

Reproducing the Experiments of “*Automated Self-Explanation of Expected versus Perceived Behavior for Interacting Digital Systems*”, DATE 2026

Mohammad Alkhiyami
Institute of Embedded Systems
Hamburg University of Technology
Hamburg, Germany
mohammad.alkhiyami@tuhh.de

Gianluca Martino
Institute of Embedded Systems
Hamburg University of Technology
Hamburg, Germany
gianluca.martino@tuhh.de

Goerschwin Fey
Institute of Embedded Systems
Hamburg University of Technology
Hamburg, Germany
goerschwin.fey@tuhh.de

January 8, 2026

1 Introduction

DuRTL is a hardware analysis framework designed to support the construction of symbolic models, simulation-ready representations, and dynamic information flow analyses for **Verilog** designs. The tool provides infrastructure for flattening hierarchical designs, exporting **SMT** representations, generating tagged **Verilog** variants, and collecting signal values across simulation or symbolic execution.

In the context of explanation-oriented verification and diagnostic research, DuRTL enables the automated construction of time-indexed hardware models suitable for consistency checking, mismatch detection, and **UNSAT-core-based** explanation extraction. Essentially, we use **Verilog** descriptions – usually targeting register transfer level digital hardware – to describe complex systems in an abstracted way using finite state machines.

2 Background

The **README** contained in the DuRTL repository describes the installation and usage of DuRTL. In this documentation, our aim is to provide an outline of how the tool can be used for the use case described in the research work [1]. This includes reproducing the explainability experiments described in work [1] and providing an analysis of these experiments.

3 Terminology

Test case inputs are defined in the files `ducode/tests/usage/explainer_turbine_experiments.cpp` and `ducode/tests/usage/explainer_vehicles_experiments.cpp`.

- **Verilog_File:** In our implementation, interacting systems (addressees M_A and explainers M_E) are integrated into a single **Verilog** file that encodes the composed system. This **Verilog** is subsequently parsed by **Yosys** into a **JSON** representation. The

Verilog and JSON representations are provided in `\ducode/tests/testfiles/turbine` and `\ducode/tests/testfiles/traffic`.

- **Unroll_factor:** Since we use Satisfiability Modulo Theories (SMT) expressions with Bounded Model Checking (BMC) the `Unroll_factor` is specified for each test case.
- We set the option of getting the `unsat_core` to true in the parameters of the solver.
- **Assumptions (Γ):** Formally, assumptions are defined as a set of permissible sequences of inputs and outputs describing temporal dependencies. The solver evaluates whether the given models, together with the specified assumptions, form a satisfiable combination. Assumptions are classified into:
 - Γ_A (Addressee Assumptions): Represent the commands, requirements, and expected behaviors from the perspective of the initiating system.
 - Γ_E (Environmental Assumptions): Capture external conditions, sensor inputs, and internal constraints that affect system operation.
 - Γ_C (Connectivity Assumptions): Define the synchronization relationships and expected correlations between system commands and observable behaviors.

4 Workflow

The explanation generation framework operates through a systematic workflow that transforms system designs into actionable conflict explanations:

4.1 Design Processing Phase

- **Model Extraction:** Verilog system descriptions are processed through `Yosys` to generate flattened JSON representations
- **System Composition:** Multiple interacting systems are integrated into a unified model that captures their interfaces and dependencies
- **Temporal Expansion:** The composed system is unrolled across multiple time steps to analyze sequential behavior and state transitions

4.2 Conflict Resolution Process

When behavioral mismatches are detected between expected and perceived system behaviors:

1. **Consistency Checking:** Encode all assumptions and system constraints into an SMT formula for satisfiability checking
2. **UNSAT Core Extraction:** Identify the minimal set of conflicting assumptions when the composed system is unsatisfiable
3. **Iterative Relaxation:** Systematically remove Γ_E assumptions based on priority weights until satisfiability is achieved
4. **Explanation Generation:** Compile the removed assumptions into a minimal explanation set that justifies the behavioral mismatch

5 Test Cases

We implemented the explanation algorithm on several examples, each with multiple mutations.

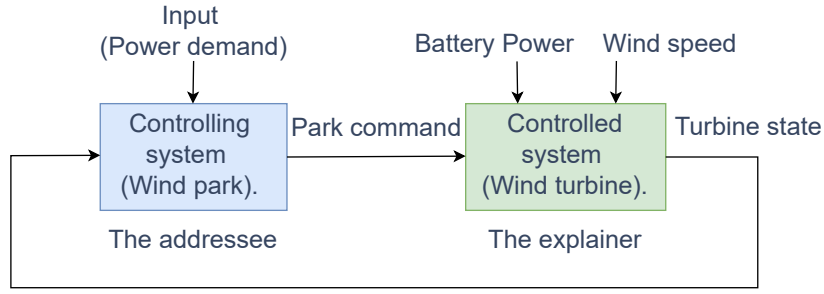


Figure 1: Illustration of the interaction between a wind park and a wind turbine within an explanatory framework.

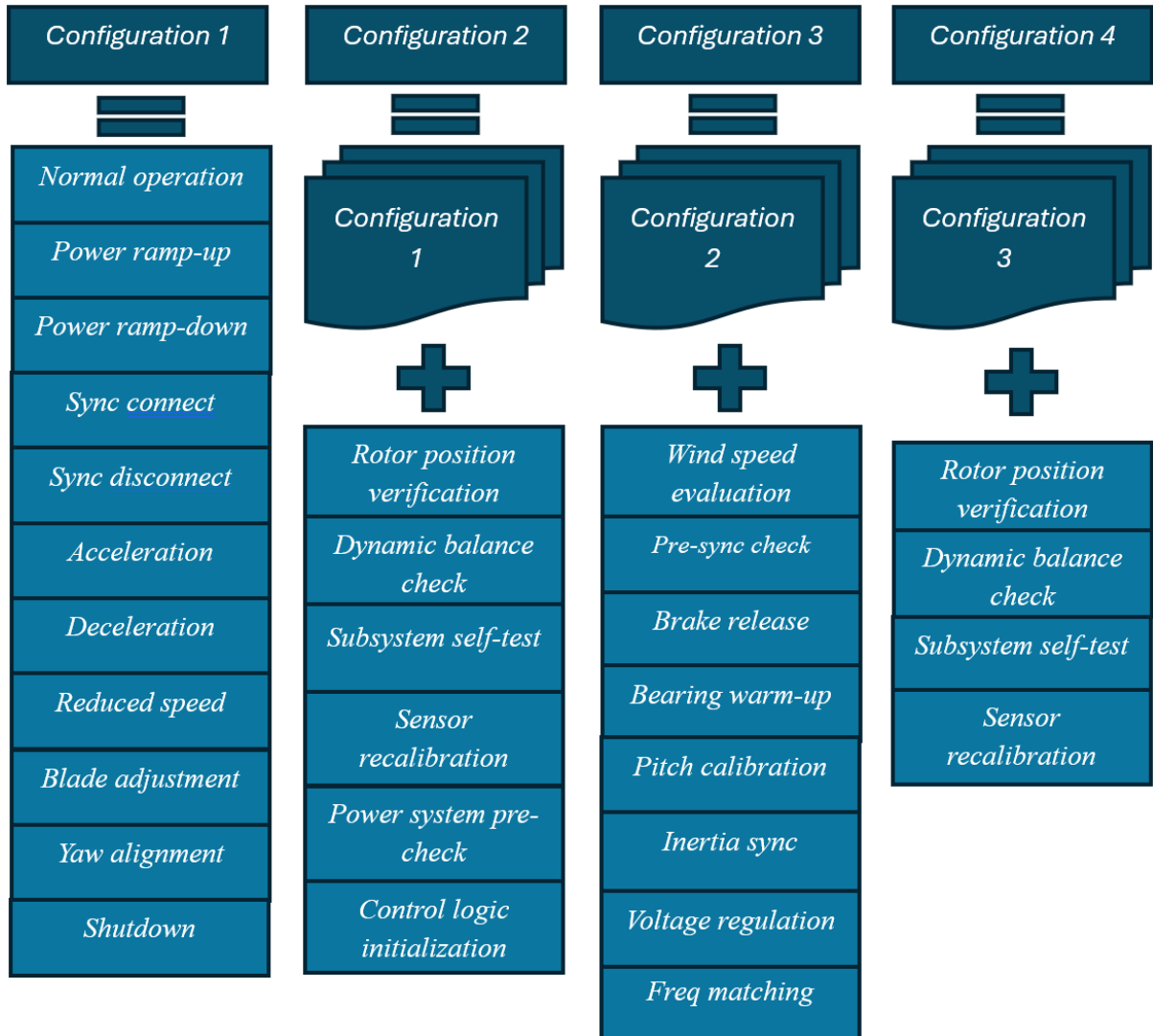


Figure 2: Depth Scaling of Turbine's Model.

```

UNSAT core literals:
  GA_final_normal_operation
  GE_high_wind_speed
  GE_high_wind_implies_blade_adjust
  GC_env_blade_adj_implies_not_normal_op
Removed: GE_high_wind_speed [fE], weight=2
UNSAT core literals:
  GA_final_normal_operation
  GA_power_need_2
  GC_power_need_low_implies_reduced_speed
  GC_env_reduced_speed_implies_not_normal_op
No fE assumptions in core – cannot proceed
Explanation set (removed assumptions):
  GE_high_wind_speed [fE], weight=2
=====
Unroll bound (K): 7
Execution Time (wall clock): 0.108533 sec
Solver Statistics(:added-eqs      1686
:arith-make-feasible 1
:arith-max-columns  4
:binary-propagations 47
:bv-bit2core        150
:conflicts           2
:max-memory          20.65
:memory              20.65
:mk-bool-var         6133
:mk-clause           334
:mk-clause-binary    1938
:num-allocs          2053350903
:num-checks           2
:propagations        58
:rlimit-count        73128)
Total SMT variables (syntactic): 1057
Total SMT variables in assertions: 5214
User CPU time: 0.08807 sec
Peak Memory Usage: 57000 KB

```

Figure 3: Snapshot of the expected output of the algorithm for the turbine scenario.

5.1 Wind Park & Turbine

The wind park acts as the addressee, issuing commands to the turbine, which acts as the explainer, executing commands subject to internal and environmental conditions (e.g., wind speed, battery power). Figure 1 illustrates the interaction when the park issues a command based on power demand, the turbine processes this command along with sensor inputs. The park perceives only three outward states (shutdown, reduced speed, normal operation) representing the expected behavior of the turbine from the park perspective, whereas the turbine transitions through additional internal states to realize these observed behaviors.

5.1.1 Depth Scaling

We consider Q_A as the state space of the addressee and Q_E as the state space of the explainer, respectively; we inflate the wind park scenario to evaluate the explanation algorithm under increasing `Unroll_factor`. We constructed four versions of the composite model M_C , each with a fixed state space of $Q_A = 3$ and varying values of Q_E with 10, 17, 24, and 31 states, respectively, as shown in Figure 2. These configurations represent increasingly detailed turbine models, each covering more operational scenarios.

- 1st configuration: The baseline model described includes the following states: normal operation, power ramp-up, power ramp-down, synchronization connect, synchronization disconnect, acceleration, deceleration, reduced speed, blade adjustment, yaw alignment, and shutdown.
- 2nd configuration: Extends the baseline by adding states such as rotor position verification, dynamic balance check, subsystem self-test, sensor recalibration, power system pre-check,

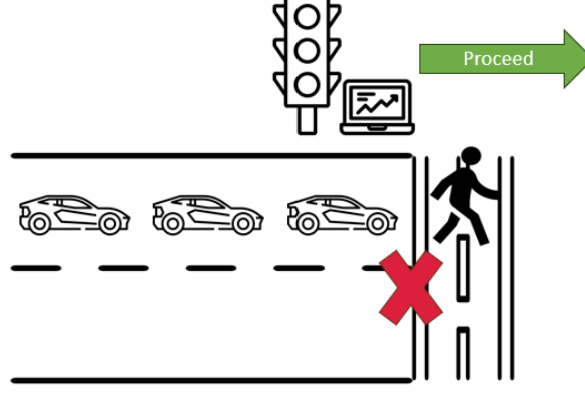


Figure 4: The interaction model (Autonomous Vehicles).

and control logic initialization.

- 3rd configuration: Further adds wind speed evaluation, grid pre-synchronization check, brake release, bearing warm-up, pitch calibration, inertia synchronization, voltage regulation, and frequency matching.
- 4th configuration: Further enriches the model with rotor position verification, dynamic balance check, subsystem self-test, sensor recalibration, power system pre-check, control logic initialization, and safety systems engagement.

5.1.2 Assumptions

- Γ_A represents the commands issued by the park to the turbine.
- Γ_C represents the synchronization assumptions that link the park's commands with the turbine's observable outputs.
- Γ_E represents the turbine's own assumptions, including environmental and internal conditions such as wind speed and battery power.

5.1.3 Mismatch Example

A mismatch arises when the park requests normal operation while wind speed exceeds the threshold of 100 m/s; the turbine enters blade adjustment mode instead. The turbine identifies $\text{wind speed} > 100 \text{ m/s}$ as the conflicting assumption and reports it to the park. This forms the basis of an explanation according to our framework. Figure 3 shows a snapshot for the expected output of the algorithm for the turbine scenario.

5.2 Traffic Controller & Autonomous Vehicles

We next consider a centralized traffic controller and multiple autonomous vehicles traveling sequentially in the same direction. In this setting, the traffic controller acts as the addressee, while each vehicle functions as an explainer.

Figure 4 shows the interaction model where the controller issues traffic commands based on external demand, while each vehicle processes these commands alongside its own local conditions such as pedestrian detection, lane occupancy, road friction, and battery health. The traffic controller can issue three outward commands (stop, slow, and proceed). These commands define the expected behavior of vehicles from the controller's perspective. Hence, the addressee's model M_A is abstract, consisting of only three states corresponding to the outward commands. In

```

UNSAT core literals:
  GE_ped1_present
  GE_occ1_clear
  GE_ped1_implies_StopOut1
  GE_StopOut1_implies_occ1
Removed: GE_ped1_present [ΓE], weight=2
SAT after 1 removal step(s)
Explanation set (removed assumptions):
  GE_ped1_present [ΓE], weight=2
=====
Unroll bound (K): 11
Execution Time (wall clock): 0.887043 sec
Solver Statistics(:added-eqs      67165
:arith-make-feasible 2
:arith-max-columns 4
:binary-propagations 22042
:bv-bit2core 55563
:bv-diseqs 88
:bv-dynamic-diseqs 5227
:bv->core-eq 6376
:conflicts 252
:decisions 1943
:del-clause 12715
:final-checks 1
:max-memory 24.47
:memory 24.47
:minimized-lits 500
:mk-bool-var 30173
:mk-clause 13809
:mk-clause-binary 3159
:num-allocs 6850029513.00
:num-checks 2
:propagations 25358
:restarts 2
:rlimit-count 404650
:time 0.76)
Total SMT variables (syntactic): 1353
Total SMT variables in assertions: 9553
=====
Unroll bound (K): 11
Execution Time (wall clock): 0.887043 sec
User CPU time: 0.879958 sec
Peak Memory Usage: 59508 KB

```

Figure 5: Snapshot for the expected output of the algorithm for the traffic controller scenario.

contrast, each vehicle’s model M_E includes nine internal states (idle, align, gap_assess, accel, sync_merge, cruise, crawl, hold, and brake). The transitions between these states depend not only on the controller’s commands but also on environmental assumptions, e.g., whether a pedestrian is detected, a lane is occupied, road friction is sufficient, and the vehicle’s battery is functional. This demonstrates how incompletely specified behavior can (1) be sufficient within our conceptual framework for explanation, (2) temporal assumptions are used to further constrain the (expected) behavior, and (3) the framework handles more than two systems.

5.2.1 Assumptions

- Γ_A represents the commands issued by the traffic controller to the vehicles.
- Γ_C represents the synchronization assumptions that align the controller’s commands with the vehicle’s outward responses.
- Γ_E represents the vehicles’ own assumptions, including environmental and safety conditions such as pedestrian detection, lane occupancy, road friction, and battery status.

5.2.2 Mismatch Example

A mismatch arises when the controller issues a proceeding command, but a pedestrian is detected, so the vehicle transitions to brake. From the perspective of the controller, this represents an unexpected deviation. The framework then requires the vehicle to generate an explanation by identifying the violated assumption pedestrian detected that caused the mismatch. Figure 5 shows a snapshot for the expected output of the algorithm for the turbine scenario.

References

- [1] M. Alkhiyami, G. Martino, and G. Fey, *Automated Self-Explanation of Expected versus Perceived Behavior for Interacting Digital Systems*, Design, Automation and Test in Europe Conference (DATE), 2026.