Description

In this example, we will build a host-target (master-slave) communication system to exchange a matrix—comprising multiple scalar signals grouped into arrays, with each signal containing historical data packed into a single frame (e.g., [200 × 3])—as illustrated in the diagram below.

The system is composed of a Host, which is a PC acting as the master, and a Target, the DL RCPCORE acting as the slave. These two components communicate in a bidirectional manner to exchange and process data.

Data transmission from the Host to the Target includes two Boolean scalar values and three uint16 scalar values. In response, the Target sends back two Boolean row vectors and four uint16 row vectors.

The workflow begins with the Host packaging the necessary data and transmitting it to the Target. Upon receiving the data, the Target unpacks it and performs simple processing operations. Once processing is complete, the Target repackages the data into a matrix format and returns it to the Host.

Upon receiving the processed data, the Host unpacks the matrix, converts it into row vectors, and then further breaks it down into scalar values. Finally, the Host displays the results and verifies their correctness, ensuring that the data exchange and processing have been executed accurately.

TX/RX Timing & Signal Dimensions

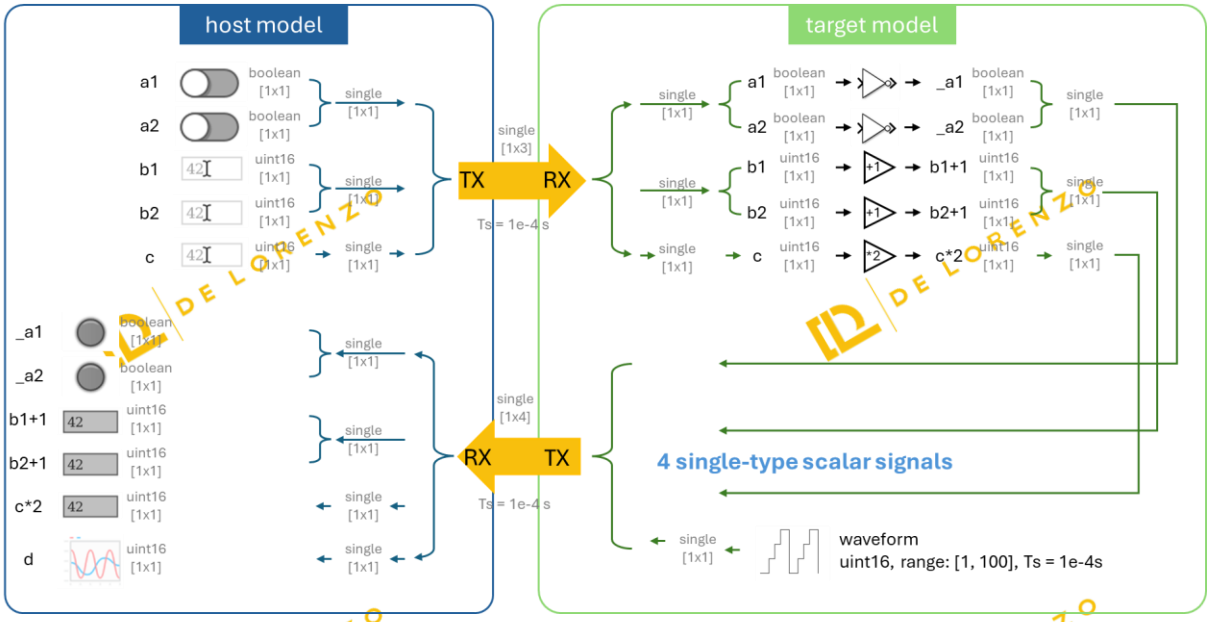
The host model performs a TX [1x3] at intervals longer than Ts = 1e-4 s;
The target model performs RX [1x3] at intervals longer than Ts = 1e-4 s;
The target model performs TX [200x4] at regular intervals of Ts = 1e-4 s;
The host model performs RX [200x4] at intervals equal to Ts*200.

Vector Demo vs Matrix Demo

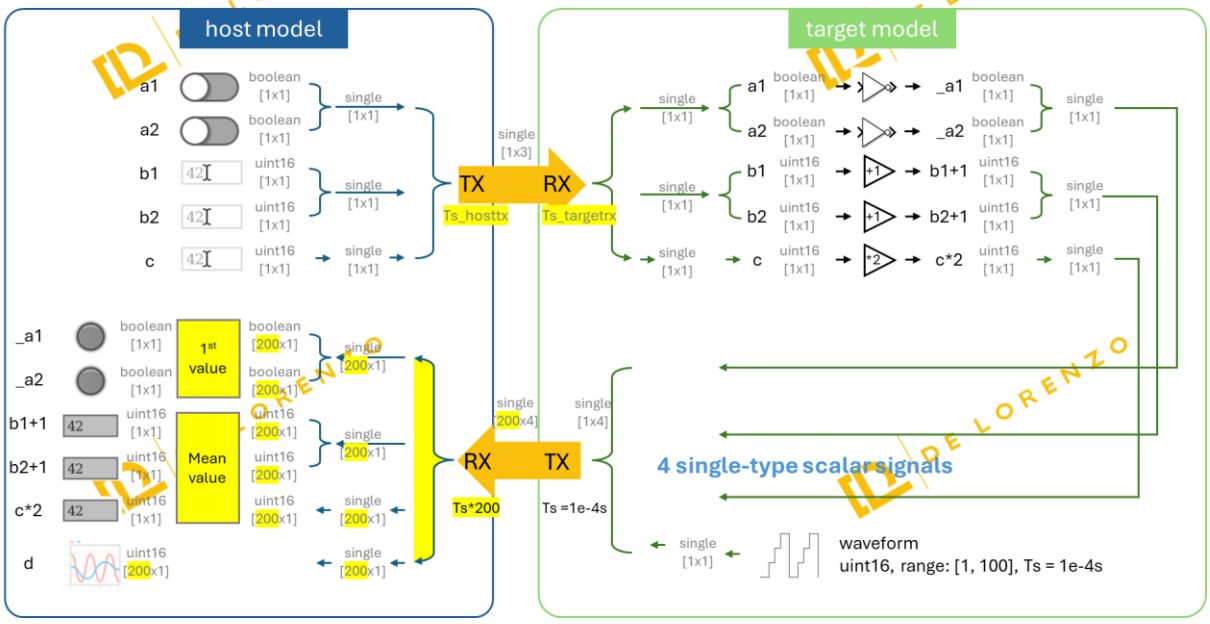
Key Differences:

- Transmission rate
- Enables frame-level data aggregation (number of rows > 1)
- Final output sent as a matrix [rows, columns]
    - Rows: number of elements in one frame (historical data)
    - Columns: number of signals muxed together

The differences are highlighted.



Row Vector Workflow



Matrix Workflow

Learning Objectives

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

Prior Knowledge

- Create New Project
- Control Model Settings
- Host-Target Communication
  - Fundamentals
  - Scalar Demo
  - Row Vector Demo

Demo Models

If you want to run the demo directly, please execute the following command:

openDLDemo('comm_matrix')

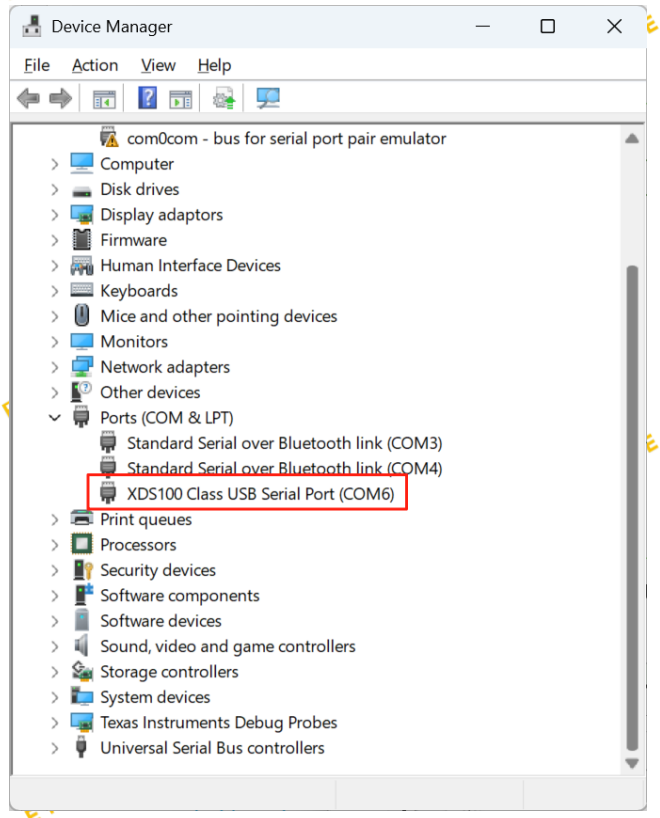
Step 0. Model creation & parameter settings

Create new host and target models using the *CreateNew* command.

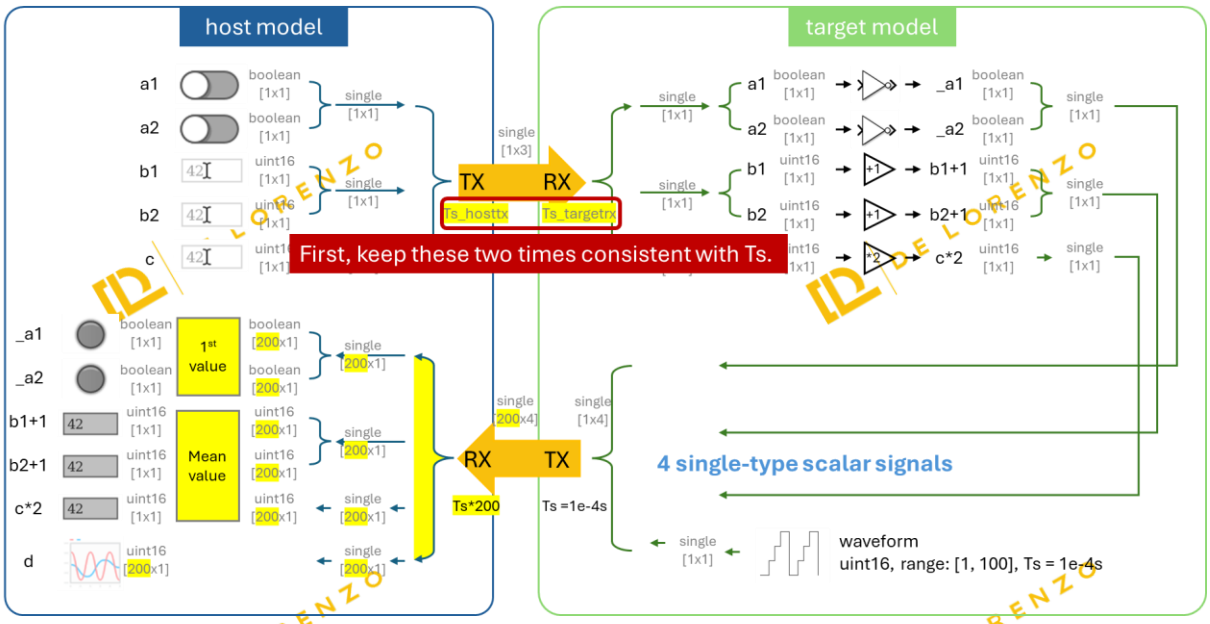
Click on ' 📌 Click here to edit model parameters' to edit the parameters.

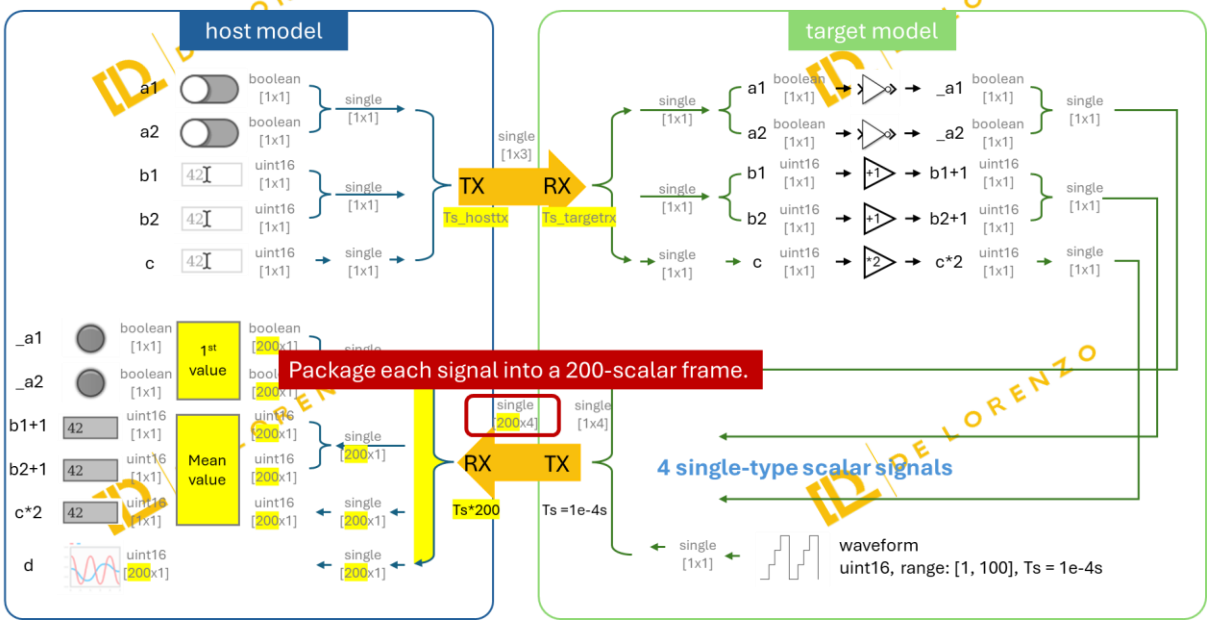
Set the model base sample time **Ts** (which here corresponds to the transmission interval) and the **COM port** recognized by the hardware in Device Manager.

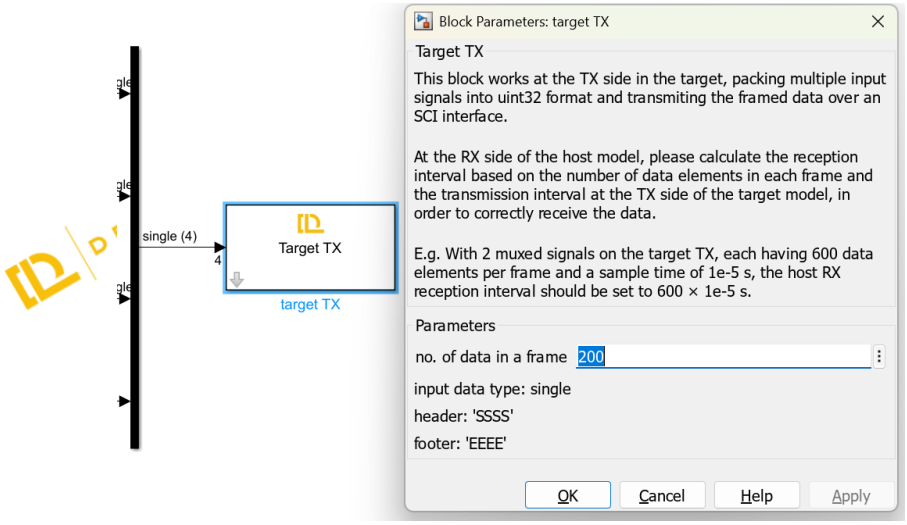
```
%% Set PWM Switching frequency
PWM_frequency    = 4e3;          %Hz          // Converter
T_pwm            = 1/PWM_frequency;  %s  // PWM switching
deadtime         = 5e-6;         %s //rising edge delay fc

%% 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');
```

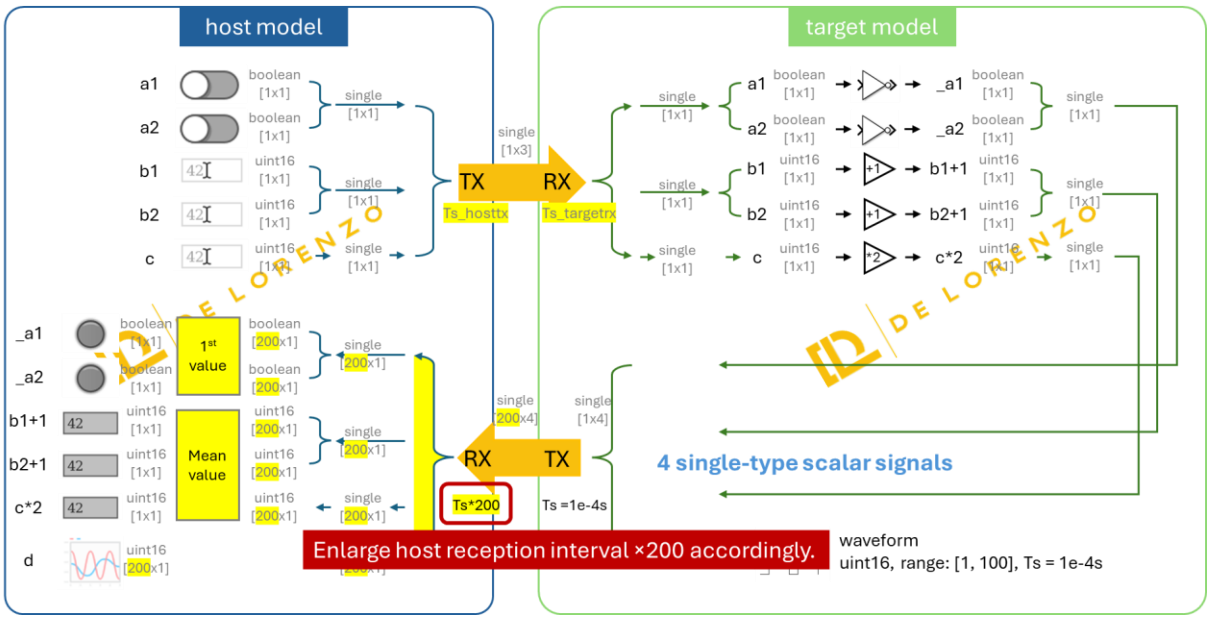


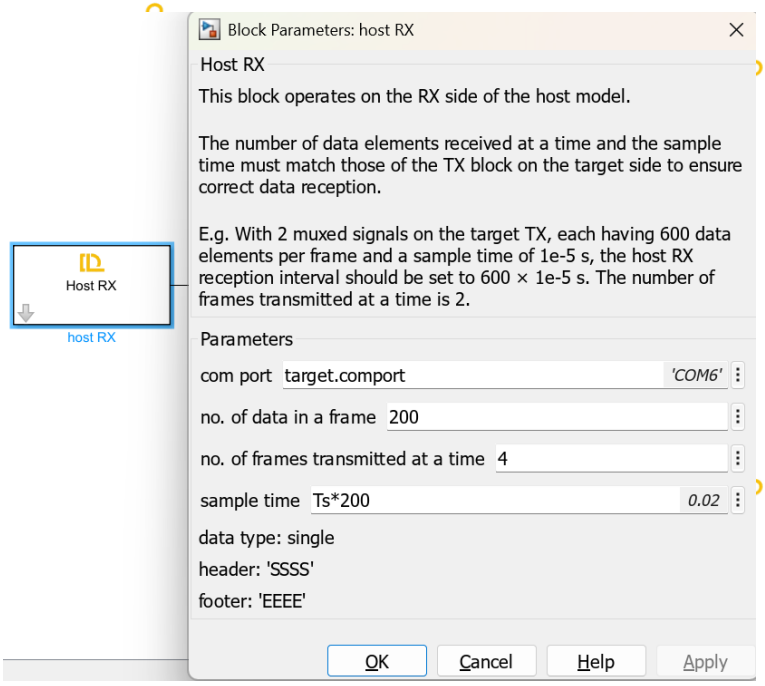
Step 1. Framing



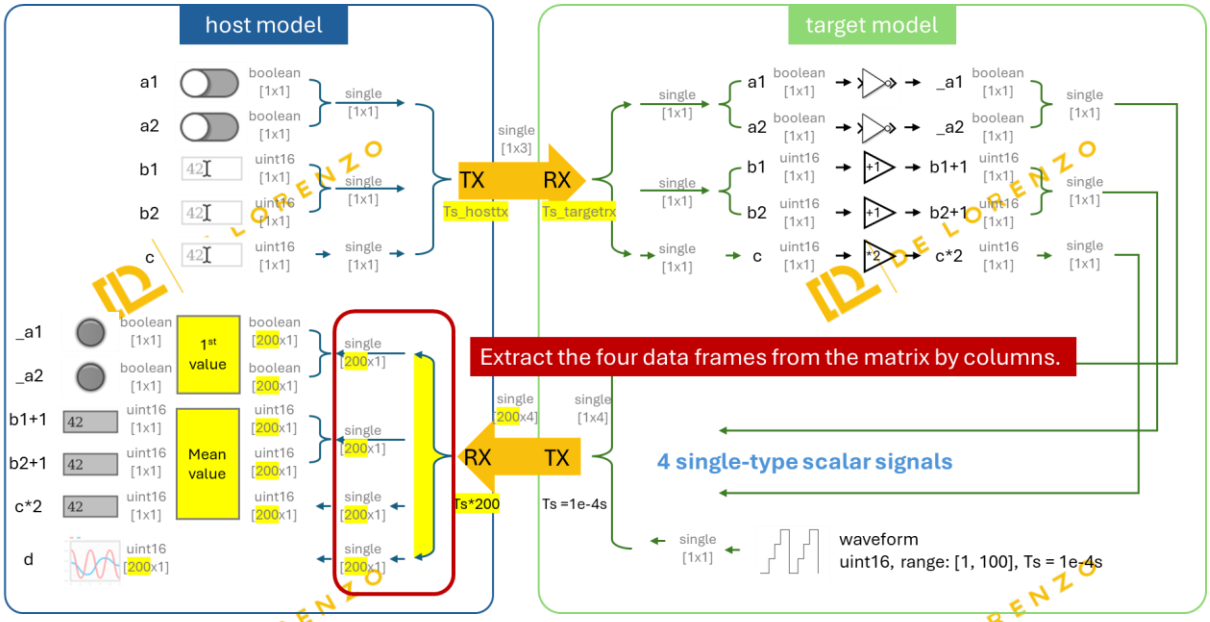




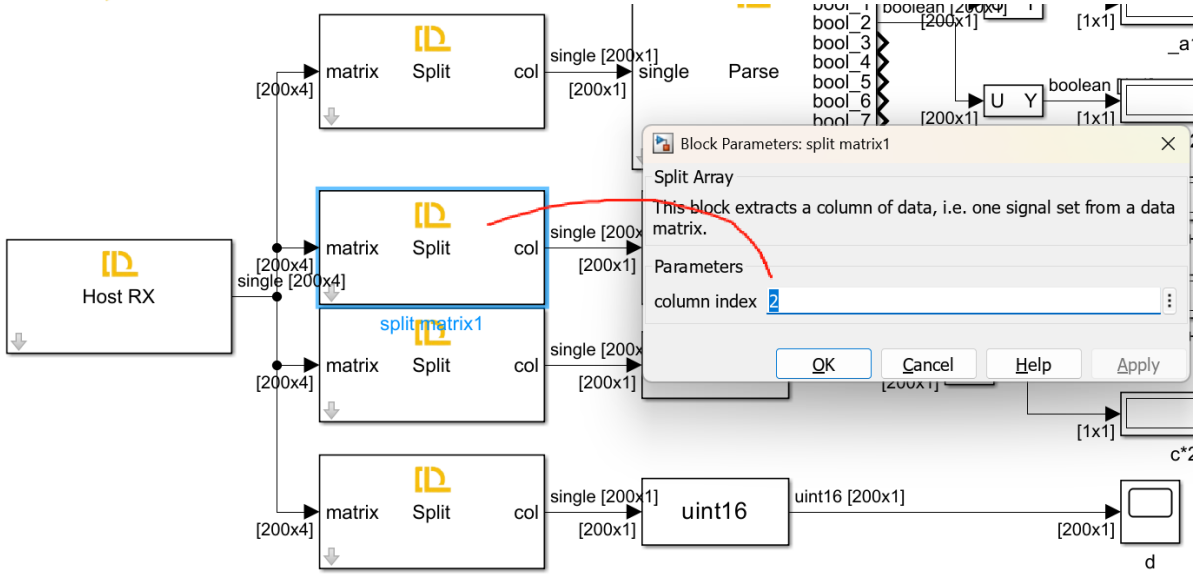

host model
target model
TX
RX
Ts_hosttx
Ts_targetrx
RX
TX
Ts*200
Ts =1e-4s
4 single-type scalar signals
Enlarge host reception interval ×200 accordingly.
waveform
uint16, range: [1, 100], Ts = 1e-4s


Block Parameters: host RX
Host RX
This block operates on the RX side of the host model.
The number of data elements received at a time and the sample time must match those of the TX block on the target side to ensure correct data reception.
E.g. With 2 muxed signals on the target TX, each having 600 data elements per frame and a sample time of 1e-5 s, the host RX reception interval should be set to 600 × 1e-5 s. The number of frames transmitted at a time is 2.
Parameters
com port target.comport 'COM6'
no. of data in a frame 200
no. of frames transmitted at a time 4
sample time Ts*200 0.02
data type: single
header: 'SSSS'
footer: 'EEEE'
OK
Cancel
Help
Apply
Host RX
host RX

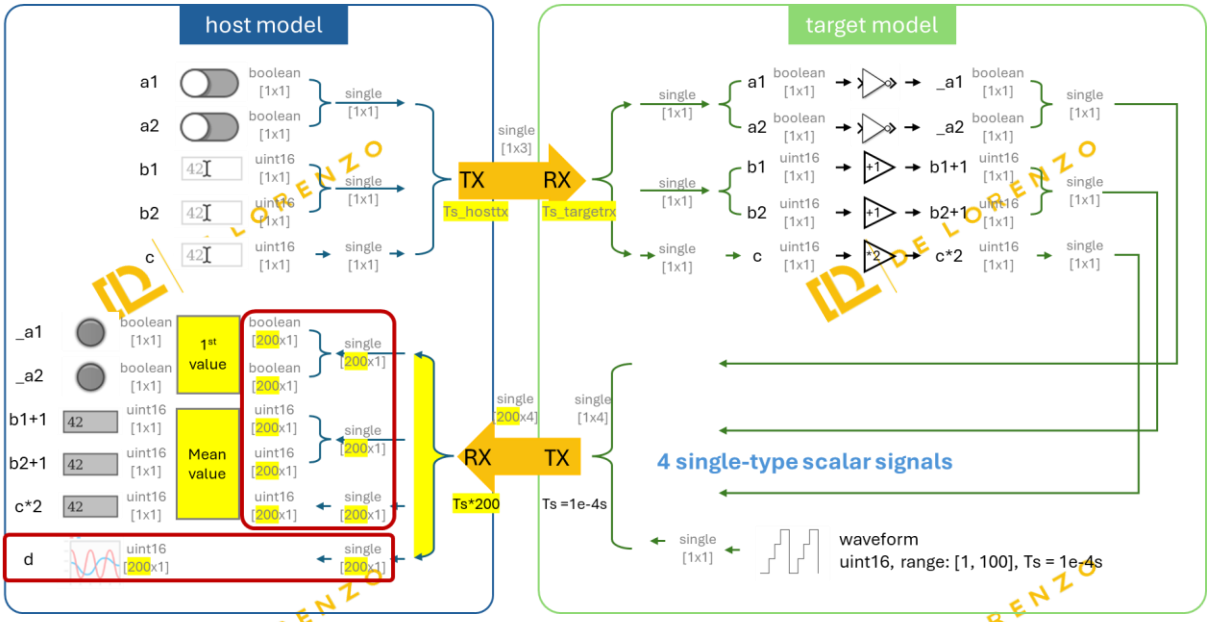
Step 2. Column-Based Data Frame Extraction

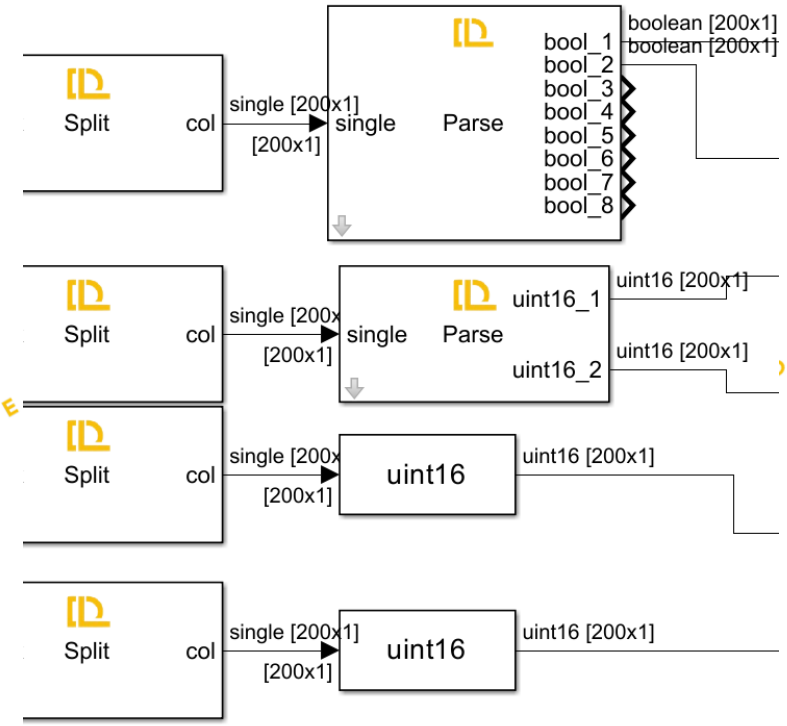


Use the **'split matrix'** block to extract a specific column from the matrix. **'Demux'** cannot be used here.

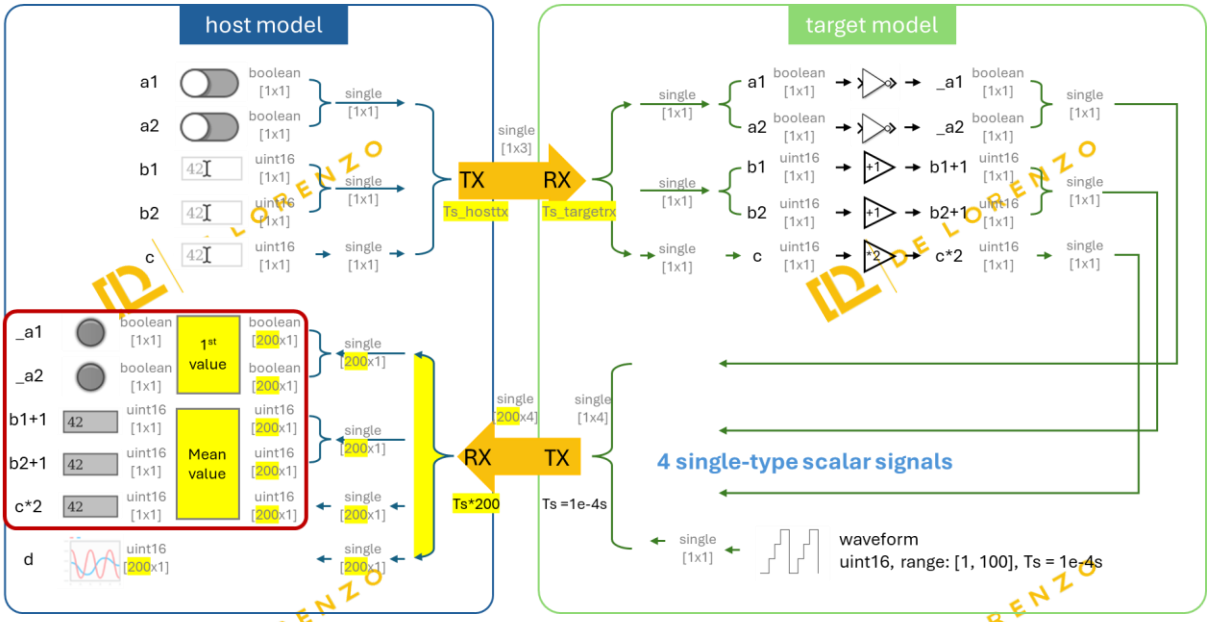


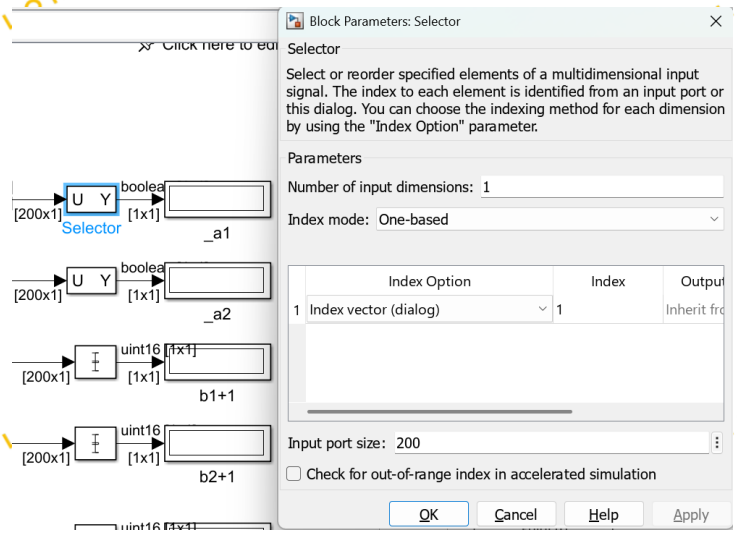
Step 3. Parse and data type convert to recover the controlled variables

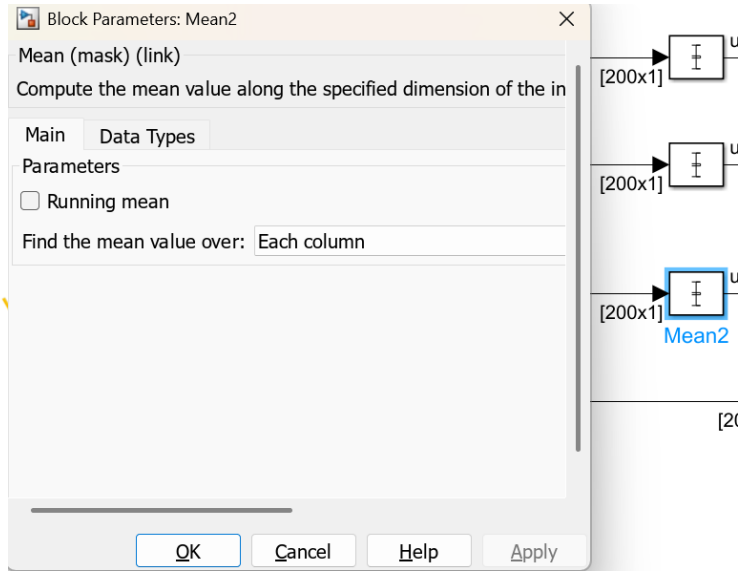




Step 4. Extracting Specific Scalar Value from Row Vector



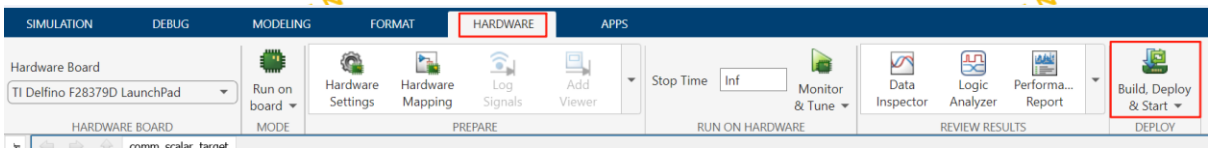




Step 5. Debug

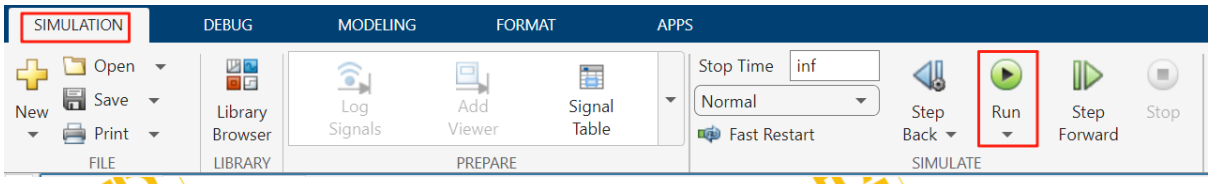
In the **target model**, compile and deploy the model to the DL RCPCORE hardware.

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.

5.1 Communication Failure Analysis

We can observe that in the host model, when controlling a1, after the sequence host TX → target RX → _a1 = invert(a1) → target TX → host RX, the value of _a1 read back does not properly respond to the changes of a1.

Meanwhile, the scope does not display the expected sawtooth waveform read back through target TX → host RX, but instead shows a 404 error code.

Therefore, we can infer that both host TX → target RX and target TX → host RX are not functioning. The most likely cause of this issue is target overload.

5.2 Reduce the load on the target

Normally, the target receives commands and parameter updates from the host, and these changes depend on the user of the application; therefore, a particularly high reading frequency is not necessary.

On the other hand, the target returns control results or intermediate values to the host for observation. To ensure signal integrity, this is usually transmitted at a relatively high and fixed frequency.

Therefore, we can reduce the RX reading speed in order to free up some of the target's resource usage.

For example, the target RX reading interval can be extended from the original **Ts = 1e-4 s** to **Ts × 1000 = 0.1 s**. A reading interval of 0.1 s for user-side updates appears reasonable.

After model compilation, an error is reported since the mux block preceding the target TX attempts to merge four signals with different rates (0.1 s and 1e-4 s), which is not permitted.

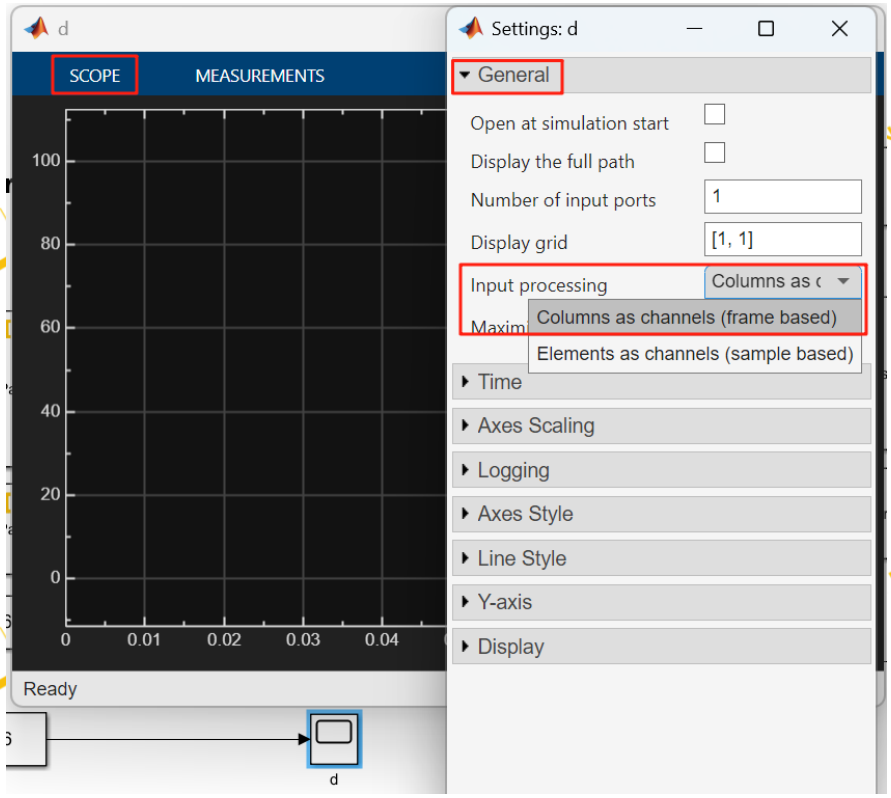
To ensure that the transmission rate of the target TX remains at 1e-4 s, we need to add **Rate Transition** blocks to the three signals with a rate of 0.1 s before the mux. This will automatically convert their rates to 1e-4 s, thus unifying the signal rates.

Click the **Build, Deploy & Start** button in the **Hardware tab** to begin compiling the model and deploy it to the DL RCPCORE.

After completion, return to the host model to retest.

5.3 Communication Failure Analysis

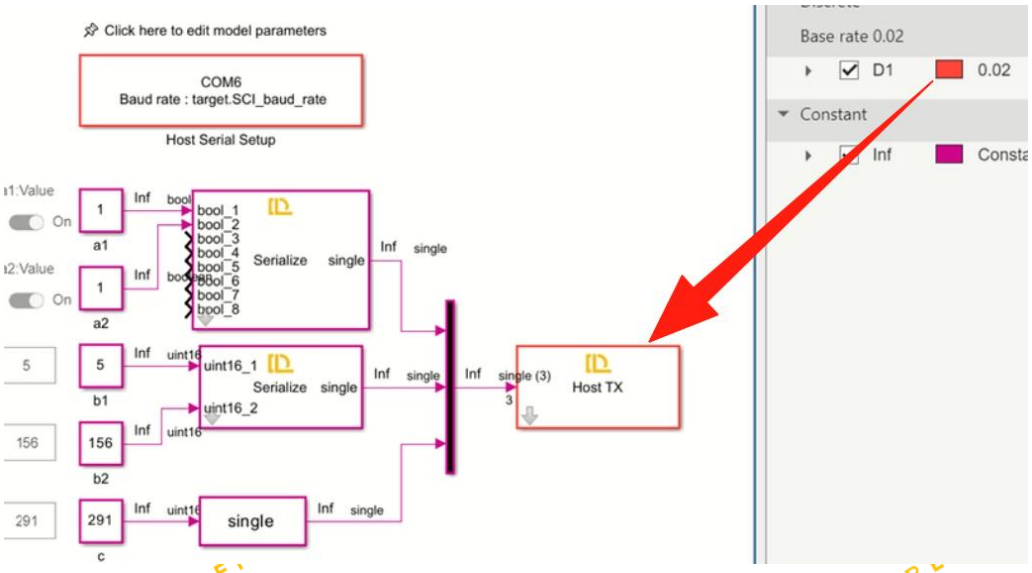
We can observe that in the host model, when controlling a1, after the sequence host TX → target RX → _a1 = invert(a1) → target TX → host RX, the value of _a1 read back does not properly respond to the changes of a1.



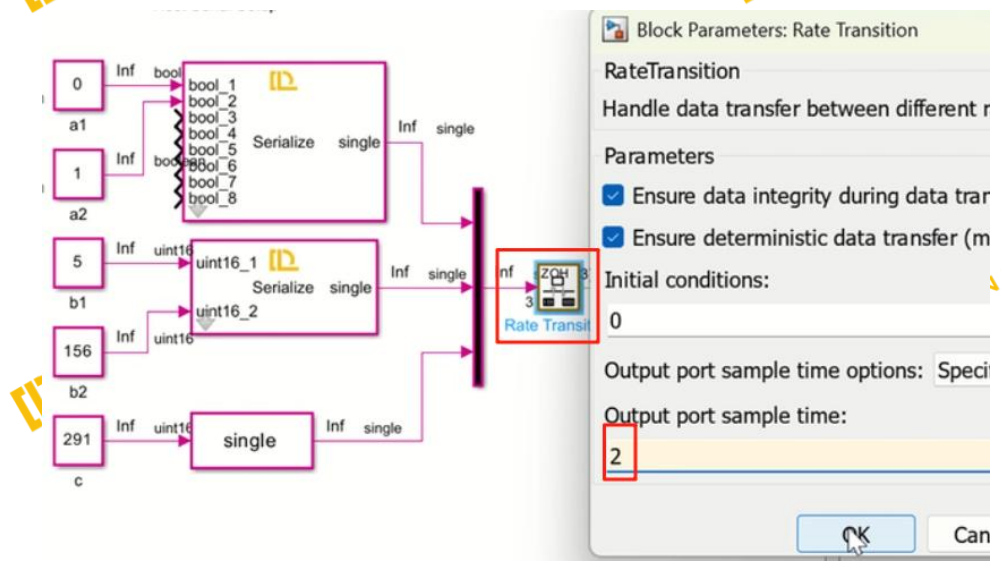
However, the scope starts to show many lines different from the 404 error code. We only need to configure the oscilloscope as follows**: Setting → General → Input Processing: Columns as Channels (frame based)**, which will allow the scope to display the sawtooth waveform.

It is evident that target TX ⟷ host RX functions correctly, whereas host TX → target RX → _a1 = invert(a1) → target TX → host RX fails to operate, i.e. host TX → target RX fails to operate. We can infer that after we reduced part of the workload, the target is able to run. Therefore, we will focus our attention on host TX.

By analyzing the rates of the various blocks in the host model, we find that the current rate of **host TX** is **0.02 s**.



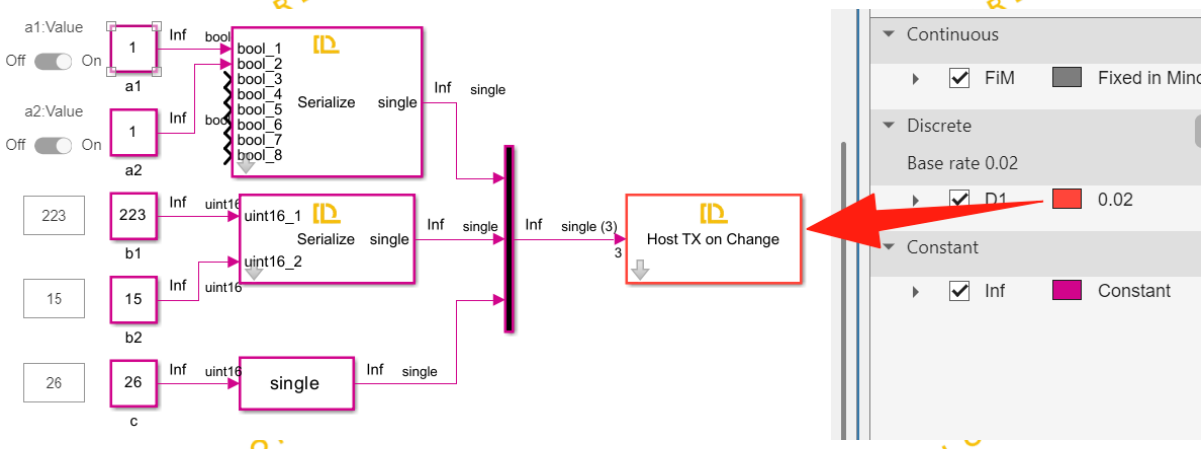
We can try reducing its transmission interval to **2 s**. The specific implementation is to add **a Rate Transition** block before **host TX** and set its output port sample time to **2 s**.



Now, all communications are functioning normally.

5.4 Optimize host-side timing

However, allowing commands to be transmitted to the hardware every 2 s does not seem ideal. Therefore, we can try sending a command only when an update occurs, while setting the polling frequency of the host TX block for command changes back to the original 0.02 s.



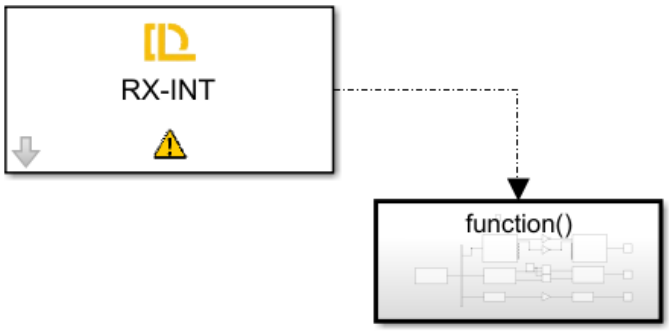
Testing shows that communication works normally, and the response speed has improved significantly.

5.5 Optimize target-side timing

A similar idea leads us to consider whether, on the target side, performing a reception only when data is actually transmitted could further reduce the target's workload.

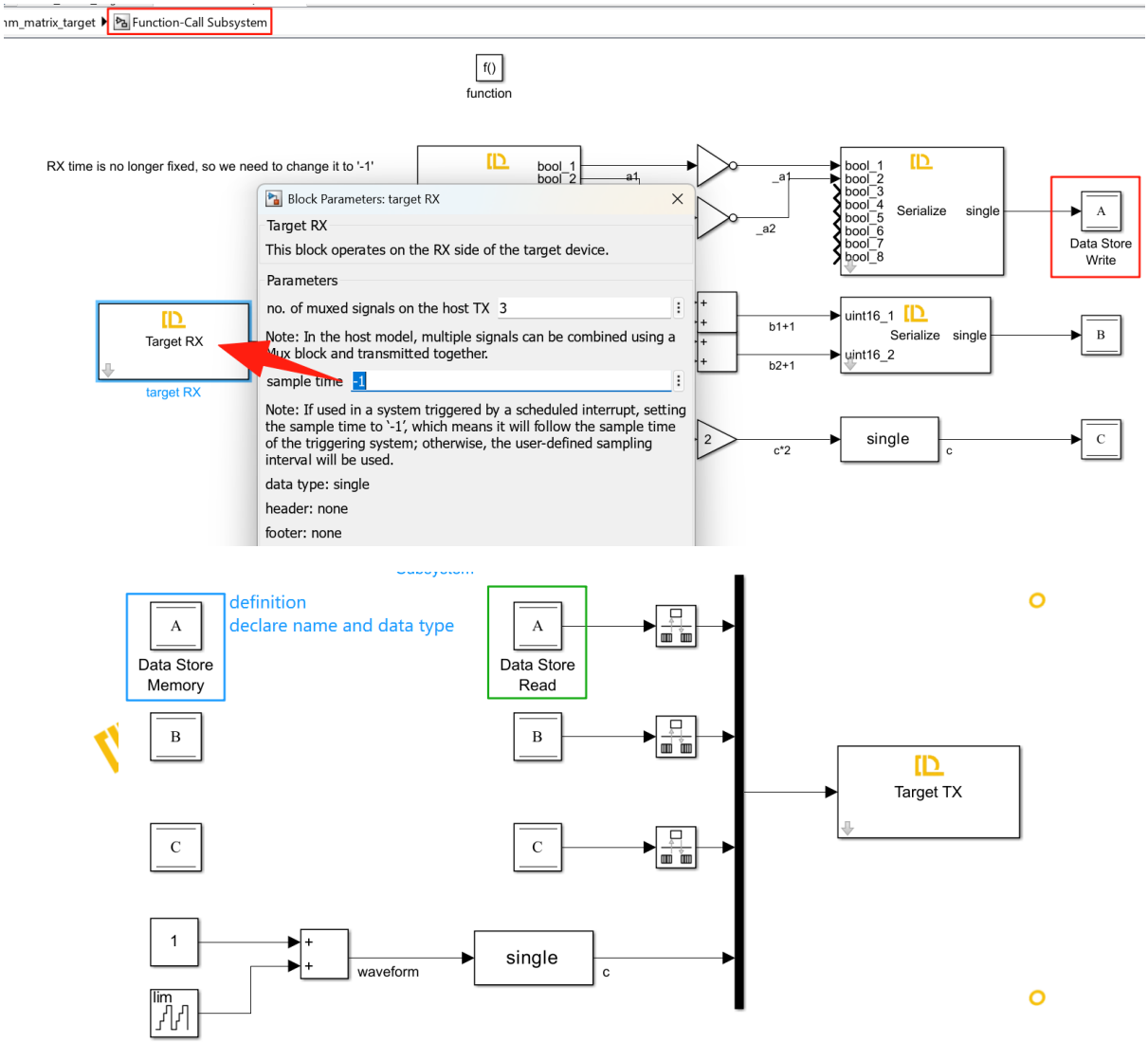
Therefore, we can rely on RX interrupts to realize this idea.

Whenever the target receives an updated command or parameter from the host, the **RX-INT** block generates an interrupt event to perform data reading and some basic processing, such as parsing, in a **Function-Call Subsystem**.



The parsed data needs to exit the function-call subsystem and reach the mux block before the target TX. Here, we recommend using **Data Store Memory/Write/Read** to achieve this.

In the interrupt routine, the sample time of target RX must be set to -1, since the interrupt trigger time cannot be predetermined.



Testing shows that communication works normally.