MULTICAN_FD_1

MULTICAN FD data transmission

AURIX™ TC2xx Microcontroller Training
V1.0.0

Please read the Important Notice and Warnings at the end of this document

Scope of work

MULTICAN in Flexible Data-Rate mode is used to exchange data between two nodes, implemented in the same device using Loop-Back mode.

The CAN messages are sent from CAN node 0 to CAN node 1 using Loop-Back mode. Both CAN nodes are set to CAN Flexible Data-rate mode (CAN FD). After each CAN message transmission and successful reception, an interrupt is generated. Inside the interrupt service routine, the content of the received CAN message is read. In case of the successful read operation, the received data is compared to the transmitted data. If all messages are received without any error detected, the LED is turned on to confirm successful message reception.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Introduction

- The MultiCAN+ module provides a communication interface which is **fully compliant with CAN specification V2.0B (active)** and to **CAN FD ISO11898-1 DIS version 2014**, providing communication up to **1 Mbit/s in Classical CAN** (ISO 11898-1:2003(E)mode) and/or **CAN FD up to 5 Mbit/s** (dependent on frequency and nodes).

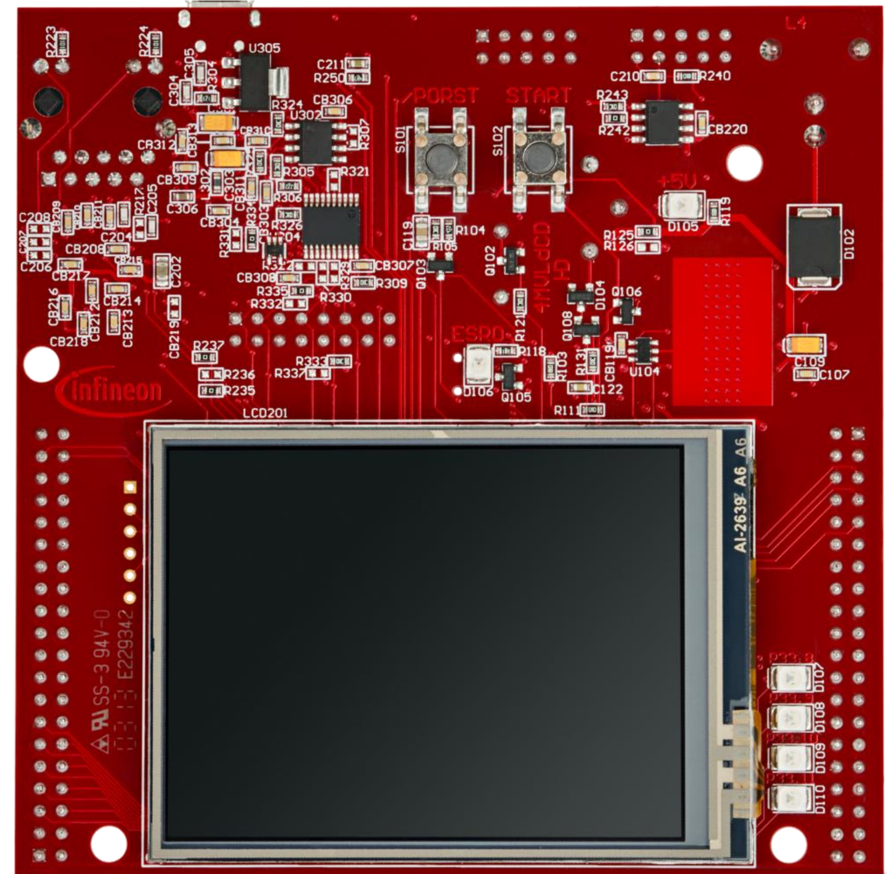
- The MultiCAN+ module consists of several **CAN nodes** (in case of AURIX™ TC29x device, 4 nodes) which are **CAN FD capable**. Each CAN node communicates over two pins (TXD and RXD). Additionally, there is an internal **Loop-Back Mode** functionality available for test purposes.

- All CAN nodes share a common set of 256 **message objects**. Each message object can be individually allocated to one of the CAN nodes. Besides serving as a **storage container for incoming and outgoing frames**, message objects can be combined to build **gateways** between the CAN nodes or to setup a **FIFO buffer**.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Hardware setup

This code example has been developed for the board KIT_AURIX_TC297_TFT_BC-Step.



Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

› This code example covers four different CAN FD use cases. The overview of the possible use cases is given in the figure below (highlighted use cases correspond to the ones where CAN node is set to CAN FD mode; the Node Control Register contains the value 1 in the bitfield FDEN **NCRx.FDEN** = 1):

FDEN	FDF	BRS	Transmit Behavior
0_B	0_B	0_B	Classical CAN Frames (ISO 11898-1)
0_B	0_B	1_B	Classical CAN Frames (ISO 11898-1)
0_B	1_B	0_B	Transmission Cancelled (i.e TXRQ bit cleared)
0_B	1_B	1_B	Transmission Cancelled (i.e TXRQ bit cleared)
1_B	0_B	0_B	Classical CAN Frames (ISO 11898-1)
1_B	0_B	1_B	Classical CAN Frames (ISO 11898-1)
1_B	1_B	0_B	Long Frame (i.e CAN FD frame with BRS = 0, whole frame transmitted with slow baudrate)
1_B	1_B	1_B	Long + Fast Frame (i.e CAN FD frame with BRS = 1, switching fast baudrate for dataphase)

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Application code can be separated into four segments:

› Initialization of the MultiCAN+ module with the accompanying node and message objects initialization, implemented in the ***initMultican()*** function
› Initialization of the port pin connected to the LED (D107 on the board). The LED is used to verify the success of a CAN message reception. This is done inside the ***initLed()*** function
› Transmission of the configured CAN messages, implemented in the ***transmitCanMessage()*** function
› Verification of the received CAN messages, implemented in the ***verifyCanMessage()*** function

An additional Interrupt Service Routine (ISR) is implemented:

› On RX interrupt, the ISR reads the received CAN message and, in case of no errors, increments the counter to indicate the number of successfully received CAN messages (realized by ***canIsrRxHandler()*** function).

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

MultiCAN+ module initialization

Initialization is performed in three phases:

- A default CAN module configuration is loaded into the configuration structure by using the function ***IfxMultican_Can_initModuleConfig()***. Afterwards, the initialization of the CAN module with the user configuration is done with the function ***IfxMultican_Can_initModule()***.

- A default CAN node configuration is loaded into the configuration structure by using the function ***IfxMultican_Can_Node_initConfig()***. Initialization of the CAN nodes 0 and 1 with the different CAN node ID values and definition of Loop-Back Mode usage for both nodes is done with the function ***IfxMultican_Can_Node_init()***. Additionally, all CAN FD related configuration options are shared between both nodes.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

MultiCAN+ module initialization

- A default CAN message object configuration is loaded into the configuration structure by using the function ***IfxMultican_Can_MsgObj_initConfig()***.
  The initialization of the CAN message objects with different configurations is done by the ***IfxMultican_Can_MsgObj_init()*** function.

All functions used for the MultiCAN+ module initialization are declared in the iLLD header ***IfxMultican_Can.h***.

Due to the multiple message objects used in this application use case and the complexity of their configuration, the following slides cover the configuration of each message object.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Source standard message objects configuration (MO0/1/2/3):

- ***canMsgObjConfig.frame*** = ***IfxMultican_Frame_transmit*** – defines the message object as a transmit message object (common setting for all source message objects)
- ***canMsgObjConfig.control.matchingId*** = ***TRUE*** – defines acceptance of the frame with only matching Identifier Extension (IDE) (common setting for all source message objects)

- ***canMsgObjConfig.msgObjId*** = ***0/1/2/3**** – defines the message object ID
- ***canMsgObjConfig.messageId*** = ***< >**** – defines the CAN message ID used during arbitration phase
- ***canMsgObjConfig.control.extendedFrame*** = ***TRUE/FALSE**** – defines the standard or extended frame to be used
- ***canMsgObjConfig.control.topMsgObjId*** = ***64/66/68/70**** – defines the message object that holds data bytes 8 to 35 (top message)
- ***canMsgObjConfig.control.bottomMsgObjId*** = ***65/67/69/71**** – defines the message object that holds data bytes 36 to 63 (bottom message)
- ***canMsgObjConfig.control.messageLen*** = ***8/8/32/64 bytes**** – defines the length of the transmitted data
- ***canMsgObjConfig.control.fastBitRate*** = ***TRUE/FALSE**** – defines the usage of bit rate switching

* depends on the exact use case covered in the slide 5

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Destination standard message objects configuration (MO10/11/12/13):

- ***canMsgObjConfig.frame*** = ***IfxMultican_Frame_receive*** – defines the message object as a receive message object (common setting for all destination message objects)
- ***canMsgObjConfig.control.matchingId*** = ***TRUE*** – defines acceptance of the frame with only matching Identifier Extension (IDE) (common setting for all destination message objects)
- ***canMsgObjConfig.rxInterrupt.enabled*** = ***TRUE*** – enables interrupt generation in case of CAN message reception (common setting for all destination message objects)
- ***canMsgObjConfig.rxInterrupt.srcId*** = ***IfxMultican_SrcId_1*** – defines interrupt node pointer to be used (all destination message objects will share the SAME node)

- ***canMsgObjConfig.msgObjId*** = ***10/11/12/13**** – defines the message object ID
- ***canMsgObjConfig.messageId*** = ***< >**** – defines the CAN message ID used during arbitration phase
- ***canMsgObjConfig.control.extendedFrame*** = ***TRUE/FALSE**** – defines the standard or extended frame to be used
- ***canMsgObjConfig.control.topMsgObjId*** = ***128/130/132/134**** – defines the message object that holds data bytes 8 to 35 (top message)
- ***canMsgObjConfig.control.bottomMsgObjId*** = ***129/131/133/135**** – defines the message object that holds data bytes 36 to 63 (bottom message)
- ***canMsgObjConfig.control.messageLen*** = ***8/8/32/64 bytes**** – defines the length of the received data

* depends on the exact use case covered in the slide 5

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Initialization of a pin connected to the LED

An LED is used to verify the success of a CAN message reception. Before using the LED, the port pin to which the LED is connected must be configured.

- First step is to set the port pin to level “HIGH”; this keeps the LED turned off as a default state (***IfxPort_setPinHigh()*** function).
- Second step is to set the port pin to push-pull output mode with the ***IfxPort_setPinModeOutput()*** function.
- Finally, the pad driver strength is defined through the function ***IfxPort_setPinPadDriver()***.

All functions are declared in the iLLD header ***IfxPort.h***.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Transmission of CAN messages

Before a CAN message is transmitted, two CAN messages (TX and RX) need to be initialized. The TX message data content (data content that is transmitted) is initialized with the combination of current data payload byte and current CAN message value, using the following format:

bit	7	6	5	4	3	2	1	0
data content	***g_currentCan Message*** range: 0 - 3		***currentDataPayloadByte*** range: 0 - 63					

The RX message (message where the received CAN message is stored) is initialized with invalid ID, data, length, and fast bitrate value (after successful CAN transmission the values are replaced with the valid content). Additionally, both the TX and RX message data content need to be invalidated. No additional CAN message is transmitted until the received data has been read by the interrupt service routine.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Transmission of CAN messages

- Initialization of both TX and RX messages is done by using ***IfxMultican_Message_longFrameInit()*** (both standard and long frame messages are initialized with the mentioned function).
- Invalidation of both TX and RX message data content by using the ***memset()*** function.
- A CAN message is transmitted by using either ***IfxMultican_Can_MsgObj_sendMessage()*** or ***IfxMultican_Can_MsgObj_sendLongFrame()*** function (depending on the size of the CAN message to be transmitted). A CAN message is continuously transmitted as long as the returned status is ***IfxMultican_Status_notSentBusy*** (this status occurs if there is a pending transmit request).

The function ***IfxMultican_Message_longFrameInit()*** is declared in the iLLD header ***IfxMultican.h*** while the ***IfxMultican_Can_MsgObj_sendMessage()*** function is declared in the iLLD header ***IfxMultican_Can.h***. The function ***memset()*** is declared in the standard C library header ***string.h***.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Verification of CAN messages

After successful reception of each CAN message, several checks are performed:

1. Message ID check (check that the received message ID matches the transmitted one) Verifies that both standard and extended IDs have been received.
2. Message length check (check that the received message length matches the transmitted one). The check is covering both classical CAN and CAN FD frame sizes.
3. Fast bitrate bit value check (check that the received fast bitrate bit value matches the transmitted one). Both bit values (***MOFCRn.BRS*** = 0/1) are covered.
4. Expected valid data check (check that the received data matches with the expected one). Both classical CAN and CAN FD data content is covered.
5. Invalid data check (check that the invalid data has not been modified with the CAN transmission).

If no error has been observed, the ***g_status*** variable holds ***CanCommunicationStatus_Success*** value upon returning from the ***verifyCanMessage()*** function.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Implementation

Interrupt Service Routine (ISR)

An ISR is triggered by the successful CAN message reception.

- The RX ISR reads the received CAN message with either ***IfxMultican_Can_MsgObj_readMessage()*** or ***IfxMultican_MsgObj_readLongFrame()*** function (depending on the size of the CAN message to be read). Due to the fact that we want to achieve common implementation to verify the CAN messages independent of their size, the ***memcpy()*** function is used. The ***memcpy()*** function copies the data from ***g_multican.rxMsg.data*** to ***g_multican.rxData*** array in case a classical CAN message has been received. Based on the return status, the ***g_status*** variable can be set to the erroneous value. If a non-erroneous return status is present, then a global variable ***g_isrRxCount*** is incremented. This variable is used as a counter to indicate the number of successfully received CAN messages.

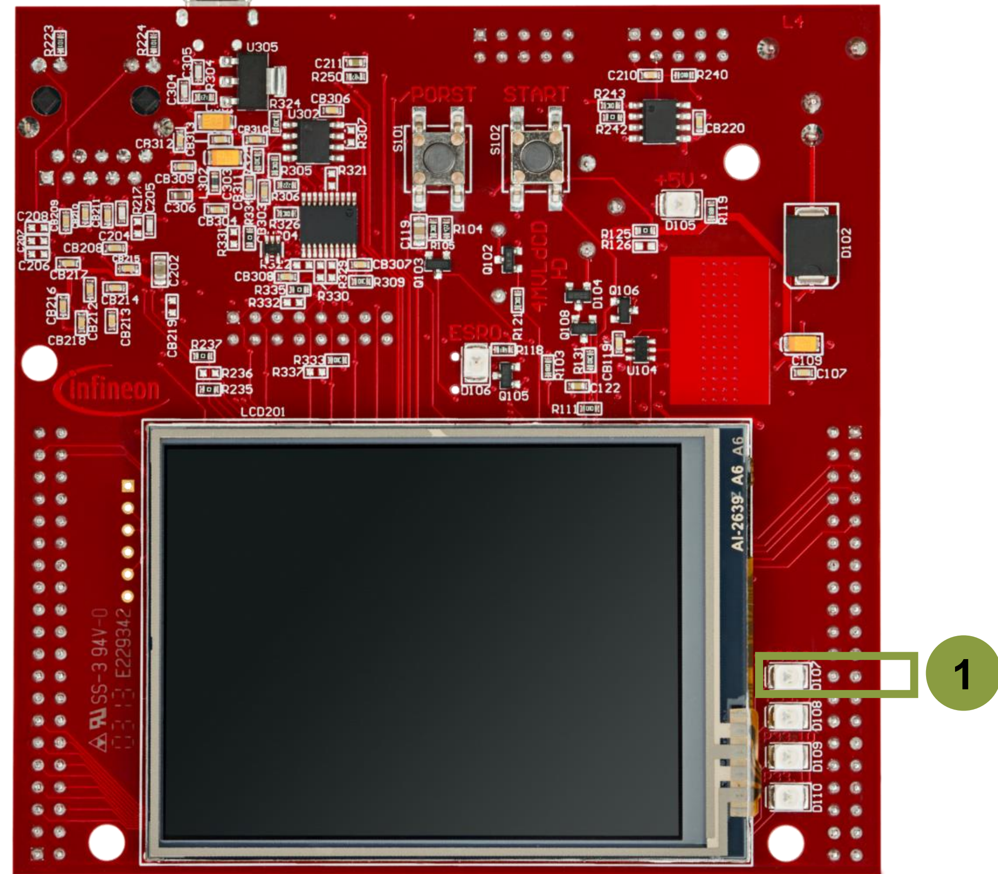
Both ***IfxMultican_Can_MsgObj_readMessage()*** and ***IfxMultican_MsgObj_readLongFrame()*** functions are declared in the iLLD header ***IfxMultican_Can.h*** while the function ***memcpy()*** is declared in the standard C library header ***string.h***.

Copyright © Infineon Technologies AG 2020. All rights reserved.

Run and Test

After code compilation and flashing the device, observe the following behavior:

- Check that the LED (1) is turned on (all CAN messages have been successfully received and all checks have been passed).



Copyright © Infineon Technologies AG 2020. All rights reserved.

References

- AURIX™ Development Studio is available online:
- https://www.infineon.com/aurixdevelopmentstudio
- Use the „*Import...*“ function to get access to more code examples.

- More code examples can be found on the GIT repository:
- https://github.com/Infineon/AURIX_code_examples

- For additional trainings, visit our webpage:
- https://www.infineon.com/aurix-expert-training

- For questions and support, use the AURIX™ Forum:
- https://www.infineonforums.com/forums/13-Aurix-Forum

Copyright © Infineon Technologies AG 2020. All rights reserved.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2020-01
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2020 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
MULTICAN_FD_1

IMPORTANT NOTICE
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.