1. Introduction

My library blocks provide CAN communication feature using CAN Controller device MCP2515 via SPI. This documentation explains how to prepare and use them by using two sample models.

2. Preparation

2.1. Required Software

- MATLAB® (Tested version: R2017a) (*1)
- Simulink®
- Arduino® Support from Simulink
- Raspberry Pi® Support from Simulink

(Recommended Software)

- Embedded Coder® (*2)
- MATLAB® Coder
- Simulink® Coder
- Fixed-Point Designer®

2.2. Required Hardware

- Arduino® UNO (*3)
- Raspberry Pi® 2/3.
- CAN device module with MCP2515 (*4)
- USB cable
- Breadboard wires

(*1) SPI Master Transfer block for Raspberry Pi® was be provided from R2017a

(*2) Embedded Coder has the code optimization option

(*3) Arduino® UNO could be built and downloaded but was not available external mode due to the out of memory.

(*4) Tested CAN device modules are the followings.

- Arduino® CAN-BUS shield
  http://wiki.seeed.cc/CAN-BUS_Shield_V1.2/
- PiCAN2 CAN-Bus Board for Raspberry Pi® 2/3
  http://skpang.co.uk/catalog/pican2-canbus-board-for-Raspberry Pi®-23-p-1475.html

If you get MCP2515+TJA1050 CAN Bus Module for Arduino®, please check appendix.

3. Library

3.1. Library setting

You can use my library on Simulink® library browser in the following steps.

(1) Add my library file paths to MATLAB® path

(2) Open Simulink® library browser

(3) Click 'F5' key

3.2. Library blocks

CAN application and mcp2515 driver's blocks contain SPI driver blocks for hardware target(Arduino® and Raspberry Pi®). CAN application blocks are used for converting between SPI data and CAN data. mcp2515 driver blocks are kinds of utility blocks for SPI communication.

3.3. CAN application

3.3.1. CANbegin

[Description]

This block is executed only once in the beginning of CAN communication because MCP2515's register is needed to be configured. And the frequency of oscillator for MCP2515 is also needed because it effects to the setting parameters of MCP2515's register.

After setting the parameters, the workspace variables "CSpin" and "Hardware" will be set.

The following figures are the oscillator on my tested hardware (CAN Bus Module).

[Parameters]

- Target Hardware: Select which hardware is executed.
  Arduino -> Hardware = 1, Raspberry Pi ->Hardware = 2
- Oscillator frequency(MHz): Select the value of oscillator frequency
- CAN speed(kHz): Select CAN speed frequency
- Chip Select Pin: Set chip select pin number. This option is appeared by setting Target Hardware = 'Arduino'.

3.3.2. CANreceive

[Description]

CANreceive outputs CAN data by converting MCP2515 RX buffer register's values. In/Out ports are no use because they are used for controlling the execution order. CAN_Received outputs 1(CAN received) or -1(No message).

3.3.3. CANtransmit

[Description]

CANtransmit inputs CAN data by converting MCP2515 TX buffer register's values. In/Out ports are no use because they are used for controlling the execution order. CAN_Transmitted outputs 1(Success) or 2(Fail).

4. Model

4.1. arduino_CAN_Echo

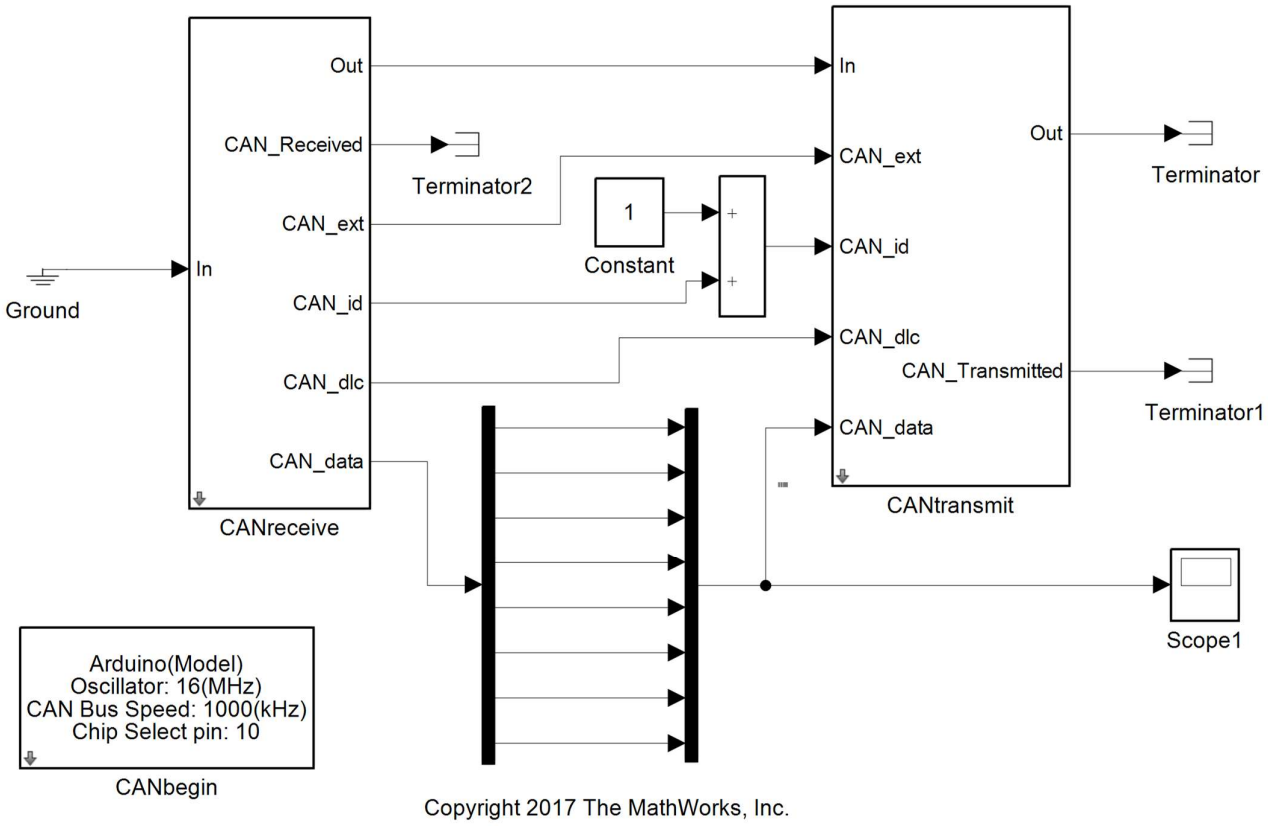
[Tested Hardware]

Arduino® Uno

Arduino® CAN-BUS shield (My tested oscillator frequency was 16(MHz))

This system is loopback the received CAN data at “received ID+1”.

The following figure is the top layer of arduino_CAN_Echo.



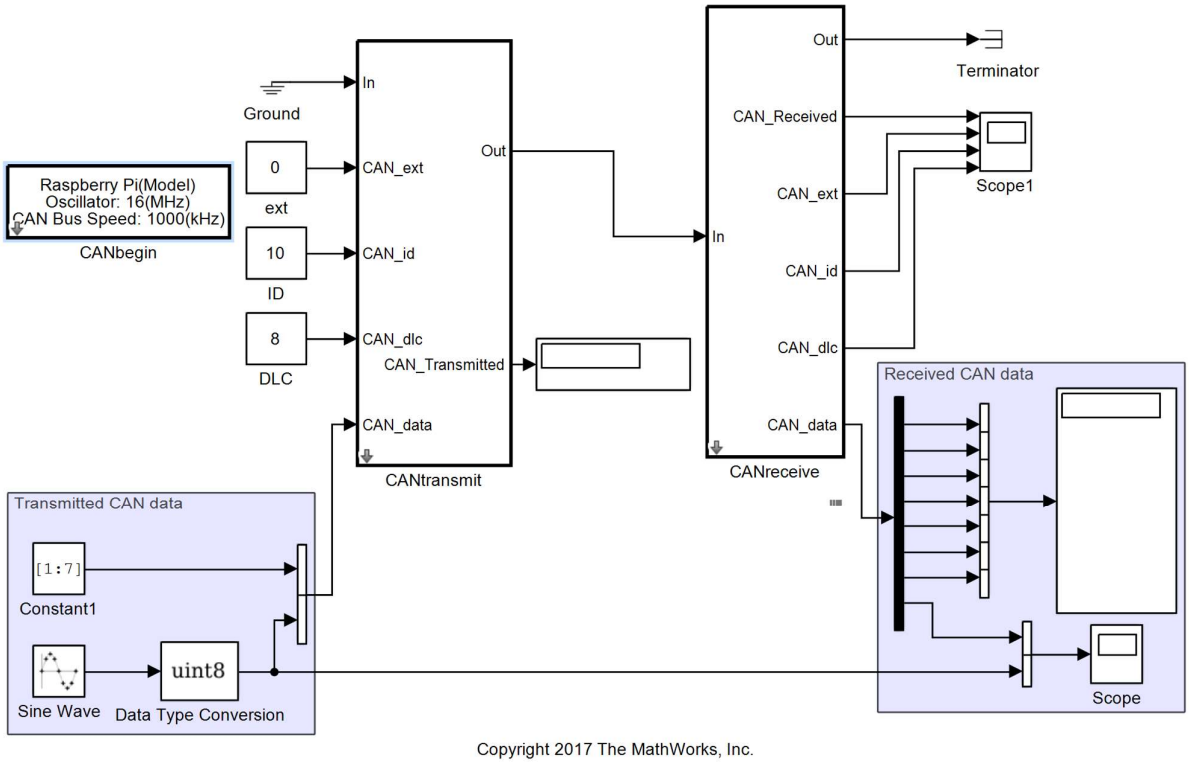
4.2. raspi_CAN_SendRecieve

[Tested Hardware]

- Raspberry Pi® 3
- PiCAN2 CAN-Bus Board for Raspberry Pi® 2/3 (My tested oscillator frequency was 16[MHz])

This model outputs the sine wave CAN data(ID:10) from CANtransmit of loop subsystem to the connected another target (Arduino® Uno). And then, CANreceive receives the echo data from Arduino® Uno.

The following figure is the top layer of raspi_CAN_SendRecieve.



5. Test steps(pattern1)

5.1. Test pattern

(1) Connect Arduino® Uno with Arduino® CAN shield to host PC via USB cable.

(2) Open arduino_CAN_Echo.slx

(3) Check the oscillator frequency of Arduino® CAN shield (*) and set the value to CANbegin.

(*) If CS pin of Arduino® CAN shield is different from 10, you need to change the SPI pin number of SPI WriteRead block for Arduino® block in SPI driver layer's blocks.

(4) Build the model and download to Arduino® Uno

(5) Change the USB connection to host PC from Arduino® Uno to Raspberry Pi® 3

(6) Connect Raspberry Pi® 3 with PiCAN2

(7) Connect CAN cable between Arduino® CAN shield and PiCAN2.

(8) Connect Arduino® Uno to the external power.

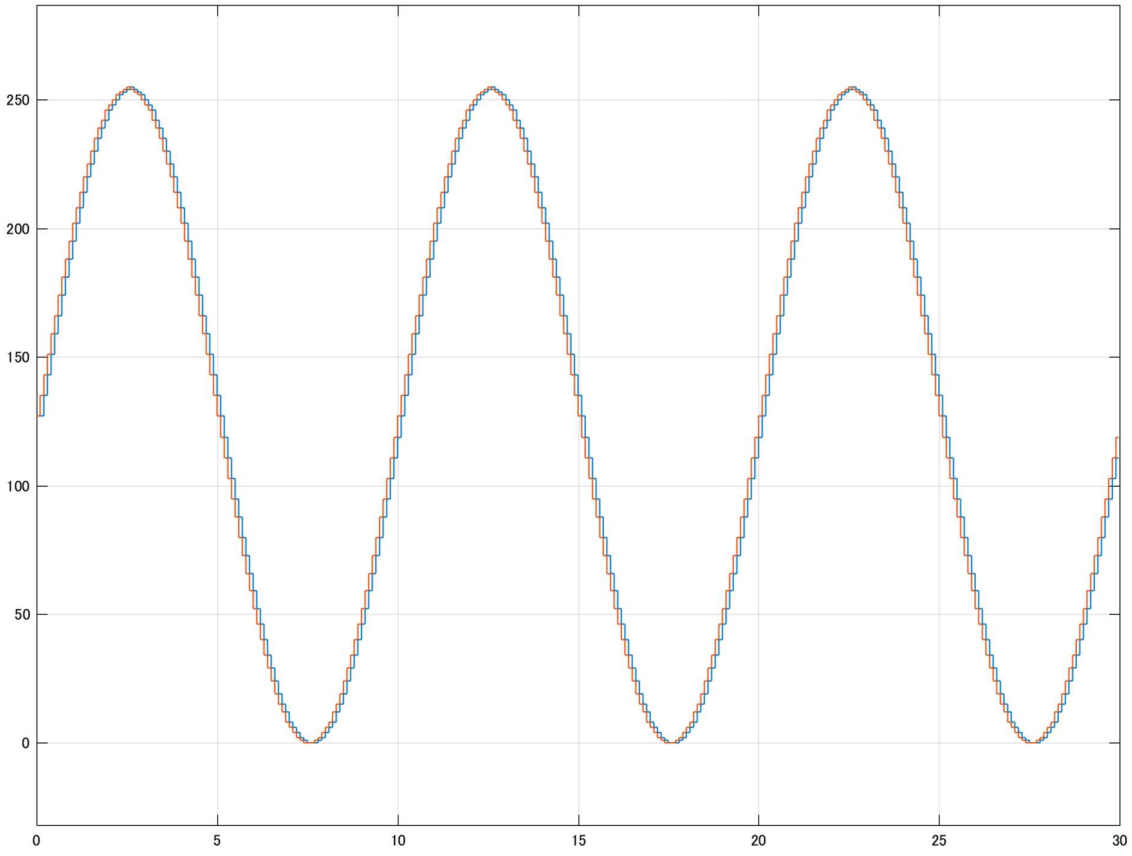
(9) Open raspi_CAN_SendRecieve.slx

(10) Set the oscillator frequency of CANbegin to 16[MHz]

(11) Execute raspi_CAN_SendRecieve.slx on external mode

(12) CANRx_id will output "11" which equals 10(CANTx_id) +1. And Scope2 block's window will show the sine wave as the transmitted and received CAN data.

The following figures are tested results of Scope.



6. Related materials

There are other features about MCP2515. And my library blocks can be modified for adding other features. The following materials are helpful for the modification.

- MCP2515 data sheet

English:http://ww1.microchip.com/downloads/en/DeviceDoc/20001801H.pdf

Japanese:http://ww1.microchip.com/downloads/jp/DeviceDoc/21801D_JP.pdf

- CAN-BUS Shield V1.2

http://wiki.seeed.cc/CAN-BUS_Shield_V1.2/

7. Appendix

- MCP2515+TJA1050 CAN Bus Module for Arduino®

If you get MCP2515+TJA1050 CAN Bus Module for Arduino®, you need to set it in the followings.

The following table is the pin mapping between them

Arduino® Mega 2560 Pin No	TJA1050 CAN Bus Module
-	INT
52	SCK
51	SI(MOSI)
50	SO(MISO)
10	CS
GND	GND
5V	VCC

The following figure is the pin connection between them