

MathWorks
**AUTOMOTIVE
CONFERENCE 2024**
North America

Using Matlab Function Blocks and Bus Element Ports with Simulink to streamline algorithm development

Jeff Runde, Allison Transmission



Purpose

- To present and discuss some Simulink modeling styles used at Allison Transmission
 - Matlab Function Blocks (MFB's)
 - Bus Element Ports (BEP's)
 - and scripts!

My purpose today is to share our experiences with these topics

Agenda

- Intro/Background
 - History of Algorithm Design at Allison Transmission
- C-code vs Matlab Function Blocks vs Simulink Basic Blocks¹
 - Comparison of these 3 approaches
- Bus Element Ports
 - Using BEP's and strategies to simplify merges

1. Basic Blocks meaning Simulink primitive blocks such as summers, multipliers, etc.

Allison Algorithm Design History

Initial Design

Code: Assembly
Spec: Microsoft Word

1980's

- Ad hoc control algorithms – simplistic and not overly complex
 - MS Word was an adequate spec tool

Allison Algorithm Design History

Evolution to Structured Analysis Spec Tool

Code: C-code

Spec: Structured Analysis with
Data Flow Diagrams (DFD's) &
Control Flow Diagrams (CFD's)

1990's

- Reformulated Control Algorithms to strong Physics-based FeedForward Models
 - More complex algorithms with more data to properly manage
- Developed/Defined the algorithms using a Structured Analysis Spec Tool^{1 2}
 - Structured Analysis was a proven design approach that handles problems with higher complexity
 - Provided a hierarchical functional-decomposition of algorithms
 - Uses Data Flow Diagrams (DFD's) to design the data dependencies between algorithms
 - Uses Control Flow Diagrams (CFD's) to design the system modes/states to control program execution

In essence, we were using a spec process that essentially was much like Simulink

Allison Algorithm Design History

Code becomes our spec

Code: C-code

Spec: C-code

2000's

- Algorithm engineers began directly writing software
 - Initially we also maintained our Structured Analysis spec
 - We found we were often designing first directly in the code itself
 - This lead to specs not being updated
- Over time, we eventually stopped maintaining our Structured Analysis specs

However, we also understood we had lost our spec design tool

Allison Algorithm Design History

Conversion to Simulink as Spec tool with automatic code generation

Code: C-code
(via SL code gen)
Spec: Simulink

2010's

- We began conversion of our core control algorithms to Simulink to regain our spec
- 1st Question: Convert from C to Simulink Basic Blocks or MFB's
 - Converting to Basic Blocks would have been a very large undertaking
 - Instead, c-code was directly pasted into MFB's
 - C-code syntax was transformed into m-file syntax
 - This was done mostly with scripts, along with some manual conversion as well
 - This greatly simplified the conversion process
- As we continued this process, debate remained concerning
 - *If converting from C to MFB's, why not just code in C, and/or*
 - *Why convert to MFB's and not to Simulink Basic Blocks*

Comparison of C-code, MFB's, Simulink Basic Blocks

	Characteristics of a good development toolset	C-Code	MFB's	Basic Blocks
Characteristics that supports good software design	Hierarchy	1*		
	Algo Processing Dependency determined by data analysis (DFD's)	2*		
	Decomposition of Operating Modes and SystemStates (CFD's)	3*		
	Testability	4*		
Characteristics that work well for a team environment	Search			5*
	Differencing			6*
	Merging			7*
	Clear Algo Description			

Best
OK
Less than OK
Poor
Personal Preference

Comparison of C-code, MFB's, Simulink Basic Blocks

	Characteristics of a good development toolset	C-Code	MFB's	Basic Blocks
Characteristics that supports good software design	Hierarchy	1*		
	Algo Processing Dependency determined by data analysis (DFD's)	2*		
	Decomposition of Operating Modes and SystemStates (CFD's)	3*		
	Testability	4*		
Characteristics that work well for a team environment	Search			5*
	Differencing			6*
	Merging			7*
	Clear Algo Description			

Best
OK
Less than OK
Poor
Personal Preference

Characteristics that supports good software design

(or, if using MFB's, why not just code in C)

- As a Structured Analysis design approach, Simulink provides
 - Hierarchy
 - Presents hierarchy in a graphical view, showing big picture view of the control algorithms
 - Supports functional decomposition design of the control algorithms
 - Data flow diagrams
 - Guarantees proper causal order of data and the equations calculating data
 - Control flow diagrams
 - Provides clear and distinct program control flow separate from algorithm data flow

- *Flat C-code does not show these well*

- Additionally, Simulink also
 - Generates code of model of algorithms
 - Is extremely testable, from low level (unit tests) to high level (full MIL/SIL/PIL testing)
 - Executes nearly the entire V process up and down!

Comparison of C-code, MFB's, Simulink Basic Blocks

	Characteristics of a good development toolset	C-Code	MFB's	Basic Blocks
Characteristics that supports good software design	Hierarchy	1*		
	Algo Processing Dependency determined by data analysis (DFD's)	2*		
	Decomposition of Operating Modes and SystemStates (CFD's)	3*		
	Testability	4*		
Characteristics that work well for a team environment	Search			5*
	Differencing			6*
	Merging			7*
	Clear Algo Description			

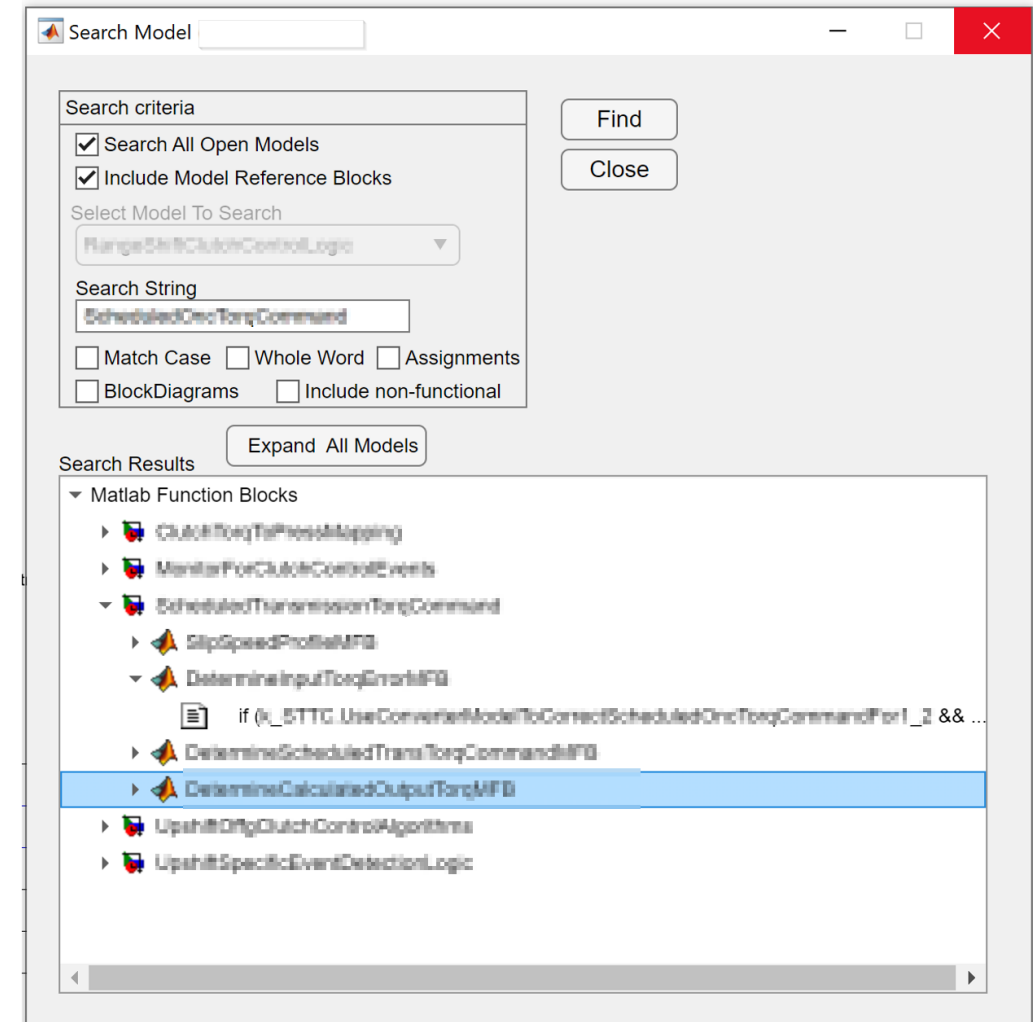
Best
OK
Less than OK
Poor
Personal Preference

Searching

- **MFB's**
 - Custom search script can be created to provide similar search capability to modern IDE's
(such as MS Visual Studio)

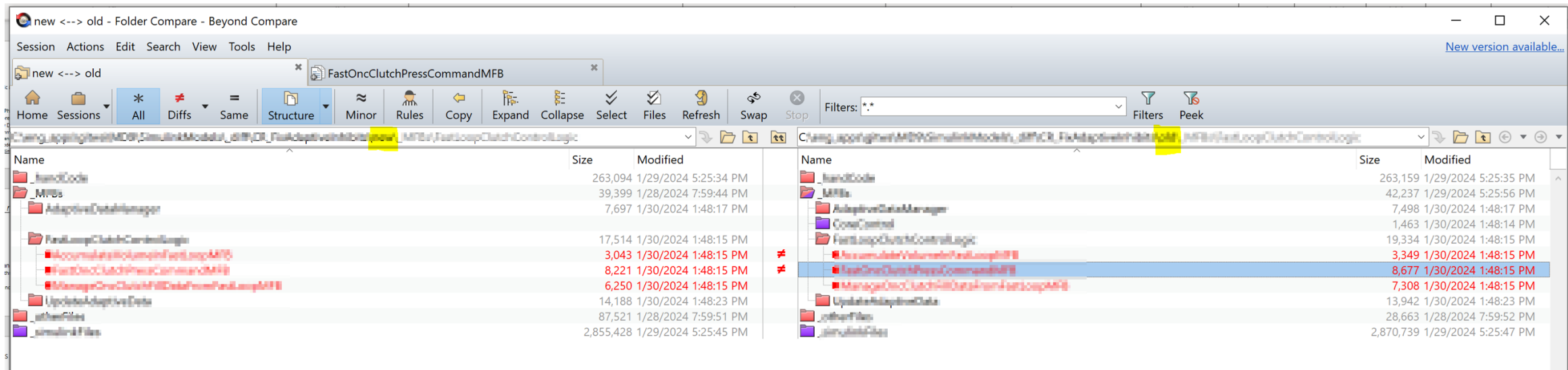
- **SL Basic Blocks**
 - Simulink Find function does not provide this same search capability as modern IDE's

Custom MFB Search GUI



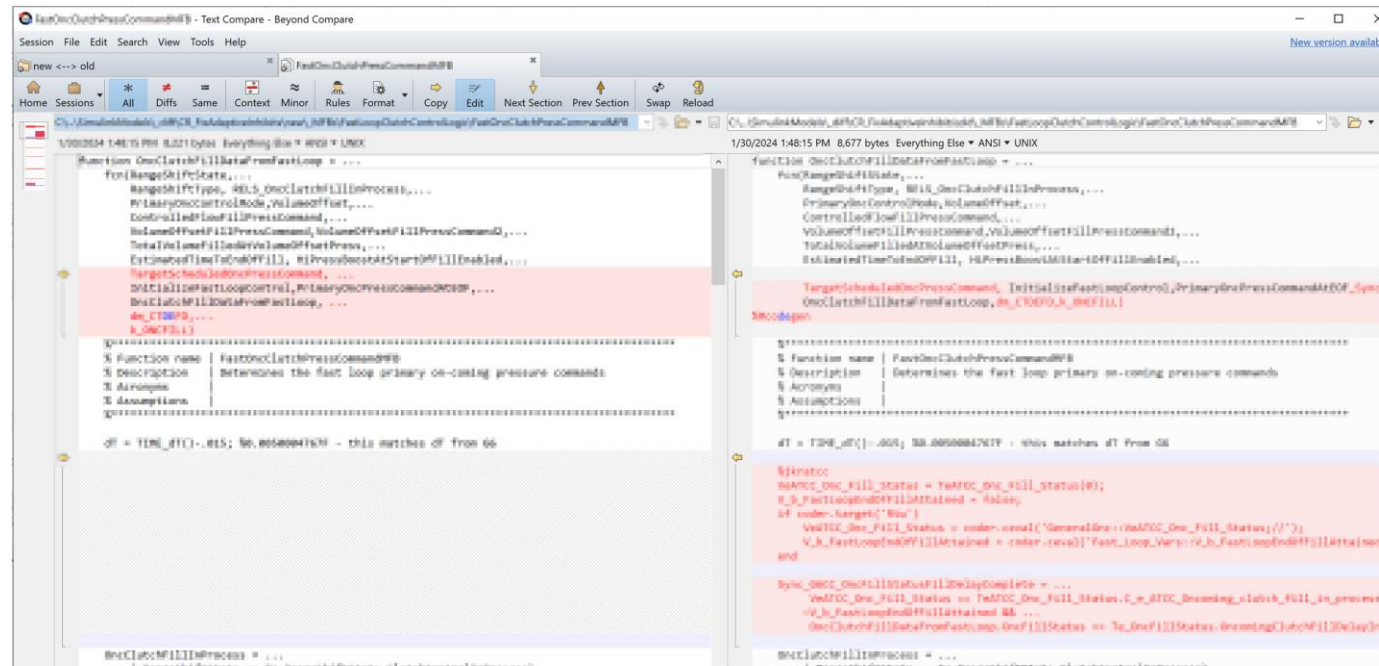
Differencing

- MFB's
 - Custom differencing script can be created to determine all models changed and create a text file of all MFB's for each model
 - The script can then call a 3rd party diff tool to perform a text difference of the changes
(*showing below using Beyond Compare as diff tool, which searches entire directories*)
- SL Basic Blocks
 - Vsdiff does not provide same differencing ability as modern text diff tools




Merging

- MFB's
 - Beyond Compare can then display the text difference between MFB's and can perform 3-way merging
- SL Basic Blocks
 - Viddiff does not provide same merge ability as modern 3-way text merge tools, but Simulink has a 3-way merge tool



Comparison of C-code, MFB's, Simulink Basic Blocks

	Characteristics of a good development toolset	C-Code	MFB's	Basic Blocks
Characteristics that supports good software design	Hierarchy	1*		
	Algo Processing Dependency determined by data analysis (DFD's)	2*		
	Decomposition of Operating Modes and SystemStates (CFD's)	3*		
	Testability	4*		
Characteristics that work well for a team environment	Search			5*
	Differencing			6*
	Merging			7*
	Clear Algo Description			



- For nearly all of us here, I think we would collectively agree that lines 1-4 are reasons why we view a design using Simulink as superior over a design using C-code
- I would say also that a requirement for nearly any production program used in a team environment is that Searching, Differencing, and Merging be met by the development toolset used.

MFB Summary

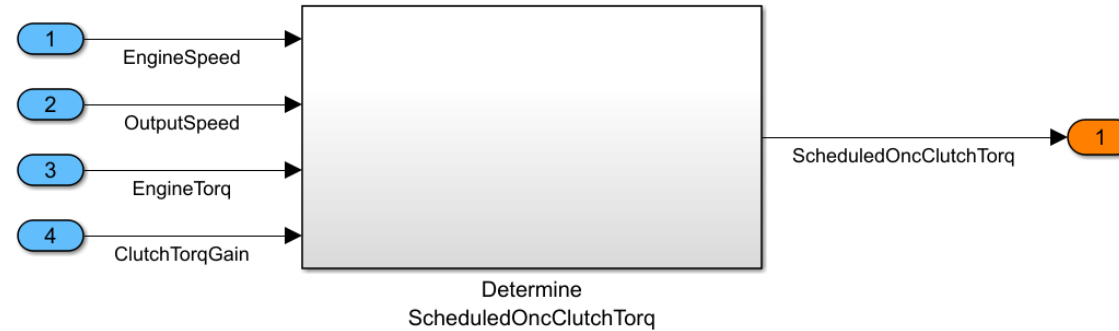
- At the 2018 MAB Conference in Natick, MA, Fred Smith, Mathworks, presented MFB's as "*The most beautiful block in Simulink*"
 - Fred's discussion focused on how MFB's are general, allowing them to be used for almost any algorithm calculation as found desired by a designer
- Additionally, we have also discussed how
 - MFB-based Simulink design brings the best of both worlds of
 - Structured Analysis design approach of Simulink
Strong design methodology through DFD's, CFD's, code generation, testing
 - Text-based programming tools
Highly functional tools for searching, differencing, merging
 - And an MFB based design vs a design in c-code **are not equivalent**, because Simulink inherently provides a Structured Analysis design foundation at its core

Next: Using BEP's to Simplify Merging

- Merging
 - In our experience, merging is a difficult pain point
 - As was just presented, MFB's can simplify the merge process using 3-way text merge tools
 - But this does not solve the merge problem of higher level block diagrams in the hierarchy
 - In this discussion, BEP's are used to reduce the number of diagrams to be merged

- To understand how we use BEP's to simplify merging, we will discuss
 - Purposes of Data Flow in both Simulink and a Structured Analysis design
 - Subsystems using Bus I/O
 - *How many buses are required*
 - Using a single BEO per subsystem
 - *And its effect on simplifying the merge process*
 - Allison Example from converting a Standard Bused System to BEP's
 - *Why we like BEP's*

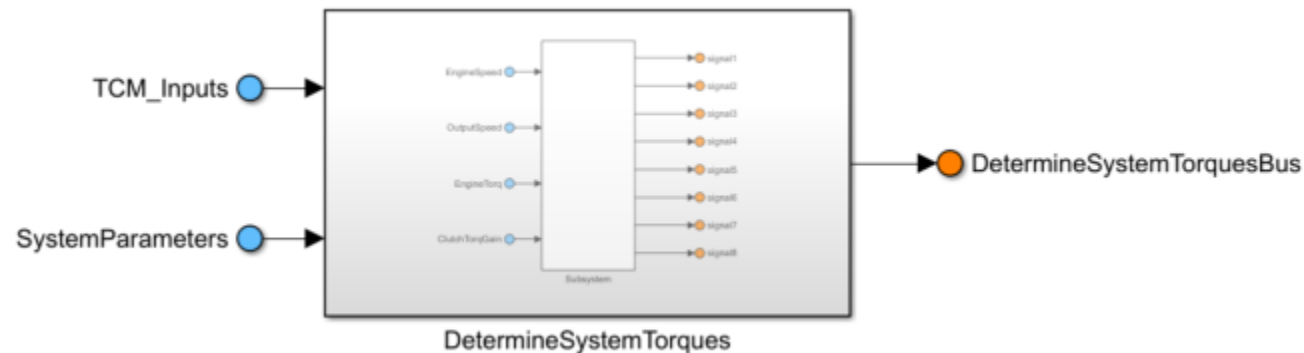
Classic Simulink Subsystem I/O



- In this classic view of a Simulink Subsystem, data flow is used to show the functional inputs and outputs of a system
- In this example,
$$\text{ScheduledOncClutchTorq} = f(\text{EngineSpeed}, \text{OutputSpeed}, \text{EngineTorq}, \text{ClutchTorqGain})$$

Bused Data Flow

- With a complex control algorithm design
 - There typically is too much data to pass all signal lines between subsystems
 - The designer will then typically use buses to simplify the view

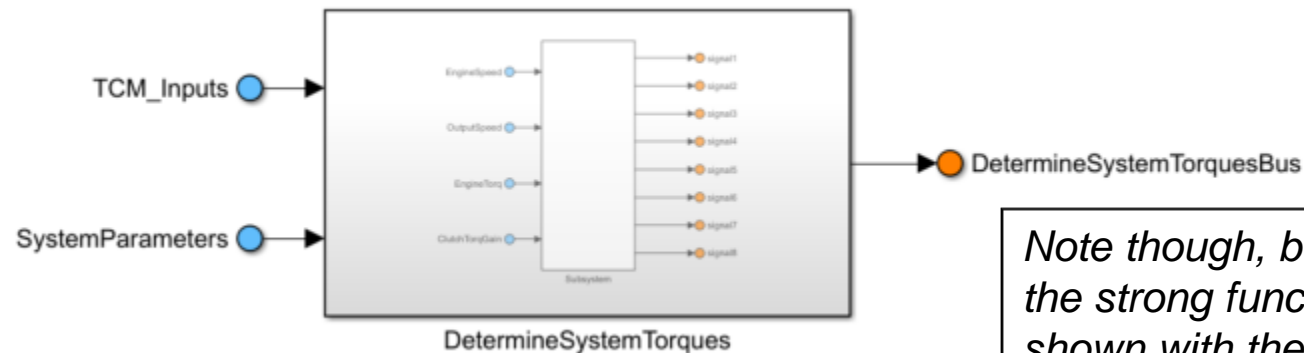


- A fundamental question with Buses
 - What number of bus I/O is ideal
 - *Input Buses* – you will use as many as required
 - *Output Buses* – you get to choose as many as desired

This means you choose the number of output buses as you see fit (e.g., organize them in a way that works best for you)

Bused Data Flow

- With a complex control algorithm design
 - There typically is too much data to pass all signal lines between subsystems
 - The designer will then typically use buses to simplify the view



Note though, buses begin to lose the strong functional description as shown with the Classic I/O example

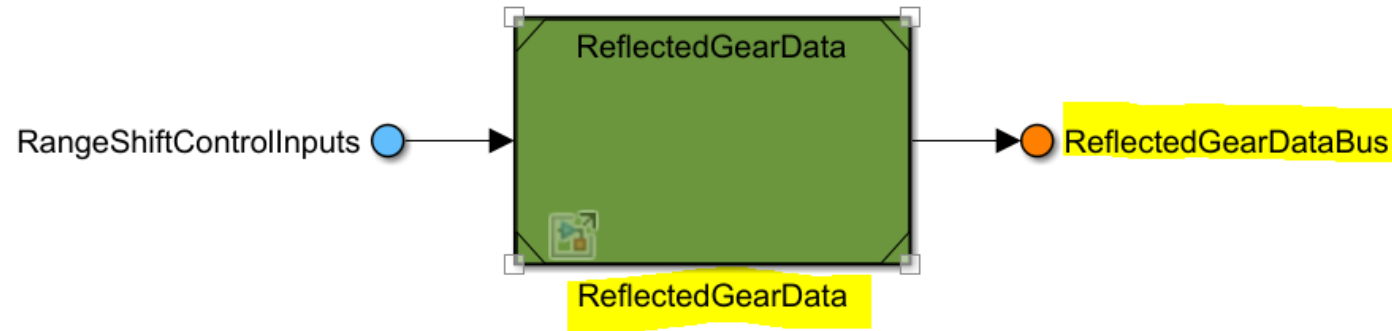
- A fundamental question with Buses
 - What number of bus I/O is ideal
 - *Input Buses* – you will use as many as required
 - *Output Buses* – you get to choose as many as desired

This means you choose the number of output buses as you see fit (e.g., organize them in a way that works best for you)

Purpose of Data Flow in a Structured Analysis Design

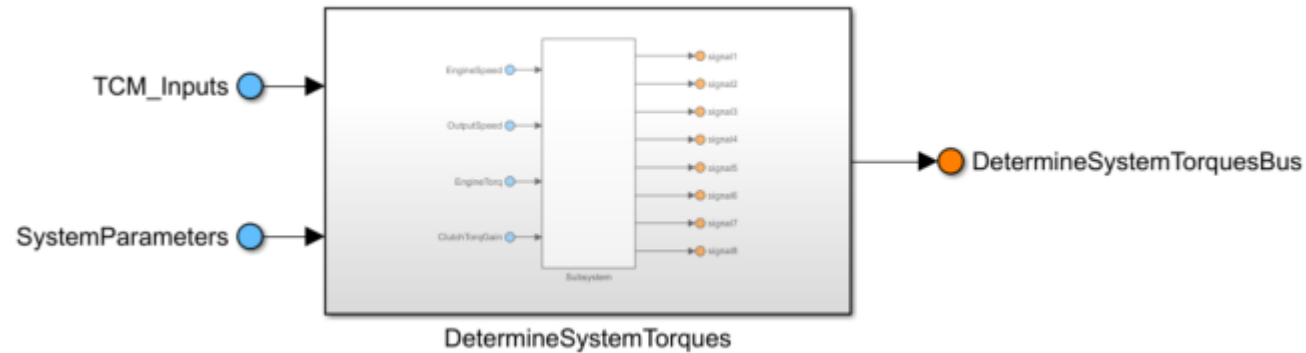
- In a Structured Analysis design, data flow is used to determine algorithm processing order for a proper design
- Since Simulink is inherently a data flow design process, this also is used to determine valid processing order as well
 - Simulink will generate an error when data that has not been calculated is routed into a process
(This is known as the classic Algebraic Loop in Simulink)
- So while data flow with a Classic Simulink Subsystem design is often thought of to show the functional inputs to a subsystem, it is good to note that it is required to guarantee that causality with the algorithm processing order has been properly defined.
 - In other words, while it is nice that data flow shows the detailed functional relationship of the outputs to the inputs, it is required because it is data flow which defines the processing order of the algorithms in a subsystem and the data being calculated

Using Single Bus Output for each Subsystem



- By having only 1 output bus for each subsystem, the bus naming can be “predetermined”
 - As such, it is easy to develop scripts to help automate the process
 - In our process, we have developed scripts to
 - Automatically name the bus as *[SubsystemName ‘Bus’]*
 - Automatically create the bus type definition (e.g., ReflectedGearDataBusType)
- A single bus output
 - Allows rules to be defined to create scripts to help automate the process
 - Satisfies the data flow causality requirement for a Structured Analysis DFD design
 - Can be seen to simplify the design, especially for a complex diagram

Bus Element Outports: *Simplify the merge process*



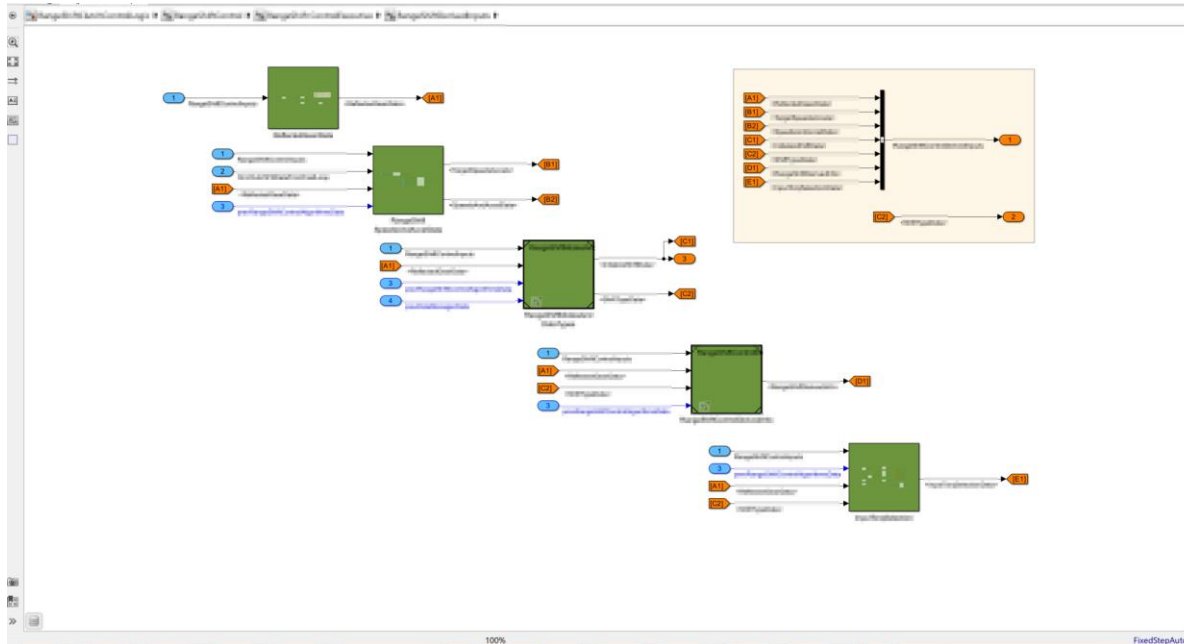
- BEO's added to the child subsystem are automatically added to the parent BEO
- In a merge process, changes in the child subsystem then do not affect the parent subsystem
- This simple rule *greatly* simplifies the merge process
 - If you are not having merge issues, this doesn't matter to you
 - If you are having merge issues, this limits the merge changes to the child subsystem

BEP Summary

- First discussed Classic view of a Simulink Subsystem and I/O
 - $\text{OutputVariables} = f(\text{ScalerInputVariables})$
- Next discussed busing to reduce complexity
 - Buses can be created to organize data however desired
- Then proposed an organization of 1 bus output per subsystem
 - This helps create rules for automated scripts to create the buses
- Finally, showed how BEP's used in this way will simplify the merge process because higher level hierarchy is not affected by lower level design changes in the data busing

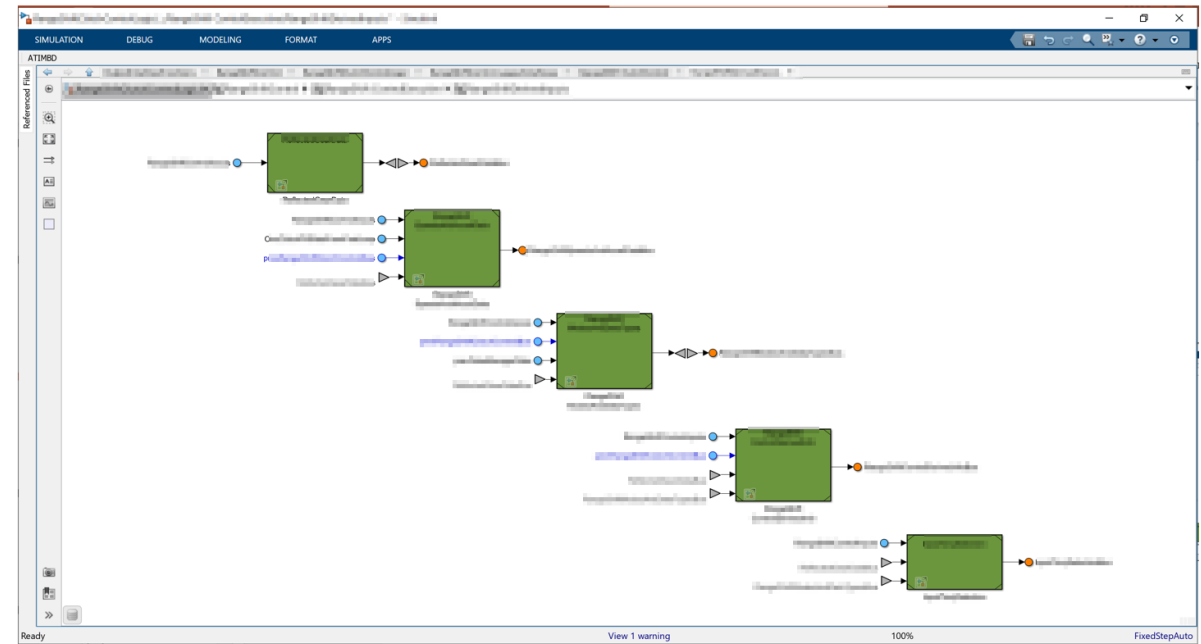
Allison Transmission Example Conversion from Standard Busing to BEP's

Standard Bus example with Bus Creator



- Here, we have multiple bus outputs for subsystems as determined by the developer

Converted to Bus Element Outputs



- Here, we have a single BEO for each subsystem

Reducing clutter in a Simulink Diagram improves clarity

MathWorks
**AUTOMOTIVE
CONFERENCE 2024**
North America

Thank you



© 2024 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [mathworks.com/trademarks](https://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

References

1. Tom DeMarco (1978), *Structured Analysis and System Specification*, Yourdon
2. Derek J. Hatley and Imtiaz A. Pirbhai (1988), *Strategies for Real-Time System Specification*, Dorset House