CAN Bus Analyzer/Simulator for LabVIEW

User Manual

Table of Contents

1 Introduce

Our CAN Bus Analyzer/Simulator Hardware can be used under LabVVIEW platform (LabView 2019 or higher) for windows OS.
We call Analyzer/Simulator CAN BUS Controller in LabVIEW.
Let us introduce CAN Bus Controller function.
Our CAN Bus controller can transmit/Receive standard or/and extended CAN Bus data and RTR frames as general CAN bus node.
And furthermore, transmitting CAN Bus Frame can be synchronized by special CAN BUS transmitting frame or receiving frame. If sync is enabled, CAN Bus controller only can transmit CAN Bus frame when it receives or transmits this special CAN BUS frame. This special CAN BUS frame is called "Sync frame"
Another feature is that our CAN Bus Controller has Watchdog function.
Watchdog purpose is for getting communication fault information.
Received Watchdog (Call it RxWatchdog) is for itself to know whether CAN BUS communication OK or not.
Transmitted Watchdog (Call it TxWatchdog) is for other CAN bus nodes to know whether CAN BUS communication OK or not.
TxWatchdog CAN Bus frame send out periodically automatically by hardware, You don't need to send by calling block diagram node.
Similarly, if "Sync frame" is "Transmit frame", "Sync frame" send out periodically automatically by hardware, You don't need to send by calling block diagram node.

RxWatchdog has 3 kinds of work mode:

1. Work mode 0: RxWatchdog initial value is 0. RxWatchdog is free running up-counter each ms. RxWatchdog value will return 0 if we receive Watchdog CAN Bus frame, and value is different from previous received value. When RxWatchdog value is over RxWatchdog Period, it will keep the Rxwatchdog value and communication fault will occur.
2. Work mode 1: RxWatchdog initial value is 0. RxWatchdog is free running up-counter each ms. When it arrive at period value, it will keep it, and one communication fault will occur. RxWatchdog Data packet (received by CAN Bus) can change counter value directly to avoid communication fault. (Not like mode0, in mode 0, it is clear counter when received watchdog value is different from previous one)
3 Work mode 2: RxWatchdog initial value is equal to period. RxWatchdog is free running down-counter each ms. When it arrive at 0, it will keep 0 and communication fault will occur. RxWatchdog Data packet (received by CAN Bus) can change counter value directly to avoid communication fault.

TxWatchdog has 4 kinds of work modes (TxWatchdog CAN Bus frame send out periodically

automatically by hardware for all modes):

1. Work mode 0: TxWatchdog value is decided by calling sendTxWatchdogValue node.
2. Work mode 1: TxWatchdog value increases 1 every TxWatchdog's period automatically. You don't need to call sendTxWatchdogValue node
3. Work mode 2: TxWatchdog value decreases 1 every TxWatchdog's period automatically. You don't need to call sendTxWatchdogValue node
4. Work mode 3: TxWatchdog will have no any value, its data packet length is zero. It is used for CANOpen Sync frame

Notes: *"Sync frame " can use "RxWatchdog/TxWatchdog frame" to replace, and in this situation, the CAN ID setting of sync frame will be ignored.*

2 How to use CAN bus Controller Sub VI?

Please follow steps below:

- **Step1** Download CAN bus Controller Sub VI from click CANBusControl.zip
- **Step2** unzip CANBusControl.zip to your any destination in your computer.
  You will see Folder : "Any Folder you like" \ CANBusControl
- **Step3** In your application VI, Right click in Block diagram, context menu popup below:

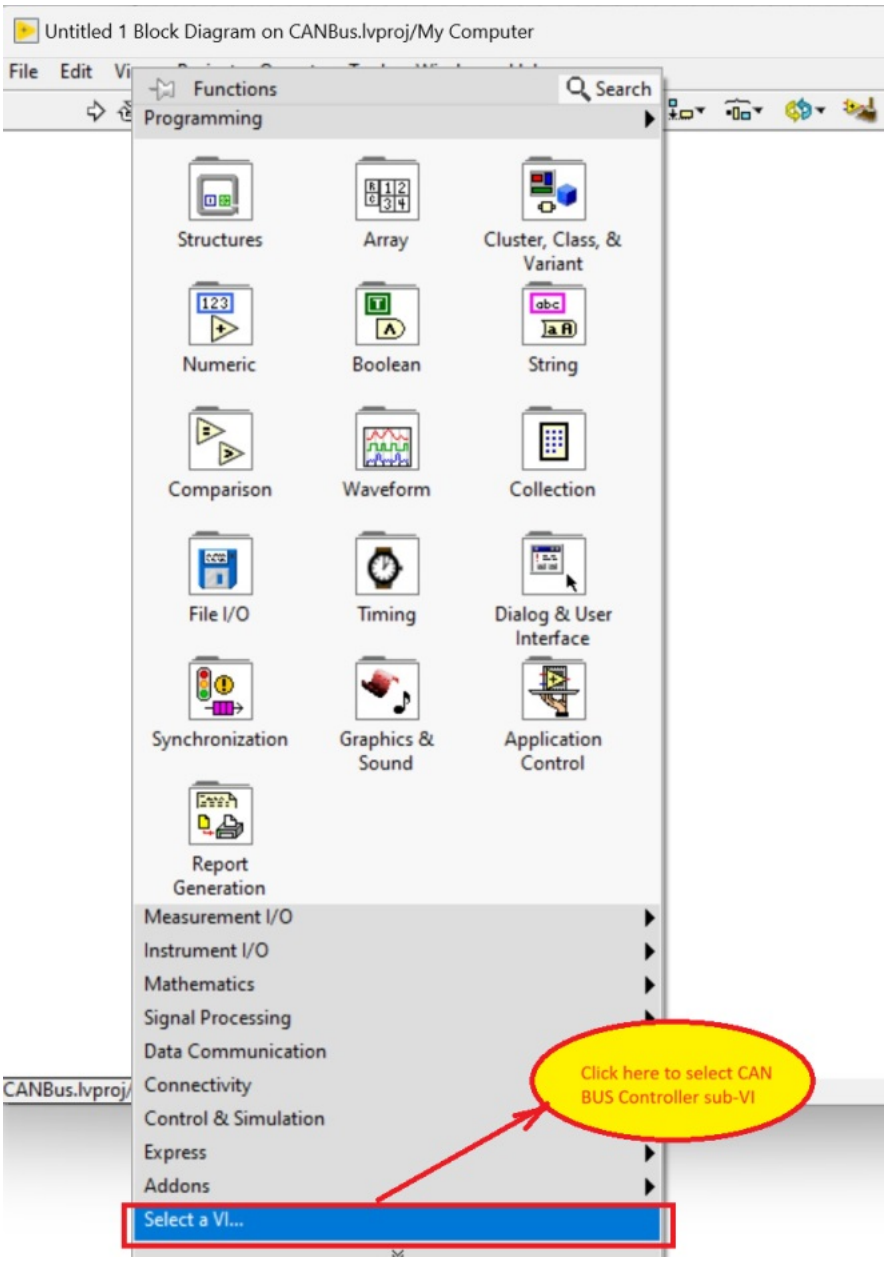


Fig.1 Open CAN Bun Controller Sub-VI

When you click context menu item "Select a VI..." , the dialog below pops up:

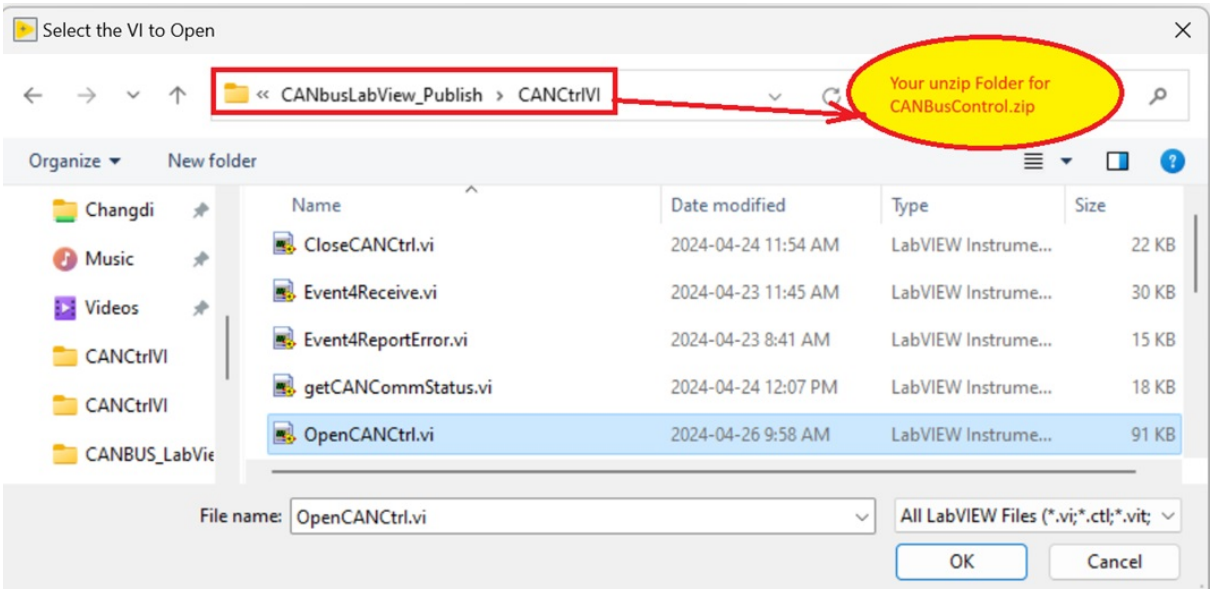


Fig2 Select CAN bus Controller block diagram node

Please select CAN bus Controller block diagram node you want as Fig.2. And clik "OK" button. For example, you want to use OpenCANCtrl node as Fig.2, you will see this node in your block diagram below:

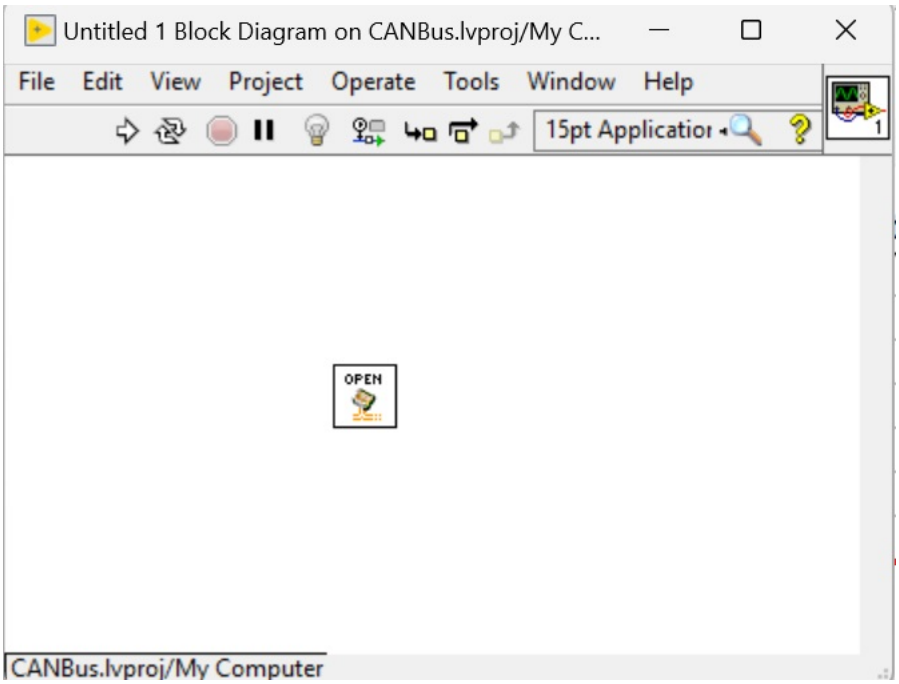


Fig.3 OpenCANCtrl Node

If you put Cursor on "Question mark" of top right corner and click, and then put your mouser on your node as Fig 4 below, and click, you will see help about this node.

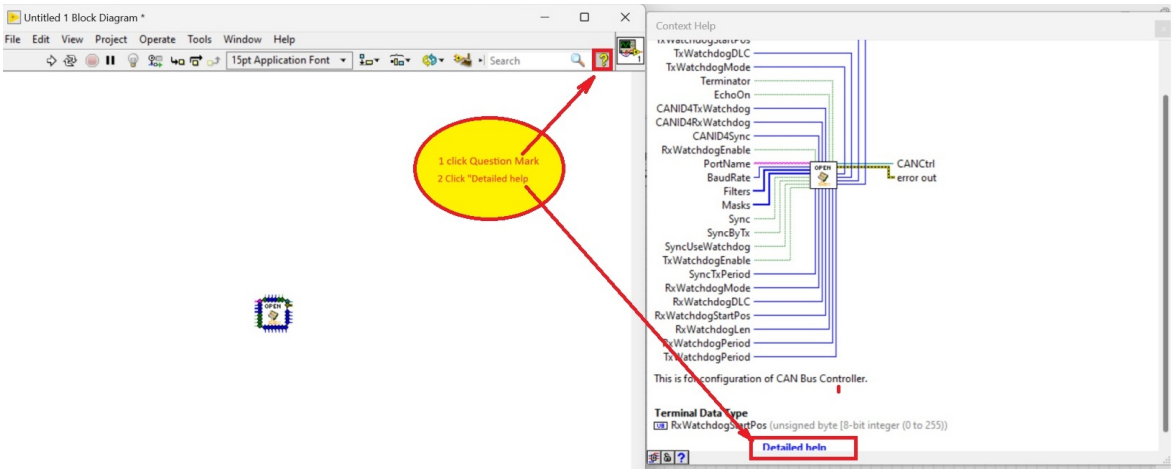


Fig.4 Node help

After you click "Detailed help" in Fig.4, you will see Fig.5 for detailed help and simple example for using this node in Fig.6:

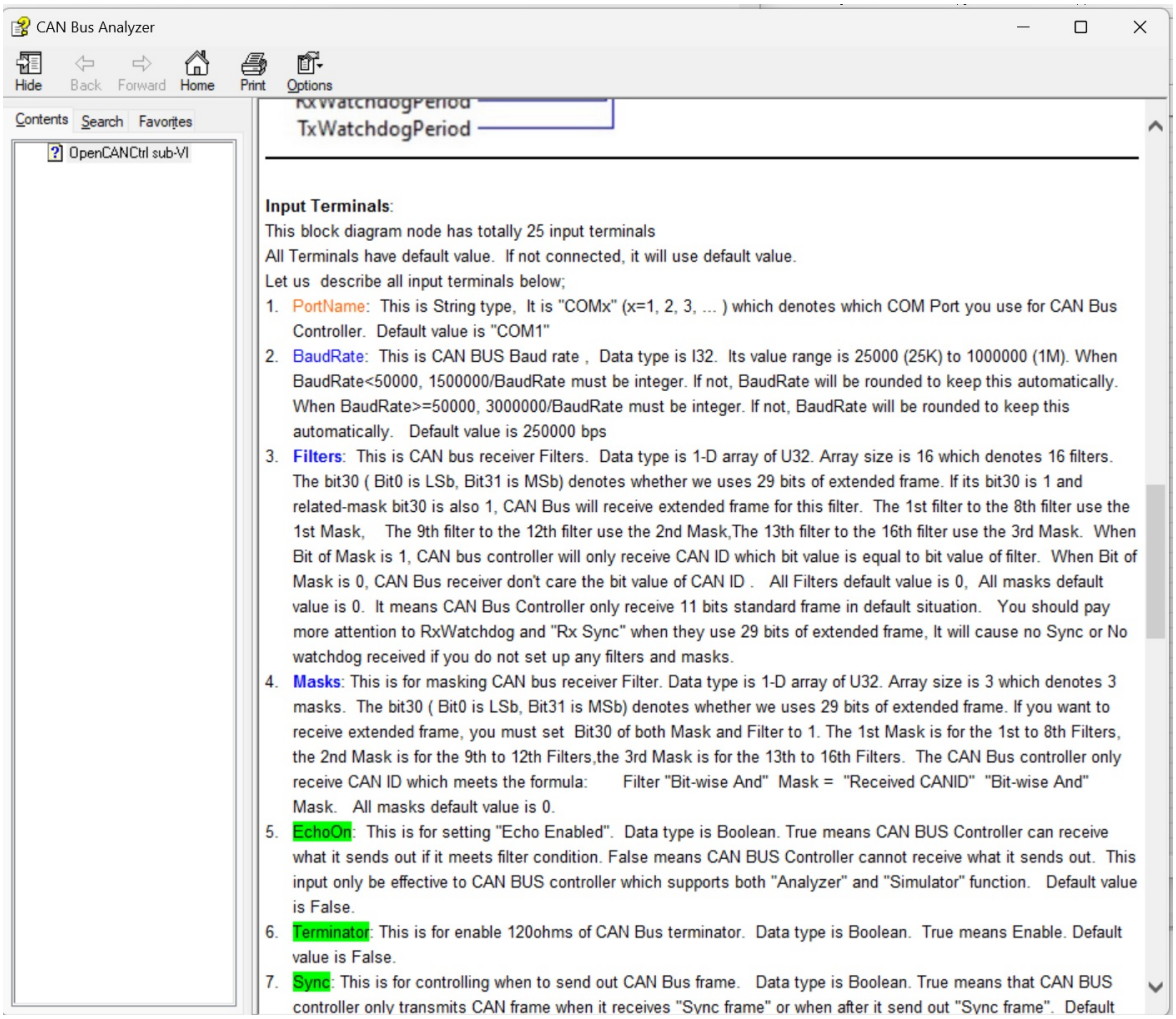


Fig.6 Detailed help

You can add many other nodes you want in this way.

3 Block Diagrams Nodes

We will introduce every block diagram node for CAN Bus controller in the following sections.

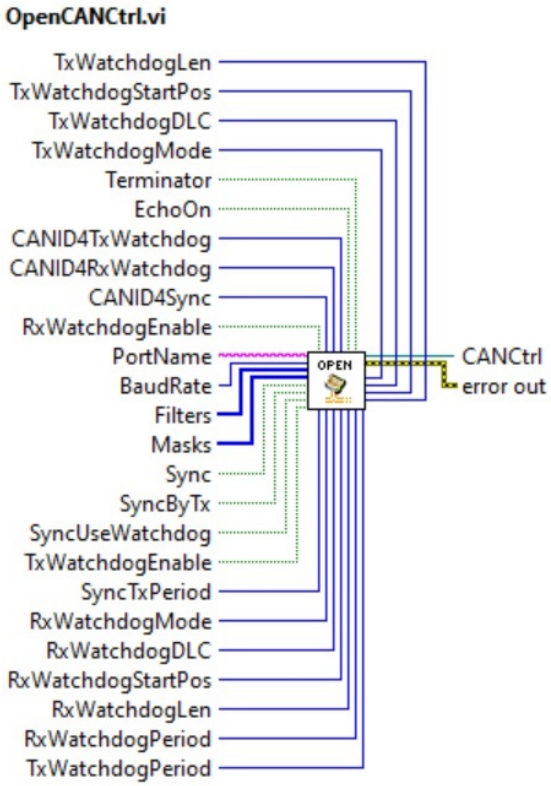
3.1 OpenCANCtrl sub-VI

This is for configuration of CAN Bus Controller.

You will set the following items below from this Block diagram node:

- COM port which used for CAN Bus Controller
- CAN Bus baud rate, and terminator enabled or not
- CAN BUS filters and masks, Sync CAN Bus frame, Watchdog.
- Enable CAN Bus and output reference for other CAN BUS blocks use.

Notes: This block diagram node only can be called once, and at last you must call closeCANCtril node to release resource when you don't use it..



Input Terminals:

This block diagram node has totally 25 input terminals

All Terminals have default value. If not connected, it will use default value.

Let us describe all input terminals below;

1. PortName: This is String type, It is "COMx" (x=1, 2, 3, ...) which denotes which COM Port you use for CAN Bus Controller. Default value is "COM1"
2. BaudRate: This is CAN BUS Baud rate , Data type is I32. Its value range is 25000 (25K) to 1000000 (1M). When BaudRate<50000, 1500000/BaudRate must be integer. If not, BaudRate will be rounded to keep this automatically. When BaudRate>=50000, 3000000/BaudRate must be integer. If not, BaudRate will be rounded to keep this automatically. Default value is 250000 bps
3. **Filters**: This is CAN bus receiver Filters. Data type is 1-D array of U32. Array size is 16 which denotes 16 filters. The bit30 (Bit0 is LSb, Bit31 is MSb) denotes whether we uses 29 bits of extended frame. If its bit30 is 1 and related-mask bit30 is also 1, CAN Bus will receive extended frame for this filter. The 1st filter to the 8th filter use the 1st Mask, The 9th filter to the 12th filter use the 2nd Mask,The 13th filter to the 16th filter use the 3rd Mask. When Bit of Mask is 1, CAN bus controller will only receive CAN ID which bit value is equal to bit value of filter. When Bit of Mask is 0, CAN Bus receiver don't care the bit value of CAN ID . All Filters default value is 0, All masks default value is 0. It means CAN Bus Controller only receive 11 bits standard frame in default situation. You should pay more attention to RxWatchdog and "Rx Sync" when they use 29 bits of extended frame, It will cause no Sync or No watchdog received if you do not set up any filters and masks.
4. **Masks**: This is for masking CAN bus receiver Filter. Data type is 1-D array of U32. Array size is 3 which denotes 3 masks. The bit30 (Bit0 is LSb, Bit31 is MSb) denotes whether we uses 29 bits of extended frame. If you want to receive extended frame, you must set Bit30 of both Mask and Filter to 1. The 1st Mask is for the 1st to 8th Filters, the 2nd Mask is for the 9th to 12th Filters,the 3rd Mask is for the 13th to 16th Filters. The CAN Bus controller only receive CAN ID which meets the formula: Filter "Bit-wise And" Mask = "Received CANID" "Bit-wise And" Mask. All masks default value is 0.
5. EchoOn: This is for setting "Echo Enabled". Data type is Boolean. True means CAN BUS Controller can receive what it sends out if it meets filter condition. False means CAN BUS Controller cannot receive what it sends out. This input only be effective to CAN BUS controller which supports both "Analyzer" and "Simulator" function. Default value is False.
6. Terminator: This is for enable 120ohms of CAN Bus terminator. Data type is Boolean. True means Enable. Default value is False.
7. Sync: This is for controlling when to send out CAN Bus frame. Data type is Boolean. True means that CAN BUS controller only transmits CAN frame when it receives "Sync frame" or when after it send out "Sync frame". Default value is False.
8. SyncByTx: This is for choosing Tx or Rx as Sync frame. Data type is Boolean. True means "Sync

frame" is transmit frame, Default value is False.

9. CANID4Sync: This is CAN ID for "Sync frame". Data type is U32. The bit30 (Bit0 is LSb, Bit31 is MSb) denotes whether we use 29 bits of extended frame. 1 is extended frame. Default value is decimal 123. Notes: If you use watchdog frame as "Sync frame", this parameter will be ignored.
10. SyncUseWatchdog: This will force Sync to use watchdog frame to replace. CANID4Sync will be ignored. Data type is Boolean. Default value is False.
11. SyncTxPeriod: This is transmitting period of "Tx Sync frame" in ms if we use Tx Sync frame. Data type is U16. Min=2, Max=65535. Default value=500
12. RxWatchdogEnable: This will enable RxWatchdog. Data type is Boolean. True means that we enable received Watchdog. when it is enabled, you can call node "getCANCommStatus" to know whether CAN BUS communication fault occurs. Default value is False.
13. CANID4RxWatchdog: This is CAN ID for "RxWatchdog frame". Data type is U32. The bit30 (Bit0 is LSb, Bit31 is MSb) denotes whether we use 29 bits of extended frame. 1 is extended frame. Default value is decimal 234. Notes: If RxWatchdogEnable is False, this parameter will be ignored.
14. RxWatchdogDLC: This is the length of RxWatchdog frame's data in bytes. Data type is U8. Default value is 8. Notes: this is not RxWatchdog data length. It is entire data frame's length in byte.
15. RxWatchdogMode: This is RxWatchdogMode. Data type is U8. Default value is 0.
16. RxWatchdogStartPos: This is RxWatchdog value's start position (1-based) in RxWatchdog data frame, Data type is U8. Default value is 1.
17. RxWatchdogLen: This is RxWatchdog value's byte quantity. Data type is U8. Default value is 1. The valid value is 1, 2, 4, 8.
18. RxWatchdogPeriod: This is RxWatchdog period. It is used for judgment of CAN BUS communication fault. Data type is U16. Min=2, Max=65535. Default value=500
19. TxWatchdogEnable: This will enable TxWatchdog. Data type is Boolean. True means that we enable TxWatchdog. when it is enabled, CAN Bus controller will send out TxWachdog frame cyclically by hardware without LabView any action. Default value is False.
20. CANID4TxWatchdog: This is CAN ID for "TxWatchdog frame". Data type is U32. The bit30 (Bit0 is LSb, Bit31 is MSb) denotes whether we use 29 bits of extended frame. 1 is extended frame. Default value is decimal 345. Notes: If TxWatchdogEnable is False, this parameter will be ignored.
21. TxWatchdogDLC: This is the length of TxWatchdog frame's data in bytes. Data type is U8. Default value is 8. Notes: this is not TxWatchdog data length. It is entire data frame's length in byte.
22. TxWatchdogMode: This is TxWatchdogMode. Data type is U8. Default value is 0.
23. TxWatchdogStartPos: This is TxWatchdog value's start position (1-based) in TxWatchdog data frame, Data type is U8. Default value is 1.
24. TxWatchdogLen: This is TxWatchdog value's byte quantity. Data type is U8. Default value is 1. The valid value is 1, 2, 4, 8.
25. TxWatchdogPeriod: This is TxWatchdog period. It is to notify hardware how often to transmit TxWatchdog frame . Data type is U16. Min=2, Max=65535. Default value=500

Output Terminals:

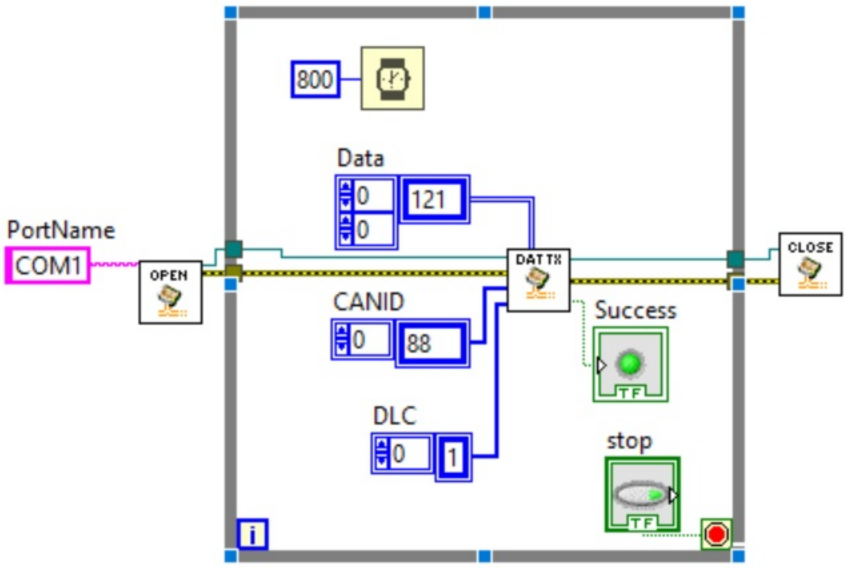
This block diagram node has totally 2 output terminals

Let us describe all output terminals below;

1. CANCtrl: This is CAN bus controller 's reference output. The other CAN bus controller's node will use it as input to know CAN bus controller.
2. error out: This contains error information. This output provides standard error out functionality

A simple example:

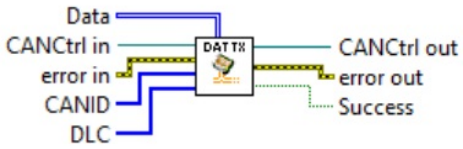
In the following example, CAN BUS Controller will transmit CAN data frame with CANID=88, DLC=1 and Data=121 once every 800ms. And if you switch stop button to true, it will exit loop and exit program



3.2 TransmitData Sub VI

This is for transmitting Data frame.

You will transmit one or multiple data frames by calling this node:



Input Terminals:

This block diagram node has totally 5 input terminals

Let us describe all input terminals below;

1. CANCtrl: This is CAN bus controller 's reference intput. It denotes CAN Bus Controller object instance.
2. error in: error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
3. **DLC**: This is 1-D array of U8. Array size denotes how many frames do you want to transmit. The element is Data byte quantity for each frame. Element value range is 0 to 8.
4. **CANID**: This is 1-D array of U32. Array size denotes how many frames do you want to transmit. So CANID size must be equal to DLC size. The element is CAN ID for each frame. The bit30 (Bit0 is LSb, Bit31 is MSb) denotes whether we use 29 bits of extended frame. 1 is extended frame.
5. **Data**: This is 2-D array of U8. The row number of array is how many frames do you want to transmit. So row quantity must be equal to DLC size. The element of i row j column denotes one data byte for data j of frame i.

Output Terminals:

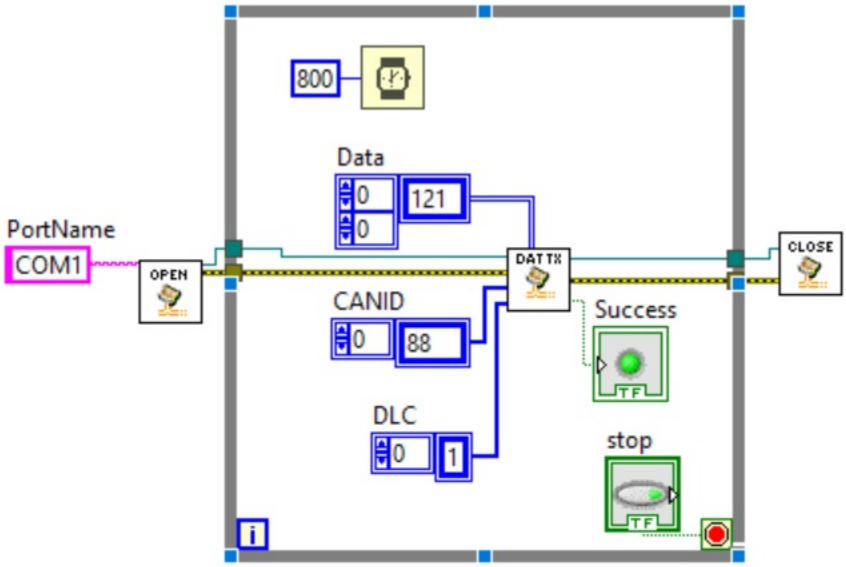
This block diagram node has totally 3 output terminals

Let us describe all output terminals below;

1. CANCtrl: This is CAN bus controller 's reference output. The other CAN bus controller's node will use it as input to know CAN bus controller.
2. error out: This contains error information. This output provides standard error out functionality
3. Success: This is for telling caller whether transmit success. Data type is Boolean. True means success.

A simple example:

In the following example, CAN BUS Controller will transmit CAN data frame with CANID=88, DLC=1 and Data=121 once every 800ms. And if you switch stop button to true, it will exit loop and exit program



3.3 TransmitRTR Sub-VI

This is for transmitting Remote frame.

You will transmit one or multiple Remote frames by calling this node:

Input Terminals:

This block diagram node has totally 3 input terminals

Let us describe all input terminals below;

1. CANCtrl: This is CAN bus controller 's reference input. It denotes CAN Bus Controller object instance.
2. error in: error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
3. **CANID**: This is 1-D array of U32. Array size denotes how many frames do you want to transmit. So CANID size must be equal to DLC size. The element is CAN ID for each frame. The bit30 (Bit0 is LSb, Bit31 is MSb) denotes whether we use 29 bits of extended frame. 1 is extended frame.

Output Terminals:

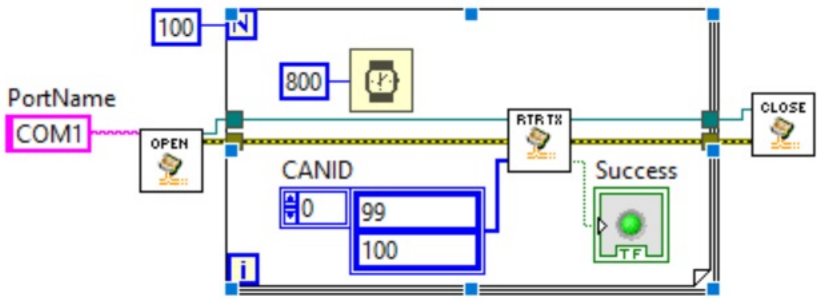
This block diagram node has totally 3 output terminals

Let us describe all output terminals below;

1. CANCtrl: This is CAN bus controller 's reference output. The other CAN bus controller's node will use it as input to know CAN bus controller.
2. error out: This contains error information. This output provides standard error out functionality
3. Success: This is for telling caller whether transmit success. Data type is Boolean. True means success.

A simple example:

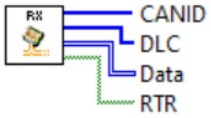
In the following example, CAN BUS Controller will transmit 2 CAN Remote frames with CANID=99 and 100 once every 800ms. And it will exit loop after 100 times (80 seconds) and exit program



3.4 ReceiveCAN Sub-VI

This is for receiving CAN Bus frame.

This is non-block node. You will receive Frame information if it gets CAN Bus frame coming (meet filter and mask) or receive Empty if no can bus frame coming or filtered out by filter and mask when calling this node:



Notes: *System uses FIFO to keep CAN BUS received frames. So even though you didn't call this ReceiveCAN Node on time, you can still get received frame information later. FIFO size is 2047 frames. If buffered content less than 2048, you won't lose data packets.*

Output Terminals:

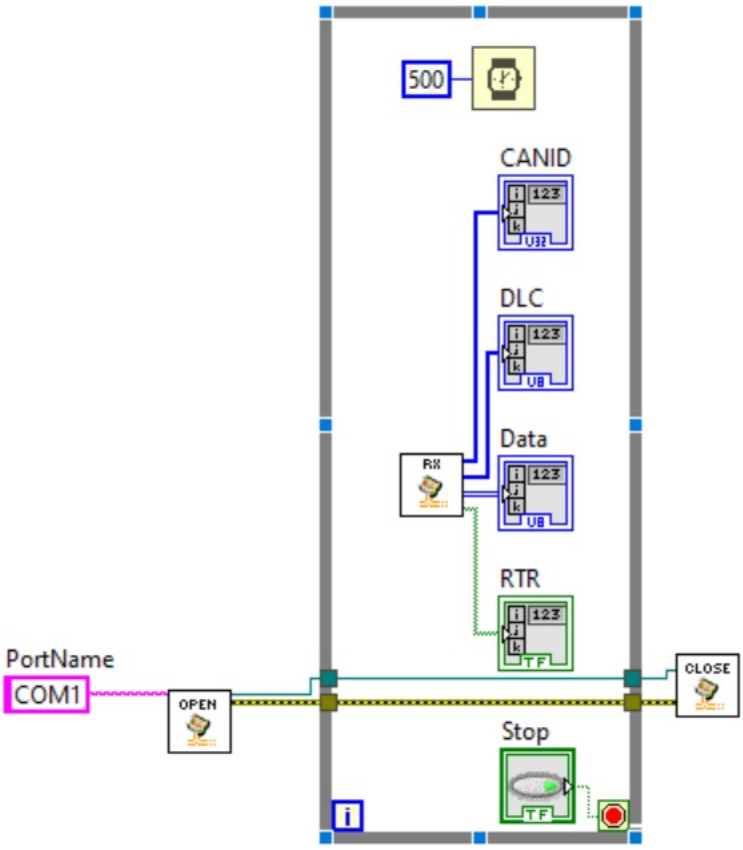
This block diagram node has totally 4 output terminals

Let us describe all output terminals below;

1. **CANID**: This is CAN ID received. Data type is 1-D array of U32. Array size denotes how many frames we received. Element of array is CAN ID we received. The bit30 (Bit0 is LSb, Bit31 is MSb) denotes whether we receive 29 bits of extended frame. 1 is extended frame. The bit29 denotes whether we receive Remote Request frame, 1 is RTR frame. If we didn't receive any frame or received frame filtered out by Filter and mask, this array will be empty. That is to say, Array size is 0. Array index denotes frame number.
2. **DLC**: This is Data byte quantity for received frame. Data type is 1-D array of U8. The valid elements are 0 to 8. Array index denotes frame number, so array size is exact that same as size of CANID.
3. **Data**: This is Data byte for received frame. Data type is 2-D array of U8. The row number of array is how many frames do you receive. So row quantity must be equal to DLC size. The element of i row j column denotes one data byte for data j of frame i.
4. RTR: This is for telling caller whether received frame is Remote frame. Data type is 1-D array of Boolean. Array index denotes frame number, so array size is exact that same as size of CANID. The element value True denotes Remote frame.

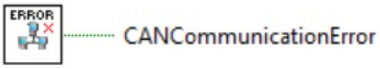
A simple example:

In the following example, CAN BUS Controller will receive CAN data frames once every 500ms. And if you switch stop button to true, it will exit loop and exit program



3.5 getCANCommStatus Sub-VI

This is to know whether CAN BUS Communication fault when RxWatchdog enabled.



Notes: This is only used when RxWatchdog enabled

Output Terminals:

This block diagram node has totally 1 output terminal

Let us describe all output terminals below;

1. CANCommunicationErrorSuccess: This is for telling caller whether CAN Bus communication fault occured. Data type is Boolean. True means Fault.

This node is very simple. We do not give example. Communication state is changeable, so you should call this node frequently.

3.6 sendTxWatchdogValue Sub-VI

This is for sending TxWatchdog value when TxWatchdog mode 0.

Hardware sends out TxWatchdog frame automatically. You only need to tell hardware what watchdog value is when mode 0.

For TxWatchdog mode 1 and 2 and 3, you don't need to tell hardware what watchdog value is because Hardware decide watchdog value by itself.

You will tell hardware what txWatchdog value is by calling this node:

Notes: This is only used when TxWatchdog enabled and TxWatchdog mode 0 is used

---

Input Terminals:

This block diagram node has totally 3 input terminals

Let us describe all input terminals below;

1. CANCtrl: This is CAN bus controller 's reference intput. It denotes CAN Bus Controller object instance.
2. error in: error in describes error conditions that occur before this node runs. This input provides standard error in functionality.
3. TxWDValue: This is TxWatchdog Value. Data type is U64. You need to set value according to TxWatchdogLen parameter (The data byte quantity of TxWatchdog Value)

Output Terminals:

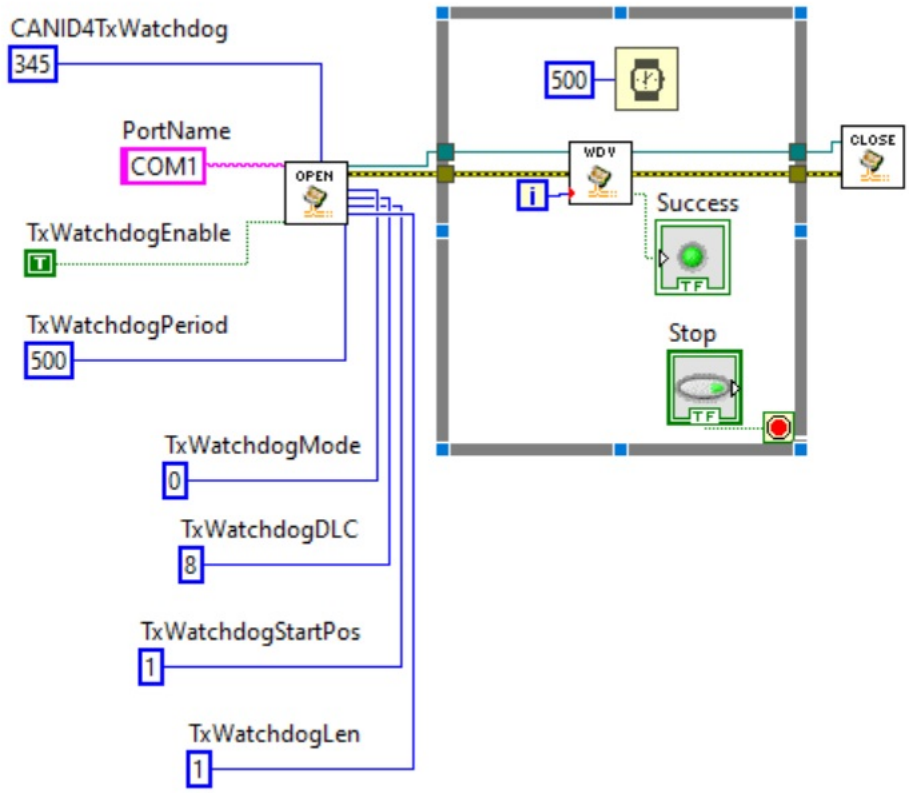
This block diagram node has totally 3 output terminals

Let us describe all output terminals below;

1. CANCtrl: This is CAN bus controller 's reference output. The other CAN bus controller's node will use it as input to know CAN bus controller.
2. error out: This contains error information. This output provides standard error out functionality
3. Success: This is for telling caller whether transmit success. Data type is Boolean. True means success.

A simple example:

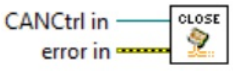
In the following example, TxWatchdog enabled and mode=0. CAN BUS Controller will send TxWatchdog Value = Loop value i once every 500ms. And if you switch stop button to true, it will exit loop and exit program



3.7 CloseCANCtrl Sub-VI

This is for close CAN Bus Controller.

You will release all resource occupied by CAN BUS Controller and disable CAN Bus:



Input Terminals:

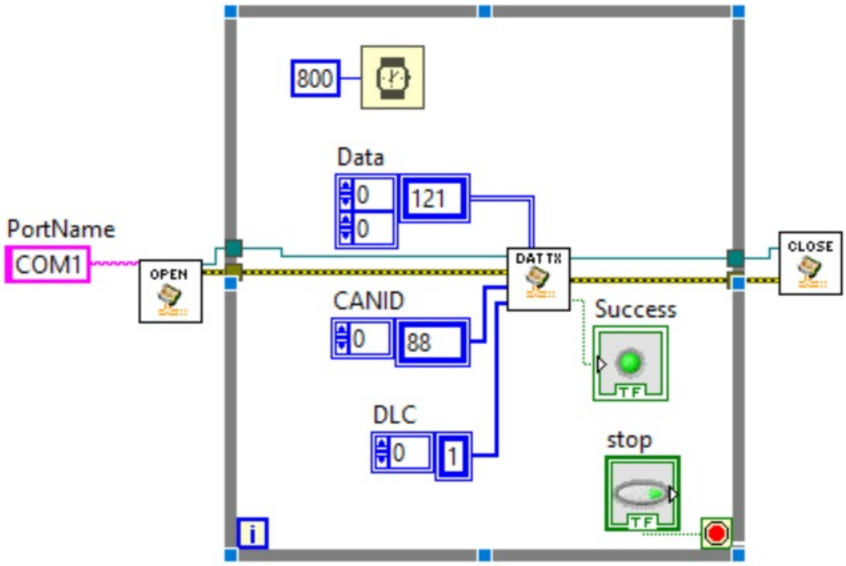
This block diagram node has totally 2 input terminals

Let us describe all input terminals below;

1. CANCtrl: This is CAN bus controller 's reference intput. It denotes CAN Bus Controller object instance.
2. error in: error in describes error conditions that occur before this node runs. This input provides standard error in functionality.

A simple example:

In the following example, CAN BUS Controller will transmit CAN data frame with CANID=88, DLC=1 and Data=121 once every 800ms. And if you switch stop button to true, it will exit loop and exit program



4 Notice

IMPORTANT NOTICE

The information in this manual is subject to change without notice.

Dafulai's products are not authorized for use as critical components in life support devices or systems. Life support devices or systems are those which are intended to support or sustain life and whose failure to perform can be reasonably expected to result in a significant injury or death to the user. Critical components are those whose failure to perform can be reasonably expected to cause failure of a life support device or system or affect its safety or effectiveness.

COPYRIGHT