DL PEL-HIL Getting Started

Host-Target Communication - Vector

DL R&D

2025-08-07

Description

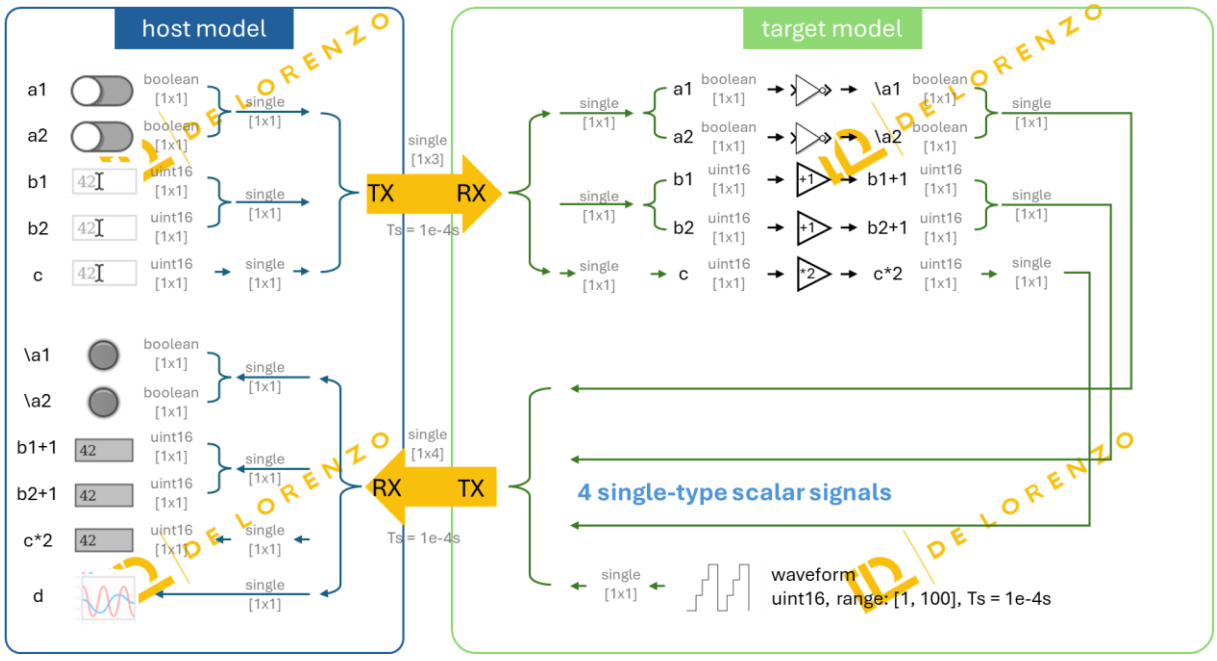
In this example, we will build a host-target (master-slave) communication system to exchange vectors—multiple scalar signals grouped as arrays (e.g., [1×3])—as illustrated in the diagram below.

The host model running in the master (PC) will package and transmit data to the target model running in the slave device (DL RCPCORE), including

- 2 Boolean values
- 3 uint16 values

The DL RCPCORE (slave) will unpack the received data, perform simple processing, then package and return the processed data along with a discrete sawtooth waveform back to the master.

The master will then receive and unpack the data, display the results, and verify the correctness of the communication.



Learning Objectives

- how to set up a host-target communication system that supports multiple data types and multiple signals
- understand and differentiate the host model and the target model through hands-on practice
- data packing and unpacking
- understand the significance of time synchronization in communication
- dashboard development

Prior Knowledge

1. **Data Types**

   In DL PEL-HIL, we recommend using three data types: **boolean**, **uint16**, and **single**.

Data Type	Size	Range	Description
boolean	1 bit	1/0	Represents logical states such as on/off or yes/no. Used for flags and condition checks.
uint16	16-bit	[0, 65535]	Efficient for counters, IDs, and hardware register values. Saves memory for positive integers.
single	32-bit	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$	Used for numeric data with decimals. Suitable for sensor values and control algorithms.

   Notes:

   - *Some DL built-in blocks specify a data type and therefore require matching input/output signals for consistency.*
   - *Use single type for precision; use boolean and uint16 for speed and memory efficiency.*

2. Signal Dimensions

In the context of communication, a signal can take various forms:

- a single data element,
- a combination of multiple single-element signals, or
- a set of signals that each include historical (past) data.

As a starting point, let's first explore the different dimensions that signals can have.

Size	Example	Terminology	Note
[1x1]	5	Scalar	A single value
[2x1]	$\begin{bmatrix} 1.1 \\ 2.1 \end{bmatrix}$	Column vector	A vector with 2 rows and 1 column
[1x3]	$\begin{bmatrix} 1.1 & 1.2 & 1.3 \end{bmatrix}$	Row vector	A vector with 1 row and 3 columns
[2x3]	$\begin{bmatrix} 1.1 & 1.2 & 1.3 \\ 2.1 & 2.2 & 2.3 \end{bmatrix}$	Matrix	A matrix with 2 rows and 3 columns

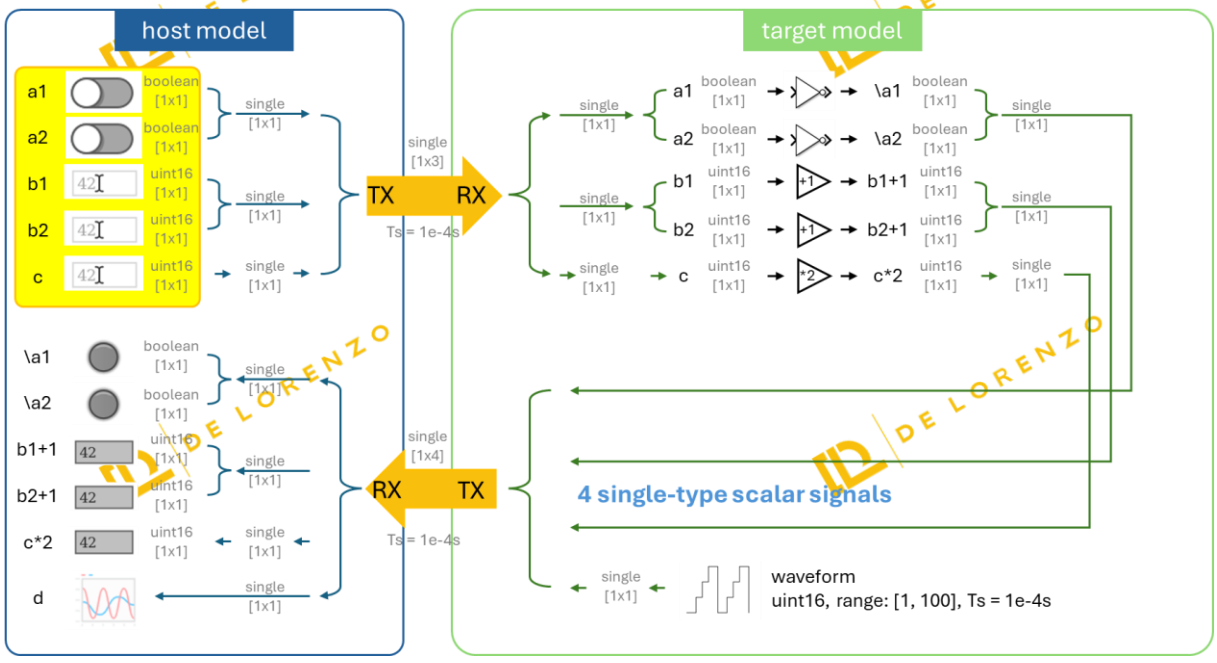
Step 0. Model parameter settings

Click on ' 📌 Click here to edit model parameters' to edit the parameters.

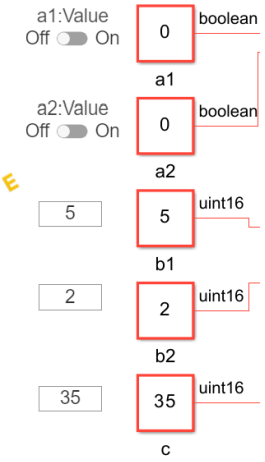
Set the model base sample time **Ts** (which here corresponds to the transmission interval) and the **COM port** recognized by the hardware in Device Manager.

```
%% Set PWM Switching frequency
PWM_frequency    = 4e3;          %Hz          // Converter
T_pwm            = 1/PWM_frequency;  %s  // PWM switching
deadtime         = 5e-6;         %s //rising edge delay fo

%% Model Parameters
Ts               = 1e-4;         %s      // Sample time for
dataType         = 'single';             % Floating point

%% Hareware parameters
% Set controller parameters
target           = SetControllerDetails('DL RCPCORE', PWM_
target.comport   = 'COM6';       % Update the appropriate
converter        = SetConverterParameters('DL 2106T06');
```

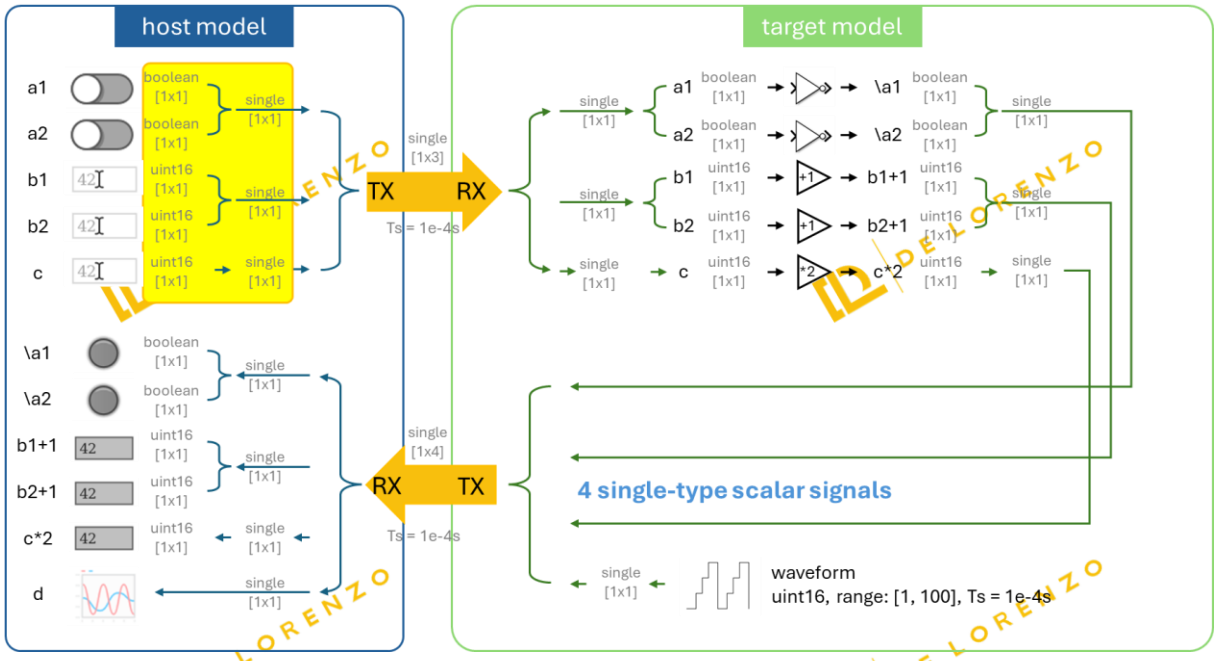
Step 1. Create controlled variables



First, we add the corresponding control variables to the host model according to the requirements (sample time, data type, and dimension).

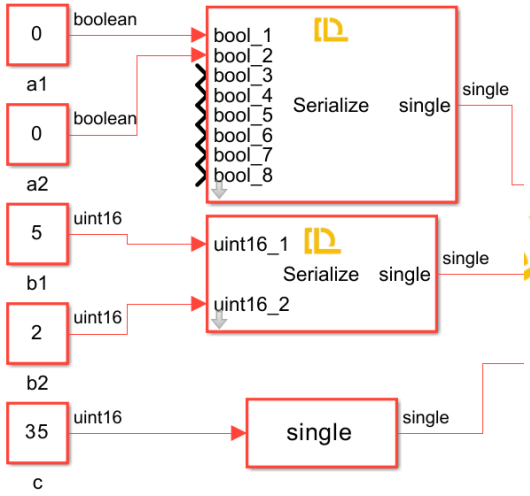


Step 2. Serialize multiple same-type signals into a single signal

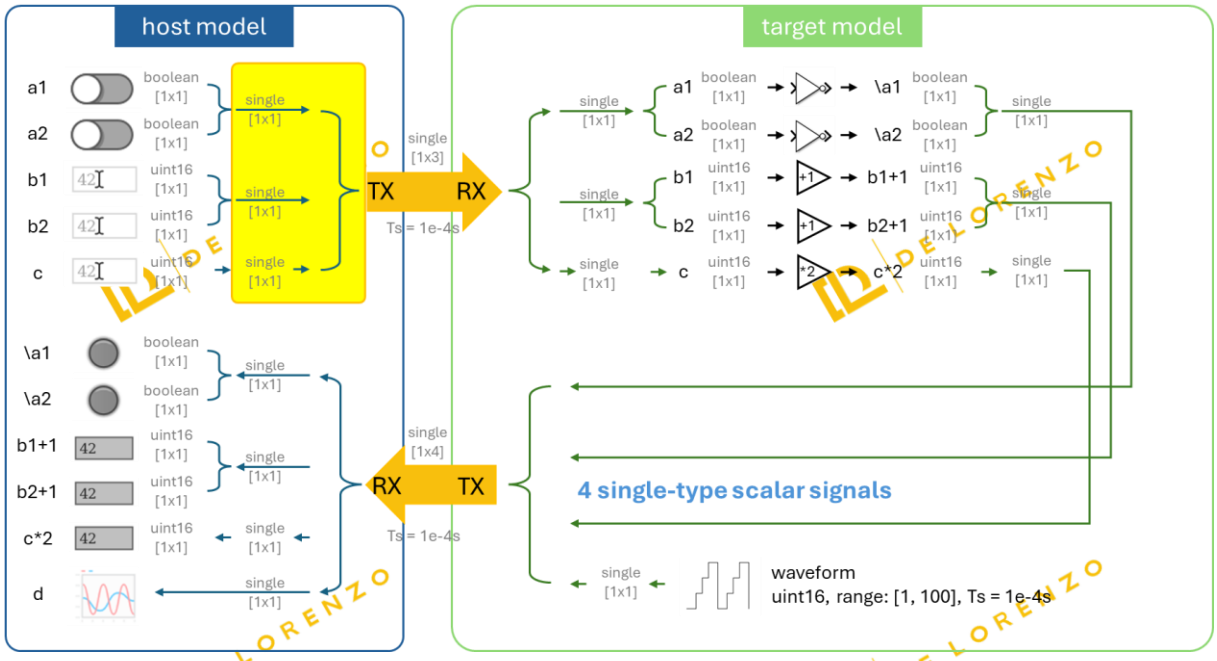


Use the **'Serialize Boolean into Single'** block and the **'Serialize Uint16 into Single'** block to combine multiple **Boolean** and **uint16** signals into **single**-precision values.

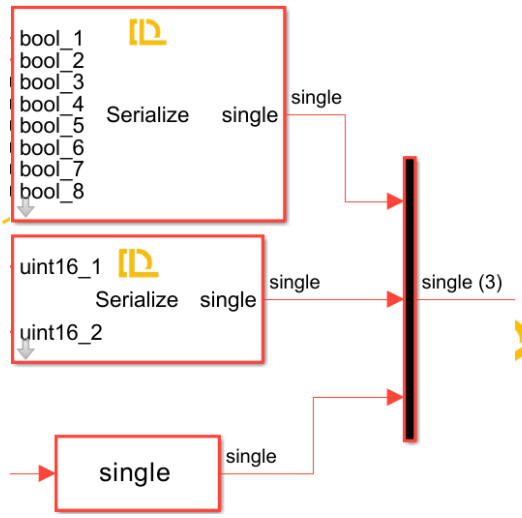
Unify all data types to single precision.



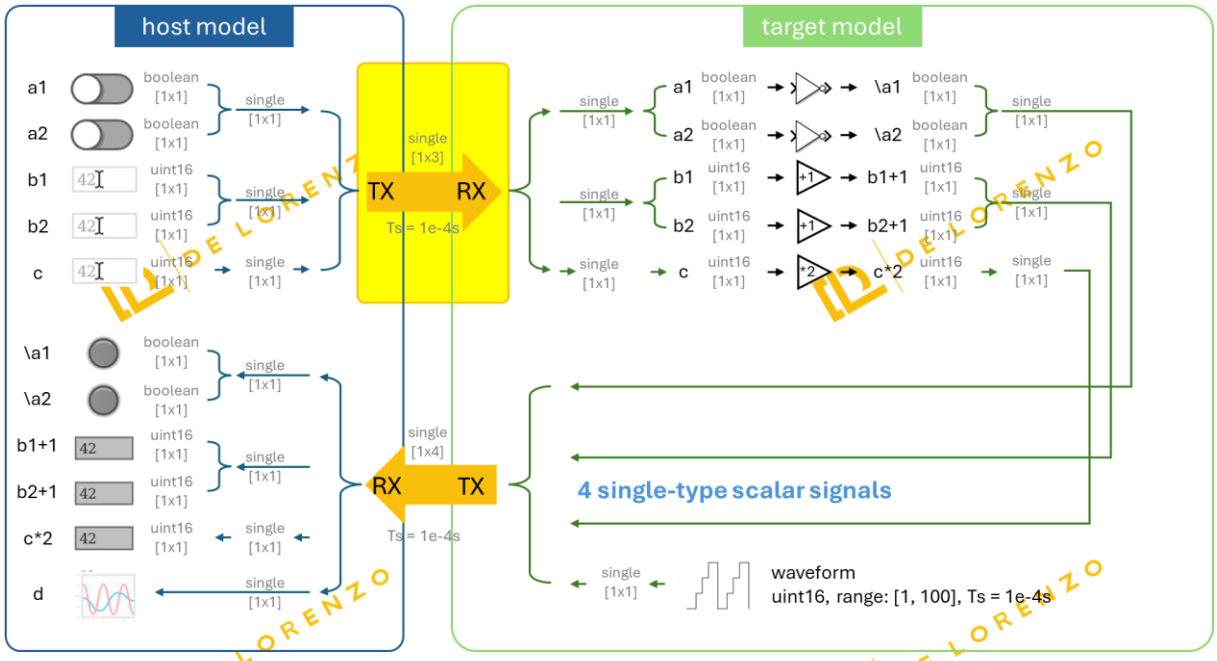
Step 3. Multiplex Same-Type Signals



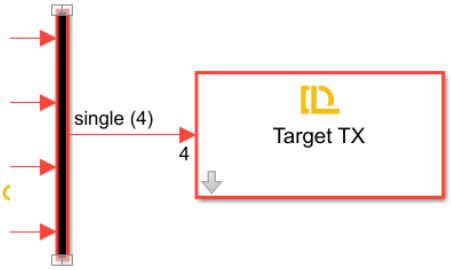
Use a **'Mux'** block to multiplex the three single-type signals into a [1×3] vector in preparation for transmission (TX).



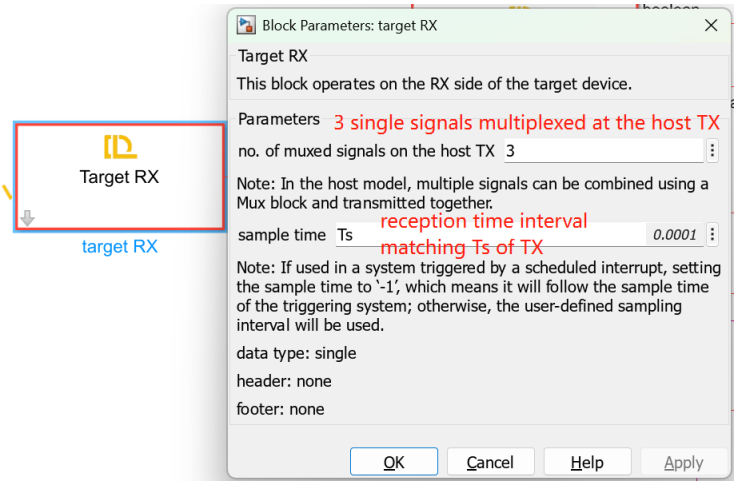
Step 4. Host-to-target transmission



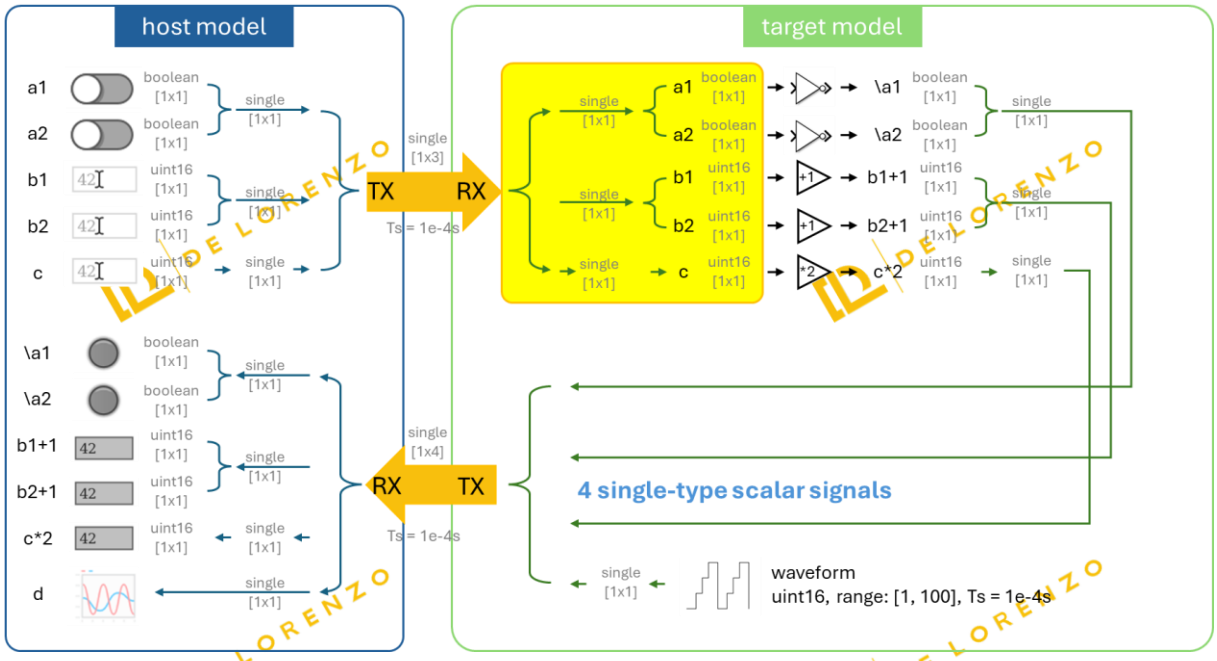
In the **host model**, place the **'Host TX'** block. The transmission rate will follow the sampling interval of the controlled variables.



In the **target model**, place the **'Target RX'** block. Since a vector of [1x3] will be transmitted from the host model, the number of columns (i.e., the number of single-type signals) at the RX end on the target is set to 3. The reception interval is identical to the transmission interval.



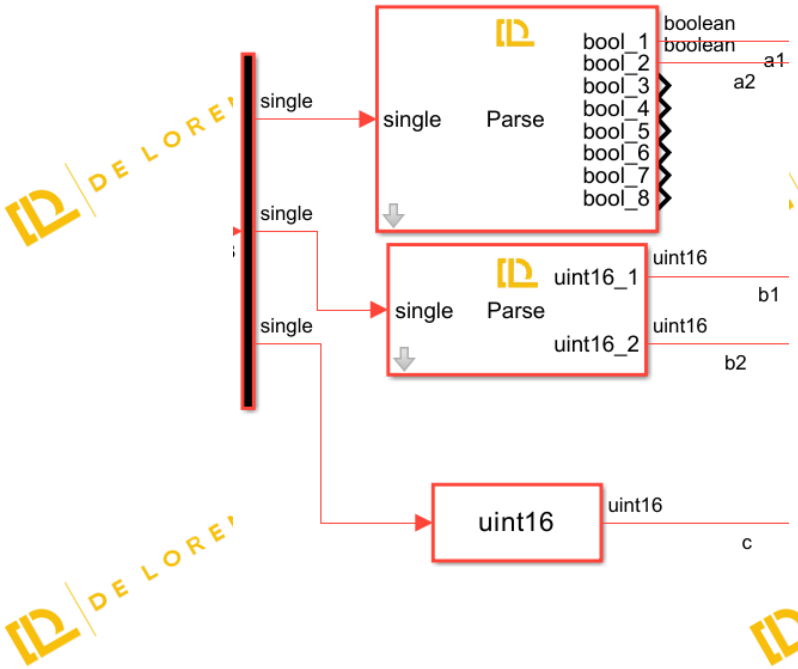
Step 5. Parse to recover the controlled variables



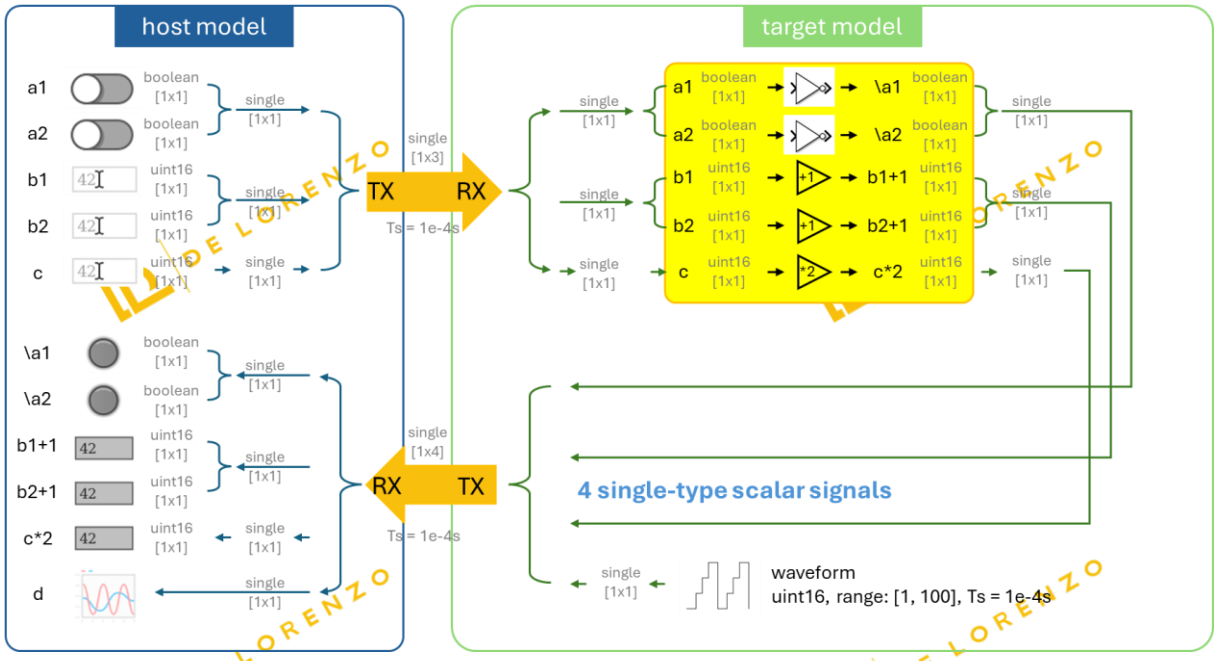
Use the **'Demux'** block to split a [1×3] vector into three [1×1] scalar signals.

Use the **'Parse Single into Boolean'** block to convert a **single**-type signal into multiple **Boolean** signals and extract the controlled variables from the corresponding channels.

Use the **'Data Type Conversion'** block to restore the data to its original type.



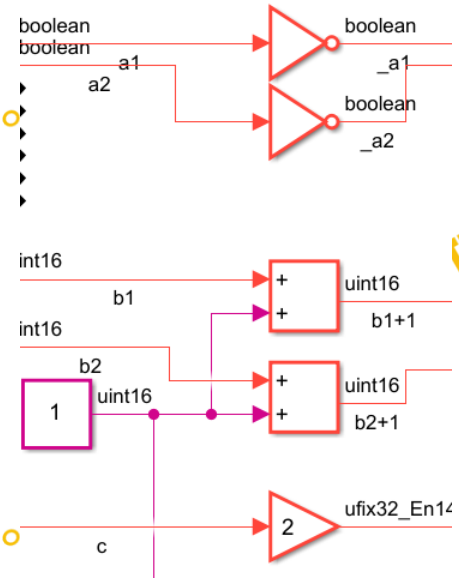
Step 6. Simple Signal Processing



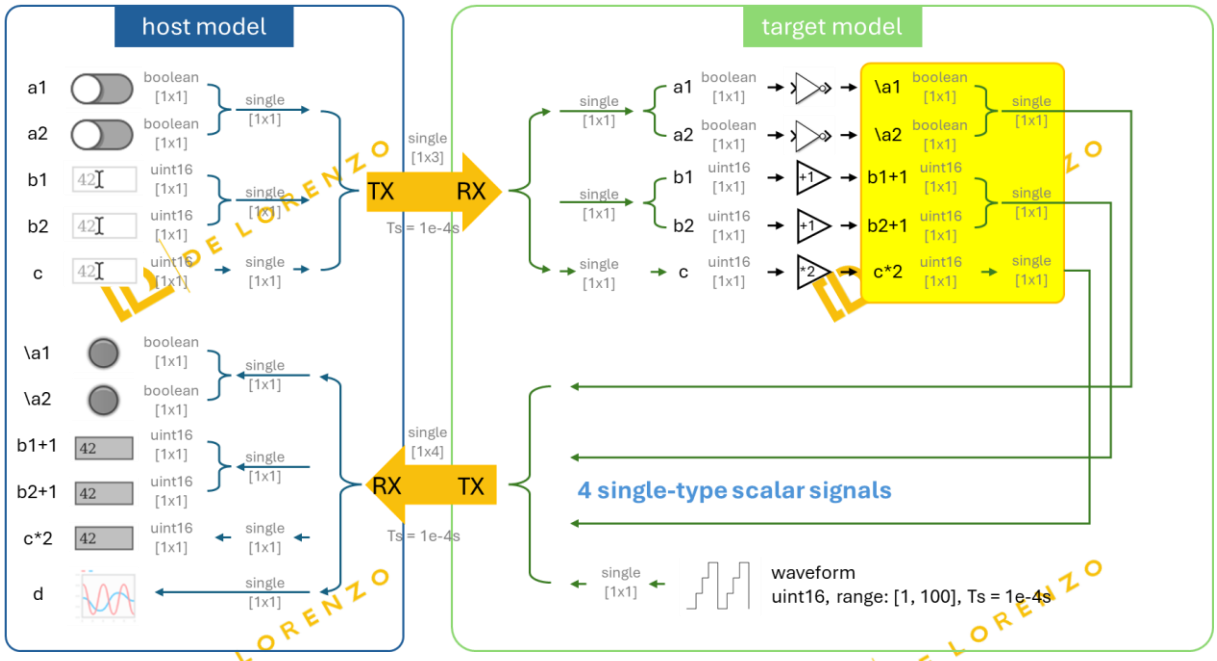
Invert Boolean signals a1 and a2 to obtain _a1 and _a2.

Add 1 to b1 and b2 to get b1+1 and b2+1.

Multiply c by 2 to get c*2.

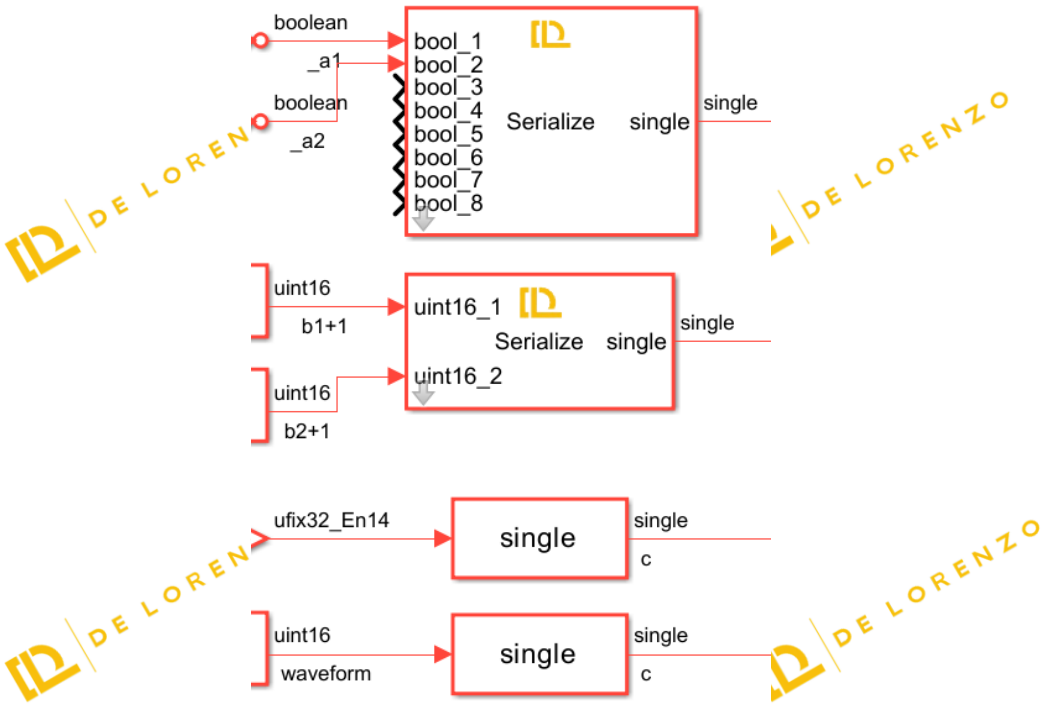


Step 7. Serialize multiple same-type signals into a single signal

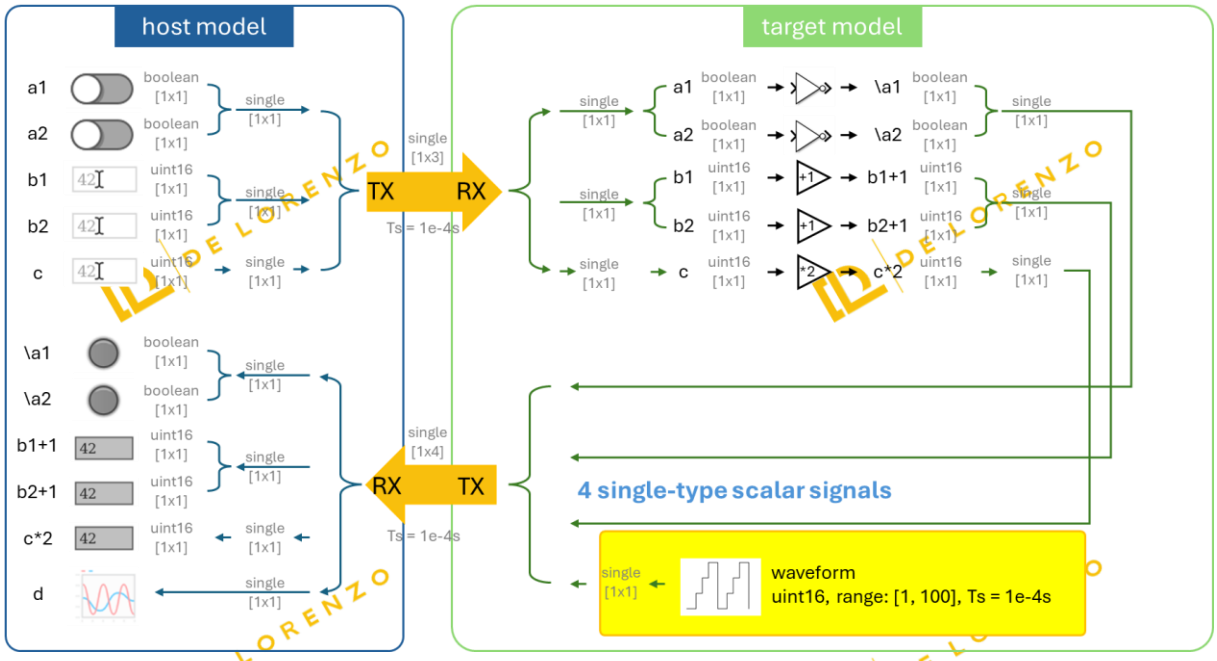


Use the **'Serialize Boolean into Single'** block and the **'Serialize Uint16 into Single'** block to combine multiple **Boolean** and **uint16** signals into **single**-precision values.

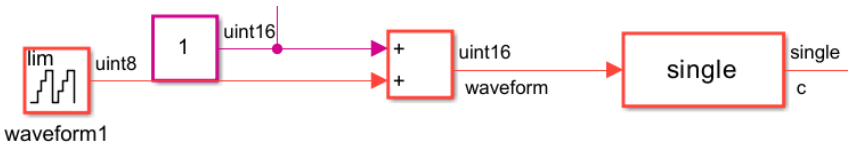
Unify all data types to single precision.



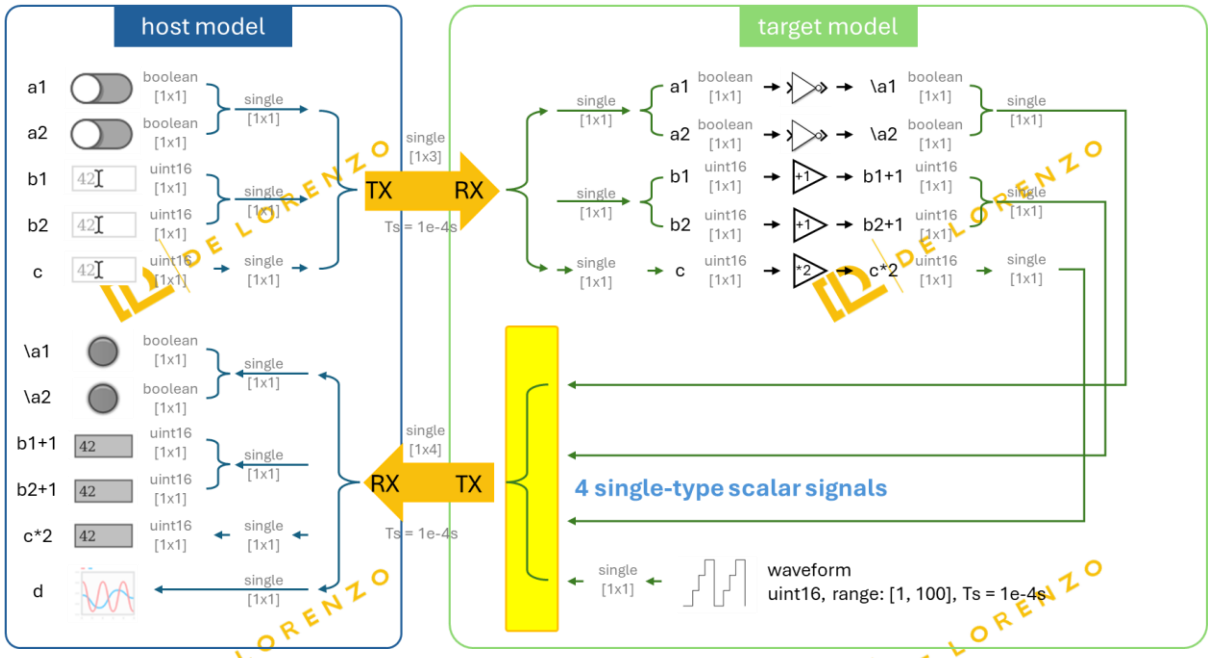
Step 8. Create a sawtooth waveform



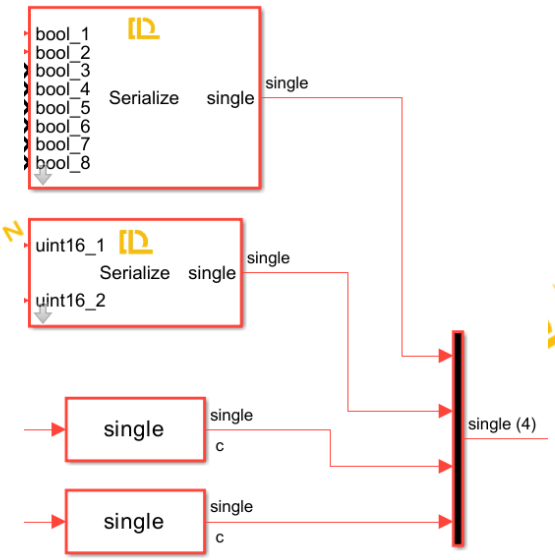
Use the **'Counter Limited'** block to create a discrete sawtooth waveform ranging from 1 to 100 with sample time of 1e-4s. Convert the final output into single type to be consistent with the other serialized signals.



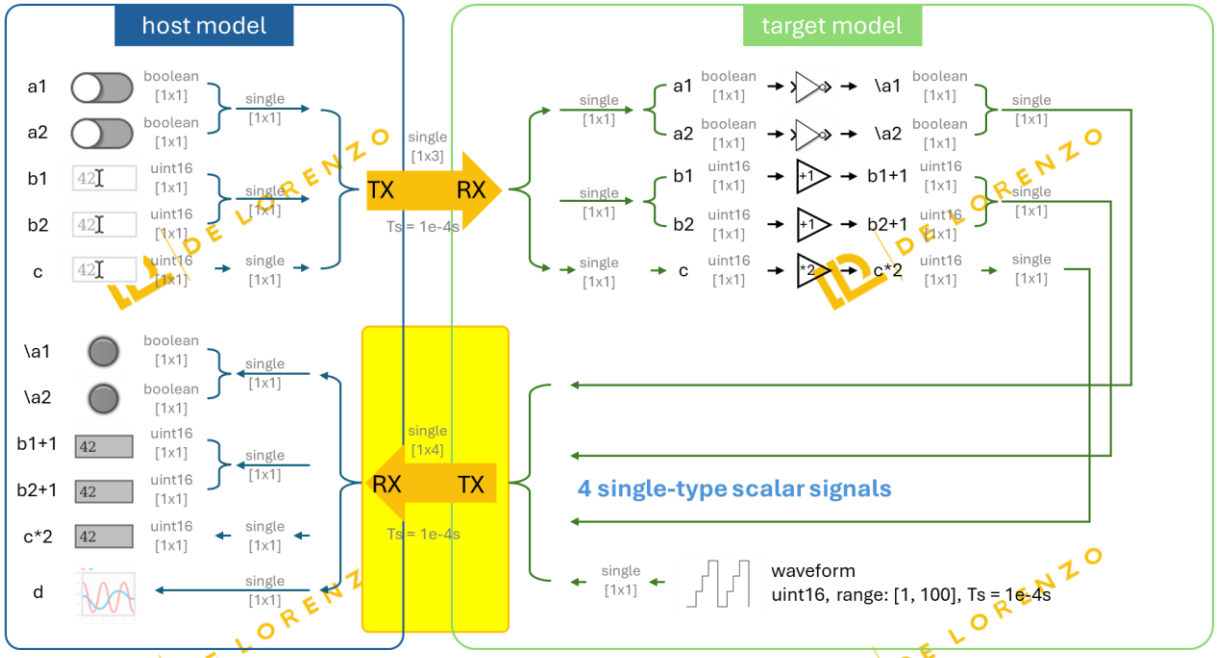
Step 9. Multiplex Same-Type Signals



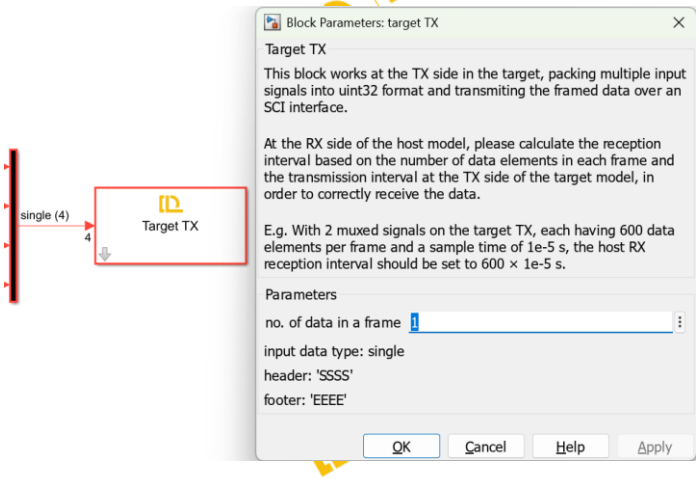
Use a **'Mux'** block to multiplex the four single-type signals into a [1×4] vector in preparation for transmission (TX).

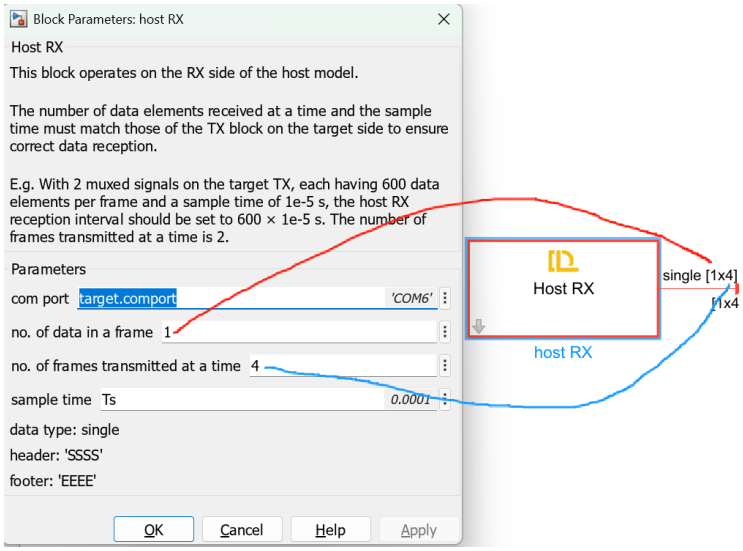


Step 10. Target-to-host transmission



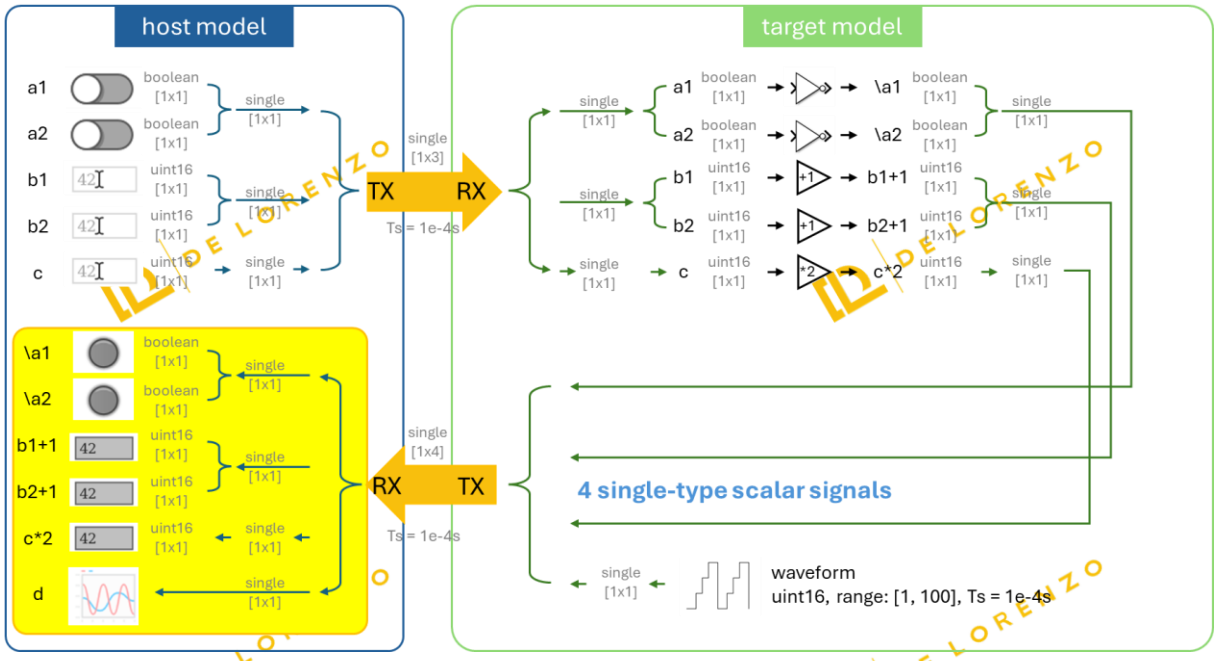
In the **target model**, place the **'Target TX'** block. The transmission rate will follow the target RX interval. Since the transmission and reception intervals on both the host and target sides are identical, and four single-scalar signals ([1×4]) are being transferred, the number of rows in the data—i.e., the amount of data per frame—is set to 1.





In the host model, place the **'Host RX'** block. For receiving four single-scalar signals ([1×4]), set the number of rows to 1 and the number of columns to 4. Ensure that the reception interval matches the transmission interval.

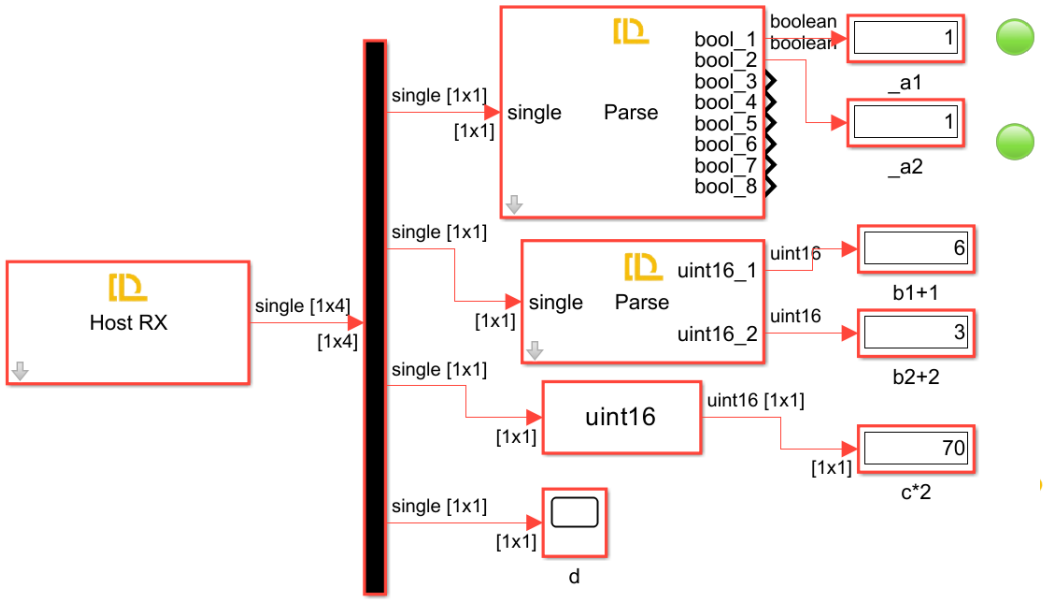
Step 10. Parse to recover the processed variables



Use the **'Demux'** block to split a [1×4] vector into four [1×1] scalar signals.

Use the **Parse** blocks to convert the **single**-type signals into multiple **Boolean** and **uint16** signals and extract the controlled variables from the corresponding channels.

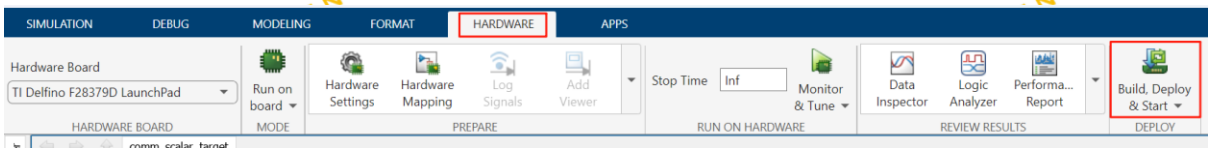
Use the **'Data Type Conversion'** block to restore the data to its original type.



Step 11. Debugging

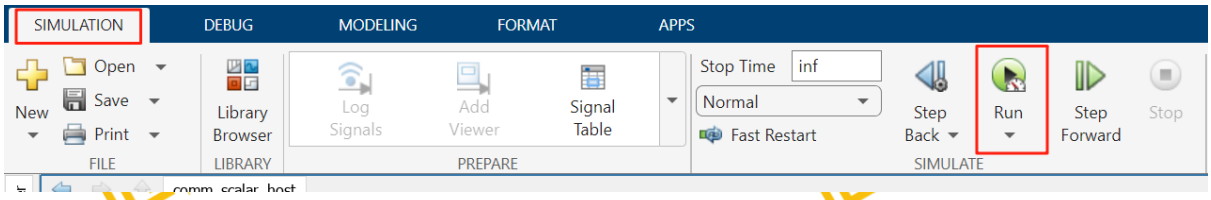
In the **target model**, compile and deploy the model to the DL RCPCORE hardware.

Wait for the 'Ready' message in the lower-left corner. If an error occurs, check and resolve it.

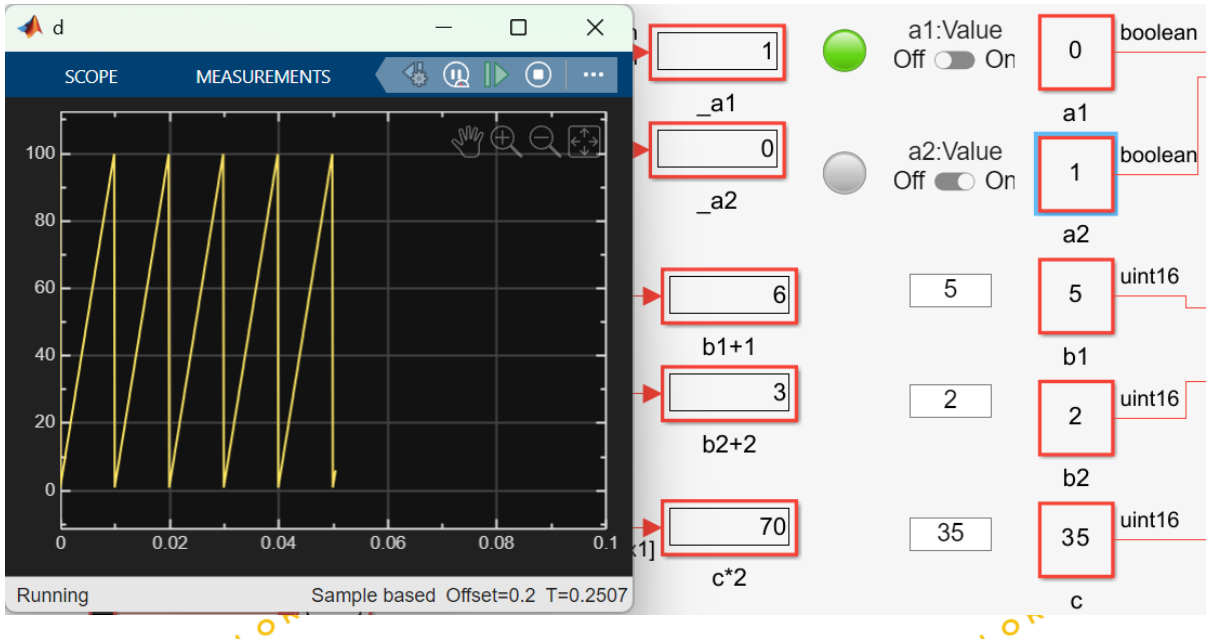


In the **host model**, after confirming that the target model has been deployed and is running on the DL RCPCORE, run the host model to begin debugging.

If an error occurs, check and resolve it.



Modify the control variables and verify whether the values processed by the target model are correctly sent back.



***Note**: Pay attention to the lamp color settings.*