Developing a Multilingual Auto-coding Interface Control for the MAVERIC-II Dynamics Simulator

Mason Nixon

Jacobs Space Exploration Group

Leidos, Inc.

A Multilingual Auto-coding Interface Control for MAVERIC

- A Brief History
- Model-Based Design
- Problem Statement
- MAVERIC Architecture
- Design Process
- Case Study

A Brief History

1996 — **MAVERIC developed**

NASA Space Launch Initiative — 2000

2001 — **MAVERIC-II developed**

NASA Orbital Space Plane — 2002

Ares-I X — 2008

Space Launch System — 2011

Marshall Aerospace Vehicle Representation in C (MAVERIC)

Model-Based Design

- Consistent Documentation & Implementation
- Elimination / Reduction of Coding Translation Errors
- Continuous Verification & Validation
- Reusable Code
- **Automatic Code Generation**

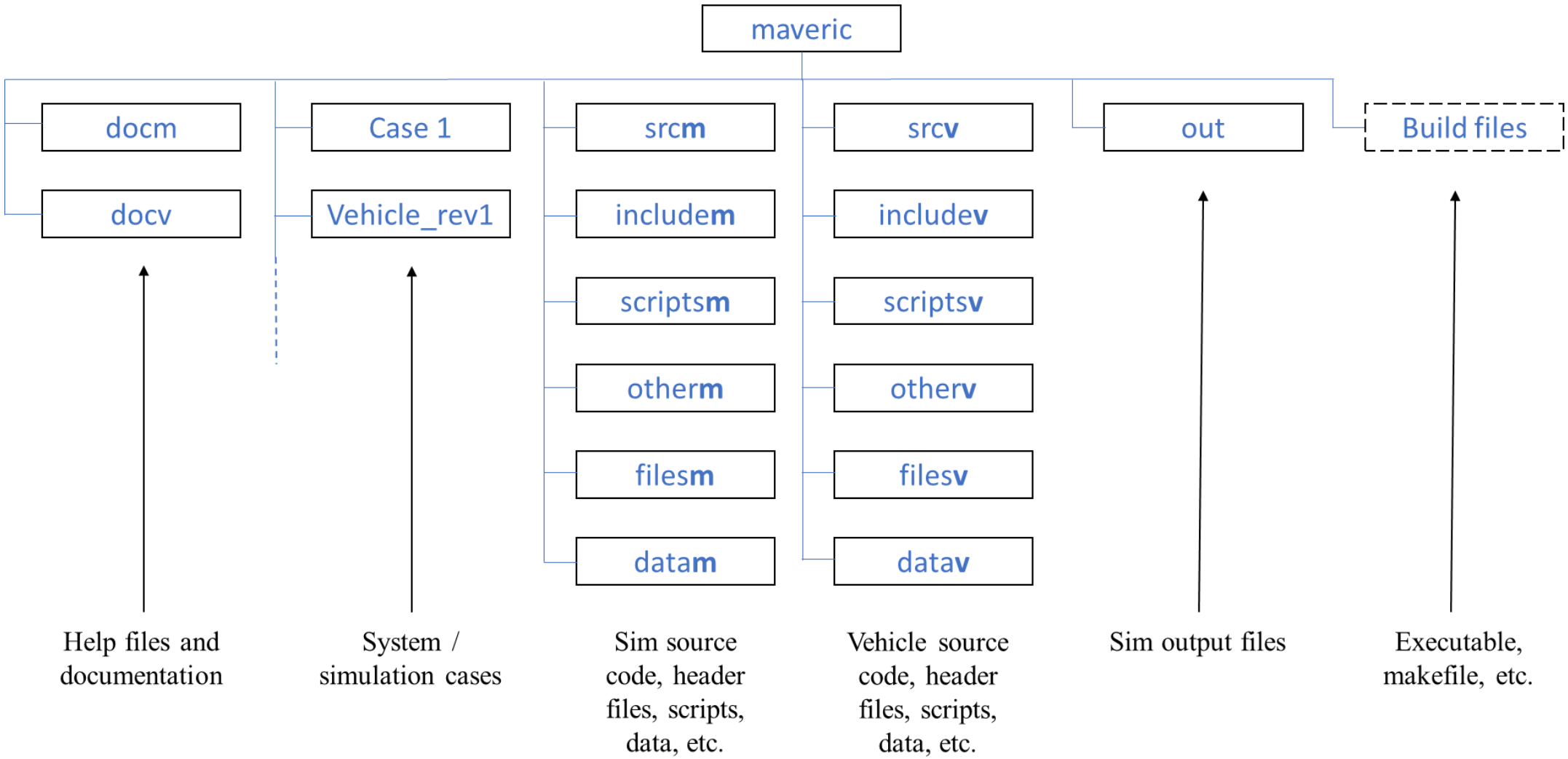
Three Steps of Model-Based Design:

1. Identify Components & Design Goals
2. Analyze System Behavior via Simulation
3. Component Design & Verification

Problem Statement

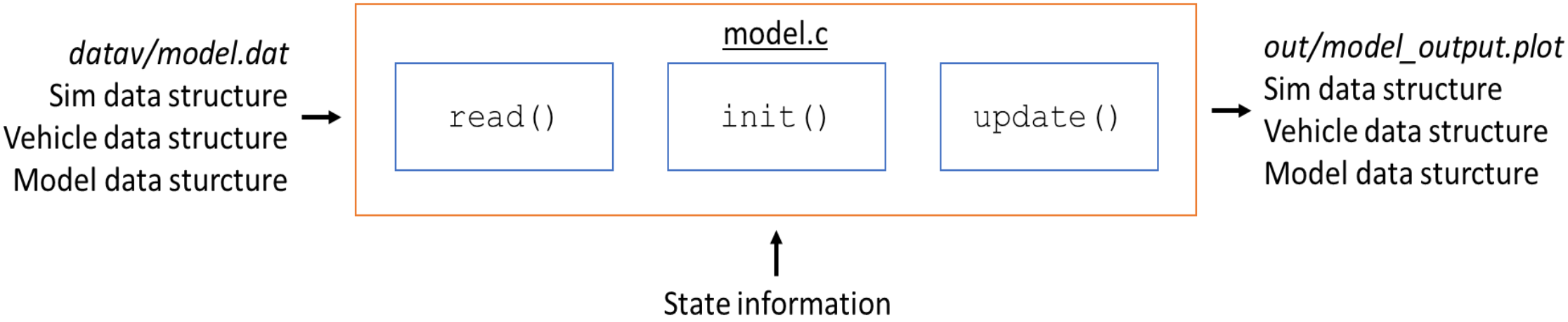
Develop a **repeatable** and **modular** design method for MAVERIC simulation model development using a **model-based design** approach

MAVERIC Architecture



MAVERIC File Structure

MAVERIC Architecture



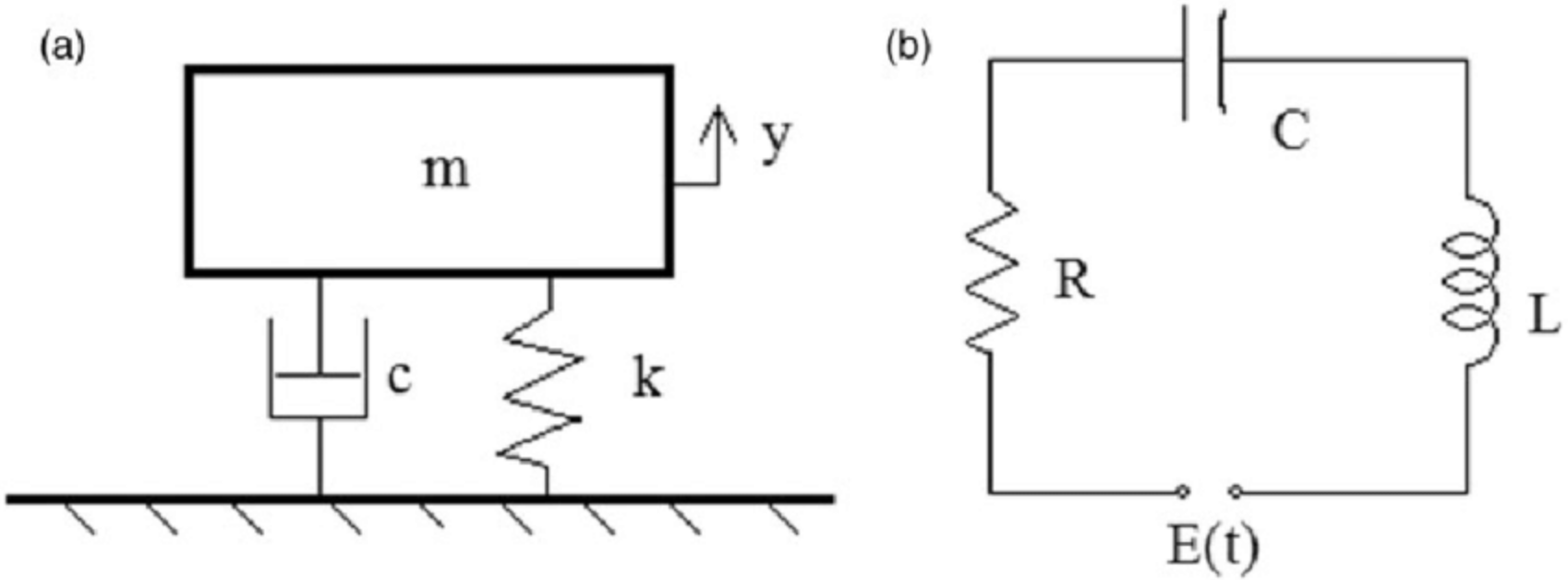
MAVERIC Model Data Flow

Design Process

- System Definition
- Inputs / Outputs
- Sampling Rate
- Implementation, Verification, & Validation

Case Study

1. Introduce the 2nd Order Oscillator model
2. Block-based Methods
3. Code-Based Methods



Case Study

Second-Order Oscillator

$$\frac{d^2x}{dt^2} + 2\zeta\omega\frac{dx}{dt} + \omega^2 x = \omega^2 u,$$

damping coefficient: $0 \leq \zeta \leq 1$

frequency of oscillation (radians): $\omega = 2\pi f$

frequency (Hertz): f

State Space Representation

Let $x_1 = x,\ \dot{x}_1 = \dot{x} = x_2$ as

$$\dot{x}_2 + 2\zeta\omega\dot{x}_1 + \omega^2 x_1 = \omega^2 u \rightarrow$$

$$\begin{bmatrix}\dot{x}_1 \\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1 \\ -\omega^2 & -2\zeta\omega\end{bmatrix}\begin{bmatrix}x_1 \\ x_2\end{bmatrix} + \begin{bmatrix}0 \\ \omega^2\end{bmatrix}u$$

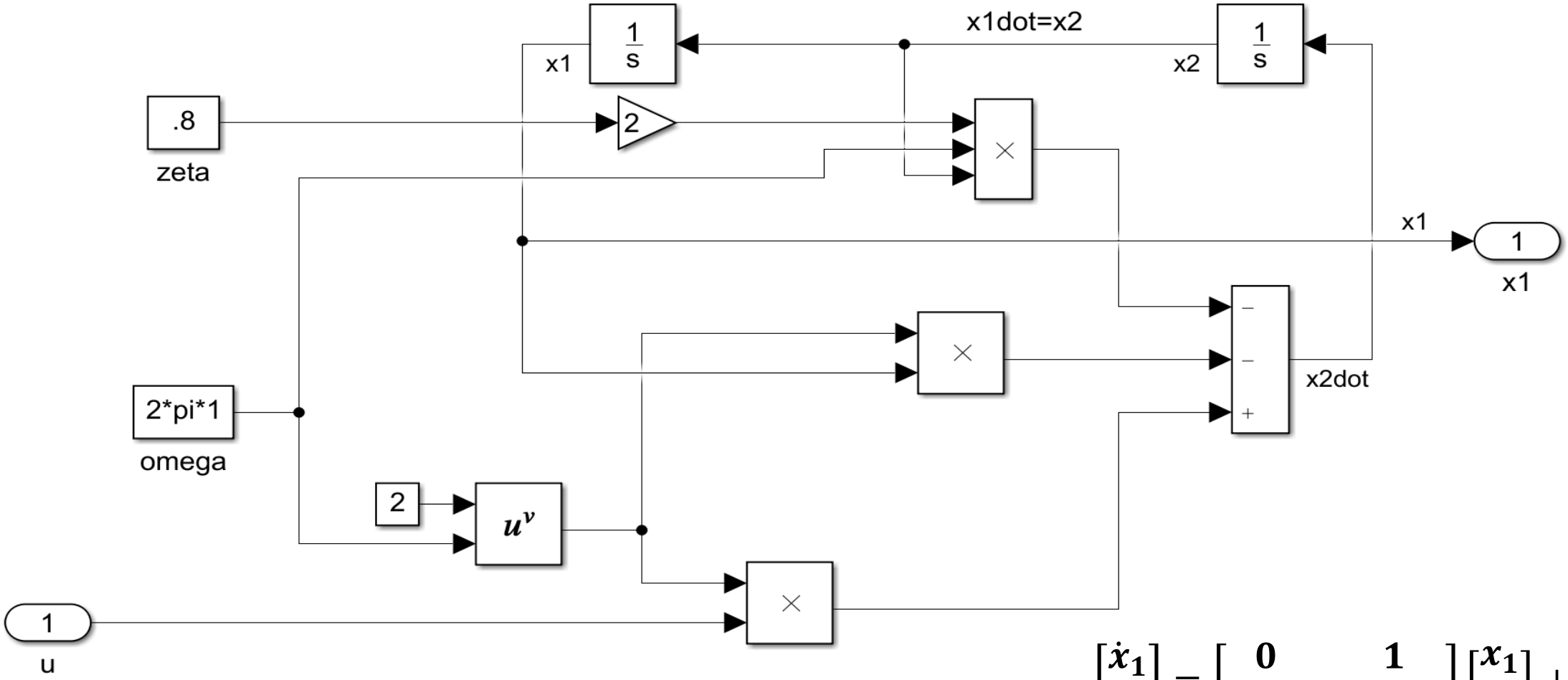
Case Study

Second-Order Oscillator State Space Representation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & -2\zeta\omega \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \omega^2 \end{bmatrix} u$$

Model-Based Method

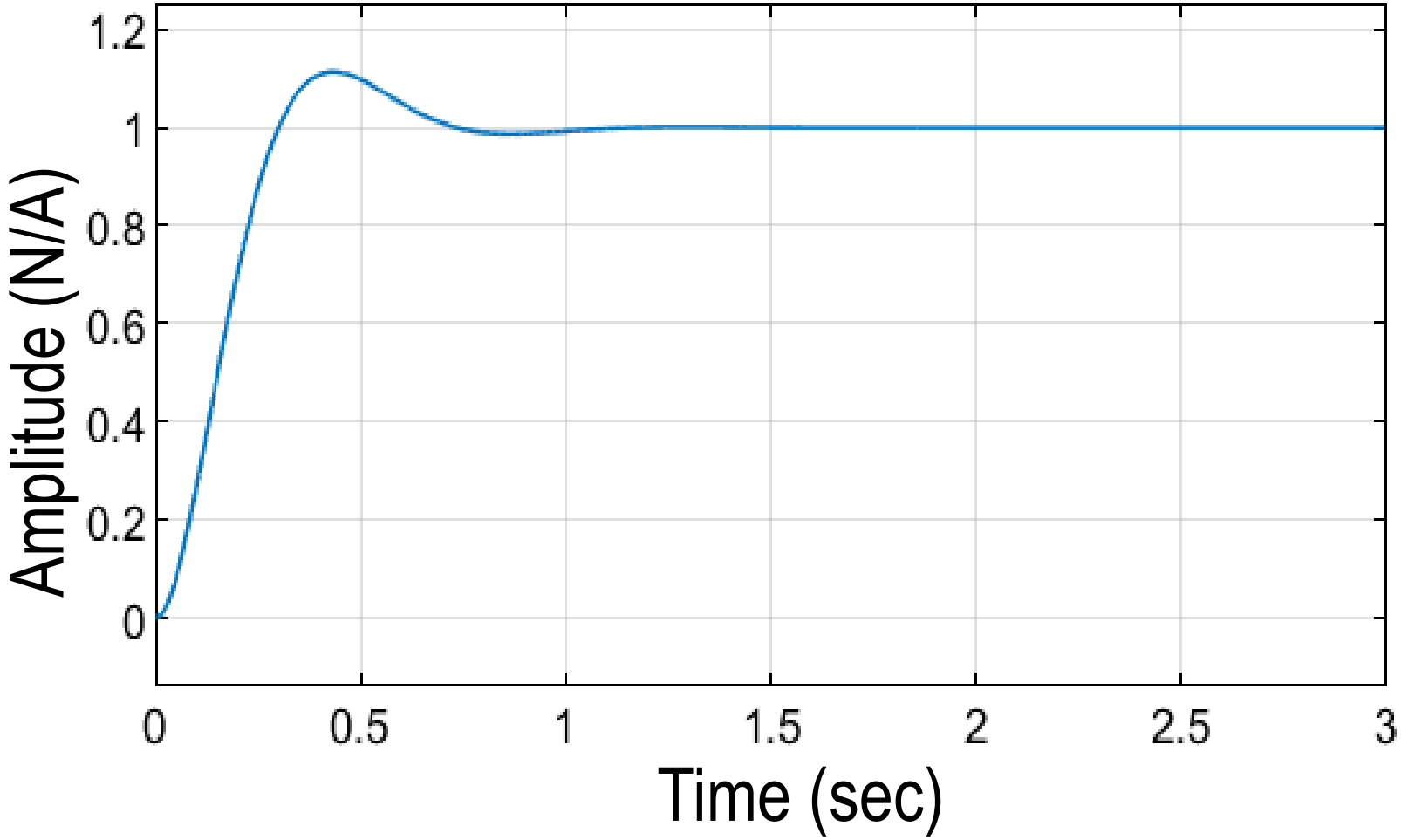
Case Study: Model-Based Method



$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & -2\zeta\omega \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \omega^2 \end{bmatrix} u$$

Second-Order Oscillator Block Diagram

Case Study: Model-Based Method

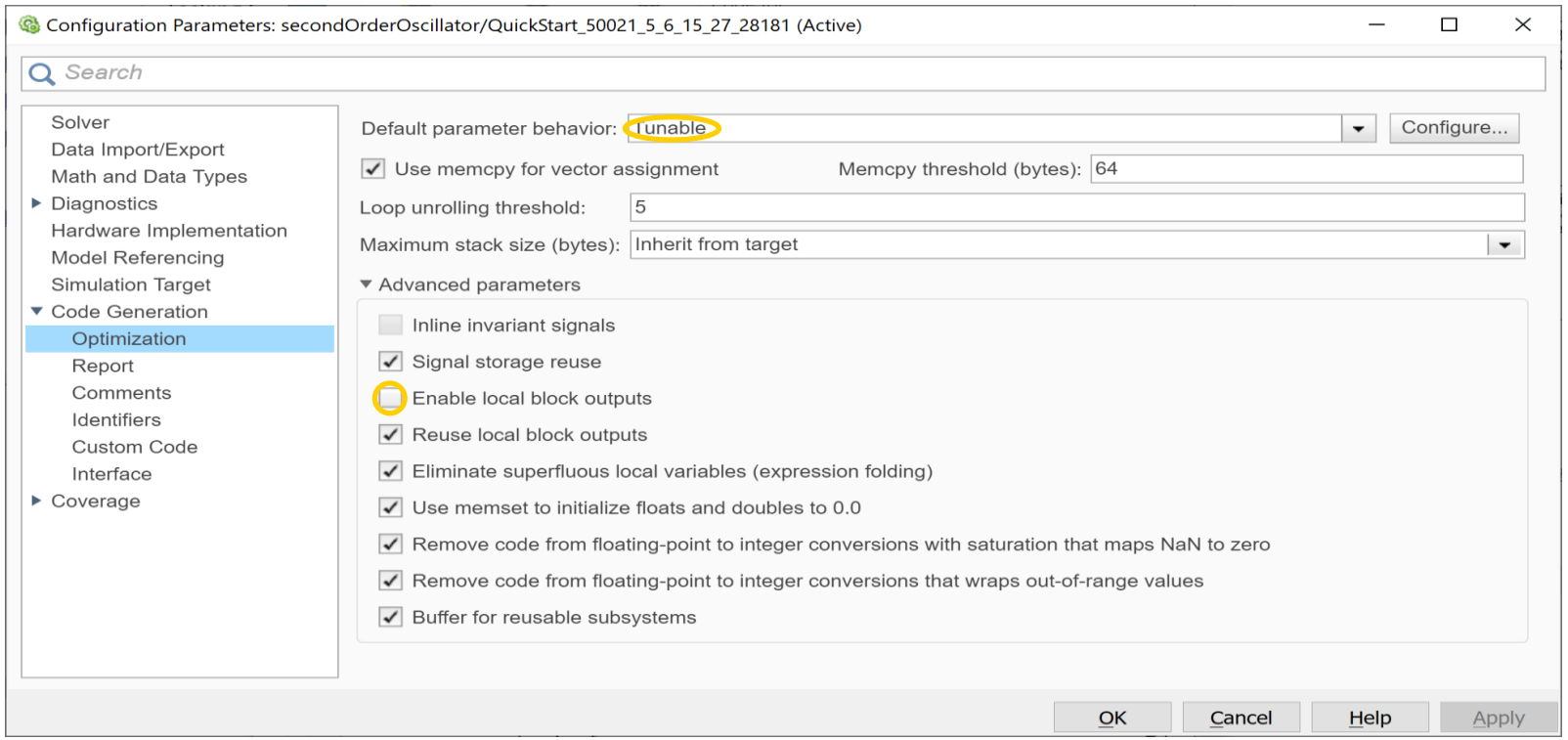


Step Input System Response

Case Study: Model-Based Method

Conversion Walk-through

- Use the Simulink Autocoder
- Construct model with input and output ports
- Note the configuration parameters: Default behavior: Tunable; Uncheck Enable local block outputs



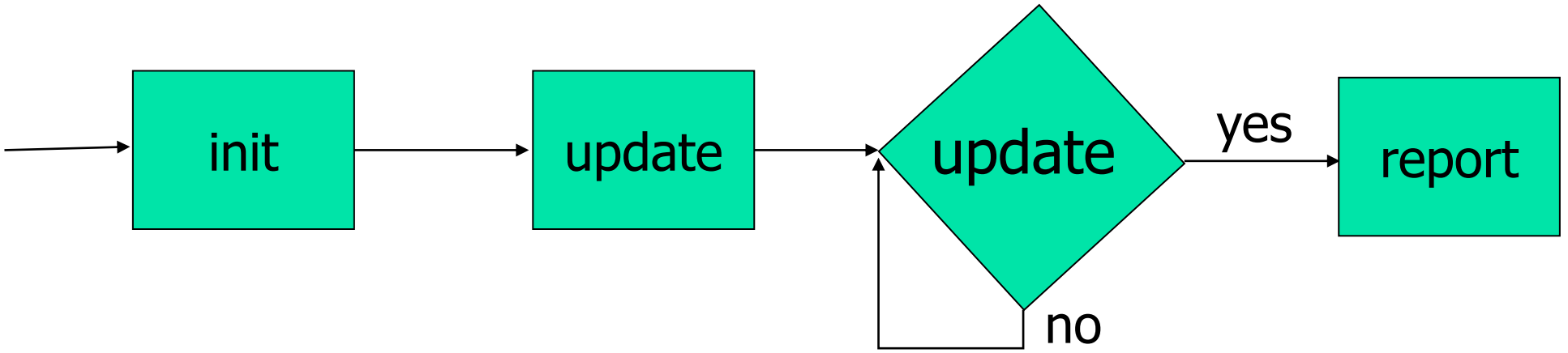
Case Study

Second-Order Oscillator State Space Representation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & -2\zeta\omega \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \omega^2 \end{bmatrix} u$$

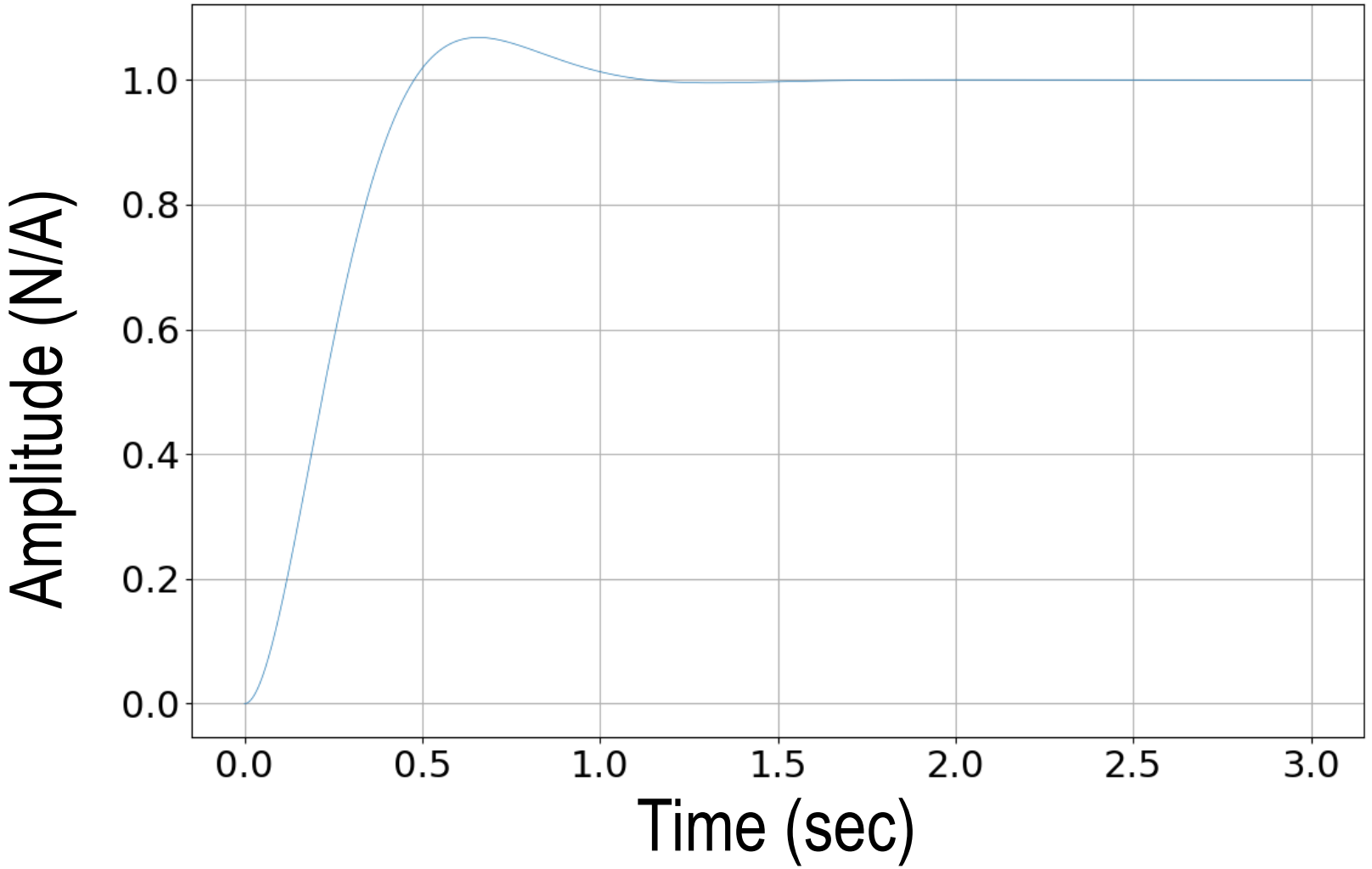
Code-Based Method

Case Study: Code-Based Method



OSK Framework Flowchart

Case Study: Code-Based Method



Python Second-Order Oscillator Step Response

Case Study: Code-Based Method

Conversion Walk-through

- Python 3 Module: *Pure Embedding*
  - Embeds the Python interpreter into the resultant C code
- Wrap the C caller function in the Python script and add the Python.h header
- Incorporate the compilation of this C code and the linker scripts to attach the Python headers into the MAVERIC Makefile

- Note that a similar approach can be used for MATLAB scripts that are structured as function calls
- The same process flow outlined can be used for any coding / scripting language with C code generation functionality

Summary

- Applied tenets of **Model-Based Design**
- Created a **repeatable** and **modular** method to reduce coding and code-translation errors
- Allowed for **continuous verification and validation** in the much simpler Python, or Simulink models
- Generated autocoding framework that is **consistent** and **repeatable**
- **Method can be generalized to other simulation environments that depend on low-level coding (i.e., C, C++, FORTRAN, etc.)**

Thank you for your time!

Mason Nixon, M.S. / Leidos, Inc.

Mason.E.Nixon@leidos.com