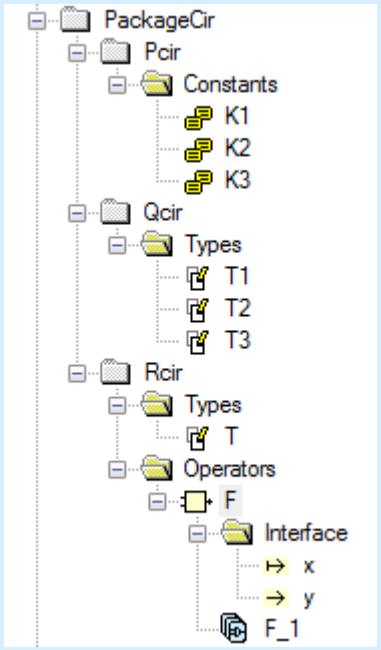
Direct code generation without additional packages

© 2016 ANSYS, Inc.

Circular Dependencies (Ada)

Ada

Example




```
package Pcir
 const K1 : Qcir:: T1 = 42;
 const K2 : int32 = K1 + 1;
 const K3 : int32 = 50;
end;

package Qcir
 type T1 = int32;
 type T2 = T1 ^ Pcir:: K1;
 type T3 = float32 ;
end;

package Rcir
 type T = Qcir:: T3 ^ 10;
```

Code generation: additional packages are produced in the Ada code

```
PackageCir::: Introducing additional package
PackageCir::: new package Kcg_Pcir_K1 to break circular dependencies between packages:
PackageCir::Pcir::K1/: The declaration of PackageCir::Pcir::K1/ depends on PackageCir::Qcir::T1/
PackageCir::Qcir::T2/: The declaration of PackageCir::Qcir::T2/ depends on PackageCir::Pcir::K1/
(declaration K1 has been moved from package Pcir to package Kcg_Pcir_K1)
PackageCir::: Introducing additional package
PackageCir::: new package Kcg_Pcir_K2 to break circular dependencies between packages:
PackageCir::Pcir::K2/: The declaration of PackageCir::Pcir::K2/ depends on PackageCir::Pcir::K1/
PackageCir::Pcir::K1/: The declaration of PackageCir::Pcir::K1/ depends on PackageCir::Qcir::T1/
PackageCir::Qcir::T2/: The declaration of PackageCir::Qcir::T2/ depends on PackageCir::Pcir::K1/
(declaration K2 has been moved from package Pcir to package Kcg_Pcir_K2)
PackageCir::: Introducing additional package
PackageCir::: new package Kcg_Qcir_T1 to break circular dependencies between packages:
PackageCir::Qcir::T2/: The declaration of PackageCir::Qcir::T2/ depends on PackageCir::Pcir::K1/
PackageCir::Pcir::K1/: The declaration of PackageCir::Pcir::K1/ depends on PackageCir::Qcir::T1/
(declaration T1 has been moved from package Qcir to package Kcg_Qcir_T1)
PackageCir::: Introducing additional package
PackageCir::: new package Kcg_Qcir_T2 to break circular dependencies between packages:
PackageCir::Qcir::T2/: The declaration of PackageCir::Qcir::T2/ depends on PackageCir::Pcir::K1/
PackageCir::Pcir::K1/: The declaration of PackageCir::Pcir::K1/ depends on PackageCir::Qcir::T1/
(declaration T2 has been moved from package Qcir to package Kcg_Qcir_T2)
```

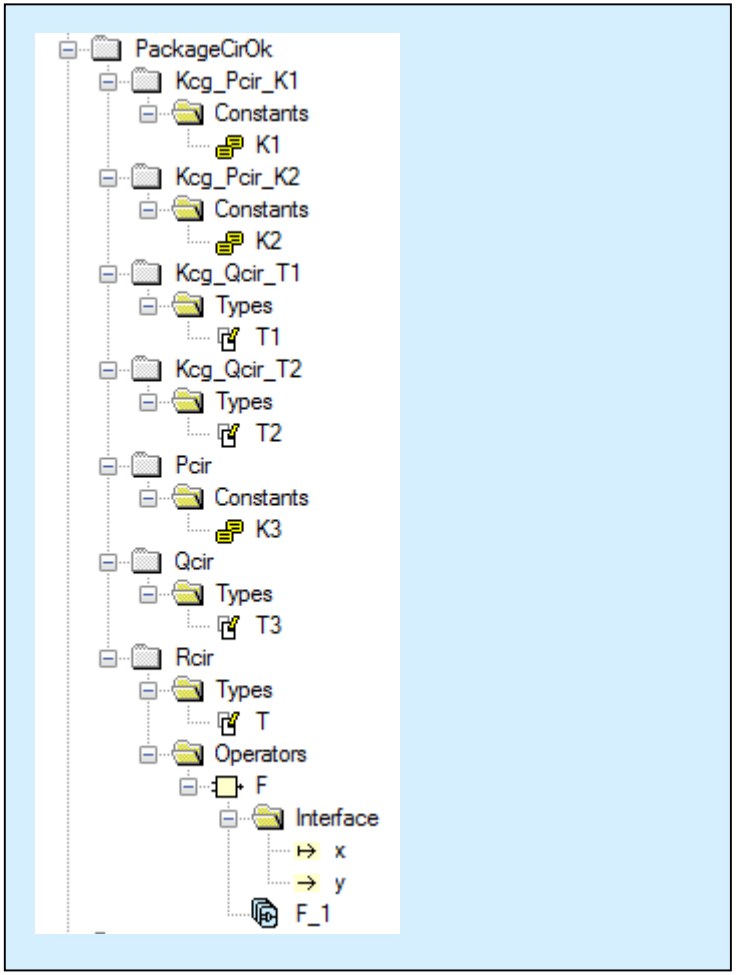

© 2016 ANSYS, Inc.

ANSYS Confidential
ANSYS

Circular Dependencies (Ada)

Ada

Example




```
package Kcg_Pcir_K1
 const K1 : Kcg_Qcir_T1 :: T1 = 42;
end;
package Kcg_Pcir_K2
 const K2 : int32 = Kcg_Pcir_K1 :: K1 + 1;
end;

package Kcg_Qcir_T1
 type T1 = int32;
end;
package Kcg_Qcir_T2
 type T2 = Kcg_Qcir_T1 :: T1 ^ Kcg_Pcir_K1 :: K1;
end;

package Pcir
 const K3 : int32 = 42;
end;

package Qcir
 type T3 = float32 ;
end;

package Rcir
 type T = Qcir:: T3 ^ 10;
 function F (x : Qcir:: T3) returns (y : T) ...
end;
```

Direct code generation without additional packages

ANSYS

© 2016 ANSYS, Inc.

Ada Code Generation

Ada

Function/procedures are translated into their own *.adb* file, with separate body feature

Boolean operators can be short-circuited or not

Predefined Scade types are defined in *Kcg_Config.ads* file located in *<installation>\SCADE\include\Ada*

© 2016 ANSYS, Inc.

Data Types

The Scade language is strongly typed:

- All the variables are explicitly typed
- Users can define as many types as necessary
- Conversion between values of different types are explicit

Scade predefined types:

- Boolean: **`bool`** and character: **`char`**
- Signed Integer: **`int8, int16, int32, int64`**
- Unsigned Integer: **`uint8, uint16, uint32, uint64`**
- Floating point: **`float32, float64`**

© 2016 ANSYS, Inc.

Data Types

User defined types:

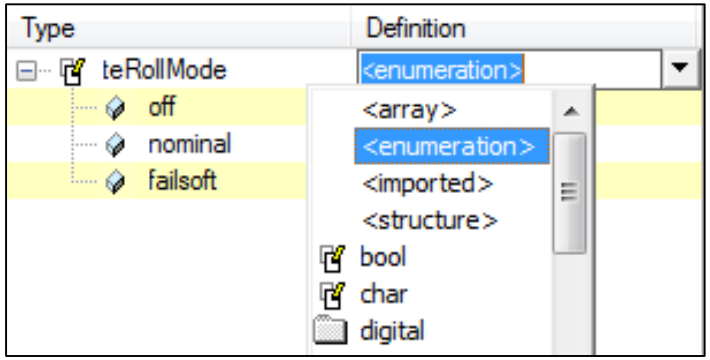
- **Equivalent types:**
  - **Speed = Velocity**
  - **Euro = Currency**
  - Enhances model readability
- **Enumerated types:**
  - **Colors = enum {blue, green, red}**
- **Structured types:**
  - Data structure: corresponds to usual structures or records
    - **Ts = {x: int8, y: float32}**
  - Array: always fixed size, the first index is 0
    - **Vector = float64^3**
    - **Table = float32^5^5**

© 2016 ANSYS, Inc.

New Type Creation

Select the package where the type must be created:

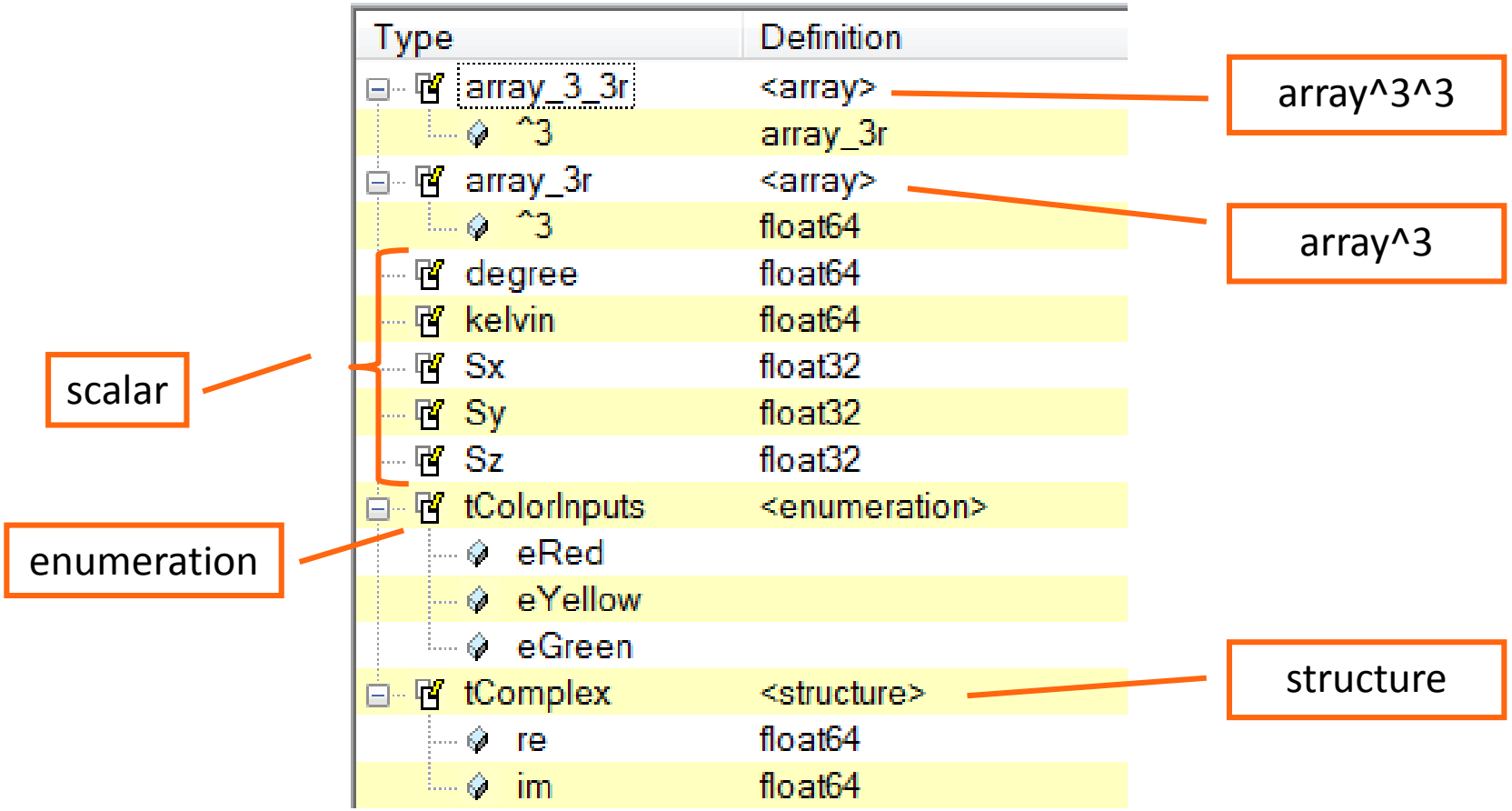
- Create a new type by clicking on 
- Select the type definition from the Definition column:
  - If the type definition needs a new definition element, click on 
  - Or right click on the type and select Insert > Definition element



© 2016 ANSYS, Inc.

New Type Creation

Example



© 2016 ANSYS, Inc.

AGENDA

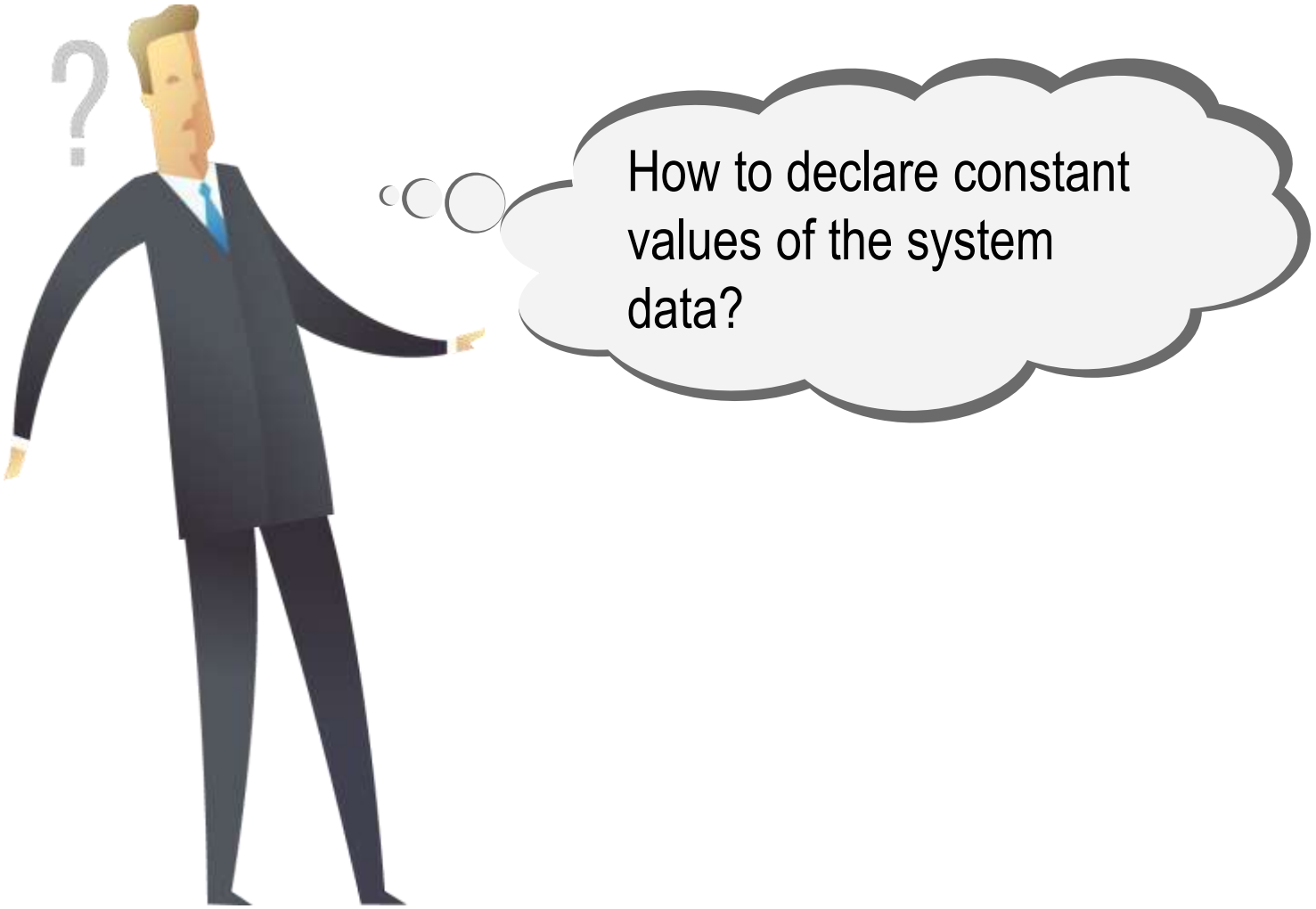
Type declaration

Constant declaration

Operator and interface declaration

Control flow declaration

© 2016 ANSYS, Inc.



 © 2016 ANSYS, Inc.

Scade Constants

Constants are data having the same value throughout the program execution

Constants can appear as:

- 3.14
- A textual expression: 4 * C1
- A name defined by a constant: C1 = 2.5 or C2 = 3 .0 * C1

Used for:

- Constant value definitions
- Parameters or settings of Scade constructs such as the size of an array

© 2016 ANSYS, Inc.

Literals

Literals  can be defined as:

A constant literal:

- `true, false`: Boolean
- `1.0` (floating-value) is different from `1` (integer)
- `10_ui32` has uint32 type
- `10_i16` has int16 type
- `10.0_f32` has float32 type
- Quote for char constants: `'a'`

An expression of constants:

- `C1 and C2`
- `a * c + b`

Expressions used to define constants must be statically defined at compilation time

© 2016 ANSYS, Inc.

Named Constants

Constants are always typed:

By named types:

- `MyCons : MyType = [3, 2];` (with MyType = uint32^2)

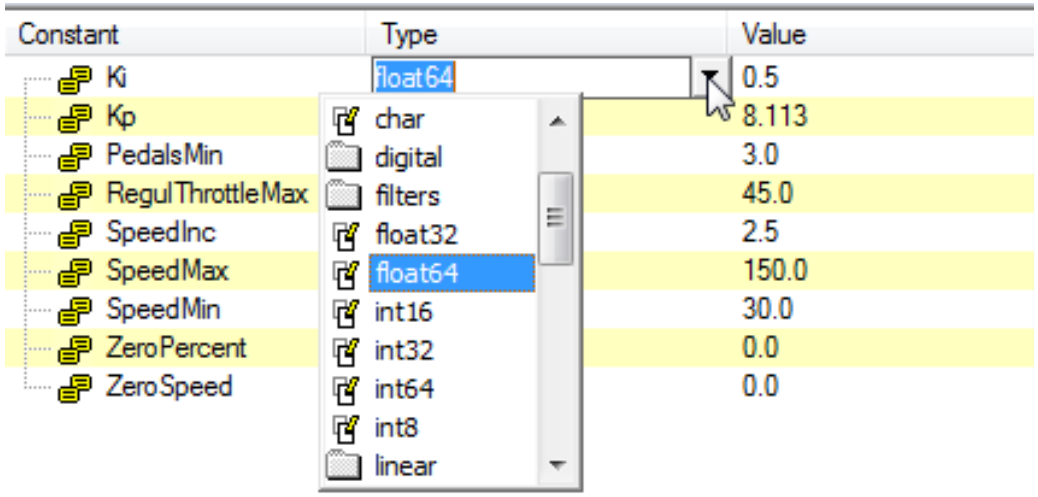
Or anonymous types:

- `MyCons : uint32^2 = [3, 2];`

Used in graphical and textual Scade

© 2016 ANSYS, Inc.

New Constant Creation



Select the package where the constant must be created:

- Create a constant by clicking on: 
- Give it a name and select its type: click on cell and browse in the displayed list
- Fill the appropriate fields according to the type definition

© 2016 ANSYS, Inc.

New Constant Creation

Example

Constant	Type	Value
BackgroundColor	tColorInputs	eGreen
cMode	char ^10	"Active "
ComplexA	tComplex	{re : 10.0, im : (-10.0)}
ComplexB	tComplex	
re	float64	5.5
im	float64	15.7
IdentityMatrix	array_3_3r	
0	array_3r	
0	float64	1.0
1	float64	0.0
2	float64	0.0
1	array_3r	[0.0, 1.0, 0.0]
2	array_3r	[0.0, 0.0, 1.0]
PosX	Sx	124.4
PosY	Sy	-4.589
PosZ	Sz	7.25
VectInit	array_3r	
0	float64	2.0
1	float64	2.0
2	float64	2.0

enumeration

structure

array^3^3

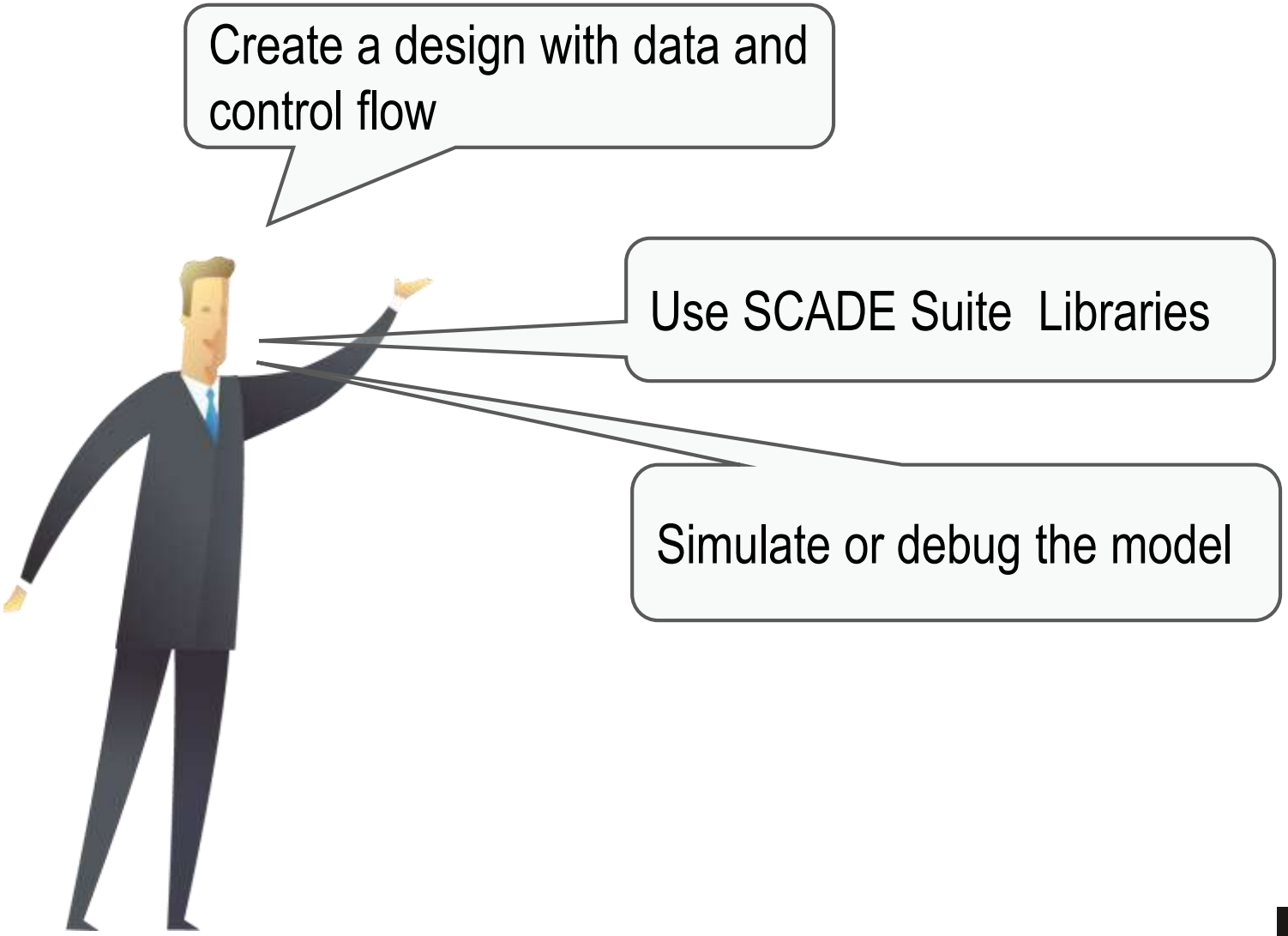
scalar

array^3

© 2016 ANSYS, Inc.

Lab Objectives

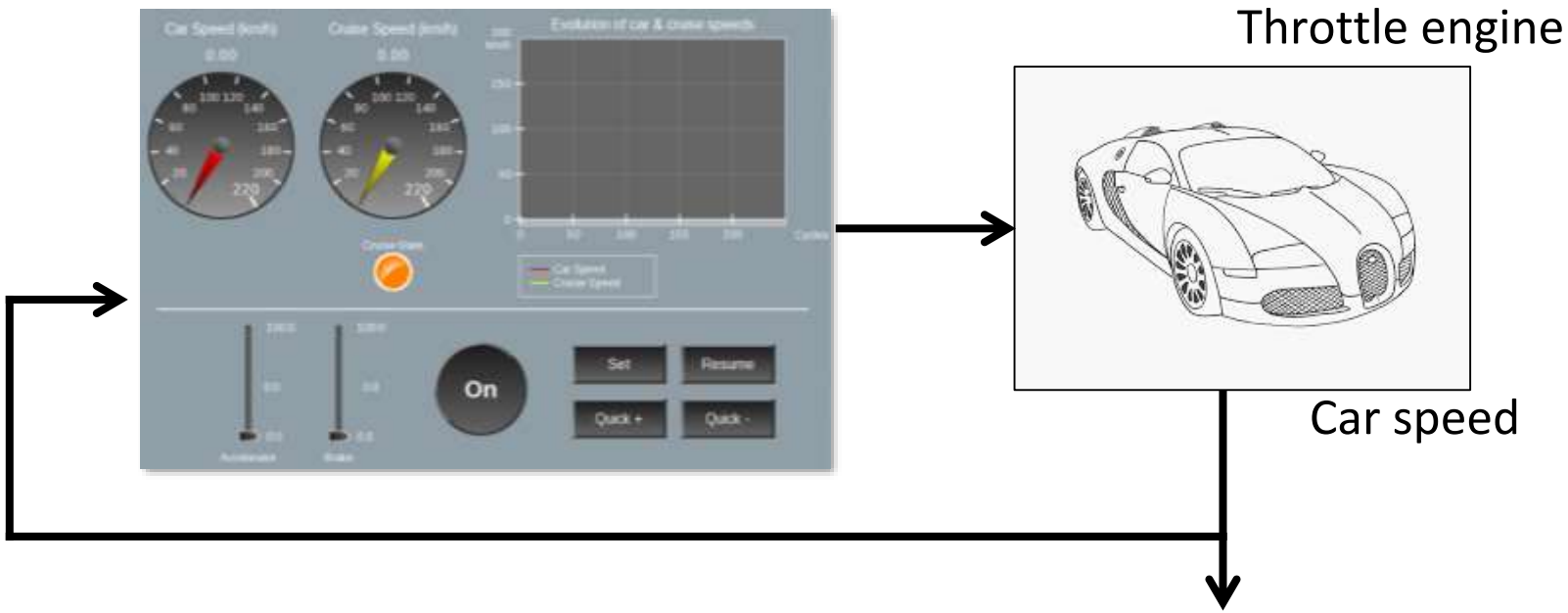
Lab



© 2016 ANSYS, Inc.

The System to Create

Build a system of a Cruise Control to drive the car speed in order to maintain a constant cruise speed



During the Lab, you will create bricks and assemble them to obtain the whole application

Lab

© 2016 ANSYS, Inc. February 1, 2017 ANSYS Confidential

Requirements

The software requirements of Cruise Control design are defined in *\Day 2\Labs66\Requirements*

The labs refer these requirements. However, you should be able to do the design without needing to read this document

Lab

© 2016 ANSYS, Inc.

Lab 1 (1/2)

Lab Support p.8-21

Objective:

Create a new type and new contants

Time: 20 min

Requirements:

Create a new SCADE Suite Project and a package:

Project Name: `Cruise Control`

SCADE Suite library: `Car.etp` (located at *Q:\Labs\Prerequisites\Lab 1\Libraries\Car*)

Package Name: `CruiseControl`

Create a new type enumerate `teCruiseState` *(defined in requirement CC_HLR_OUT_03)*

Type	Definition
teCruiseState	enum {OFF, INT, STDBY, ON}

Lab

© 2016 ANSYS, Inc. February 1, 2017 ANSYS Confidential

Lab 1 (2/2)

Lab

Create the constants *(defined in requirement CC_HLR_CCP_01 to CC_HLR_CCP_07)*

Name	Type	Value
KI	float64	0.5
KP	float64	8.113
PEDALS_MIN	CarType::tPercent	3.0
REGUL_THROTTLE_MAX	CarType::tPercent	45.0
SPEED_INC	CarType::tSpeed	2.5
SPEED_MAX	CarType::tSpeed	150.0
SPEED_MIN	CarType::tSpeed	30.0
ZERO_PERCENT	CarType::tPercent	0.0
ZERO_SPEED	CarType::tSpeed	0.0

© 2016 ANSYS, Inc. February 1, 2017 ANSYS Confidential

AGENDA

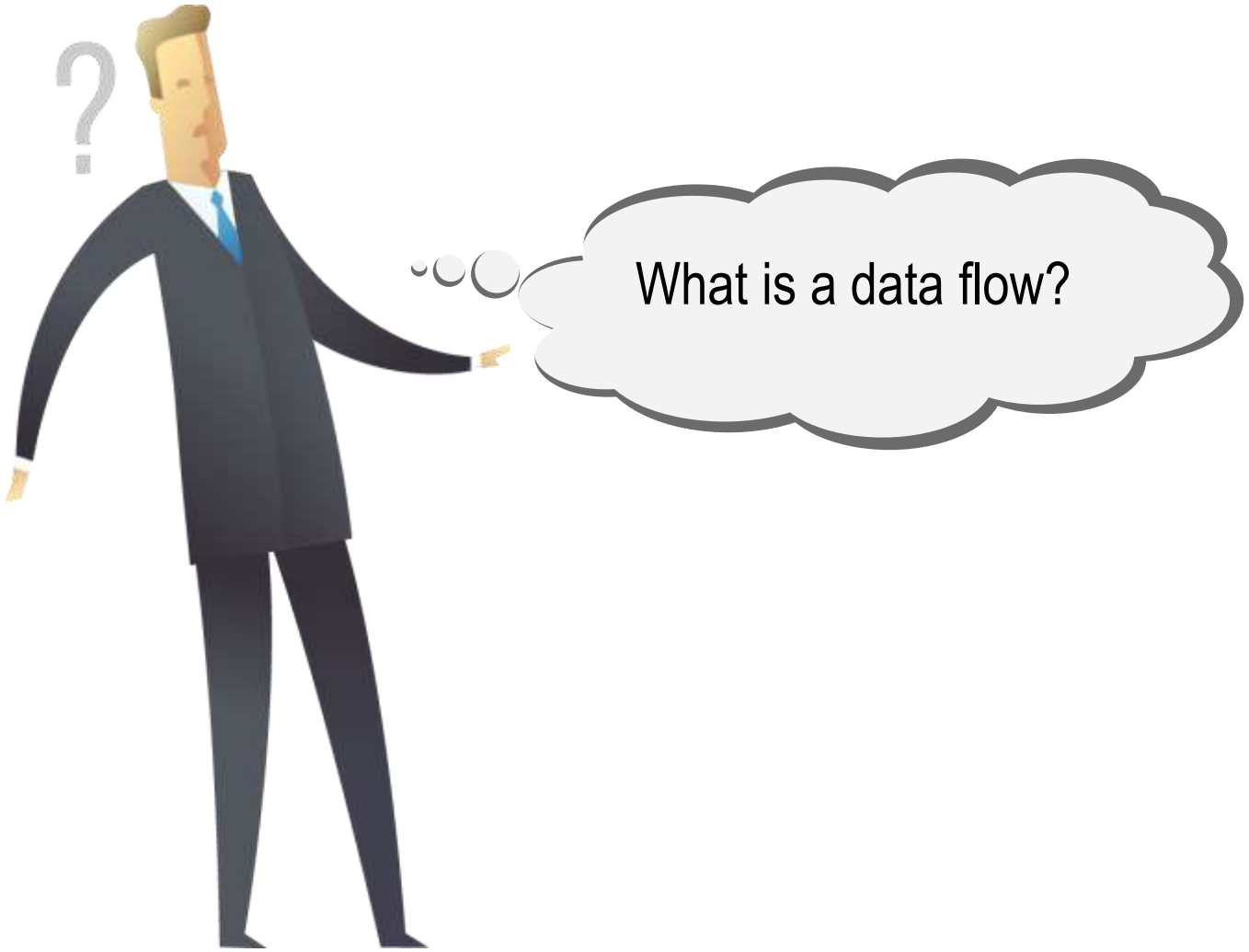
Type declaration

Constant declaration

Operator and interface declaration

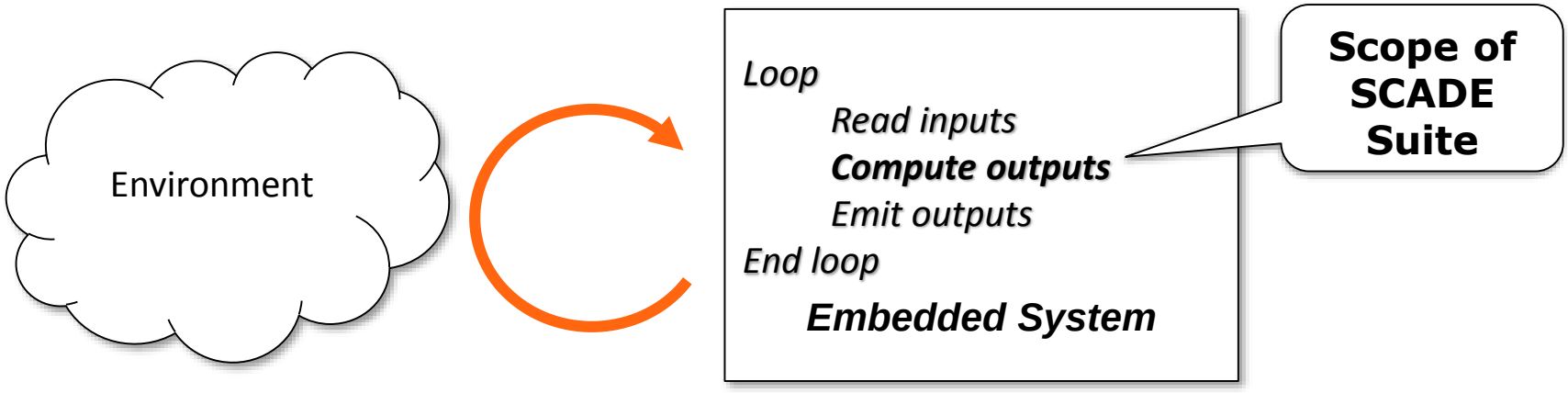
Control flow declaration

© 2016 ANSYS, Inc.



 © 2016 ANSYS, Inc.

Scade Works on Data Flows



- A SCADE Suite model is a cycle-based computation model where the program examines its inputs and provides the outputs in the same cycle
- Inputs and outputs are data flows
- A flow has a single definition and must always be defined
- Flows are streams of values, that are infinite sequences of values of a given type

© 2016 ANSYS, Inc.

Scade Works on Data Flows

Example

If v is a data flow expression, its values sequence are v1, v2, ...

3 denotes the constant flow 3, 3, ...

a + b denotes the flow a1 + b1, a2 + b2, ...

Outputs computation may require input and output values of previous cycles:

The relationship between inputs and outputs is time-dependent

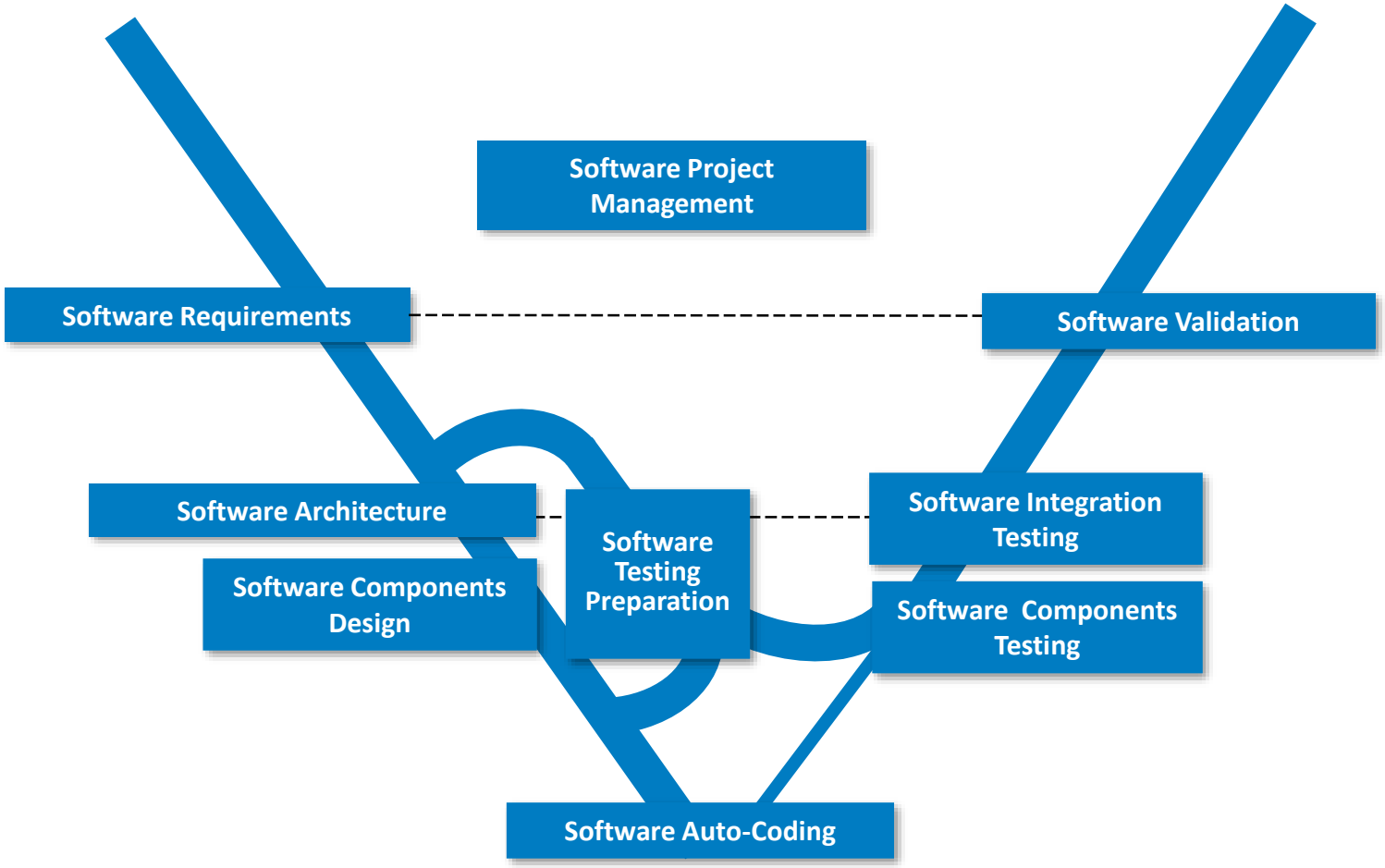
Cycle	1	2	3	4	…
3	3	3	3	3	…
a	a1	a2	a3	a4	...
b	b1	b2	b3	b4	...
a+b	a1+b1	a2+b2	a3+b3	a4+b4	…
pre(a)	nil	a1	a2	a3	...

© 2016 ANSYS, Inc.

How to implement algorithms?

How to implement computations and equations between the system data?

SCADE V-Cycle



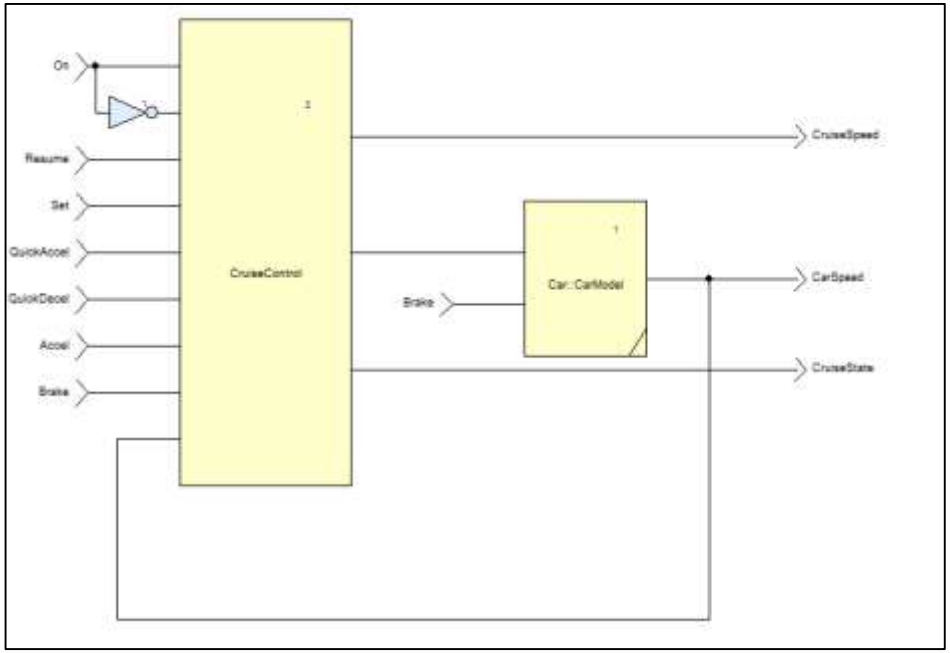
© 2016 ANSYS, Inc.

Software Architecture

Once the software requirements phase is achieved, software engineers define the software architecture before implementing the design.

A software architecture in a SCADE Suite model is a network of top level user-defined operators drawn as boxes.

Software architecture can be done with SCADE Architect, bundled with SCADE Suite



© 2016 ANSYS, Inc.

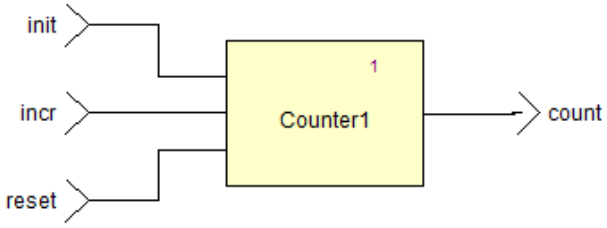
Software Design

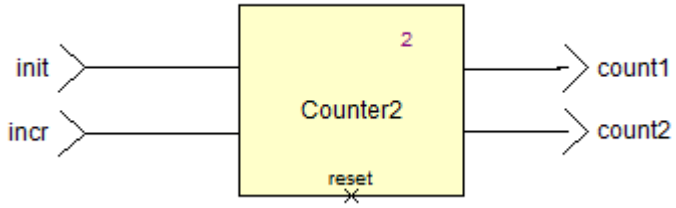
A SCADE Suite design of a software is composed of

- A network of user-defined operators and
- Predefined operators such as '+', 'and'

Operators may be purely combinatorial, or may have memory (like a counter or an integrator)

- Independent operators data are not ordered and compute in parallel
- Operators only communicate through the flows





© 2016 ANSYS, Inc.

User Operator

Set of equations:

- Graphical, State Machine or textual operator
- One definition for each output and each local variable
- Indifferent order in equations

Modularity / Reuse:

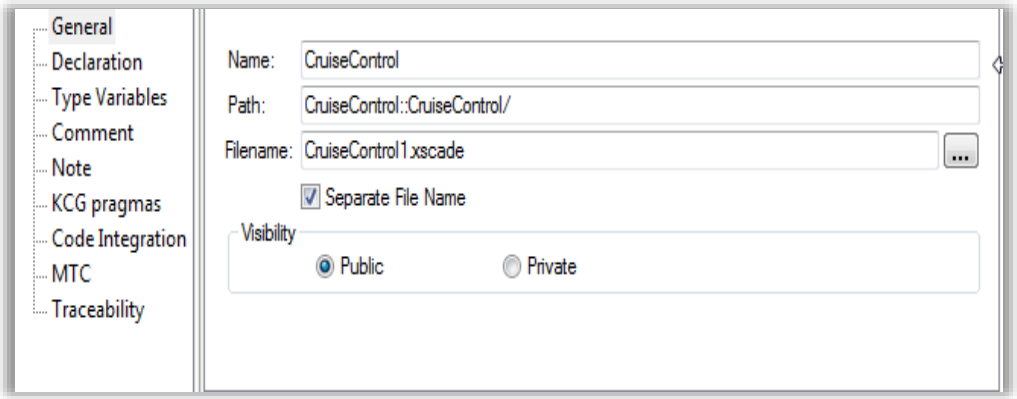
- An operator is a black box and is connected through its formal interface
- Each operator defined by the user can be reused

© 2016 ANSYS, Inc.

Operator Creation

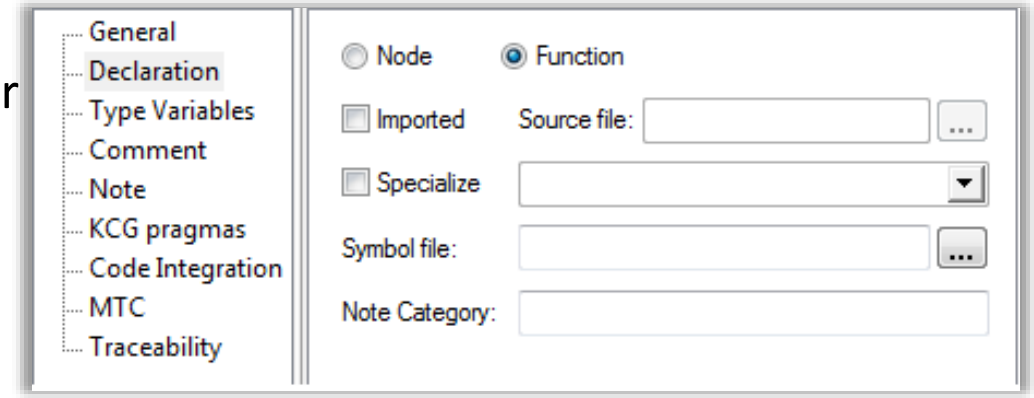
Click on:  (*Shortcut*: *Ctrl+Shift+N*)

Select the created operator
in the Scade view



Give a meaningful name to:

- The operator
- The file where the operator is saved if separated



© 2016 ANSYS, Inc.

Operator Creation

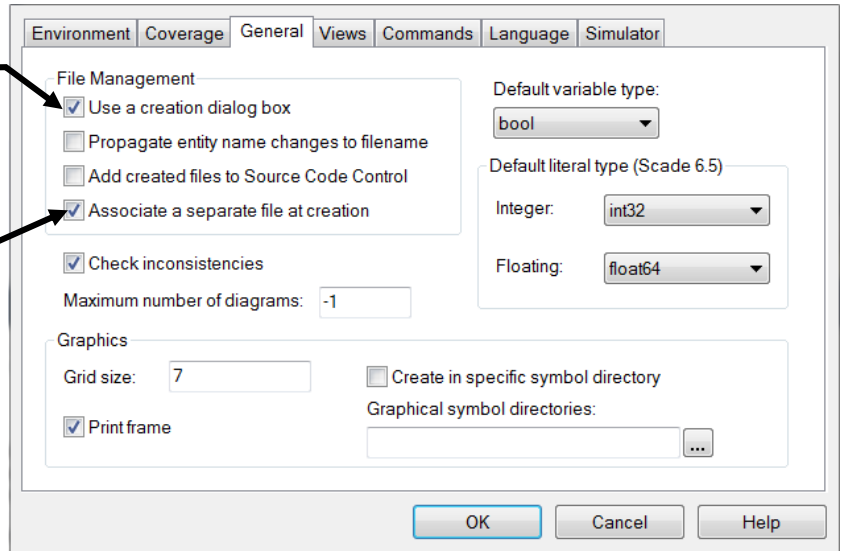
It is possible to set global options to:

- Directly set the name of the operator and its file using a creation dialog box
- Propagate changes on the operator name to the file name

Open *Tools > Options > General tab*

Select to have a dialog box entry to edit the operator's name

Select to propagate the operator's name to the underlying file name



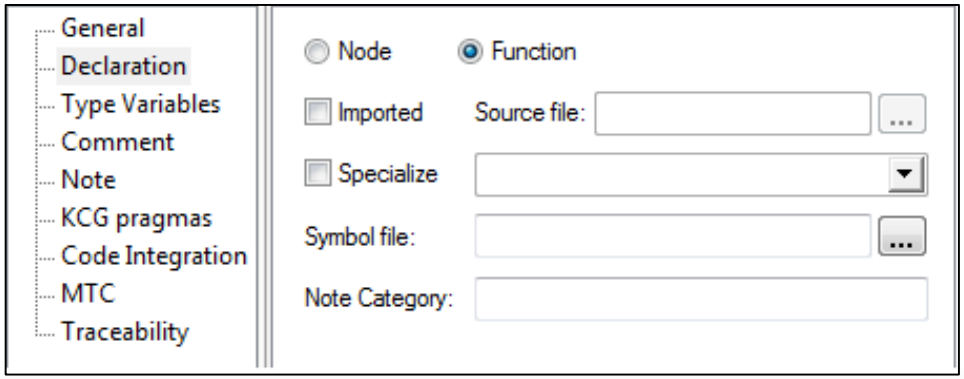
© 2016 ANSYS, Inc.

Functions/Nodes

By default, an operator is declared as a node (operator with internal memory)

A function is a stateless operator:

- Contains no internal memory (State machine, last, pre, fby, etc. do have memory!)
- Enables function specific code generator optimizations
- Setting accessed via the property dialog:

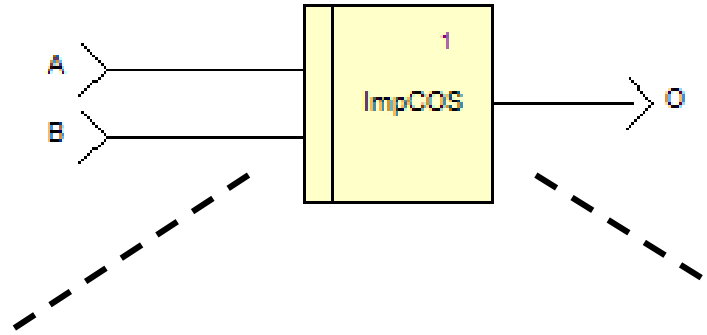


© 2016 ANSYS, Inc.

Imported Operators (C Code)

The aim of an imported operator is to use external code:

- The operator's name and interface are defined in SCADE Suite, but not the operator's body
- The implementation is provided through the target code




```
/*********************************************/
/*      My function                          */
/*********************************************/
#include "kcg_types.h"
#include <math.h>
kcg_float32 ImpCOS(kcg_float32 A, kcg_float32 B)
{
   return cos (A + B);
}
```

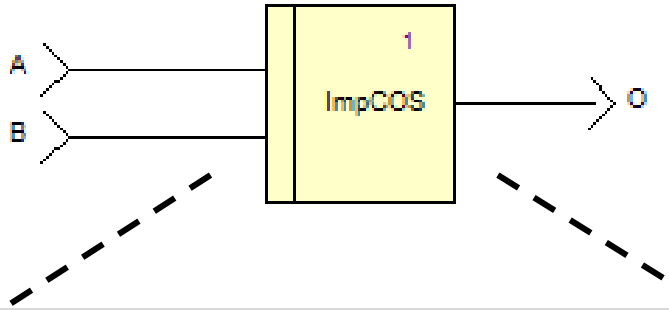

© 2016 ANSYS, Inc.

Imported Operators (Ada)

Ada

The aim of an imported operator is to use external code:

- The operator's name and interface are defined in SCADE Suite, but not the operator's body
- The implementation is provided through the target code



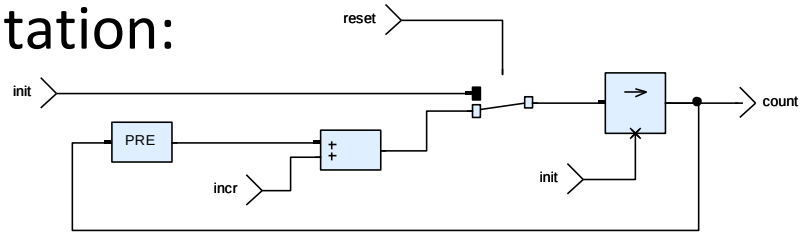

```
function ImpCOS(
  A : in Kcg_Config.Kcg_Float32;
  B : in Kcg_Config.Kcg_Float32) return
Kcg_Config.Kcg_Float32
is
  O : Kcg_Config.Kcg_Float32;
begin
  O:= cos (A + B);
  return O;
end ImpCOS;
```


© 2016 ANSYS, Inc.

Operator Interface

Graphical SCADE Suite representation:

- An operator is a block diagram

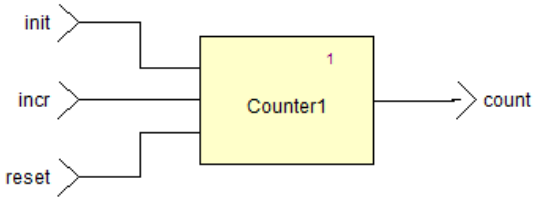


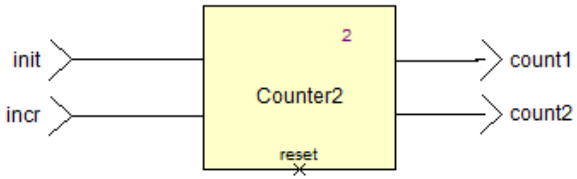
An operator is seen as a black box and is connected through its formal I/O interface

Textual Scade representation:

```
node Counter1(init:int32; incr:int32; reset:bool)
returns (count:int32)

node Counter2(init:int32; incr:int32; reset:bool)
returns (count1:int32; count2:int32)
```





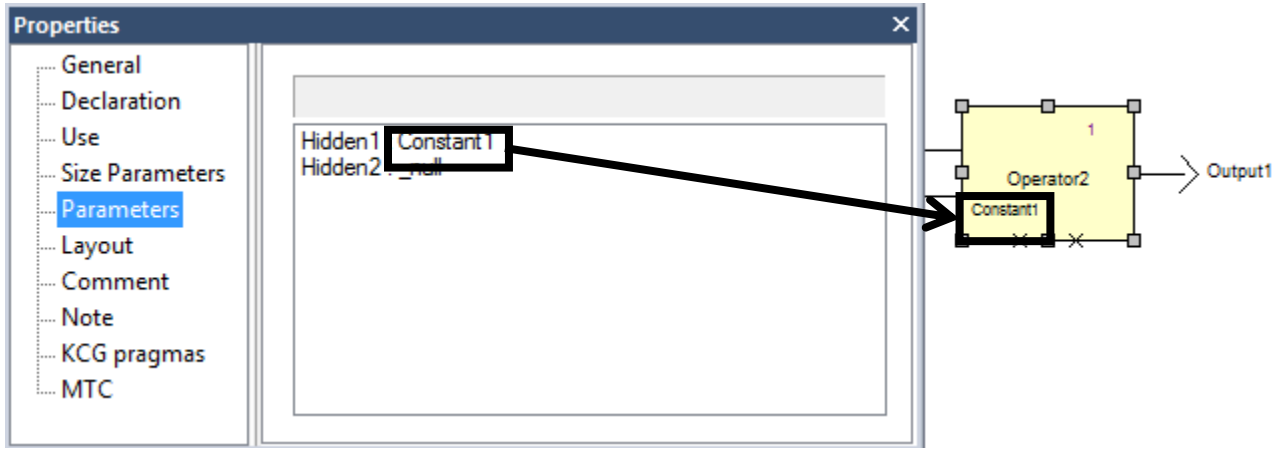
© 2016 ANSYS, Inc.

Hidden Inputs

Goals:

- Reduces the number of links and increases readability
- Allows to distinguish between parameters and data flows
- Makes a graphical difference when the operator is instantiated

Direct assignment by connection or by name:



From a code generation point of view, no difference between a hidden input and an input

© 2016 ANSYS, Inc.

Operator Interface Creation

Input: *(Shortcut: Ctrl+Shift+I)*

Output: *(Shortcut: Ctrl+Shift+O)*

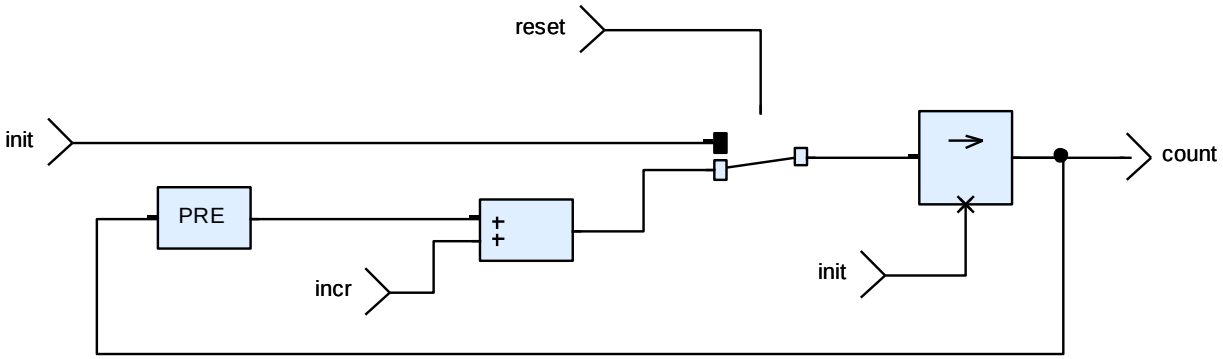
Hidden input: *(Shortcut: Ctrl+Shift+H)*

- Select the created input/output
- Give a meaningful name
- Select the type of the input or output

 © 2016 ANSYS, Inc.

Operator Body

A block diagram example:



A textual diagram example:

```
count = init -> if reset then init else pre(count) + incr;
```


© 2016 ANSYS, Inc.

Local Variables

Seen only by the operator/conditional block/state in which it is declared:

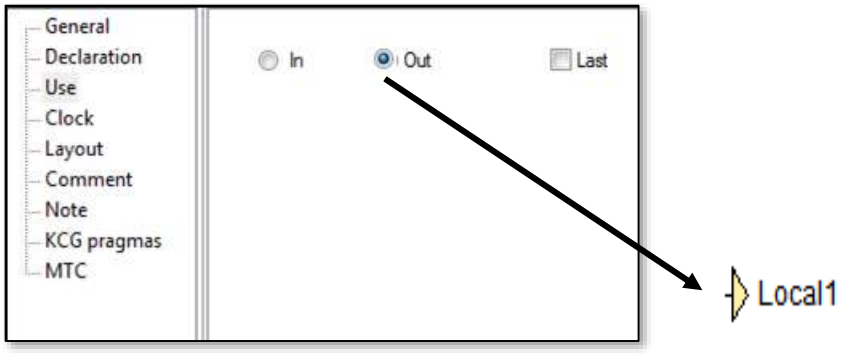
- Must be defined only once
- Can be read several times

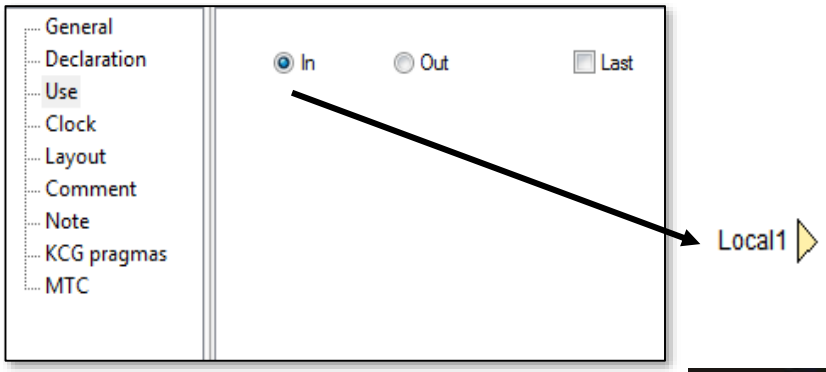
To produce it :

- Open its Properties window
- Click under Use tab on Out
- Or press on CTRL + Drag&Drop it from its definition

To use it :

- Open its Properties window
- Click under Use tab on In





© 2016 ANSYS, Inc.

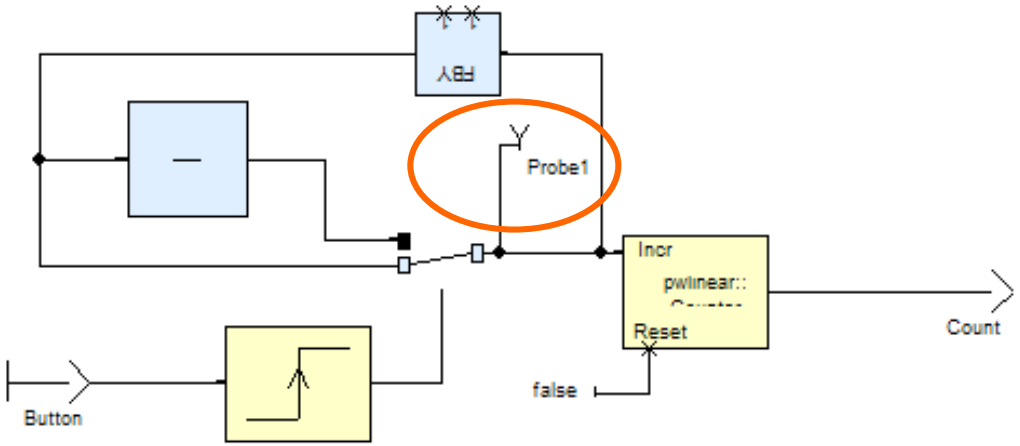
Probe

Defined an observation point of internal variable

Can be accessed during simulation or execution of the generated code

Created by clicking on button 

Enhance observation by allowing the watch of any internal flow during simulation or execution of the generated code



© 2016 ANSYS, Inc.

Sensor

Global variable accessible in read-only mode

Assumed to be provided by the environment

Remains constant during the whole execution cycle

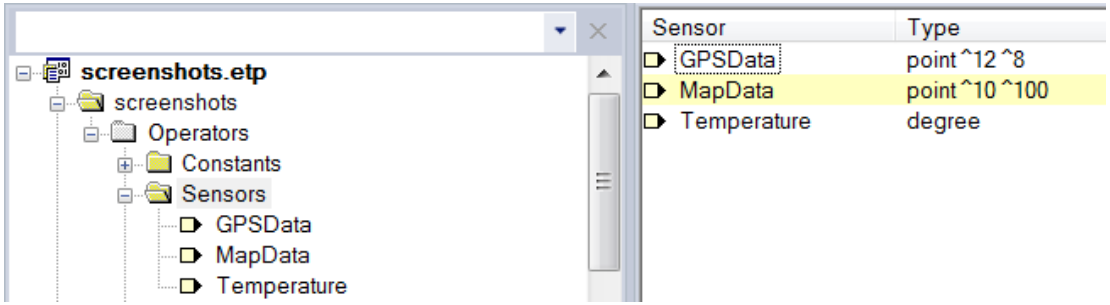
Provides an alternative to the operator input interface, intended for data that must be accessed throughout the design

© 2016 ANSYS, Inc.

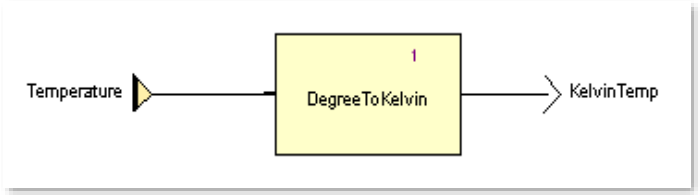
Creating and Using Sensors

Select a package/project where the sensor must be created

- Create a sensor by clicking on: 
- Give it a name and select the type of the sensor



Use a sensor by drag and drop on a diagram



© 2016 ANSYS, Inc.

Lab 2

Lab Support p.23-24

Objective:

Define the architecture operators

Time: 10 min

Requirements:

Create the 4 following operators (and packages)

Operators	Package	Comments
`CruiseRegulation`	`CruiseControl`	manages the vehicle speed when the Cruise Control is active
`CruiseSpeedMgt`	`CruiseControl`	manages the value of the cruise speed according to the driver commands
`CruiseControl`	`CruiseControl`	manages the cruise control behavior
`SystemSimul`	`System`	connects the other operators and the Car model

Lab

© 2016 ANSYS, Inc.

Scade Predefined Operators

Basic operators of the Scade language

Can be used either in textual or graphical design

- Graphically, they have a blue fill color to quickly identify them

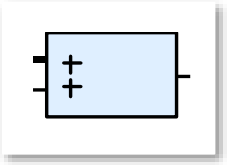
Fully documented in the Scade Language Reference Manual

Grouped by functionality:

- Mathematical
- Logical
- Bitwise
- Time
- Comparison
- Choice
- Structure/arrays
- Higher order: activation constructs

© 2016 ANSYS, Inc.

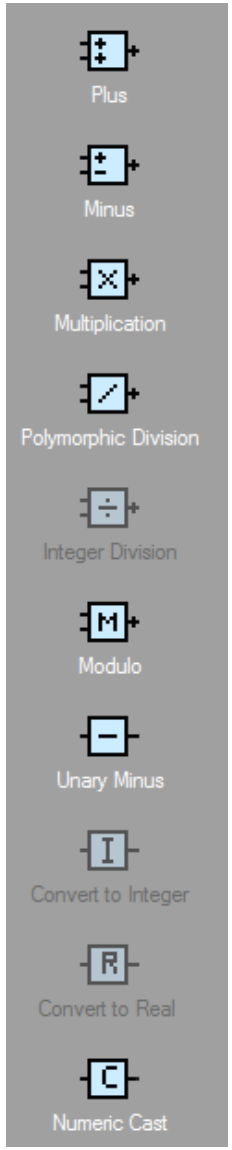
Mathematical Operators



They apply to numeric types

- Type interface: 'N or 'N, 'N → 'N, where N is Numeric (float or integer)
- Example

Cycle	1	2	3	4
a	a1	a2	a3	a4
b	b1	b2	b3	b4
c = a+b	a1+b1	a2+b2	a3+b3	a4+b4

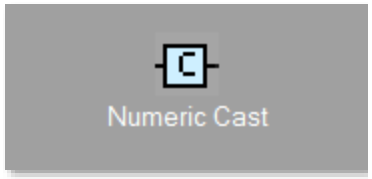


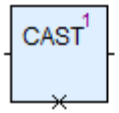
© 2016 ANSYS, Inc.

Cast

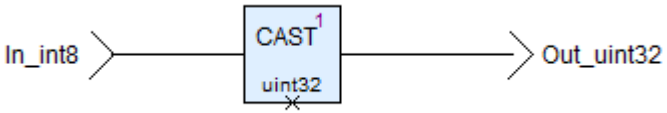
A **cast** operator allows the conversion of any data into an expression of the proper numeric type:

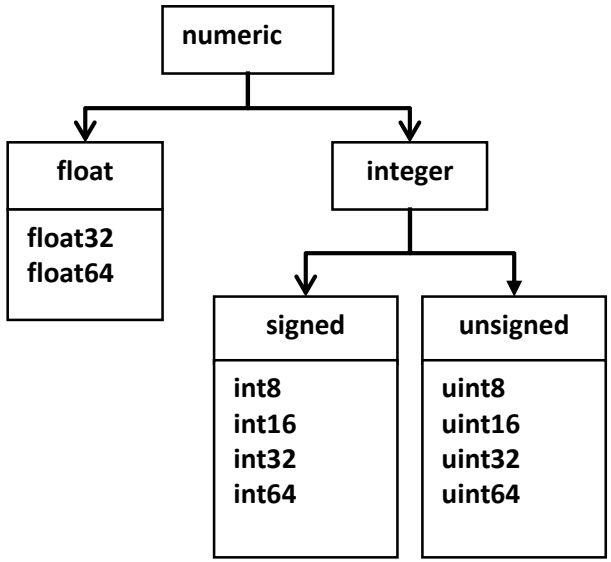
- (a + b: uint16): conversion to uint16
- (4: int8): specification of a literal





o1 = (i1 : **type**)

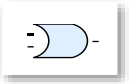




© 2016 ANSYS, Inc.

Logical Operators

They apply to the bool type:

- Type interface: bool or bool, bool → bool
- Note: #(e1,...,en) is true if zero or one of the ei is true
- Example 

Cycle	1	2	3	4
a	a1	a2	a3	a4
b	b1	b2	b3	b4
C = a or b	a1orb1	a2orb2	a3orb3	a4orb4

© 2016 ANSYS, Inc.

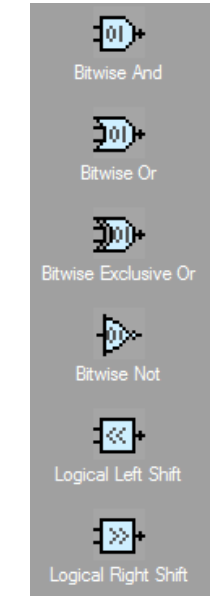
Bitwise Operators

They apply to unsigned integer types

- Type interface:

unsigned integer, unsigned integer→ unsigned integer

- Example: 



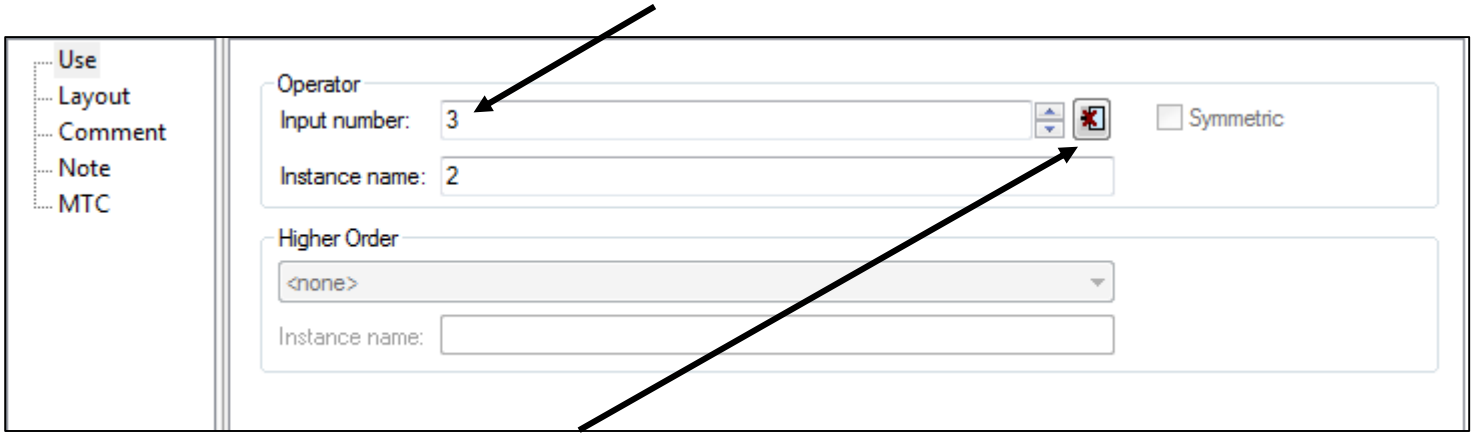
Cycle	1	2	3	4
a	a1	a2	a3	a4
b	b1	b2	b3	b4
C = a & b	a1&b1	a2&b2	a3&b3	a4&b4

© 2016 ANSYS, Inc.

Add New Input for Predefined Operators

Select a predefined operator in the diagram view:

- From the Properties Use tab:
  - Add new input pins:
    - Use “Input number” selector
    - Enter the number of inputs directly in the field

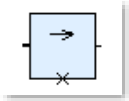


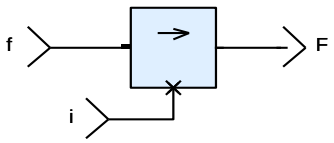
    - Remove input pins:

© 2016 ANSYS, Inc.

Time Operators: Initialization ->

Initializes flows

- Type interface: 'T, 'T → 'T, where 'T is any Type
- Example: 

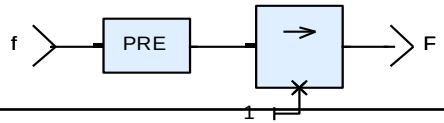
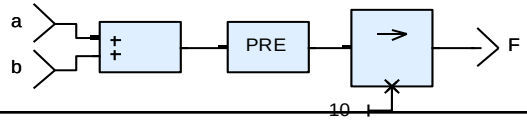
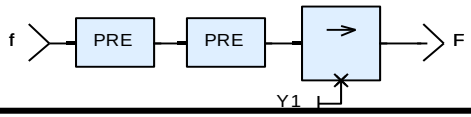
Cycle	1	2	3	4
`i`	`i1`	`i2`	`i3`	`i4`
`f`	`f1`	`f2`	`f3`	`f4`
`i -> f` 	`i1`	`f2`	`f3`	`f4`

© 2016 ANSYS, Inc.

Time Operators: pre

Produces the previous value of its input

- Type interface: 'T → 'T, where 'T is any Type
- At the first cycle, the previous value is not defined
- The Initialization operator ensures the complete definition
- Examples: 

f	f1	f2	f3	f4
pre(f) 	nil	f1	f2	f3
1 -> pre(f) 	1	f1	f2	f3
10 -> pre(a+b) 	10	a1+b1	a2+b2	a3+b3
y -> pre(pre(f)) 	Y1	nil	f1	f2

© 2016 ANSYS, Inc.

Exercise 1: Rising Edge

Exercise

Objective

Implement a rising edge operator using `pre/init` primitives

Requirements

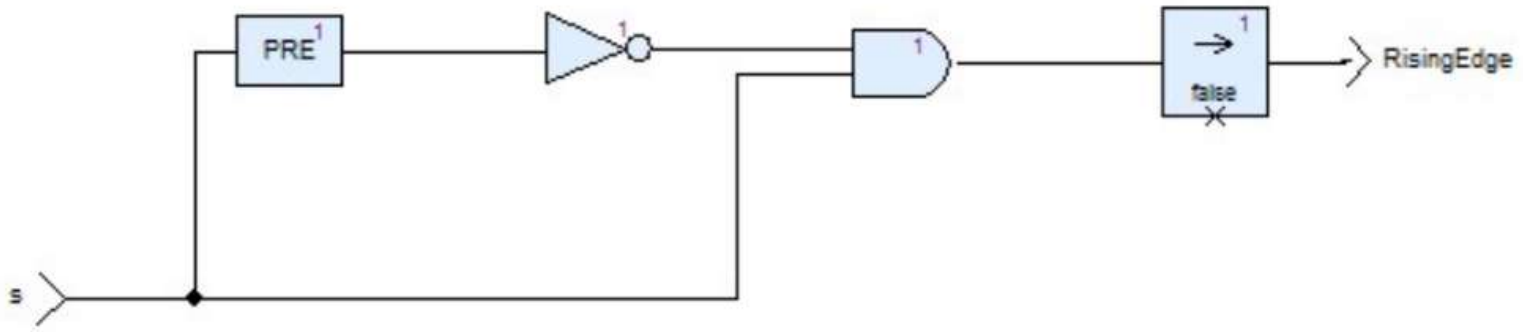
The `RisingEdge` operator take boolean input `s` and output a boolean `R` according to:

```
if (s_t and s_{t-1})
        R_t = true
else
        R_t = false
```


© 2016 ANSYS, Inc.

Exercice 1: Solution

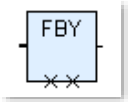
Exercise

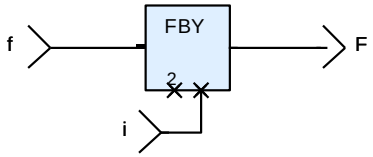


© 2016 ANSYS, Inc.

Time Operators: fby

Delays its inputs by a certain number of cycles while providing a default value:

- Type interface: 'T, int, 'T → 'T, where T is any type
- Introduces a delay
- The number of past cycles must be a strictly positive integer value
- Example: 

f	f1	f2	f3	f4	f5
fby(f;2;i) 	i1	i1	f1	f2	f3

- fby(f;n;i) is equivalent to :

  i -> pre(i -> pre(... -> pre(f)))

© 2016 ANSYS, Inc.

Exercise 2: Rising Edge

Exercise

Objective

Implement a rising edge operator using `fby` primitives

Requirements

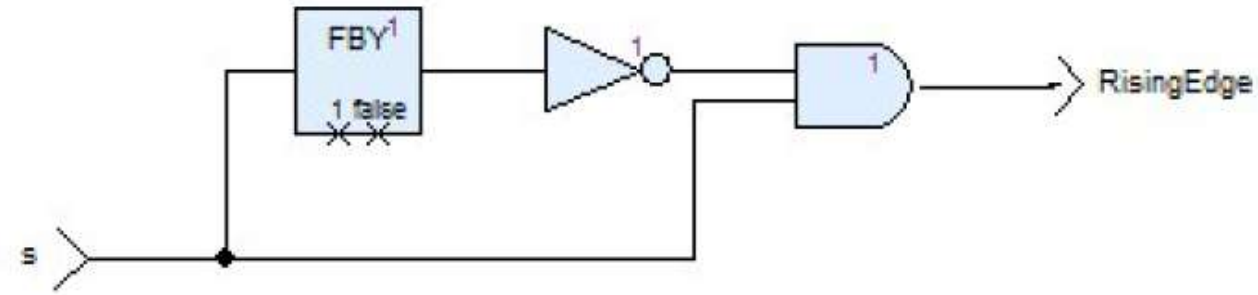
The `RisingEdge` operator take boolean input `s` and output a boolean `R` according to:

```
if (s_t and s_t-1)
        R_t= true
else
        R_t= false
```


© 2016 ANSYS, Inc.

ANSYS

Exercice 2: Solution

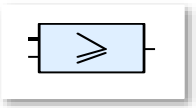


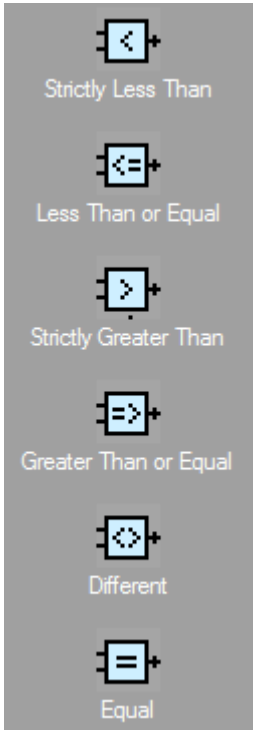
Exercise

© 2016 ANSYS, Inc.

Comparison Operators

They apply to integer or real types

- Type interface: ‘N, ‘N → bool, where ‘N is a numeric type
- Example 



Cycle	1	2	3	4
a	-145	-1	154	-1456
b	-1	-2	154	1785
c = a >= b	false	true	true	false

© 2016 ANSYS, Inc.

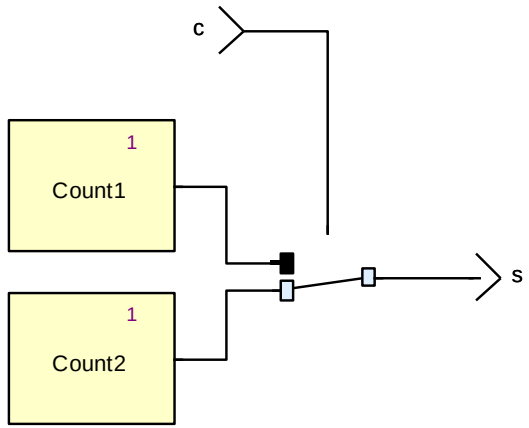
Choice Operators: if-then-else (Reminder)

Receives two flows and outputs one of them depending on a Boolean condition

Both “then” and “else” expressions are always evaluated independently of the condition value.

Example:

- Consider two counter operators count1 and count2
- s = if c then Count1() else Count2();
  - Each counter is incremented at each cycle regardless of c
  - This behaviour complies with the intuition behind the reading of the graphical representation

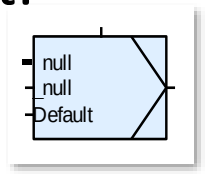


© 2016 ANSYS, Inc.

Choice Operators: case

Receives 2N+1 inputs and produces one output:

- N inputs Vi, candidate value for the output
- N constant inputs, one per Vi plus a default case
- 1 input to be tested against the constants



All the inputs are computed before the choice

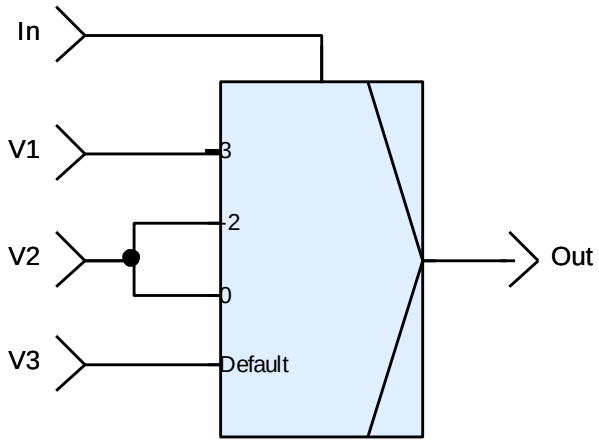
Example:

- Out = case In of

  3 : (V1)
  -2 : (V2)
  0 : (V2)
  default : (V3);

- Equivalent to:

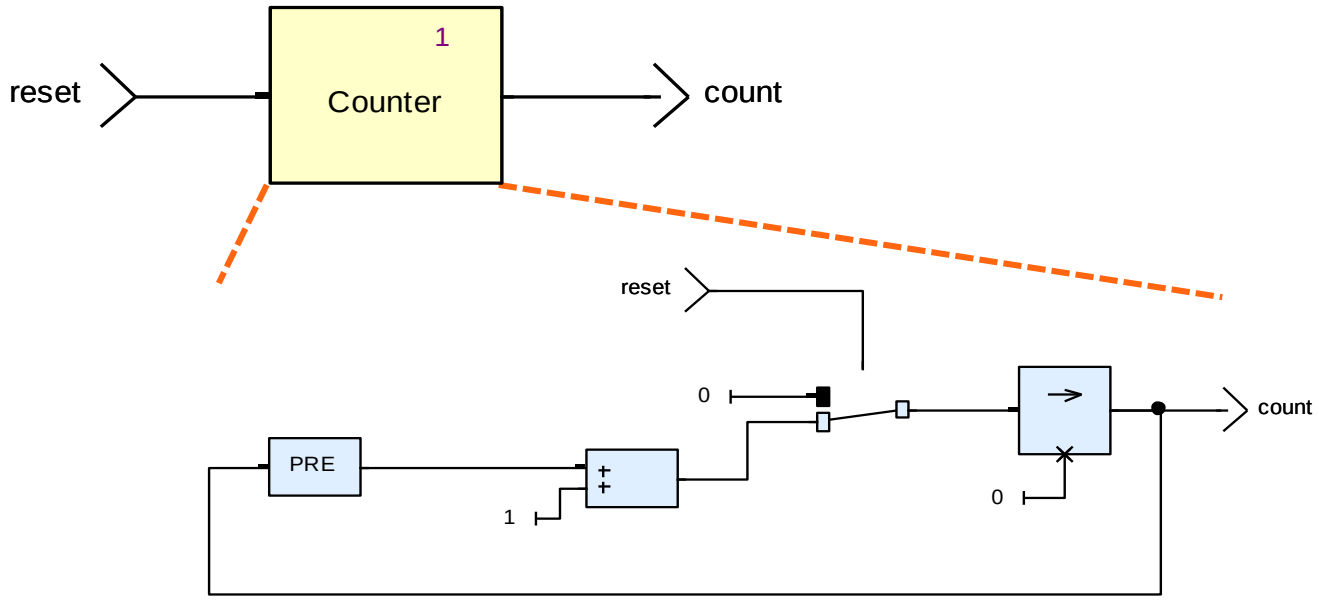
  Out = if In=3 then V1
  else (if In=-2 or 0 then V2
  else V3)



© 2016 ANSYS, Inc.

Choice Operators Example: Simple Counter

Example




```
node Counter (reset: bool) returns (count: int32)
count = 0 -> if reset then 0 else PRE(count) + 1;
```

Cycle	1	2	3	4	5
count (reset =false)	0	1	2	3	4

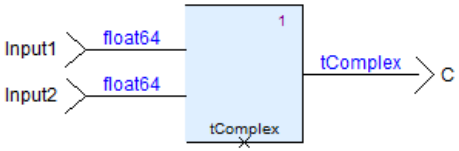
© 2016 ANSYS, Inc.

Structured Type Operators

Operators on structures:

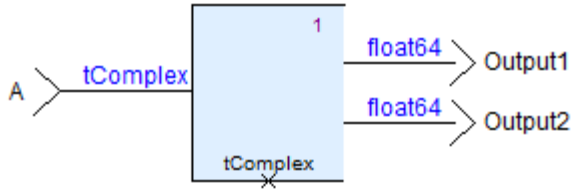
Make

- CTRL+Drag&Drop on type
- Compose on output
- Properties/Parameters on Make

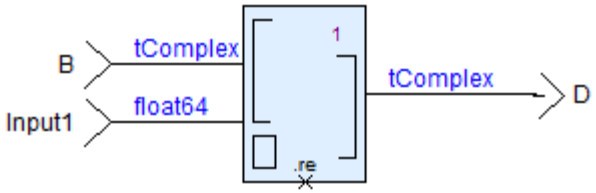


Flatten

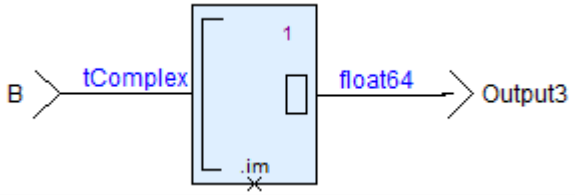
- Drag&Drop on type
- Decompose on input(s)
- Properties/Parameters on Flatten



New Assign of a Structured Element



New Projection (select a field)



© 2016 ANSYS, Inc.

Higher order: Activation Constructs

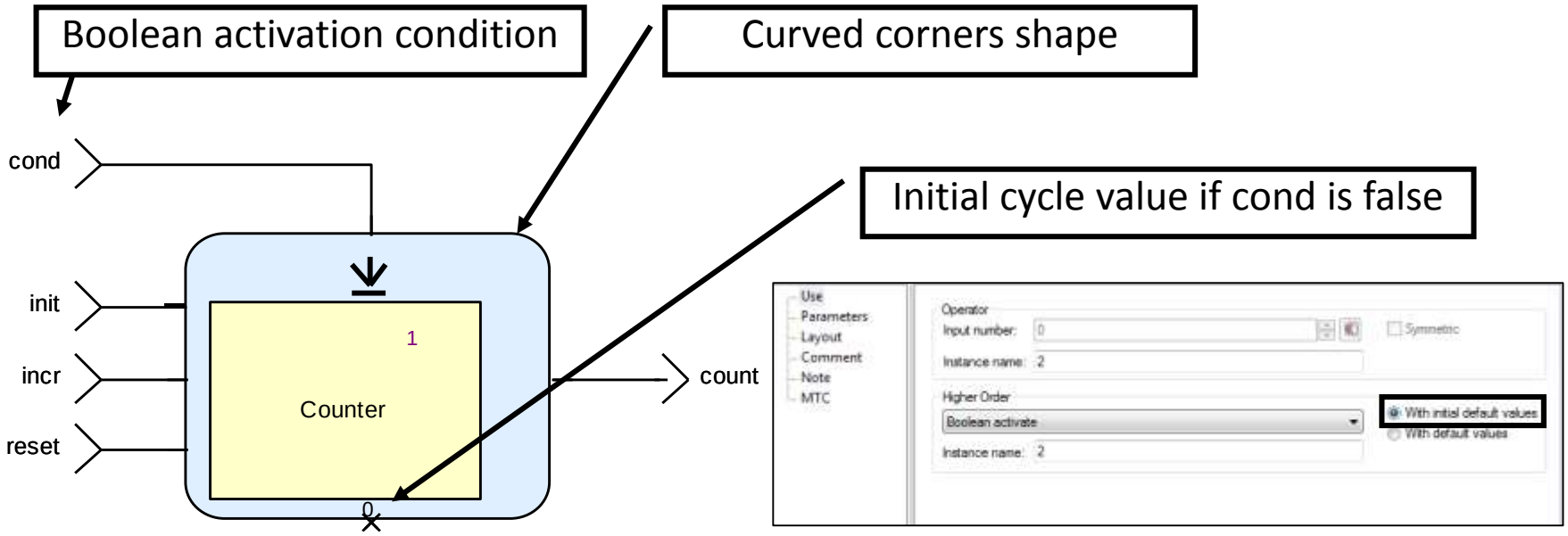
SCADE Suite provides constructs to control the activation of operators:

- Boolean Activate
- Restart

© 2016 ANSYS, Inc.

Boolean Activate Operators: Memorized Outputs

Operator outputs are memorized when the activation condition is false




```
count =
(activate Counter every cond initial default (0))(reset, init, incr);
```


© 2016 ANSYS, Inc.

Exercise 3: Memorized Output

Exercise

Objective

Implement boolean activate higher order

Requirements

Create an operator which

```
if enable = true
```

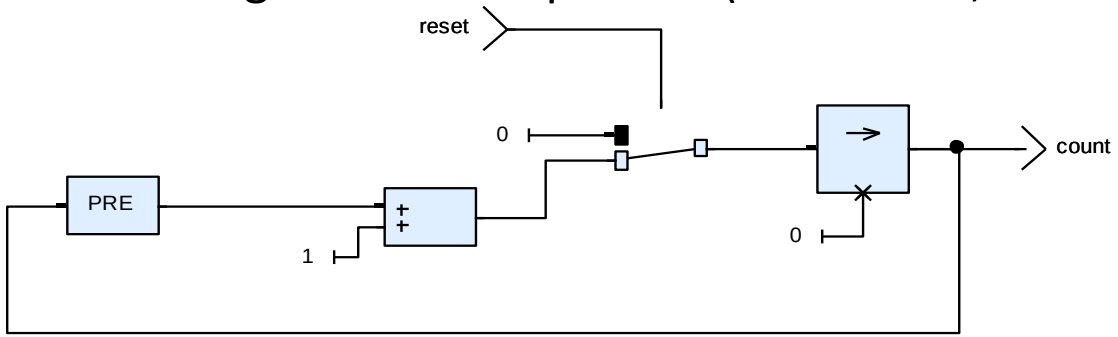
$count_t = count_{t-1} + 1$

```
Else
```

$count_t = count_{t-1}$

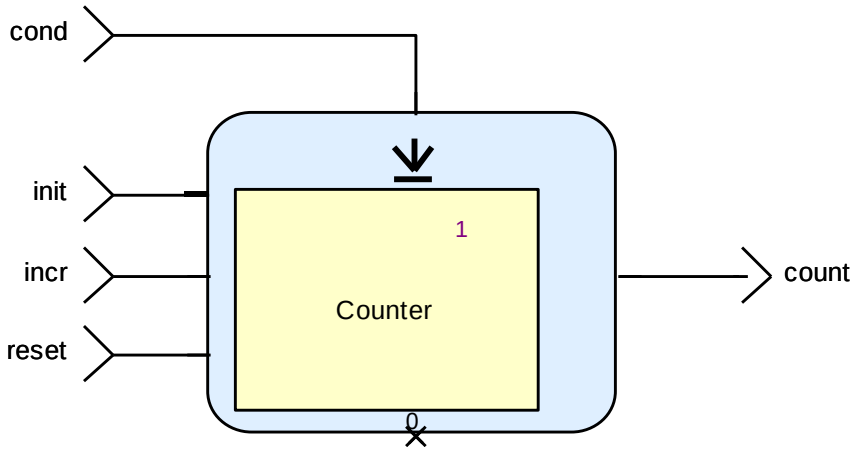
Name	Kind	Type
enable	input	bool
count	output	uint16

Tip: create the following `Counter` operator (reset: bool, count: uint16)



© 2016 ANSYS, Inc.

Exercise 3: Solution



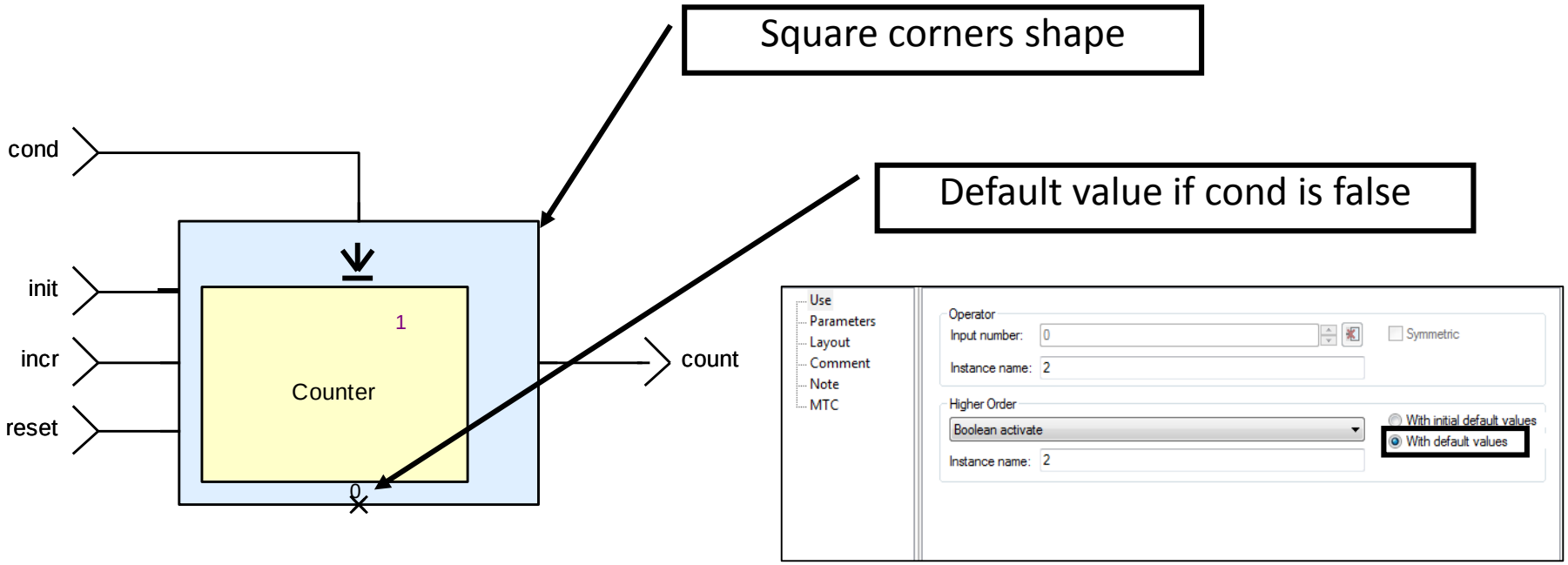
cond	true	false	true	false	false
Activate Counter	count1= Counter()	count1	count3= Counter()	count3	count3

cond	false	false	true	false	false
Activate Counter	0	0	count3= Counter()	count3	count3

© 2016 ANSYS, Inc. February 1, 2017 ANSYS Confidential

Boolean Activate Operators: Default Outputs

Operator outputs have a default value when the activation condition is false




```
count = (activate Counter every cond default (0))(reset, init, incr);
```


© 2016 ANSYS, Inc.

Exercise 4: Default Output

Exercise

Objective

Implement boolean activate higher order

Requirements

Create an operator which

```
if enable = true
```

$count_t = count_{t-1} + 1$

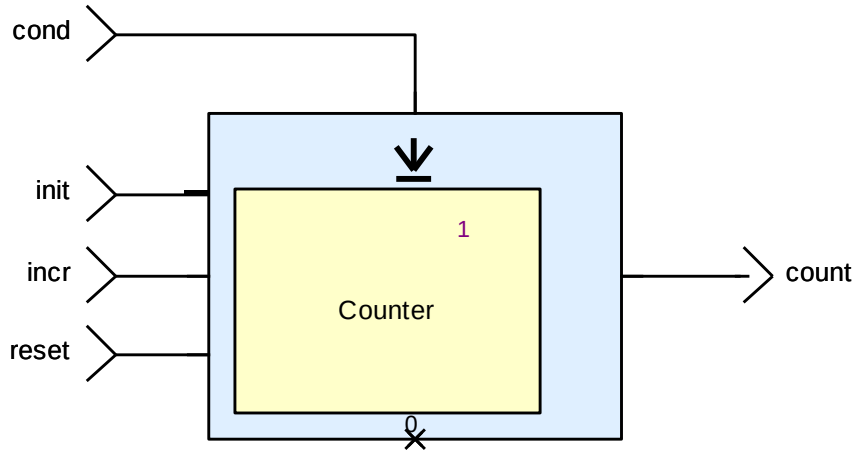
```
Else
```

$count_t = 0$

Name	Kind	Type
enable	input	bool
count	output	uint16

© 2016 ANSYS, Inc.

Exercise 4: Solution



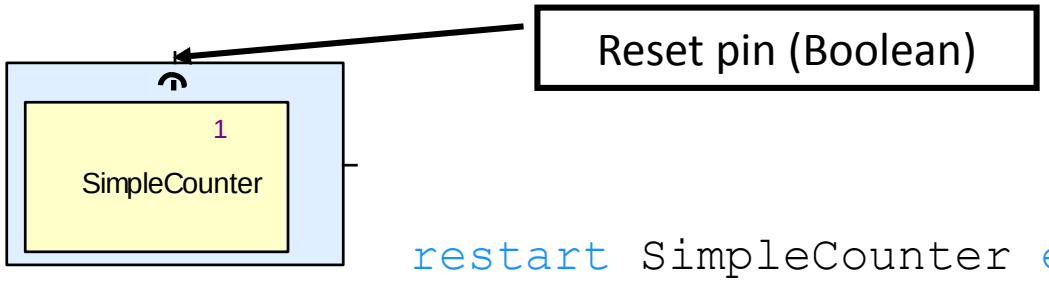
cond	true	false	true	false	false
activate Counter	count1= Counter()	0	count3= Counter()	0	0

cond	false	false	true	false	false
activate Counter	0	0	count3= Counter()	0	0

© 2016 ANSYS, Inc. February 1, 2017 ANSYS Confidential

Restart Operator

The restart operator applies on operators




```
restart SimpleCounter every…
```

When activated, resets the operator to its initial cycle

- Applies on the whole sub-hierarchy of the operator

The effect is the resetting of the memories of the operator, as Init, pre/Fby, State machines, …

© 2016 ANSYS, Inc.

Lab 3 (1/4)

Lab Support p.26-47

Objective:

Design the `CruiseRegulation` operator

Requirements:

Time: 30 min

Implement the `CruiseRegulation` function

(requirements CC_HLR_CDC_01 to 05)

The `CruiseRegulation` operator manages automatically the vehicle speed when the Cruise Control is on: the regulation is reset when the Cruise Control is on, and frozen when the throttle output is saturated

Lab

© 2016 ANSYS, Inc.

Lab 3 (2/4)

Create the `SaturateThrottle` operator *(requirement CC_HLR_CDC_05)*

Name	Kind	Type
throttleIn	input	CarType::tPercent
throttleOut	output	CarType::tPercent
saturate	output	bool

```
A = if (throttleIn < ZERO_PERCENT)
       then ZERO_PERCENT
       else throttleIn;
B = if (throttleIn > REGUL_THROTTLE_MAX)
       then REGUL_THROTTLE_MAX
       else A;
throttleOut = B;
```

Lab

© 2016 ANSYS, Inc.

Lab 3 (3/4)

Create the `CruiseRegulation` operator *(requirement CC_HLR_CDC_01 to 05)*

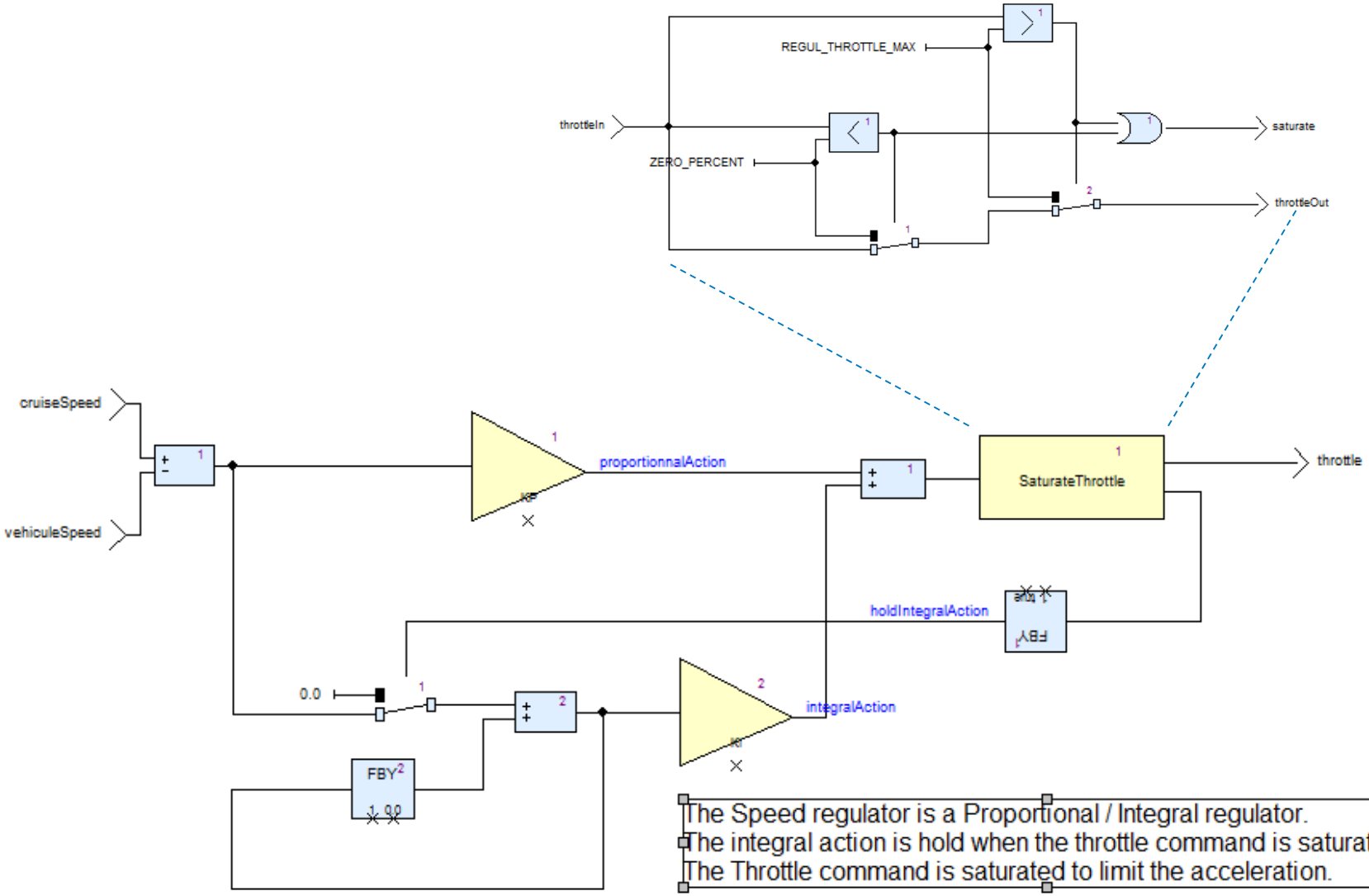
Name	Kind	Type
cruiseSpeed	input	CarType::tSpeed
vehicleSpeed	input	CarType::tSpeed
throttle	output	CarType::tPercent

- `Throttle= Kp*(CruiseSpeed_t - VehicleSpeed_t) + ∑_0^t (Ki*(CruiseSpeed_t - VehicleSpeed_t)`
- Saturate the throttle with `SaturateThrottle` operator
- `Saturate = (ThrottleIn > REGUL_THROTTLE_MAX) or (ThrottleIn < ZERO_PERCENT)`
- `if saturate then Integral = 0.0`
  `else Integral = cruiseSpeed_t - vehicleSpeed_t`
- `IntegralAction =` $\sum_0^t \mathrm{Ki}(\mathrm{CruiseSpeed\_t} - \mathrm{VehicleSpeed\_t})$

Lab

© 2016 ANSYS, Inc.

Lab 3 (4/4): CruiseRegulation



© 2016 ANSYS, Inc.

AGENDA

Type declaration

Constant declaration

Operator and interface declaration

Control flow declaration

© 2016 ANSYS, Inc.

How to implement decision logic of reactive systems?

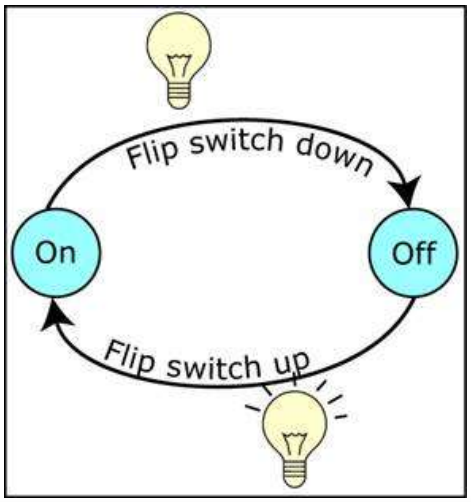
How to implement discrete behaviors of reactive systems?

Control Flows

Control flows are flows modelling discrete behaviors or decision logic of reactive systems

In SCADE Suite, find the following control flow constructs:

- Conditional blocks: If Block, When Block
- State machines:



© 2016 ANSYS, Inc.

When Do You Use a Conditional Block?

A Conditional Block implements control flows to perform a test and then take action based on the test result

Because it allows a program to take a different path depending on some condition(s)

A Conditional Block has no memory so

- The generated code does not contain memories
- Use it when the conditions do not depend on the past and do not depend on the previous mode

© 2016 ANSYS, Inc.

Applications for Conditional Blocks

Example

Flight-control, cockpit displays

- Modes, and user interaction control (buttons, navigation controls, etc.)

Train speed control, interlocking, traffic lights, cruise control

- Modes, and user interaction control

Vending machines, Automatic Teller Machine, elevators, etc.

- Modes and user interaction control

© 2016 ANSYS, Inc.

Applications for Conditional Blocks

Example

Used to implement the cruise speed management:

- CruiseSpeedMgt operator manages the value of the cruise speed according to the driver commands
- Conditional Block is used to fix, increase, reduce speed when the associated button is pressed

Requirements:

CC_HLR_CSM_01 to CC_HLR_CSM_05

- Set or QuickAcccel or QuickDecel buttons pressed when the CruiseControl is enabled

© 2016 ANSYS, Inc.

When Do You Use a State Machine?

A State Machine implements control flows to perform a predetermined sequence of actions depending on a sequence of events or several conditions

Because

- it is in only one state at a time among a finite number of states
- it can change from one state to another when initiated by a triggering event or condition

A State Machine has memory so

- The generated code contains always memories
- Use it when the events or conditions depend on the past or the previous mode

© 2016 ANSYS, Inc.

Applications for State Machines

Example

Flight-control, cockpit displays

- Display modes, user interaction control

Train speed control, interlocking, traffic lights

- Modes, current states and transition logic

Vending machines, Automatic Teller Machine, elevators, cruise control, etc.

- Modes, user interactions
- Hierarchy for the organization modes
- Concurrent events and behavior
- Synchronization of events
- Preemptive events

ANSYS

© 2016 ANSYS, Inc.

Applications for State Machines

Example

Used to implement the Cruise Control behavior:

- A State Machine is implemented to define the different states, ON, OFF, STDBY, etc.

Requirements:

CC_HLR_CCB_01

- When the driver starts the car, the CruiseControl shall be off
- The CruiseState output shall be set to OFF

CC_HLR_CCB_02

- The CruiseControl shall be set on when the driver presses the on button

CC_HLR_CCB_05

- The CruiseControl system shall be automatically disabled when the accelerator pedal is pressed, or when the car speed is outside the speed limit
- The CruiseState output shall be set to STDBY

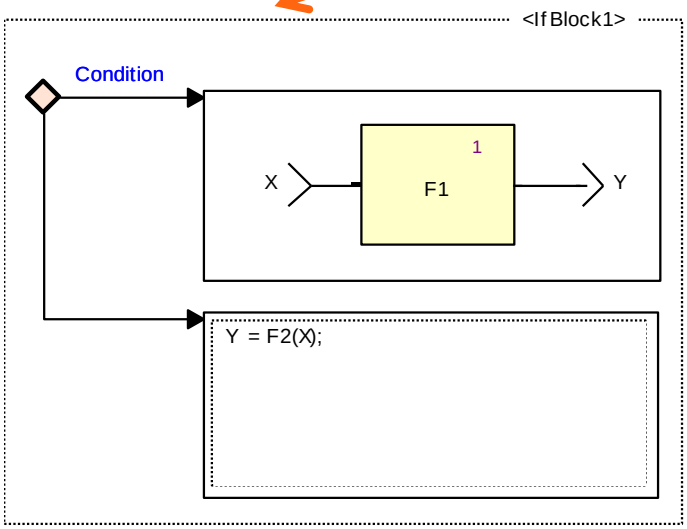
© 2016 ANSYS, Inc.

Conditional Blocks

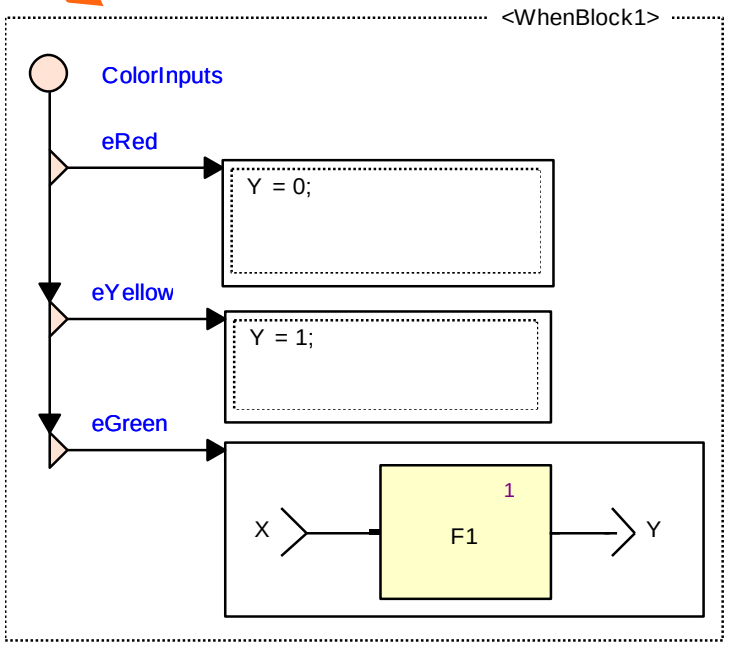
If and When Blocks are control flow constructs

Their activation is a function of conditions computable in the current cycle

Boolean



Enumeration

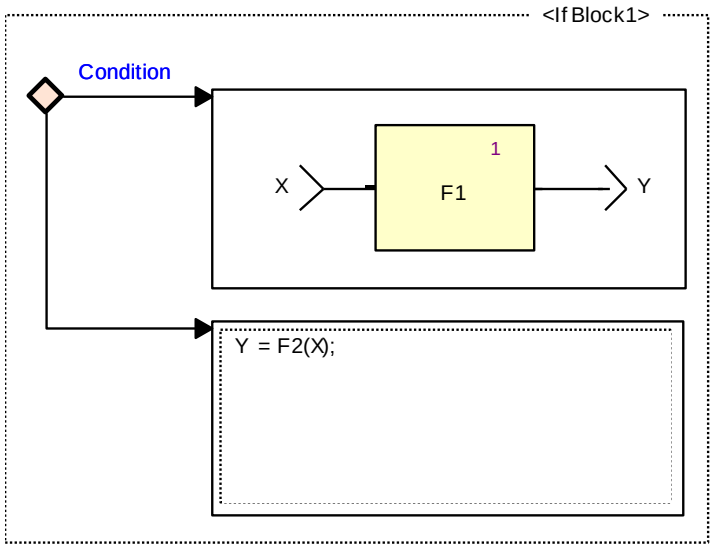


© 2016 ANSYS, Inc.

Conditional Blocks: If Block

The control blocks implement any SCADE Suite equations, graphical or textual, including calls of user operators

If Block (activate if)



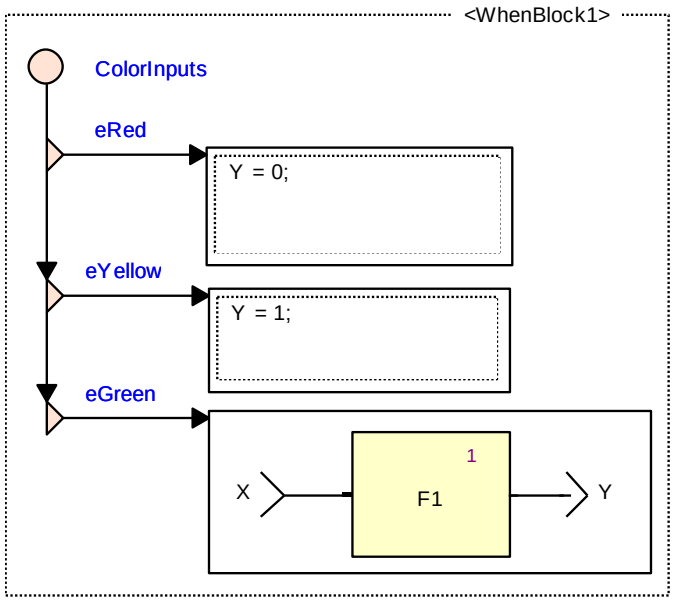

```
activate <IfBlock1> if
Condition
then
   Y = F1(X);
else
   Y = F2(X);
returns Y;
```


© 2016 ANSYS, Inc.

Conditional Blocks: When Block

Relies on an enumeration type

When Block (`activate when match`)




```
activate <WhenBlock1> when
ColorInputs match
| eRed : Y = 0;
| eYellow: Y = 1;
| eGreen : Y = F(X);
returns Y;
```


© 2016 ANSYS, Inc.

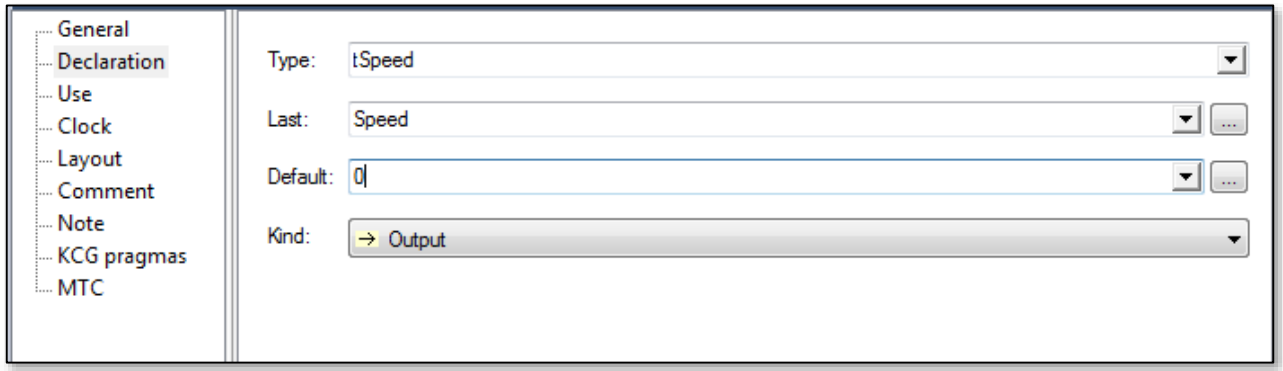
Conditional Block: Data Flow Inside Block

Equations are calculated only when the block is active like an imperative-If and different from the If-Scade

Each declared flow (local or output) must have exactly one definition for each cycle where its block is active

What happens when a definition is missing in a given block?

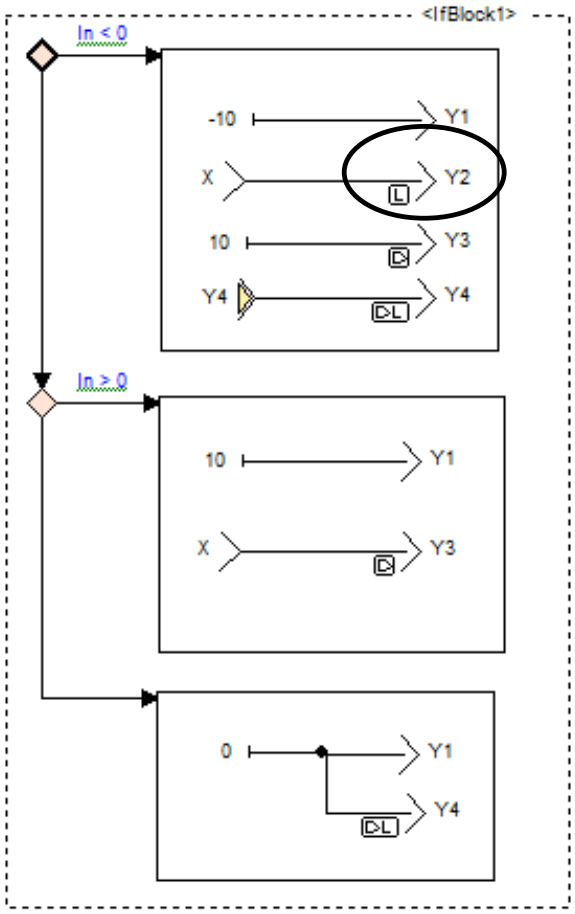
- Users can set either **a default value** or **maintain the last value of the flow**

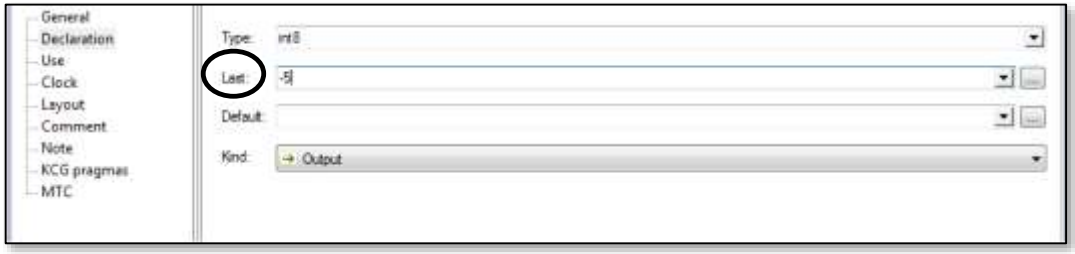


© 2016 ANSYS, Inc.

Conditional Block: Data Flow Inside Block

Maintain the latest value: Y2





Y2 is produced
with value X

"maintain the latest value"

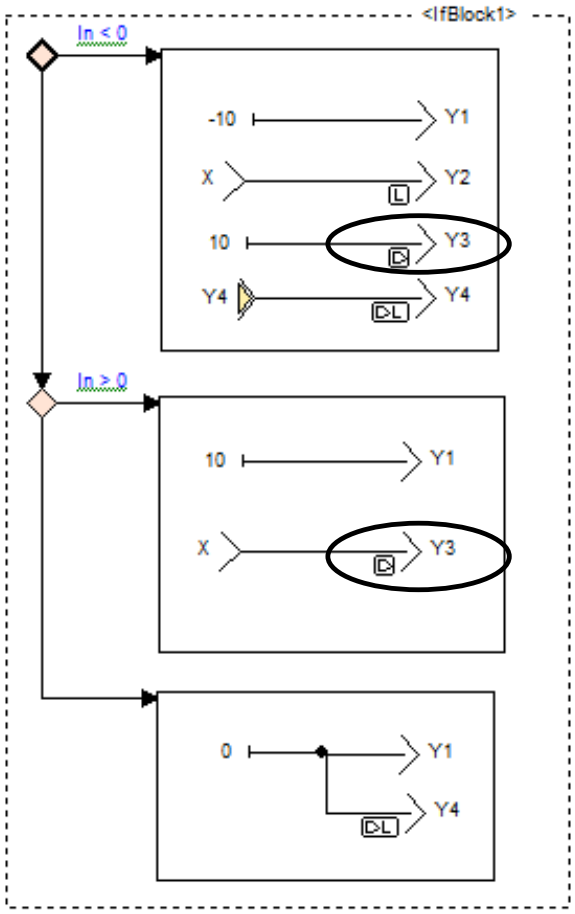
Y2 is produced
with last value of Y2, or -5 at the first cycle

"maintain the latest value"

Y2 is produced
with last value of Y2, or -5 at the first cycle

Conditional Block: Data Flow Inside Block

Produce a default value: Y3



Y3 is produced with 10

Y3 is produced with X

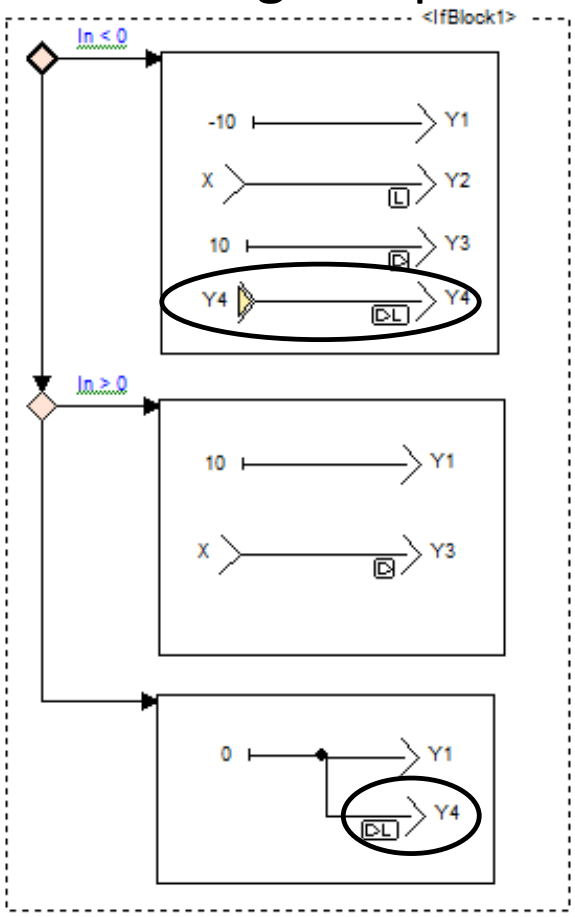
"Produce a default value"

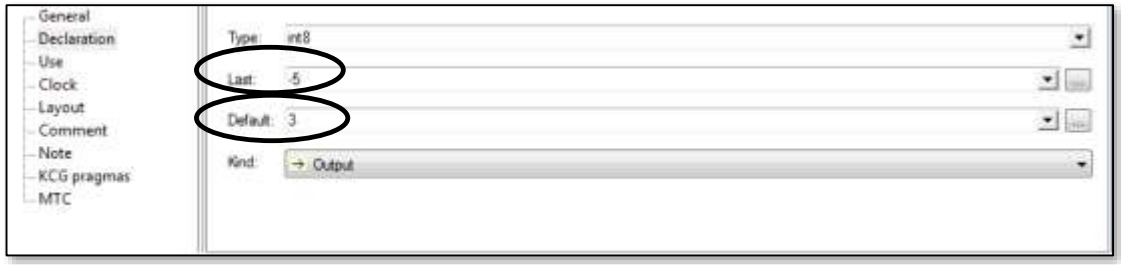
Y3 is produced with 3

 © 2016 ANSYS, Inc.

Conditional Block: Data Flow Inside Block

If both Last and Default are defined, the Default value has the highest priority: Y4





Which value is produced for Y4?

(see next slides)

"Produce a default value"

Y4 is produced with 3

Y4 is produced with 0

 © 2016 ANSYS, Inc.

Last Primitive

The last primitive is applied to an input/output/variable/signal to memorize and return its previous cycle value.

A flow has a single definition and must always be defined hence the need for last (and default) value to cover this rule.

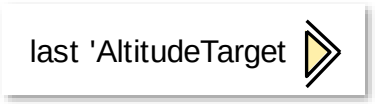
Differences Between last and pre:

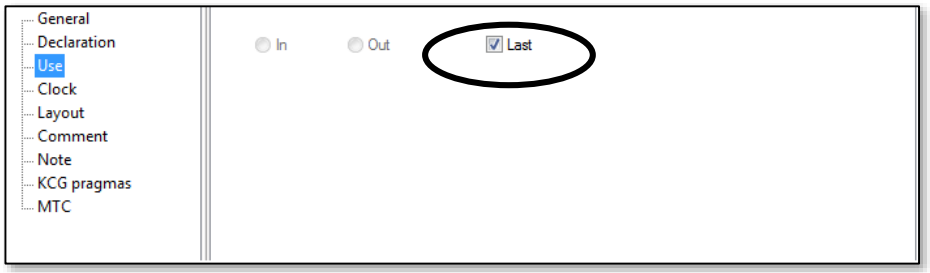
- **pre** applies to an expression, **last** to an identifier (variables, inputs, outputs, signals)
- Same behavior outside control blocks
- Within a control block, pre provides the value at the previous cycle where the block was active, last provides the value at the previous cycle whether the block was active or not.

© 2016 ANSYS, Inc.

Create Last Primitive

Drag & drop the variable and click Last in the “Use” tab

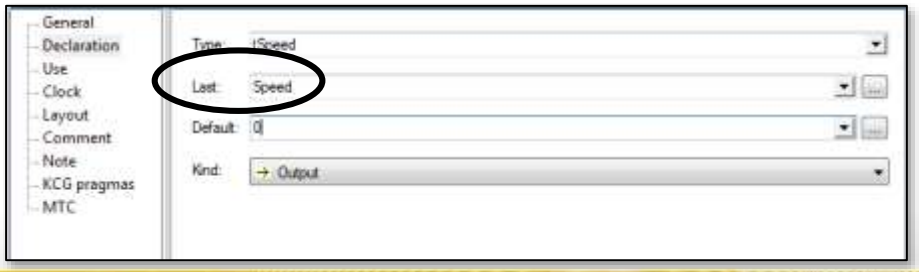




Use a textual expression of the form: last'VarName (Note the single quote in this expression)

A “last” variable must be defined on the first cycle.

Specify this initial value in the “Last” text box in the variable declaration tab



© 2016 ANSYS, Inc.

Exercise 5: last primitive

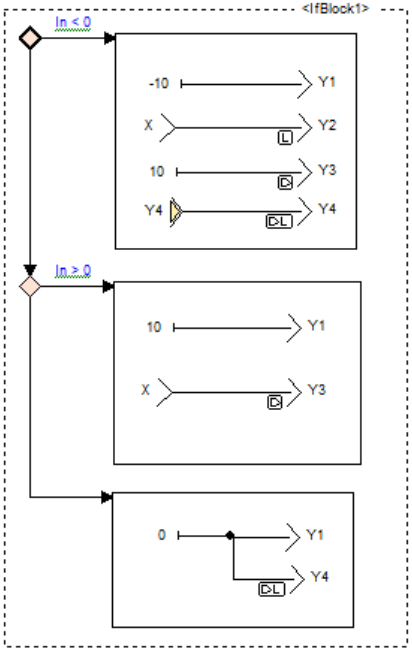
Exercise

Objective

Observe the `last` primitive behavior

Requirements

Create the following operator and simulate to observe the behavior



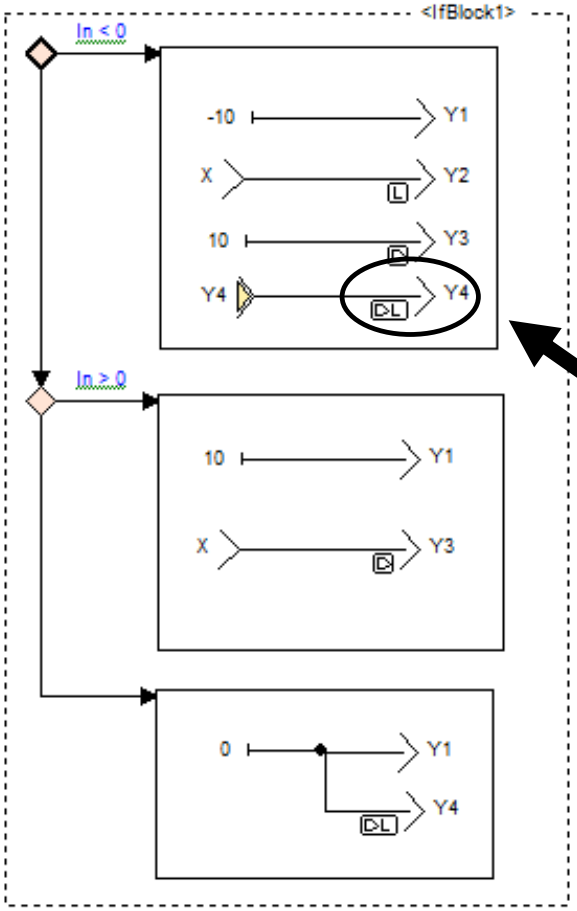
Name	Kind	Type	default/last
X	input	int8	-
Y1	output	int8	-
Y2	output	int8	default: -5
Y3	output	int8	last: 3
Y4	output	int8	default: -5 last: 3

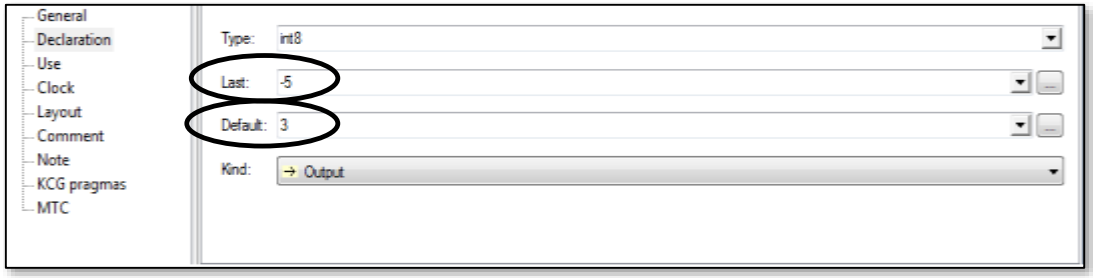
© 2016 ANSYS, Inc. February 1, 2017 ANSYS Confidential

Exercise 5: Solution

Exercise

If both Last and Default are defined, the Default value has the highest priority: Y4



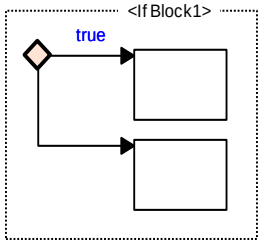


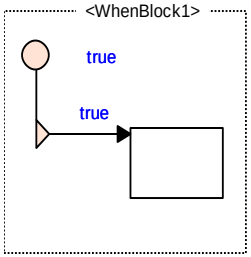
Y4 is produced with last value of Y4, or -5 at the first cycle

© 2016 ANSYS, Inc.

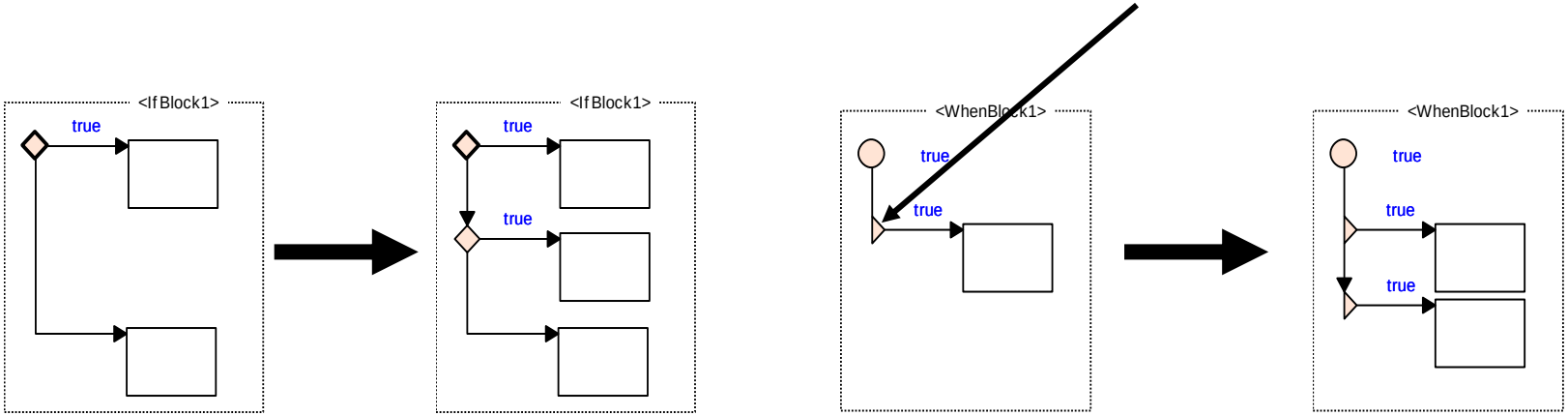
Conditional Block: Creation

Click on  or  and draw a rectangle in the diagram to add an If or When Block.





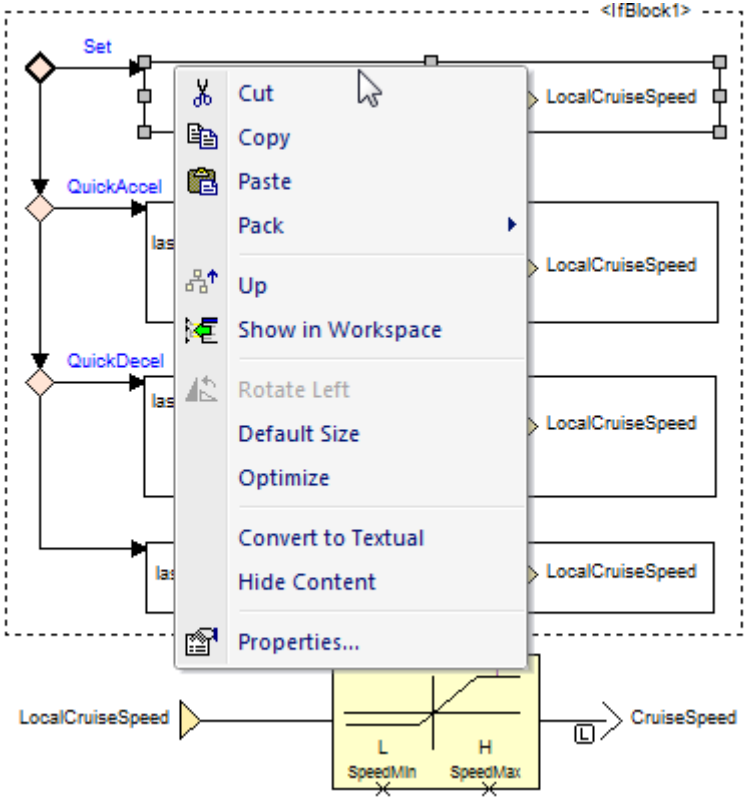
It is possible to add a branch by clicking on  (If) or  (When) and clicking on an else, or case, to add the new one.



© 2016 ANSYS, Inc.

Block Activation: Textual Equations

Right click on the control block to enter textual equations and select "Convert to Textual"



 © 2016 ANSYS, Inc.

Lab 4 (1/3)

Use Lab Support p.49-63

Objective:

Implement the `CruiseSpeedMgt` operator (**use of a conditional block**)

Requirements:

Time: 20 min

The `CruiseSpeedMgt` operator manages the value of the cruise speed according to the driver's commands (Set or QuickAcccel or QuickDecel buttons pressed) when the Cruise Control is enabled

(requirements CC_HLR_CSM_01 to 05)

Lab

© 2016 ANSYS, Inc.

Lab 4 (2/3)

Lab

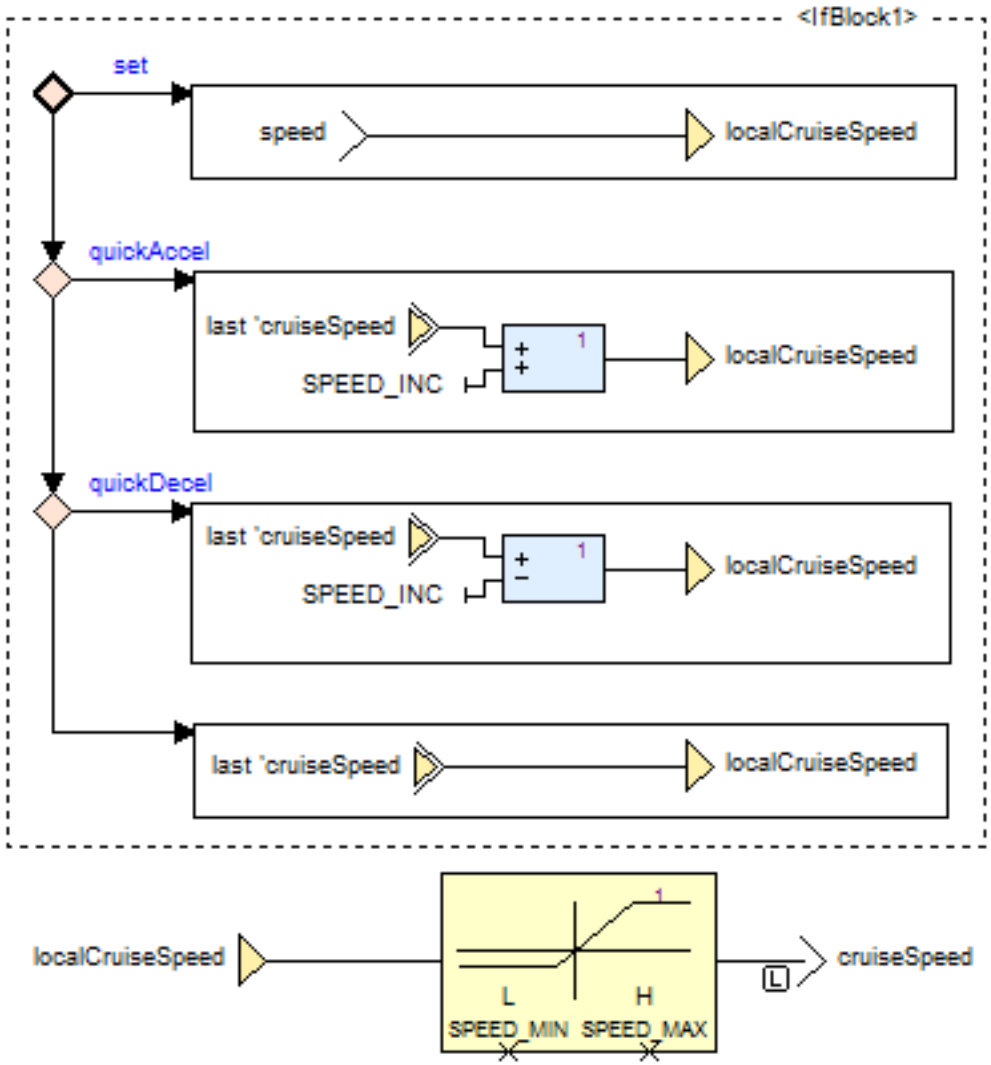
In package `CruiseControl`, create `CruiseSpeedMgt` operator

Name	Kind	Type
Set	input	bool
QuickAccel	input	bool
QuickDecel	input	bool
Speed	input	CarType::tSpeed
CruiseSpeed	output	CarType::tSpeed

- Set the cruise speed to the current speed when the Set Button is pressed
- Increase the cruise speed of SpeedInc when the QuickAccel button is pressed
- Decrease the cruise speed of SpeedInc when the QuickDecel button is pressed
- Maintain the cruise speed to the last value (the default value of Last is Speed) when no button is pressed
- The cruise speed is well maintained between SpeedMin and SpeedMax km/h *(use `pwlinear::LimiterUnSymmetrical` library operator)*

© 2016 ANSYS, Inc.

Lab 4 (3/3): CruiseSpeedMgt



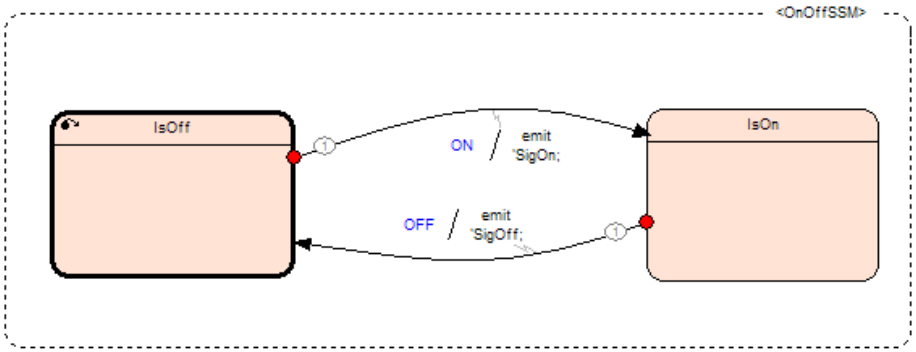
© 2016 ANSYS, Inc.

State Machines

State machines are models describing the behavior of a software

Composed of a finite number of states and transitions

Considered as Control flow constructs where their activation is a function of the previous state

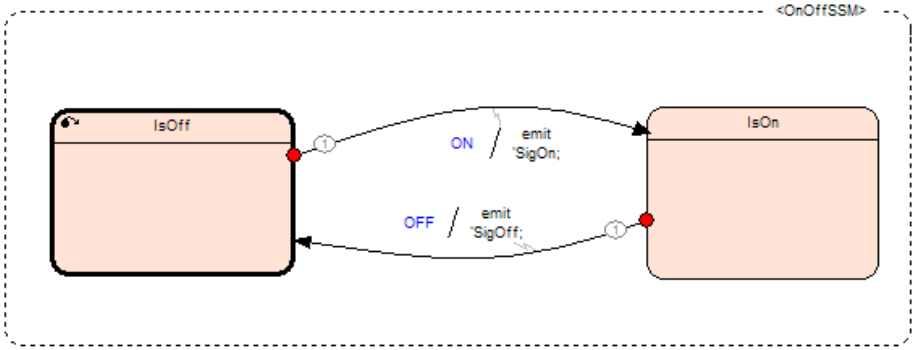


© 2016 ANSYS, Inc.

State Machines

States are represented by boxes

Transitions are represented by arcs



States and transitions have labels for names and behaviors

© 2016 ANSYS, Inc.

State Machines

A SCADE State machine has following properties:

- Hierarchy for state architecture
- Parallelism for capturing the concurrency
- Synchronization of parallel state machines
- Preemption (transition) for handling errors or exceptions
- Smooth and consistent mix with data-flow design

=> Factorization of the model architecture: write things once and better readability

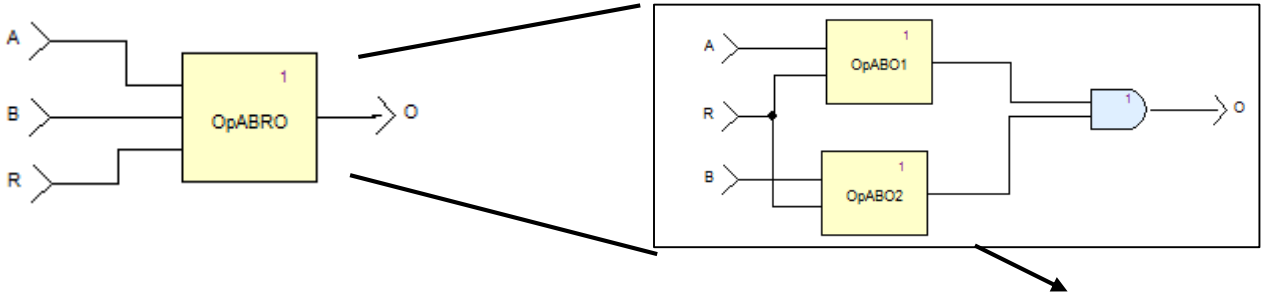
© 2016 ANSYS, Inc.

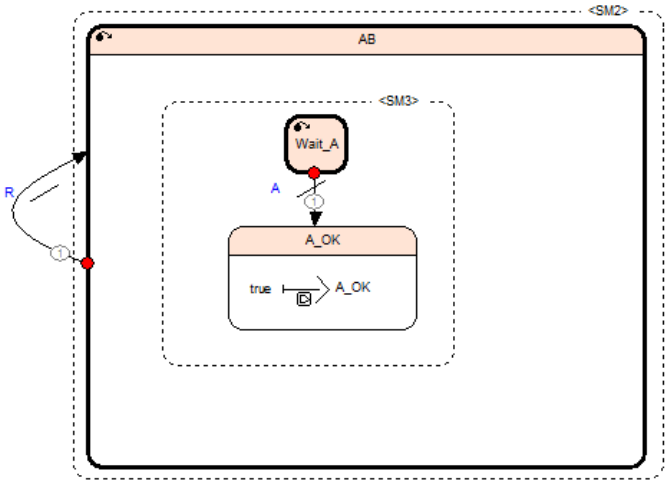
Design without Parallelism

Example

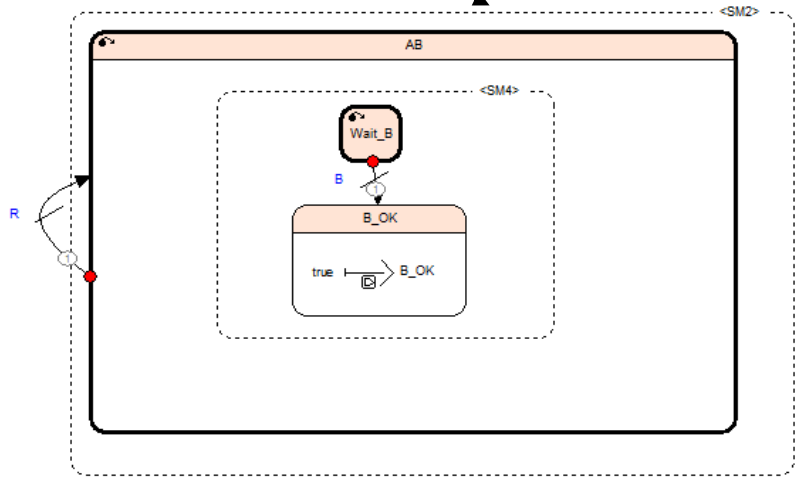
O is set to true when A and B are set to true and associated states are reached

AB state is reset when R is set to true





OpABO1 node



OpABO2 node

© 2016 ANSYS, Inc.

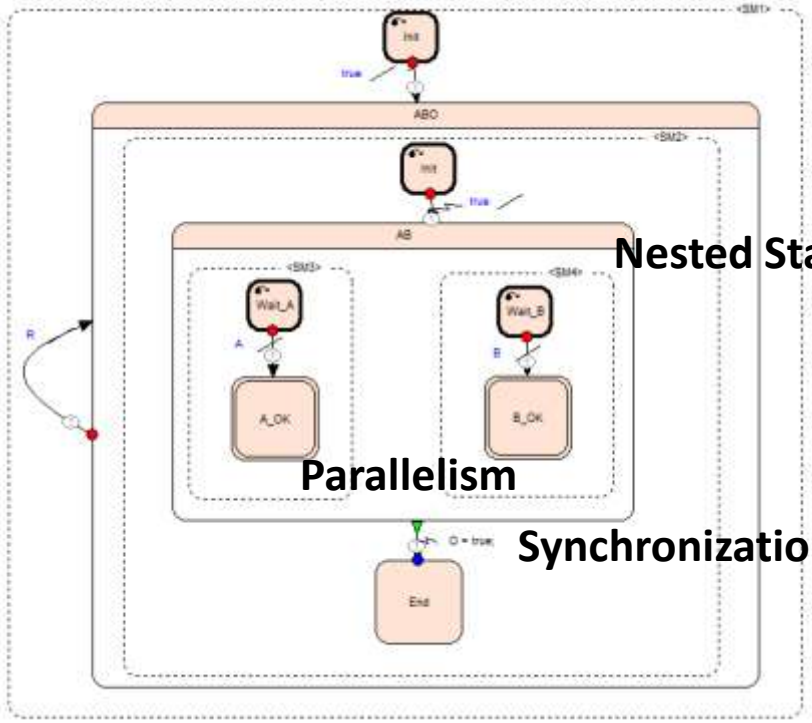
Design with Parallelism

Example

O is set to true when A and B are set to true and associated states are reached

- Synchronization of parallel States Machines

ABO state is reset when R is set to true



© 2016 ANSYS, Inc.

State Machine Definition

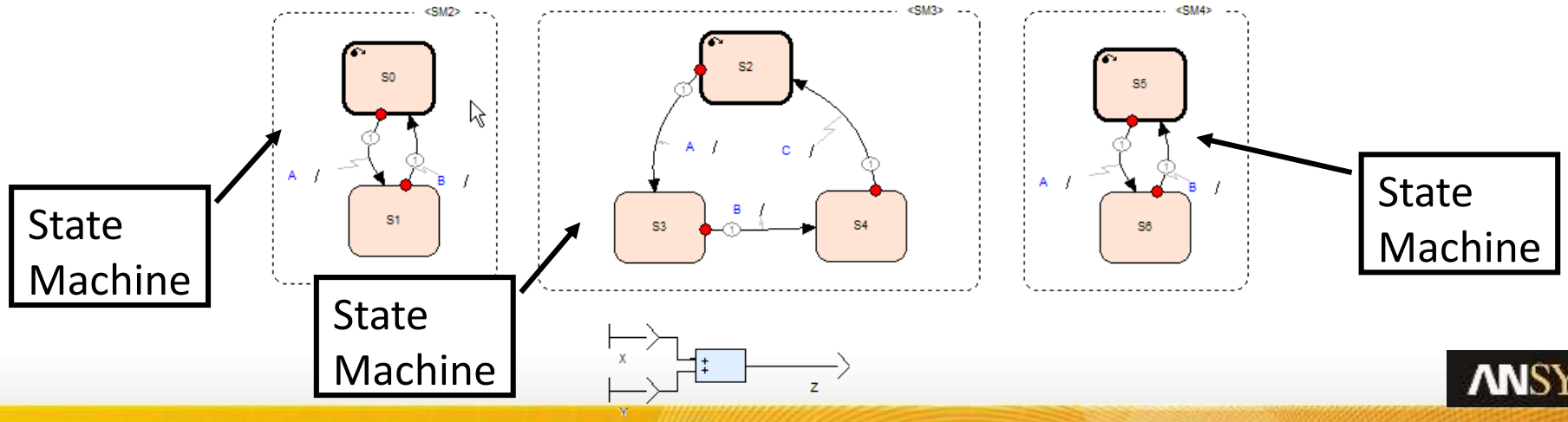
Created within a SCADE Suite operator:

- Only contains states and transitions
- Can own several parallel State Machines, each containing one or more states

Has a unique initial state:

- When another state is set as initial, any existing initial state is reset

Surrounded by dashed lines to delimit it from the other State Machines or data-flow behaving in parallel

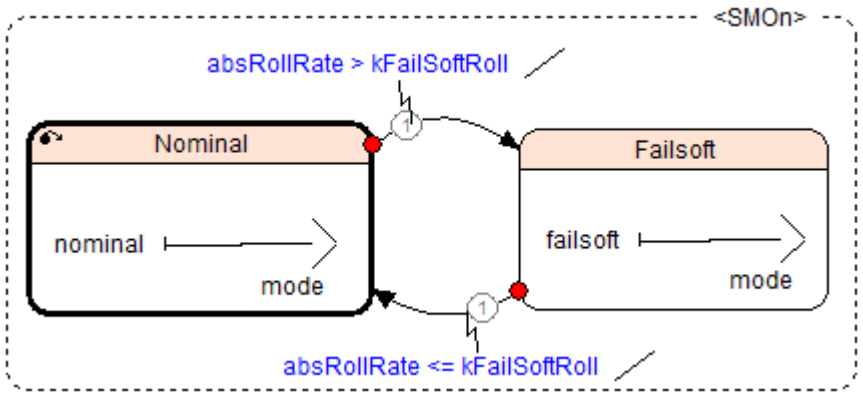


© 2016 ANSYS, Inc.

State Machine States

States:

- Graphical rectangle that must be named
- At each cycle, a state is either active or not
- Only one state can be active/executed in a State Machine during a cycle

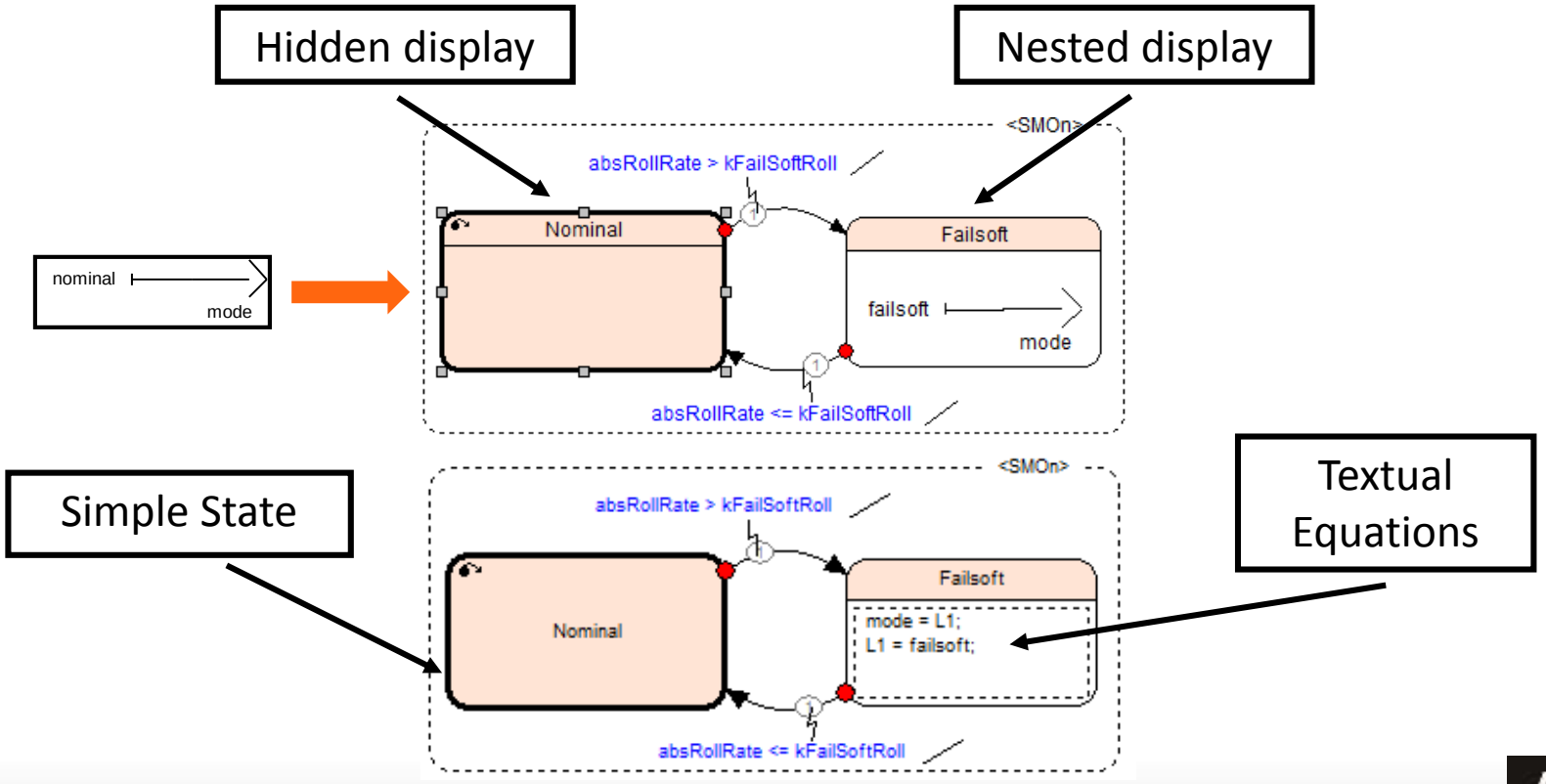


© 2016 ANSYS, Inc.

State Machine States

States may contain nested State Machine and graphical or textual data flows

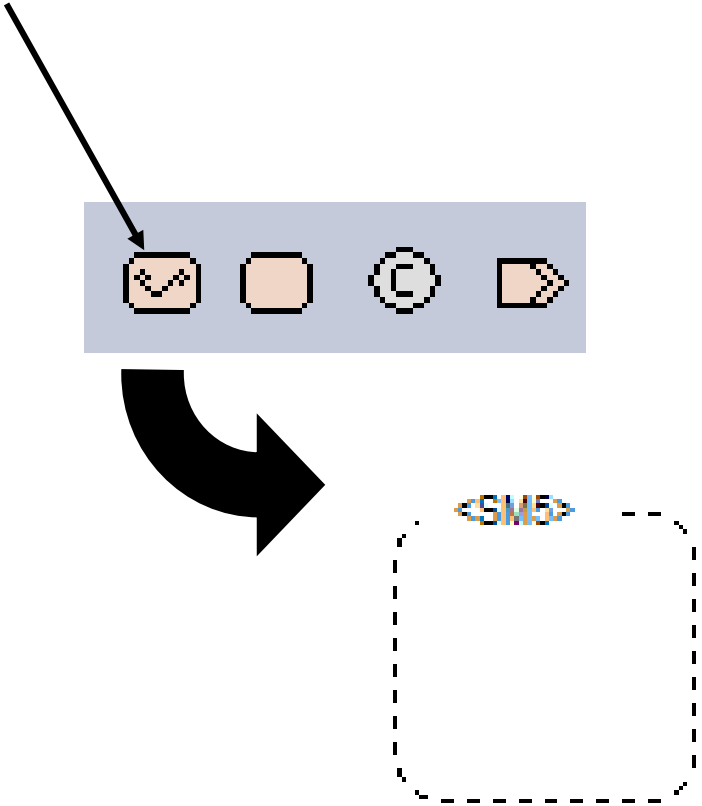
Easy way to model control structures and state hierarchy



© 2016 ANSYS, Inc.

Creating a State Machine

Click on the “New State Machine” to create an empty State Machine



 © 2016 ANSYS, Inc.

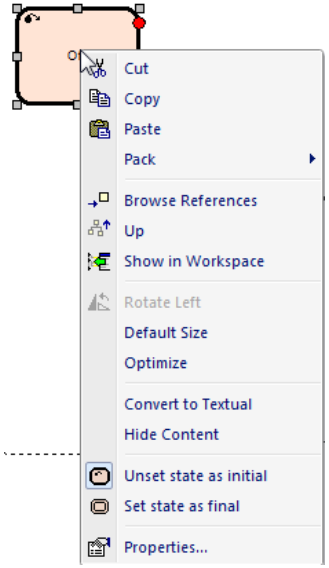
Initial State

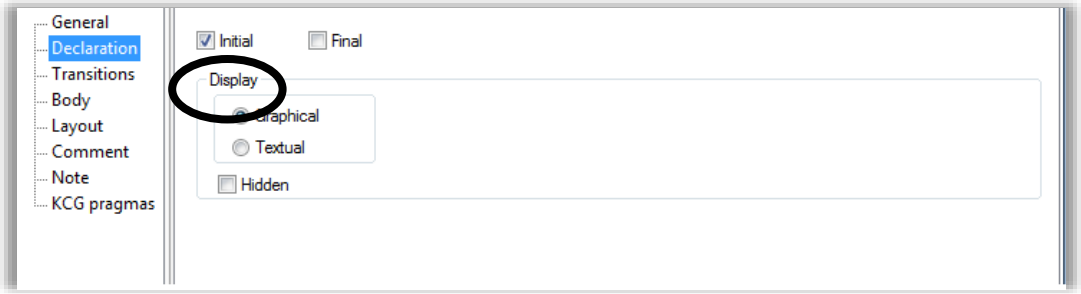
Graphically depicted with a bolder outline

The initial state is active whenever the State Machine is started

How to set a state as initial?

- From its contextual menu “Set/Unset state as initial”
- From is Properties window “Initial” checkbox





- From the State[illegible]hine toolbar dedicated button

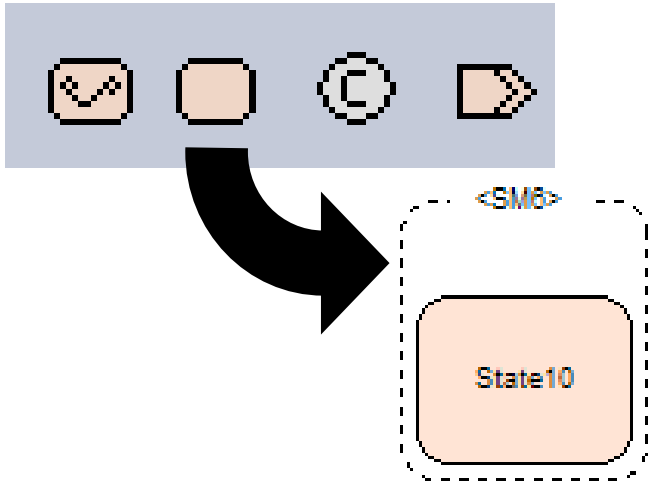
When a state is added to a State Machine, with no initial state, the state becomes initial

© 2016 ANSYS, Inc.

State Editing: Creating States

Creating states:

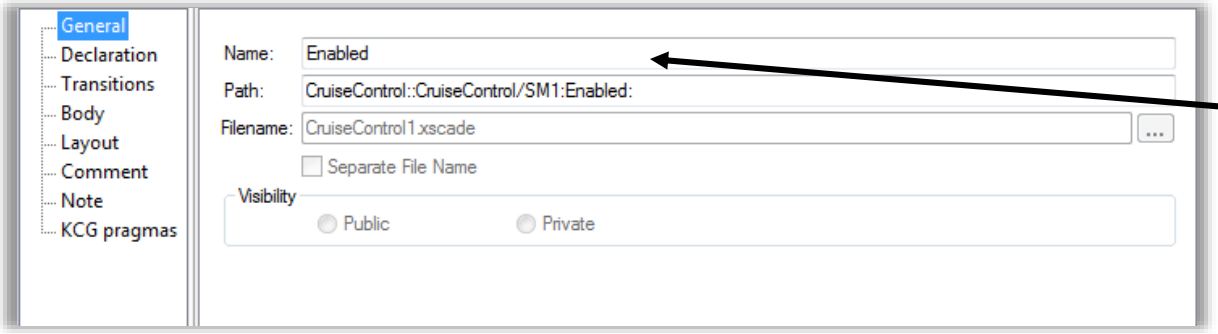
- Click the New State button
  - Click into state machine
  - Click & drag to custom its size



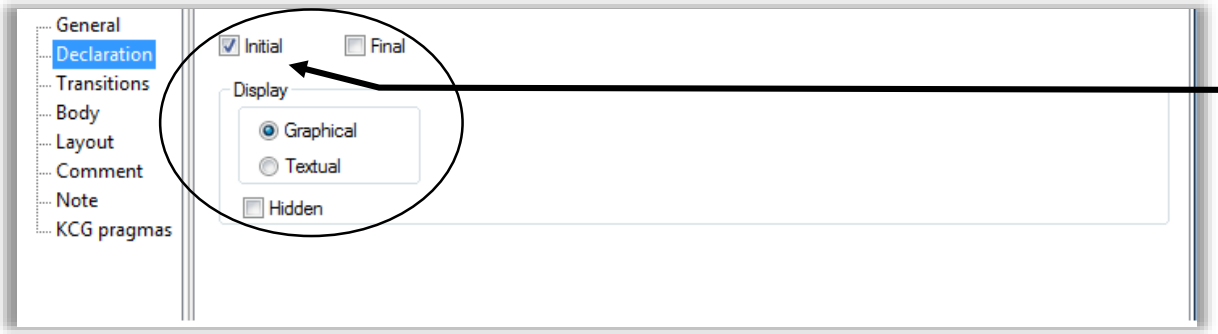
States are visible in the Scade View under their State Machine entries

 © 2016 ANSYS, Inc.

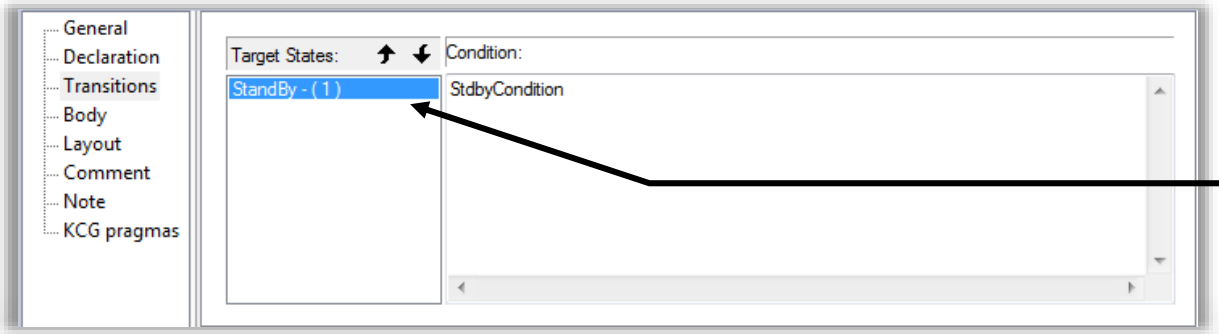
State Editing: State Properties



Name: a SCADE Suite identifier: Enabled …



Mark state as initial or final, graphical, textual and toggle hidden



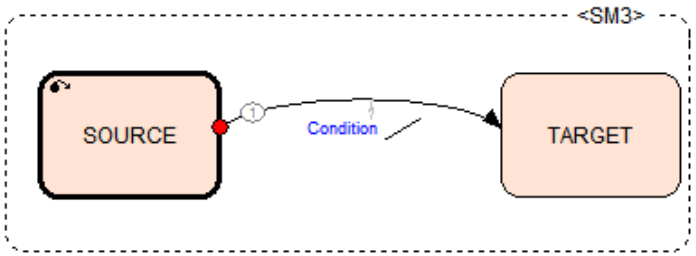
Outgoing transitions with priority and respective textual Scade code

© 2016 ANSYS, Inc.

State Machine Transitions

Have one source state and one target state

Label of the form Condition / Actions



© 2016 ANSYS, Inc.

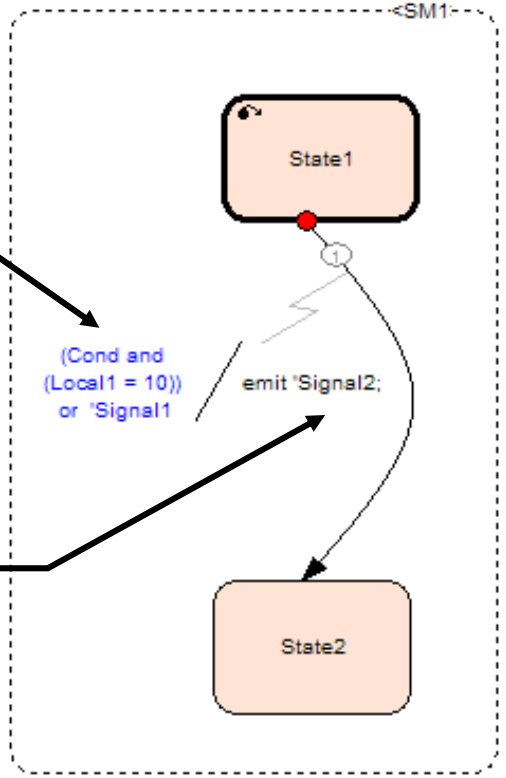
Transition Labeling

Scade textual expression / *Scade textual expression*

Condition *Actions*

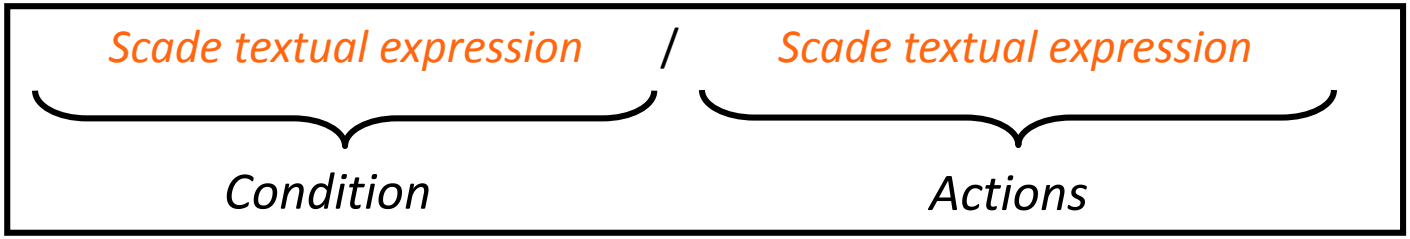
Condition:
Boolean condition

Actions: list of textual expressions



 © 2016 ANSYS, Inc.

Transition Labeling



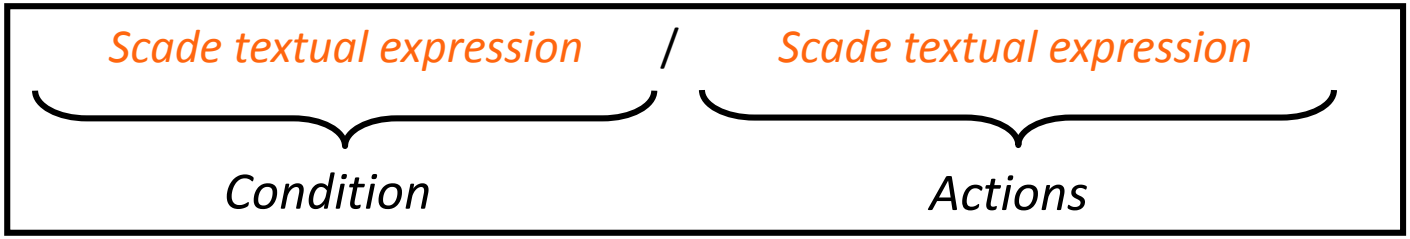
Condition:

- Boolean Scade expression
- Useful operators: times and last

```
2 times On
3 times (Local1 > 18.0)
5 times 'Signal1
last 'On
last 'Local1 > 18.0
last 'Signal1
2 times (last 'Local1 > 18.0)
```


© 2016 ANSYS, Inc.

Transition Labeling



Actions:

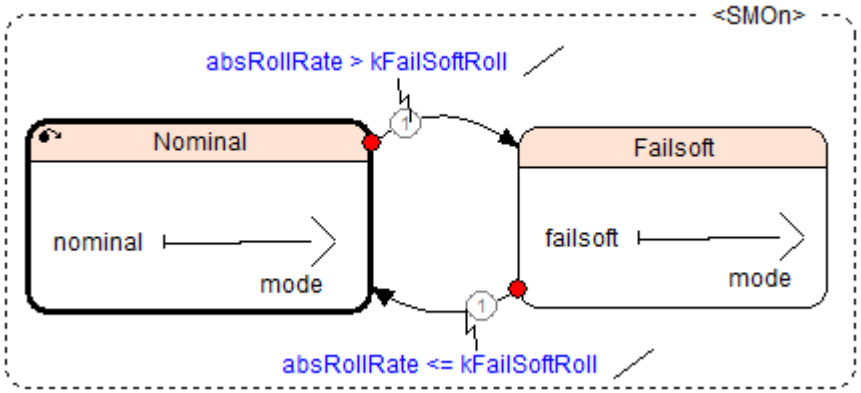
List of signal emissions and variable definitions, which can contain any Scade expression:

```
emit 'Signal1;
Local1 = 15.0;
StopEngine = true;
MaxValue = Max(Value1, Value2);
```


© 2016 ANSYS, Inc.

Transition Firing

A transition can be fired if its trigger condition is true and if its source state was active at the previous cycle:



- If absRollRate>kFailSoftRoll is true and if the Nominal state was active at the previous cycle, this state is preempted and state Failsoft is activated

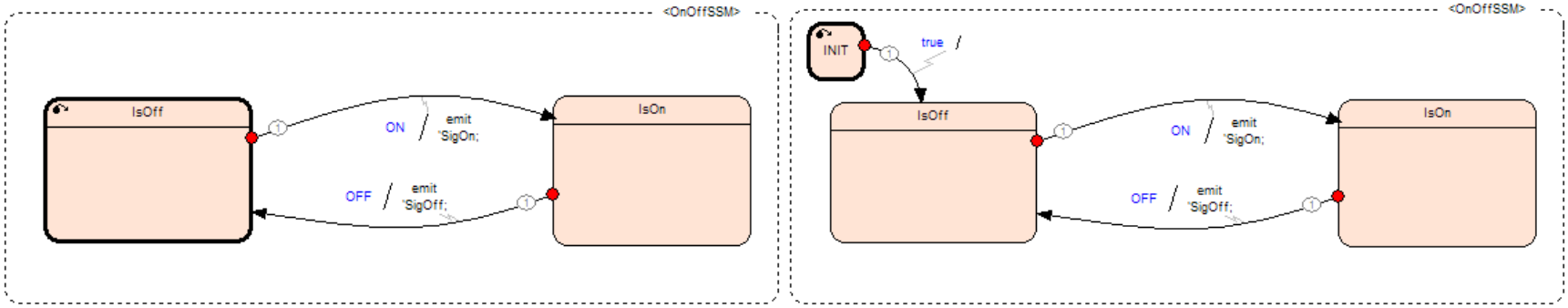
At most one transition is fired in a cycle (per State Machine).

© 2016 ANSYS, Inc.

Transitions and Initial State

At the initial cycle, the transitions of the initial state are immediately fire-able.

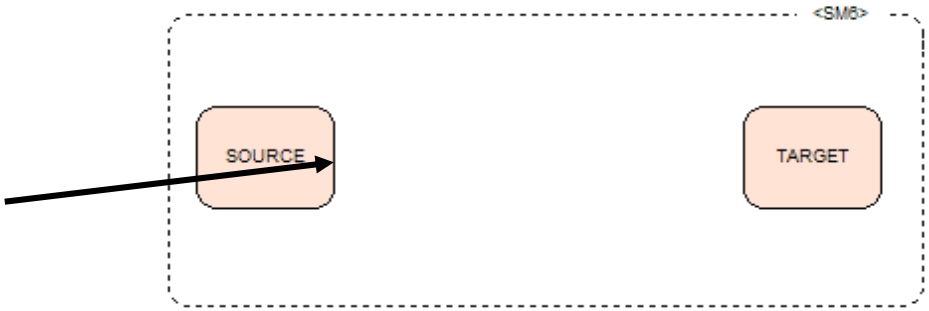
At the other cycles, the initial state behaves like any other state.



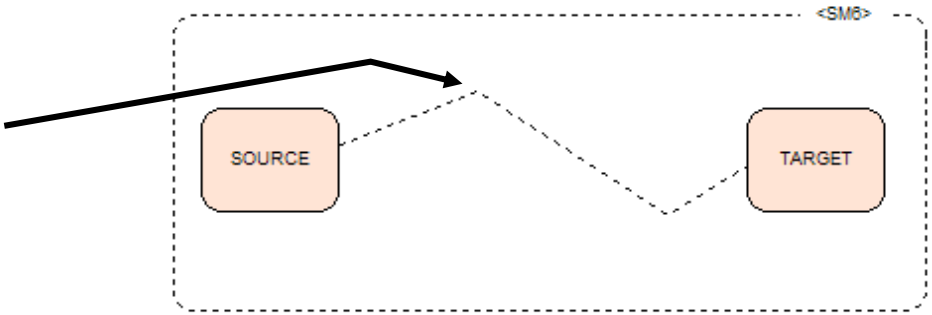
© 2016 ANSYS, Inc.

Transition Editing: Creating a Transition

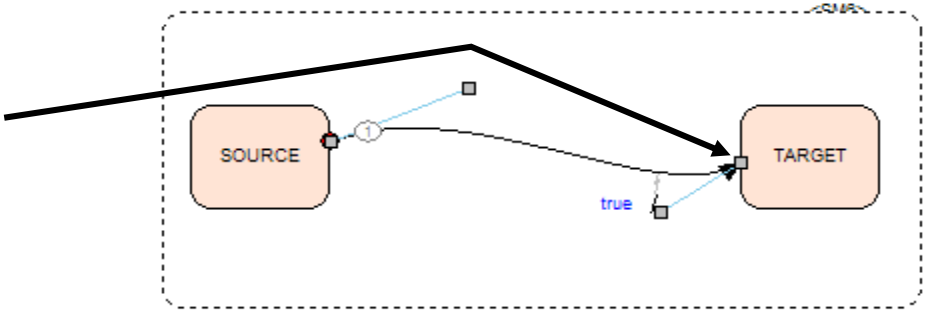
1. Position the mouse cursor over the edge of the source state and click the left mouse button



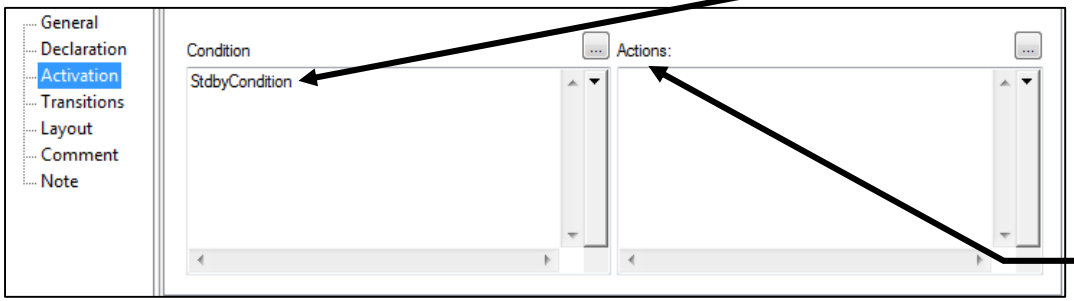
2. Click as many times as needed for a path to the border of the target state



3. Click on the border of the target state to create the final transition

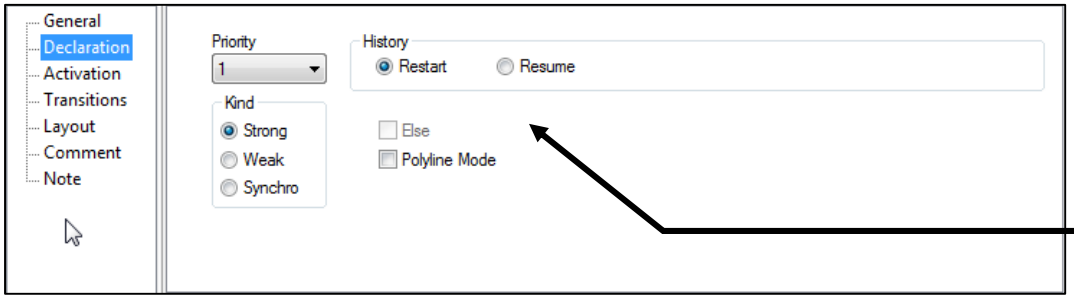


Transition Editing: Transition Properties



Condition: Scade expression that triggers the transition when true

Actions: Scade expression that describes actions to be taken when the transition is triggered



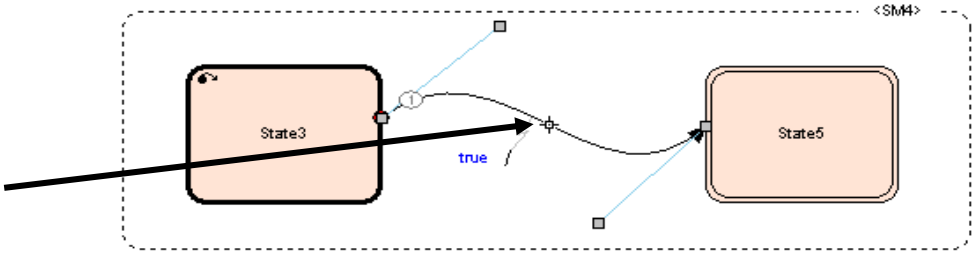
Priority: ensures determinism
Kind: preemption type
History: resume states
Polyline: straight lines

© 2016 ANSYS, Inc.

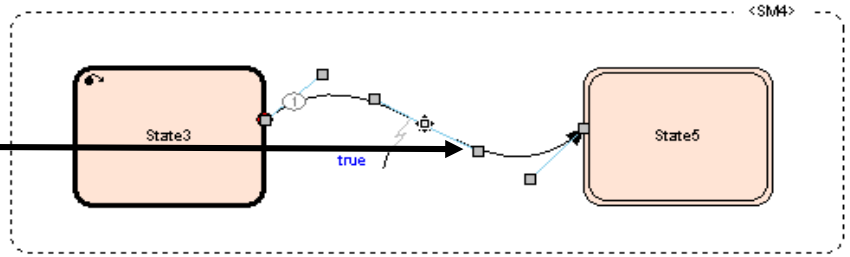
Transition Editing: Modifying Layout

Additional control vectors

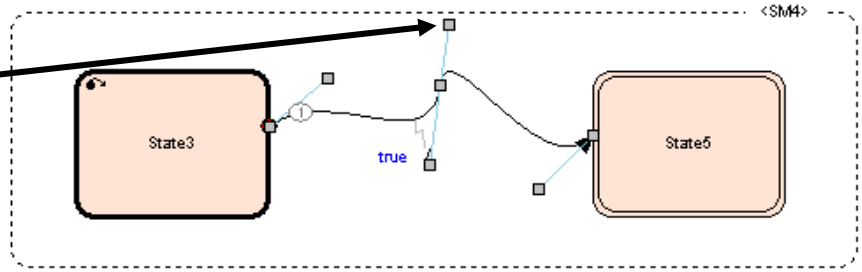
1. Position mouse cursor over a selected transition and holding down CTRL key



2. Keep CTRL pressed and click to add control vector



3. Move control points to modify transition



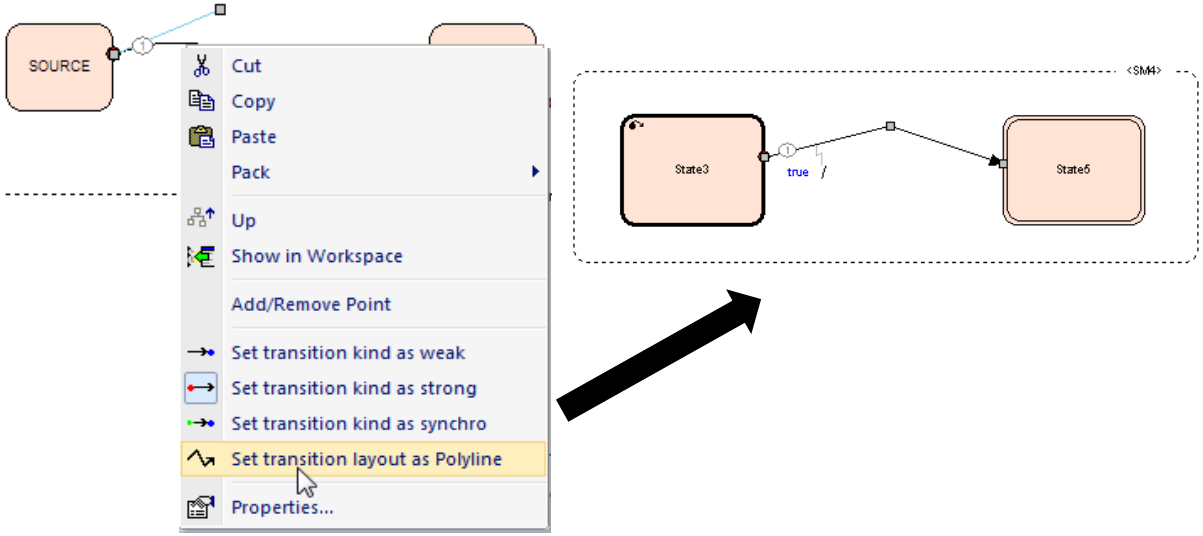
© 2016 ANSYS, Inc.

Transition Editing: Modifying Layout

Two layout types:

- Bezier
- Polyline

Contextual access:



Menu access:

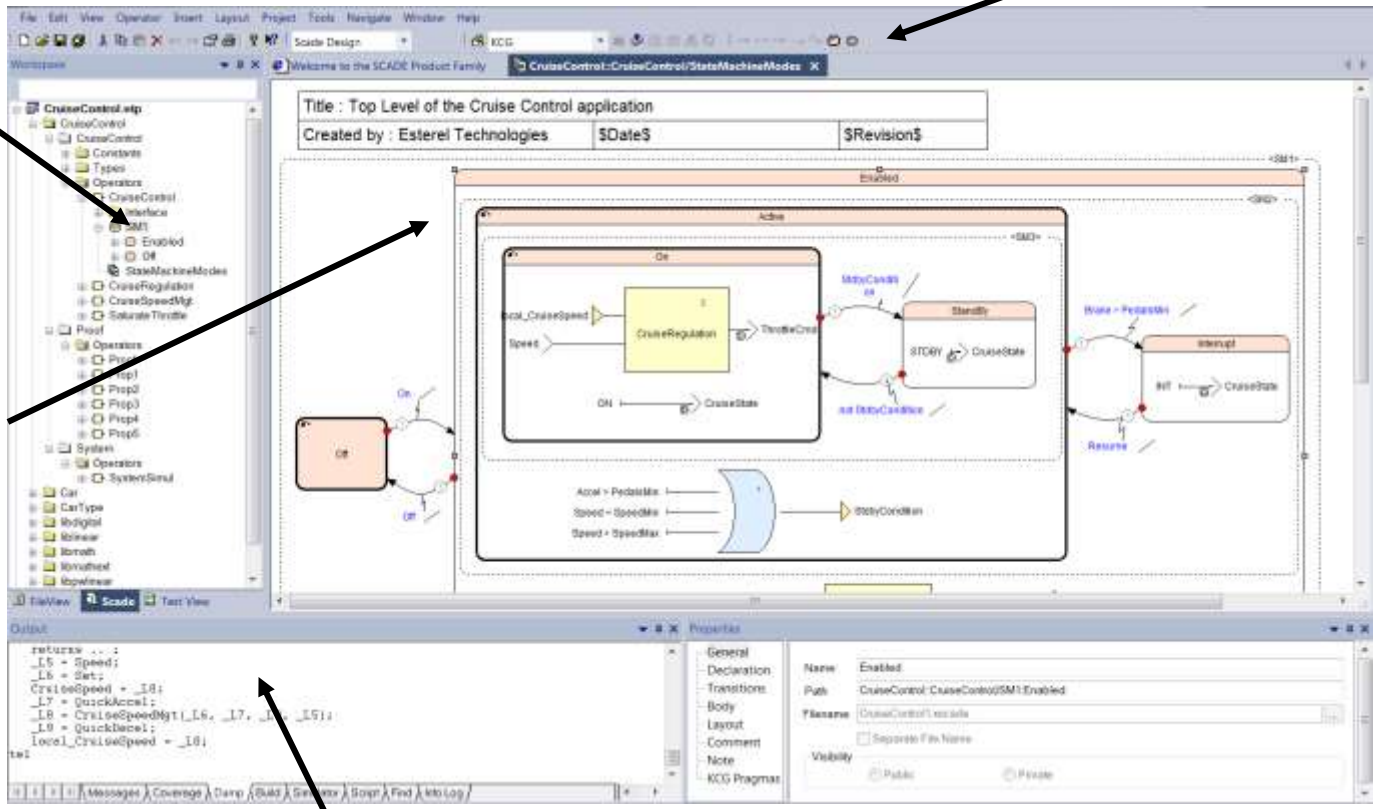
© 2016 ANSYS, Inc.

State Machine in Scade View

State Machine Toolbar

State Machine in Scade View

State Machine in diagrams



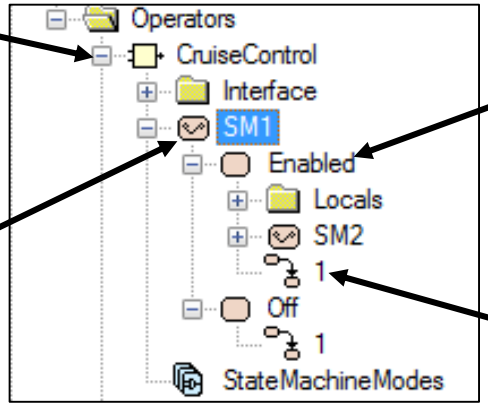
Dump tab:
State Machine textual representation

 © 2016 ANSYS, Inc.

State Machine in Scade View

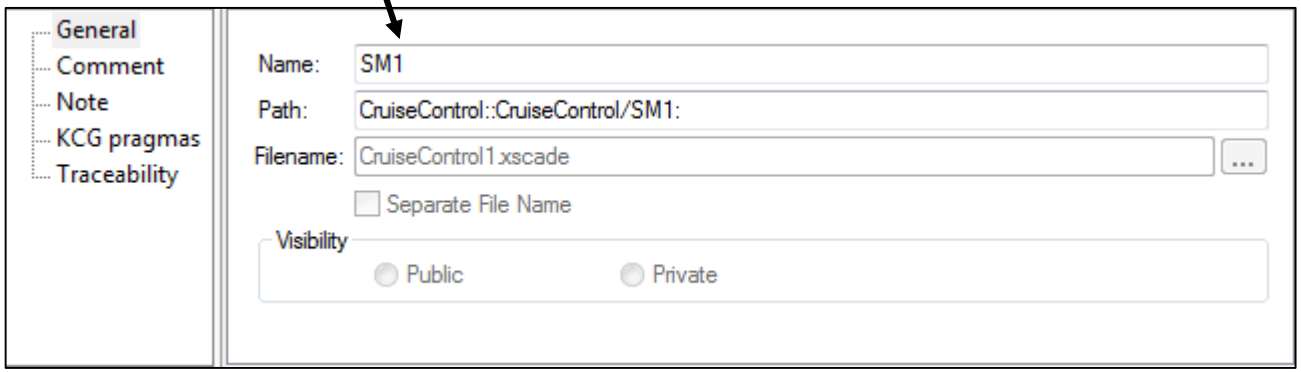
Node containing a State Machine

SM Name:
- Must be unique in the node
- Can be set and modified in the State Machine Properties Window



State Machine State

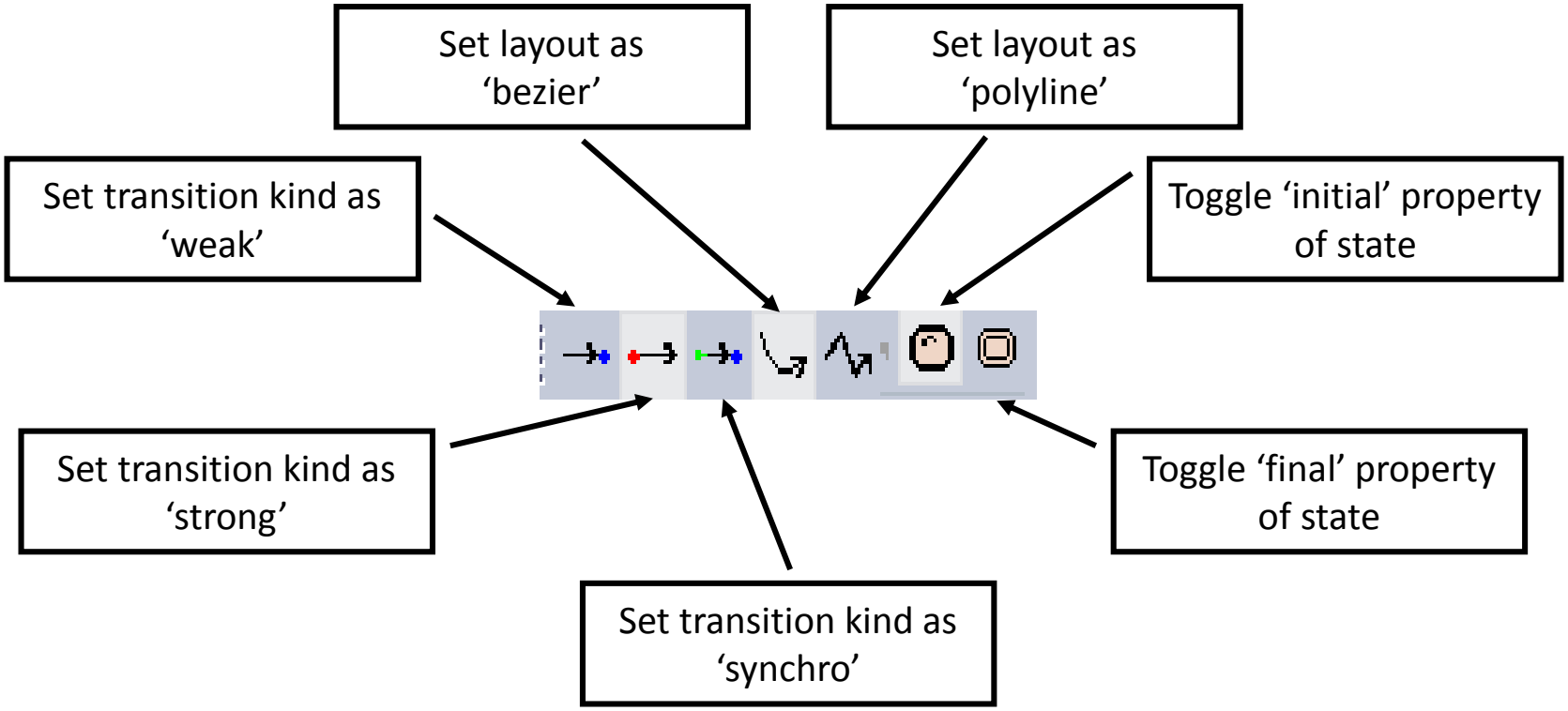
State Machine Transition



© 2016 ANSYS, Inc.

State Machine Toolbar

Shortcuts to some properties and settings:



© 2016 ANSYS, Inc.

Exercise

Exercise 6

Objective

Create a State Machine

Requirement

Create the `OnOff` operator in a new project

Project Name: `SSM_Exercises`

Operator Name: `OnOff`

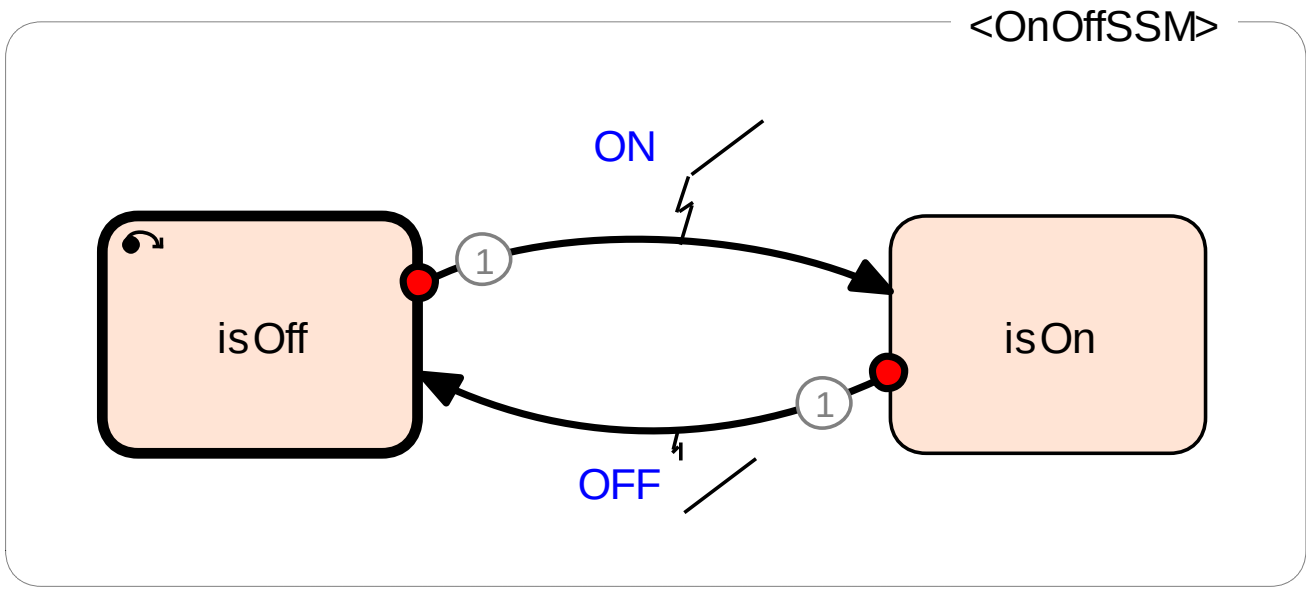
Name	Kind	Type
`ON`	input	bool
`OFF`	input	bool

Initial state: button is in `isOFF` state

Behavior: if `ON` is true and `OnOff` is in `isOFF` state, `OnOff` goes to `isON` state. If `OFF` is true and `OnOff` is in `isON` state, `OnOff` goes to `isOFF` state

© 2016 ANSYS, Inc.

Exercise 6: Solution



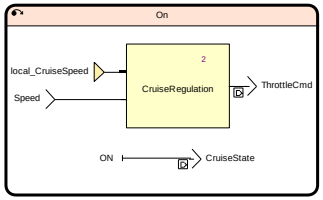
Exercise

© 2016 ANSYS, Inc.

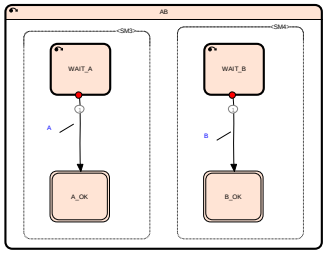
State Machine State Activities

A state, "similar" to a SCADE operator, can contain:

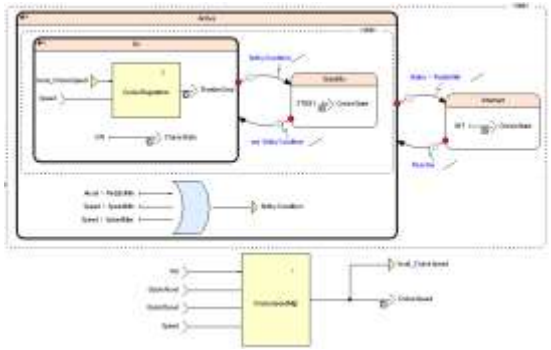
- Graphical data flow equations



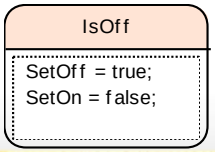
- Others State Machines



- And both



Can also be converted to a textual state



© 2016 ANSYS, Inc.

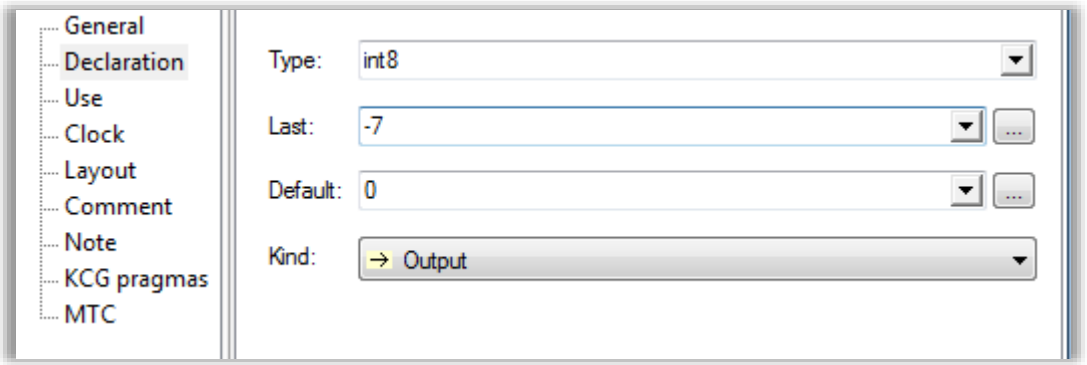
Data Flow Inside State

Equations are calculated only when the state is active.

Each declared flow (local or output) must have exactly one definition for each cycle where its scope is active.

What happens when a definition is missing in a given state?

- Users can set either **a default value** or **maintain the last value of the flow**



© 2016 ANSYS, Inc.

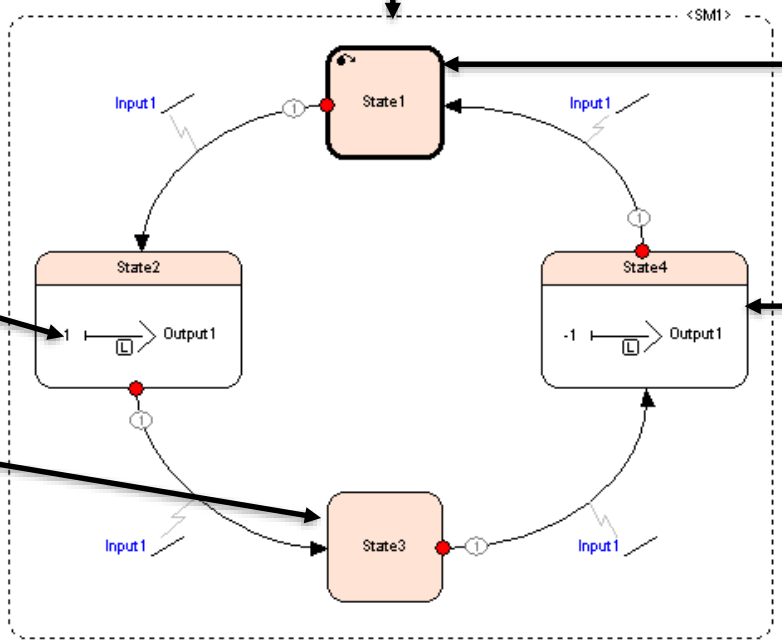
Data Flow Inside State

Maintain the latest value

Initial State1 - First cycle

"maintain the latest value"

Output1 not yet produced.
then set to 0 (initial last value)



State1 - Other cycles

"maintain the latest value"

Last value of Output1
(equal to –1) is maintained

State2

Output1 is produced
with value 1

State4

Output1 is produced
with value -1

State3

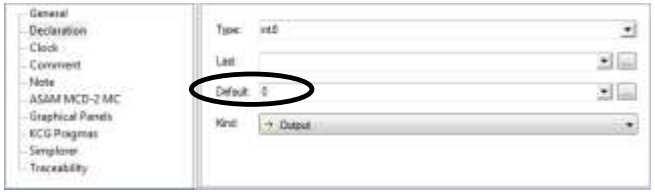
"maintain the latest value"

Last value of Output1
(equal to 1) is maintained

© 2016 ANSYS, Inc.

Data Flow Inside State

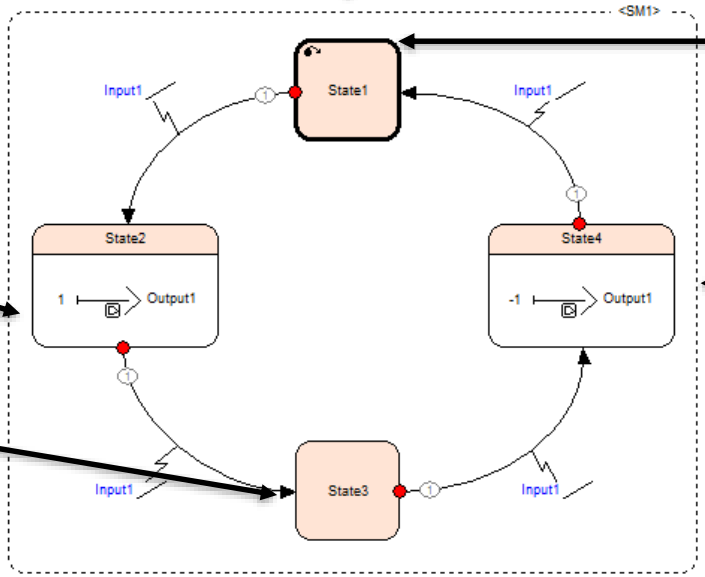
Produce a default value



Initial State1 - First cycle

"Produce a default value"

Output1 is produced
with default value 0



State1 - Other cycles

"Produce a default value"

Output1 is produced
with default value 0

State2

Output1 is produced
with value 1

State4

Output1 is produced
with value -1

State3

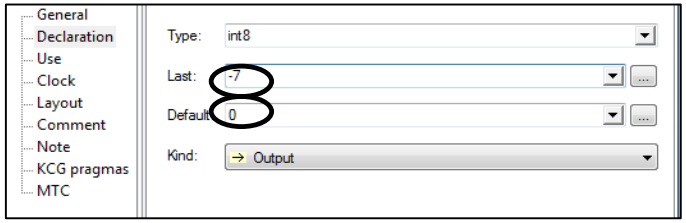
"Produce a default value"

Output1 is produced
with default value 0

© 2016 ANSYS, Inc.

Data Flow Inside State

If both Last and Default are defined, Default has a higher priority than Last one



Initial State1 - First cycle

"Produce a default value"

Output1 is produced with default value 0

State1 - Other cycles

"Produce a default value"

Output1 is produced with default value 0

State2

Output1 is produced with value 1

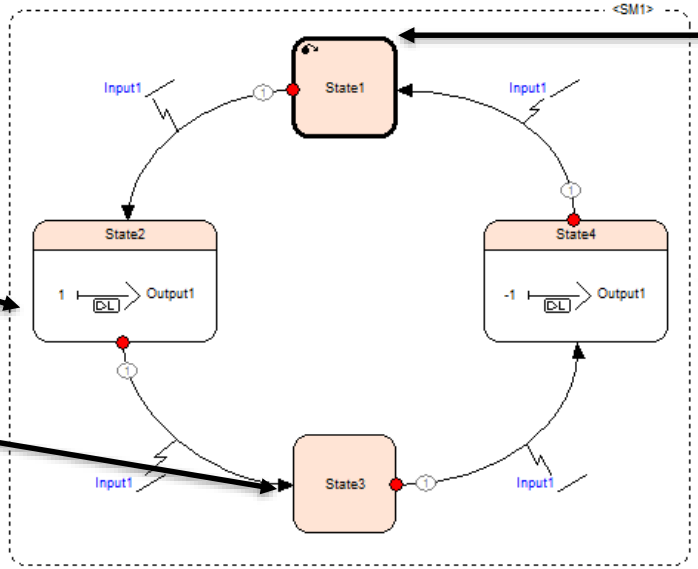
State4

Output1 is produced with value -1

State3

"Produce a default value"

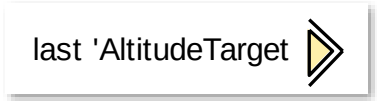
Output1 is produced with default value 0

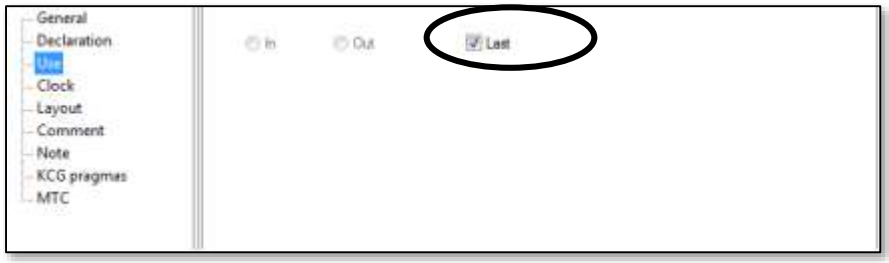


© 2016 ANSYS, Inc.

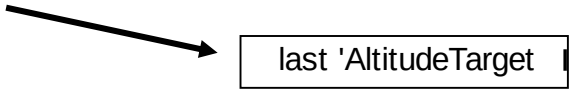
Create a Last Primitive

Drag & drop the variable and click Last in the “Use” tab



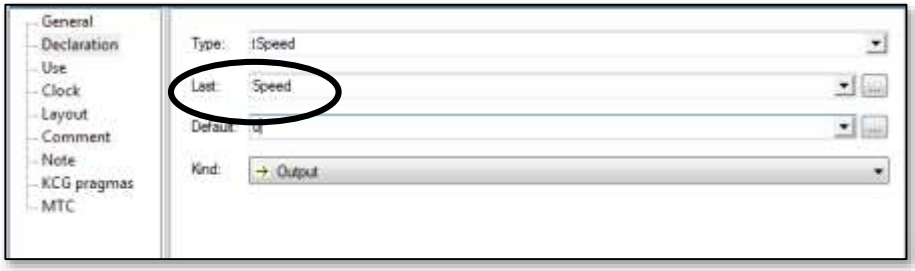


Use a textual expression of the form: last'VarName (Note the single quote in this expression)



A “last” variable must be defined on the first cycle.

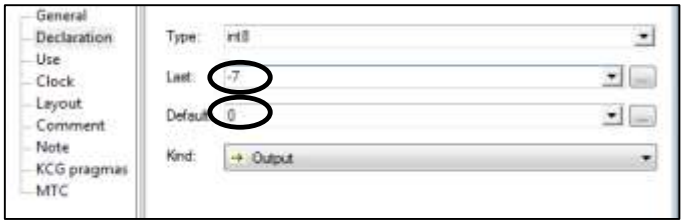
Specify this initial value in the “Last” text box in the variable declaration tab



© 2016 ANSYS, Inc.

Data Flow Inside State

Exercise



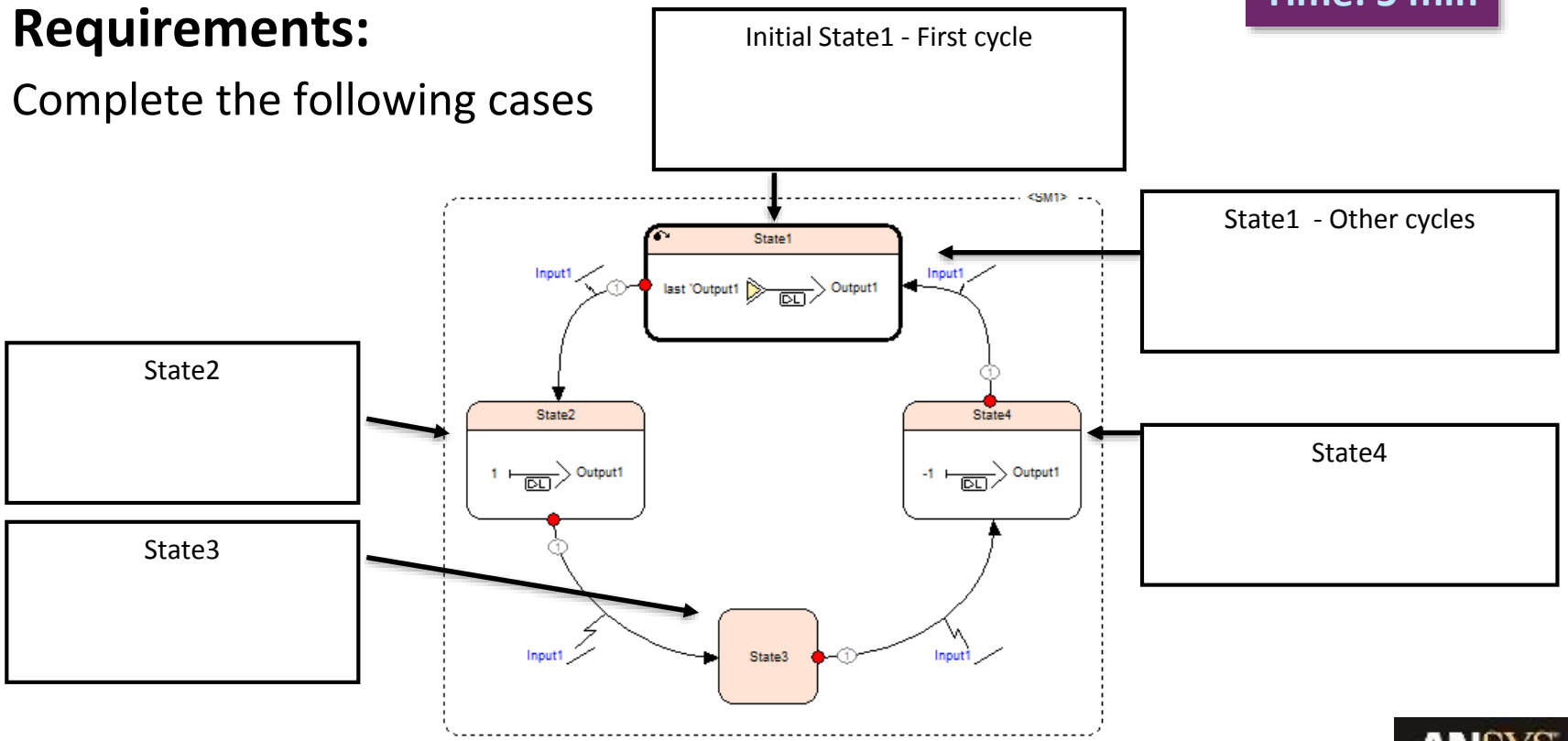
Objective:

Define the produced value for Output1

Time: 5 min

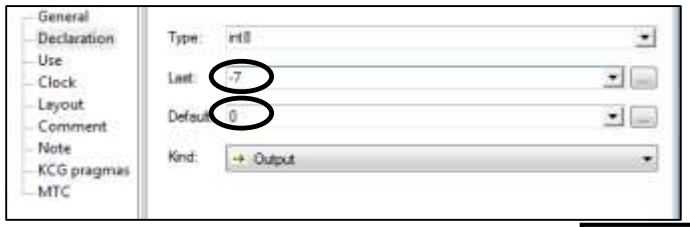
Requirements:

Complete the following cases



© 2016 ANSYS, Inc.

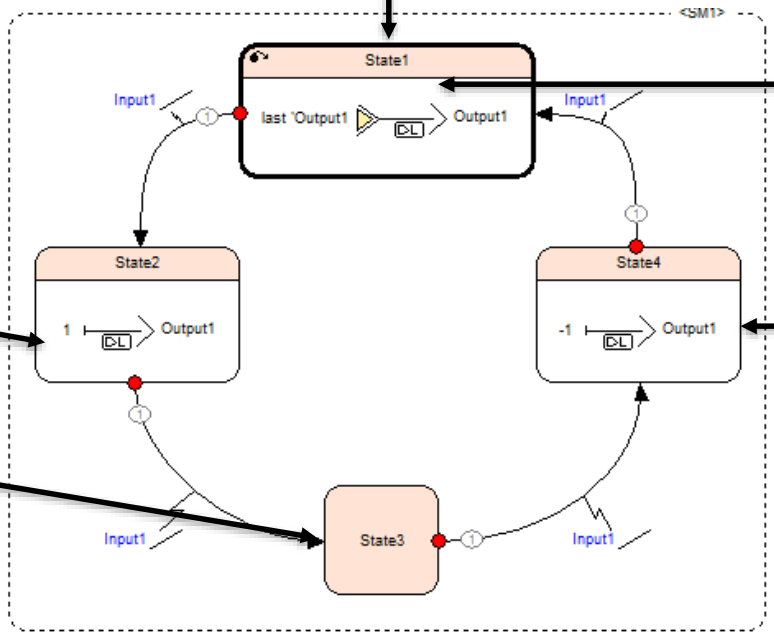
Solution



Initial State1 - First cycle

"maintain the latest value"

Output1 not yet produced.
then set to -7 (initial last value)



State1 - Other cycles

"maintain the latest value"

Last value of Output1
(equal to –1) is maintained

State2

Output1 is produced
with value 1

State4

Output1 is produced
with value -1

State3

""Produce a default value"

Output1 is produced
with default value 0

© 2016 ANSYS, Inc.

Data Flow Inside State

If the variable is produced in the initial state but no default and last value is defined: the last behavior is kept, the variable must have a definition in every state accessible in the first cycle

State1

"maintain the latest value "

Last value of Output1
(equal to -1) is maintained

State0

This initial state is not executed when the transition is true.

The value of Output1 must still be defined

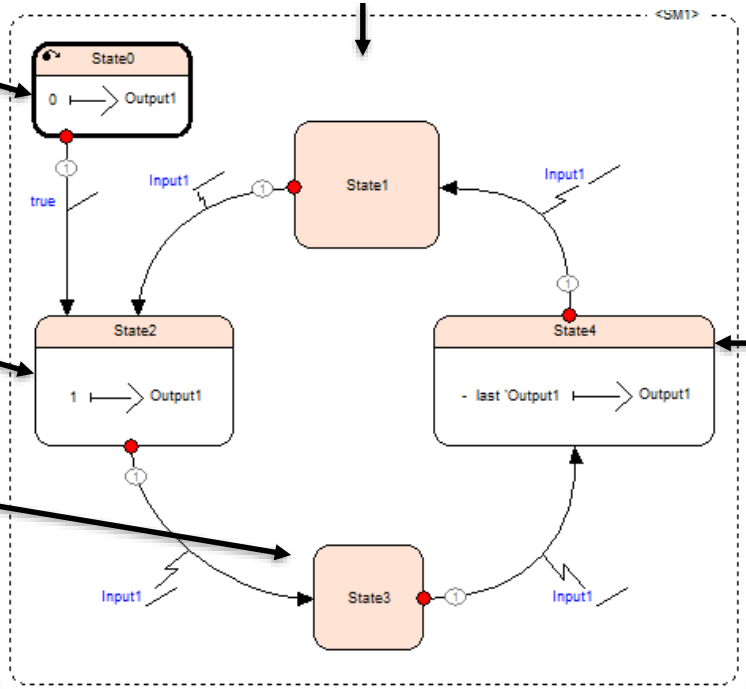
State2

Output1 is produced with value 1

State3

"maintain the latest value"

Last value of Output1
(equal to 1) is maintained



State4

Output1 is produced with value -1

© 2016 ANSYS, Inc.

Exercises

OPTIONAL

Exercise

Exercise 7

Objective

Observe the behavior of State Machine

Requirement

Add 2 output to the `OnOff` operator

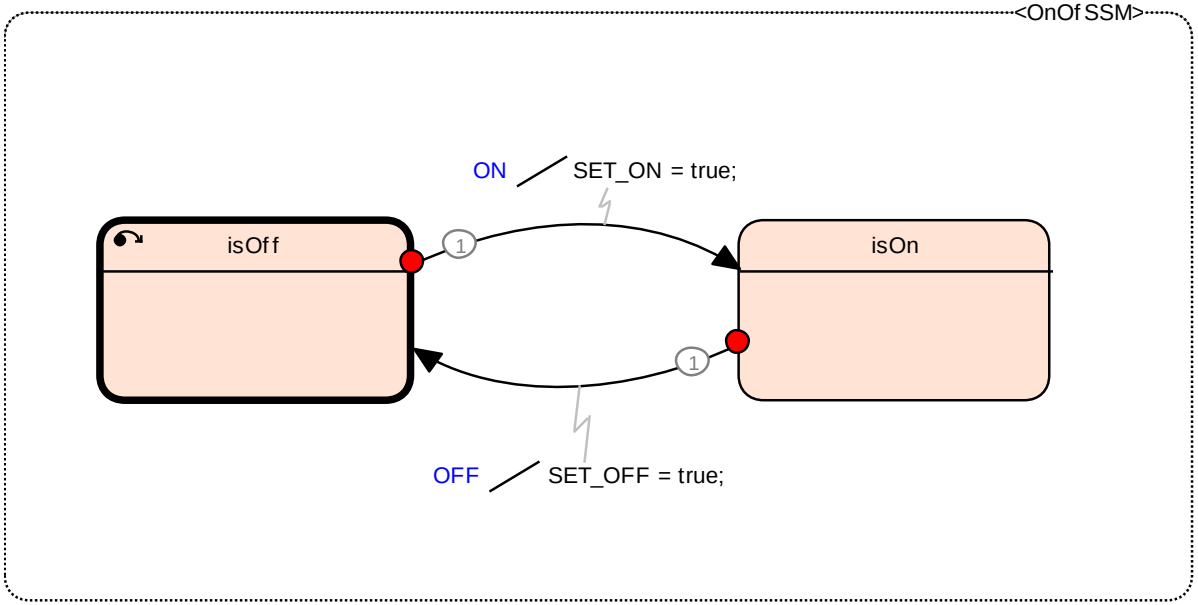
Name	Kind	Type
`ON`	input	bool
`OFF`	input	bool
`SET_ON`	output	bool
`SET_OFF`	output	bool

The outputs `SET_ON` and `SET_OFF` shall be true only when transition is fired (no data flow in state)

Use actions on transition and default values

© 2016 ANSYS, Inc.

Exercise 7: Solution



- Upon transition firing, `SET_ON` and `SET_OFF` are defined as true
- When no definition is activated the outputs' **default value** is false

Exercise

© 2016 ANSYS, Inc.

Exercise

Exercise 8

Objective

Observe the behavior of State Machine

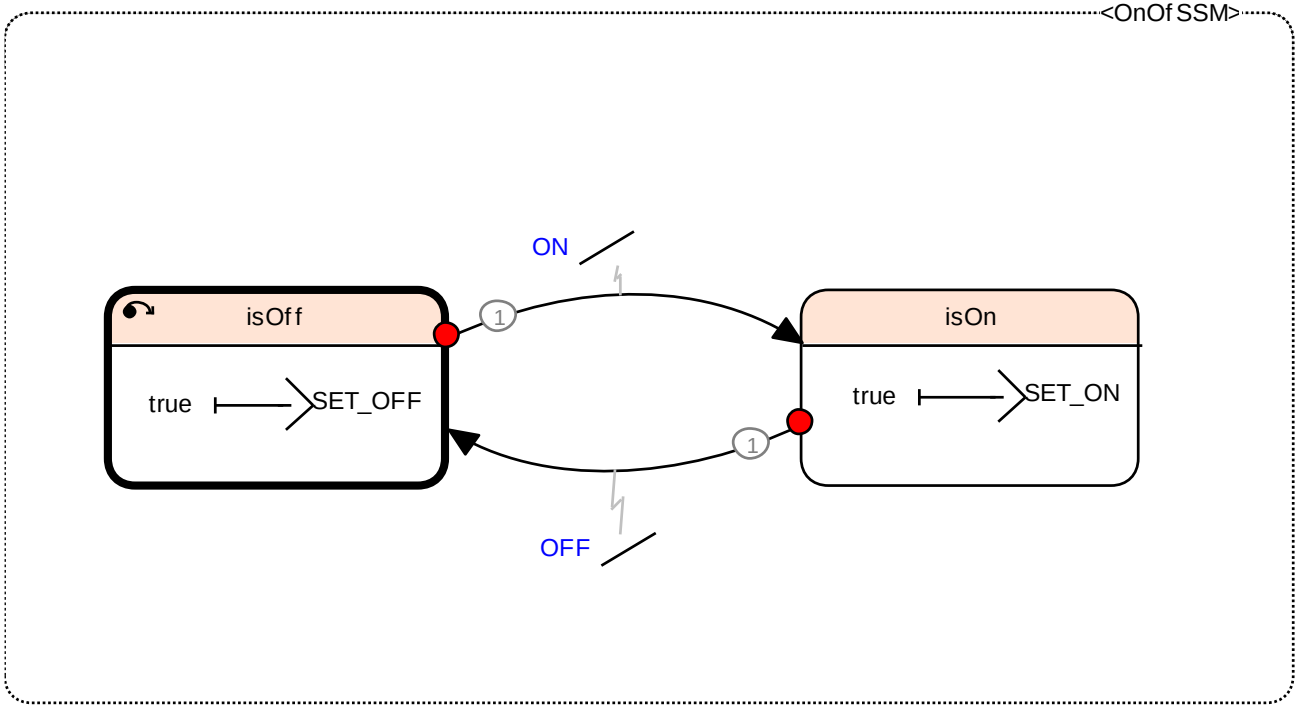
Requirement

- The `SET_ON` output shall be true at all times when state `ON` is active
- The `SET_OFF` output shall be true at all times when state `OFF` is active

Use dataflow inside states and default values

© 2016 ANSYS, Inc.

Exercise 8: Solution



- Upon transition firing, states `IsOn` resp. `IsOff` are activated
- The operator outputs `SET_ON` and `SET_OFF` are continuously updated
- When no definition is activated the outputs' default value is false

Exercise

© 2016 ANSYS, Inc.

Exercise

Exercise 9

Objective

Observe the behavior of State Machine

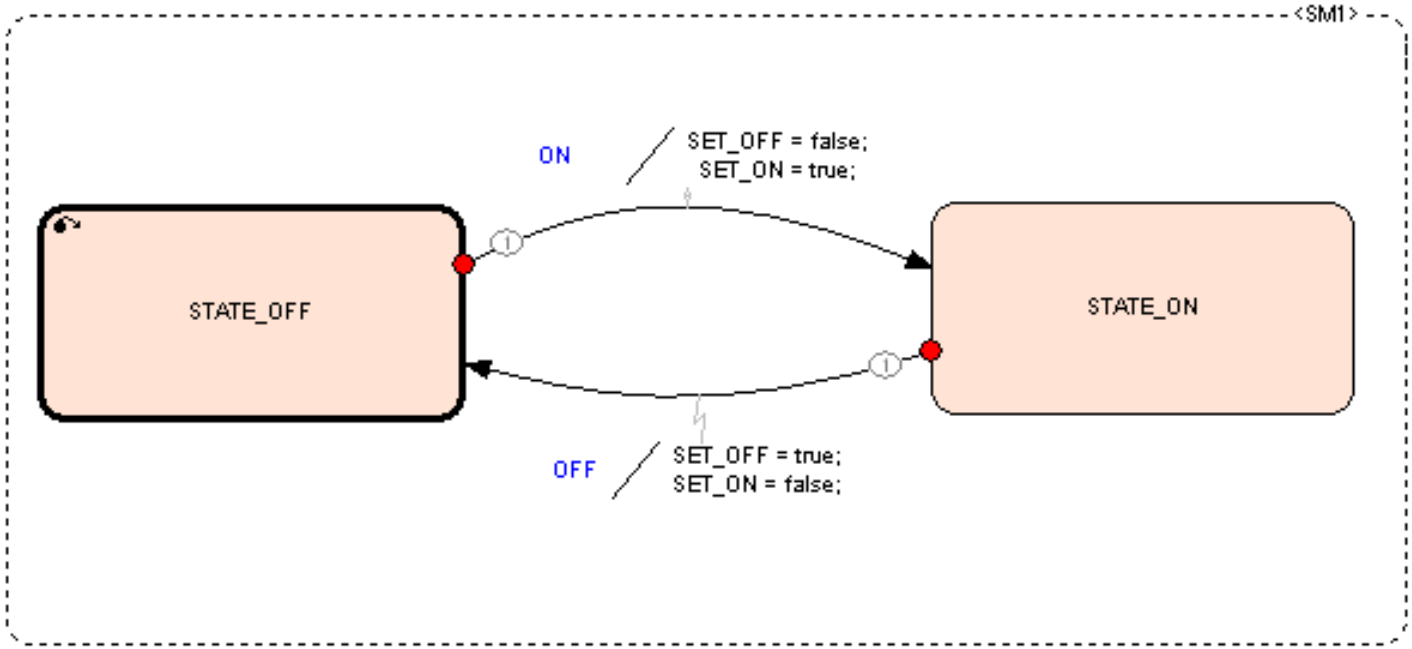
Requirement

- The `SET_ON` output shall be true at all times when `isON` state is active
- The `SET_OFF` output shall be true at all times when `isOFF` state is active
- No data flow in state

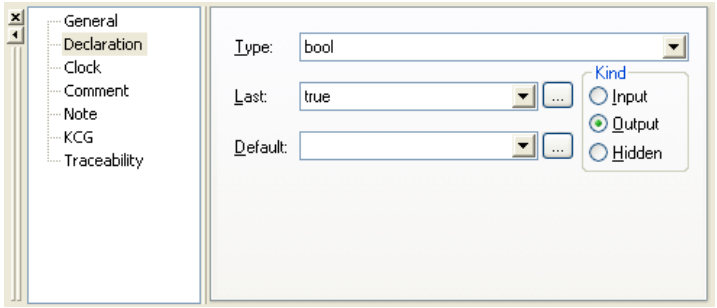
Use transition actions and last values

© 2016 ANSYS, Inc.

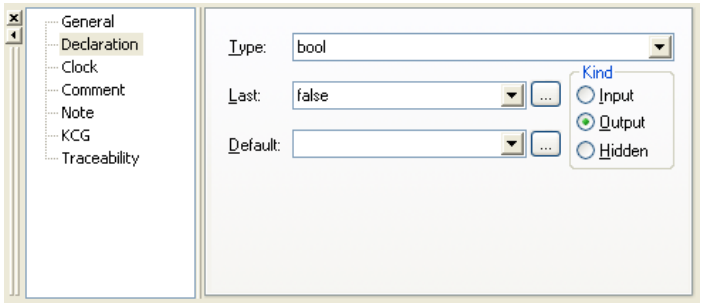
Exercise 9: Solution



`SET_OFF` properties window



`SET_ON` properties window



© 2016 ANSYS, Inc.

Exercise

Exercise 10

Objective

Observe the behavior of State Machine

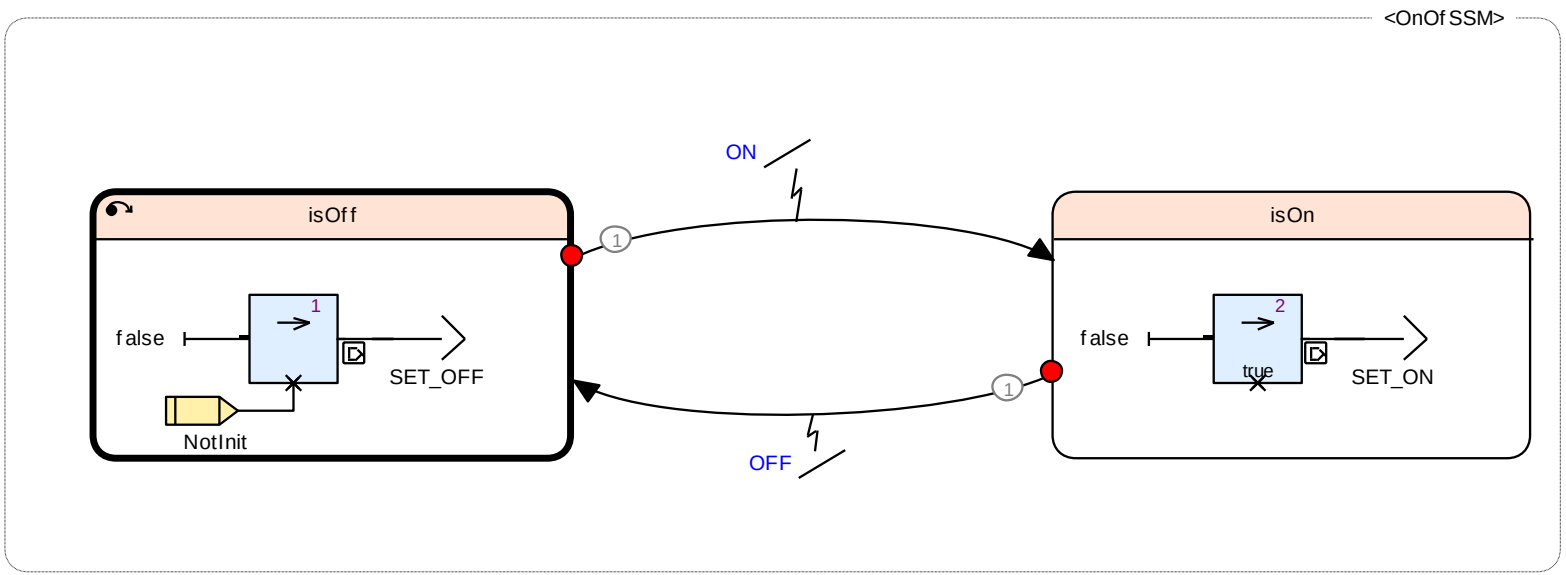
Requirement

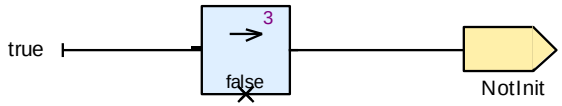
`SET_ON` and `SET_OFF` true only when transition is fired

Use dataflow inside state and the init operator

© 2016 ANSYS, Inc.

Exercise 10: Solution





When no definition is activated the outputs' **default value** is false

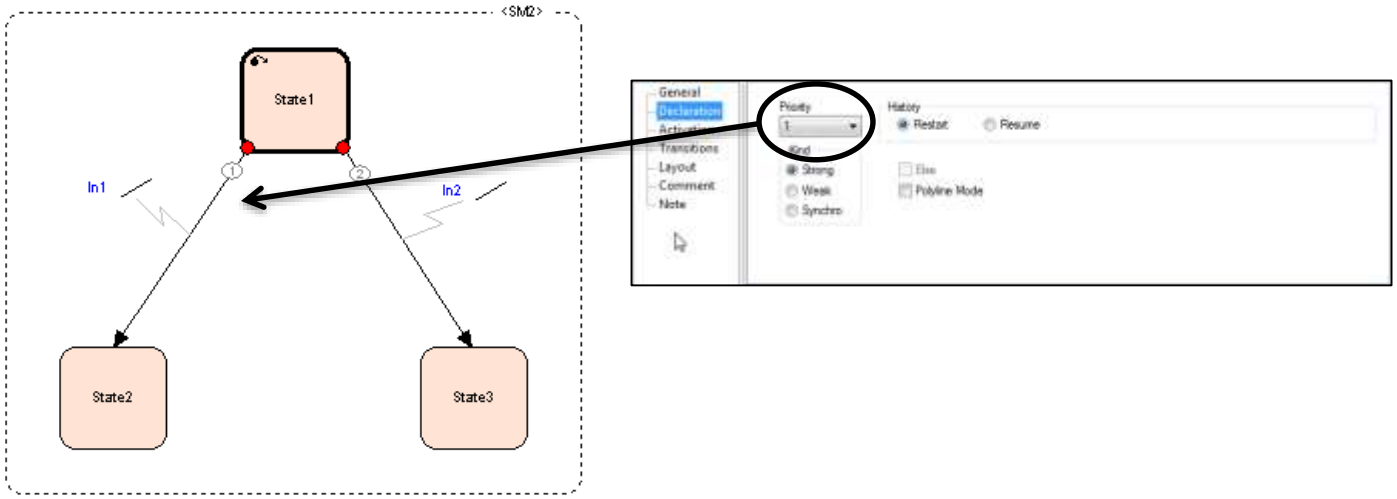
Exercise

© 2016 ANSYS, Inc.

Transition Priorities

When conditions of several transitions starting from the same active state are true, only the one with the highest priority (1 is the highest) is fired

- The priority is mandatory to ensure determinism
- A default value is automatically added by the SCADE Suite Editor, but it can be manually changed by the designer



© 2016 ANSYS, Inc.

Transition Preemption Types

Three types of transition

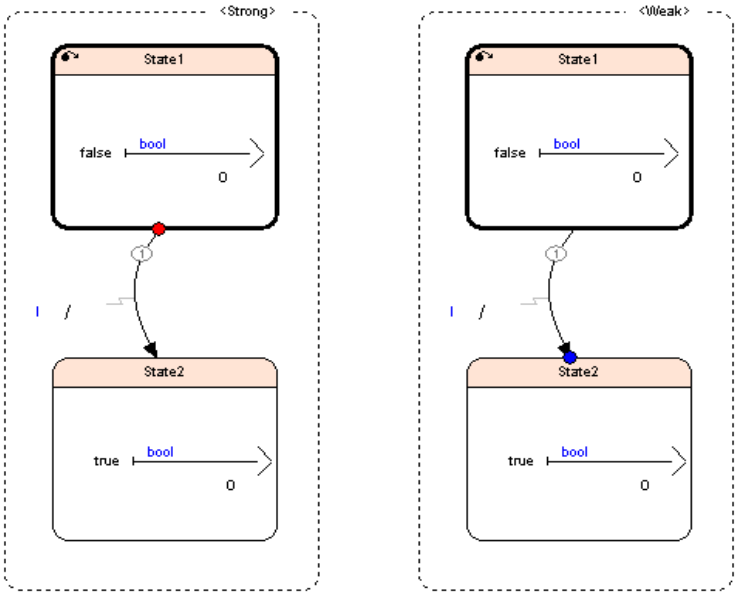
- Strong: preempt the source state and set the target state as active, beginning its internal activity

- Weak delayed: let the source state finish its work during the cycle the transition is triggered. Target state activity is started at the next cycle

- Synchro: triggered on special condition (detailed after).
  Target state activity is started at next cycle (like weak)

 © 2016 ANSYS, Inc.

Strong and Weak Preemption



I	false	true	–
STRONG	O = false	O = true	O = true
WEAK DELAYED	O = false	O = false	O = true

© 2016 ANSYS, Inc.

Exercise 11

Exercise

Objective

Using weak preemption transitions

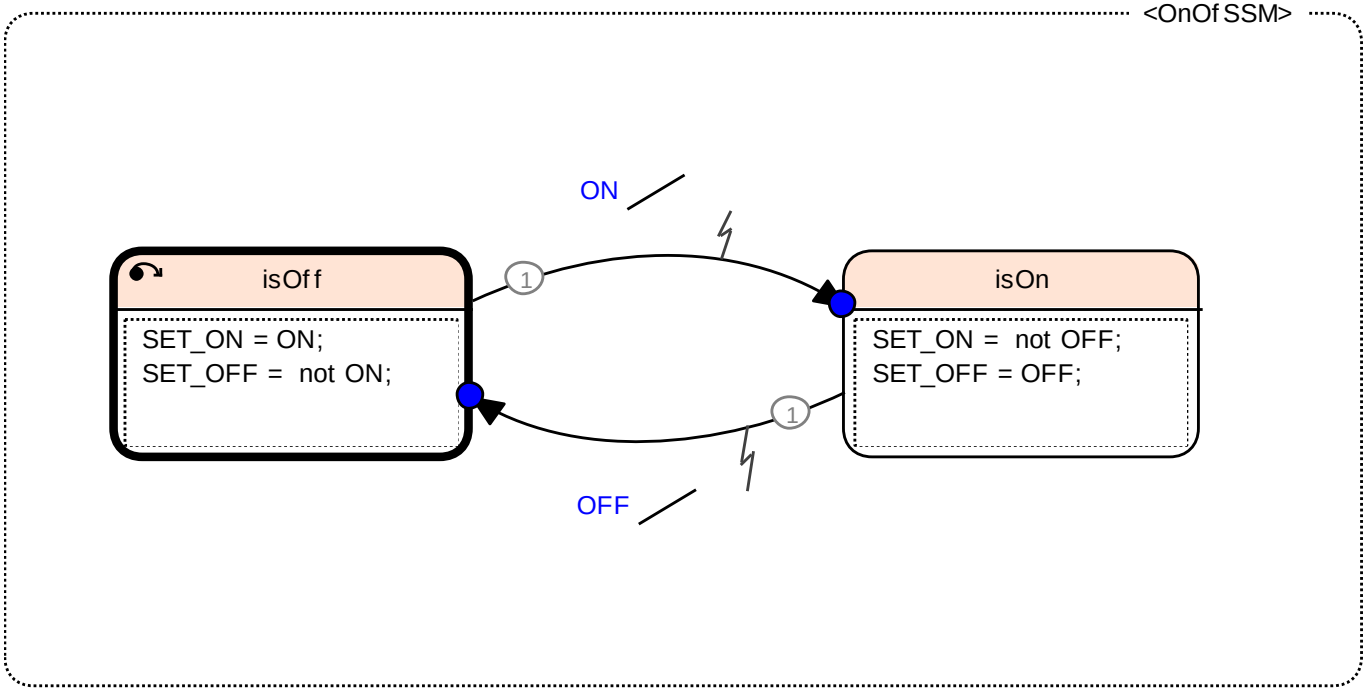
Requirement

- The outputs `SET_ON` shall be true immediatelly when the input `ON` is true, and stay true until the input `OFF` is true

- The outputs `SET_OFF` shall be true immediatelly when the input `OFF` is true, and stay true until the input `ON` is true

© 2016 ANSYS, Inc.

Exercise 11: Solution



Exercise

© 2016 ANSYS, Inc.

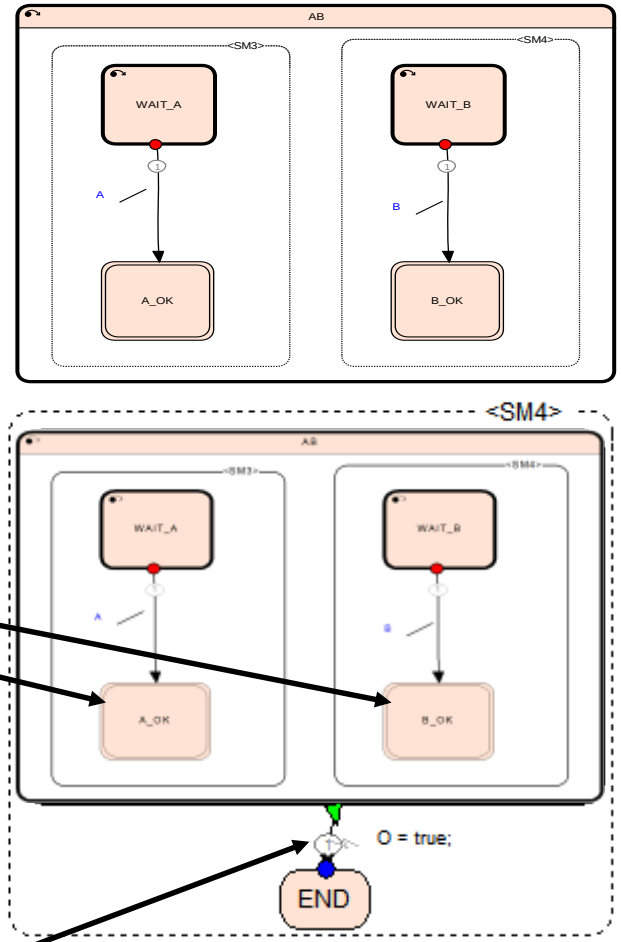
Synchro Transition

Synchro transitions start with a green triangle

A synchro transition is fireable when all the final states in the source state are activated

Transitions can be defined out a final state

An effect action can be specified

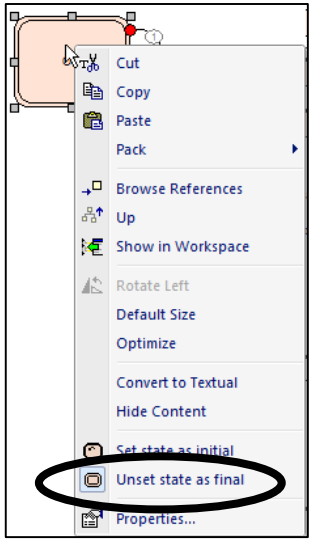


© 2016 ANSYS, Inc.

Final State

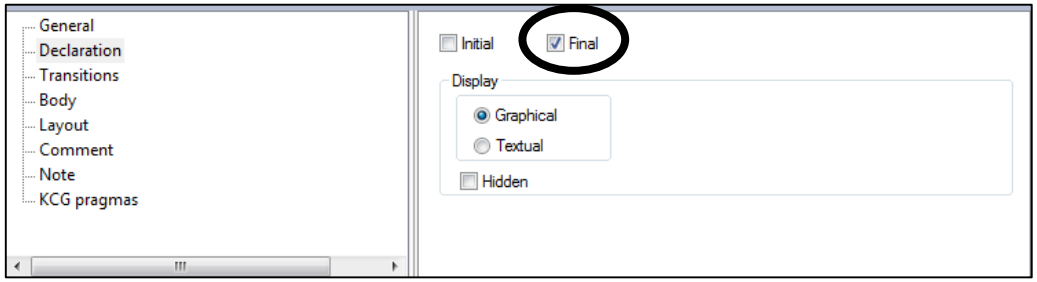
Graphically depicted with a doubled outline

Used to mark a state for synchronization

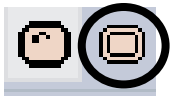


How to set a state as final ?

- From its contextual menu: “Set/Unset state as final”
- From is Properties window: “Final” checkbox



- From the State Machine toolbar dedicated button

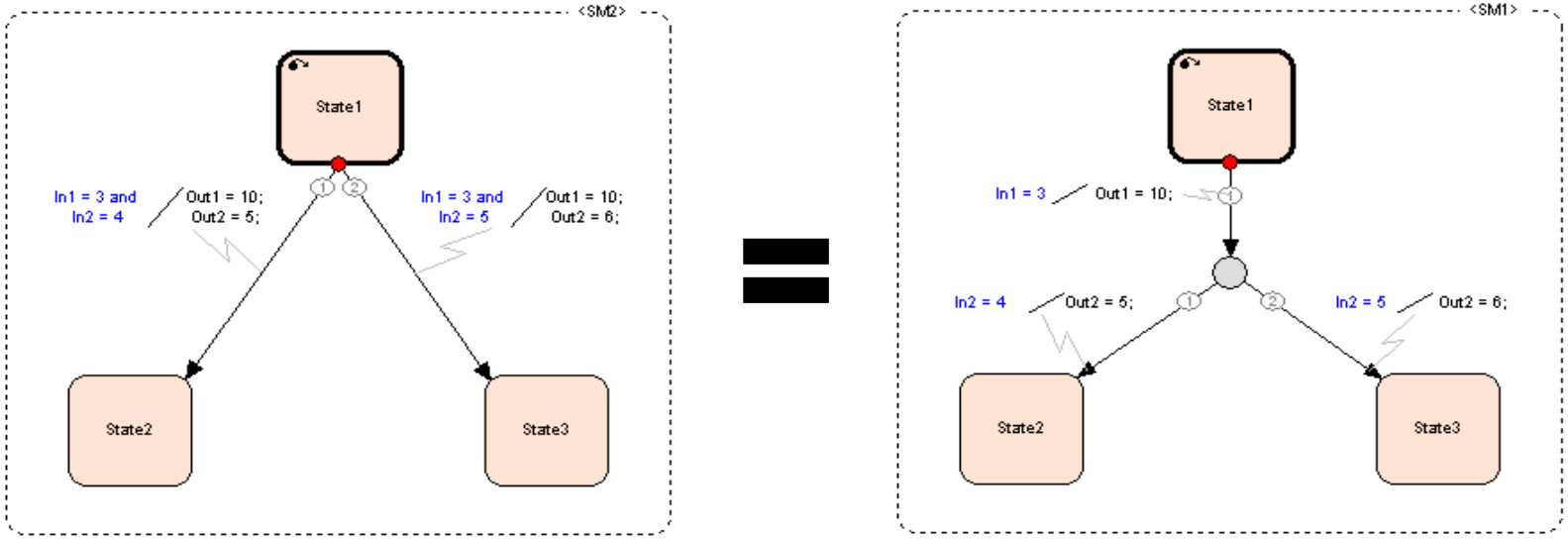


 © 2016 ANSYS, Inc.

Fork Transition

Fork transitions merge transitions with a common part in the condition:

- Common actions can also be merged on the common branch
- Actions are executed only on the fired transition
- The transition is only fired if a new destination state can be reached
- The else branch is not mandatory

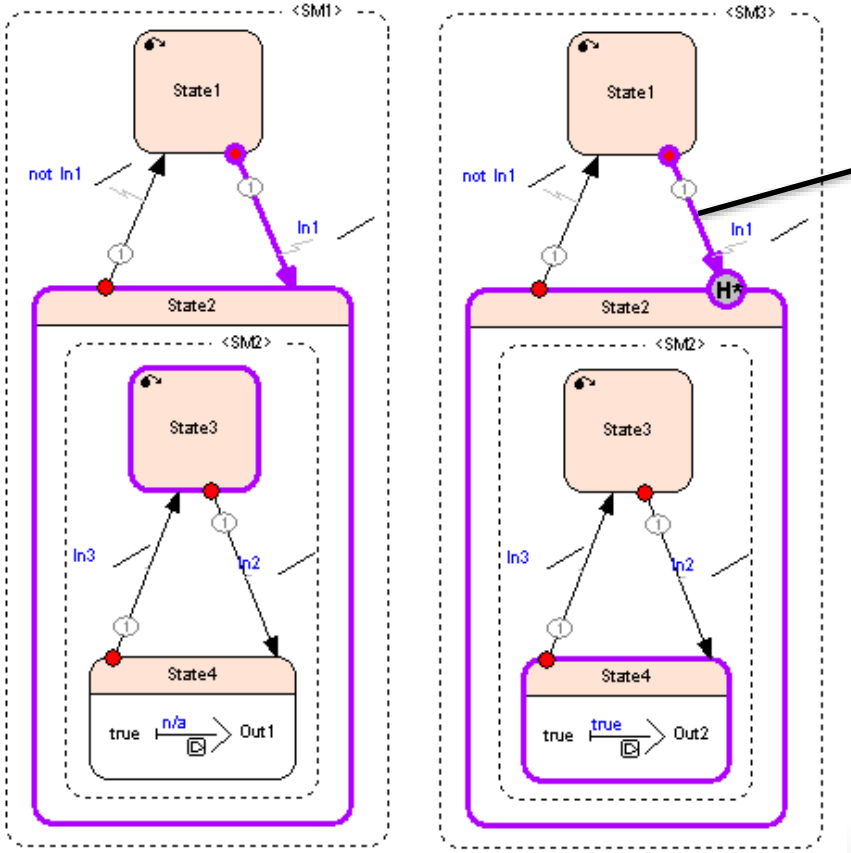


 © 2016 ANSYS, Inc.

History Transition

By default, a transition restarts its target state

But it is possible to resume the target state



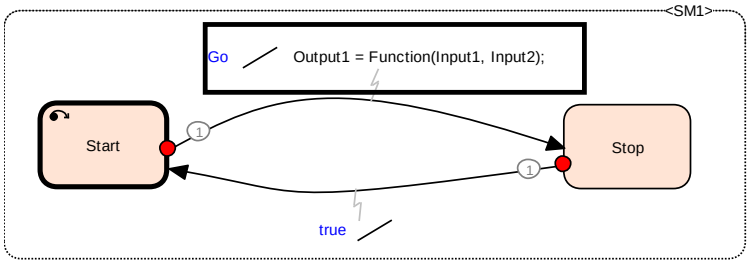
All memory (PRE, FBY, initial states) within the state hierarchy are either reset to their initial values or resumed from the last cycle the state was active

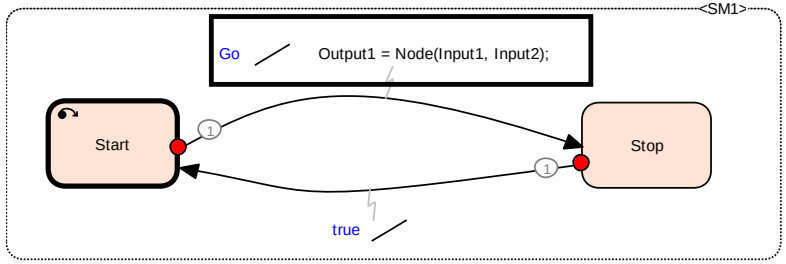
© 2016 ANSYS, Inc.

Calling Operators in Actions

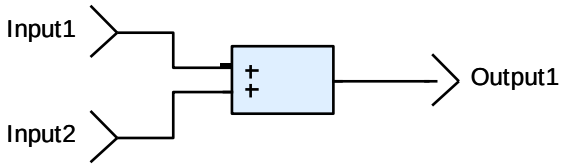
Functions and nodes are accepted as actions in transitions. Memories of nodes are restarted each time the transition is triggered.

These following operators are equivalent:

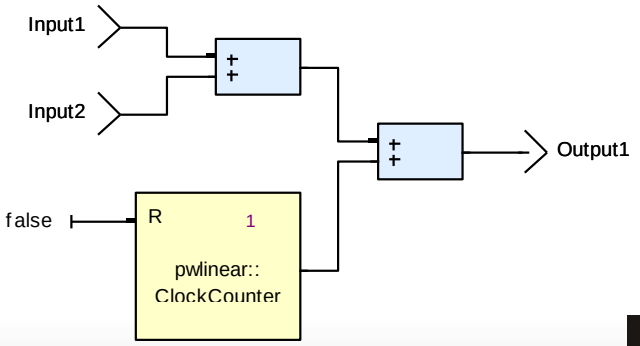




Function



Node

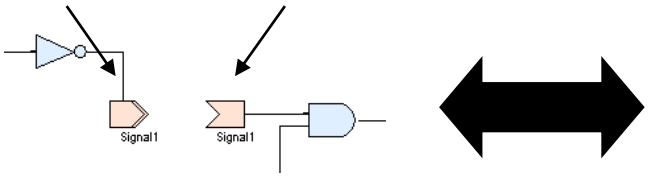


© 2016 ANSYS, Inc.

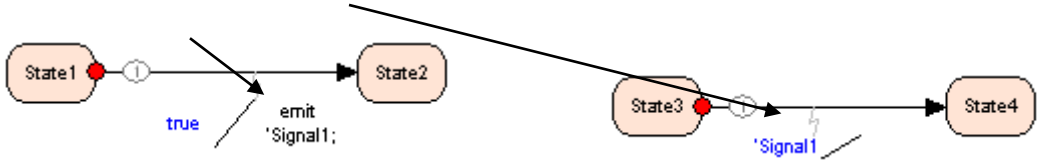
Signals

Signal usage type:

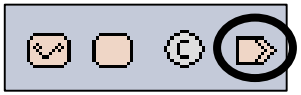
- Emit or Present



- Emit or Present



Create signals:

- Using the New Signal button 
- Using the contextual menu Insert > Signal in Scade view

© 2016 ANSYS, Inc.

Signal Emission

Emitted signals are visible inside the context (operator, branch, state) in which the signal is defined.

A signal can be emitted several times in a given cycle.

In a given cycle, if a signal is emitted at least once then it is present in that cycle.

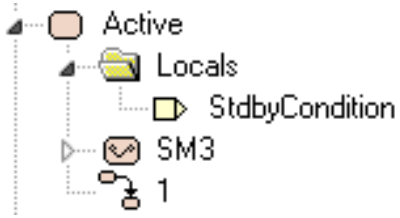
© 2016 ANSYS, Inc.

Communication and Scope

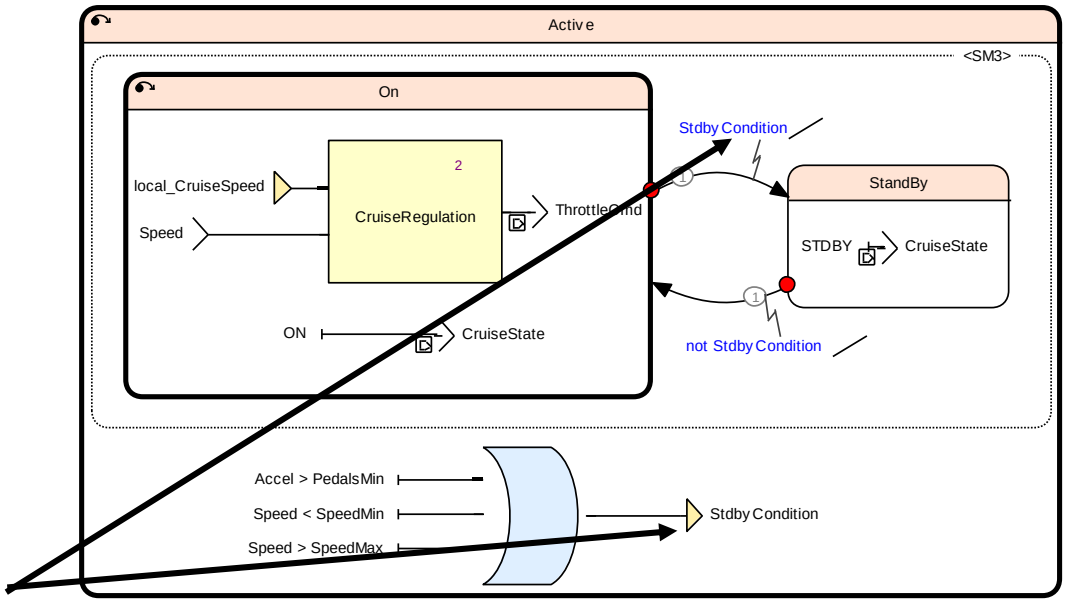
Inputs and Outputs are visible throughout the operator they are defined in.

Signals and Locals are visible throughout the scope they are defined in:

- Operator
- State



StdbyCondition is only defined in the Active state and its hierarchy

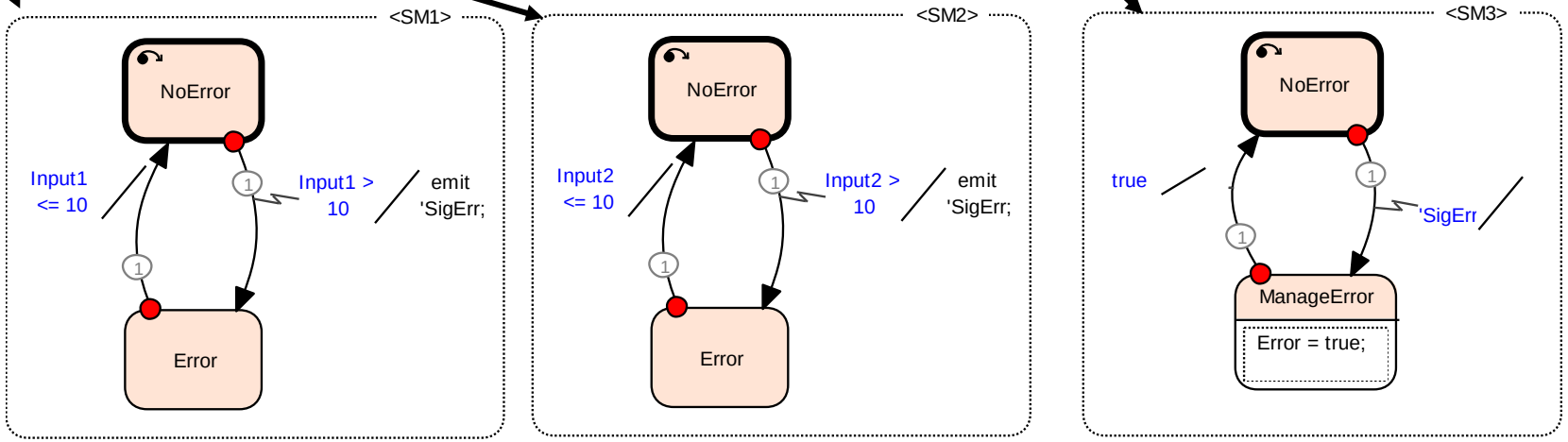


© 2016 ANSYS, Inc.

Communication and Scope

Signals are useful to catch a specific situation in several SMs:

- A signal is emitted in several parallel SMs when conditions are met
- A parallel State Machine waits for the presence of the signal to respond to the event



© 2016 ANSYS, Inc.

State Machine: Causality Loops

Causality loops are:

- Cyclic dependencies of flow computation
- Or a mix of State/Transition execution and flow computation

Causality loops are detected by KCG and so by the SCADE Suite Checker

The error message indicates the list of flows, states, and transitions linked to the cyclic dependency

© 2016 ANSYS, Inc.

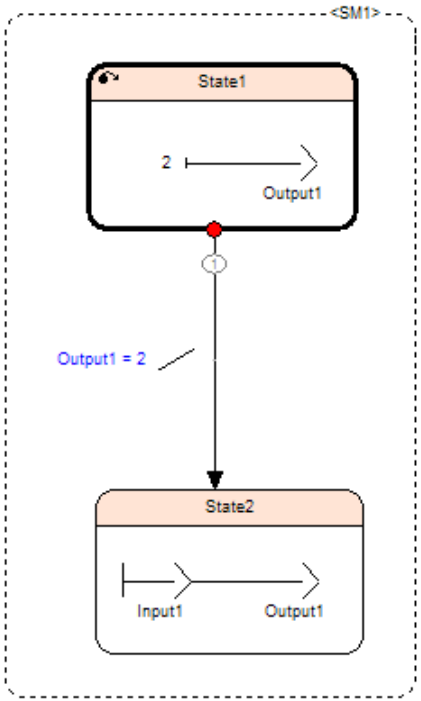
State Machine: Causality Loops

Example

Transition from State1 to State2 is triggered if Output1 = 2

Output1 is set to 2 by State1 exactly when the transition is not fired

Hence, transition firing depends on State1 execution, which depends on the transition



The Model Checker checks if such problems exist:

1 error(s) detected - 0 warning(s) detected

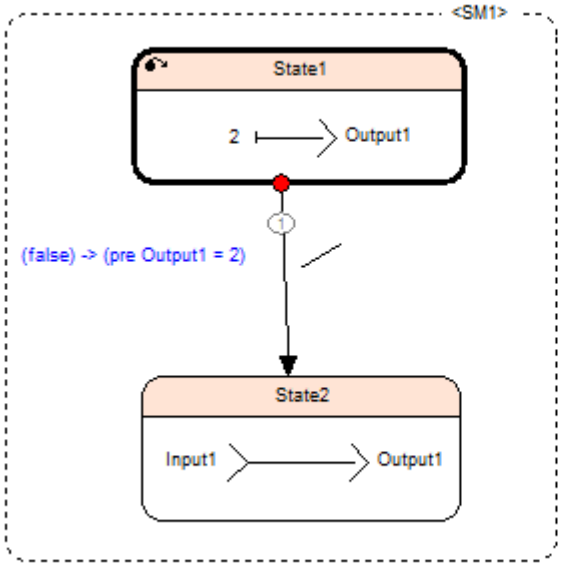
Category	Code	Message
Semantic Error	ERR_400	**Causality at** Package1::CyclicSSM/SM1:State1: the definition of the strong guards of state State1 depends on flow Output1 ; (Package1::CyclicSSM/SM1:State2:Output1=) the definition of flow Output1 depends on the state of automaton SM1 via the control context ; (Package1::CyclicSSM/SM1:State1:) the definition of the state of automaton SM1 depends on the strong guards of state State1 ;

© 2016 ANSYS, Inc.

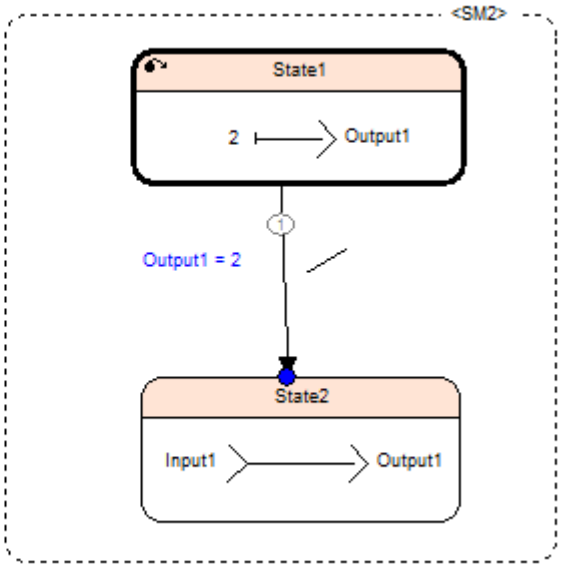
Causality Loops

Causality loops can be solved by the introduction of additional memory to break the cycle

Memory on one of the flows using pre or last



Memory on a transition using the weak delayed transition



© 2016 ANSYS, Inc.

SCADE Suite Checker

Semantic errors may be introduced while designing models and must be corrected before users can ever use a model for simulation or code generation

The SCADE Suite Checker analyses the selected items based on the active configuration set in the Code Generator toolbar:

Note: see details related to these options in the code generation section

© 2016 ANSYS, Inc.

SCADE Suite Checker

Select an operator from the Scade view and *Right-click > Check*

or

Select Check on the Code Generator toolbar

 © 2016 ANSYS, Inc.

Lab 5 (1/2)

Use Lab Support p.65-74

Objective:

Implement the `CruiseControl` (CC) operator:

- Use of states machines, mixed data and control flows

Requirements:

Time: 35 min

Close all SCADE suite sessions and load a copy of "*Prerequisites\Lab 5\CruiseControl\CruiseControl.etp*"

The `CruiseControl` operator calls:

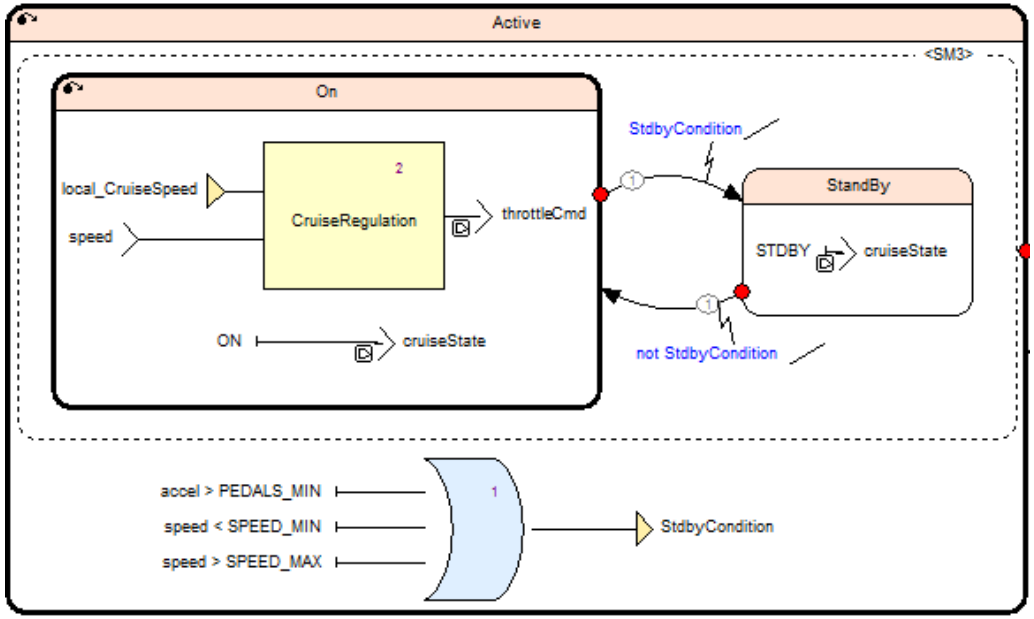
- The `CruiseRegulation` operator computes the throttle command
- The `CruiseSpeedMgt` operator manages the value of the cruise speed according to the driver commands when the Cruise Control is enabled

Lab

© 2016 ANSYS, Inc.

Lab 5 (2/2)

Modelling of the system when the Cruise Control is active (ON, STDBY states) in the Active state



Set default value of output

Output	Default Value
throttleCmd	accel
cruiseState	OFF

© 2016 ANSYS, Inc. February 1, 2017 ANSYS Confidential

Lab 6 (1/5)

Use Lab Support p.76-85

Objective:

Simulate the `SystemSimul` operator which integrates both subsystems, the car and the `CruiseControl` operators to model the complete system

Time: 30 min

Requirements:

The `SystemSimul` operator calls:

- The `CruiseControl` operator developed in the previous exercise
- The `CarModel` operator from the `Car` library

Lab

© 2016 ANSYS, Inc.

Lab 6 (2/5)

Lab

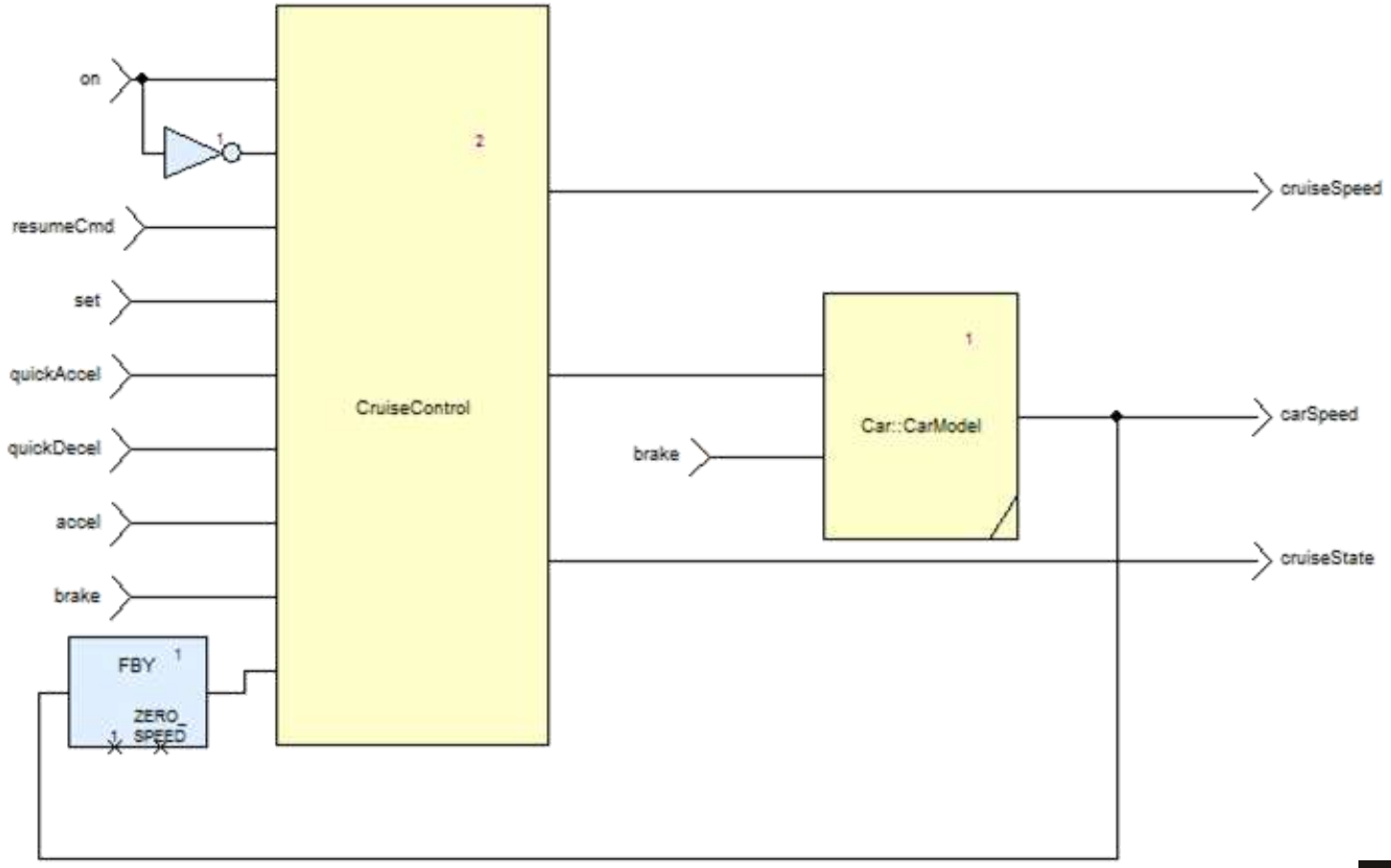
Create the interface of `SystemSimul` operator

Name	Kind	Type
on	Input	bool
resumeCmd	Input	bool
Set	Input	bool
quickAccel	Input	bool
quickDecel	Input	bool
accel	Input	CarType::tPercent
brake	Input	CarType::tPercent
cruiseSpeed	Output	CarType::tSpeed
cruiseState	Output	CruiseControl::tCruiseState
carSpeed	Output	float64

© 2016 ANSYS, Inc.

Lab 6 (3/5)

Design the `SystemSimul` diagram



© 2016 ANSYS, Inc.

Lab 6 (4/5)

Lab

Simulate the scenario 1

Cycle duration	Input values	Output values to check
1	on = false resumeCmd = false set = false quickAccel = false quickDecel = false accel = 0.0 brake = 0.0	-
2		cruiseSpeed = CarSpeed = 0.0 cruiseState = CruiseControl::OFF
5	Accel 50.0	cruiseState = cruiseControl::OFF

Save the scenario1 (.sss format)

© 2016 ANSYS, Inc. February 1, 2017 ANSYS Confidential

Lab 6 (5/5)

Lab

Simulate the scenario 2 (after a reset of the simulation)

Cycle duration	Input values	Output values to check
1	-	-
1	`accel = 100.0`	-
1	`on = true`	-
1	-	`cruiseState=CruiseControl::STDBY`
12	`brake = 10.0`	`cruiseState=CruiseControl::INT`
1	`Brake = 0.0` `resumeCmd = true`	`cruiseState = CruiseControl::STDBY`
1	`accel = 1.0`	`cruiseState = CruiseControl::ON`

Save the scenario 2 (.sss format)

You can reload and replay your scenarios

ANSYS

© 2016 ANSYS, Inc.

Contacts

Legal Contact
Esterel Technologies SAS
14/15, Place Georges Pompidou
78180 Montigny Le Bretonneux
FRANCE
Phone: +33 1 30 68 61 60
Fax: +33 1 30 68 61 61

Technical Support
Esterel Technologies SAS
Parc Avenue - 9 rue Michel Labrousse
31100 Toulouse FRANCE
Phone: +33 5 34 60 90 50
Fax: +33 5 34 60 90 41

Submit questions to Technical Support: scade-support@ansys.com
Contact one of our Sales representatives at: scade-sales@ansys.com
Direct general questions about Esterel Technologies to: scade-info@ansys.com
Discover the latest news on our products and technology at: http://www.ansys.com/products/embedded-software

Legal Information

Copyrights ©2017 ANSYS, Inc. All rights reserved. ANSYS®, SCADE®, SCADE Suite®, SCADE Display®, SCADE Architect®, SCADE LifeCycle® are trademark or registered trademarks of ANSYS, Inc or its subsidiaries in the U.S. or other countries. All other trademarks and trade names contained herein are the property of their respective owners.