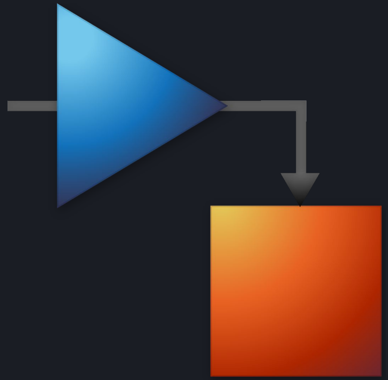
Subsystem, Solvers and code Generation Process

Session Content

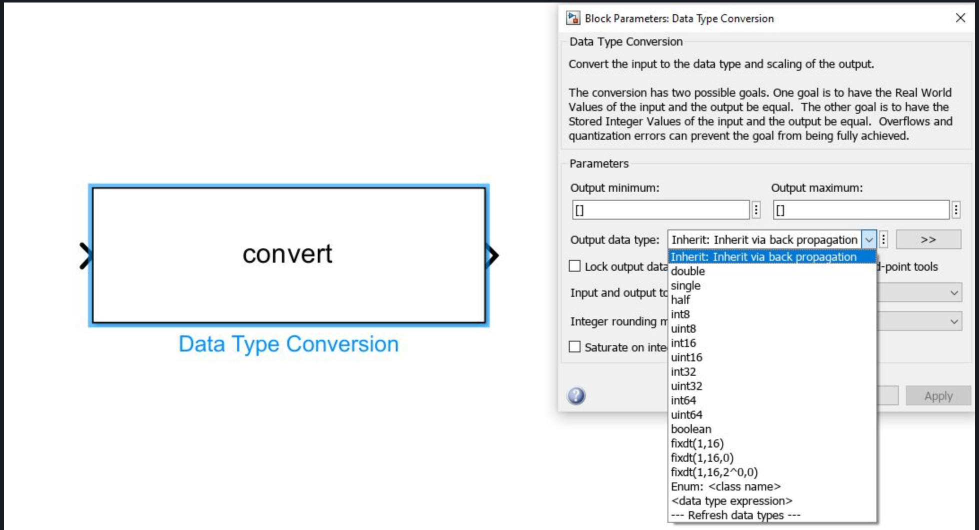
- Commonly Used Blocks
- Code Generation Process
- Subsystems



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

- Data Type Conversion
  In Simulink, the Data Type Conversion block is used to convert signals from one data type to another. This block is particularly useful when working with systems that require different data types for compatibility or precision reasons. It allows you to seamlessly convert signals between various numeric data types, such as changing from a double-precision floating-point to a single-precision floating-point or from an integer to a floating-point format.



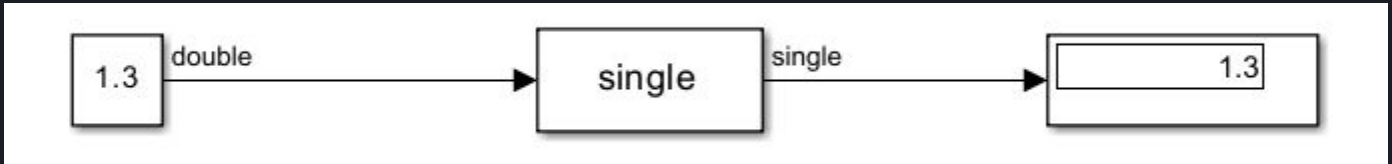
Subsystem, Solvers and code Generation Process

Commonly Used Blocks

- Data Type Conversion

  Double to Single Precision Conversion:
  - Input data type: double
  - Output data type: single

This configuration of the Data Type Conversion block converts a double-precision input signal to a single-precision output signal. This may be useful when working with systems that require lower memory usage or when interfacing with components that only accept single-precision data.

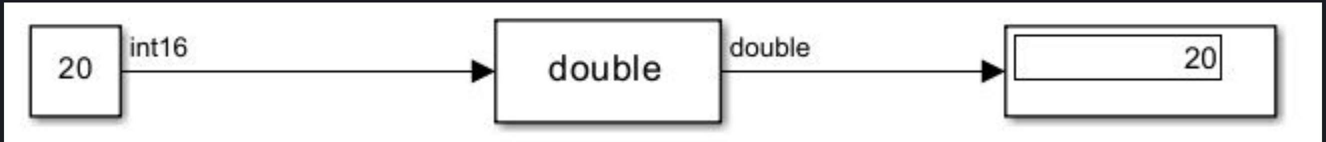


Subsystem, Solvers and code Generation Process

Commonly Used Blocks

- Data Type Conversion
  Integer to Floating-Point Conversion:
  - Input data type: int16
  - Output data type: double

In this example, the Data Type Conversion block is used to convert a 16-bit integer input signal to a double-precision floating-point output signal. This type of conversion is common when performing calculations that require higher precision.

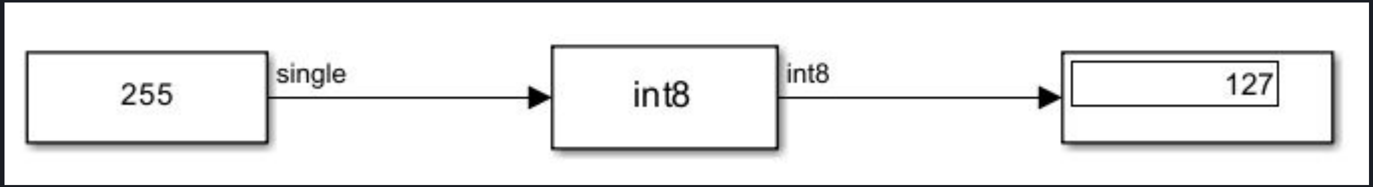


Subsystem, Solvers and code Generation Process

Commonly Used Blocks

- Data Type Conversion
  Rounding and Overflow Handling:
  - Input data type: single
  - Output data type: int8
  - Rounding mode: Round
  - Overflow mode: Saturate

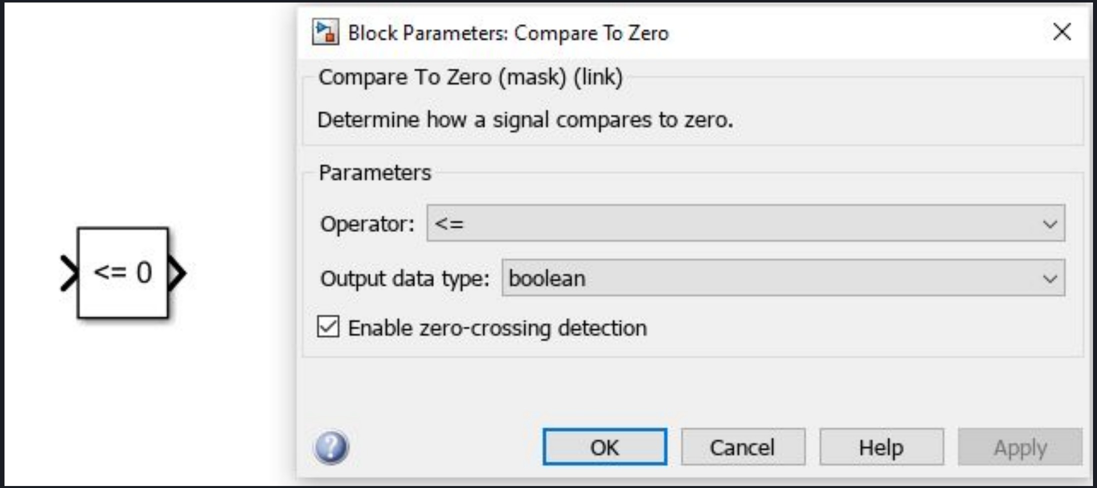
This example converts a single-precision floating-point input signal to an 8-bit integer output signal with rounding and overflow handling. The rounding mode is set to "Round," and overflow handling is set to "Saturate," which means that values exceeding the range of the output data type will be saturated to the minimum or maximum representable value.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

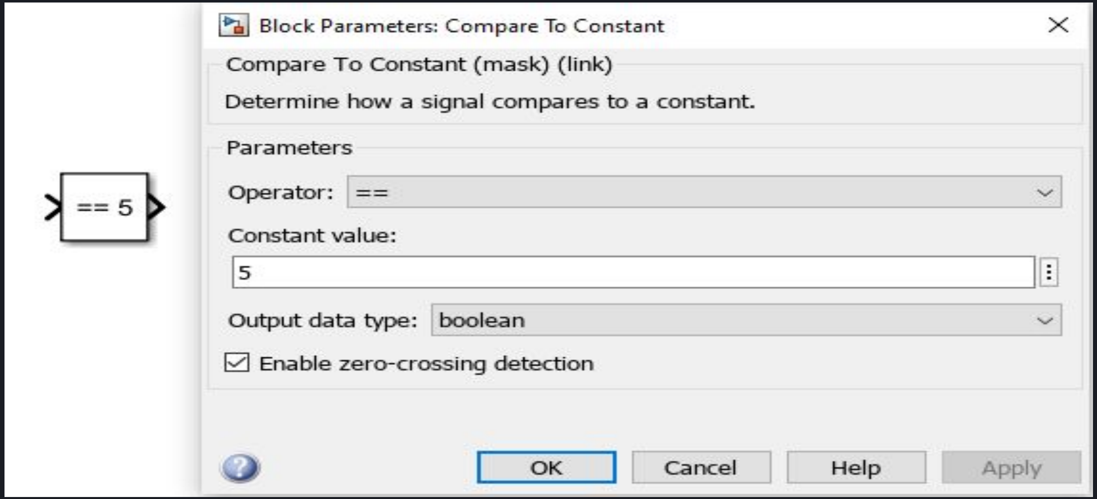
- Compare To Zero
  In Simulink, the "Compare to Zero" block is similar to the "Compare to Constant" block, but it specifically compares an input signal to zero. It outputs a logical (Boolean) value indicating whether the input signal is greater than, less than, equal to, or not equal to zero. This block is often used for conditional checks and logical operations where the comparison is with zero.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

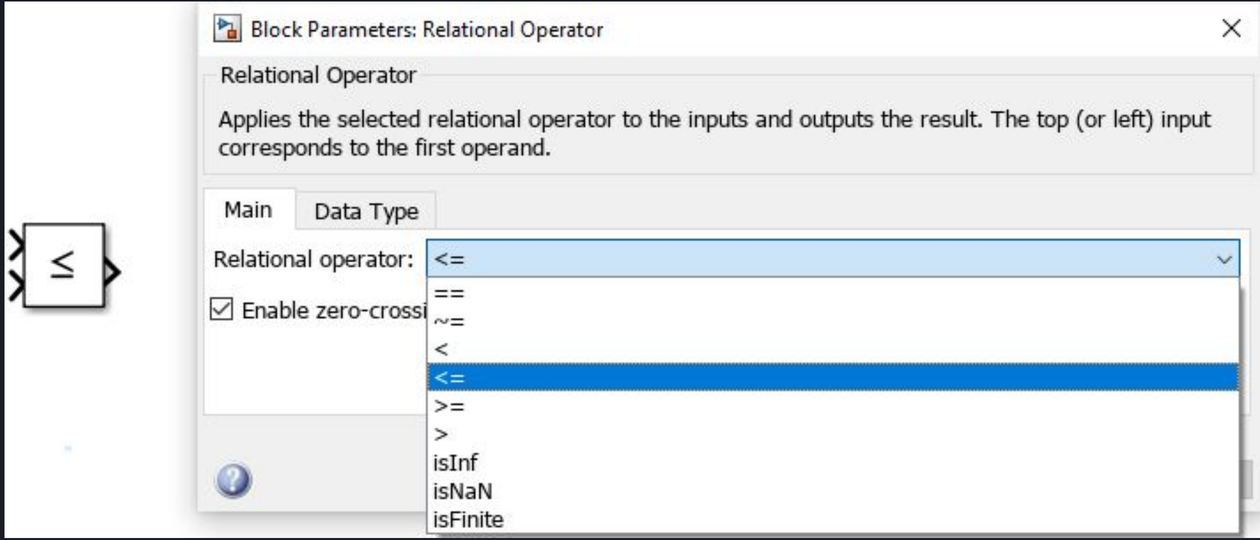
- Compare To Constant
  In Simulink, the "Compare to Constant" block is used to compare an input signal to a specified constant value. It outputs a logical (Boolean) value indicating whether the input signal is greater than, less than, equal to, or not equal to the specified constant. This block is commonly used in control systems, logical decision-making, and conditional operations.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

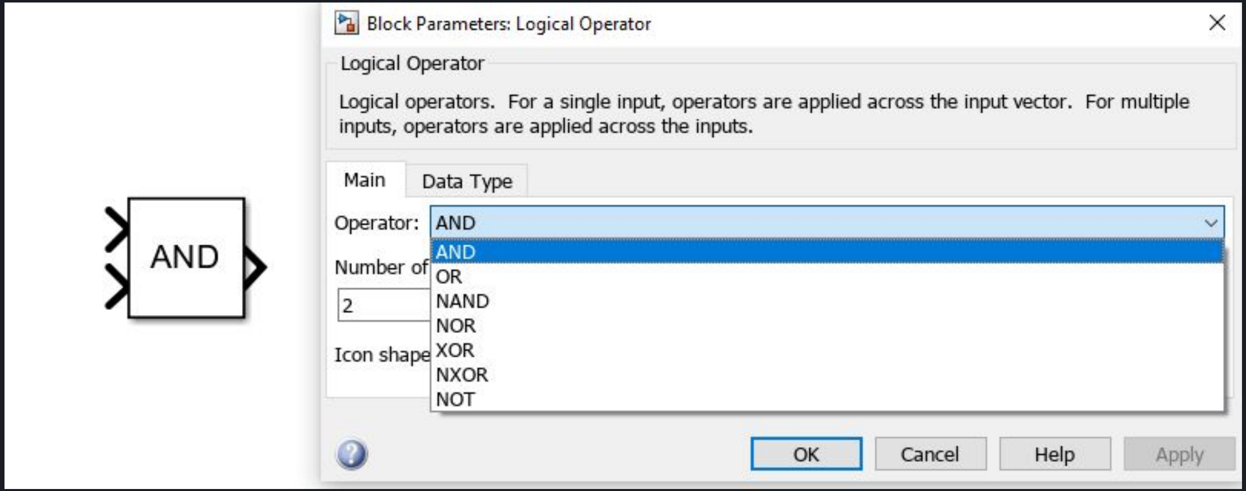
- Relational Operator block
  In Simulink, relational operators are used to perform comparisons between signals or operands. These operators evaluate conditions and produce logical (Boolean) outcomes, such as true or false. Common relation operators include



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

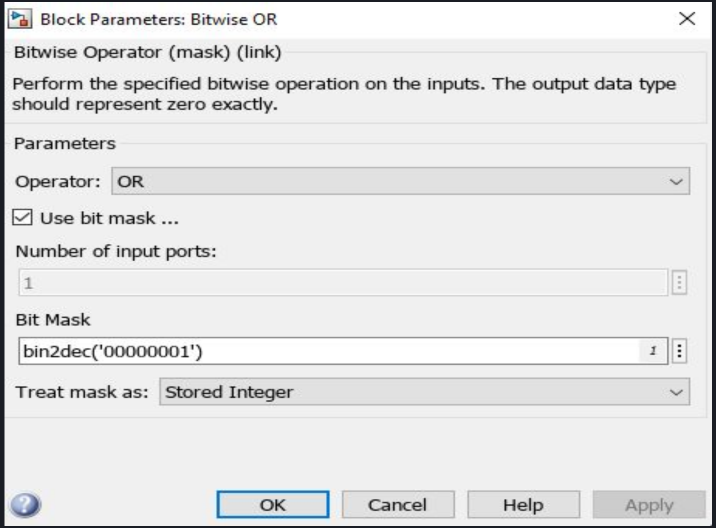
- Logical Operator block
  In Simulink, logical operators are used to perform logical operations on signals. These operators evaluate conditions and produce logical (Boolean) outcomes, such as true or false. Common logical operators include AND, OR, NOT, XOR, and others.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

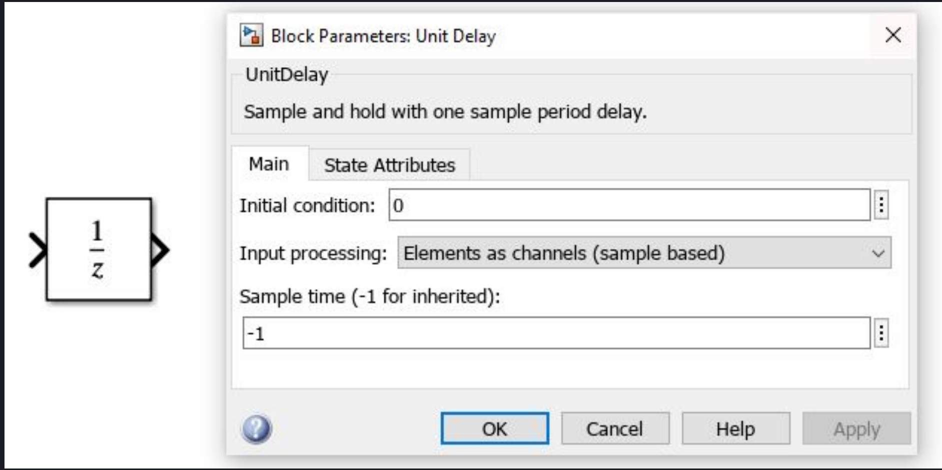
- Bitwise Operator block
  In Simulink, bitwise operators are used to perform bitwise operations on integer signals. Unlike logical operators that operate on Boolean (true/false) values, bitwise operators work at the level of individual bits within integer signals. Common bitwise operators include AND, OR, XOR, NOT, and bit-shifting operators.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

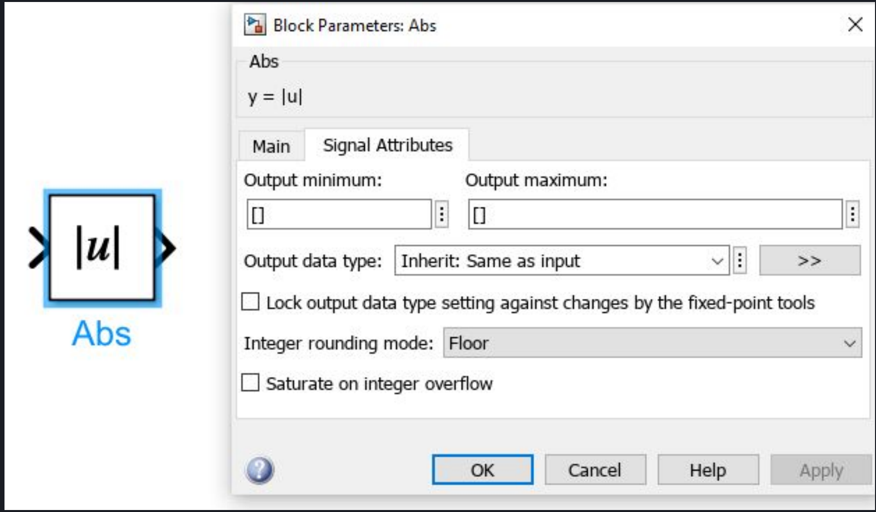
- Unit Delay
  In Simulink, the "Unit Delay" block represents a one-sample delay in a discrete-time system. It is commonly used to model systems where the output at the current time step depends on the input at the previous time step. The unit delay block helps capture the dynamic behavior of systems and is fundamental for building discrete-time models.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

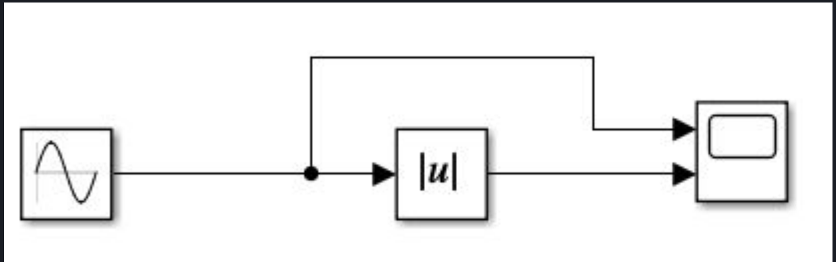
- Abs block
  In Simulink, the "Abs" block is used to compute the absolute value of an input signal. The absolute value of a number is its distance from zero on the number line, regardless of the direction. The "Abs" block outputs the magnitude of the input signal.

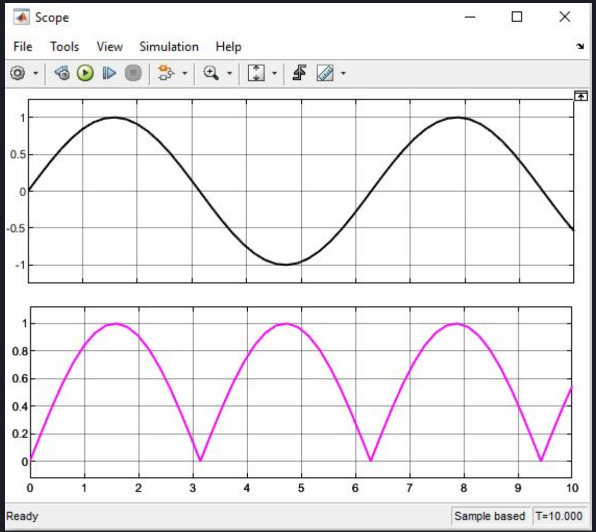


Subsystem, Solvers and code Generation Process

Commonly Used Blocks

- Abs block

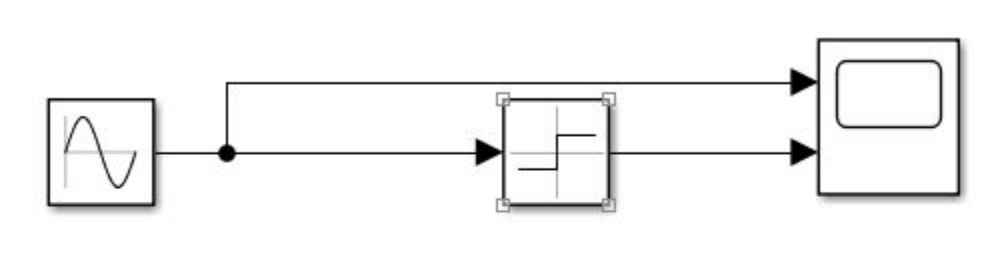


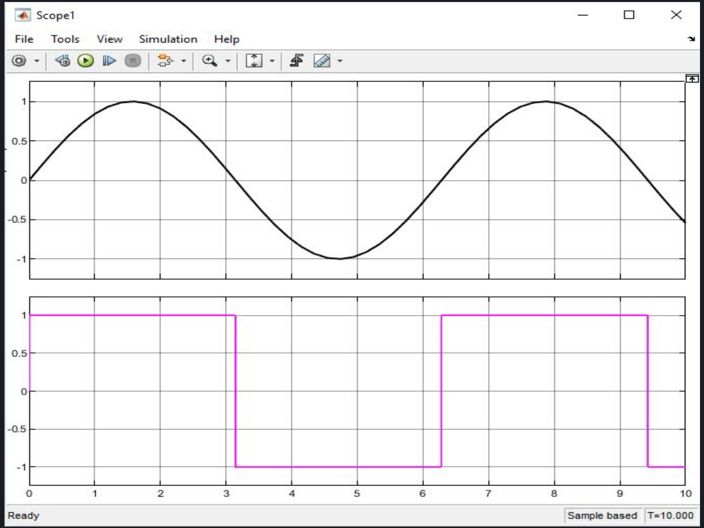


Subsystem, Solvers and code Generation Process

Commonly Used Blocks

- sign block
  In Simulink, the "Sign" block is used to determine the sign of an input signal. It outputs a signal that represents the sign of the input, which can be either positive, negative, or zero. The output typically takes the values +1 for positive input, -1 for negative input, and 0 for zero input.

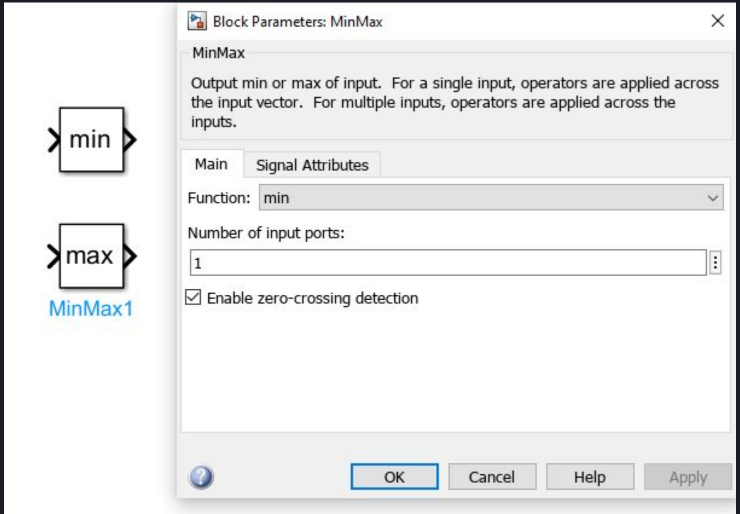




Subsystem, Solvers and code Generation Process

Commonly Used Blocks

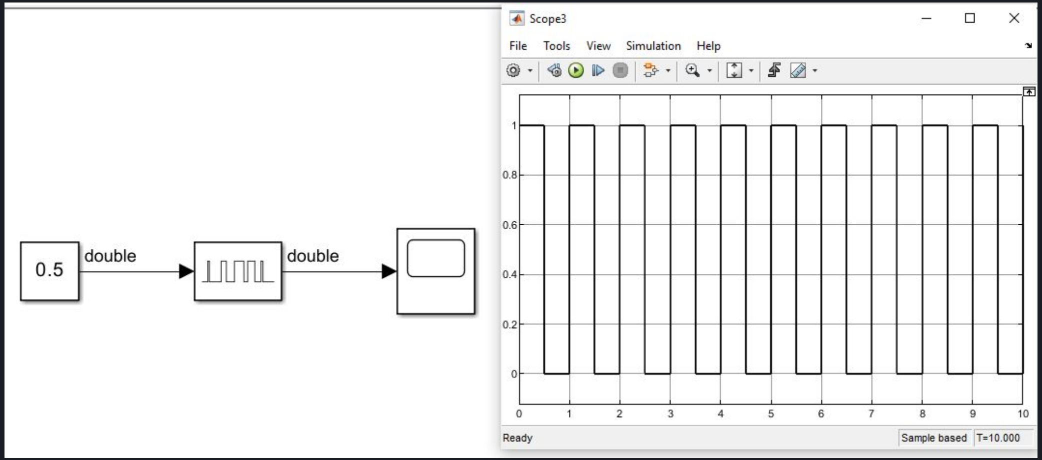
- MinMax block
  In Simulink, the "MinMax" block is used to compute the minimum and maximum values of two input signals. This block helps determine the minimum and maximum values between two signals at each simulation time step.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

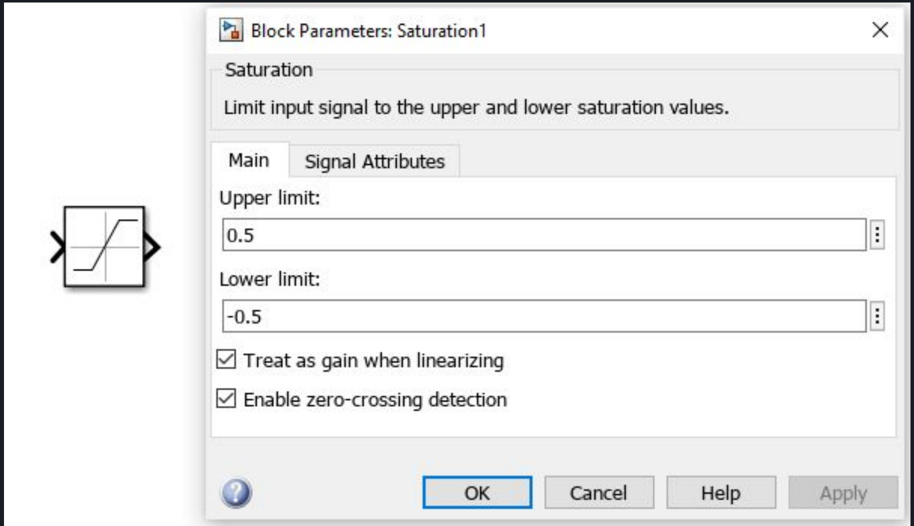
- PWM block
  In Simulink, the "PWM (Pulse Width Modulation)" block is used to generate a pulse-width modulated signal. Pulse Width Modulation is a technique commonly employed in control systems and power electronics to control the average value of a signal by varying the width of its pulses. PWM is often used in applications such as motor control, power inverters, and communication systems.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

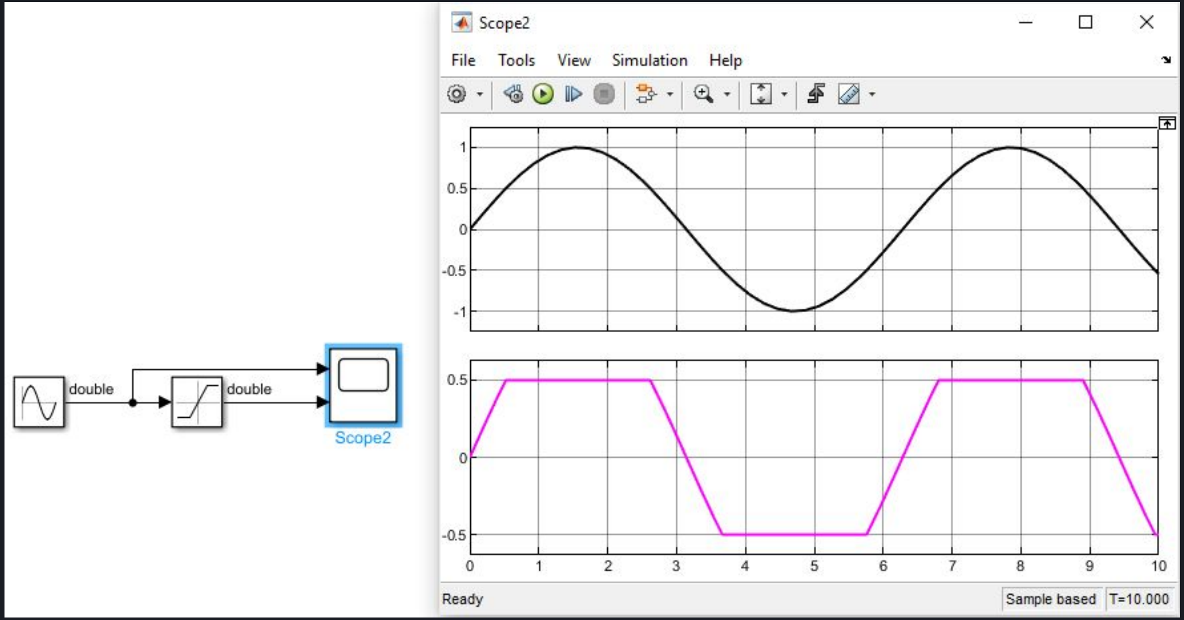
- Saturation block
  In Simulink, the "Saturation" block is used to limit the range of an input signal within a specified upper and lower bound. This block is particularly useful for preventing signals from exceeding certain limits, providing a way to model physical constraints or constraints imposed by a system.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

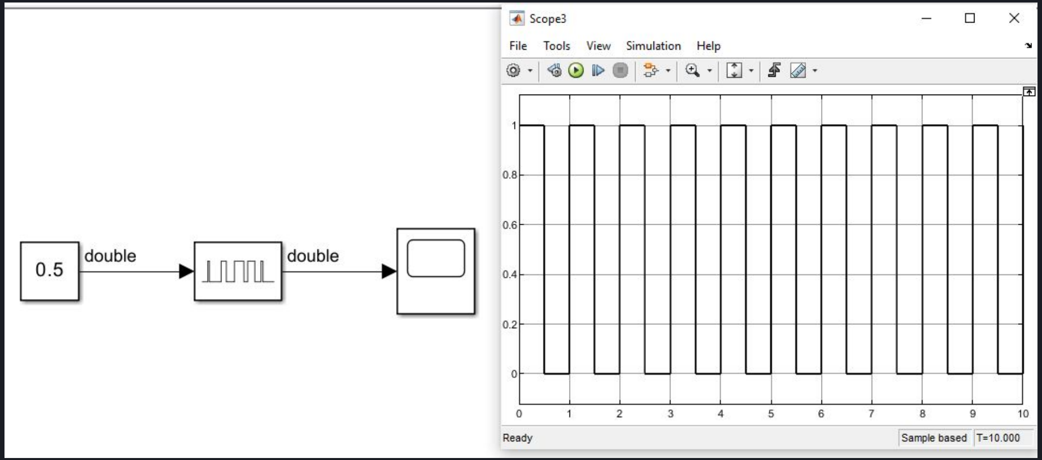
- Saturation block



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

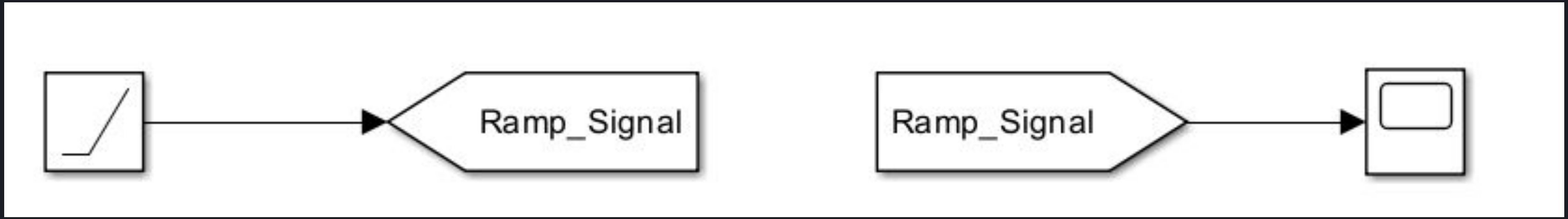
- PWM block
  In Simulink, the "PWM (Pulse Width Modulation)" block is used to generate a pulse-width modulated signal. Pulse Width Modulation is a technique commonly employed in control systems and power electronics to control the average value of a signal by varying the width of its pulses. PWM is often used in applications such as motor control, power inverters, and communication systems.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

- Goto and from block
  In Simulink, the "Goto" and "From" blocks are used to create signal connections between different parts of a model. These blocks help organize and simplify the layout of a Simulink model by allowing you to create explicit signal connections without drawing lines across the entire diagram.



Subsystem, Solvers and code Generation Process

Commonly Used Blocks

- Saturation block

Now, Try to implement Saturation Block
With another methods

Subsystem, Solvers and code Generation Process

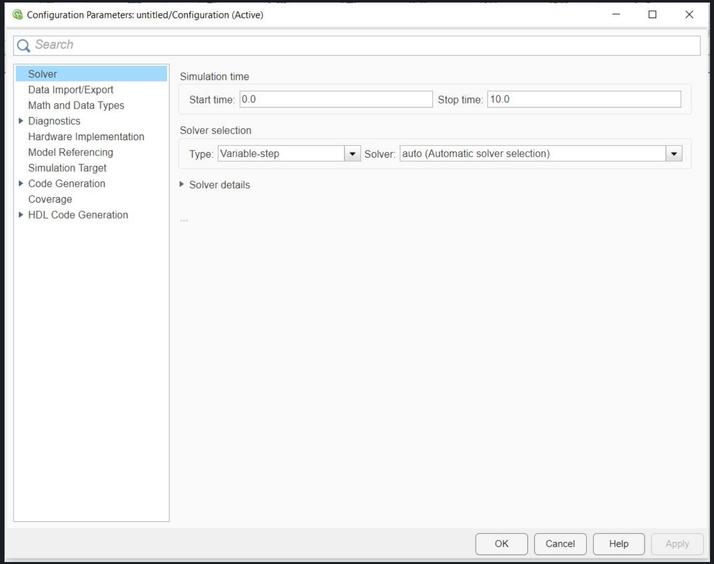
Quiz 3: Click Here To Start

Subsystem, Solvers and code Generation Process

Solvers (How Simulink Simulate Models?)

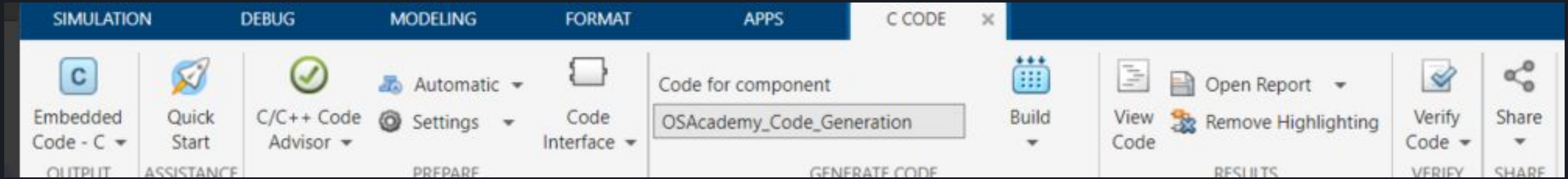
In Simulink, solvers are algorithms used to numerically solve the ordinary differential equations (ODEs) that describe the behavior of dynamic systems. These solvers determine how the simulation advances in time and how accurately it captures the system's behavior. Simulink provides various solver options to address different simulation requirements. Here's an explanation of solvers in Simulink:

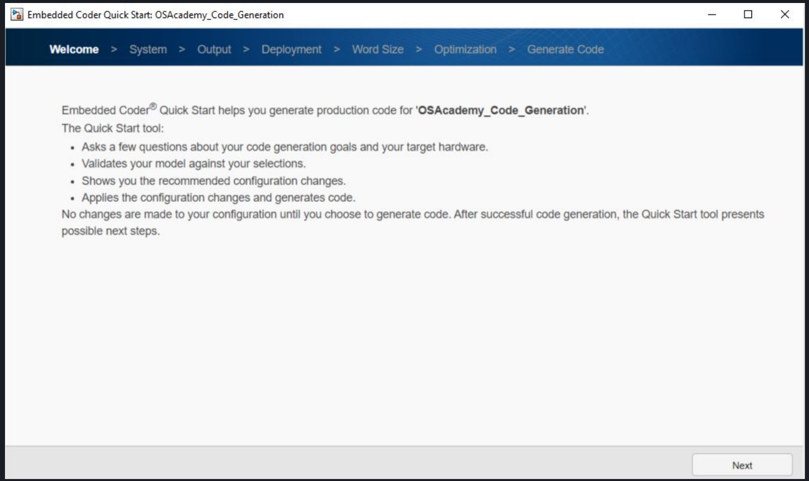
1. **Fixed-Step Solvers**: These solvers use a constant time step during the simulation. They are suitable for systems with a known and constant step size. Common fixed-step solvers include the Fixed Step and Variable Step solvers.
2. **Variable-Step Solvers**: Variable-step solvers adjust the time step dynamically based on the system's behavior. They are more efficient for systems with varying dynamics or when a high level of accuracy is needed. Common variable-step solvers include ODE1, ODE2, and ODE3.



Subsystem, Solvers and code Generation Process

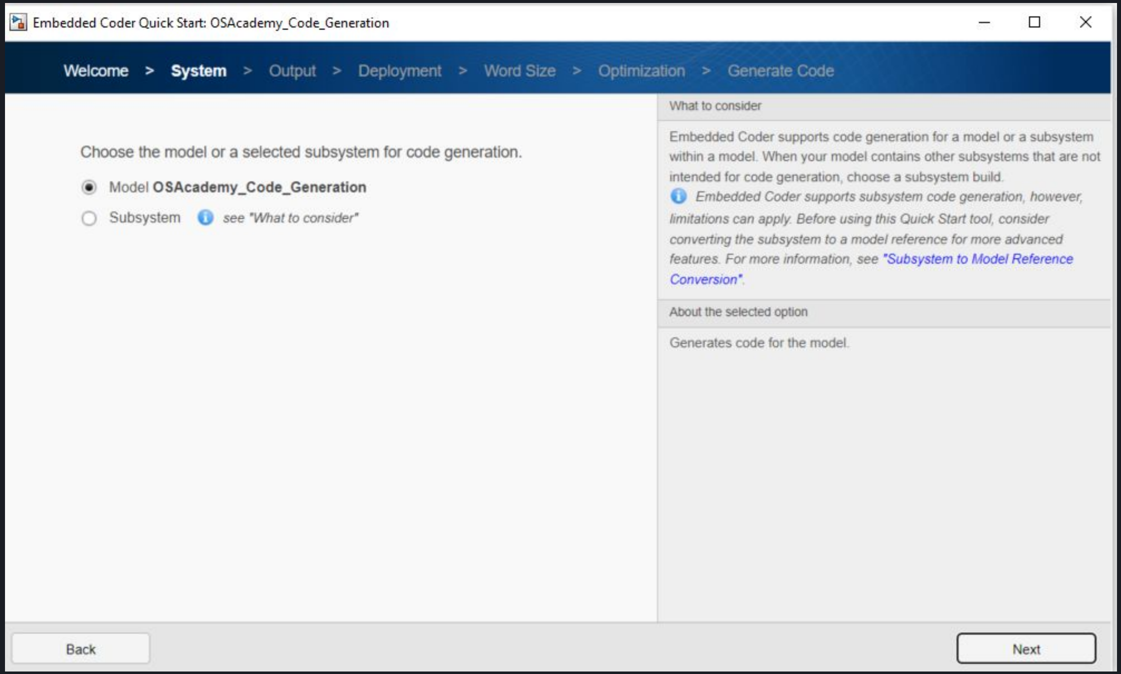
Code Generation Process





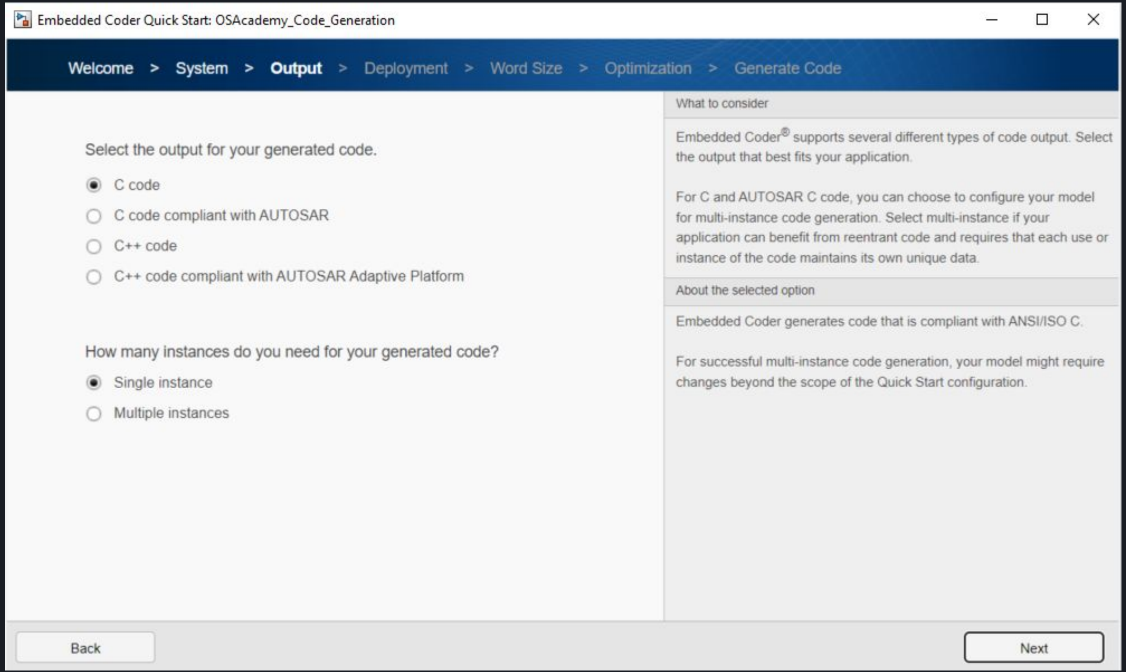
Subsystem, Solvers and code Generation Process

Code Generation Process



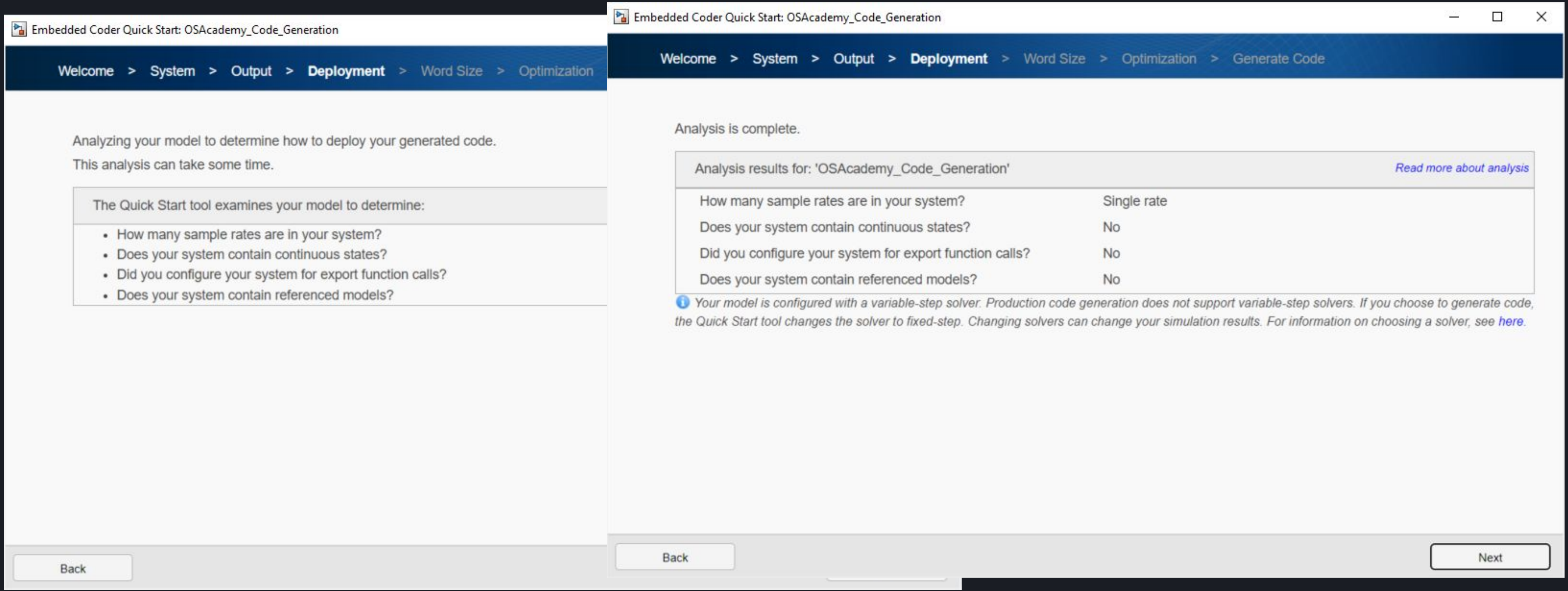
Subsystem, Solvers and code Generation Process

Code Generation Process



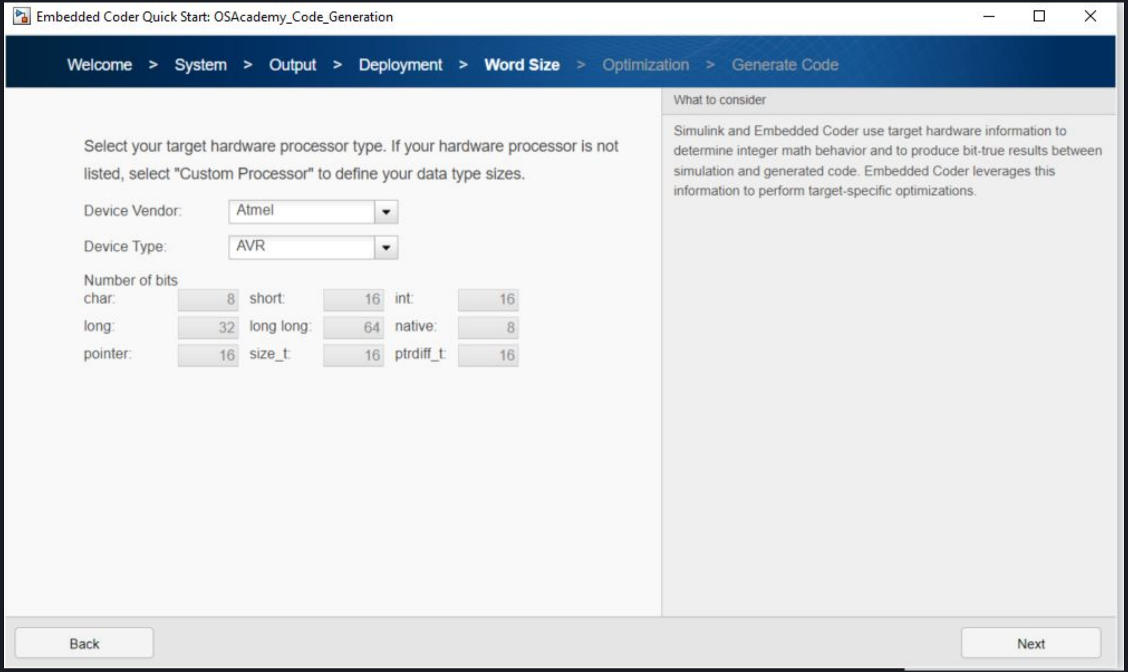
Subsystem, Solvers and code Generation Process

Code Generation Process



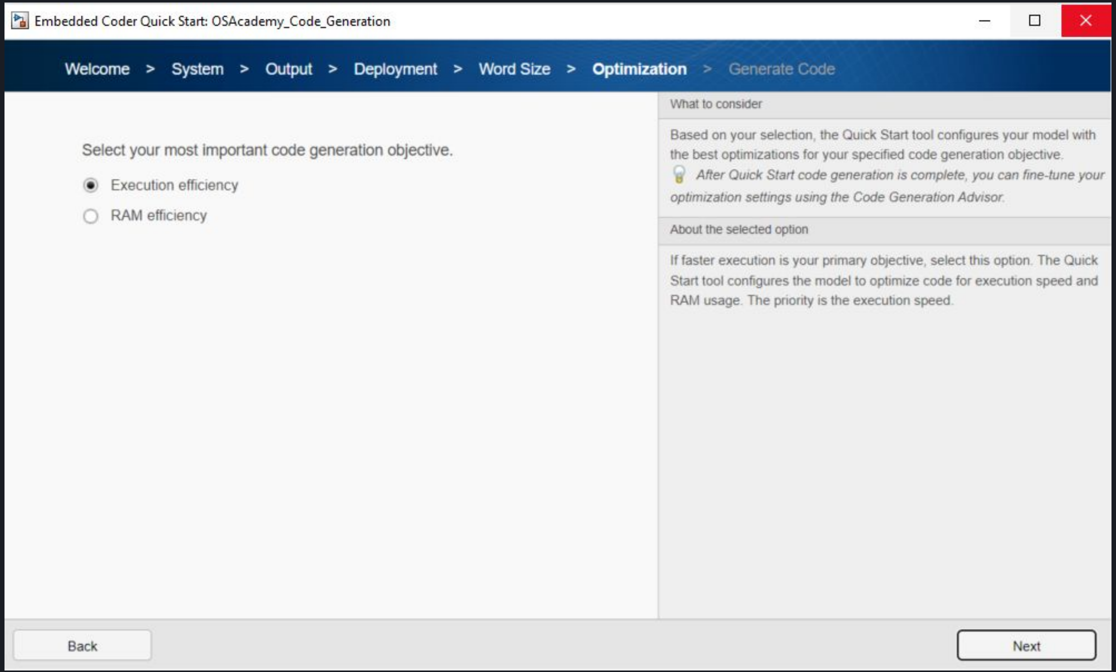
Subsystem, Solvers and code Generation Process

Code Generation Process



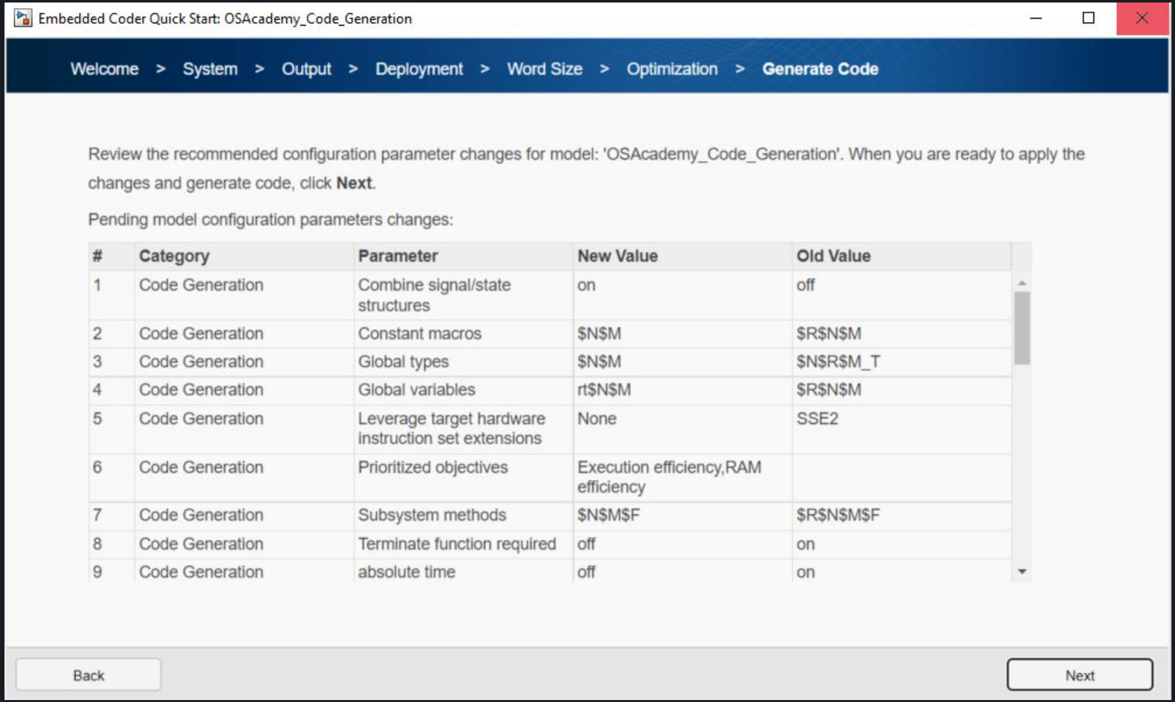
Subsystem, Solvers and code Generation Process

Code Generation Process



Subsystem, Solvers and code Generation Process

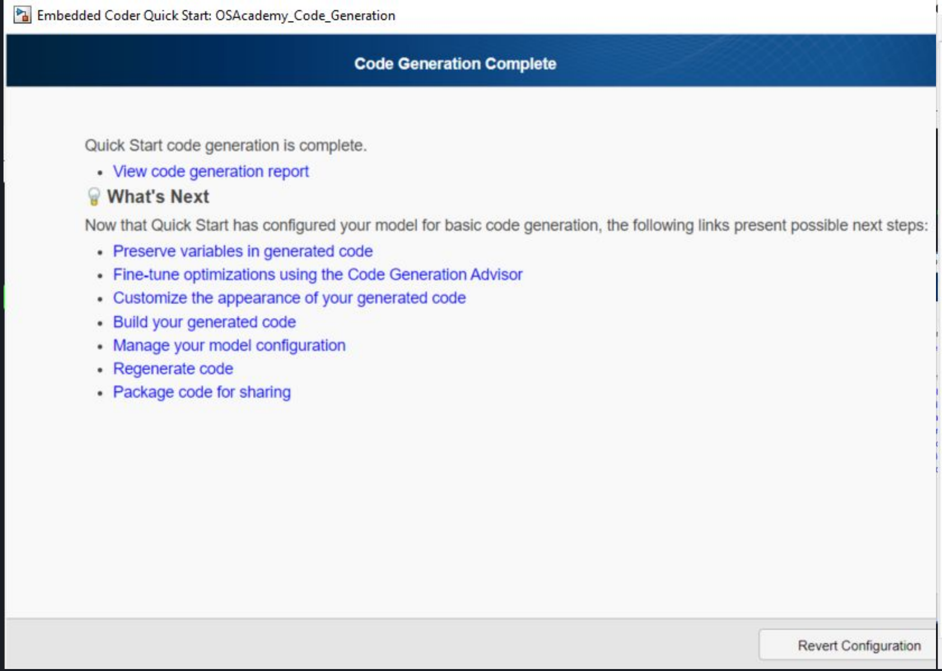
Code Generation Process

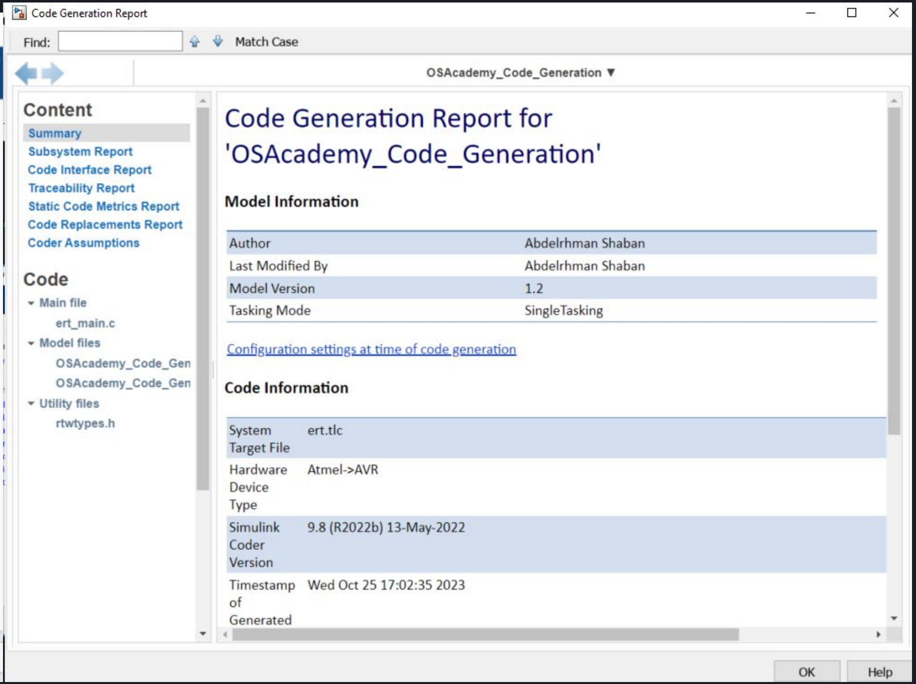


#	Category	Parameter	New Value	Old Value
1	Code Generation	Combine signal/state structures	on	off
2	Code Generation	Constant macros	NM	RN$M
3	Code Generation	Global types	NM	NR$M_T
4	Code Generation	Global variables	rt$N$M	RN$M
5	Code Generation	Leverage target hardware instruction set extensions	None	SSE2
6	Code Generation	Prioritized objectives	Execution efficiency,RAM efficiency	
7	Code Generation	Subsystem methods	NM$F	RNMF
8	Code Generation	Terminate function required	off	on
9	Code Generation	absolute time	off	on

Subsystem, Solvers and code Generation Process

Code Generation Process





Subsystem, Solvers and code Generation Process

Code Generation Process

Using Embedded Coder in Simulink streamlines the process of generating code for embedded systems, making it easier to transition from simulation to real-world deployment. This is a critical step in the development of embedded control systems and helps ensure the reliability and performance of your systems in real-time applications.

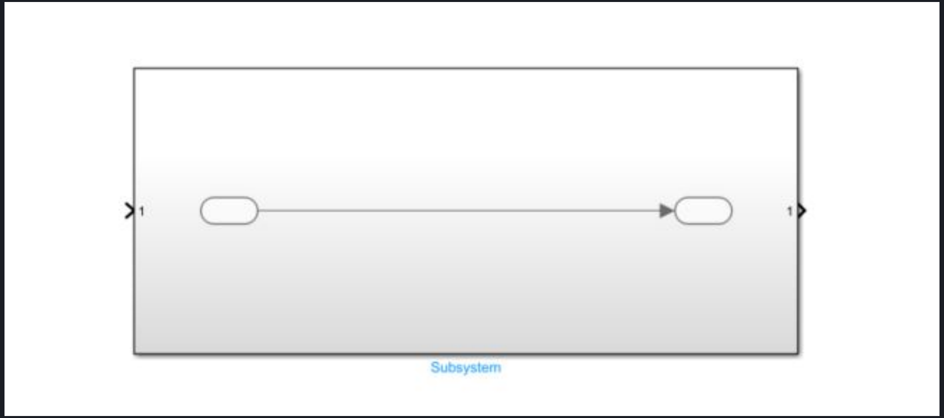
Subsystem, Solvers and code Generation Process

Quiz 4: Click Here To Start

Subsystem, Solvers and code Generation Process

Subsystems

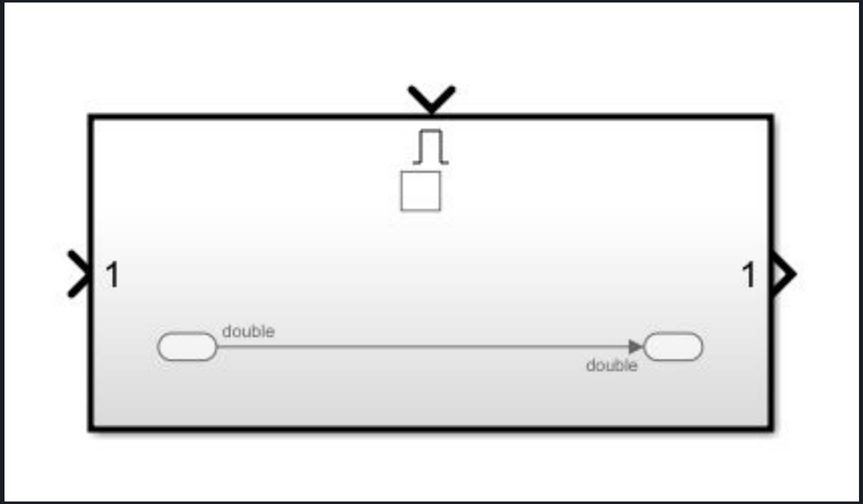
- Subsystem



Subsystem, Solvers and code Generation Process

Subsystems

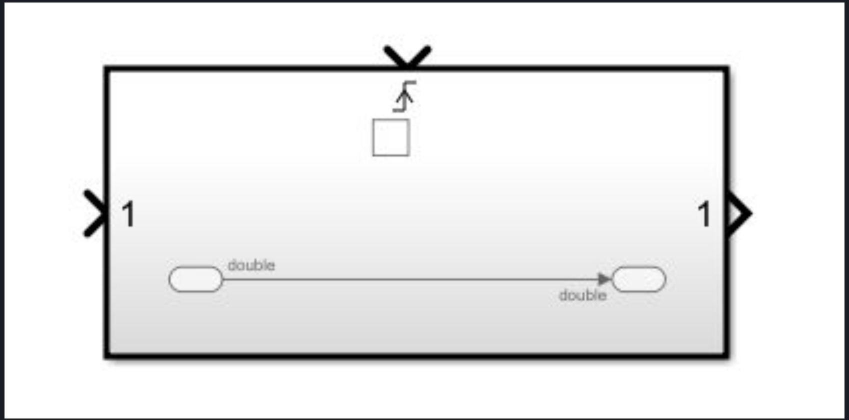
- Enable Subsystem



Subsystem, Solvers and code Generation Process

Subsystems

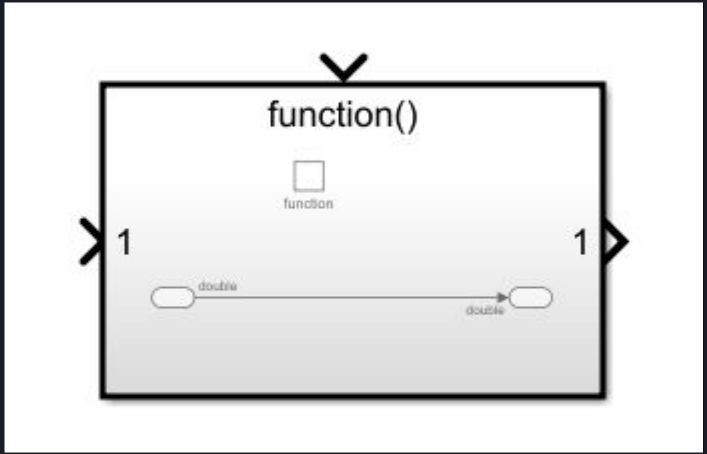
- Triggered Subsystem



Subsystem, Solvers and code Generation Process

Subsystems

- Function Call Subsystem



Subsystem, Solvers and code Generation Process

[Lab 2: Click Here To Start]