



BTC EmbeddedPlatform - Tutorial

Formal Verification

Version 2.5



1. About Formal Verification
2. Start BTC EmbeddedPlatform
3. Perform a Formal Verification
4. Debug a Vector
5. Recheck Proof on fixed version
6. Perform a Formal Verification – Input Restrictions
7. Reporting

Formal Verification is a method that is used to ensure the correctness of a system under test against a Formal Specification of its required behavior.

In addition to the Formal Specification technology, the Formal Verification allows to perform a complete mathematical proof on the system-under-test, to give evidence that a requirement can never be violated.

This means that the user gets a guarantee that there is no combination of input signals and calibrations which can lead the system into a state where the requirement is violated.

This tutorial is based on the tutorial for Formal Specification.
You need to go through that tutorial at first.

Start BTC EmbeddedPlatform

- To prepare the tutorial, please copy the folder with the demo model to a location with standard access rights and ensure that the “Read Only” attribute of the tutorials folder is not set after copying it. The location of the demo model usually is “C:\Program Files\BTC\<version>\tutorials\PowerWindow\PowerWindow_TargetLink”.
- Start BTC EmbeddedPlatform from the Windows Start Menu, Matlab or the Desktop shortcut if available. Once the BTC EmbeddedPlatform has started, please get an overview of the tool. The welcome screen shown below provides a good entry point for different topics.


Welcome to
BTC EmbeddedPlatform®

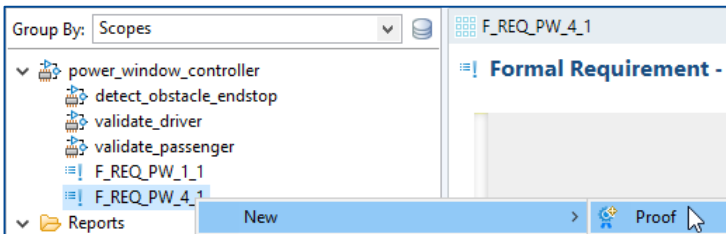
Create a new Profile Recent Profiles

- Embedded Platform Basics**
Basic functions and features of the platform.
[Basics Guide](#)
[Concepts and Use Cases](#)
[Tutorial for Profile Creation](#)
- Requirement Based Testing**
Define and execute functional test cases on your design.
Optionally includes Formal Testing.
[Requirement-based Testing Guide](#)
[Tutorial for Requirements-based Testing](#)
[Tutorial for Formal Test and Requirements-based test generation](#)
- Formal Specification**
Express your functional requirements (semi-)formally.
Optionally generate RTT-Observer for HIL testing.
[Formal Specification Guide](#)
[Tutorial for Formal Specification](#)
[Tutorial for Real Time Testing Observer](#)
- Formal Verification**
Formally prove the correctness of your design using Model Checking technology.
[Formal Verification Guide](#)
[Tutorial for Formal Verification](#)
- Back to Back Testing**
Perform back-to-back testing between two implementations, for example, between model and code
[Back to Back Testing User Guide](#)
[Tutorial for Structural Test Generation and Back-to-Back Testing](#)

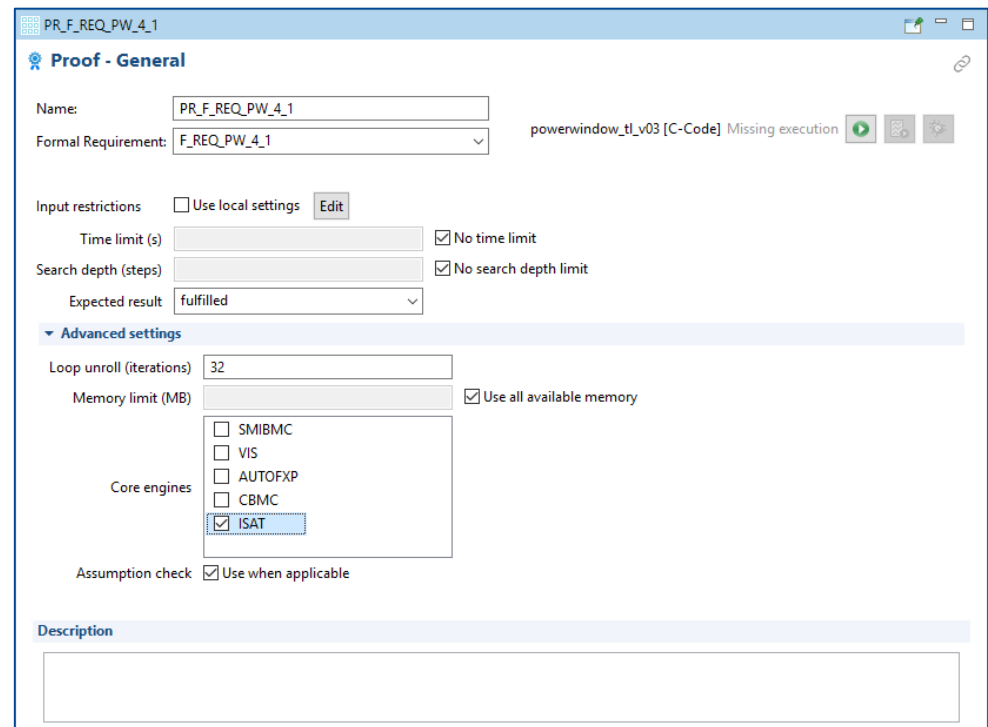
BTC | embedded systems

Perform a Formal Verification (1/2)

1. First of all open the profile created during the Formal Specification tutorial.
2. Switch to the Formal Verification Perspective and select the formal requirement F_REQ_PW_4_1 and right-click on the item in the tree.
3. Select New → Proof from the upcoming context menu.
4. Select the new proof item PR_F_REQ_PW_4_1 that is now attached to the formal requirement in the tree
5. Open the Advanced Settings: For proofs that are expected to be fulfilled, the ISAT engine is a good choice. To force EmbeddedValidator to use this engine, unselect the others and choose “fulfilled” as Expected result.
6. Now run the proof by clicking on the  button




The idea of this proof is to verify, if the requirement can never be violated by any possible input combination over time. It provides a mathematical proof that a requirement can never be violated. In the case it can be violated, a counter example is provided. The counter example can reproduce the violation in a debug environment.



Perform a Formal Verification (2/2)

powerwindow_tl_v03 [C-Code] Violated (4 steps)



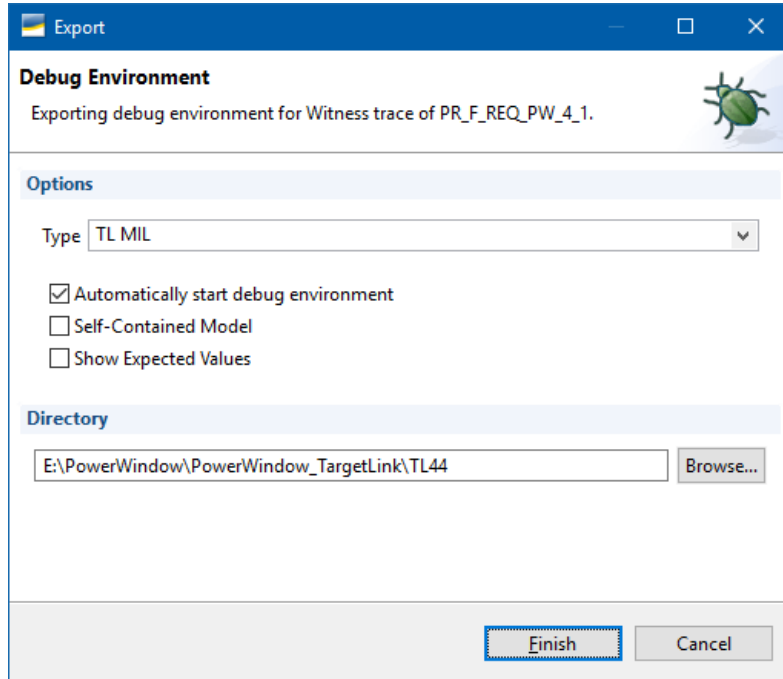
As you can see, BTC EmbeddedPlatform has found a scenario that violates the requirement in the 4th step. Click on the  button to have a look on the vector. The screenshot below shows that there is an obstacle detected in Step 2 and that in Step 3 the window is not moving down.

The model has a sampling time of 0.01, so each step is 10 ms and you expect the window to move down in the next step, because the requirement says the window has to start moving down in 10 ms.


Witness Trace · Fri Mar 22 14:31:40 CET 2019					
Name	Mode	Step 0	Step 1	Step 2	Step 3
Formal Requirement	Status				Violated
Commitment	SUP phase	Startup→∇(Tr)	∇(Tr)	→Tr→∇(Act)	→
obstacle_detection	output	0.0	0.0	1.0	0.0
move_up	output	0.0	1.0	0.0	0.0
driver_up	input	0.0	1.0	0.0	1.0
driver_down	input	0.0	0.0	0.0	1.0
passenger_up	input	0.0	0.0	0.0	1.0
passenger_down	input	1.0	1.0	1.0	0.0

To find the source of the violation, debug the created vector, please.

Debug a vector (1/3)



Debugging the vector will help to find the root cause directly in the model or in the generated code. In this case you will debug it in the model.

1. Click on the  icon in the proof dashboard
2. In the upcoming dialog in the "Options" section select TL_MIL from the drop down.
3. In the directory section you can change the location for the debug model.
4. Click on the "Finish" button to create a debug model.

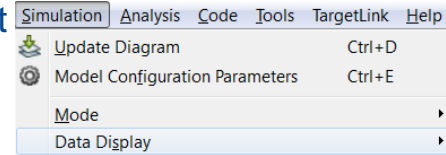
Debug a vector (2/3)

Navigate to the TargetLink subsystem as seen below.

Show the value labels of the ports that you need to observe.

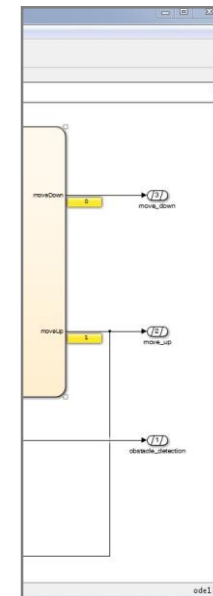
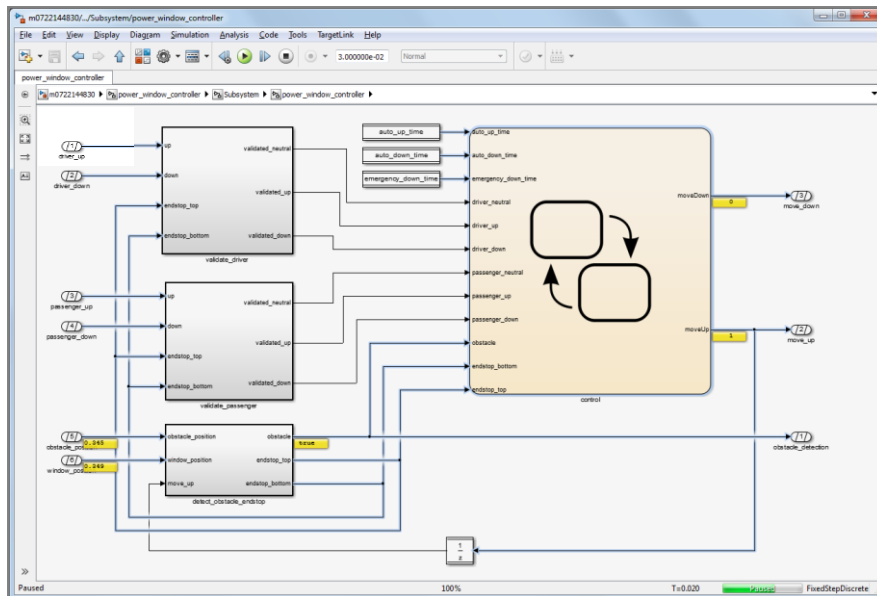
By simulating step-wise you can see the values on each port in each point of time.

- As you can see, an obstacle is recognized in step 3 (T=0,02). You expect the window to move down in the next step ("move_down" (T+1) == 1).
- As shown on the bottom right, the actual value in step 4 is '0' which means that the obstacle is not recognized.



- Remove All Value Labels
- Show Value Label of Selected Port
- Show Value Labels When Hovering
- Toggle Value Labels When Clicked
- O

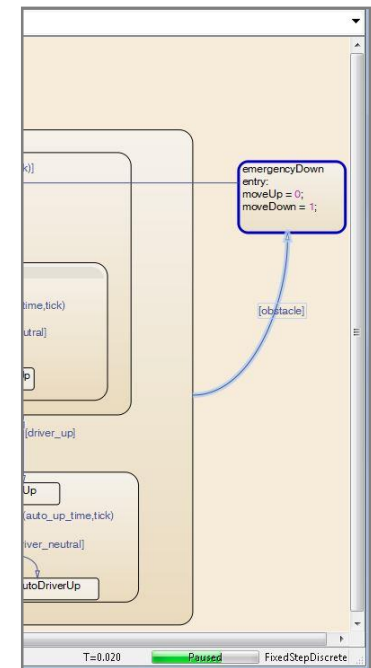
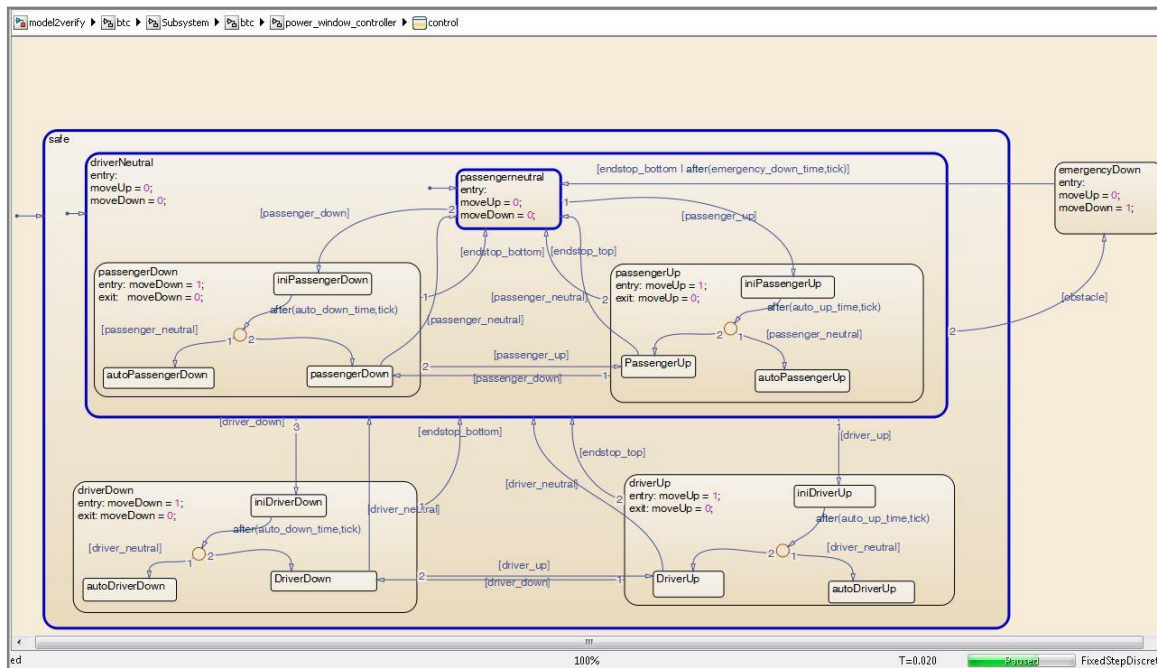
ptions...



Debug a vector (3/3)

The problem in this case is that the state chart, that computes the output signals, is not correctly implemented. The transition which indicates an obstacle detection is connected to the wrong sub-state.

- Please connect this transition to the outer border (“safe” state) to fix the problem.
- Execute the simulation step-wise again and check that the obstacle will be observed correctly and the “emergencyDown” state becomes active.

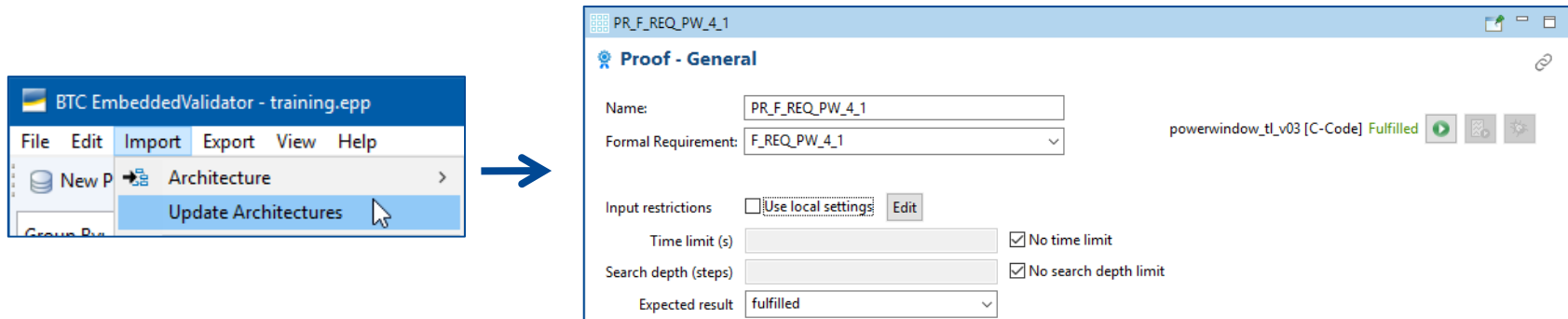


Recheck Proof on fixed version

Open the original model in Matlab and fix this issue in the state chart so the changes can be proven again.

1. Please update the architecture to propagate the changes in the model to the profile data. Therefore select “Import → Update Architectures”. The updated model will now be taken into account.
2. Please re-execute the proof of requirement REQ_PW_4_1 to check whether the model behaves correctly now.

As shown by the execution report, the requirement cannot be violated any more, so the model and the generated code work fine in this regard.



Now the most safety critical requirement of the model is proven successfully to never fail.

Perform a Formal Verification – Local Input Restrictions

Excursus

BTC EmbeddedPlatform takes Min and Max values into account. They can come from the model, DataDictionary (TargetLink) or be imported once a profile is created.

However, for the Formal Verification you might want to have several proofs with different input restrictions to e.g. freeze some calibration variables to a specific value or reduce or even extend the Min and Max ranges for input signals.

BTC EmbeddedPlatform allows the user to define proof specific input restrictions. Therefore tick the checkbox “Use local settings” and click on the edit button to modify your proof specific input restrictions. To return to the general input restriction untick the checkbox again.

Proof - General

Name: PR_F_REQ_PW_4_1

Formal Requirement: F_REQ_PW_4_1

Input restrictions ☒ Use local settings **Edit**

Time limit (s)

Input restriction settings

Edit input restrictions for PR_F_REQ_PW_4_1

Define input restriction settings

Scope: power_window_controller **Select**

☐ Show relative path ☐ Show type range

type filter text

Name	Kind	Data type	Resolution	Offset	Model range	Lower bound	Upper bound
Sa5_position_endstop_top	Parameter	const volatile ...	0.001	0.0	[0.35, 0.45]	0.35	0.45
Sa1_auto_down_time	Parameter	const volatile ...	2^0	0.0	[0.0, 255.0]	0.0	255.0
Sa1_auto_up_time	Parameter	const volatile ...	2^0	0.0	[0.0, 255.0]	0.0	255.0
Sa1_emergency_down_time	Parameter	const volatile ...	2^0	0.0	[0.0, 255.0]	0.0	255.0
Sa1_obstacle_position	Input	UInt16	0.001	0.0	[0.0, 0.45]	0.0	0.45
Sa1_window_position	Input	UInt16	0.001	0.0	[0.0, 0.45]	0.0	0.45
Sa1_driver_down	Input	Bool	2^0	0.0	[0.0, 1.0]	0.0	1.0
Sa1_driver_up	Input	Bool	2^0	0.0	[0.0, 1.0]	0.0	1.0
Sa1_passenger_down	Input	Bool	2^0	0.0	[0.0, 1.0]	0.0	1.0
Sa1_passenger_up	Input	Bool	2^0	0.0	[0.0, 1.0]	0.0	1.0

Input restrictions

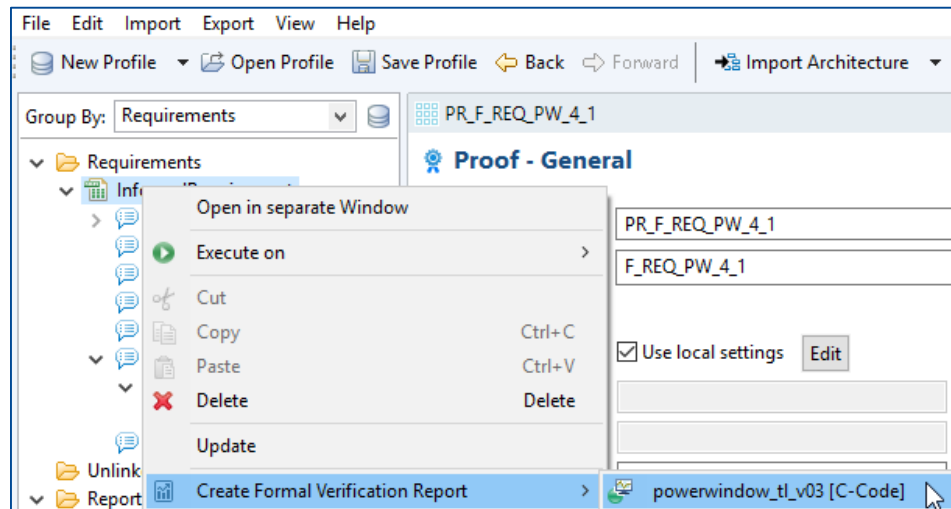
Lower bound Upper bound **Apply** **Reset to model range** **Use defaults for all parameters**

Close

Reporting (1/2)

During testing, the dashboards show the current status of the test project, related to the selected item in the Profile Navigator. In addition it is sometimes necessary to export these information for documentation of the current status or presenting the test status to the customer. For the Formal Verification BTC EmbeddedPlatform provides the Formal Verification report.

To create the report, right-click on the requirements source and select Create Formal Verification Report from the upcoming context menu. Creating the report on the requirements source will take all requirements and therefore all proofs into account. If you create the report on a single requirement, only one proof is reported.



Formal Verification Report

The screenshot displays the 'Formal Verification Report' interface. It features a sidebar with a 'Contents' list and a main content area. The 'Contents' list includes: 1. Meta Information, 2. Summary, 3. F_REQ_PW_A_1, 4. Environmental Assumptions, and 5. Macros. The main content area is currently showing the 'Summary' section for 'F_REQ_PW_A_1'. This section includes a table with columns: Proof, Formal Requirement, Requirement, Requirement ID, Scope, and Status. The table shows a single entry for 'F_REQ_PW_A_1' with a status of 'Proved (10 steps)'. Below the table, there is a 'Proof settings' section with various parameters like Time limit, Memory limit, Search depth, and Loop unroll. A 'Formal Specification' section follows, containing a diagram with 'Trigger' and 'Action' blocks. Below the diagram are two tables: 'Parameter' and 'Macro definitions'. The 'Parameter' table lists parameters like 'interpretation', 'action mode', 'startup phase', 'trigger condition', and 'action condition'. The 'Macro definitions' table lists macros like 'obstacleDetected' and 'windowHasToStartMovingDown'. At the bottom, there are sections for 'Environmental Assumptions' and 'Macros'.

Proof	Formal Requirement	Requirement	Requirement ID	Scope	Status
F_REQ_PW_A_1	F_REQ_PW_A_1	REQ_PW_A_1	REQ_PW_A_1	power_window_controller	Proved (10 steps)

Parameter	Definition
interpretation	progress
action mode	cyclic
startup phase	immediate
trigger condition	SubtaskDetected
action condition	SubtaskDetected
Local scope	[0 ms, 10 ms]
Global scope	*

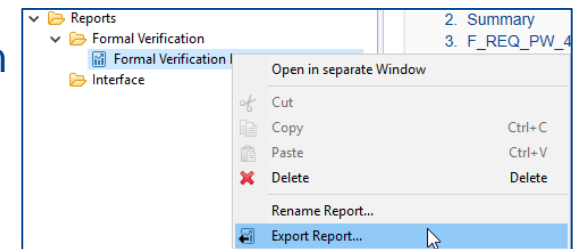
Name	Definition	Description
obstacleDetected	obstacle_detected	(... obstacle is detected(...)
windowHasToStartMovingDown	move_down	(... window has to start moving down(...)

The Formal Verification Report is showing the proof result, the textual and formalized requirements as well as Assumptions and the used macros. Each section can be expanded or collapsed.

The available sections are:

1. Meta information
2. Summary
3. Requirements (textual and formal)
4. Environmental Assumptions
5. Macros

The report can easily be exported with a right-click on the report in the Profile Navigator and select “Export Report ...” from the context menu.





BTC Embedded Systems AG

Gerhard-Stalling-Straße 19, 26135 Oldenburg, GERMANY

Tel.: +49 441 969738 - 50

Fax: +49 441 969738 - 64

www.btc-es.de

