Stupid Tool Tricks for Smart Model Based Design

Mark Lawford & Many Others

McMaster Centre for Software Certification (McSCert)
McMaster University
Hamilton, ON, Canada

"Workshop on Formal and Model-Driven Techniques for Developing Trustworthy Systems"
FM&MDD Workshop at ICFEM 2016
Tokyo, Japan
November 14, 2016

Outline

1. Introduction
   - Stupid Tricks
   - Background
2. Tool Qualification
3. Coding Guideline Compliance & Documentation
   - Data Store Push-Down Tool
   - Signature Tool
4. Augmenting Existing Development Processes
   - Calibration Generation
   - HEV Background
   - MapleSim+ATP for a specific Powertrain Architecture Model

Self examination

I can't do anything. I'm just an anchor for really good people.
– A well respected academic

The unexamined life isn't worth living. –Socrates

McSCert

Stupid Pet Tricks

McSCert

Stupid Pet Tricks ⇒ Stupid Tool Tricks

Stupid pet tricks

Now please, the pets are not stupid. The people who taught them the tricks are not stupid. It's just that it's a colloquialism for . . .
"Oh! Isn't that cute!"

Stupid tool tricks

Now please, the tools are not stupid. The people who programmed the tool tricks are not stupid. It's just that it's a colloquialism for . . .
"Oh! Isn't that useful!"

McSCert

Tool Links & References

Tools for Matlab/Simulink (Tool names are hyperlinked!):

- Tabular Expression Toolbox
- Auto Layout Tool & Line-Goto/From Tool
- Data Rescoping (Pushdown) Tool
- Reach/Coreach Tool
- Signature Tool

References:

1. V Pantelic, S Postma, M Lawford, A Korobkine, B Mackenzie, M Bialy, M Bender, J Ong, G Marks, A Wassyng, “Software Engineering Practices and Simulink: Bridging the Gap”, *International Journal on Software Tools for Technology Transfer (STTT)*, Accepted 03/11/2016.
2. C. Eles, M. Lawford, “A tabular expression toolbox for Matlab/Simulink,” *NASA Formal Methods*, LNCS 6617, Springer, 2011, 494-499.
3. M. Bender, K. Laurin, M. Lawford, V. Pantelic, A. Korobkine, J. Ong, B. Mackenzie, M. Bialy, S. Postma, “Signature required: Making Simulink data flow and interfaces explicit”, *Science of Computer Programming*, Volume 113, Part 1, 1 December 2015, 29 - 50.
4. V Pantelic, S Postma, M Lawford, A Korobkine, B Mackenzie, J Ong, M Bender, "A Toolset for Simulink: Improving Software Engineering Practices in Development with Simulink," *Proceedings of 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2015)*, SCITEPRESS, 2015, 50-61.

McSCert

Subsection 2

Background

McSCert

A Failure Example from Aerospace - Rushby

Fuel emergency on Airbus A340-642, G-VATL, on 8 February 2005 (AAIB SPECIAL Bulletin S1/2005)

- Toward the end of a flight from Hong Kong to London:
  1. two engines flamed out,
  2. crew found some tanks were critically low on fuel, declared an emergency,
  3. landed at Amsterdam

The plane basically landed on fumes in the tanks of the remaining 2 engines even though there was sufficient fuel in other tanks.

McSCert

What went wrong?

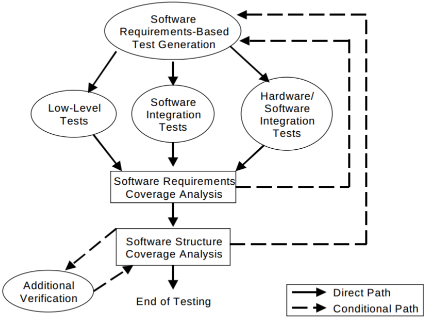
- Two Fuel Control Monitoring Computers (FCMCs) on this type of airplane; each a self-checking pair with a backup (so 6-fold redundant in total); they cross-compare and the “healthiest” one drives the outputs to the data bus
- Both FCMCs had fault indications, and one of them was unable to drive the data bus
- Unfortunately, this one was judged the healthiest and was given control of the bus even though it could not exercise it
- The backups were suppressed because the FCMCs indicated they were not both failed

It’s the requirements, stupid!

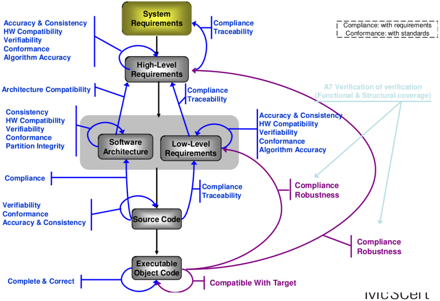
FCMCs functioned as specified. Requirements were incorrect.

Lesson: Requirements are (the only) killer in Civilian Aerospace - John Rushby@SRI

- This example is typical:
  - all software incidents in commercial aircraft have been due to flawed system requirements
  - None due to development or implementation flaws
- How is this possible? 100% MCDC coverage from SRS +

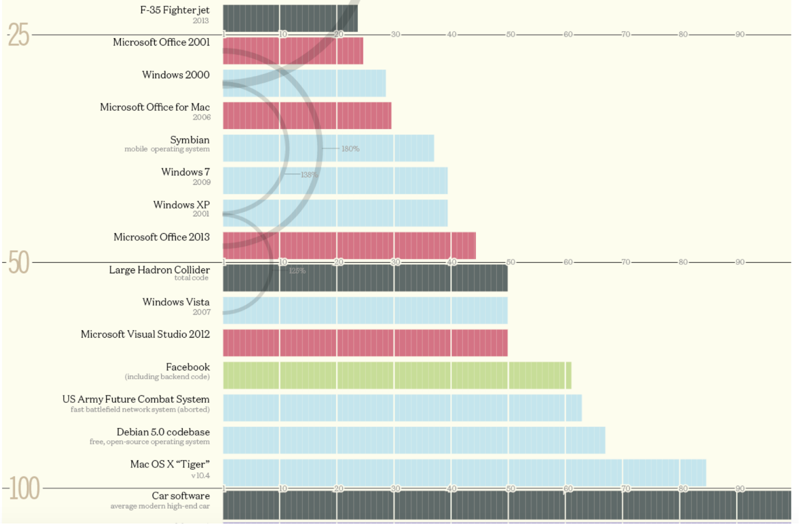

Software Requirements-Based Test Generation
Low-Level Tests
Software Integration Tests
Hardware/ Software Integration Tests
Software Requirements Coverage Analysis
Software Structure Coverage Analysis
Additional Verification
End of Testing
Direct Path
Conditional Path

DO-178/ED-12 – Verification Process



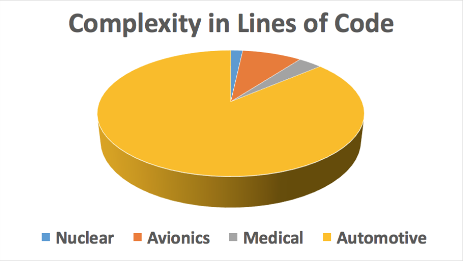
- Seems DO-178B/C is effective, though *very* expensive

Automotive vs. Other Industries



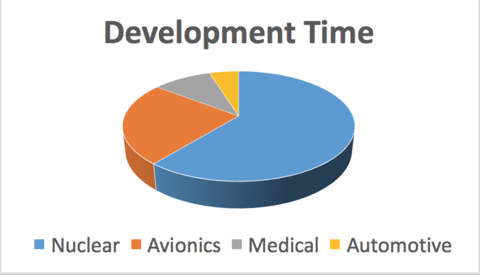
Source: Information is Beautiful

Automotive vs. Other Industries



Automotive has to do a lot more . . .

. . . in a lot more less time!



"Coding is over!" - John Knight

Push to generate Hardware

Push to generate Software

Push to Validate

It doesn't matter what language you teach anymore. Java, Python, C, C# are irrelevant. What matters is models. Engineers will create models and generate the code.

Long live encoding! (of models . . . in MATLAB/Simulink)

So we don’t need software engineers?

Bad news for Managers. Good news for Engineers.

You’ll still need the Engineers to create the models and the Software Engineers to help manage the models and abstractions.

- Engineers will provide insight into how to model and design control systems.
- A recurring theme of MDD is moving the focus up the levels of abstraction to a more productive layer closer to the engineering problem.
- The problem is that software engineering principles still need to be applied to models, but most engineers developing the models are not taught SE Fundamentals.

Coding is mostly over. (Software) Engineering is *definitely* not over.

Automotive clearly needs help

Tesla Model S fatal Autopilot crash

Source: Reuters

So why is the industry not using our tools/theories/methods?

The unexamined life isn't worth living. -Socrates

McSCert

Section 3

Tool Qualification

McSCert

Be like Dudley!

Make it easy for industry to use your tools.

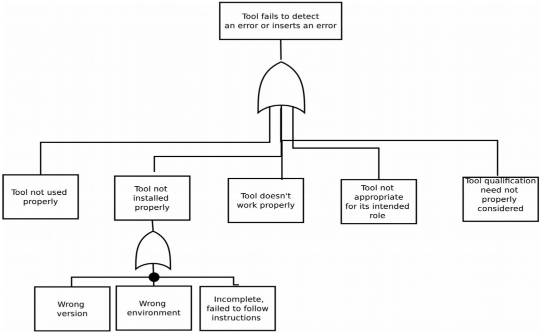
What is Tool Qualification?

In a nutshell?

- Fit for use

Why should I care about it?

Because it's one of the biggest hurdles to getting your tools and theories used.

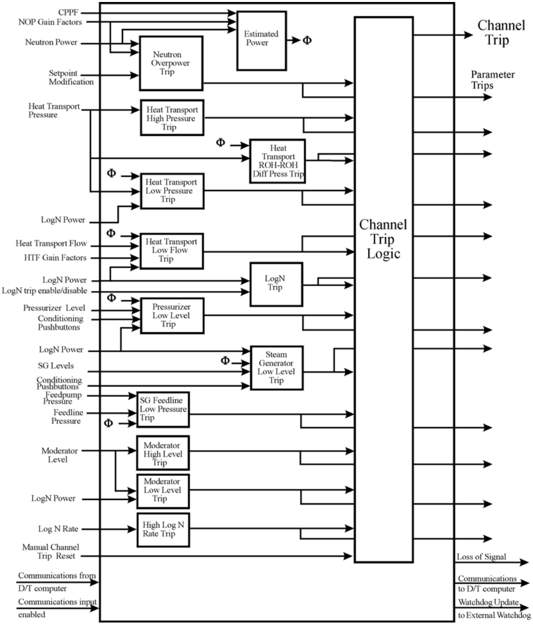

Tool fails to detect an error or inserts an error
Tool not used properly
Tool not installed properly
Tool doesn't work properly
Tool not appropriate for its intended role
Tool qualification need not properly considered
Wrong version
Wrong environment
Incomplete, failed to follow instructions

McSCert

Nuclear Example: The Darlington SDS Redesign Project

- OPG worked with the regulator to get agreement on:
  - what would be an acceptable development process
  - what evidence would be sufficient for licensing
- For the SDS Redesign Project formal techniques were integrated in the forward development process.
- Forward development process produced evidence for certification/evaluation

McSCert



Using Information Hiding

60 modules
280 access programs
40,000 lines of code
(including comments)
33,000 FORTRAN
7,000 assembler
84 monitored variables
27 controlled variables

McSCert

The Standard Used

The CANDU Computer Systems Engineering Centre for Excellence *Standard for Software Engineering of Safety Critical Software* first fundamental principle states:

> *"The required behavior of the software shall be documented using mathematical functions in a notation which has well defined syntax and semantics."*

Determinism: Want unambiguous description of safety critical behavior

Clarity: Easier to understand functional requirements

Preference: Engineers prefer to specify precise behavior and appeal to tolerances when necessary

Sufficient: Functional methods often sufficient - Work "most of the time" & are easily automated

McSCert

The Assurance Case Implicit in the CANDU Standard

A part of the assurance case implicitly embodied in the standard employed in developing the SDS was as follows:

1. The requirements are specified mathematically and checked for completeness and consistency. A hazards analysis is required to document risks and especially to identify sources of single point failures. These hazards have to be mitigated in the specified requirements.
2. Compliance between requirements and software design is mathematically verified.
3. Compliance between the code and software design is verified through both mathematical verification and testing. Compliance between code and requirements is shown explicitly through testing. However, there is an implicit argument of compliance between code and requirements through the transitive closure of the mathematical verification - code to design, and design to requirements.

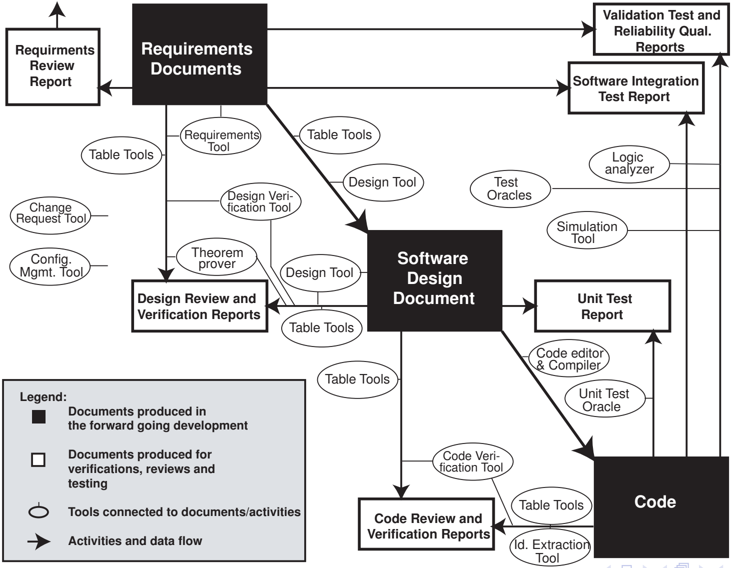
Tabular Expressions (Parnas Tables)

- readable & precise documentation for complex relations
- amenable to formal verification (e.g., PVS)
  - completeness – no missing cases
  - disjointness – deterministic, unambiguous behaviour
- used in the two Darlington nuclear reactor SDSs [e.g., *f_NOPsp*]

		Result
Condition		**f**
C_1	$C_{1.1}$	res_1
	$C_{1.2}$	res_2
	$\ldots$	$\ldots$
	$C_{1.m}$	res_m
$\ldots$		$\ldots$
C_n		res_n

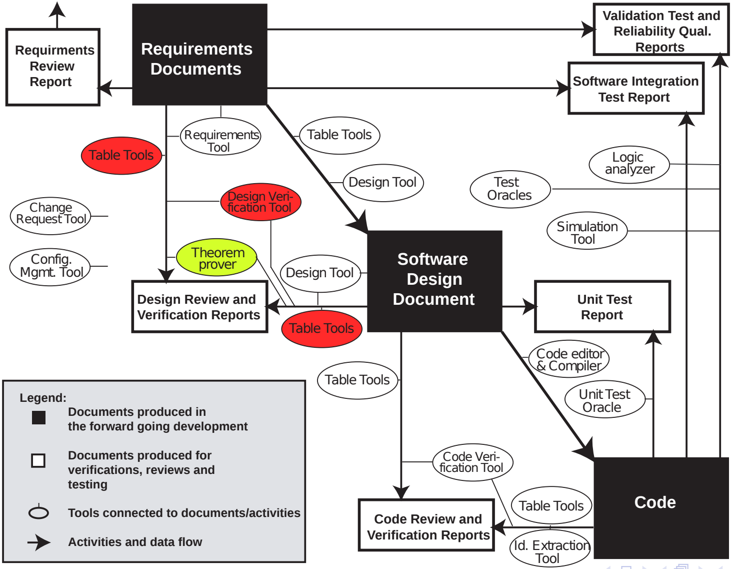
```
IF $C_1$
   IF       $C_{1.1}$ THEN f = $res_1$
   ELSEIF   $C_{1.2}$ THEN f = $res_2$
   ...
   ELSEIF   $C_{1.m}$ THEN f = $res_m$
ELSEIF   ...
ELSEIF   $C_n$ THEN f = $res_n$
```

Idealized Development Process & Tools



McSCert

Idealized Development Process & Tools



Legend:

■ Documents produced in the forward going development

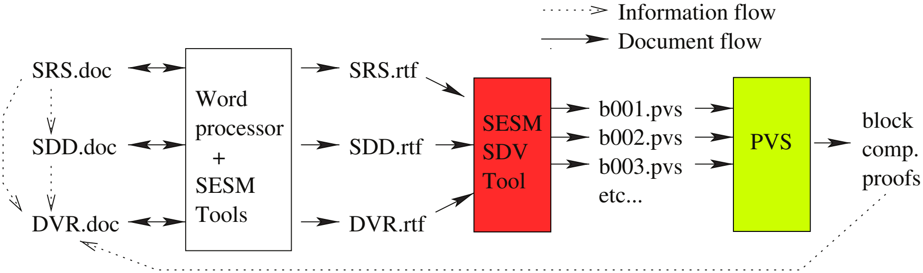
□ Documents produced for verifications, reviews and testing

⬭ Tools connected to documents/activities

➔ Activities and data flow

Tool Supported Formal Methods

A formal method should be tightly integrated with the software development process - i.e. it is directly applied to project documents used by all parties as part of the process.

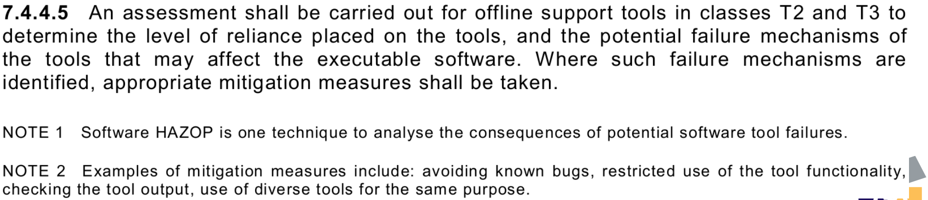


$$SOF_i \circ Abst_{V_{i-1}} = Abst_{V_i} \circ REQ_i$$

Every Formal Verification was done manually too!

Say what?

- Tools are great, but they don't buy you as much as you think if they can be a single point of failure.
- At the time the regulator wanted to mitigate a failure of PVS with a known method, manual proof.
- Latest version of IEC-61508-3 now provides better guidance here.



7.4.4.5 An assessment shall be carried out for offline support tools in classes T2 and T3 to determine the level of reliance placed on the tools, and the potential failure mechanisms of the tools that may affect the executable software. Where such failure mechanisms are identified, appropriate mitigation measures shall be taken.

NOTE 1 Software HAZOP is one technique to analyse the consequences of potential software tool failures.

NOTE 2 Examples of mitigation measures include: avoiding known bugs, restricted use of the tool functionality, checking the tool output, use of diverse tools for the same purpose.

Tool Qualification in DO-178C/DO-330

> *An assessment on all the tools used in the framework of the tool life cycle processes should be conducted in order to identify the need for qualification of these tools. Qualification of these tools is needed when processes of this document are eliminated, reduced, or automated by the use of a tool without its output verified as specified in section 6.*

DO-330 S. 4.4(e)
Tool Planning Process Activities

Example of Elimination

Doing formal proof that code conforms to low level requirements then saying that as a result you don't need to do MCDC test from low level requirements.

McSCert

Tool Qualification in ISO 26262

11.4 Requirements and recommendations
11.4.1 General requirement
11.4.1.1 If the safety lifecycle incorporates the use of a software tool for the development of a system, or its hardware or software elements, such that activities or tasks required by ISO 26262 rely on the correct functioning of a software tool, and where the relevant outputs of that tool are not examined or verified for the applicable process step(s), such software tools shall comply with the requirements of this clause.

McSCert

Solving the Tool Qualification Problem

The bad news:

You will, in all likelihood, need two different tools in order to avoid having to do it manually because “demonstrating soundness of the tools” will likely be difficult or impossible

The good news:

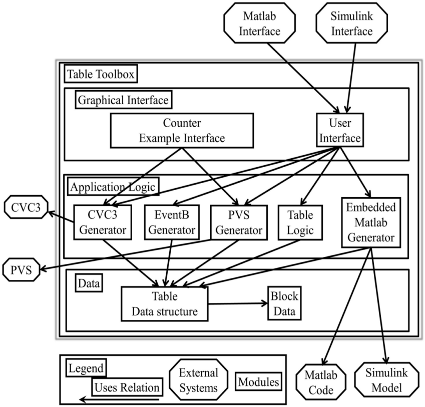
Its not as hard as you might think to knock the tool qualification requirements down a level by doing the same thing with 2+ tools.

- DSLs can be used to generate code for multiple theorem provers, or SMT solvers, or model checkers
- There is often more than one way to get a result
- This can help avoid vendor lock-in

Consider this in developing your tools and process

McSCert

New Tabular Expression Toolbox Architecture



Completeness & disjointness checks generated 2 ways:

1. TET→PVS Table
2. TET→CVC3 queries

Embedded Matlab generator only path to code

Table Data structure is also single point failure for everything

Design for Qualification

No need to qualify PVS or SMT solver.

Stupid Tool Trick #1

Do everything twice in two different ways.

Be like Dudley.

Make it easy for industry to use your tools by thinking about tool qualification!

McSCert

Section 4

Coding Guideline Compliance & Documentation

McSCert

Be like Twig!

Go with the [data] flow!

Subsection 1

Data Store Push-Down Tool

McSCert

Motivation

- Some of the industrial models we have been working with define most of their data stores at the top level of a model's hierarchy
  - This is analogous to programming with many global variables
- Data stores, like variables in traditional programming languages, should be restricted in scope in order to
  - Avoid inadvertent/unwanted access
  - Increase understandability
  - Hide low-level details
  - Reduce the number of (implicit) inputs for testing, resulting in decreased number of test cases

Application

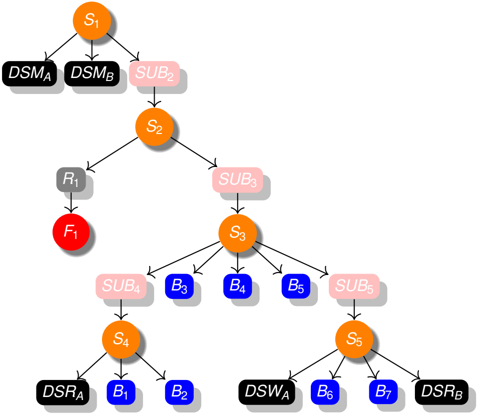


Figure: Data store push-down: before

Data store localization: after

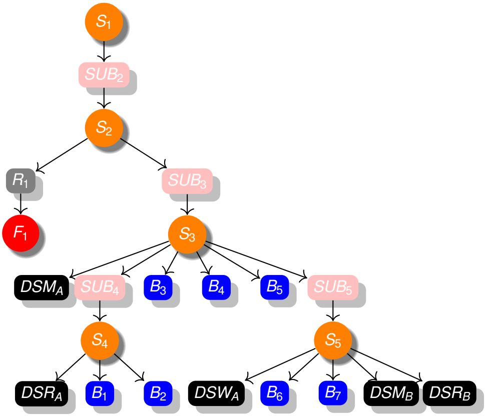
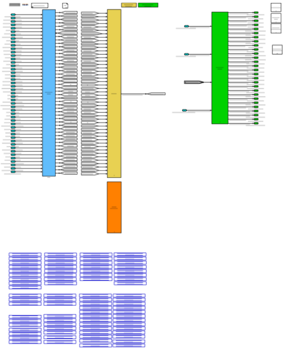


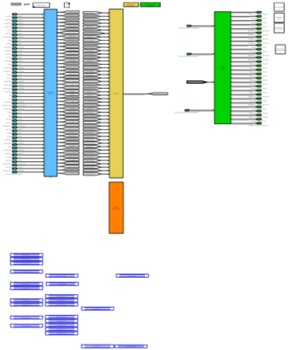
Figure: Data store push-down: after

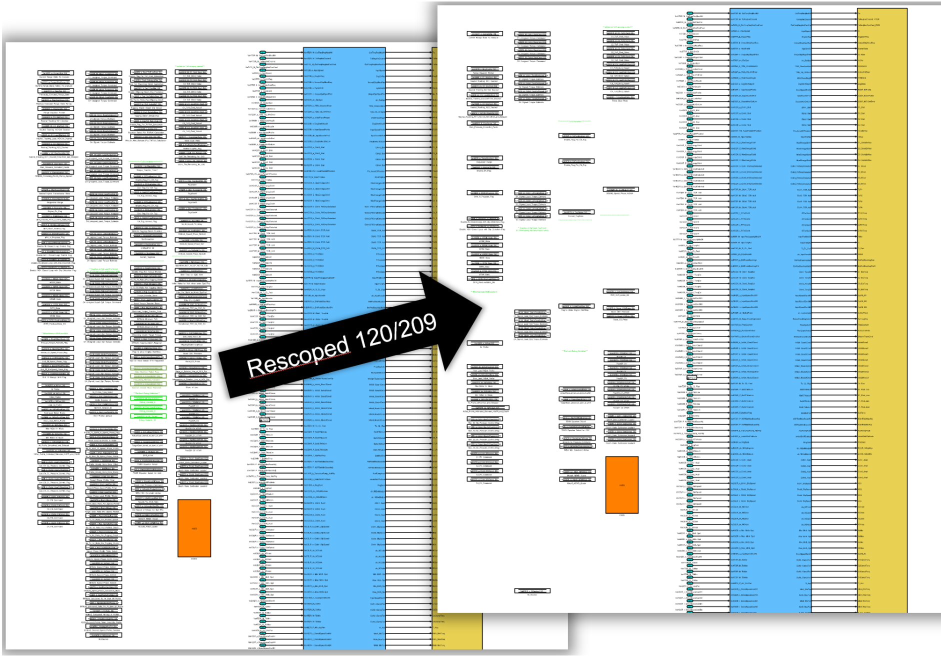
Illustration of Push-Down on an Industrial Model

Before: 78 top level data stores



After: 24 top level data stores




Rescoped 120/209

Application

- The push-down operation improves models’ modularity, comprehensibility, maintainability, reusability
- Proper scoping of data stores can be enforced as a rule in guidelines
    - Companies typically use guidelines for MBD with Simulink
        - E.g., naming conventions, grouping blocks into subsystems, etc.
- Japan MathWorks Automotive Advisory Board (JMAAB) guidelines strongly recommends positioning Data Store Memory blocks as low as possible in the model hierarchy

Stupid Tool Trick # 2

Be like Twig!

> *When in Rome, do as the Romans do.* – *St. Ambrose*

- No company is going to throw out all of its process, tools & code and start using your pet method tomorrow

Be like Twig.

- Recognize that a seemingly "trivial" tool to you could be a useful trick that gets collaborative research going.

- No tool is stupid if its useful. It will only be used if you put it in their environment.
- Trust is the Trojan horse that will help you slowly try to get industry to use your methods.

Subsection 2

Signature Tool

McSCert

Motivation

- Bring the basic self-documentation components of traditional programming languages to Simulink
  - In C, the interface to a module is defined in a header file
- Understanding data flow in Simulink models can be very difficult
  - Blocks can be connected by signal lines
  - Further, Simulink includes mechanisms such as *Goto/From* pairs and *data stores* to implicitly connect blocks
  - One may need to search through many levels in the system hierarchy

What is a Signature?

*A **representation** of the interface of a Simulink subsystem*

- *Explicit* (ports)
- *Implicit* (inherited data stores and non-local Goto/From tags)
- *Imposed* (declarations)

The tool identifies two signatures for a subsystem S

- **Weak signature $\boldsymbol{Sig^w}$**(S): the resources the subsystem S *can access*
- **Strong signature $\boldsymbol{Sig^s}$**(S): the resources the subsystem S is *actually accessing*

Data Flow in Simulink: An Illustration

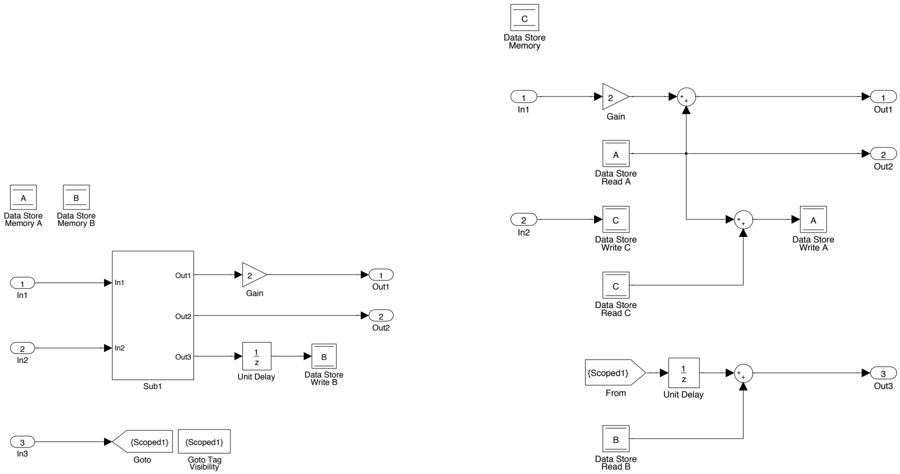


Figure: A top level of a model (left) and the contents of its subsystem Sub1 (right)

Strong Signature for Sub1

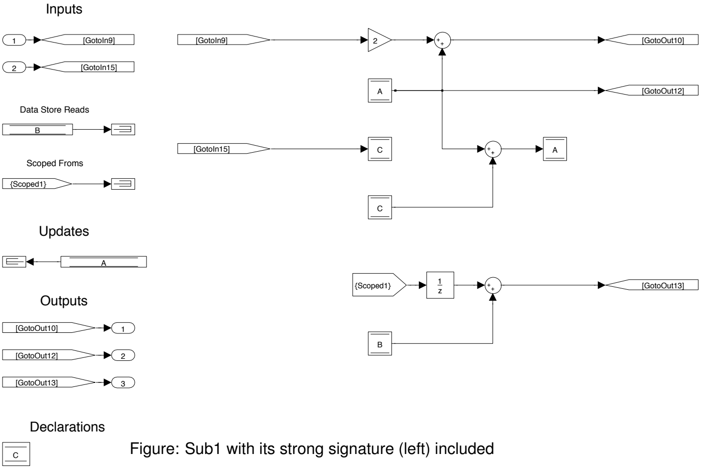
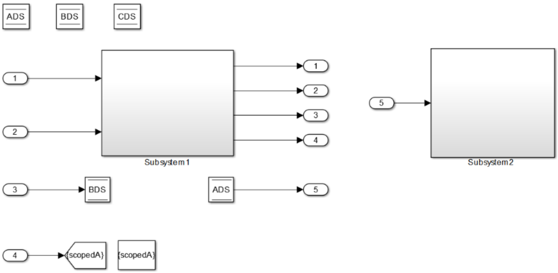


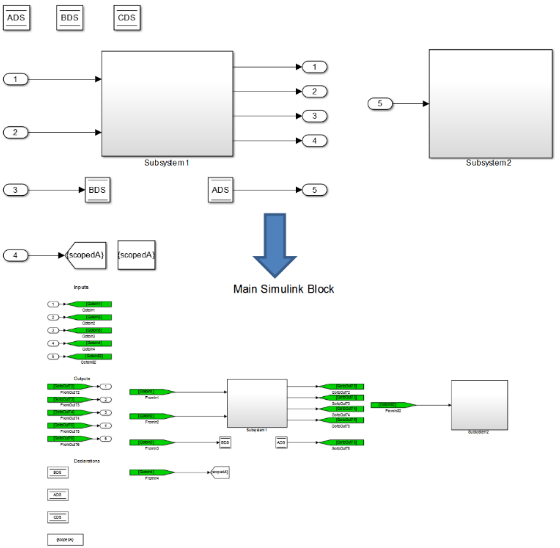
Figure: Sub1 with its strong signature (left) included

Signature Tool



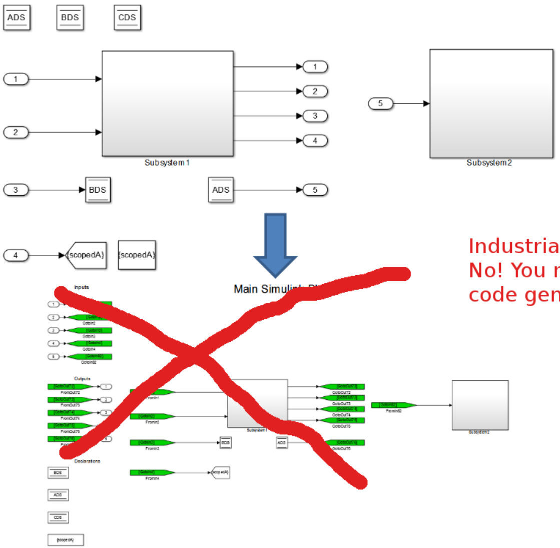
Signature Tool

Signature Tool



Signature Tool

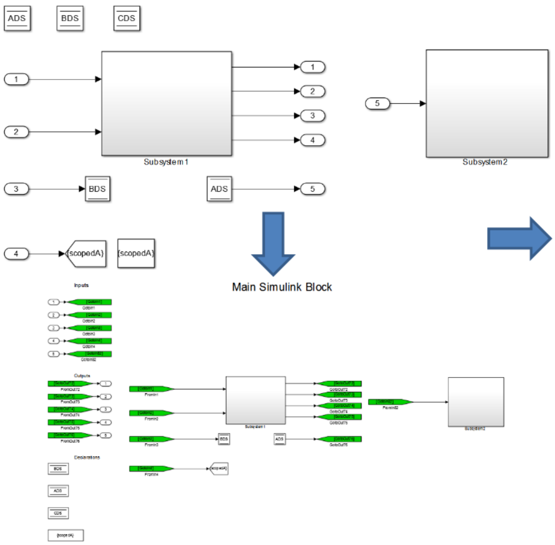
Signature Tool Defeat



Signature Tool

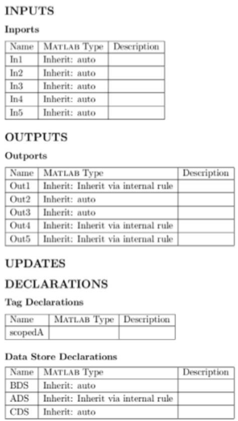
Industrial Partner:
No! You might affect
code generation.

Signature Tool Victory #1!



Signature Tool

Automatically generate Word & LaTeX doc



INPUTS

Inports

Name	MATLAB Type	Description
In1	Inherit: auto	
In2	Inherit: auto	
In3	Inherit: auto	
In4	Inherit: auto	
In5	Inherit: auto	

OUTPUTS

Outports

Name	MATLAB Type	Description
Out1	Inherit: Inherit via internal rule	
Out2	Inherit: auto	
Out3	Inherit: auto	
Out4	Inherit: Inherit via internal rule	
Out5	Inherit: Inherit via internal rule	

UPDATES

DECLARATIONS

Tag Declarations

Name	MATLAB Type	Description
scopedA		

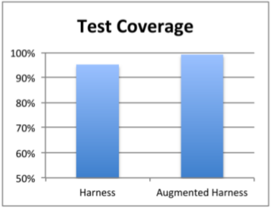
Data Store Declarations

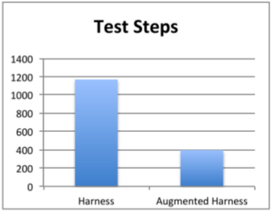
Name	MATLAB Type	Description
BDS	Inherit: auto	
ADS	Inherit: Inherit via internal rule	
CDS	Inherit: auto	

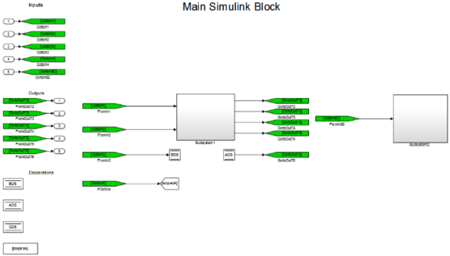
McSCert

Signature Tool Victory #2

Significantly improve testing

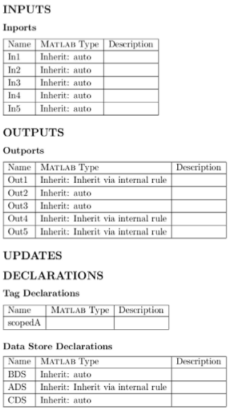





Main Simulink Block

Signature Tool

Automatically generate Word & LaTeX doc



INPUTS

Inports

Name	MATLAB Type	Description
In1	Inherit: auto	
In2	Inherit: auto	
In3	Inherit: auto	
In4	Inherit: auto	
In5	Inherit: auto	

OUTPUTS

Outports

Name	MATLAB Type	Description
Out1	Inherit: Inherit via internal rule	
Out2	Inherit: auto	
Out3	Inherit: auto	
Out4	Inherit: Inherit via internal rule	
Out5	Inherit: Inherit via internal rule	

UPDATES

DECLARATIONS

Tag Declarations

Name	MATLAB Type	Description
scopedA		

Data Store Declarations

Name	MATLAB Type	Description
BDS	Inherit: auto	
ADS	Inherit: Inherit via internal rule	
CDS	Inherit: auto	

Stupid Tool Trick # 3

> *If the tool isn't useful to your partner in the way you think it should be, ask yourself if the tool could be used in another way.*

- Listen to the engineers.
- Sometimes you have to be wrong before you get it right.

Coverage and Testing Effort Reductions

Harness Generated by Commercial Testing Tool (H)
vs. Signature-Extended Harness + Commercial Tool (EH)

	Harness (*H*)	Extended Harness (*EH*)	Improvement $(\frac{\lvert EH-H\rvert}{H} \times 100\%)$
Test Steps	1165	392	66%
Branch	88% (58/66)	100% (66/66)	14%
Decision	100% (34/34)	100% (34/34)	0%
MC/DC	95% (19/20)	95% (19/20)	0%
Boundary	100% (15/15)	100% (21/21)	0%
TOTAL	95% (126/135)	99% (140/141)	4%

Reduced # of Test Steps is significant

Before a new vehicle SW release Hardware In the Loop (HIL) access is a major bottleneck.

Application of Signatures

- Signatures can be:
  - Expressed in Simulink and included in the subsystem itself
  - Exported into separate software documentation
    - This is how our industrial partner is using the Signature Tool
- Signatures aid in comprehension
- Signatures can be employed as interface specifications for subsystems
  - Enforce encapsulation by restricting access to data (For example, by forcing a data store to be read-only)
- Can be used for inputs classification, typing, test harnessing, etc.
- Can be used to improve test coverage with reduced effort

McSCert

Signatures as Metric

Let $Size(Sig^X(S)) =$ # of inputs + # outputs + 2×# updates
Then define the signature metric as

$$d(S) = Size(Sig^w(S)) - Size(Sig^s(S))$$

- gives indication of how “modular” subsystem hierarchy is.
- $d(S)$=0 indicates that the subsystem S only has access to resources it is using
- $d(S) >> 0$ indicates that it is potentially possible for a subsystem to influence or be influenced by many other subsystems.

Signatures as Metric + Datastore Pushdown = Improved Modularity

Syst	Qnty	Before pushdown (BP)	After pushdown (AP)	$BP - AP$	$\frac{BP-AP}{BP} \times 100\%$
Sub_1	$Size(Sig^w(S))$	182	74	108	59.3
	$Size(Sig^s(S))$	12	12	0	0.0
	$d(S)$	170	62	108	63.6
Sub_2	$Size(Sig^w(S))$	231	123	108	46.8
	$Size(Sig^s(S))$	155	87	68	43.9
	$d(S)$	76	36	40	52.6
Sub_3	$Size(Sig^w(S))$	156	150	6	3.9
	$Size(Sig^s(S))$	8	6	2	25.0
	$d(S)$	148	144	4	2.7
Sub_4	$Size(Sig^w(S))$	277	169	108	39.0
	$Size(Sig^s(S))$	95	95	0	0.0
	$d(S)$	182	74	108	59.3
Sub_5	$Size(Sig^w(S))$	214	196	108	50.5
	$Size(Sig^s(S))$	57	57	0	0.0
	$d(S)$	157	49	108	68.8
Sub_6	$Size(Sig^w(S))$	220	214	6	2.7
	$Size(Sig^s(S))$	70	70	0	0.0
	$d(S)$	150	144	6	4.0
Sub_7	$Size(Sig^w(S))$	104	104	0	0.0
	$Size(Sig^s(S))$	91	91	0	0.0
	$d(S)$	13	13	0	0.0

Sub_i has 800 blocks on average, with a hierarchical depth of 6.

Stupid Tool Trick # 4

Ask yourself: “Why is that stupid tool trick is useful?”

- The stupid tool trick is useful for a reason.
- We built the data store rescoping tool to help us understand models.
- It was suprisingly useful . . . because it improved the modularity of the system.

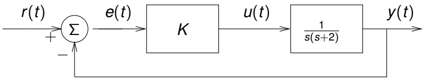
Section 5

Augmenting Existing Development Processes

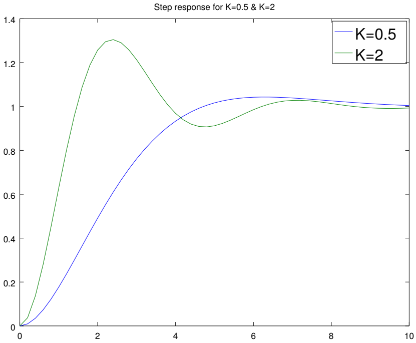
McSCert

Be like Nicky!

Automotive “Calibrations” circa 1980



1. “Tune” the gain in an analog control loop for the vehicle
2. Home ($y = 0$) position corresponds to potentiometer voltage 1.46



Automotive “Calibrations” circa 2016

1. Torque transfer function from electric motor 1 to wheels is $G(z)$
2. This product doesn’t have an onboard charger - disable the onboard charger code

“He’s dead Jim.”
“No, he’s deactivated Bones.”

How this subproject started

Academic: (writing grant) Project deliverable: Use of symbolic tools for SE of CPS

Academic: We think this would be really useful to you.

Industry: Not interested. We use Matlab/Simulink.

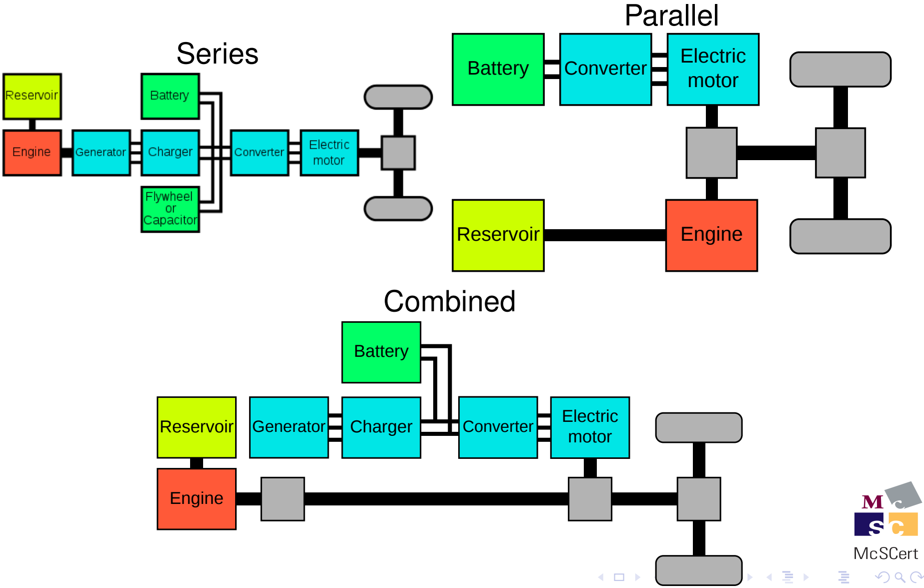
Academic: Okay. What can we do for you?

Industry: Help us make sure our Hybrid Electric Drive train software can be used across all our current & future product line.

Academic: OK. I think that if we use symbolic tools to model all of the different architectures we can help determine if your drive train architecture hardware hiding module will work.

McSCert

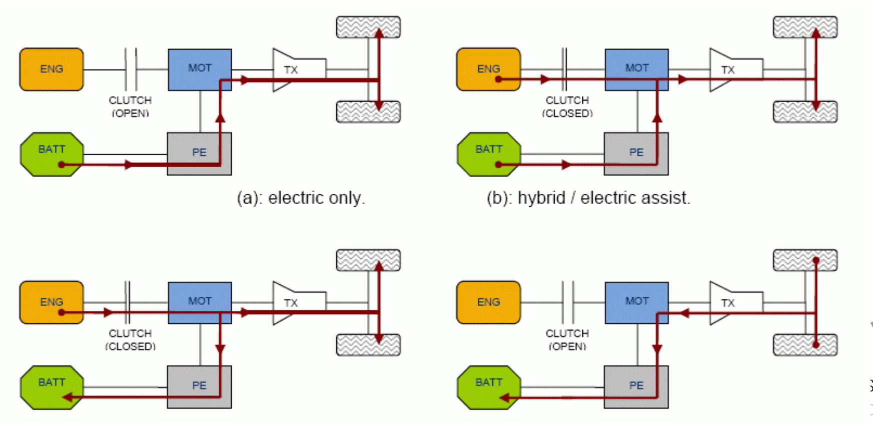
A Few Hybrid Drive train Architectures [Wikipedia]



McSCert

Parallel Hybrid Modes [Wikipedia]

For a given architecture there are many different modes



(a): electric only.

(b): hybrid / electric assist.

How current "calibration" of architecture hardware hiding module works

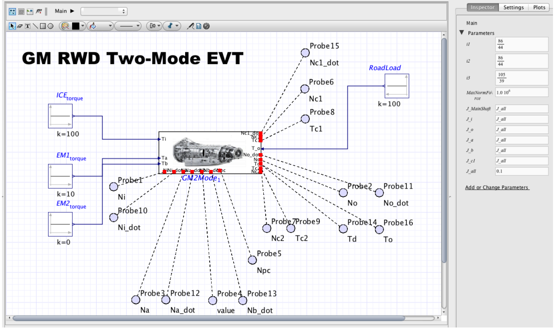
1. Derive equations for different modes by hand and linearize
2. Use in house Matlab scripts to produce relations and transfer functions between different relevant quantities
3. Check equations and transfer functions by hand
4. Encode equations for transfer functions as calibration in hardware hiding module

MapleSim for a Powertrain Architecture Model

A first attempt

1. The powertrain architecture was modeled using MapleSim
2. State-space representations of the four operation modes (M1/M2/MB/M12) were retrieved
3. Speed and torque equations (in fully symbolic form) defining the four operation modes were retrieved
   - Equation retrieval is fully automated using a Maple script
   - Equations were verified against the equations from inhouse matlab scripts and they are identical
   - Finished generating and validating – calibration coefficients are identical (within 1e^{-5} tolerance)

McSCert



Main

Parameters

i1	$\frac{86}{44}$
i2	$\frac{86}{44}$
i3	$\frac{105}{39}$
MaxNormForce	$1.0\ 10^{6}$
J_MainShaft	J_all
J_i	J_all
J_o	J_all
J_a	J_all
J_b	J_all
J_cl	J_all
J_all	0.1

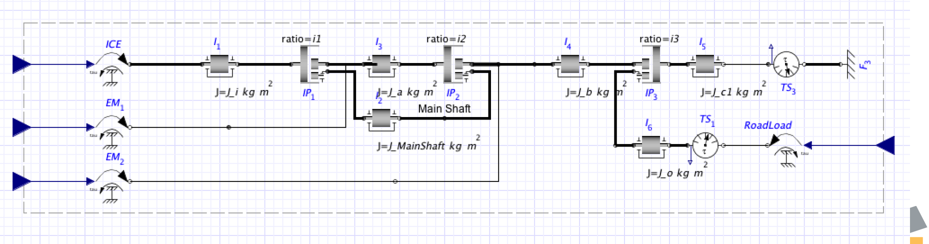
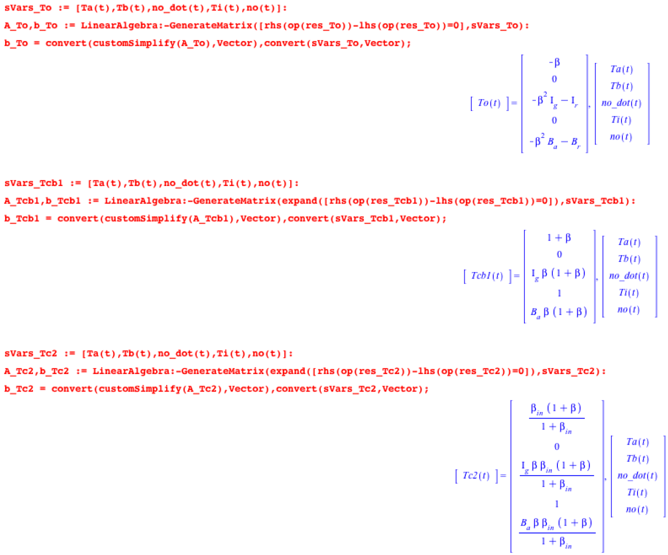


Figure: MapleSim Model for a Powertrain Architecture (MB)



```
sVars_To := [Ta(t),Tb(t),no_dot(t),Ti(t),no(t)]:
A_To,b_To := LinearAlgebra:-GenerateMatrix([rhs(op(res_To))-lhs(op(res_To))=0],sVars_To):
b_To = convert(customSimplify(A_To),Vector),convert(sVars_To,Vector);
```

$$\left[\ To(t)\ \right] = \begin{bmatrix} -\beta \\ 0 \\ -\beta^2 I_g - I_r \\ 0 \\ -\beta^2 B_a - B_r \end{bmatrix}, \begin{bmatrix} Ta(t) \\ Tb(t) \\ no\_dot(t) \\ Ti(t) \\ no(t) \end{bmatrix}$$

```
sVars_Tcb1 := [Ta(t),Tb(t),no_dot(t),Ti(t),no(t)]:
A_Tcb1,b_Tcb1 := LinearAlgebra:-GenerateMatrix(expand([rhs(op(res_Tcb1))-lhs(op(res_Tcb1))=0]),sVars_Tcb1):
b_Tcb1 = convert(customSimplify(A_Tcb1),Vector),convert(sVars_Tcb1,Vector);
```

$$\left[\ Tcb1(t)\ \right] = \begin{bmatrix} 1+\beta \\ 0 \\ I_g\,\beta\,(1+\beta) \\ 1 \\ B_a\,\beta\,(1+\beta) \end{bmatrix}, \begin{bmatrix} Ta(t) \\ Tb(t) \\ no\_dot(t) \\ Ti(t) \\ no(t) \end{bmatrix}$$

```
sVars_Tc2 := [Ta(t),Tb(t),no_dot(t),Ti(t),no(t)]:
A_Tc2,b_Tc2 := LinearAlgebra:-GenerateMatrix(expand([rhs(op(res_Tc2))-lhs(op(res_Tc2))=0]),sVars_Tc2):
b_Tc2 = convert(customSimplify(A_Tc2),Vector),convert(sVars_Tc2,Vector);
```

$$\left[\ Tc2(t)\ \right] = \begin{bmatrix} \dfrac{\beta_{in}\,(1+\beta)}{1+\beta_{in}} \\ 0 \\ \dfrac{I_g\,\beta\,\beta_{in}\,(1+\beta)}{1+\beta_{in}} \\ 1 \\ \dfrac{B_a\,\beta\,\beta_{in}\,(1+\beta)}{1+\beta_{in}} \end{bmatrix}, \begin{bmatrix} Ta(t) \\ Tb(t) \\ no\_dot(t) \\ Ti(t) \\ no(t) \end{bmatrix}$$

Figure: Part of the Maple Script for Equation Retrieval (MB)

McSCert

How new “calibration” of architecture hardware hiding module works

1. Model in Maplesim
2. Extract equations from model
3. Solve for equations and transfer functions
4. Check equations and transfer functions by hand
5. Encode equations for transfer functions as calibration in hardware hiding module

The next step is to replace the manual check using automatically generated PVS code. This has been done and proofs are largely automated.

Automated Checking of Generated Calibrations

	Architecture					
# of	BSG		Arch 2		Arch 3	
Equations	Speed	Torque	Speed	Torque	Speed	Torque
Generated	5	26	8	60	5	56
Solved for	2	3	5	4	2	3
AutoProved	Yes	Yes	Yes	1/4	Yes	No

McSCert

Stupid Tool Trick # 5

If at first you don't succeed, try, try again.

Be like Nicky.

Eventually you can make it work.

Collaborators

Many people are responsible for this work, such as:

- Faculty: Alan Wassyng, Tom Maibaum, Jacques Carette
- Research Engineers: Vera Pantelic, Alex Korobkine, Steven Postma
- Postdocs: Marc Bender, Jason Joskola
- Students: Monika Bialy, Romi Bomier, Kevin Bruer, Matthew Dawson, Colin Eles, Bennet Mackenzie, Jeff Ong, Alexander Schaap, . . .

Conclusions

Remember stupid tricks might have interesting outcomes.