*Realize Your Product Promise®*

Model-Based Design
with
**SCADE Suite®**

# Training Activities

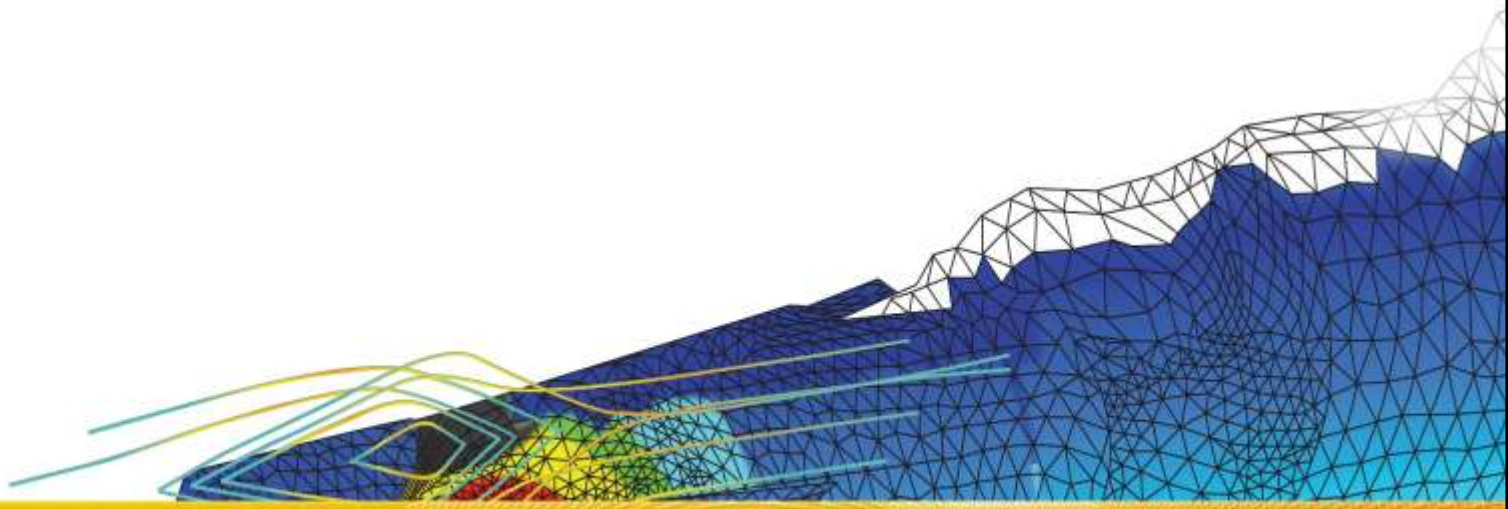
## Day 2

ANSYS

ANSYS®

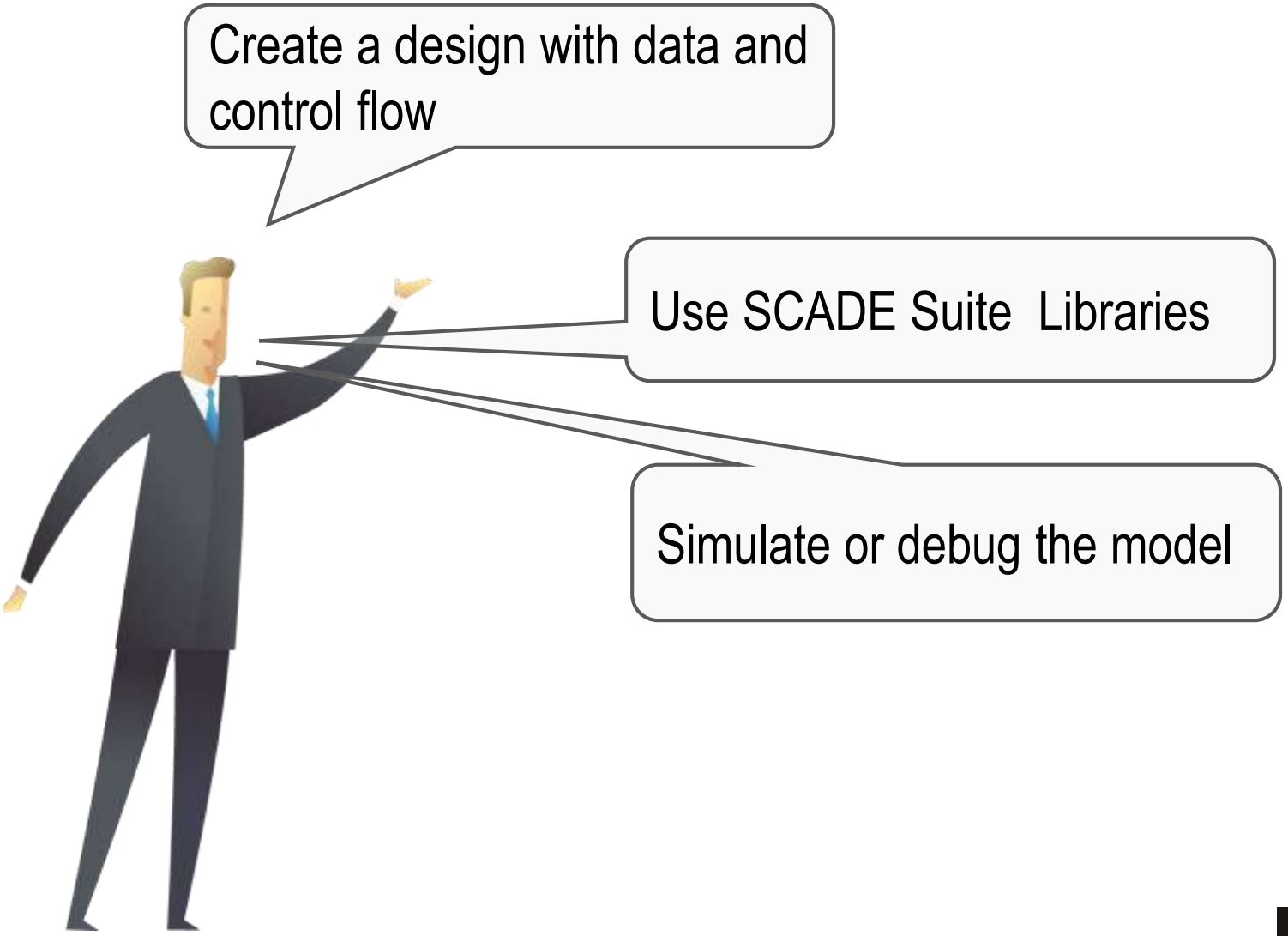
# Model-Based Design with SCADE Suite

Lab Support Day 2



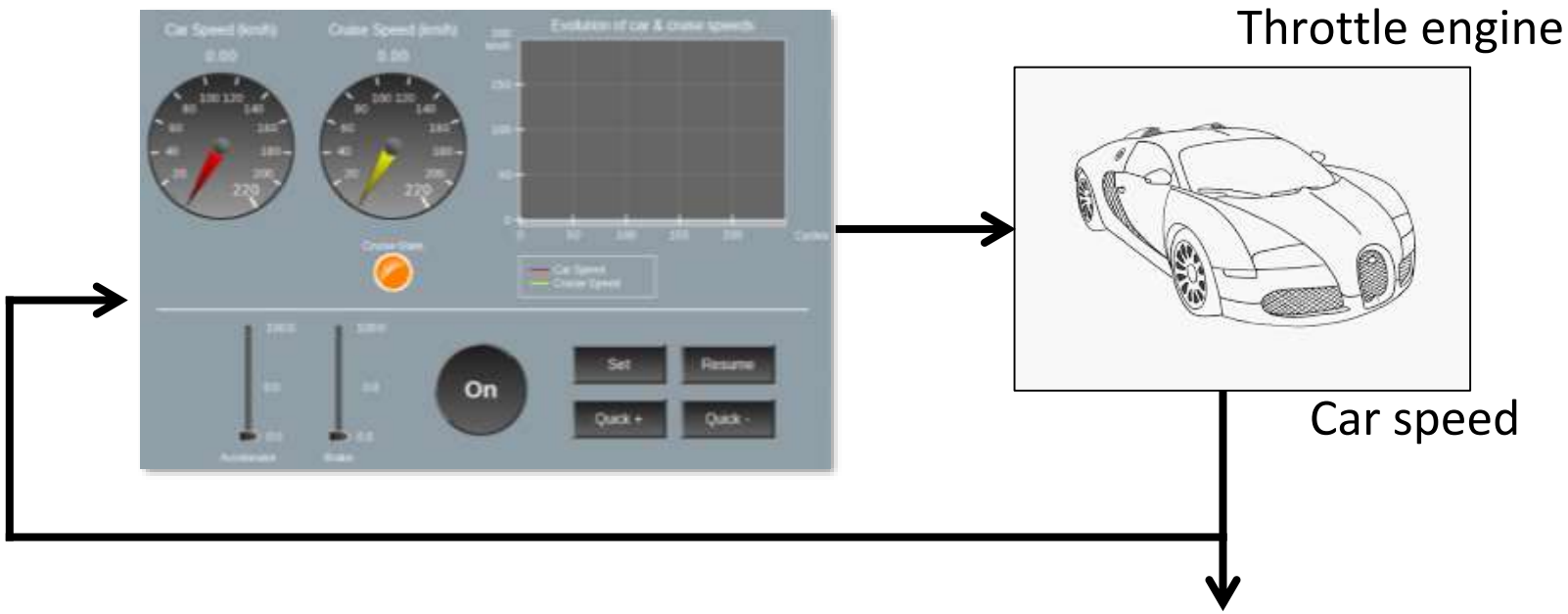
# Lab Objectives

Lab



# The System to create

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




Lab Prerequisite

# The System to create

Build a system which designs a Cruise Control driving the speed of a car with mixed data and control flows.

The system contains two parts:

- A Car representation calculating the speed of the car depending on the information given by the driver and the Cruise Control
- The Cruise Control subsystem, based on states machines, that computes the engine throttle to maintain a constant cruise speed depending on the driver's commands and the current vehicle speed

Both subsystems will be integrated together:

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

Lab Prerequisite

# Prerequisites

The design is done with a bottom-up approach.

This choice is only driven by the will to design and simulate  parts of the system as early as possible.

SCADE Suite does not require any specific design approach. Top down or mixed approaches are often used as well.

**Notes:**

When we write CruiseControl, we refer to the “CruiseControl” operator / model or system, depending on the context.

Req or req. is the abbreviation of requirement.

Lab Prerequisite

# 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

# Lab 1

# Lab 1 (1/2)

Lab

## Objective:

Create a new type and new contants

## 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 1 (2/2)

Use Lab Support p.13-25

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

Time: 20 min

| 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 |

Lab

# Lab 1: Solution

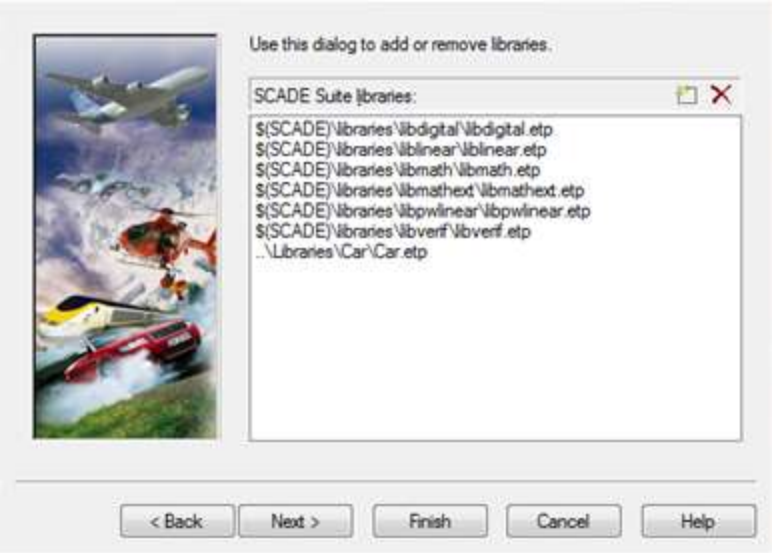
Lab Solution

Open SCADE Suite, …\SCADE<X>\SCADE\bin\VCS.EXE, where X represents the product version such as R17

Create a new SCADE Suite project:

- Set the project name, CruiseControl

Add the Car.etp library located at Q:\Labs\Prerequisites\Lab 1\Libraries\Car



# Lab 1: Solution

Look at the CruiseControl project.

Show that the car representation is provided and implemented into the Car library

The Car library is defined into two libraries as follows:

- The car design, located at the Car package and stored in Q:\Labs\Prerequisites\Libraries\Car\Car.etp
- The Scade types used in the car model, located at the CarType package stored in Q:\Labs\Prerequisites\Libraries\CarType\CarType.etp:
  - These types are also used in the whole application

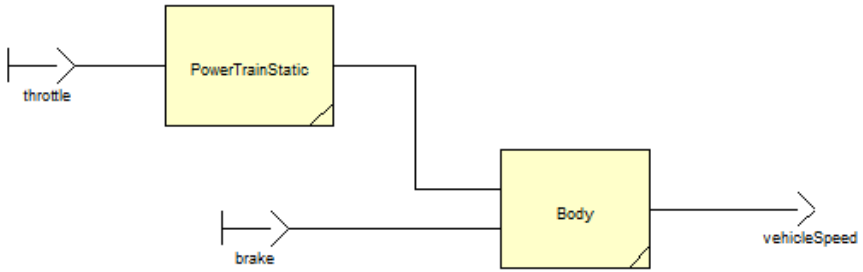
Lab Solution

# Lab 1: Solution

Lab Solution

Look at the car design:

- CarModel Operator. It is a simplified car model:
  - It calculates the speed of the car in the Body operator by integrating the car acceleration
  - The acceleration is computed from the engine torque, the brake effect and the car load at the given speed (to simplify, the car rolls on a flat road)
  - The engine torque is calculated from the throttle command and the engine speed, into the PowerTrainStatic operator




CarModel Operator

# Lab 1: Solution

Lab Solution

Car model types:

- The Scade types used by the car design are defined as follows:

| Name | Definition | Comments and Information |
|---|---|---|
| tPercent | float64 | Percentage |
| tRpm | float64 | Rotations per minute |
| tSpeed | float64 | Speed |
| tTorq | float64 | Torque |

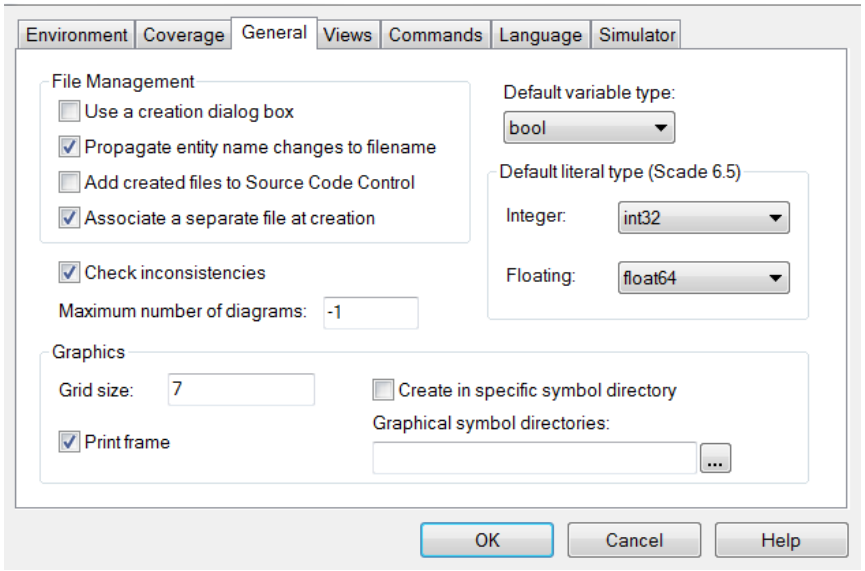
- These types are also used in the whole application.

# Lab 1: Solution

Lab Solution

Change several general options of the SCADE Suite session:

- *Tools > Options*… menu
- The Options dialog is displayed
- Go to the General tab and check following options:




- Propagate entity name to filename:
  - When the name of the package operator is changed, the filename is replaced by the new name
- Associate a separate file at the creation:
  - The content of a new package / operator is stored into a different file

# Lab 1: Solution

Read the Cruise Control Interface and parameters requirements:

- CC_HLR_OUT_01 to CC_HLR_OUT_03
- CC_HLR_CCP_01 to CC_HLR_CCP_07

Create the CruiseControl package

**Tip: Once a SCADE Suite object is declared, to rename it, either:**

- Type directly the new name in the Scade view
- Or use the F2 key
- Or modify the Name field from the General tab of Properties

Lab Solution



# Lab 1: Solution

Lab Solution

Implement the user's types defined by the requirements:

- According to CC_HLR_OUT_01 and CC_HLR_CCP_01 requirements, need the type named tSpeed (float64): it is declared into the CarType library
- According to CC_HLR_OUT_02 and CC_HLR_CCP_06 requirements, need the type named tPercent (float64): it is declared into the CarType library
- According to CC_HLR_OUT_03 requirement, need of teCruiseState type (enumeration: OFF, INT, STDBY, ON): it must be declared by the user

Declare the user type named teCruiseState:

- Select the CruiseControl package from the Scade view
- Click on the New Type icon (Create toolbar) to declare a type:
  - It will be created in the Types folder under the CruiseControl package
  - Rename the new type to teCruiseState

# Lab 1: Solution

Lab Solution

Set the definition of teCruiseState type:

- Select teCruiseState from the Scade view and double-click on it
- The Types view opens: it is a table with Type, Definition and Comment columns
- Rename a type in the Type column, set its definition in the Definition column and add a comment in the Comments column:
  - Select the cell and press the F2 key to edit and modify the content
  - Choose <enumeration> as teCruiseState definition from the drop-down list

# Lab 1: Solution

Lab Solution

Show that a definition element, value1, is automatically added:

- Rename it into OFF (select the cell and press F2 key)
- Declare other definition elements as defined below:
  - Select a cell in the table and click on the New Definition Element icon
  - Rename each definition element according to the enumeration values:

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

# Lab 1: Solution

Lab Solution

Implement the Constants defined by the Cruise Control parameters requirements as follows:

| 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 |

# Lab 1: Solution

Lab Solution

Select the CruiseControl package from the Scade view.

Declare a constant:

- Click on the New Constant icon (Create toolbar) :
  - Created in the Constants folder under the CruiseControl package

- Or open the Constants view:
  - Double-click on a constant in the Scade view
  - The Constants view opens: a table with Constant, Type, Definition and Comment columns
  - Rename a constant, set its type and value in the associated columns as in the Types view:
    - Select the cell and press the F2 key to edit and modify it

# Lab 1: Solution

Lab Solution

Rename the new constants as defined in the table:

| 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 |

Once selected, set its type and value from the Declaration tab of the Properties:

- Select the type and the value from the drop-down list or write them directly
- The syntax of a floating value is a.b, in SCADE Suite (integer a and unsigned integer b): e.g. for 30 write 30.0

# Lab 2

# Lab 2

Lab

## Objective:

Define the architecture operators

Time: 10 min

## Requirements:

Create the 4 following operators

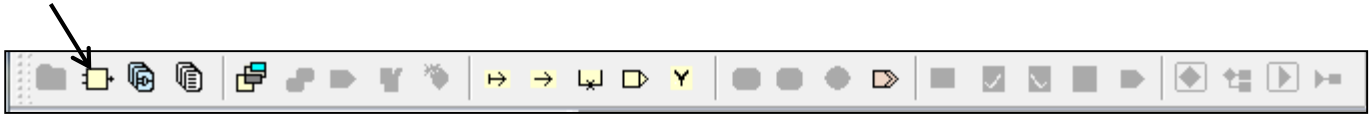
| Operators | Package | Comments |
|---|---|---|
| `CruiseRegulation` | `CruiseControl` | manages the vehicle speed when the Cruise Control is active |
| `CruiseSpeedMgt` | `CruiseControl` | manages the value of the cruise speed according to the driver commands |
| `CruiseControl` | `CruiseControl` |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

# Lab 2: Solution

Lab Solution

Create the CruiseRegulation function:

- Select the CruiseControl package from the Scade view and click on the New Operator icon to create a user operator:
  - It is created in the Operators folder under the CruiseControl package



- Rename the new user function into CruiseRegulation
- Set it as Function in the Declaration properties

Redo the same previous steps to declare SaturateThrottle, CruiseSpeedMgt, CruiseControl nodes and SystemSimul root node (SystemSimul will be created in the System package that is located at the same level of CruiseControl package)

ANSYS

# Lab 3

# Lab 3 (1/4)

Lab Support p.26-44

## 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

# 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

# Lab 3 (3/4)

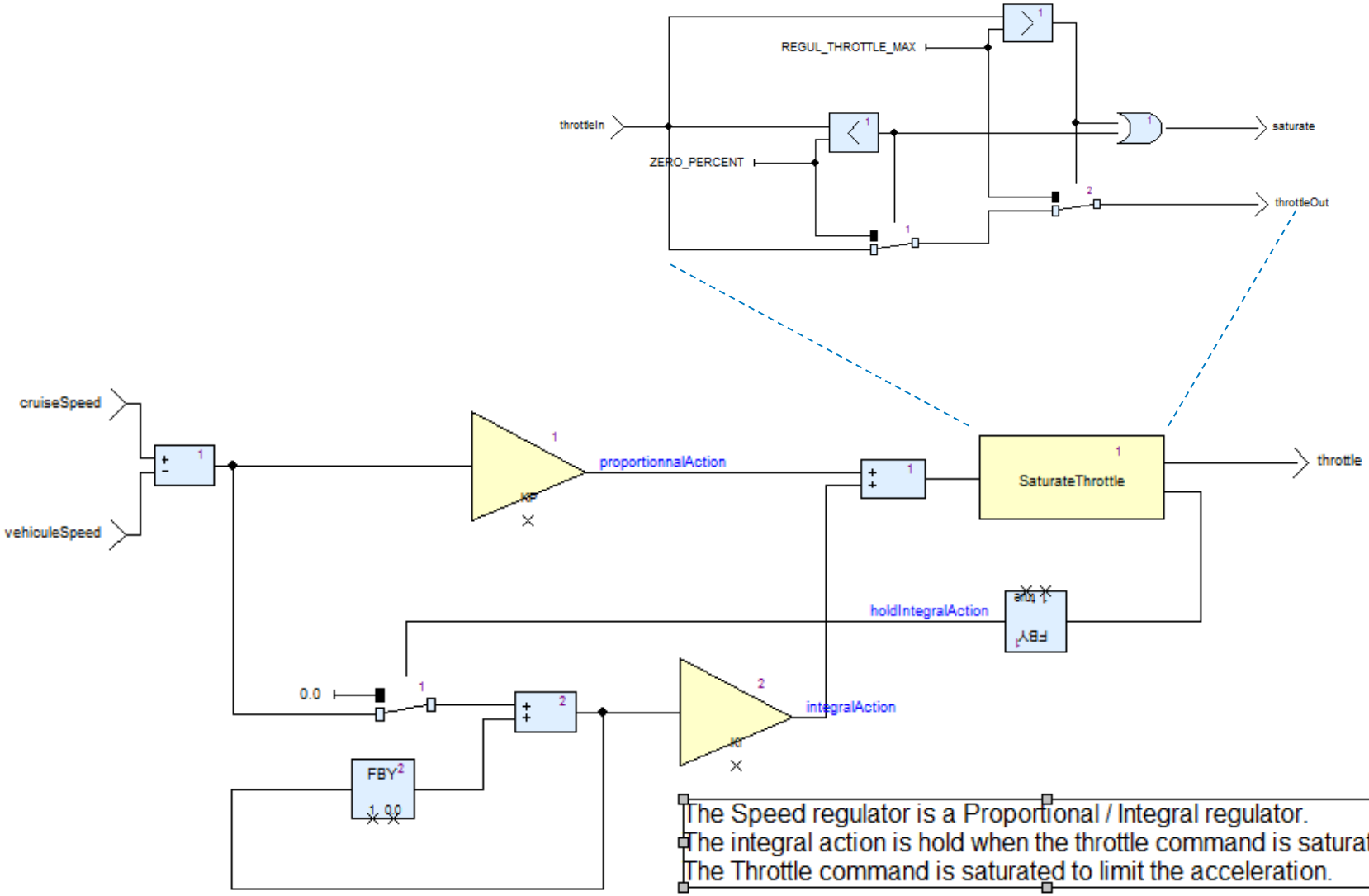
Create the `CruiseRegulation` operator *(requirement CC_HLR_CDC_01 to 05)*

| 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

# Lab 3 (4/4): CruiseRegulation



# Lab 3: Prerequisite

Lab Prerequisite

Implement the CruiseRegulation function:

- Read the Car driving control requirements (CC_HLR_CDC_01 to CC_HLR_CDC_05)
- According to the CC_HLR_CDC_03 requirement, the regulation is operated by a proportional and integral algorithm, with Kp and Ki factors:

Throttle= Kp*ε_t + Ki$\int_0^t$ ε_$t$ dt;  where ε_t = CruiseSpeed_t  – VehicleSpeed_t

Throttle= Kp*(CruiseSpeed_t - VehicleSpeed_t)  + ∑_0^t (Ki*(CruiseSpeed_t  – VehicleSpeed_t)

- The regulation is frozen when the throttle output is saturated

# Lab 3: Solution

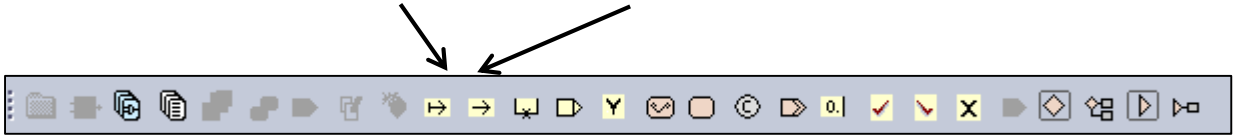
Lab Solution

Declare the interface of CruiseRegulation operator:

- According to Car driving control requirements (CC_HLR_CDC_01 to CC_HLR_CDC_05), declare the following I/Os:

| Name | Type | Direction |
|---|---|---|
| cruiseSpeed | CarType::tSpeed | Input |
| vehicleSpeed | CarType::tSpeed | Input |
| throttle | CarType::tPercent | Output |

- Select the CruiseRegulation operator from the Scade view,
- Click on the New Input or New Output (Create toolbar) to declare respectively one input or one output:

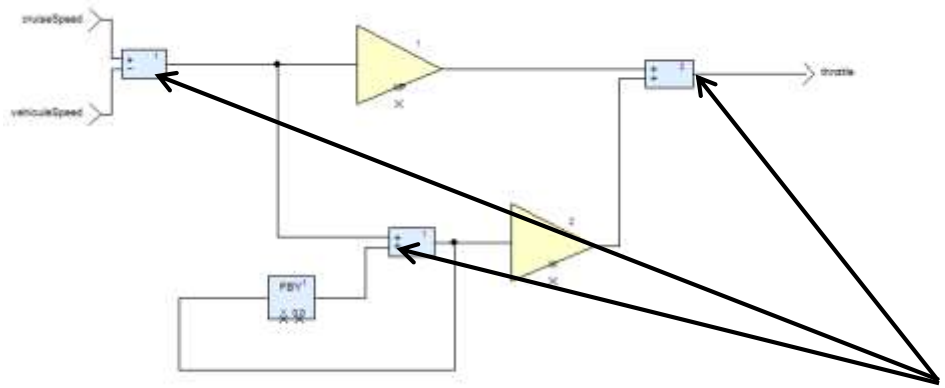


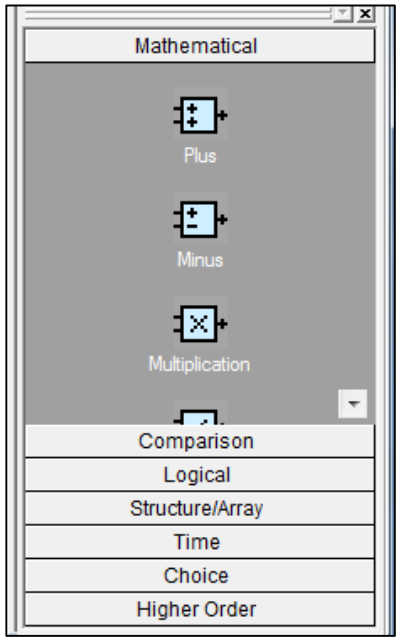
- Rename them and set their type from the Declaration properties.
- Rename their diagram to diagram_ThrottleRegul

# Lab 3: Solution

Lab Solution

Select the CruiseRegulation operator from the Scade view and double-click on the diagram_ThrottleRegul diagram to open it and implement the algorithm as follows:



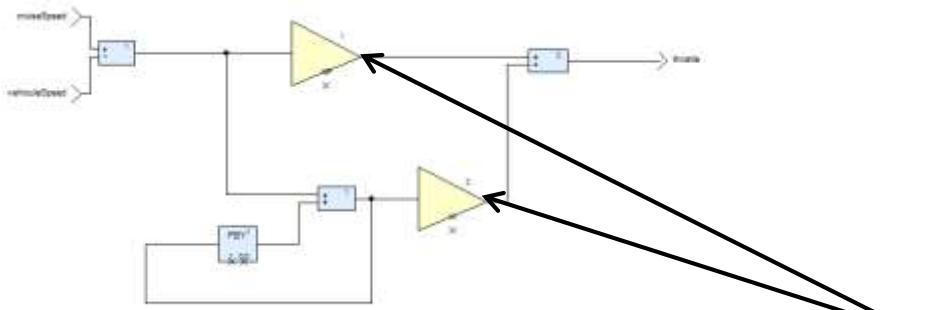



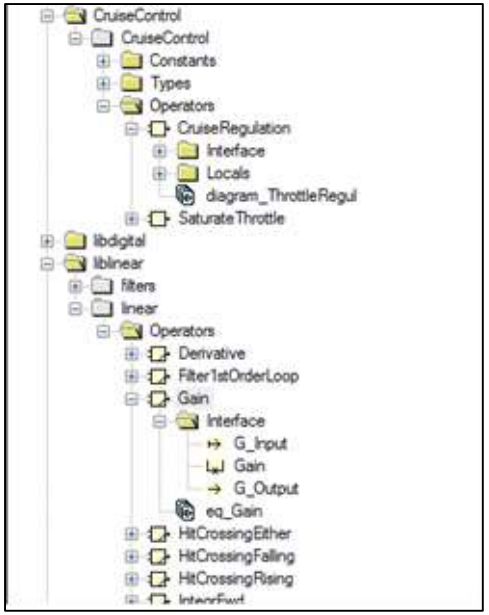

Drag and drop cruiseSpeed, vehicleSpeed, throttle I/Os from the Scade view into the diagram

Drag and drop into the diagram the predefined operators, Plus (twice) and Minus from the Mathematical toolbar (Shortcuts window)

# Lab 3: Solution

Drag and drop two instances of linear::Gain operator from the standard SCADE Suite liblinear library:

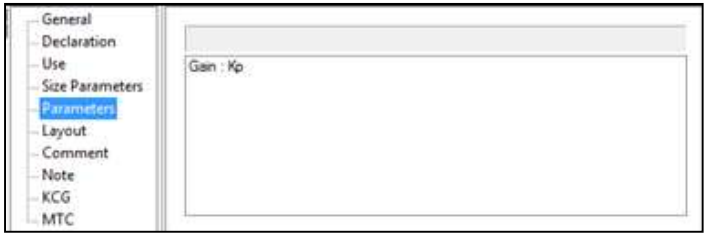





Set Kp and Ki as Gain parameters in the Parameters tab of properties:

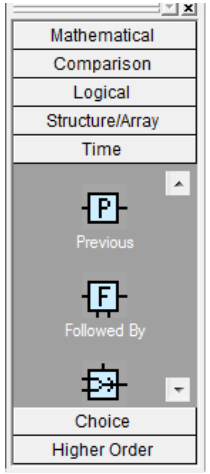
Select the cell and press F2 key to modify it

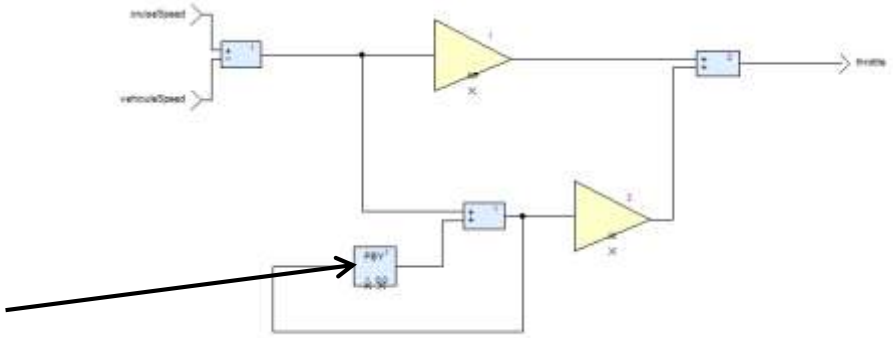
Write directly the parameter or choose it from the drop-down list



# Lab 3: Solution

- To complete the integral part, `∑_0^t (Ki*(CruiseSpeed_t - VehicleSpeed_t)`, implement `Ki*(CruiseSpeed_t - VehicleSpeed_t)`:
- Drag and drop into the diagram, the Followed By operator (FBY) from the Time toolbar to keep the value of the expression at the previous cycle



Use the Parameters tab of properties to set the FBY delay, 1 for one cycle, and the default output value, 0.0, for the first cycle (it is not defined before).

# Lab 3: Solution

Lab Solution

Connect cruiseSpeed, vehicleSpeed, throttle I/Os to the different predefined operators.

Left-click on the I/O when the target icon is displayed and move the mouse to the operator to create the link between them.

# Lab 3: Solution

Lab Solution

Perform a check of the model to verify if the CruiseRegulation operator is semantically correct:
Select the CruiseRegulation operator from the Scade view and *Right-click > Check*,
Show that the following error is raised.

**Result of check for operator CruiseControl::CruiseRegulation/ in model CruiseControl**

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

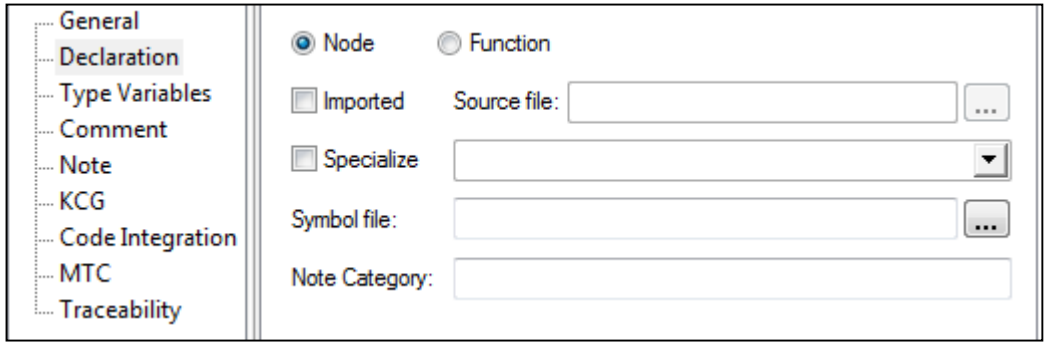
| Category | Code | Message |
|---|---|---|
| Semantic Error | ERR_111 | **Unexpected internal state at** CruiseControl::CruiseRegulation/<br>Function CruiseRegulation has an internal state<br>(it should probably be declared as a node) |

# Lab 3: Solution

Lab Solution

Resolve the unexpected internal state error:

- The CruiseRegulation operator needs a memory as it keeps the value of an expression at the previous cycle so it must be declared as a node and not as a function
- Select the operator and check the Node field from the Declaration tab of properties

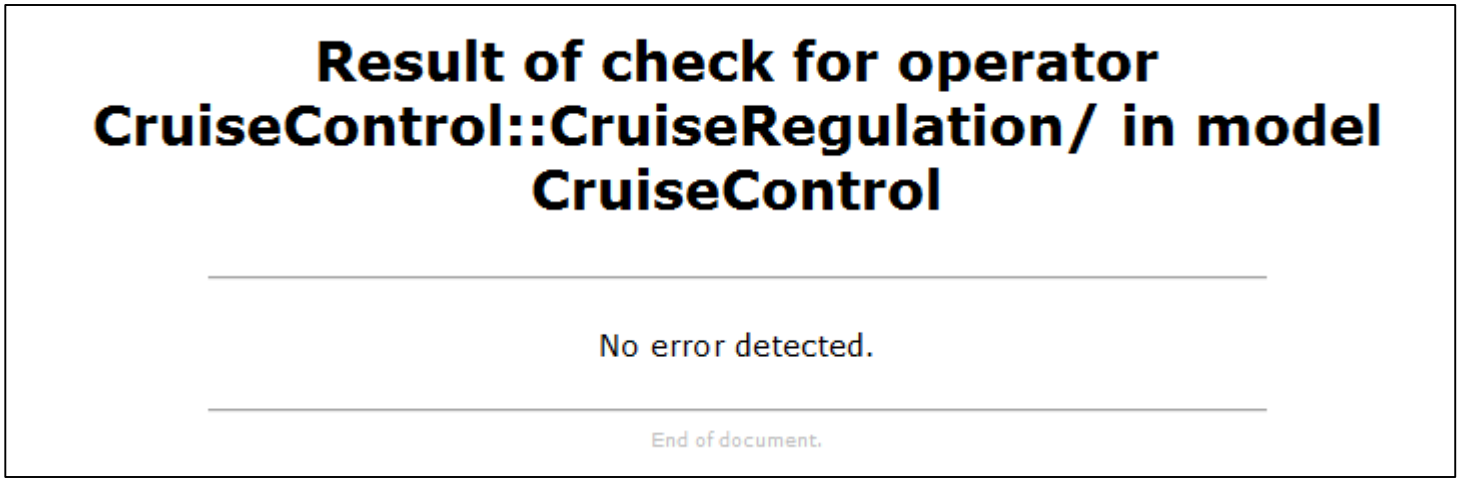


# Lab 3: Solution

Lab Solution

Perform again a check of the model to verify that no error is raised:

- Select the CruiseRegulation operator from the Scade view and *Right-click > Check*
- Show that no error is displayed



# Lab 3: Solution

- According to CC_HLR_CDC_05 req., the throttle output must be saturated at REGUL_THROTTLE_MAX during the regulation.
- It will be limited to ZERO_PERCENT when throttle < ZERO_PERCENT:

```
A = if (ThrottleIn < ZERO_PERCENT) then ZERO_PERCENT
else ThrottleIn;
B = if (ThrottleIn > REGUL_THROTTLE_MAX) then
REGUL_THROTTLE_MAX else A;
ThrottleOut = B;
```

Where ThrottleIn is the value of Throttle before the verification and ThrottleOut, its final value.

Implement a new operator named SaturateThrottle to perform this action, it will be called into the CruiseRegulation operator.

Lab Solution

# Lab 3: Solution

- According to C_HLR_CDC_04 req., the integral part must be reset when the CruiseControl is going on, and frozen when Throttle is saturated:

A boolean output named saturate will allow to know if the throttle is saturated, the test will be added in the SaturateThrottle operator:

```
Saturate = (ThrottleIn > REGUL_THROTTLE_MAX) or
(ThrottleIn < ZERO_PERCENT)
```

Lab Solution

# Lab 3: Solution

Lab Solution

Implement the SaturateThrottle operator:

```
A = if (ThrottleIn < ZERO_PERCENT) then ZERO_PERCENT
else
ThrottleIn;
B = if (ThrottleIn > REGUL_THROTTLE_MAX) then
REGUL_THROTTLE_MAX
else A;
ThrottleOut = B;
```

Select the SaturateThrottle operator from the Scade view and declare its required I/Os:

| Name | Type | Direction |
|---|---|---|
| throttleIn | CarType::tPercent | Input |
| throttleOut | CarType::tPercent | Output |
| saturate | bool | Output |

- Open the diagram_SaturateThrottle_1 diagram and drag and drop from the Scade view into the diagram:
  - throttleIn, throttleOut and saturate I/Os
  - ZERO_PERCENT and REGUL_THROTTLE_MAX constants

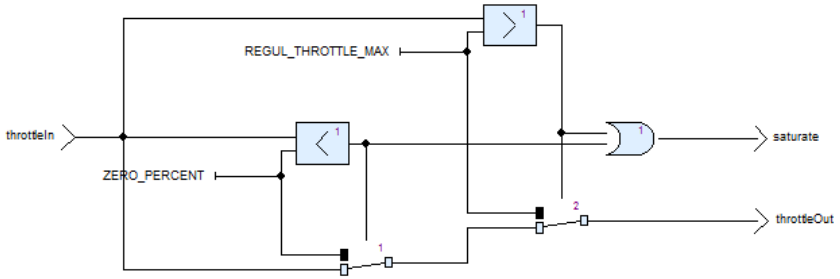
# Lab 3: Solution

Drag and drop the following predefined operators (Shortcuts window):

Strictly Less Than and Strictly Greater Than from the Comparison toolbar

Or from the Logical toolbar

If … Then …Else from the Choice toolbar




```
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 3: Solution

Lab Solution

Connect throttleIn, throttleOut, saturate I/Os, ZERO_PERCENT and REGUL_THROTTLE_MAX constants to the different predefined operators.

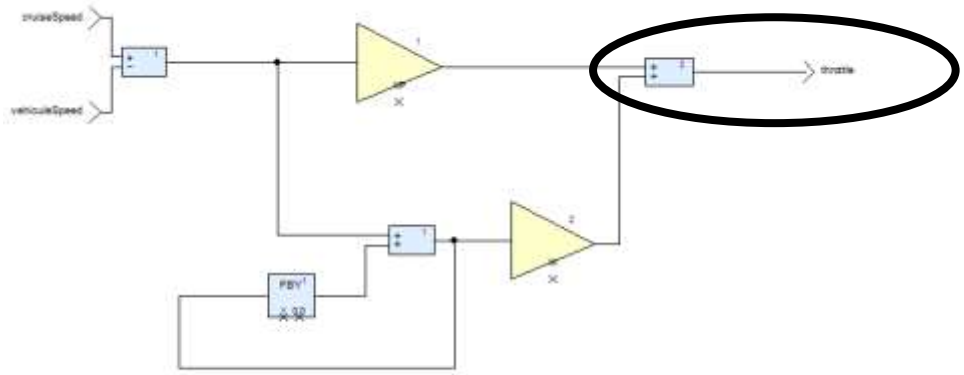
Left-click on the I/O  or constant when the target icon is displayed and move the mouse to the operator to create the link between them.

# Lab 3: Solution

Lab Solution

Open diagram_ThrottleRegul (CruiseRegulation diagram):

- Move throttle to the right side to have some space to connect it with the SaturateThrottle operator
- Drag and drop the SaturateThrottle operator into the diagram between the Plus operator and Throttle to create a link between them, to saturate the throttle: throttle will be connected to throttleOut



# Lab 3: Solution

Lab Solution

Use the second output of SaturateThrottle operator, saturate, to protect the algorithm against the overshoot of its integral part:

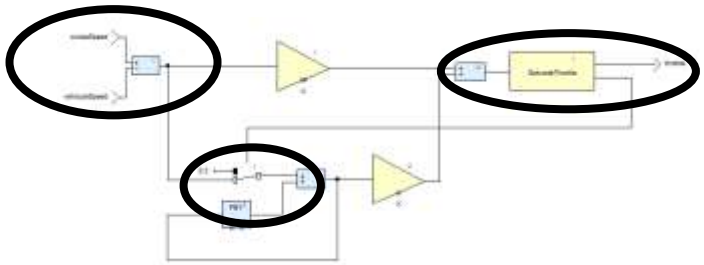
- if saturate then Integral = 0.0 else

  Integral = cruiseSpeed_t – vehicleSpeed_t
- IntegralAction = $\sum_0^t \mathrm{Ki}(\mathrm{CruiseSpeed\_t} - \mathrm{VehicleSpeed\_t})$

Drag and drop If … Then …Else operator (Choice toolbar) into the diagram,

Set saturate as the If boolean condition and 0.0 as the If output value,

Set cruiseSpeed_t – vehicleSpeed_t as the Else output value.



# Lab 3: Solution

Lab Solution

Perform a check of the model to verify if the CruiseRegulation operator is always semantically correct:

- Select CruiseRegulation operator from the Scade view and *right-click > Check*
- Show that a causality error is raised

**Result of check for operator CruiseControl::CruiseRegulation/ in model CruiseControl**

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

| Category | Code | Message |
|---|---|---|
| Semantic Error | ERR_400 | **Causality error at CruiseControl::CruiseRegulation/ L40=** the definition of flow _L41 depends on flow _L4 ; ( CruiseControl::CruiseRegulation/ L4= ) the definition of flow _L4 depends on flow _L31 ; ( CruiseControl::CruiseRegulation/ L31= ) the definition of flow _L31 depends on flow _L34 ; ( CruiseControl::CruiseRegulation/ L34= ) the definition of flow _L34 depends on flow _L42 ; ( CruiseControl::CruiseRegulation/ L42= ) the definition of flow _L42 depends on flow _L41 ; |

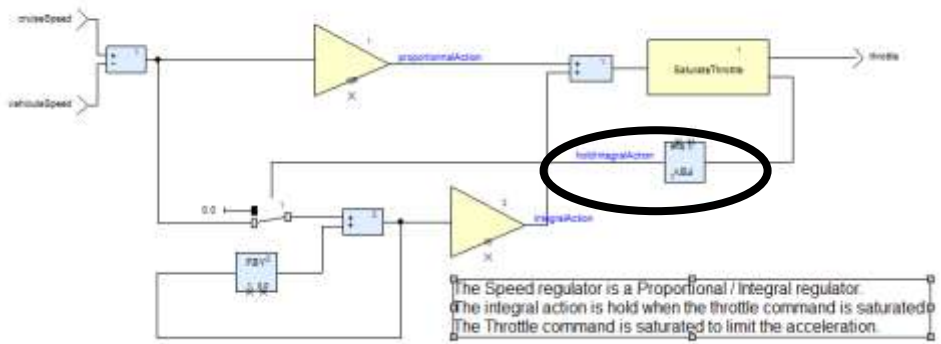
ANSYS

# Lab 3: Solution

Lab Solution

Resolve the causality error:

- The causality analysis aims to verify that no flow instantaneously depends on itself:
- In the current case, Saturate output is used to produce a value consumed in the computation of throttleIn, and as input of the SaturateThrottle operator:
  - Hence the flows between throttleIn and saturate are not instantaneously independent:




- To solve the problem:
  - To connect Saturate as condition of If… Then…Else, use a FBY with a delay of one cycle and take true as default value of the Saturate output
  - Perform again a check of the model to verify that no error is raised

# Lab 4

# Lab 4 (1/3)

Lab

## 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 4 (2/3)

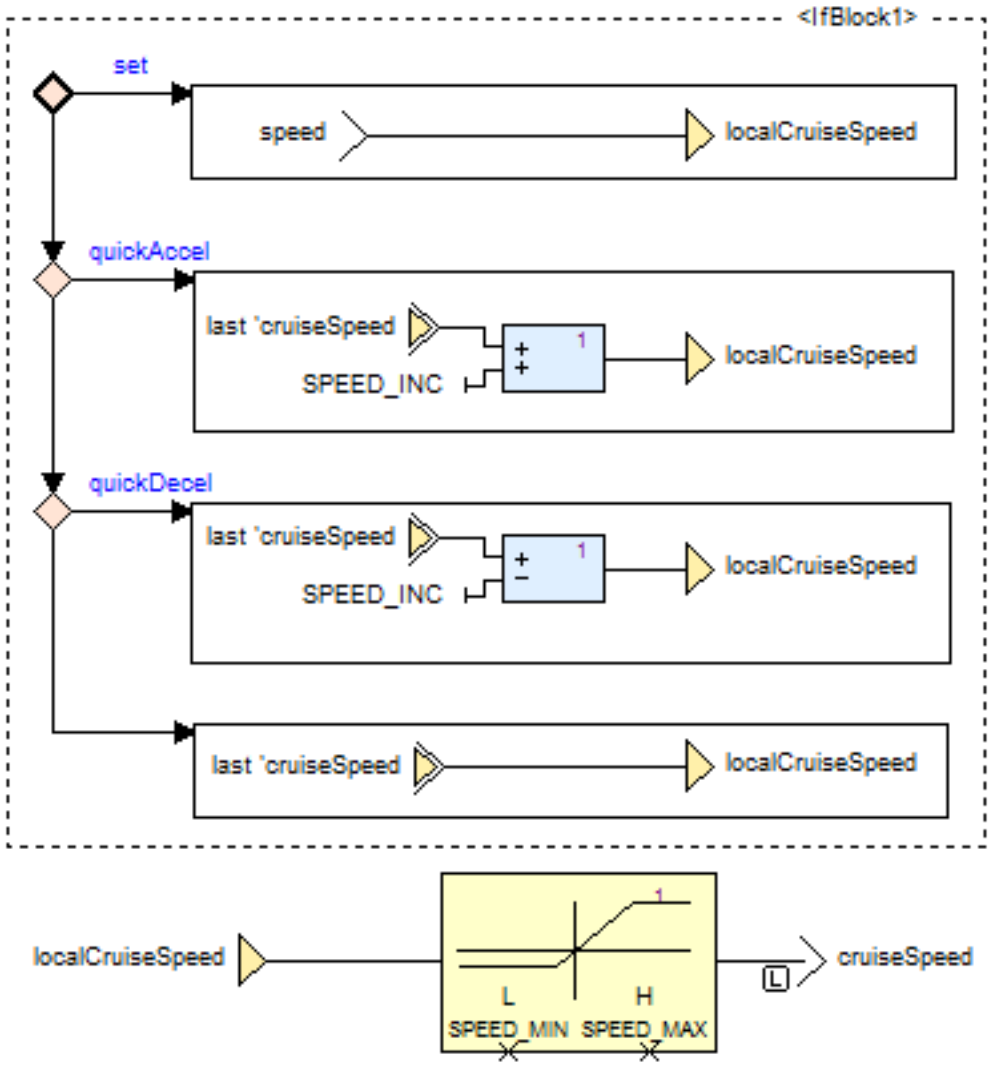
In package CruiseControl, create `CruiseSpeedMgt` operator

| 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)*

Lab

# Lab 4 (3/3): CruiseSpeedMgt



# Lab 4: Prerequisite

Read the Cruise speed management requirements

- CC_HLR_CSM_01 to CC_HLR_CSM_05

Design the CruiseSpeedMgt operator and its interface:

- Create the conditional Block inserting an IfBlock

Implement CC_HLR_CSM_02 to CC_HLR_CSM_04.

Implement CC_HLR_CSM_05: check that the cruise speed is well maintained between SpeedMin and SpeedMax km/h

- Use a limiter operator such as pwlinear::LimiterUnSymmetrical from the libpwlinear library

Lab Prerequisite

# Lab 4: Solution

Lab Solution

Implement the CruiseSpeedMgt operator:

- Read the Cruise speed management requirements (CC_HLR_CSM_01 to CC_HLR_CSM_05)
- The CruiseSpeedMgt operator manages the value of the cruise speed according to the driver commands (Set or QuickAcccel or QuickDecel buttons pressed) when the CruiseControl is enabled.

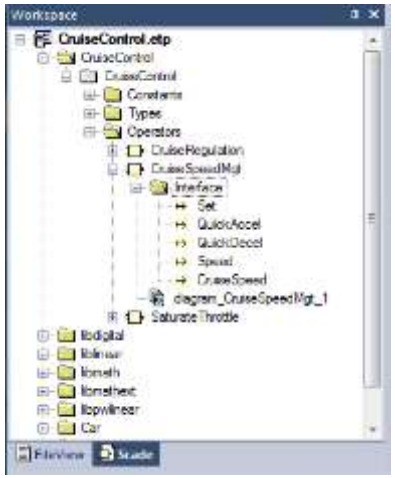
# Lab 4: Solution

Lab Solution

Check the interface of CruiseSpeedMgt operator:

- According to CC_HLR_IN_01 to CC_HLR_IN_09 interface requirements and CC_HLR_CSM_01 to CC_HLR_CSM_05 cruise speed management requirements, following I/Os are declared:

| Name | Type | Direction |
|---|---|---|
| Set | bool | Input |
| QuickAccel | bool | Input |
| QuickDecel | bool | Input |
| Speed | CarType::tSpeed | Input |
| CruiseSpeed | CarType::tSpeed | Output |



# Lab 4: Solution

Lab Solution

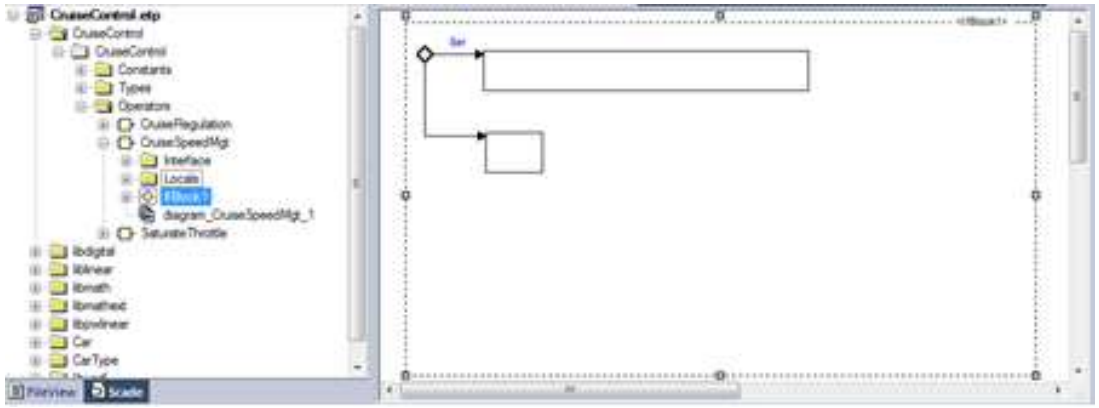
Create the conditional Block inserting an IfBlock:

- Double-Click on the diagram_CruiseSpeedMgt_1 diagram from the Scade view to open it
- Click on the New IfBlock icon (Create toolbar) to declare one IfBlock

# Lab 4: Solution

Lab Solution

Select the shape (boundary) and increase the content size of IfBlock1 to insert other If Node branches:



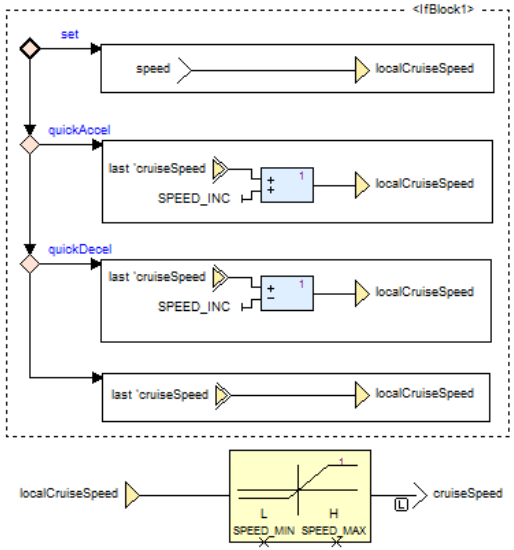

It needs four branches (CC_HLR_CSM_02 to CC_HLR_CSM_04 req.):

- 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

# Lab 4: Solution

The user must check that the cruise speed is well maintained between SpeedMin and SpeedMax km/h (CC_HLR_CSM_05 requirement) and integrate this verification into the diagram with the conditional Block:

- Use of a limiter operator such as pwlinear::LimiterUnSymmetrical from the libpwlinear library

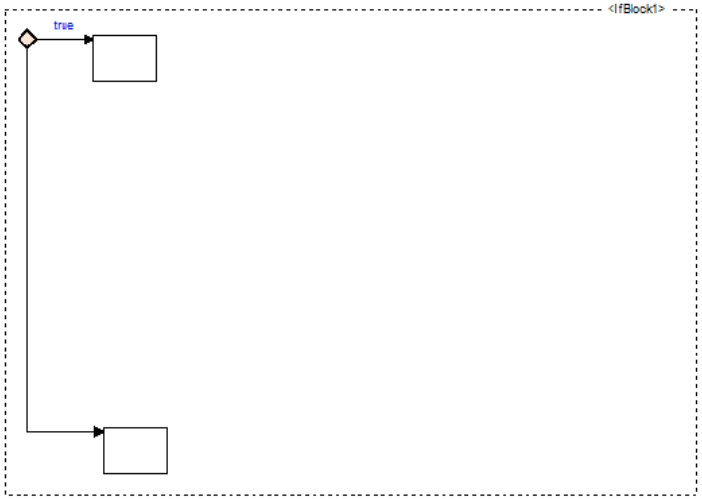



- Declare a local variable to get the current value of the cruise speed into the active branch of the IfBlock
- Connect this local variable to the limiter to produce the final value of the cruise speed
- Use the last value of the cruise speed to keep, increase or decrease it and set the new value to this local variable

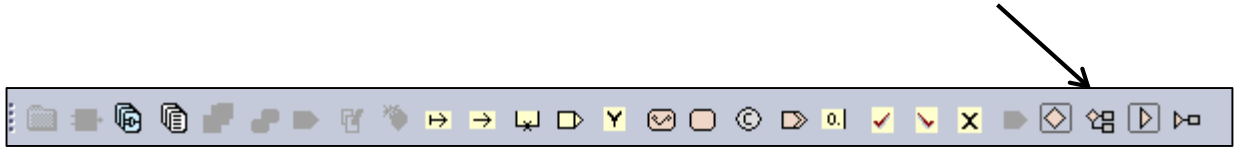
# Lab 4: Solution

Lab Solution

Select the default If Node branch and move it down to allow inserting other branches:



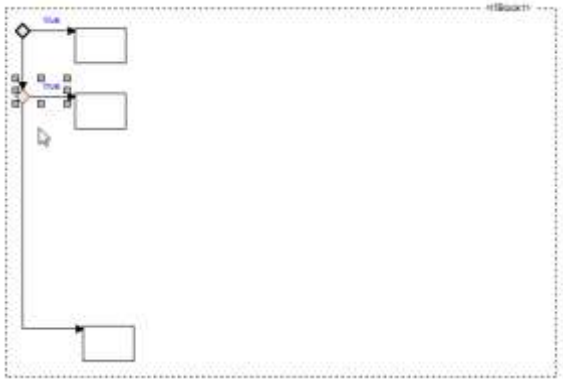

Click on the New If Node icon (Create toolbar) to declare one If Node branch:



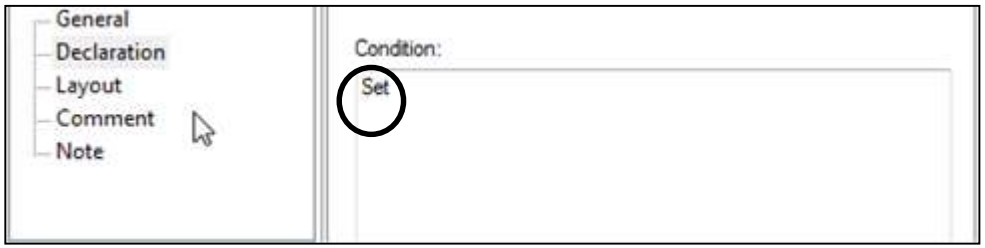
# Lab 4: Solution

Lab Solution

Click on the vertical link at the location where this new If Node branch will be created, to insert it:



Click on the square of the horizontal link to set the condition flag in the Declaration properties, to have the active branch



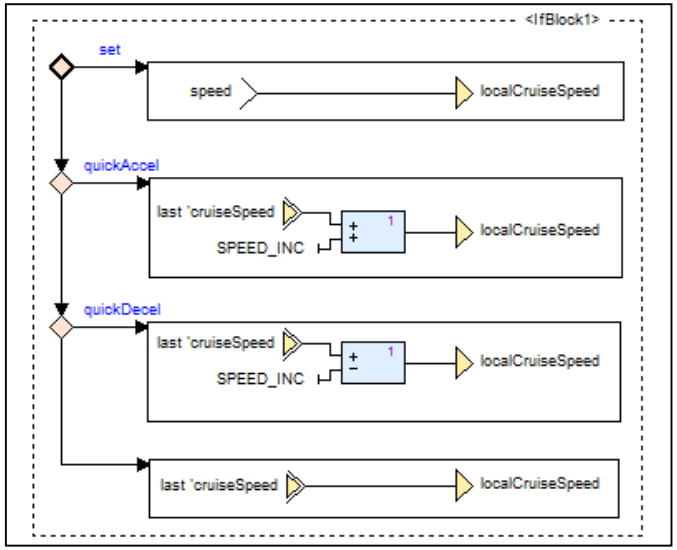

Type the expression in the Condition field

# Lab 4: Solution

Lab Solution

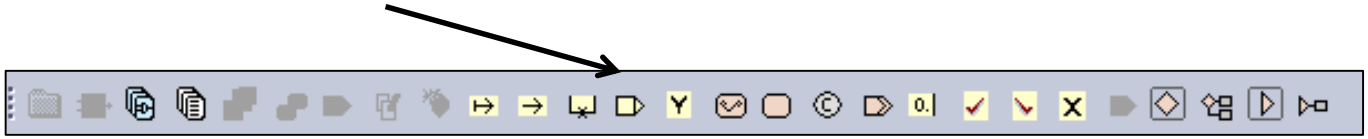
Create the If Node branches and set the conditions as follows:

- Set for the first branch:
  - QuickAccel for the second branch
  - QuickDecel for the third branch
  - No condition for the default one




Declare a local variable named LocalCruiseSpeed to compute the new value of the cruise speed

Select the CruiseSpeedMgt operator from the Scade view and click on the New Local variable icon from the Create toolbar to declare it

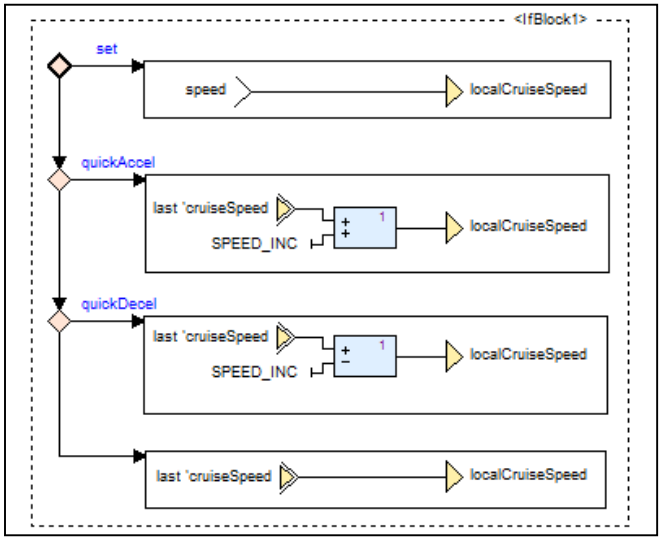


Set the CarType::tSpeed type to it (Declaration properties)

ANSYS

# Lab 4: Solution

Drag and drop LocalCruiseSpeed local variable from the Scade view into the If Node branches




Implement each If Node branch:

- Set branch:

```
LocalCruiseSpeed = Speed
```

- QuickAccel branch:

```
LocalCruiseSpeed = last value of CruiseSpeed + SpeedInc
```

- QuickDecel branch:

```
LocalCruiseSpeed = last value of CruiseSpeed - SpeedInc
```

- Default branch:

```
LocalCruiseSpeed = last value of CruiseSpeed
```

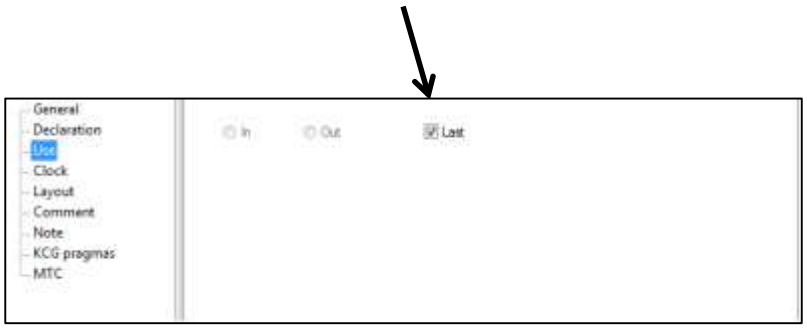
Lab Solution

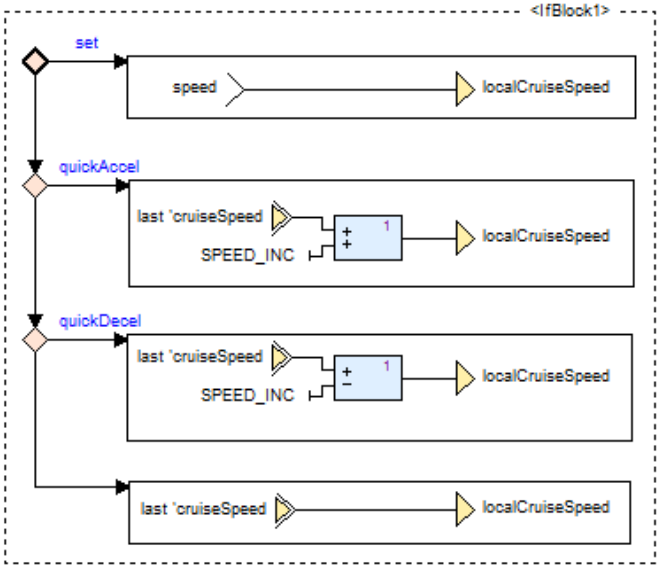
# Lab 4: Solution

Drag and drop CruiseSpeed output from the Scade view into the If Node branches.

Select CruiseSpeed from the branch and set the last value from the Use tab of Properties:

- Check the Last field



# Lab 4: Solution

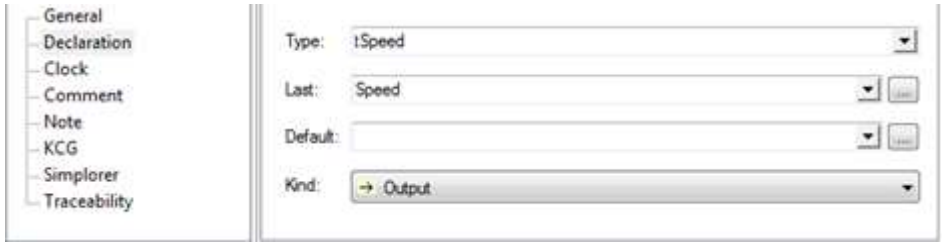
Drag and drop pwlinear::LimiterUnSymmetrical from the libpwlinear library and put it in the diagram (not possible in the IfBlock)

Connect LocalCruiseSpeed and CruiseSpeed to the limiter to produce the final value of the cruise speed

- Set SpeedMin and SpeedMax as LowLimit and HighLimit parameters of the limiter in the Declaration tab of properties

Set Speed to CruiseSpeed as the Last value

- Select CruiseSpeed in the Scade view and set the Last value in the Declaration tab of properties




Lab Solution

# Lab 5

# Lab 5 (1/2)

Lab

## Objective:

Implement the `CruiseControl` (CC) operator:

- Use of states machines, mixed data and control flows

## Requirements:

**Time: 35 min**

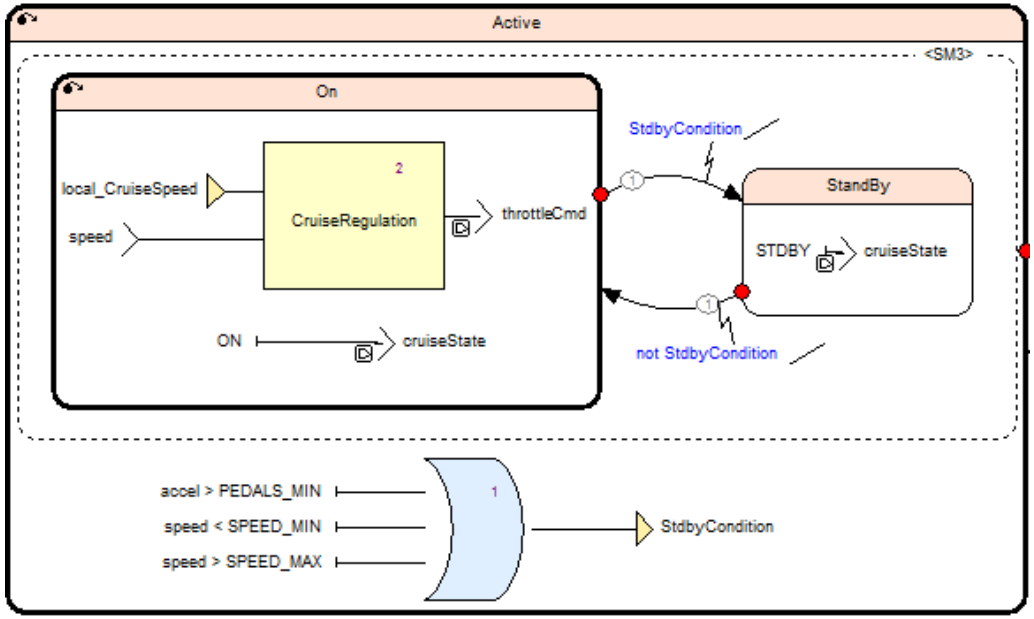
Close all SCADE suite sessions and load a copy of "*Prerequisites\Lab 5\CruiseControl\CruiseControl.etp*"

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 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 |

# Lab 5: Prerequisites

Close all SCADE Suite sessions

Load a copy of the SCADE Suite project, CruiseControl.etp, stored in Q:\Labs\Prerequisites\Lab 5\CruiseControl

Lab Prerequisite

# Lab 5: Prerequisites

Lab Prerequisite

Implement the `CruiseControl` operator

- Read the Cruise Control behavior requirements (CC_HLR_CCB_01 to CC_HLR_CCB_08)
- The CruiseControl operator will call:
  - The CruiseRegulation operator, computes the throttle command (see the Car driving control requirements, CC_HLR_CDC_03 and CC_HLR_CDC_04)
    - The throttle command is saturated at the REGUL_THROTTLE_MAX value in the SaturateThrottle operator (see CC_HLR_CDC_05)
  - The CruiseSpeedMgt operator, manages the value of the cruise speed according to the driver commands when the CruiseControl is enabled (see the Cruise speed management requirements, CC_HLR_CSM_01 to CC_HLR_CSM_05)

# Lab 5: Solution

Lab Solution

Check the interface of `CruiseControl` operator:

- According to CC_HLR_IN_01 to CC_HLR_IN_09 and CC_HLR_OUT_01 to CC_HLR_OUT_03 interface requirements, following I/Os are declared:

| Name | Type | Direction |
|---|---|---|
| on | bool | Input |
| off | bool | Input |
| resumeCmd | bool | Input |
| set | bool | Input |
| quickAccel | bool | Input |
| quickDecel | bool | Input |
| accel | CarType::tPercent | Input |
| brake | CarType::tPercent | Input |
| speed | CarType::tSpeed | Input |
| cruiseSpeed | CarType::tSpeed | Output |
| throttleCmd | CarType::tPercent | Output |
| cruiseState | CruiseControl::teCruiseState | Output |

# Lab 5: Solution

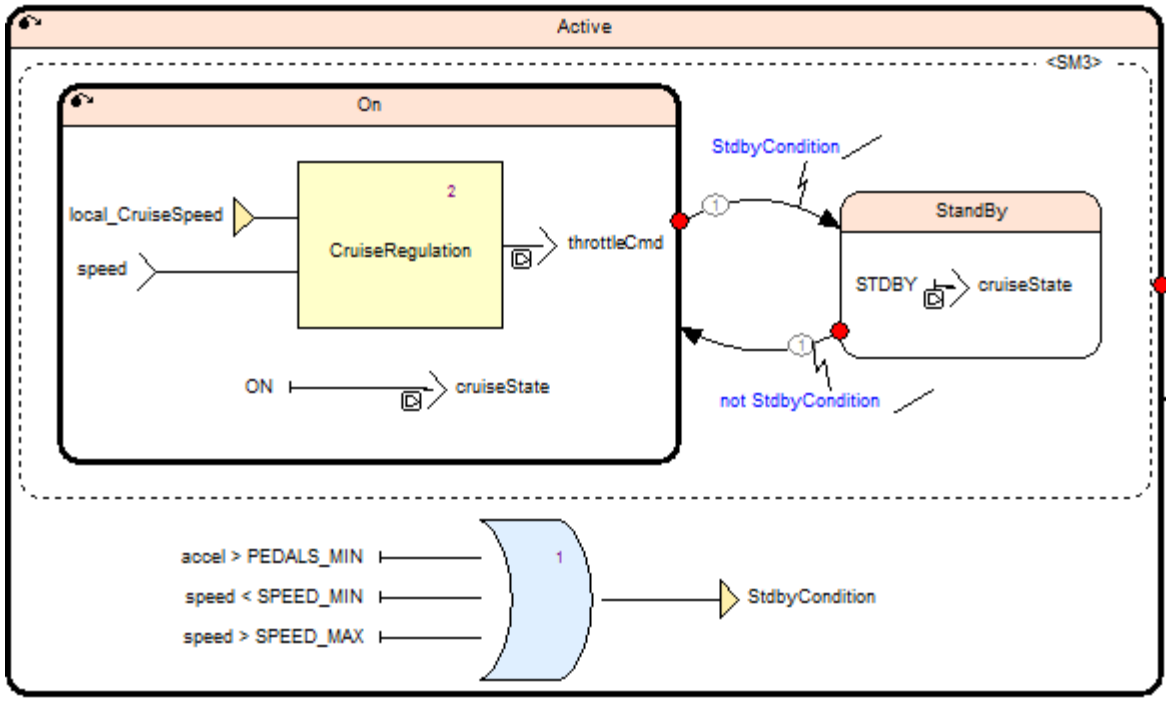
Lab Solution

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

- Insert into SM3, On and StandBy states (and associated transitions) for modelling the system when the CruiseControl is ON or in the STDBY state
- Set the On state as the initialization state
- Insert a strong transition
  - From On to StandBy with StdbyCondition local variable as activation condition
  - From StandBy to On with not StdbyCondition as activation condition
- Implement the equation to compute StdbyCondition (outside SM3 otherwise it is not possible)
  - True when when the accelerator pedal is pressed, or the car speed is outside the speed limit (CC_HLR_CCB_05)
  - Click on the New Local variable icon from the Create toolbar to declare it

# Lab 5: Solution

Lab Solution

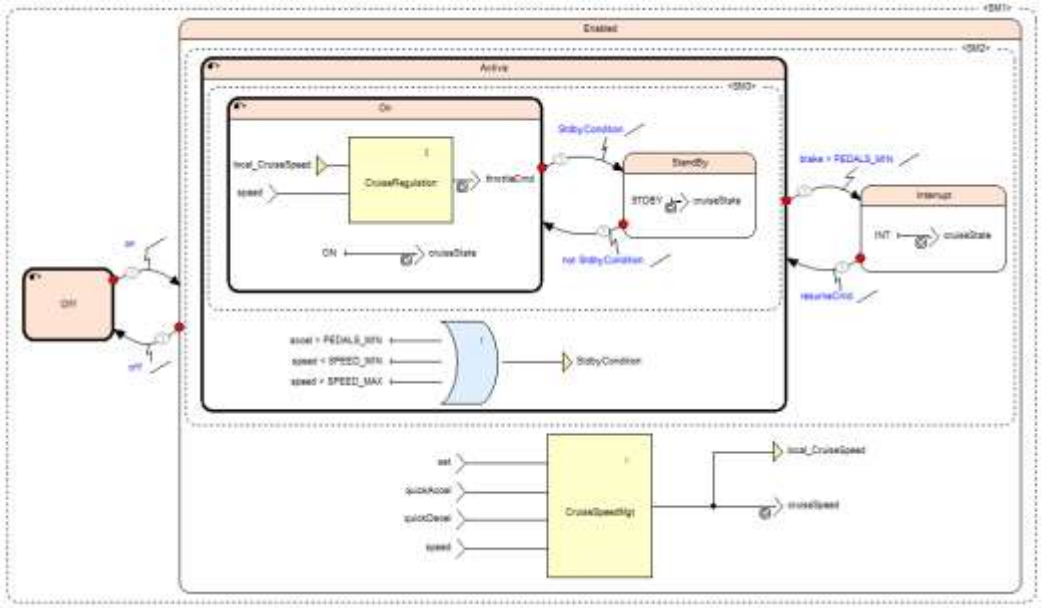



Active State

# Lab 5: Solution

When the CruiseControl is on, the car speed is automatically regulated (see the requirement CC_HLR_CDC_02). Insert into the On state, the call of CruiseRegulation operator:

- This operator needs as input, cruiseSpeed computed by the CruiseSpeedMgt operator:

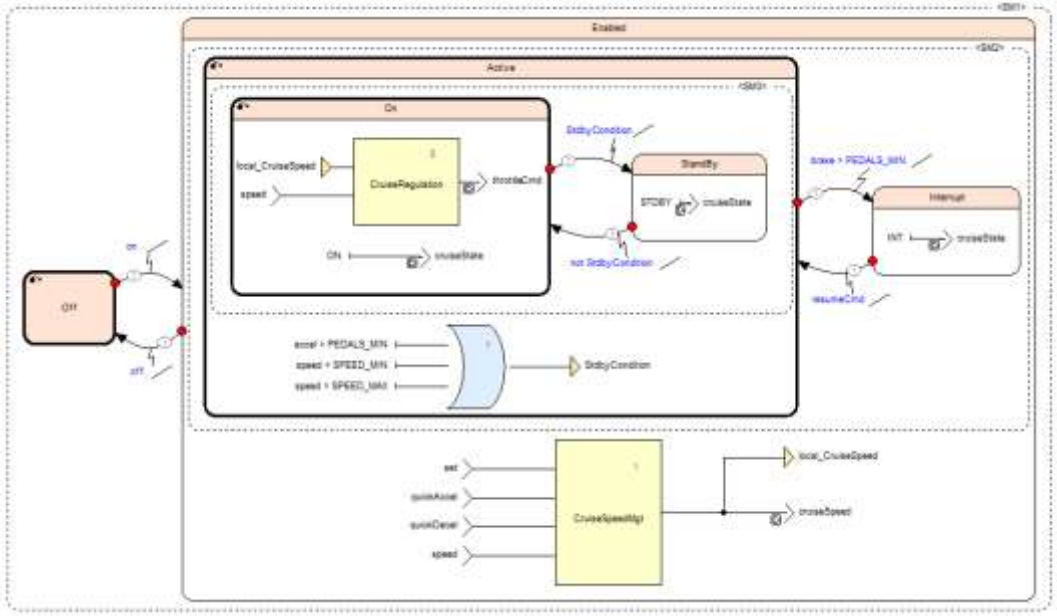



- Use a local variable (declared in Enabled), local_CruiseSpeed, connect it to CruiseSpeed output of CruiseSpeedMgt operator
- Set Accel input as default value to ThrottleCmd Output (Declaration properties)

# Lab 5: Solution

Set the values to the CruiseState output according to CC_HLR_CCB_01, CC_HLR_CCB_04 and CC_HLR_CCB_05 :

- Drag and drop from the Scade view, cruiseState into the states:



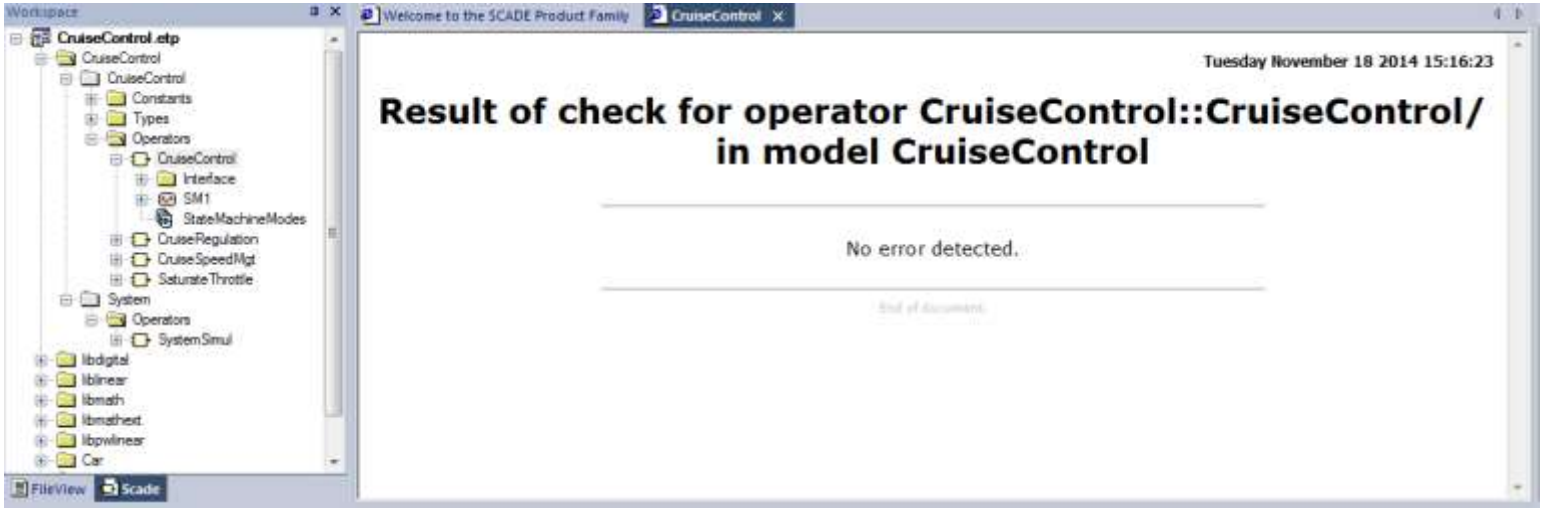

- Use the enumeration type, teCruiseState
- State ON: cruiseState = ON
- State StandBy: cruiseState = STDBY
- State Interrupt: cruiseState = INT
- Set OFF as the default value to CruiseState output

# Lab 5: Solution

Lab Solution

Perform a check of the model to verify if CruiseControl is semantically correct:

- Select CruiseControl from the Scade view and Right-click -> Check
- Verify that no error is raised



# Lab 6

# Lab 6 (1/5)

Use Lab Support p.83-92

## 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

# Lab 6: Prerequisite

Close all SCADE Suite sessions

Load a copy of the SCADE Suite project, CruiseControl.etp, stored in Q:\Labs\Prerequisites\Lab 6\CruiseControl

Lab
Prerequisite

# Lab 6: Solution

Lab Solution

Check the interface of `SystemSimul` operator:

- According to the interface of `CruiseControl` and `CarModel` operators, following I/Os are declared:

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

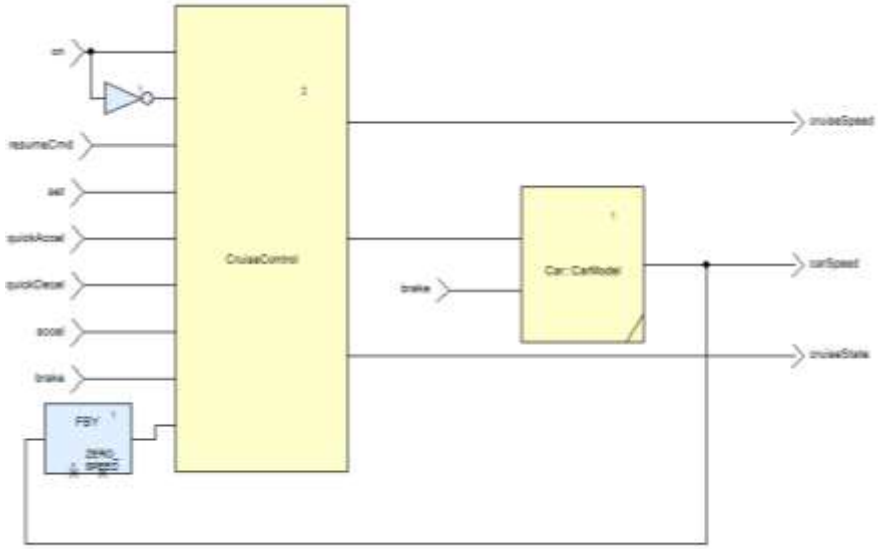
- No need to declare off input because off = not on

# Lab 6: Solution

Lab Solution

Implement the `SystemSimul` diagram:

- Drag and drop into the diagram (from the Scade view): `CruiseControl` and `CarModel` operators and I/Os of SystemSimul operator
- Connect I/Os to `CruiseControl` and `CarModel` and between these operators:
- Select I/Os and the operators and Right-click->Connect->By Name



# Lab 6: Solution

Lab Solution

Simulate the whole application:

- Click on the top menu of the SCADE Suite editor and check the Code Generator toolbar to display it
- Select the Simulation configuration (SimulationAda for Ada target code) from the drop-down list
- Open the Settings window of the SCADE Suite Code Generator
  - Click on Settings icon:


  - Show that the Settings window is displayed
  - Show that the System::SystemSimul operator is selected and code generator is KCG 6.6 C or KCG 6.6 Ada according to the selected language and target code (General tab)

# Lab 6: Solution

Lab Solution

Go to the Integration tab

Show that the Simulator target is selected

Generate the simulation code, build and run it:

- Click on Run icon:


- Confirm the generation of the simulation dynamic-link library (DLL)
- Show that the simulation is launched: Instances tab and simulator with I/Os are displayed

Check the behavior going forward step by step

- For each cycle:

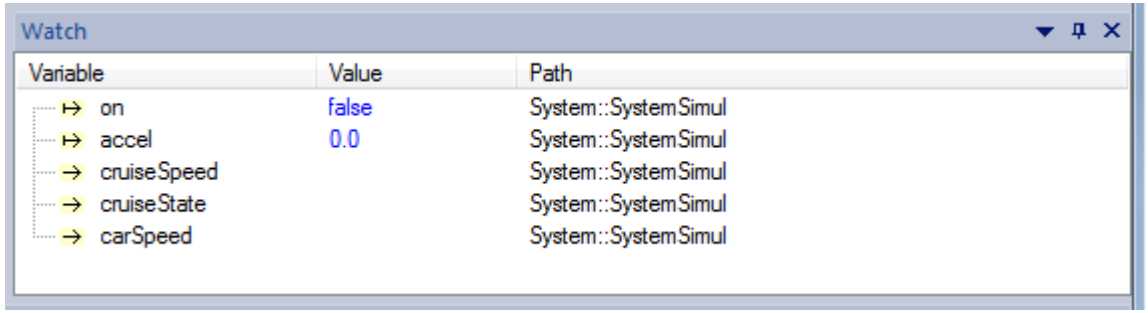

- Click on several inputs in the Instances view to set a value
- Click on Step icon:
- Check the values of computed outputs related to inputs ones

# Lab 6: Solution

Lab Solution

To verify the values of several I/Os, put them on the Watch window:

- For each following I/O, select it in the Instances view and right-click- > Watch:
  - On and Accel inputs
  - CruiseSpeed, CruiseState and CarSpeed outputs
- Show that all selected I/Os are well listed on the Watch window:




  - Click one input (or select it and F2 key) to set a value

**Tip: *Click on View->Docking Windows->Watch* menu to display / hide the Watch window**

# Lab 6: Solution

Lab Solution

First cycle: Set several values to inputs as follows:

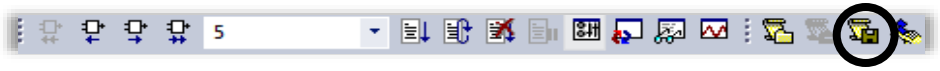
```
on false
resumeCmd false
set false
quickAccel false
quickDecel false
accel 0.0
brake 0.0
```

During 2 cycles, check that

```
cruiseSpeed = CarSpeed = 0.0
cruiseState  = CruiseControl::OFF
```

- First test: `Set Accel 50.0`
- Check that `cruiseState = cruiseControl::OFF` during 5 cycles

Save the steps into a scenario file




- Click Save scenario… icon:
- Set the filename to first_test (save as type: .sss) and click Save

# Lab 6: Solution

Lab Solution

Second test :

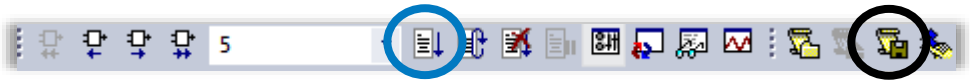

- Click Reset button on the simulator toolbar to reset the simulation
  - Go step and set `accel 100.0`
  - Go step and set `on true`
  - Go step
  - Check that `cruiseState = CruiseControl::STDBY` during 2 cycles
  - Set `Brake 10.0`
  - Check that `cruiseState = CruiseControl::INT` during 2 cycles
  - Go Fast Step twice, forward 10 cycles
  - Set `brake 0.0` and `resumeCmd true`
  - Check that `cruiseState = CruiseControl::STDBY` during 1 cycle
  - `set accel 1.0`
  - Check that `cruiseState = CruiseControl::ON` during 1 cycle
- Save the steps into second_test.sss scenario file

# Lab 6: Solution

Lab Solution

Reset the simulation

Click Load Scenario… to load second_test.sss scenario file




Click Go to play it

Check that I/Os take the same values as done previously

Close the simulation

## 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