Verification of Cyber-Physical Controller Software Using the AVM Meta Tool Suite and HybridSAL

*Joseph Porter* (jporter@isis.vanderbilt.edu),
*Ashish Tiwari* (ashish.tiwari@sri.com),
and
*Xenofon Koutsoukos* (xenofon.koutsoukos@vanderbilt.edu)

S5 Symposium
June 2014
Controller Design

**Design space exploration with simulation** can answer early questions about controller choices
- Simulation over alternatives is much cheaper than building prototypes
- Automated DSE helps answer questions about the evolving design

**Formal verification tools** can detect hard-to-find errors
- Testing may fail to detect (large scenario space)
- Requires property specification
- CyPhy enables easy property addition, model editing, and visualize verification results
OpenMETA-CyPhy Cyber Toolchain (DARPA AVM)

Cyber Components Authoring

- Simulink/Stateflow
- Cyber Modeling Language
  * Cyber Modeling
  - Auto Import
  - Verification Condition Generation
  - Cyber Code Generator

System Design Space Authoring

- CyPhy System Design / Design Space
  - CyPhy System Design
  - Dynamics TB
  - Verification TB

Physical Components Authoring

- Custom Components
  - Dymola/OpenModelica
  - C2M2L Components

Cyber Components Authoring

- Cyber Modeling
- Component Use

Hybrid System Evaluation

- Modelica Sim (.mo, .lib)
- Cyber XML
- DAE XML
- HybridSAL
- Formal Verification

Cyber XML

Modelica Simulation

HybridSAL
Why Cyber Modeling?

Everything has software – cars, refrigerators, even chainsaws.

How do we model controller software and combine it with the mechanical design?

• Keep the design processes for the physical and cyber in sync.
• Building and testing prototypes is expensive. Integrating controllers into mechanical designs is costly and time-consuming.

Too many scenarios to test cyber and physical systems exhaustively. Testing will miss important cases. Some examples:
• Toyota Acceleration problem
• MARS polar lander
• Ariane 5
How are META Cyber capabilities different from designing and simulating controllers using Modelica or Simulink alone?

1. Cyber controller component in Modelica is not a simulation model, but is the actual embedded code scheduled by a discrete-time periodic sampler. The simulation is much more realistic.

2. Integrated formal hybrid verification toolchain.

3. All of the benefits of using CyPhyML:
   - design space exploration
   - curated component library
   - integrated test bench models
   - parametric exploration
   - cloud-based simulation
Introduction

We will show how the Cyber tools can integrate multiple transmission controllers modeled in Simulink/Stateflow with a Driveline modeled in Modelica.

We will use DSE to assess the controller alternatives in simulation, and then verify the candidate controller.

Scenario:
If we have a cheaper fixed-point processor, will the controller performance be degraded?
Creating a CyPhy model

The CyPhy model is the “integration” model.

It contains:

- The imported Modelica model (*The System*)
- The interface to the Cyber model (*The Controller*)
- A design describing how the two are connected
- A test bench describing how to evaluate the design
All modern designs have software controllers. How do we incorporate them into a design?
CyPhyML Controller Composition Concepts

AVM Component Interface (Dynamics)
- Connectors
- Parameters

Modelica Component Interface
- Signals (causal)
- Physical variables (acausal)
- Buses (aggregate)
- Parameters

Modelica Components and Ports each have a Class string indicating the location of its type definition in the library.

CyPhyML Controller Composition Concepts

AVM Component Interface (Dynamics)
- Connectors
- Parameters

Modelica Component Interface
- Signals (causal)
- Physical variables (acausal)
- Buses (aggregate)
- Parameters

Modelica Components and Ports each have a Class string indicating the location of its type definition in the library:

In composition, the underlying port types must match exactly (by type string).

Modelica Components and Ports each have a Class string indicating the location of its type definition in the library.

C2M2L_Ext Interfaces.Context Interfaces.Driver.Driver_Bus
CyPhyML Controller Composition Concepts

Controller components must be causal:
- Only connect to signals and bus ports containing signals.
- Physical variable ports can not be connected to controller components.

Parameter ports are exposed in assemblies and test benches. Parameter values are propagated down into the components.
CyPhyML Controller Composition Concepts

Cyber Component
(refers to external Cyber model)

Controller Component

Physical Component
(Modelica Component
(refers to external model in Modelica component libraries)

AVM and Cyber component model interfaces are identical.

Cyber interface parameters refer directly to parameter objects deep in the Simulink hierarchy.

Bus ports aggregate signals. Bus signals are identified by name. Only one component can produce (output) a bus signal. The others must consume it.
Translating controllers to the Cyber language

Controllers in Simulink/Stateflow

- Shift controller
- Torque converter controller
- Torque reduction controller

Controllers in GME

Cyber language

MDL2MGA Translator
Design Space Exploration

DSE allows us to evaluate different controller variants, to compare performance.
META-Cyber Simulation Workflow

1. Create or obtain physical component models in Modelica.
2. Create a CyPhyML model using GME.
3. Import Modelica components and build the system design space in the CyPhy model.
4. Create your controllers in Simulink/Stateflow.
5. Create a Cyber model in META and import your controllers into this model.
6. Define the controller interface in the CyPhyML model using GME (6a) and link it to the Cyber model (6b).
7. Create a test bench for the design in CyPhyML.
8. From the test bench, run the Master Interpreter to generate Modelica simulation code (8a). The Modelica generator also invokes the code generator to create Modelica controller components which are integrated into the simulation (8b).
9. Run the simulation.
Tool Flow

- Test Bench model defines inputs & environment for system under test
- Invoke DSE to explore alternatives
- DESERT tool presents design choices
- Select designs to generate
- Specify component Library paths
- Generate the Modelica model
- Simulate the designs and compare
Generated Code

Generated code is placed in Simulink subdirectory

Generated code can be inspected by opening the generated Visual Studio solution file.
Alternative #1

Simulation results #1

Selected gear vs time (s)

Engine rpm vs time (blue)
Transmission output rpm vs time (red)
Alternative #2

Simulation results #2

- Selected gear vs time (s)
- Engine rpm vs time (blue)
- Transmission output rpm vs time (red)
Once we have identified our candidate controller, we can use formal verification to assess the controller logic.
Formal Verification

**Testing/simulation** can fail to detect design errors that are manifested only under certain scenarios

**Formal verification** can verify designs for all scenarios

Formal verification complements simulation in improving confidence in system design
**Formal Verification**

What is formal verification?

Techniques for verifying the system that are based on **symbolic algebra**, rather than **numerical simulations**

Achieves the equivalent of **exhaustive testing**
Verification Workflow

Identical to the *simulation workflow* except:

- Temporal **properties** are attached to models
- Verification results, namely
  - status (pass, fail, or error) and explanation are displayed on the dashboard

**Controllers** often include intricate logic that can be difficult to exhaustively test

Apply formal verification to **ShiftController** here
Temporal Properties

Properties capture the intent of the controller design

ShiftController:
Inputs:
1. driver_gear_select
2. shift_request_state
3. input_speed_TC
4. output_speed_TC
Output:
1. gear_selected
Shift Controller Inputs-Output

The input variable `driver_gear_select` takes values:
- reverse=1
- park=2
- swim=3
- neutral=4
- neutral_pivot=5
- low=6
- drive=7

The input variable `shift_request_state` takes values:
- down_shift=1
- no_shift=2
- up_shift=3

The output variable `gear_selected` takes values:
- 0, 1, 2, 3, 4

A correct controller should guarantee something about the output under certain assumptions on the inputs.
Temporal Properties as Specification

Some desired properties for such a ShiftController:

1. If shift_request_state==3 (up_shift) and driver_gear_select==7 (drive), then eventually gear_selected==4 (fourth gear)
   • \( \square( srs==3 \land dgs==7 \Rightarrow <>(gear==4) ) \)
   • \( \square( \square(srs==3 \land dgs==7) \Rightarrow <>(gear==4) ) \)

2. If dgs==6(low), then eventually gear <= 1
   • \( \square( \square(dgs==6) \Rightarrow <>(gear \leq 1) ) \)
   • \( \square( \square(dgs==6) \Rightarrow <>(gear \leq 2) ) \)
Pattern-based Property Specification

All properties above have the same “Global Response” pattern

Another useful pattern is “Absence before R”

Property: “The output gear_selected does not take value 4 before driver_gear_select is 7 (drive)”

\[
P := \text{gear}_\text{selected} == 4 \\
R := \text{driver}_\text{gear}_\text{select} == 7
\]

Property: P is absent before R
Adding LTL Properties to SL/SF Models
If \textit{driver\_gear\_select} is 6(\textit{low}), then eventually \textit{gear\_selected} $\leq$ 1 (\textit{first gear})
Viewing Properties in SL/SF Models

Set of all properties = Specification of the component
The Verification TestBench

After controller models have been annotated with desired temporal properties, they are translated into CyberComposition language, and verified.

Verification results can be viewed in the dashboard.
Verification Results in the Dashboard

Design Space Analyzer
[FANG: SURVIVABILITY / HUMAN FACTORS CHALLENGE] | BETA TESTING

Leader Board  Requirements Analysis  Design Space  Design Space Analysis
Parametric Design Analysis  Probabilistic Certificate of Correctness
Multi-Attribute Decision Analysis  File Load Summary  Surrogate Model Performance

Layout:  Locked  Unlocked  Refresh  Help

Formal Verification

Component Assembly

Green = SUCCESS, Red = FAIL, Yellow = UNKNOWN
Missing squares represent missing simulation data.
Understanding Verification Results

1. If shift_request_state==3 (up_shift) and driver_gear_select==7 (drive), then eventually gear_selected==4
   •  []( srs==3 && dgs==7 => <> (gear==4) ) Violated
   •  []( [](srs==3 && dgs==7) => <> (gear==4) ) Verified

2. If dgs==6(low), then eventually gear <= 1
   •  []( [](dgs==1) => <> (gear <= 1) ) Violated
   •  []( [](dgs==1) => <> (gear <= 2) ) Verified

3. The event gear_selected==4 is absent before dgs is 7 (drive) Verified
Property is False in the Model: Details

Source: SRI
Result: FAIL
ReqName: AbsenceProperty
DesignName: ComponentAssembly
DesignID: 87ee5753-77d6-4189-b879-a12cc33859305
TestBench: TestBench
DesignSpace:

Counterexample generated by Cyber Composition Verification

001 ---- Step 0:
002 ---- Input Variables:
003 ---- output_speed_torque_converter = -1/100
004 ---- driver_gear_select = 1
005 ---- input_speed_torque_converter = 0
006 ---- shift_request_state = 8
007 ---- System Variables:
008 ---- dummy = 9
009 ---- current_time = -1/50
010 ---- SamplePeriod = 1/100
011 ---- gear_selected = 5
012 ---- lockout_time = -1/100
013 ---- ShiftControllerMode = gear0
Simulating the Counter-Example
Visulizing the Counter-Example

\([\square(\square(dgs==6) \Rightarrow \langle\rangle(gear \leq 1))\) Violated
Refining the Model or Property

When a property is found to be false in the model, the user can view and analyze the counter-example, and based on that, go back to the controller design and fix either:

- The controller
  - Edit transition guards
  - Add/delete transitions
- The property
  - Make the property weaker
  - Constrain the inputs of the controller
Refining the Property

Since output variable, \textit{gear\_selected}, gets stuck at value 2, we can see if our controller satisfies a weaker specification.

\[
\Box( \Box (\text{dgs}==6) \Rightarrow \neg\neg(\text{gear} \leq 2) ) \quad \text{Verified}
\]
Refining the Model

Designer changes **condition** on the downshift transition out of Gear2 in SL/SF
Refining the Model

The condition on the outgoing transition from state Gear2 to state lockoutD1 is changed from:

\[ srs==1 \land \neg dgs==6 \land \neg dgs==1 \land \text{in\_tc} > \text{out\_tc} \]

\[ \downarrow \]

\[ ((srs==1 \land \neg dgs==6 \land \neg dgs==1 \land \text{in\_tc} > \text{out\_tc}) \lor (dgs==6 \lor dgs==1)) \]

Enable transition additionally when driver\_gear\_select is 6 (low)
Verification Results for the Updated Model

Design Space Analyzer
[FANG: SURVIVABILITY / HUMAN FACTORS CHALLENGE] | BETA TESTING

- Leader Board
- Requirements Analysis
- Design Space
- Design Space Analysis
- Parametric Design Analysis
- Probabilistic Certificate of Correctness
- Multi-Attribute Decision Analysis
- File Load Summary
- Surrogate Model Performance

**Formal Verification**

- ComponentAssembly
- ResponseProperty1
- ResponseProperty2
- ResponseProperty3
- ResponseProperty4
- ResponseProperty5

Green = SUCCESS, Red = FAIL, Yellow = UNKNOWN
Missing squares represent missing simulation data.

Source: [%%Location of the resource%%]
Property is True in the Model: Details

Source: SRI
Result: SUCCESS
ReqName: AbsenceProperty
DesignName: Component/Assembly
DesignID: 336e5a8-fc01-494d-807f-8928a944fcb4
TestBench: TestBench
DesignSpace:
Old CounterExample on the Fixed Model

\[ \Box( \Box(\text{dgs}==6) \implies \Diamond\Diamond(\text{gear} \leq 1) ) \] Verified
Conclusions

Design space exploration with simulation can answer early questions about controller choices
• Simulation over alternatives is much cheaper than building prototypes
• Automated DSE helps answer questions about the evolving design

Formal verification tools can detect hard-to-find errors
• Testing may fail to detect (large scenario space)
• Requires property specification
• CyPhy enables easy property addition, model editing, and visualize verification results
Questions