Technical Report

CAN-bus Controller with AXI4-lite Interface

http://isca.hmu.gr/wp-content/uploads/2020/02/can_report.pdf

List of Authors

Othon Tomoutzoglou

List of Supervisors

Dr. George Kornaros

Document version

v1.0

HELLENIC MEDITERRANEAN UNIVERSITY

Department Of Electrical & Computer Engineering

Intelligent Systems & Computer Architecture Lab

February 7, 2020

Abstract

CAN-bus controllers hold a dominant position basically in automotive and other application areas, e.g., elevators, agricultural equipment, industrial automation and mechanical control. The first official protocol release was back in 1986 and over the years CAN protocol evolved into versions CAN 2.0A,B (1991) and CAN FD 1.0 (2011). This protocol is mostly preferred due to its characteristics, i.e., shared bus communication (reduced wiring and topology complexity), low cost, error correction, , robustness, flexibility (message-based), low cost, fault tolerance in electrically noisy environments etc. This work concerns CAN 2.0 version of the protocol and hopes that will be helpful to any interested community by providing the necessary tools in order to start validating, testing, prototyping etc.

Disclaimer

The CAN protocol is developed by Robert Bosch GmbH and protected by patents. Anybody who wants to implement this CAN IP core on silicon has to obtain a CAN protocol license from Bosch.

Contents

List of Figures

List of Acronyms

API	**A**pplication **P**rogramming **I**nterface
ASIC	**A**pplication-**S**pecific **I**ntegrated **C**ircuit
AXI	**A**dvanced e**X**tensible **I**nterface
CAN	**C**ontroller **A**rea **N**etwork
EU	**E**uropean **U**nion
FPGA	**F**ield-**P**rogrammable **G**ate **A**rray
HW	**H**ard**W**are
IRQ	**I**nterrupt **R**e**Q**uest
ISCA	**I**ntelligent **S**ystems & **C**omputer **A**rchitecture lab
IP	**I**ntellectual **P**roperty
MCU	**M**icro**C**ontroller **U**nit
MSB	**M**ost **S**ignificant **B**it
RX	**R**eceive
SJW	**S**ynchronization **J**ump **W**idth
SW	**S**oft**W**are
TAPPS	**T**rusted **APPS** for open CPSs
TSEG	**T**ime **SEG**ment
TX	**T**ransmit
HDL	**H**ardware **D**escription **L**anguage

Chapter 1

Introduction

A Controller Area Network (CAN bus) is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each others' applications without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on copper, but can also be used in many other contexts. For each device the data in a packet is transmitted sequentially but in such a way that if more than one device transmits at the same time the highest priority device is able to continue while the others back off. Packets are received by all devices, including by the transmitting device [1].

This document provides information about CAN/CAN-bus, CAN protocol and the resources, in terms of hardware (synthesizable HDL code), software (C code) and documentation that enable the evaluation and testing of CAN-bus controllers. Development boards equipped with FPGA and/or micro-controller and/or processor, which offers communication links between them, are suitable for porting and testing. The provided sources has been tested over development boards equipped 32-bit ARM processors/micro-controllers, i.e., Zedboard (Xilinx Zynq-7000 + Cortex-A9) and SEcube (Lattice MachXO2 + Cortex-M4).

The implementation was partially funded by TAPPS a EU-funded project in Horizon2020 framework and ISCA-lab, to fulfill project's and lab's goals; A related to TAPPS publication, with additional HW and SW security extentions based on the requirements, can be found at [2]. A preprint of this publication can be found at ISCA-lab's site http://isca.hmu.gr.

Figure 1.1 depicts one of the available demo videos related to TAPPS, with additional HW and SW extentions based on the requirements. Can be found at

ISCA-lab's Youtube channel at
https://www.youtube.com/channel/UCrP1FrI0Lc0TXkrVJ9Qgu0w/videos. The CAN controller is implemented in SEcube's FPGA, which is the top left device, as shown below at figure 1.1.

Figure 1.1 Secure update demo.

1.1 CAN-bus licenses from Bosch and aspects

As stated at [3], *Bosch offers two CAN protocol licenses. The CAN Protocol License for ASIC-manufacturerslicense fee is a lump sum payment of 10,200 EUR for the first 100,000 CAN products. For volumes greater than 100,000, a royalty of 2% of the net sales price of the CAN product is charged, with a maximum of 0.102 EUR per each CAN product.Bosch also offers a CAN Protocol License for FPGA mass programming, charging a license fee lump sum payment of 2,500 EUR for the first 10,000 programmed FPGAs. For volumes greater than 10,000 units, a separate CAN Protocol License for ASIC-manufacturers agreement has to be concluded with Bosch.Both the CAN Protocol License for FPGA mass programming and the CAN Protocol License for ASIC-manufacturersare standard license agreements from Bosch that do not require detailed negotiations. Customers going through the process and obtaining licenses from Bosch have given positive reports.*

Aspects of CAN have been codified in various ISO standards.

- *ISO 11898-5:2007 specifies the CAN physical layer for transmission rates up to 1 Mbps.*
- *ISO 11898-1:2003 specifies the CAN data link layer (DLL) and physical signaling.*
- *ISO 15765-2:2004 specifiestheCAN network protocol.*
- *ISO 15765-4:2005 specifies a CAN variant specifically for emissions-related systems based on ISO 15765-2, ISO 11898-1, and ISO 11898-2, with the addition of various constraints.*

CAN was targeted originally for automotive applications, but adoption of CAN has spread to other markets. CAN is now used in vehicles from trains to ships, as well as in military, aerospace, and industrial control applications.

Chapter 2

Design Information

Source files related to this work can be found online at **https://github.com/otto-tom/can_axi4lite**

2.1 Hardware

The following, already implemented, cores have been used in order to achieve this project's goals:

- CAN Protocol Controller, authored by Igor Mohor (`igorm@opencores.org`)
- Simple AXI4-Lite bridges for IPbus and Wishbone, authored by Wojciech M. Zabolotny (`wzab@ise.pw.edu.pl`)

CAN Protocol Controller

The CAN protocol controller, which is described in verilog, supports the following features [4]:

- Non-Destructive bit-wise arbitration (CSMA/CA)
- Message Based Addressing/Filtering
- Broadcast Communication
- 1 Mbit/Sec Operation
- WISHBONE SoC interface
- 8051 interface
- SJA1000 (Philips) compatible interface.

The basic logic blocks of the CAN protocol controller are depicted in figure 2.1.

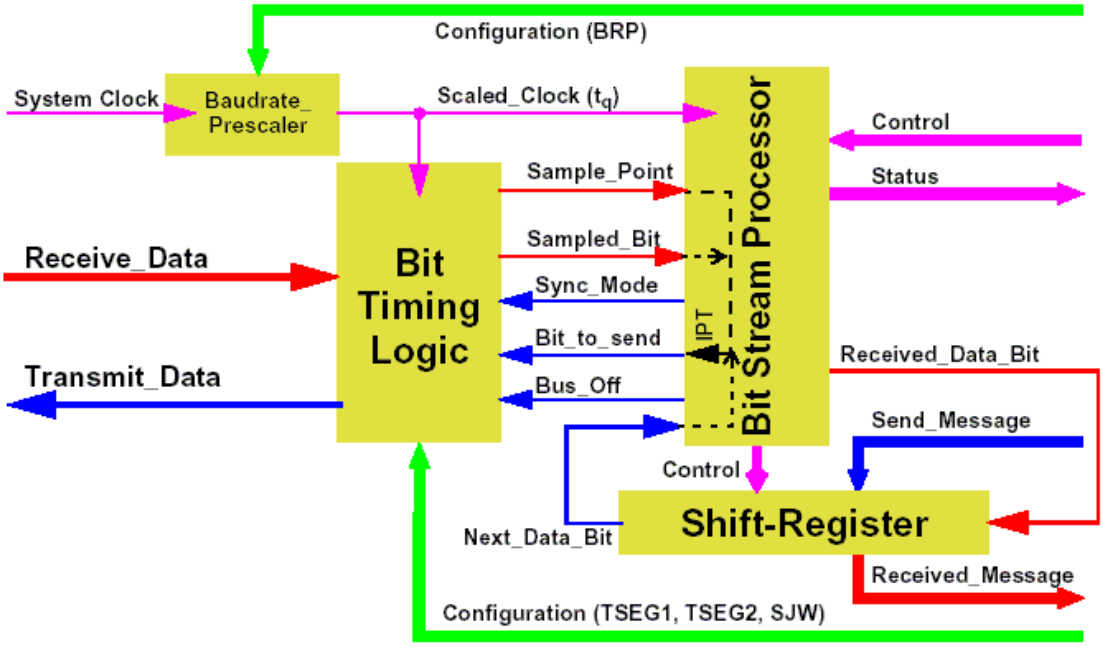


Figure 2.1 CAN protocol controller block diagram.
Image source https://opencores.org/projects/can

Simple AXI4-Lite bridges for IPbus and Wishbone

As stated at [5], this project provides very simple bridges from AXI4-Lite interface to IPbus and Wishbone buses. The design has been tested on Xilinx Zynq (Z-Turn board with xc7z020 chip) and Altera Cyclone V (DE0 NANO SoC board with 5CSEMA4U23C6). The complete demo designs based on those bridges for Z-Turn board, together with scripts for building the Vivado project may be found at https://github.com/wzabvextproj in version_2 directory. Implementation of Wishbone bridge is very limited. Currently it supports only "Classic Standard" mode.

Contribution

The above mentioned implementations have been utilized in order to provide an AXI-4 compatible CAN protocol controller. The following additions have been made in order to

- Patch files for the CAN controller that enables 8-bit addressing over a 32-bit addressable environment, provide access to extra added or already implemented registers, enriching interaction with the CAN controller; prevent faulty interrupt generation under certain circumstances etc. More informa-

tion about the modifications made can be found in the header of the patched verilog sources
- Patch file for the AXI lite to WishBone adapter, to properly communicate with CAN controller's WishBone IF.
- A top module that wraps CAN controller and axi4-lite to wishbone adapter (axi_can.v)
- Test-bench sources (axi_can_tb.v)

Testbench information

- Period T = 10ns
- Two CAN controllers instances
  1. Operating in 2B (extended) mode with extended CAN frame format enabled
  2. Baud rate: 1000 kbps
  3. Filter (drop) all the CAN frames with header ID[28] equal to zero
  4. Single filtering
  5. Interrupt mode activated only for receiving CAN frames (active high)

During the test-bench two CAN controllers are instantiated; the first instance is ordered to send two CAN frames, where the first one's header MSB is '0', while the other one's is '1'; consecutively it is ordered to the second instance to read a CAN frame (IRQ mode). Although the second instance acknowledges both the frames (physical-layer), only one of the frames is forwarded to upper-levels (i.e., two frames, one RX interrupt), as the one with header's MSB equal to '0' is dropped by the filter's rules, as depicted in figure 2.2.

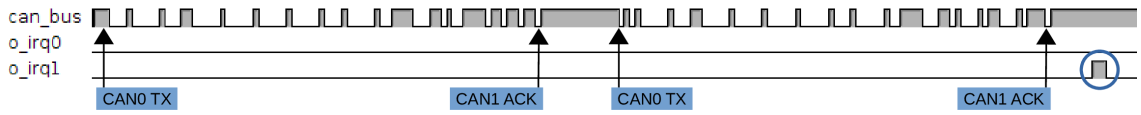


Figure 2.2 Expected waveform (only CAN bus & IRQs).

BAUD rate formula

$$Baud\ Rate = \frac{1}{(BPR * clk\_period) * (1 + TSEG1 + TSEG2))} \tag{2.1}$$

NOTE 0

BRP (Baud Rate Prescaler), TSEG1 (Time SEGment), TSEG2 and SJW (Synchronization Jump Width)are the final values that CAN controller's Bit Time Logic uses, not the ones set through the API. The following list gives the mapping from the API given to the actual value that HW perceives:

$$\begin{aligned} BRP &\mapsto 2 * (BRP + 1) \\ TSEG1 &\mapsto TSEG1 + 1 \\ TSEG2 &\mapsto TSEG2 + 1 \\ SJW &\mapsto SJW + 1 \end{aligned} \tag{2.2}$$

NOTE 1

The proper constraints should applied for on TX and RX pins based on the CAN transceiver and the development board's characteristics, in terms of voltage, ampere, hysteresis, slew rates etc.

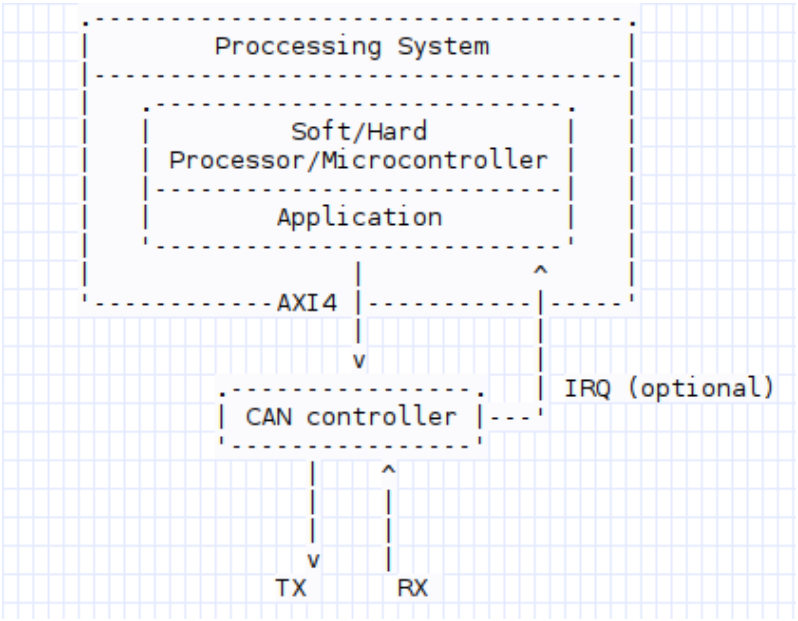


Figure 2.3 Test design abstract diagram.

2.2 Software

The implemented software gives the ability to a programmer to communicate with the CAN controller, i.e., initialization, setup, transmit(TX) and receive(RX) CAN frames etc., alongside a usage example contained in *main.c*

NOTE 2

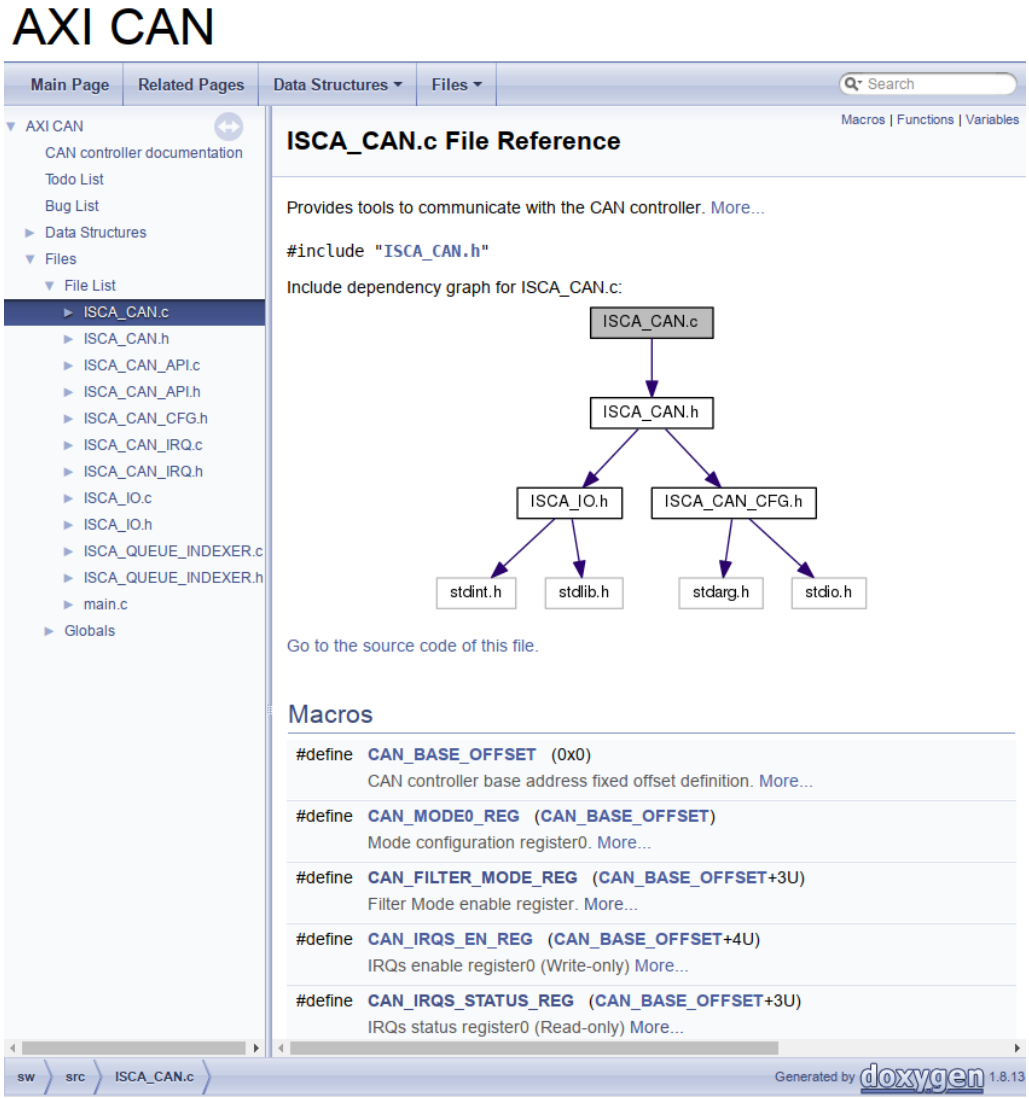
> Some of the functionality that the code provides, i.e. CAN-2A/CAN-2B, POLL/INTERRUPT modes, cannot be set at run-time, only at compile time; this can be achieved either by modifying *ISCA_CAN_CFG.h* or by passing to the compiler the appropriate options, e.g., *gcc ... -DCAN_MODE=1*. There follows all the available options.

For more detailed information about the provided software, file dependencies etc. you are encouraged to modify to your needs or taste and compile the provided Doxygen file and gain access to the provided software's documentation (installation of Doxygen is required. Tested over version 1.8.13). A pre-compiled version based on the Doxygen file HTML output exists zipped under `drax/doc/SW/`; ISCA_CAN_CFG.h definitions during compilation were the same as the current ones. Extract `drax/doc/SW/html.zip` and open *html/index.html* with your web browser; guide yourself through the code documentation.

NOTE 3

> If it happens to have problem with the relative paths defined in Doxyfile, please consider updating them manually.

Doxygen generated code documentation sample output:



Chapter 3

Porting Information

3.1 Hardware

Follow the next steps in order to produce a functional HW design:

- Clone the provided repository
- Download the sources for the **CAN Protocol Controller** project
- Copy all files from `can/trunk/rtl/verilog/` to `hw_srcs/rtl/` folder
- Copy all files from `can/trunk/bench/verilog/`, except *can_testbench.v*, to `hw_srcs/bench/` folder
- Download the sources for the **Simple AXI4-Lite bridges for IPbus and Wishbone** project
- Copy `ax4lbr/trunk/rtl/axil2wb` to the `hw_srcs/rtl/` folder
- Apply all the patch files found under `hw_patches/` folder
  - Example:
    ```
    patch hw_srcs/rtl/axil2wb.vhd <
    hw_patches/axil2wb.patch
    ```

NOTE 4

> All patch files were created using **GNU diffutils'** version 3.5 *diff* command.

After finishing the steps above you may use your EDA tool (synthesizer should support mixed VHDL-Verilog), in order to produce the appropriate programming file for your device, by importing all the files from *hw_srcs/rtl/* and *hw_srcs/bench/* folder. The following list shows the RTL and bench sources in the expected hier-

archical order:

Follow your EDA tool work flow to properly connect, synthesize and implement the design based on the selected development board; program the FPGA.

3.2 Software

Use your IDE or manually compile the code provided under *sw/*, after taking into consideration the appropriate code modification you should make in order to port the code according to your development board specifications; consulting to SW documentation might be necessary , by opening *drax/doc/SW/html/index.html* with a web browser.

Appendices

Appendix A

CAN controller documentation

Appendix generated by Doxygen and subsequently properly amended to be included in this document.

Provides information about the developted software in order to properly manage and control the CAN controller.

A.1 Todo List

File ISCA_CAN.c

Implemented filtering modes:

- Extended mode. ID match for standard format (11-bit ID). Using double filter.
- Extended mode. ID match for extended format (29-bit ID). Using double filter.

Global isca_can_receive_frame (can_ctrl_s ∗can_ctrl, can_frame_s ∗rx_↩ frame, uint8_t req_type)
Sleep while waiting for a frame to arrive

Notify user if during CAN 2B mode and filtering is set to basic, an extended frame is has been received or vice versa

Global isca_can_receive_frame (can_ctrl_s ∗can_ctrl, can_frame_s ∗rx_↩ frame, uint8_t req_type)
Sleep while waiting for a frame to arrive

Notify user if during CAN 2B mode and filtering is set to basic, an extended frame is has been received or vice versa

Global isca_can_transmit_frame (can_ctrl_s ∗can_ctrl, can_frame_s ∗tx_↩ frame, uint8_t req_type)
Sleep while TX requested and controller is busy

File ISCA_IO.c

Implement read/write for more data types

File ISCA_QUEUE_INDEXER.c

Implement lock mechanisms for multi-threaded environments

A.2 Bug List

Global isca_queue_rd_ptr (uint8_t q_index, uint8_t ∗queue_rd_ptr)

When asking from RX queue with isca_queue_rd_ptr the library advances the read pointer; it is possible that the slot (read pointer) might be overwritten by a newly arrived frame.

A.3 Data Structure Index

A.4 Data Structures

Here are the data structures with brief descriptions:

A.5 File Index

A.6 File List

Here is a list of all files with brief descriptions:

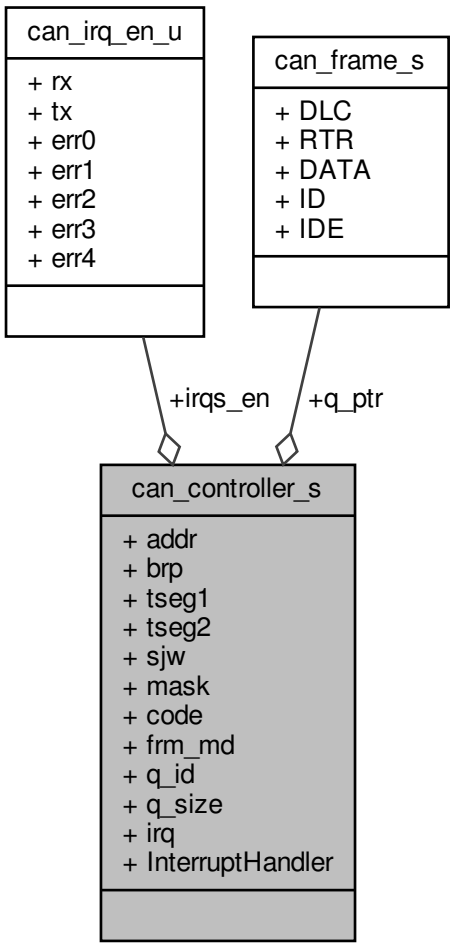
A.7 Data Structure Documentation

A.8 can_controller_s Struct Reference

CAN controller description structure.

`#include <ISCA_CAN.h>`

Collaboration diagram for can_controller_s:



Data Fields

- size_t addr
- uint8_t brp
- uint8_t tseg1
- uint8_t tseg2
- uint8_t sjw
- uint32_t mask
- uint32_t code
- can_frame_s ∗ q_ptr
- uint8_t frm_md
- int8_t q_id
- uint8_t q_size
- uint8_t irq
- irq_en irqs_en
- void(∗ InterruptHandler)(void ∗)

A.8.1 Detailed Description

CAN controller description structure.

Definition at line 119 of file ISCA_CAN.h.

A.8.2 Field Documentation

A.8.2.1 addr

`size_t can_controller_s::addr`

Controller's physical address

Definition at line 120 of file ISCA_CAN.h.

Referenced by config_can(), isca_can_ack_irq_generic(), isca_can_init(), IS↩ CA_CAN_IntrHandler(), isca_can_receive_frame(), isca_can_receive_pkt_irq(), isca_can_set_filter(), isca_can_switch_mode(), and isca_can_transmit_frame().

A.8.2.2 brp

`uint8_t can_controller_s::brp`

Baud rate prescalar

Definition at line 121 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.8.2.3 tseg1

`uint8_t can_controller_s::tseg1`

Timming segment 1

Definition at line 122 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.8.2.4 tseg2

`uint8_t can_controller_s::tseg2`

Timming segment 2

Definition at line 123 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.8.2.5 sjw

`uint8_t can_controller_s::sjw`

Synchronization jump width

Definition at line 124 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.8.2.6 mask

`uint32_t can_controller_s::mask`

Filter mask

Definition at line 125 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_set_filter().

A.8.2.7 code

`uint32_t can_controller_s::code`

Filter code

Definition at line 126 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_set_filter().

A.8.2.8 q_ptr

```
can_frame_s* can_controller_s::q_ptr
```

RX queue pointer

Definition at line 127 of file ISCA_CAN.h.

Referenced by isca_can_receive_pkt_irq(), lbr_isca_can_init(), and lbr_isca_↩ can_receive_pkt().

A.8.2.9 frm_md

```
uint8_t can_controller_s::frm_md
```

Frame mode: CAN_FRAME_EXT / CAN_FRAME_STD

Definition at line 128 of file ISCA_CAN.h.

Referenced by config_can(), isca_can_receive_frame(), and isca_can_set_↩ filter().

A.8.2.10 q_id

```
int8_t can_controller_s::q_id
```

RX queue identifier

Definition at line 129 of file ISCA_CAN.h.

Referenced by isca_can_receive_pkt_irq(), lbr_isca_can_init(), lbr_isca_can_↩ start(), and lbr_isca_can_stop().

A.8.2.11 q_size

`uint8_t can_controller_s::q_size`

RX queue size in frames

Definition at line 130 of file ISCA_CAN.h.

Referenced by lbr_isca_can_init(), and lbr_isca_can_start().

A.8.2.12 irq

`uint8_t can_controller_s::irq`

IRQ mode, CAN_IRQ_ON or CAN_IRQ_OFF

Definition at line 131 of file ISCA_CAN.h.

Referenced by config_can(), isca_can_init(), lbr_isca_can_init(), and lbr_isca_↩ can_receive_pkt().

A.8.2.13 irqs_en

`irq_en can_controller_s::irqs_en`

IRQs enable union

Definition at line 132 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.8.2.14 InterruptHandler

```
void(* can_controller_s::InterruptHandler) (void *)
```

Interupt callback pointer

Definition at line 133 of file ISCA_CAN.h.

Referenced by lbr_isca_can_init().

The documentation for this struct was generated from the following file:

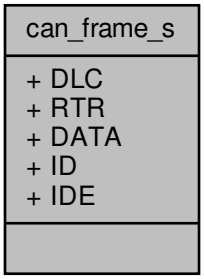
- ISCA_CAN.h

A.9 can_frame_s Struct Reference

Extended mode CAN frame description structure.

```
#include <ISCA_CAN.h>
```

Collaboration diagram for can_frame_s:



Data Fields

- uint16_t DLC
- uint16_t RTR
- uint8_t DATA [8]
- uint32_t ID
- uint32_t IDE

A.9.1 Detailed Description

Extended mode CAN frame description structure.

Definition at line 67 of file ISCA_CAN.h.

A.9.2 Field Documentation

A.9.2.1 DLC

`uint16_t can_frame_s::DLC`

Frame data length, multiple of byte

Definition at line 68 of file ISCA_CAN.h.

Referenced by can_example(), and isca_can_transmit_frame().

A.9.2.2 RTR

`uint16_t can_frame_s::RTR`

1: Remote frame request

Definition at line 69 of file ISCA_CAN.h.

Referenced by isca_can_transmit_frame().

A.9.2.3 DATA

`uint8_t can_frame_s::DATA[8]`

Frame data, i.e., payload

Definition at line 70 of file ISCA_CAN.h.

Referenced by can_example(), isca_can_receive_frame(), and isca_can_↩ transmit_frame().

A.9.2.4 ID

`uint32_t can_frame_s::ID`

Frame Identification

Definition at line 71 of file ISCA_CAN.h.

Referenced by can_example(), and isca_can_transmit_frame().

A.9.2.5 IDE

`uint32_t can_frame_s::IDE`

Frame format, CAN_FRAME_EXT or CAN_FRAME_STD)

Definition at line 72 of file ISCA_CAN.h.

Referenced by can_example(), isca_can_receive_frame(), and isca_can_↩ transmit_frame().

The documentation for this struct was generated from the following file:

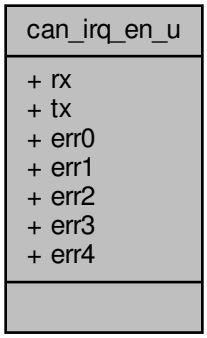
- ISCA_CAN.h

A.10 can_irq_en_u Union Reference

CAN controller IRQ enable union.

`#include <ISCA_CAN.h>`

Collaboration diagram for can_irq_en_u:



Data Fields

- uint8_t rx
- uint8_t tx
- uint8_t err0
- uint8_t err1
- uint8_t err2
- uint8_t err3
- uint8_t err4

A.10.1 Detailed Description

CAN controller IRQ enable union.

Definition at line 108 of file ISCA_CAN.h.

A.10.2 Field Documentation

A.10.2.1 rx

`uint8_t can_irq_en_u::rx`

RX IRQ; 0: disable / 1: enable

Definition at line 109 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.10.2.2 tx

`uint8_t can_irq_en_u::tx`

TX IRQ; 0: disable / 1: enable

Definition at line 110 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.10.2.3 err0

`uint8_t can_irq_en_u::err0`

Error IRQ; 0: disable / 1: enable

Definition at line 111 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.10.2.4 err1

`uint8_t can_irq_en_u::err1`

RX FIFO overrun IRQ; 0: disable / 1: enable

Definition at line 112 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.10.2.5 err2

`uint8_t can_irq_en_u::err2`

Passive error IRQ; 0: disable / 1: enable

Definition at line 113 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.10.2.6 err3

`uint8_t can_irq_en_u::err3`

Arbitration lost IRQ; 0: disable / 1: enable

Definition at line 114 of file ISCA_CAN.h.

Referenced by config_can(), and isca_can_init().

A.10.2.7 err4

`uint8_t can_irq_en_u::err4`

Bus error IRQ; 0: disable / 1: enable

Definition at line 115 of file ISCA_CAN.h.

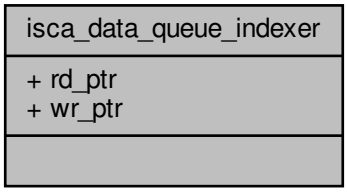
Referenced by config_can(), and isca_can_init().

The documentation for this union was generated from the following file:

- ISCA_CAN.h

A.11 isca_data_queue_indexer Struct Reference

Collaboration diagram for isca_data_queue_indexer:



Data Fields

- uint8_t rd_ptr
- uint8_t wr_ptr

A.11.1 Detailed Description

Definition at line 51 of file ISCA_QUEUE_INDEXER.c.

A.11.2 Field Documentation

A.11.2.1 rd_ptr

```
uint8_t isca_data_queue_indexer::rd_ptr
```

Definition at line 52 of file ISCA_QUEUE_INDEXER.c.

Referenced by isca_queue_acquire(), isca_queue_rd_ptr(), and isca_queue_↩ release().

A.11.2.2 wr_ptr

```
uint8_t isca_data_queue_indexer::wr_ptr
```

Definition at line 53 of file ISCA_QUEUE_INDEXER.c.

Referenced by isca_queue_acquire(), isca_queue_release(), and isca_queue_↩ wr_ptr().

The documentation for this struct was generated from the following file:

- ISCA_QUEUE_INDEXER.c

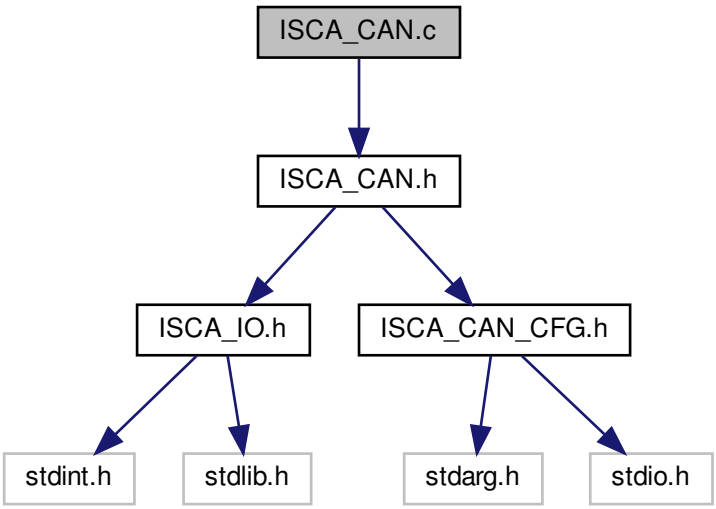
A.12 File Documentation

A.13 ISCA_CAN.c File Reference

Provides functions to communicate with the CAN controller.

```
#include "ISCA_CAN.h"
```

Include dependency graph for ISCA_CAN.c:



Macros

- #define CAN_BASE_OFFSET (0U)
  CAN controller base address fixed offset definition.
- #define CAN_MODE0_REG (CAN_BASE_OFFSET)
  Mode configuration register0.
- #define CAN_FILTER_MODE_REG (CAN_BASE_OFFSET+12U)
  Filter Mode enable register.
- #define CAN_IRQS_EN_REG (CAN_BASE_OFFSET+16U)
  IRQs enable register0 (Write-only)
- #define CAN_IRQS_STATUS_REG (CAN_BASE_OFFSET+12U)
  IRQs status register0 (Read-only)
- #define CAN_IRQ_CLEAR (CAN_IRQ_CLEAR_BASE_ADDRESS+4U)

CAN clear IRQ register (Read-only)

- #define CAN_COMMAND_TX_REG (CAN_BASE_OFFSET+4U)
  CAN command TX register (Write-Only)
- #define CAN_COMMAND_RX_REG (CAN_BASE_OFFSET+8U)
  CAN command RX register (Write-Only)
- #define CAN_STATUS_REG (CAN_BASE_OFFSET+8U)
  CAN status register (Read-only)
- #define CAN_CODE_REG0 (CAN_BASE_OFFSET+64U)
  CAN code register 0.
- #define CAN_CODE_REG1 (CAN_BASE_OFFSET+68U)
  CAN code register 1.
- #define CAN_CODE_REG2 (CAN_BASE_OFFSET+72U)
  CAN code register 2.
- #define CAN_CODE_REG3 (CAN_BASE_OFFSET+76U)
  CAN code register 3.
- #define CAN_MASK_REG0 (CAN_BASE_OFFSET+80U)
  CAN mask register 0.
- #define CAN_MASK_REG1 (CAN_BASE_OFFSET+84U)
  CAN mask register 1.
- #define CAN_MASK_REG2 (CAN_BASE_OFFSET+88U)
  CAN mask register 2.
- #define CAN_MASK_REG3 (CAN_BASE_OFFSET+92U)
  CAN mask register 3.
- #define CAN_BTR0_REG (CAN_BASE_OFFSET+24U)
  Bus timing register 0.
- #define CAN_BTR1_REG (CAN_BASE_OFFSET+28U)
  Bus timing register 1.
- #define CAN_RX_COUNTER_REG (CAN_BASE_OFFSET+116U)
  RX counter register.
- #define CAN_INVALID_RX_ACK_REG (CAN_BASE_OFFSET+4U)
  Acknowledge invalid RX register (Read-only)
- #define CAN_TX0_REG (CAN_BASE_OFFSET+64U)
  TX buffer0 register, layout:
- #define CAN_TX1_REG (CAN_BASE_OFFSET+68U)
  TX buffer1 register, layout:
- #define CAN_TX2_REG (CAN_BASE_OFFSET+72U)
  TX buffer2 register, layout:
- #define CAN_TX3_REG (CAN_BASE_OFFSET+76U)
  TX buffer3 register, layout:

- #define CAN_TX4_REG (CAN_BASE_OFFSET+80U)
  TX buffer4 register, layout:
- #define CAN_TX5_REG (CAN_BASE_OFFSET+84U)
  TX buffer5 register, layout:
- #define CAN_TX6_REG (CAN_BASE_OFFSET+88U)
  TX buffer6 register, layout:
- #define CAN_TX7_REG (CAN_BASE_OFFSET+92U)
  TX buffer7 register, layout:
- #define CAN_TX8_REG (CAN_BASE_OFFSET+96U)
  TX buffer8 register, layout:
- #define CAN_TX9_REG (CAN_BASE_OFFSET+100U)
  TX buffer9 register, layout:
- #define CAN_TX10_REG (CAN_BASE_OFFSET+104U)
  TX buffer10 register, layout:
- #define CAN_TX11_REG (CAN_BASE_OFFSET+108U)
  TX buffer11 register, layout:
- #define CAN_TX12_REG (CAN_BASE_OFFSET+112U)
  TX buffer12 register, layout:
- #define CAN_RX0_REG (CAN_BASE_OFFSET+64U)
  RX buffer0 register, layout:
- #define CAN_RX1_REG (CAN_BASE_OFFSET+68U)
  RX buffer1 register, layout:
- #define CAN_RX2_REG (CAN_BASE_OFFSET+72U)
  RX buffer2 register, layout:
- #define CAN_RX3_REG (CAN_BASE_OFFSET+76U)
  RX buffer3 register, layout:
- #define CAN_RX4_REG (CAN_BASE_OFFSET+80U)
  RX buffer4 register, layout:
- #define CAN_RX5_REG (CAN_BASE_OFFSET+84U)
  RX buffer5 register, layout:
- #define CAN_RX6_REG (CAN_BASE_OFFSET+88U)
  RX buffer6 register, layout:
- #define CAN_RX7_REG (CAN_BASE_OFFSET+92U)
  RX buffer7 register, layout:
- #define CAN_RX8_REG (CAN_BASE_OFFSET+96U)
  RX buffer8 register, layout:
- #define CAN_RX9_REG (CAN_BASE_OFFSET+100U)
  RX buffer9 register, layout:
- #define CAN_RX10_REG (CAN_BASE_OFFSET+104U)

RX buffer10 register, layout:

- #define CAN_RX11_REG (CAN_BASE_OFFSET+108U)
  RX buffer11 register, layout:
- #define CAN_RX12_REG (CAN_BASE_OFFSET+112U)
  RX buffer12 register, layout:
- #define CAN_CDR_REG (CAN_BASE_OFFSET+124U)
  Clock divider register, layout:
- #define CAN_TRIPLE_SAM (0x0U)
  Disable triple sampling
- #define CAN_CLK_O_OFF (0x1U)
  Disable output clock.
- #define CAN_CLK_DIV (0x0U)
  Disable triple sampling.
- #define CAN_CMNT_TRIGGER_TX (0x1U)
  Trigger TX default value.
- #define CAN_CMNT_RX_ACK (0x4U)
  Acknowledge RX default value.
- #define CAN_CMNT_CLR_OVERRUN (0x8U)
  Clear RX FIFO overrun default value.
- #define CAN_Q_TX_BUF_STATUS(status) (!((status>>2U) & 1U))
  Query TX logic state.
- #define CAN_Q_RX_STATUS(status) (((status)>>4U) & 1U)
  Query RX logic state.
- #define CAN_Q_RX_BUF_STATUS(status) (((status)>>0U) & 1U)
  Query RX buffer status.
- #define CAN_Q_RX_OVERRUN(status) (((status)>>1U) & 1U)
  Query RX FIFO overrun status.

Functions

- int isca_can_init (can_ctrl_s *can_ctrl)
  CAN controller initialize function.
- int isca_can_transmit_frame (can_ctrl_s *can_ctrl, can_frame_s *tx_frame, uint8_t req_type)
  CAN controller transmit frame.
- int isca_can_receive_frame (can_ctrl_s *can_ctrl, can_frame_s *rx_frame, uint8_t io_type)
  CAN controller receive frame.
- int isca_can_switch_mode (can_ctrl_s *can_ctrl, uint8_t reset_mode)

Switch can controller's reset mode on/off.

- void isca_can_set_filter (can_ctrl_s *can_ctrl)
  CAN controller set filter function.

Variables

- static uint8_t const can_tx_ext_payload_addr [ ]
- static uint8_t const can_rx_ext_payload_addr [ ]
- static uint8_t const can_tx_payload_addr [ ]
- static uint8_t const can_rx_payload_addr [ ]

A.13.1 Detailed Description

Provides functions to communicate with the CAN controller.

Version

1.0

Todo Implemented filtering modes:

- Extended mode. ID match for standard format (11-bit ID). Using double filter.
- Extended mode. ID match for extended format (29-bit ID). Using double filter.

A.13.2 Macro Definition Documentation

A.13.2.1 CAN_BASE_OFFSET

```
#define CAN_BASE_OFFSET (0U)
```

CAN controller base address fixed offset definition.

Definition at line 56 of file ISCA_CAN.c.

A.13.2.2 CAN_MODE0_REG

```
#define CAN_MODE0_REG (CAN_BASE_OFFSET)
```

Mode configuration register0.

Layout (0 disable, 1 enable):

```
CAN_MODE0_REG[7:4] -> reserved,
CAN_MODE0_REG[3:3] -> single_acceptance_filter_mode,
CAN_MODE0_REG[2:2] -> self_test_mode,
CAN_MODE0_REG[1:1] -> listen_only_mode,
CAN_MODE0_REG[0:0] -> reset_mode
```

Definition at line 87 of file ISCA_CAN.c.

Referenced by isca_can_init(), and isca_can_switch_mode().

A.13.2.3 CAN_FILTER_MODE_REG

```
#define CAN_FILTER_MODE_REG (CAN_BASE_OFFSET+12U)
```

Filter Mode enable register.

Layout (0 disable, 1 enable):

```
CAN_FILTER_MODE_REG[7:1] -> reserved
CAN_FILTER_MODE_REG[0:0] -> filter_mode
```

Definition at line 99 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.4 CAN_IRQS_EN_REG

```
#define CAN_IRQS_EN_REG (CAN_BASE_OFFSET+16U)
```

IRQs enable register0 (Write-only)

Layout (0 disable, 1 enable):

```
CAN_IRQS_EN_REG[7] -> bus_error_irq_en
CAN_IRQS_EN_REG[6] -> arbitration_lost_irq_en
CAN_IRQS_EN_REG[5] -> error_passive_irq_en
CAN_IRQS_EN_REG[4] -> reserved
CAN_IRQS_EN_REG[3] -> data_overrun_irq_en_ext
CAN_IRQS_EN_REG[2] -> error_warning_irq_en_ext
CAN_IRQS_EN_REG[1] -> transmit_irq_en_ext
CAN_IRQS_EN_REG[0] -> receive_irq_en_ext
```

Definition at line 118 of file ISCA_CAN.c.

Referenced by isca_can_init().

A.13.2.5 CAN_IRQS_STATUS_REG

```
#define CAN_IRQS_STATUS_REG (CAN_BASE_OFFSET+12U)
```

IRQs status register0 (Read-only)

Layout (0 inactive, 1 active):

```
CAN_IRQS_STATUS_REG[7] -> bus_error_irq_en
CAN_IRQS_STATUS_REG[6] -> arbitration_lost_irq_en
CAN_IRQS_STATUS_REG[5] -> error_passive_irq_en
CAN_IRQS_STATUS_REG[4] -> reserved
CAN_IRQS_STATUS_REG[3] -> data_overrun_irq_en_ext
CAN_IRQS_STATUS_REG[2] -> error_warning_irq_en_ext
CAN_IRQS_STATUS_REG[1] -> transmit_irq_en_ext
CAN_IRQS_STATUS_REG[0] -> receive_irq_en_ext
```

Definition at line 135 of file ISCA_CAN.c.

A.13.2.6 CAN_IRQ_CLEAR

```
#define CAN_IRQ_CLEAR (CAN_IRQ_CLEAR_BASE_ADDRESS+4U)
```

CAN clear IRQ register (Read-only)

A read request to the CAN clear IRQ register clears the interrupt at the CAN controller

Definition at line 145 of file ISCA_CAN.c.

A.13.2.7 CAN_COMMAND_TX_REG

```
#define CAN_COMMAND_TX_REG (CAN_BASE_OFFSET+4U)
```

CAN command TX register (Write-Only)

Layout:

```
CAN_COMMAND_TX_REG[7:1] -> reserved
CAN_COMMAND_TX_REG[4]   -> trigger TX (1 enable)
```

Definition at line 156 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.2.8 CAN_COMMAND_RX_REG

```
#define CAN_COMMAND_RX_REG (CAN_BASE_OFFSET+8U)
```

CAN command RX register (Write-Only)

Layout:

```
CAN_COMMAND_RX_REG[7:4] -> reserved
CAN_COMMAND_RX_REG[2]   -> release RX buffer, decrease RX counter (1 enable)
CAN_COMMAND_RX_REG[3]   -> clear RX FIFO overrun (1 enable)
CAN_COMMAND_RX_REG[1:0] -> reserved
```

Definition at line 169 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.9 CAN_STATUS_REG

```
#define CAN_STATUS_REG (CAN_BASE_OFFSET+8U)
```

CAN status register (Read-only)

Layout:

```
CAN_STATUS_REG[0] -> receive buffer status
CAN_STATUS_REG[1] -> overrun status (1:overrun)
CAN_STATUS_REG[2] -> transmit buffer status(1:idle, 0:busy)
CAN_STATUS_REG[3] -> transmission complete status
CAN_STATUS_REG[4] -> receive status
CAN_STATUS_REG[5] -> transmit status
CAN_STATUS_REG[6] -> error status
CAN_STATUS_REG[7] -> node buss off
```

Definition at line 186 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame(), and isca_can_transmit_frame().

A.13.2.10 CAN_CODE_REG0

```
#define CAN_CODE_REG0 (CAN_BASE_OFFSET+64U)
```

CAN code register 0.

Layout **extended** frame single filtering:

```
CAN_CODE_REG0[0] -> id[21]  CAN_CODE_REG0[1] -> id[22]
CAN_CODE_REG0[2] -> id[23]  CAN_CODE_REG0[3] -> id[24]
CAN_CODE_REG0[4] -> id[25]  CAN_CODE_REG0[5] -> id[26]
CAN_CODE_REG0[6] -> id[27]  CAN_CODE_REG0[7] -> id[28]
```

Layout **standard** frame, single filtering:

```
CAN_CODE_REG0[0] -> id[3]  CAN_CODE_REG0[1] -> id[4]
CAN_CODE_REG0[2] -> id[5]  CAN_CODE_REG0[3] -> id[6]
CAN_CODE_REG0[4] -> id[7]  CAN_CODE_REG0[5] -> id[8]
CAN_CODE_REG0[6] -> id[9]  CAN_CODE_REG0[7] -> id[10]
```

See also

CAN_MODE0_REG

Definition at line 255 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.11 CAN_CODE_REG1

```
#define CAN_CODE_REG1 (CAN_BASE_OFFSET+68U)
```

CAN code register 1.

Layout **extended** frame single filtering:

```
CAN_CODE_REG1[0] -> id[13]  CAN_CODE_REG1[1] -> id[14]
CAN_CODE_REG1[2] -> id[15]  CAN_CODE_REG1[3] -> id[16]
CAN_CODE_REG1[4] -> id[17]  CAN_CODE_REG1[5] -> id[18]
CAN_CODE_REG1[6] -> id[19]  CAN_CODE_REG1[7] -> id[20]
```

Layout **standard** frame single filtering:

```
CAN_CODE_REG1[0] -> none   CAN_CODE_REG1[1] -> none
CAN_CODE_REG1[2] -> none   CAN_CODE_REG1[3] -> none
CAN_CODE_REG1[4] -> rtr    CAN_CODE_REG1[5] -> id[0]
CAN_CODE_REG1[6] -> id[1]  CAN_CODE_REG1[7] -> id[2]
```

See also

CAN_MODE0_REG

Definition at line 278 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.12 CAN_CODE_REG2

```
#define CAN_CODE_REG2 (CAN_BASE_OFFSET+72U)
```

CAN code register 2.

Layout **extended** frame single filtering:

```
CAN_CODE_REG2[0] -> id[5]   CAN_CODE_REG2[1] -> id[6]
CAN_CODE_REG2[2] -> id[7]   CAN_CODE_REG2[3] -> id[8]
CAN_CODE_REG2[4] -> id[9]   CAN_CODE_REG2[5] -> id[10]
CAN_CODE_REG2[6] -> id[11]  CAN_CODE_REG2[7] -> id[12]
```

Layout **standard** frame single filtering:

```
CAN_CODE_REG2[0] -> data0[0] , CAN_CODE_REG2[1] -> data0[1]
CAN_CODE_REG2[2] -> data0[2] , CAN_CODE_REG2[3] -> data0[3]
CAN_CODE_REG2[4] -> data0[4] , CAN_CODE_REG2[5] -> data0[5]
CAN_CODE_REG2[6] -> data0[6] , CAN_CODE_REG2[7] -> data0[7]
```

See also

CAN_MODE0_REG

Definition at line 301 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.13 CAN_CODE_REG3

```
#define CAN_CODE_REG3 (CAN_BASE_OFFSET+76U)
```

CAN code register 3.

Layout **extended** frame single filtering:

```
CAN_CODE_REG3[0] -> none  , CAN_CODE_REG3[1] -> none
CAN_CODE_REG3[2] -> rtr   , CAN_CODE_REG3[3] -> id[0]
CAN_CODE_REG3[4] -> id[1] , CAN_CODE_REG3[5] -> id[2]
CAN_CODE_REG3[6] -> id[3] , CAN_CODE_REG3[7] -> id[4]
```

Layout **standard** frame single filtering:

```
CAN_CODE_REG3[0] -> data1[0]  CAN_CODE_REG3[1] -> data1[1]
CAN_CODE_REG3[2] -> data1[2]  CAN_CODE_REG3[3] -> data1[3]
CAN_CODE_REG3[4] -> data1[4]  CAN_CODE_REG3[5] -> data1[5]
CAN_CODE_REG3[6] -> data1[6]  CAN_CODE_REG3[7] -> data1[7]
```

See also

CAN_MODE0_REG

Definition at line 323 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.14 CAN_MASK_REG0

```
#define CAN_MASK_REG0 (CAN_BASE_OFFSET+80U)
```

CAN mask register 0.

Layout **extended** frame single filtering:

```
CAN_MASK_REG0[0] -> id[21]  CAN_MASK_REG0[1] -> id[22]
CAN_MASK_REG0[2] -> id[23]  CAN_MASK_REG0[3] -> id[24]
CAN_MASK_REG0[4] -> id[25]  CAN_MASK_REG0[5] -> id[26]
CAN_MASK_REG0[6] -> id[27]  CAN_MASK_REG0[7] -> id[28]
```

Layout **standard** frame single filtering:

```
CAN_MASK_REG0[0] -> id[3]  CAN_MASK_REG0[1] -> id[4]
CAN_MASK_REG0[2] -> id[5]  CAN_MASK_REG0[3] -> id[6]
CAN_MASK_REG0[4] -> id[7]  CAN_MASK_REG0[5] -> id[8]
CAN_MASK_REG0[6] -> id[9]  CAN_MASK_REG0[7] -> id[10]
```

See also

CAN_MODE0_REG

Definition at line 345 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.15 CAN_MASK_REG1

```
#define CAN_MASK_REG1 (CAN_BASE_OFFSET+84U)
```

CAN mask register 1.

Layout **extended** frame single filtering:

```
CAN_MASK_REG1[0] -> id[13]  CAN_MASK_REG1[1] -> id[14]
CAN_MASK_REG1[2] -> id[15]  CAN_MASK_REG1[3] -> id[16]
CAN_MASK_REG1[4] -> id[17]  CAN_MASK_REG1[5] -> id[18]
CAN_MASK_REG1[6] -> id[19]  CAN_MASK_REG1[7] -> id[20]
```

Layout **standard** frame single filtering:

```
CAN_MASK_REG1[0] -> none   CAN_MASK_REG1[1] -> none
CAN_MASK_REG1[2] -> none   CAN_MASK_REG1[3] -> none
CAN_MASK_REG1[4] -> rtr    CAN_MASK_REG1[5] -> id[0]
CAN_MASK_REG1[6] -> id[1]  CAN_MASK_REG1[7] -> id[2]
```

See also

CAN_MODE0_REG

Definition at line 367 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.16 CAN_MASK_REG2

```
#define CAN_MASK_REG2 (CAN_BASE_OFFSET+88U)
```

CAN mask register 2.

Layout **extended** frame single filtering:

```
CAN_MASK_REG2[0] -> id[ 5]  CAN_MASK_REG2[1] -> id[ 6]
CAN_MASK_REG2[2] -> id[ 7]  CAN_MASK_REG2[3] -> id[ 8]
CAN_MASK_REG2[4] -> id[ 9]  CAN_MASK_REG2[5] -> id[10]
CAN_MASK_REG2[6] -> id[11]  CAN_MASK_REG2[7] -> id[12]
```

Layout **standard** frame single filtering:

```
CAN_MASK_REG2[0] -> data0[0]  CAN_MASK_REG2[1] -> data0[1]
CAN_MASK_REG2[2] -> data0[2]  CAN_MASK_REG2[3] -> data0[3]
CAN_MASK_REG2[4] -> data0[4]  CAN_MASK_REG2[5] -> data0[5]
CAN_MASK_REG2[6] -> data0[6]  CAN_MASK_REG2[7] -> data0[7]
```

See also

CAN_MODE0_REG

Definition at line 389 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.17 CAN_MASK_REG3

```
#define CAN_MASK_REG3 (CAN_BASE_OFFSET+92U)
```

CAN mask register 3.

Layout **extended** frame single filtering:

```
CAN_MASK_REG3[0] -> none   CAN_MASK_REG3[1] -> none
CAN_MASK_REG3[2] -> rtr    CAN_MASK_REG3[3] -> id[0]
CAN_MASK_REG3[4] -> id[1]  CAN_MASK_REG3[5] -> id[2]
CAN_MASK_REG3[6] -> id[3]  CAN_MASK_REG3[7] -> id[4]
```

Layout **standard** frame single filtering:

```
CAN_MASK_REG3[0] -> data1[0]  CAN_MASK_REG3[1] -> data1[1]
CAN_MASK_REG3[2] -> data1[2]  CAN_MASK_REG3[3] -> data1[3]
CAN_MASK_REG3[4] -> data1[4]  CAN_MASK_REG3[5] -> data1[5]
CAN_MASK_REG3[6] -> data1[6]  CAN_MASK_REG3[7] -> data1[7]
```

See also

CAN_MODE0_REG

Definition at line 411 of file ISCA_CAN.c.

Referenced by isca_can_set_filter().

A.13.2.18 CAN_BTR0_REG

```
#define CAN_BTR0_REG (CAN_BASE_OFFSET+24U)
```

Bus timing register 0.

Layout:

```
CAN_BTR0_REG[7:6] -> Synch Jump Width: (value+1)
CAN_BTR0_REG[5:0] -> Baud rate prescaler: 2*(value+1)
```

Definition at line 424 of file ISCA_CAN.c.

Referenced by isca_can_init().

A.13.2.19 CAN_BTR1_REG

```
#define CAN_BTR1_REG (CAN_BASE_OFFSET+28U)
```

Bus timing register 1.

Layout:

```
CAN_BTR1_REG[7:7] -> Triple sampling (0 disable, 1 enable)
CAN_BTR1_REG[6:4] -> TSEG1: (value+1)
CAN_BTR1_REG[3:0] -> TSEG2: (value+1)
```

Definition at line 436 of file ISCA_CAN.c.

Referenced by isca_can_init().

A.13.2.20 CAN_RX_COUNTER_REG

```
#define CAN_RX_COUNTER_REG (CAN_BASE_OFFSET+116U)
```

RX counter register.

Layout:

```
CAN_RX_COUNTER_REG[6:0] -> unread frames
```

Definition at line 449 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.21 CAN_INVALID_RX_ACK_REG

```
#define CAN_INVALID_RX_ACK_REG (CAN_BASE_OFFSET+4U)
```

Acknowledge invalid RX register (Read-only)

RX FIFO overrun occurred; reading CAN_INVALID_RX_ACK_REG acknowledges overrun. Layout:

```
CAN_INVALID_RX_ACK_REG[7:1] -> reserved
CAN_INVALID_RX_ACK_REG[0:0] -> RX state ('1' valid)
```

Note

After acknowledging you should also decrease RX counter

Definition at line 463 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.22 CAN_TX0_REG

```
#define CAN_TX0_REG (CAN_BASE_OFFSET+64U)
```

TX buffer0 register, layout:

```
CAN_TX0_REG[7]   -> can_frame_s.IDE
CAN_TX0_REG[6]   -> can_frame_s.RTR
CAN_TX0_REG[5:4] -> Reserved
CAN_TX0_REG[3:0] -> can_frame_s.DLC
```

Definition at line 640 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.2.23 CAN_TX1_REG

```
#define CAN_TX1_REG (CAN_BASE_OFFSET+68U)
```

TX buffer1 register, layout:

```
CAN_TX1_REG[7:0] -> can_frame_s.ID[28:21]
```

Definition at line 648 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.2.24 CAN_TX2_REG

```
#define CAN_TX2_REG (CAN_BASE_OFFSET+72U)
```

TX buffer2 register, layout:

```
CAN_TX2_REG[7:0] -> can_frame_s.ID[20:13]
```

Definition at line 656 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.2.25 CAN_TX3_REG

```
#define CAN_TX3_REG (CAN_BASE_OFFSET+76U)
```

TX buffer3 register, layout:

```
CAN_TX3_REG[7:0] -> can_frame_s.ID[12:5]
```

Definition at line 664 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.2.26 CAN_TX4_REG

```
#define CAN_TX4_REG (CAN_BASE_OFFSET+80U)
```

TX buffer4 register, layout:

```
CAN_TX4_REG[7:3] -> can_frame_s.ID[4:0]
CAN_TX4_REG[2:0] -> Reserved
```

Definition at line 673 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.2.27 CAN_TX5_REG

```
#define CAN_TX5_REG (CAN_BASE_OFFSET+84U)
```

TX buffer5 register, layout:

```
CAN_TX5_REG[7:0] -> can_frame_s.DATA[0]
```

Definition at line 681 of file ISCA_CAN.c.

A.13.2.28 CAN_TX6_REG

```
#define CAN_TX6_REG (CAN_BASE_OFFSET+88U)
```

TX buffer6 register, layout:

```
CAN_TX6_REG[7:0] -> can_frame_s.DATA[1]
```

Definition at line 689 of file ISCA_CAN.c.

A.13.2.29 CAN_TX7_REG

```
#define CAN_TX7_REG (CAN_BASE_OFFSET+92U)
```

TX buffer7 register, layout:

```
CAN_TX7_REG[7:0] -> can_frame_s.DATA[2]
```

Definition at line 697 of file ISCA_CAN.c.

A.13.2.30 CAN_TX8_REG

```
#define CAN_TX8_REG (CAN_BASE_OFFSET+96U)
```

TX buffer8 register, layout:

```
CAN_TX8_REG[7:0] -> can_frame_s.DATA[3]
```

Definition at line 705 of file ISCA_CAN.c.

A.13.2.31 CAN_TX9_REG

```
#define CAN_TX9_REG (CAN_BASE_OFFSET+100U)
```

TX buffer9 register, layout:

```
CAN_TX9_REG[7:0] -> can_frame_s.DATA[4]
```

Definition at line 713 of file ISCA_CAN.c.

A.13.2.32 CAN_TX10_REG

```
#define CAN_TX10_REG (CAN_BASE_OFFSET+104U)
```

TX buffer10 register, layout:

```
CAN_TX10_REG[7:0] -> can_frame_s.DATA[5]
```

Definition at line 721 of file ISCA_CAN.c.

A.13.2.33 CAN_TX11_REG

```
#define CAN_TX11_REG (CAN_BASE_OFFSET+108U)
```

TX buffer11 register, layout:

```
CAN_TX11_REG[7:0] -> can_frame_s.DATA[6]
```

Definition at line 729 of file ISCA_CAN.c.

A.13.2.34 CAN_TX12_REG

```
#define CAN_TX12_REG (CAN_BASE_OFFSET+112U)
```

TX buffer12 register, layout:

```
CAN_TX12_REG[7:0] -> can_frame_s.DATA[7]
```

Definition at line 737 of file ISCA_CAN.c.

A.13.2.35 CAN_RX0_REG

```
#define CAN_RX0_REG (CAN_BASE_OFFSET+64U)
```

RX buffer0 register, layout:

```
CAN_RX0_REG[7]   -> can_frame_s.IDE
CAN_RX0_REG[6]   -> can_frame_s.RTR
CAN_RX0_REG[5:4] -> Reserved
CAN_RX0_REG[3:0] -> can_frame_s.DLC
```

Definition at line 748 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.36 CAN_RX1_REG

```
#define CAN_RX1_REG (CAN_BASE_OFFSET+68U)
```

RX buffer1 register, layout:

```
CAN_RX1_REG[7:0] -> can_frame_s.ID[28:21]
```

Definition at line 756 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.37 CAN_RX2_REG

```
#define CAN_RX2_REG (CAN_BASE_OFFSET+72U)
```

RX buffer2 register, layout:

```
CAN_RX2_REG[7:0] -> can_frame_s.ID[20:13]
```

Definition at line 764 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.38 CAN_RX3_REG

```
#define CAN_RX3_REG (CAN_BASE_OFFSET+76U)
```

RX buffer3 register, layout:

```
CAN_RX3_REG[7:0] -> can_frame_s.ID[12:5]
```

Definition at line 772 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.39 CAN_RX4_REG

```
#define CAN_RX4_REG (CAN_BASE_OFFSET+80U)
```

RX buffer4 register, layout:

```
CAN_RX4_REG[7:3] -> can_frame_s.ID[4:0]
CAN_RX4_REG[2:0] -> Reserved
```

Definition at line 781 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.40 CAN_RX5_REG

```
#define CAN_RX5_REG (CAN_BASE_OFFSET+84U)
```

RX buffer5 register, layout:

```
CAN_RX5_REG[7:0] -> can_frame_s.DATA[0]
```

Definition at line 789 of file ISCA_CAN.c.

A.13.2.41 CAN_RX6_REG

```
#define CAN_RX6_REG (CAN_BASE_OFFSET+88U)
```

RX buffer6 register, layout:

```
CAN_RX6_REG[7:0] -> can_frame_s.DATA[1]
```

Definition at line 797 of file ISCA_CAN.c.

A.13.2.42 CAN_RX7_REG

```
#define CAN_RX7_REG (CAN_BASE_OFFSET+92U)
```

RX buffer7 register, layout:

```
CAN_RX7_REG[7:0] -> can_frame_s.DATA[2]
```

Definition at line 805 of file ISCA_CAN.c.

A.13.2.43 CAN_RX8_REG

```
#define CAN_RX8_REG (CAN_BASE_OFFSET+96U)
```

RX buffer8 register, layout:

```
CAN_RX8_REG[7:0] -> can_frame_s.DATA[3]
```

Definition at line 813 of file ISCA_CAN.c.

A.13.2.44 CAN_RX9_REG

```
#define CAN_RX9_REG (CAN_BASE_OFFSET+100U)
```

RX buffer9 register, layout:

```
CAN_RX9_REG[7:0] -> can_frame_s.DATA[4]
```

Definition at line 821 of file ISCA_CAN.c.

A.13.2.45 CAN_RX10_REG

```
#define CAN_RX10_REG (CAN_BASE_OFFSET+104U)
```

RX buffer10 register, layout:

```
CAN_RX10_REG[7:0] -> can_frame_s.DATA[5]
```

Definition at line 829 of file ISCA_CAN.c.

A.13.2.46 CAN_RX11_REG

```
#define CAN_RX11_REG (CAN_BASE_OFFSET+108U)
```

RX buffer11 register, layout:

```
CAN_RX11_REG[7:0] -> can_frame_s.DATA[6]
```

Definition at line 837 of file ISCA_CAN.c.

A.13.2.47 CAN_RX12_REG

```
#define CAN_RX12_REG (CAN_BASE_OFFSET+112U)
```

RX buffer12 register, layout:

```
CAN_RX12_REG[7:0] -> can_frame_s.DATA[7]
```

Definition at line 845 of file ISCA_CAN.c.

A.13.2.48 CAN_CDR_REG

```
#define CAN_CDR_REG (CAN_BASE_OFFSET+124U)
```

Clock divider register, layout:

```
CAN_CDR_REG[7:7] -> extended_mode (0 disable, 1 enable)
CAN_CDR_REG[6:4] -> reserved
CAN_CDR_REG[3:3] -> clock_off (0 disable, 1 enable)
CAN_CDR_REG[2:0] -> clock divider (integer valid range 0 - 6)
```

Definition at line 858 of file ISCA_CAN.c.

Referenced by isca_can_init().

A.13.2.49 CAN_TRIPLE_SAM

```
#define CAN_TRIPLE_SAM (0x0U)
```

Disable triple sampling

CAN_BTR1_REG[7] default (hard-coded) value

Definition at line 865 of file ISCA_CAN.c.

Referenced by isca_can_init().

A.13.2.50 CAN_CLK_O_OFF

```
#define CAN_CLK_O_OFF (0x1U)
```

Disable output clock.

CAN_CDR_REG[3] default (hard-coded) value

Definition at line 872 of file ISCA_CAN.c.

Referenced by isca_can_init().

A.13.2.51 CAN_CLK_DIV

```
#define CAN_CLK_DIV (0x0U)
```

Disable triple sampling.

CAN_CDR_REG[2:0] default (hard-coded) value

Definition at line 879 of file ISCA_CAN.c.

Referenced by isca_can_init().

A.13.2.52 CAN_CMNT_TRIGGER_TX

```
#define CAN_CMNT_TRIGGER_TX (0x1U)
```

Trigger TX default value.

Default value for CAN_COMMAND_TX_REG register to trigger TX

Note

You should initialize TX buffers first

Definition at line 887 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.2.53 CAN_CMNT_RX_ACK

```
#define CAN_CMNT_RX_ACK (0x4U)
```

Acknowledge RX default value.

Default value for CAN_COMMAND_RX_REG register to acknowledge RX

Definition at line 894 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.2.54 CAN_CMNT_CLR_OVERRUN

```
#define CAN_CMNT_CLR_OVERRUN (0x8U)
```

Clear RX FIFO overrun default value.

Default value for CAN_COMMAND_RX_REG register to clear RX FIFO overrun

Definition at line 902 of file ISCA_CAN.c.

A.13.2.55 CAN_Q_TX_BUF_STATUS

```
#define CAN_Q_TX_BUF_STATUS(
            status ) ( !((status>>2U) & 1U) )
```

Query TX logic state.

Decode CAN_STATUS_REG value to find out TX logic state

Returns

0 idle
1 active

Definition at line 911 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.2.56 CAN_Q_RX_STATUS

```
#define CAN_Q_RX_STATUS(
            status ) ( ((status)>>4U) & 1U )
```

Query RX logic state.

Decode CAN_STATUS_REG value to find out RX logic state

Returns

0 idle
1 active

Definition at line 920 of file ISCA_CAN.c.

A.13.2.57 CAN_Q_RX_BUF_STATUS

```
#define CAN_Q_RX_BUF_STATUS(
            status ) ( ((status)>>0U) & 1U )
```

Query RX buffer status.

Decode CAN_STATUS_REG value to find out RX FIFO status

Returns

0 (empty)
1 full (frame received)

Definition at line 929 of file ISCA_CAN.c.

A.13.2.58 CAN_Q_RX_OVERRUN

```
#define CAN_Q_RX_OVERRUN(
            status ) ( ((status)>>1U) & 1U )
```

Query RX FIFO overrun status.

Decode CAN_STATUS_REG value to find out RX FIFO status

Returns

0 no overrun
1 overrun occurred

Definition at line 938 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.3 Function Documentation

A.13.3.1 isca_can_init()

```
int isca_can_init (
           can_ctrl_s * can_ctrl )
```

CAN controller initialize function.

Parameters

in	*can_ctrl*	CAN controller instance pointer

Returns

ISCA_CAN_OK

See also

CAN_MODE0_REG
CAN_IRQS_EN_REG

Definition at line 1013 of file ISCA_CAN.c.

References can_controller_s::addr, can_controller_s::brp, CAN_BTR0_REG, CAN_BTR1_REG, CAN_CDR_REG, CAN_CLK_DIV, CAN_CLK_O_OFF, CAN_IRQ_ON, CAN_IRQS_EN_REG, CAN_MODE0_REG, CAN_TRIPLE_SAM, can_irq_en_u::err0, can_irq_en_u::err1, can_irq_en_u::err2, can_irq_en_u::err3, can_irq_en_u::err4, can_controller_s::irq, can_controller_s::irqs_en, ISCA_CAN_MODE_RESET_OFF, ISCA_CAN_MODE_RESET_ON, ISCA_CAN_OK, isca_can_set_filter(), ISCA_FPGA_Write8Bit(), can_irq_en_u::rx, can_controller_s::sjw, can_controller_s::tseg1, can_controller_s::tseg2, and can_irq_en_u::tx.

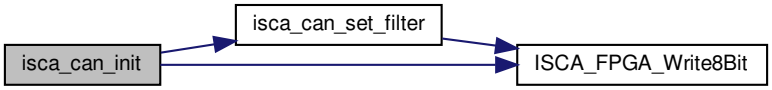
Referenced by lbr_isca_can_init().

```
1014 {
1015
1016 #if (CAN_MODE == CAN_2B)
1017     uint8_t irqs;
1018 #endif // (CAN_MODE == CAN_2B)
1019
1020     uint8_t btr0;
1021     uint8_t btr1;
1022     uint8_t cdr;
1023     uint8_t cfg0;
1024     size_t  addr = can_ctrl->addr;
1025
1026     // Switch on can controller's reset mode, and clear ext_mode values
1027     ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG,
      ISCA_CAN_MODE_RESET_ON);
1028
1029     /* Set bus timing register 1*/
1030     btr0  = 0x0U;
1031     btr0 |= (can_ctrl->sjw << 6U);
1032     btr0 |= (can_ctrl->brp & 0x3F);
```

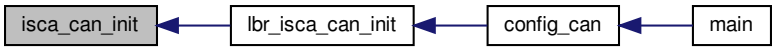
```
1033     ISCA_FPGA_Write8Bit(addr+CAN_BTR0_REG, btr0);
1034
1035     /* Set bus timing register 2*/
1036     btr1  = 0x0U;
1037     btr1 |= (CAN_TRIPLE_SAM << 7U);
1038     btr1 |= ((can_ctrl->tseg2 & 0x7U) << 4U);
1039     btr1 |=  (can_ctrl->tseg1 & 0xFU);
1040
1041     ISCA_FPGA_Write8Bit(addr+CAN_BTR1_REG, btr1);
1042
1043     /* Set bus timing register 1*/
1044     cdr  = 0x0U;
1045     cdr |= (CAN_CLK_O_OFF  << 3U);
1046     cdr |= (CAN_CLK_DIV    &  7U);
1047 #if (CAN_MODE == CAN_2B)
1048     // Enable CAN 2B (extended) mode*/
1049     cdr |= (0x1U << 7U);
1050 #endif // !(CAN_MODE == CAN_2B)
1051
1052     ISCA_FPGA_Write8Bit(addr+CAN_CDR_REG, cdr);
1053
1054     /* Set up can filter*/
1055 #if !(CAN_MODE == CAN_2B)
1056     isca_can_set_filter(can_ctrl);
1057 #else
1058     isca_can_set_filter(can_ctrl);
1059 #endif // !(CAN_MODE == CAN_2B)
1060
1061 #if !(CAN_MODE == CAN_2B)
1062     /* Set CAN mode register*/
1063     cfg0  = ISCA_CAN_MODE_RESET_OFF;
1064     if ( can_ctrl->irq == CAN_IRQ_ON ) {
1065         cfg0 |= ((can_ctrl->irqs_en.err0&1U) << 0x4U); // overrun IRQ
1066         cfg0 |= ((can_ctrl->irqs_en.err1&1U) << 0x3U); // error IRQ
1067         cfg0 |= ((can_ctrl->irqs_en.tx&1U)   << 0x2U); // transmit IRQ
1068         cfg0 |= ((can_ctrl->irqs_en.rx&1U)   << 0x1U); // receive IRQ
1069     }
1071     ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG, cfg0);
1072
1073 #else
1074
1075     /* Set CAN mode register*/
1076     cfg0  = ISCA_CAN_MODE_RESET_OFF;
1077     cfg0 |= 0x8U; //dual filtering not supported in current version
1078     ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG, cfg0|0x8U);
1079
1080     irqs = 0U;
1081     if ( can_ctrl->irq == CAN_IRQ_ON ) {
1082
1083         /* Activate IRQs as ordered*/
1084         irqs |= ((can_ctrl->irqs_en.err4&1U) << 0x7U); /* bus error IRQ */
1085         irqs |= ((can_ctrl->irqs_en.err3&1U) << 0x6U); /* arbitration lost IRQ */
1086         irqs |= ((can_ctrl->irqs_en.err2&1U) << 0x5U); /* error passive IRQ */
1087         irqs |= ((can_ctrl->irqs_en.err1&1U) << 0x3U); /* overrun IRQ */
1088         irqs |= ((can_ctrl->irqs_en.err0&1U) << 0x2U); /* error IRQ */
1089         irqs |= ((can_ctrl->irqs_en.tx&1U)   << 0x1U); /* transmit IRQ */
1090         irqs |= ((can_ctrl->irqs_en.rx&1U)   << 0x0U); /* receive IRQ */
1091
```

```
1092     }
1094     ISCA_FPGA_Write8Bit(addr+CAN_IRQS_EN_REG, irqs);
1095
1096 #endif // !(CAN_MODE == CAN_2B)
1097
1098     return ISCA_CAN_OK;
1099 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.13.3.2 isca_can_transmit_frame()

```
int isca_can_transmit_frame (
          can_ctrl_s * can_ctrl,
          can_frame_s * tx_frame,
          uint8_t req_type )
```

CAN controller transmit frame.

Precondition

CAN controller initialization

Parameters

in	*can_ctrl*	CAN controller struct
in	*tx_frame*	CAN frame struct pointer
in	*req_type*	/ref CAN_REF_BLOCKING or CAN_REF_NONBLOCKING

Returns

ISCA_CAN_BUSY
ISCA_CAN_INV_IO_TYPE
ISCA_CAN_OK

<

Todo Sleep while TX requested and controller is busy

Definition at line 1111 of file ISCA_CAN.c.

References can_controller_s::addr, CAN_CMNT_TRIGGER_TX, CAN_COMM↩
AND_TX_REG, CAN_FRAME_EXT, CAN_Q_TX_BUF_STATUS, CAN_REQ_↩
BLOCKING, CAN_REQ_NONBLOCKING, CAN_STATUS_REG, CAN_TX0_R↩
EG, CAN_TX1_REG, CAN_TX2_REG, CAN_TX3_REG, CAN_TX4_REG, can↩
_tx_ext_payload_addr, can_tx_payload_addr, can_frame_s::DATA, can_frame↩
_s::DLC, can_frame_s::ID, can_frame_s::IDE, ISCA_CAN_BUSY, ISCA_CAN↩
_INV_REQ_TYPE, ISCA_CAN_OK, ISCA_FPGA_Read8Bit(), ISCA_FPGA_↩
Write8Bit(), and can_frame_s::RTR.

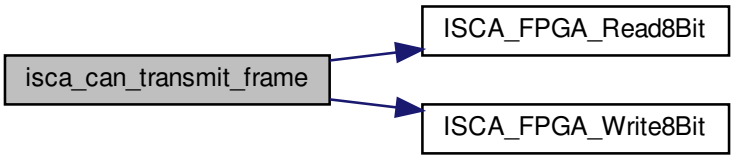
Referenced by lbr_isca_can_transmit_pkt().

```
1112 {
1113
1114 #if !(CAN_MODE == CAN_2B)
1115     uint8_t tx_header[2] = {0x0U, 0x0U};
1116 #else
1117     uint8_t  tx_header[5] = {0x0U, 0x0U, 0x0U, 0x0U, 0x0U};
1118 #endif // !(CAN_MODE == CAN_2B)
1119
1120     uint8_t i;
1121     size_t  addr = can_ctrl->addr;
1122
1123     if (req_type == CAN_REQ_NONBLOCKING) {
1124         if (CAN_Q_TX_BUF_STATUS(ISCA_FPGA_Read8Bit(addr+
      CAN_STATUS_REG))) {
1125             return ISCA_CAN_BUSY;
```

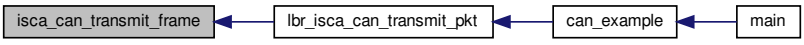
```
1126         }
1127     }
1128     else if (req_type == CAN_REQ_BLOCKING) {
1129         while(CAN_Q_TX_BUF_STATUS(ISCA_FPGA_Read8Bit(addr+
     CAN_STATUS_REG))) {
1130             //TODO: sleep
1132         }; /*tx busy*/
1133     }
1134     else {
1135         return ISCA_CAN_INV_REQ_TYPE;
1136     }
1139 #if !(CAN_MODE == CAN_2B)
1140     tx_header[0] |= (tx_frame->ID  & 0x7F8U) >> 3U; //ID[10:3]
1141     ISCA_FPGA_Write8Bit(addr+CAN_TX0_REG, tx_header[0]);
1142
1143     tx_header[1] |= (tx_frame->ID  & 0x7U)   << 5U; //ID[2:0]
1144     tx_header[1] |= (tx_frame->RTR & 0x1U)   << 4U;
1145     tx_header[1] |= (tx_frame->DLC & 0xFU);
1146     ISCA_FPGA_Write8Bit(addr+CAN_TX1_REG, tx_header[1]);
1147 #else
1148     //tx_header[0] |= (tx_frame->IDE & 0x4U) << 5U; // SCAN
1149     tx_header[0] |= (tx_frame->IDE & 0x1U) << 7U;
1150     tx_header[0] |= (tx_frame->RTR & 0x1U) << 6U;
1151     tx_header[0] |= (tx_frame->DLC & 0xFU);
1152     ISCA_FPGA_Write8Bit(addr+CAN_TX0_REG, tx_header[0]);
1153
1154     if (tx_frame->IDE == CAN_FRAME_EXT) {
1155         tx_header[1] = (tx_frame->ID & 0x1FE00000U) >> 21U;
1156         ISCA_FPGA_Write8Bit(addr+CAN_TX1_REG, tx_header[1]);
1157
1158         tx_header[2] = (tx_frame->ID & 0x1FE000U) >> 13U;
1159         ISCA_FPGA_Write8Bit(addr+CAN_TX2_REG, tx_header[2]);
1160
1161         tx_header[3] = (tx_frame->ID & 0x1FE0U) >> 5U;
1162         ISCA_FPGA_Write8Bit(addr+CAN_TX3_REG, tx_header[3]);
1163
1164         tx_header[4] = (tx_frame->ID & 0x1F) << 3U;
1165         ISCA_FPGA_Write8Bit(addr+CAN_TX4_REG, tx_header[4]);
1166     }
1167     else {
1168         tx_header[1] = (tx_frame->ID & 0x7F8) >> 3U;
1169         ISCA_FPGA_Write8Bit(addr+CAN_TX1_REG, tx_header[1]);
1170
1171         tx_header[2] = (tx_frame->ID & 0x7U) << 5U;
1172         ISCA_FPGA_Write8Bit(addr+CAN_TX2_REG, tx_header[2]);
1173     }
1174 #endif // !(CAN_MODE == CAN_2B)
1175
1176 #if !(CAN_MODE == CAN_2B)
1177     // Frame payload may be of variable size
1178     for (i = 0; i < tx_frame->DLC; i++) {
1179         ISCA_FPGA_Write8Bit(addr+can_tx_payload_addr[i], tx_frame->
     DATA[i]);
1180     }
1181 #else
1182
1183     if (tx_frame->IDE == CAN_FRAME_EXT) { /*CAN_2B extended frame ID*/
1184         /*Frame payload may be of variable size*/
```

```
1185         for (i = 0; i < tx_frame->DLC; i++) {
1186             ISCA_FPGA_Write8Bit(addr+can_tx_ext_payload_addr[i],
     tx_frame->DATA[i]);
1187         }
1188     } else { /*CAN_2B basic frame ID*/
1189         /*Frame payload may be of variable size*/
1190         for (i = 0; i < tx_frame->DLC; i++) {
1191             ISCA_FPGA_Write8Bit(addr+can_tx_payload_addr[i], tx_frame
     ->DATA[i]);
1192         }
1193     }
1194 #endif // !(CAN_MODE == CAN_2B)
1195
1196     /*trigger TX*/
1197     ISCA_FPGA_Write8Bit(addr+CAN_COMMAND_TX_REG,
     CAN_CMNT_TRIGGER_TX);
1198
1199     return ISCA_CAN_OK;
1200 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.13.3.3 isca_can_receive_frame()

```
int isca_can_receive_frame (
          can_ctrl_s * can_ctrl,
          can_frame_s * rx_frame,
          uint8_t io_type )
```

CAN controller receive frame.

Precondition

CAN controller initialization

Parameters

`in`	*can_ctrl*	CAN controller struct pointer
`out`	*rx_frame*	CAN frame struct pointer
`in`	*req_type*	CAN_IO_BLOCKING or CAN_IO_NONBLOCKING

Returns

ISCA_CAN_RX_FIFO_EMPTY
ISCA_CAN_INV_IO_TYPE
ISCA_CAN_OK

<

Todo Sleep while waiting for a frame to arrive

<

Todo Notify user if during CAN 2B mode and filtering is set to basic, an extended frame is has been received or vice versa

Definition at line 1212 of file ISCA_CAN.c.

References can_controller_s::addr, CAN_CMNT_RX_ACK, CAN_COMMAND↩ _RX_REG, CAN_FRAME_EXT, CAN_INVALID_RX_ACK_REG, CAN_Q_RX_↩ OVERRUN, CAN_REQ_BLOCKING, CAN_REQ_NONBLOCKING, CAN_RX0↩ _REG, CAN_RX1_REG, CAN_RX2_REG, CAN_RX3_REG, CAN_RX4_REG, CAN_RX_COUNTER_REG, can_rx_ext_payload_addr, can_rx_payload_addr, CAN_STATUS_REG, can_frame_s::DATA, can_controller_s::frm_md, can_↩ frame_s::IDE, ISCA_CAN_INV_REQ_TYPE, ISCA_CAN_RX_FIFO_EMPTY, I↩ SCA_FPGA_Read8Bit(), and ISCA_FPGA_Write8Bit().

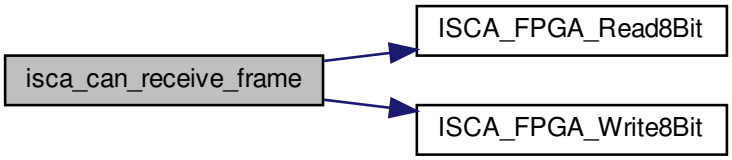
Referenced by isca_can_receive_pkt_irq(), and lbr_isca_can_receive_pkt().

```
1213 {
1214
1215 #if !(CAN_MODE == CAN_2B)
1216     uint8_t rx_header[2] = {0x0U, 0x0U};
1217 #else
1218     uint8_t rx_header[5] = {0x0U, 0x0U, 0x0U, 0x0U, 0x0U};
1219     uint8_t ide = 0x0U;
1220 #endif // !(CAN_MODE == CAN_2B)
1221
1222     int32_t  remain_frames = 0x0U;
1223     size_t   addr  = can_ctrl->addr;
1224     uint32_t id    = 0x0U;
1225     uint8_t  rtr   = 0x0U;
1226     uint8_t  dlc   = 0x0U;
1227     uint8_t  i;
1228
1229     if ( CAN_Q_RX_OVERRUN(ISCA_FPGA_Read8Bit(addr+
      CAN_STATUS_REG)) ) {
1230
1231         while (ISCA_FPGA_Read8Bit(addr+CAN_INVALID_RX_ACK_REG) != 0
      ) {
1232             /*Acknowledge RX buff, should decrease frames counter*/
1233             ISCA_FPGA_Write8Bit(addr+CAN_COMMAND_RX_REG,
      CAN_CMNT_RX_ACK);
1234         }
1236     }
1238     remain_frames = ISCA_FPGA_Read8Bit(addr+CAN_RX_COUNTER_REG);
1239
1240     if ( io_type == CAN_REQ_NONBLOCKING ) {
1241
1242         if ( remain_frames == 0x0U ) {
1243             return ISCA_CAN_RX_FIFO_EMPTY;
1244         }
1246     }
1247     else if (io_type == CAN_REQ_BLOCKING) {
1248
1249         while (remain_frames == 0x0U) {
1250             remain_frames = ISCA_FPGA_Read8Bit(addr+
      CAN_RX_COUNTER_REG);
1251             /*TODO: sleep*/
1253         }; /*wait for a frame to arrive*/
1254
1255     }
1256     else {
1257
1258         return ISCA_CAN_INV_REQ_TYPE;
1259
1260     } /*Invalid io_type*/
1261
1262 #if !(CAN_MODE == CAN_2B)
1263     // read frame header
1264     rx_header[0] = ISCA_FPGA_Read8Bit(addr+CAN_RX0_REG);
1265     rx_header[1] = ISCA_FPGA_Read8Bit(addr+CAN_RX1_REG);
1266
1268     id  |= ( (uint32_t)rx_header[0] << 3);
1269     id  |= ((rx_header[1] >> 5) & 0x7U);
1270     rtr  = ((rx_header[1] >> 4) & 0x1U);
1271     dlc  = ( rx_header[1]       & 0xFU);
```

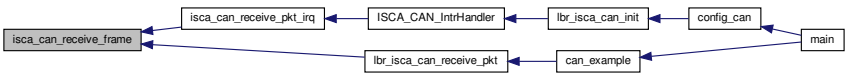
```
1272
1273     // Frame payload may be of variable size
1274     dlc = (dlc==0x8U) ? 0x8U : (dlc & 0x7U); //SCAN
1275     for (i = 0; i < dlc; i++) {
1276         rx_frame-> DATA[i] = ISCA_FPGA_Read8Bit(addr+
      can_rx_payload_addr[i]);
1277     } /*Read frame payload*/
1278
1279 #else
1280     /*Read frame header*/
1281     rx_header[0] = ISCA_FPGA_Read8Bit(addr+CAN_RX0_REG);
1282
1283     // Decode CAN frame header
1284     //ide = ((rx_header[0] & 0x80U) >> 5U);  // SCAN
1285     ide = ((rx_header[0] & 0x80U) >> 7U);
1286     rtr = ((rx_header[0] & 0x40U) >> 6U);
1287     dlc =  (rx_header[0] & 0x0FU);
1288     //dlc_l = (dlc>0x8U) ? 0x8U : dlc; // support SCAN
1289
1290     rx_frame->IDE = ide;
1291
1292     if (ide != can_ctrl->frm_md) {
1293         /* CAN frame received header with mode
1294          * different than the one CAN controller
1295          * is initialized to filter (basic <-> extended)
1296          * */
1300     }
1301
1302     // Read part of CAN header ID
1303     rx_header[1] = ISCA_FPGA_Read8Bit(addr+CAN_RX1_REG);
1304     rx_header[2] = ISCA_FPGA_Read8Bit(addr+CAN_RX2_REG);
1305     if (ide == CAN_FRAME_EXT) {
1306         // Read rest of CAN header ID
1307         rx_header[3] = ISCA_FPGA_Read8Bit(addr+CAN_RX3_REG);
1308         rx_header[4] = ISCA_FPGA_Read8Bit(addr+CAN_RX4_REG);
1309
1310         // Decode CAN frame header ID
1311         id |= (rx_header[1] << 21U);
1312         id |= (rx_header[2] << 13U);
1313         id |= (rx_header[3] << 5U);
1314         id |= ((rx_header[4] & 0xF8U) >> 3U);
1315
1316         // Frame payload may be of variable size
1317         for (i = 0; i < dlc; i++) {
1318             rx_frame-> DATA[i] = ISCA_FPGA_Read8Bit(addr+
      can_rx_ext_payload_addr[i]);
1319         }
1320     }
1321     else {
1322
1323         id |= (rx_header[1] << 3U);
1324         id |= ((rx_header[2] & 0xE0U) >> 5U);
1325
1326         // Frame payload may be of variable size
1327         for (i = 0; i < dlc; i++) {
1328             rx_frame->DATA[i] = ISCA_FPGA_Read8Bit(addr+
      can_rx_payload_addr[i]);
1329         }
```

```
1330
1331     }
1332 #endif // !(CAN_MODE == CAN_2B)
1333
1334     rx_frame-> ID  = id;
1335     rx_frame-> RTR = rtr;
1336     rx_frame-> DLC = dlc;
1337
1338     //Acknowledge RX buff read, should decrease frames counter
1339     ISCA_FPGA_Write8Bit(addr+CAN_COMMAND_RX_REG,
      CAN_CMNT_RX_ACK);
1340
1341     //Return remain frames
1342     return (remain_frames-1);
1343 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.13.3.4 isca_can_switch_mode()

```
int isca_can_switch_mode (
          can_ctrl_s * can_ctrl,
          uint8_t reset_mode )
```

Switch can controller's reset mode on/off.

Parameters

in	*can_ctrl*	CAN controller struct pointer
in	*reset*	mode ISCA_CAN_MODE_RESET_OFF or ISCA_CAN_MODE_RESET_ON

Returns

Controller's previous CAN_MODE0_REG value
ISCA_CAN_INV_RST_MODE

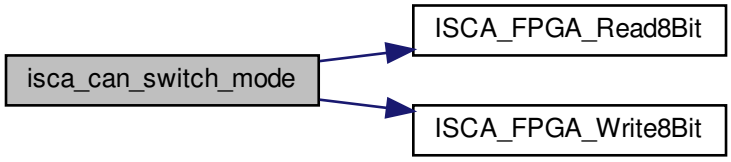
Definition at line 1353 of file ISCA_CAN.c.

References can_controller_s::addr, CAN_MODE0_REG, ISCA_CAN_INV_RS↩
T_MODE, ISCA_FPGA_Read8Bit(), and ISCA_FPGA_Write8Bit().

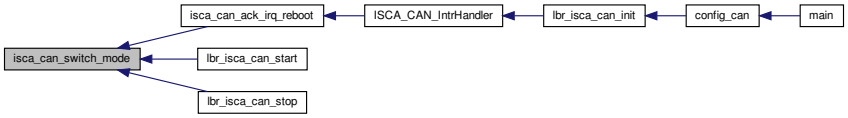
Referenced by isca_can_ack_irq_reboot(), lbr_isca_can_start(), and lbr_isca_↩
can_stop().

```
1354 {
1355
1356     uint8_t prev_mode;
1357     size_t  addr = can_ctrl->addr;
1358
1359     prev_mode = ISCA_FPGA_Read8Bit(addr+CAN_MODE0_REG);
1360
1361     if ((prev_mode & 0x1U) != (reset_mode & 0x1U)) {
1362 #if !(CAN_MODE == CAN_2B)
1363         /*mode register also includes the IRQs mode in 2A mode*/
1364         ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG, (prev_mode&0x01EU)|(
      reset_mode&0x1U));
1365 #else
1366         ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG, (prev_mode&0x0EU)|(
      reset_mode&0x1U));
1367 #endif // !(CAN_MODE == CAN_2B)
1368     }
1369     else {
1370         return ISCA_CAN_INV_RST_MODE;
1371     }
1373     //return the CAN_MODE0_REG value before update, if any
1374     return prev_mode;
1375 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.13.3.5 isca_can_set_filter()

```
void isca_can_set_filter (
          can_ctrl_s * can_ctrl )
```

CAN controller set filter function.

Parameters

in	*can_ctrl*	32bit mask

Returns

None At least can_ctrl->addr, can_ctrl->mask and can_ctrl->code should have been set

Definition at line 1384 of file ISCA_CAN.c.

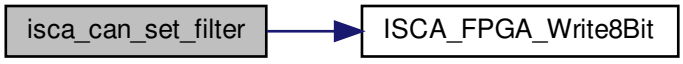
References can_controller_s::addr, CAN_CODE_REG0, CAN_CODE_REG1, CAN_CODE_REG2, CAN_CODE_REG3, CAN_FILTER_MODE_REG, CAN_FRAME_EXT, CAN_FRAME_STD, CAN_MASK_REG0, CAN_MASK_REG1, CAN_MASK_REG2, CAN_MASK_REG3, can_controller_s::code, can_controller_s::frm_md, ISCA_FPGA_Write8Bit(), and can_controller_s::mask.

Referenced by isca_can_init().

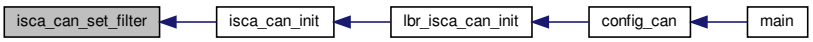
```
1385 {
1386
1387     size_t    addr = can_ctrl->addr;
1388     uint32_t  code = can_ctrl->code,
1389               mask = can_ctrl->mask;
1390
1391     // Need to enter filter mode since can version 2.2
1392     // Enable filter mode
1393     ISCA_FPGA_Write8Bit(addr+CAN_FILTER_MODE_REG, 0x1U);
1394
1395     // Set mask and code registers
1396 #if !(CAN_MODE == CAN_2B) /*Standard mode*/
1397     // CAN_CODE_REG0[7:0] -> CODE(ID)[10:3] */
1398     ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG0, (uint8_t)((code & 0x7F8U) >> 3U
      ));
1399
1400     // CAN_CODE_REG1[2:0] -> CODE(ID)[2:0] */
1401     ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG1, (uint8_t)(code & 0x7U));
1402
1403     // CAN_MASK_REG0[7:0] -> MASK(ID)[10:3] */
1404     ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG0, (uint8_t)((mask & 0x7F8U) >> 3U
      ));
1405
1406     // CAN_MASK_REG1[2:0] -> MASK(ID)[2:0] */
1407     ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG1, (uint8_t)(mask & 0x7U));
1408
1409 #else /*Extended mode*/
1410     switch (can_ctrl->frm_md) {
1411         case CAN_FRAME_EXT /*extended ID single filtering*/ :
1412             //CAN_CODE_REG0[7:0] = ID[28:21]
1413             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG0, (uint8_t)(code >>21U));
1414             //CAN_CODE_REG1[7:0] = ID[20:13]
1415             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG1, (uint8_t)(code >>13U));
1416             //CAN_CODE_REG2[7:0] = ID[12:5]
1417             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG2, (uint8_t)(code >> 5U));
1418             //CAN_CODE_REG3[7:3] = ID[4:0], CAN_CODE_REG3[2] = rtr2
1419             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG3, (uint8_t)((code&0x1FU)
      << 3U));
1420
1421             //CAN_MASK_REG0[7:0] = ID[28:21]
1422             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG0, (uint8_t)(mask >>21U));
1423             //CAN_MASK_REG1[7:0] = ID[20:13]
1424             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG1, (uint8_t)(mask >>13U));
1425             //CAN_MASK_REG2[7:0] = ID[12:5]
1426             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG2, (uint8_t)(mask >> 5U));
1427             //CAN_MASK_REG3[7:3] = ID[4:0], CAN_MASK_REG3[2] = rtr2 (='1' always allow)
1428             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG3, (uint8_t)((mask&0x1FU)
```

```
      << 3U)|0x4U);
1429
1430            break;
1431
1432        case CAN_FRAME_STD /*standard ID single filtering*/ :
1433            //CAN_CODE_REG0[7:0] = ID[10:3]
1434            ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG0, (uint8_t)(code >> 3U));
1435            //CAN_CODE_REG1[7:5] = ID[2:0]
1436            ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG1, (uint8_t)((code&0x7U)
      << 5U));
1437
1438            //CAN_MASK_REG0[7:0] = ID[10:3]
1439            ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG0, (uint8_t)(mask >> 3U));
1440            //CAN_MASK_REG1[7:5] = ID[2:0], set never check RTR (CAN_MASK_REG1[4])
1441            ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG1, (uint8_t)(((mask&0x7U)
      << 5U)) | 0x10U);
1442
1443            /*Never check for data fields (i.e.payload); set mask to '1'*/
1444            //CAN_MASK_REG2[7:0] = DATA0[7:0]
1445            ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG2, (uint8_t)(0xFFU));
1446            //CAN_MASK_REG3[7:0] = DATA1[7:0]
1447            ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG3, (uint8_t)(0xFFU));
1448
1449            break;
1450
1451        default:
1452            break;
1453    }
1454
1455    // Disable filter mode
1456    ISCA_FPGA_Write8Bit(addr+CAN_FILTER_MODE_REG, 0x0U);
1457
1458 #endif // !(CAN_MODE == CAN_2B)
1459
1460 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.13.4 Variable Documentation

A.13.4.1 can_tx_ext_payload_addr

```
uint8_t const can_tx_ext_payload_addr[] [static]
```

Initial value:

```
= {
        CAN_TX5_REG,  CAN_TX6_REG,
        CAN_TX7_REG,  CAN_TX8_REG,
        CAN_TX9_REG,  CAN_TX10_REG,
        CAN_TX11_REG, CAN_TX12_REG
}
```

TX payload buffers registers when can_frame_s::IDE == CAN_FRAME_EXT

Definition at line 965 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.4.2 can_rx_ext_payload_addr

```
uint8_t const can_rx_ext_payload_addr[] [static]
```

Initial value:

```
= {
        CAN_RX5_REG,  CAN_RX6_REG,
        CAN_RX7_REG,  CAN_RX8_REG,
        CAN_RX9_REG,  CAN_RX10_REG,
        CAN_RX11_REG, CAN_RX12_REG
}
```

RX payload buffers registers when can_frame_s::IDE == CAN_FRAME_EXT

Definition at line 975 of file ISCA_CAN.c.

Referenced by isca_can_receive_frame().

A.13.4.3 can_tx_payload_addr

```
uint8_t const can_tx_payload_addr[] [static]
```

Initial value:

```
= {
        CAN_TX3_REG,  CAN_TX4_REG,
        CAN_TX5_REG,  CAN_TX6_REG,
        CAN_TX7_REG,  CAN_TX8_REG,
        CAN_TX9_REG, CAN_TX10_REG
}
```

TX payload buffers registers when can_frame_s::IDE == CAN_FRAME_STD

Definition at line 985 of file ISCA_CAN.c.

Referenced by isca_can_transmit_frame().

A.13.4.4 can_rx_payload_addr

```
uint8_t const can_rx_payload_addr[] [static]
```

Initial value:

```
= {
        CAN_RX3_REG,  CAN_RX4_REG,
        CAN_RX5_REG,  CAN_RX6_REG,
        CAN_RX7_REG,  CAN_RX8_REG,
        CAN_RX9_REG,  CAN_RX10_REG
}
```

RX payload buffers registers when can_frame_s::IDE == CAN_FRAME_STD

Definition at line 995 of file ISCA_CAN.c.

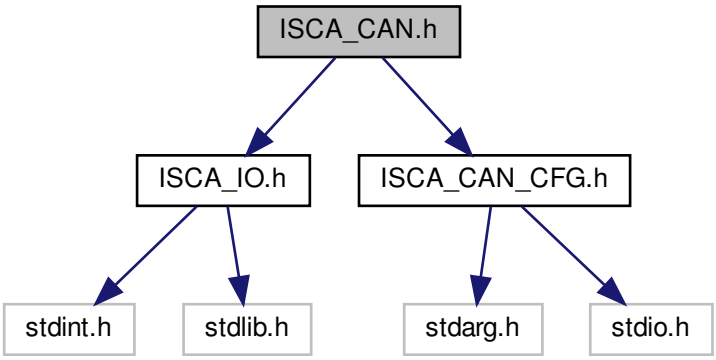
Referenced by isca_can_receive_frame().

A.14 ISCA_CAN.h File Reference

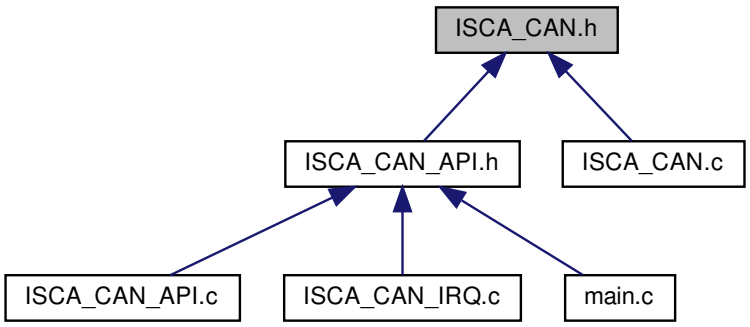
Provides functions to communicate with the CAN controller.

```
#include "ISCA_IO.h"
#include "ISCA_CAN_CFG.h"
```

Include dependency graph for ISCA_CAN.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct can_frame_s

Extended mode CAN frame description structure.

- union can_irq_en_u
  CAN controller IRQ enable union.
- struct can_controller_s
  CAN controller description structure.

Macros

- #define ISCA_CAN_OK (0)
- #define ISCA_CAN_BUSY (-1)
- #define ISCA_CAN_INV_REQ_TYPE (-2)
- #define ISCA_CAN_RX_FIFO_EMPTY (-3)
- #define ISCA_CAN_INV_ID_TYPE (-4)
- #define ISCA_CAN_INV_DLC (-5)
- #define ISCA_CAN_INV_RESET_MODE (-6)
- #define ISCA_CAN_QUEUES_OCCUPIED (-7)
- #define ISCA_CAN_INV_RST_MODE (-8)
- #define ISCA_CAN_ERROR (-9)
- #define CAN_REQ_BLOCKING (0U)
- #define CAN_REQ_NONBLOCKING (1U)
- #define ISCA_CAN_MODE_RESET_OFF (0U)
- #define ISCA_CAN_MODE_RESET_ON (1U)
- #define CAN_IRQ_OFF (0U)
- #define CAN_IRQ_ON (1U)
- #define CAN_FRAME_STD (0U)
- #define CAN_FRAME_EXT (1U)

Typedefs

- typedef struct can_frame_s can_frame_s
  Extended mode CAN frame description structure.
- typedef union can_irq_en_u irq_en
  CAN controller IRQ enable union.
- typedef struct can_controller_s can_ctrl_s
  CAN controller description structure.

Functions

- int isca_can_init (can_ctrl_s ∗can_ctrl)

CAN controller initialize function.

- int isca_can_transmit_frame (can_ctrl_s ∗can_ctrl, can_frame_s ∗tx_frame, uint8_t req_type)
  CAN controller transmit frame.
- int isca_can_receive_frame (can_ctrl_s ∗can_ctrl, can_frame_s ∗rx_frame, uint8_t req_type)
  CAN controller receive frame.
- void isca_can_set_filter (can_ctrl_s ∗can_ctrl)
  CAN controller set filter function.
- int isca_can_switch_mode (can_ctrl_s ∗can_ctrl, uint8_t reset_mode)
  Switch can controller's reset mode on/off.

A.14.1 Detailed Description

Provides functions to communicate with the CAN controller.

Version

1.0

A.14.2 Macro Definition Documentation

A.14.2.1 ISCA_CAN_OK

```
#define ISCA_CAN_OK ( 0)
```

CAN generic success return value

Definition at line 140 of file ISCA_CAN.h.

Referenced by isca_can_init(), isca_can_transmit_frame(), lbr_isca_can_↩ receive_pkt(), and main().

A.14.2.2 ISCA_CAN_BUSY

```
#define ISCA_CAN_BUSY (-1)
```

CAN controller return value

Definition at line 141 of file ISCA_CAN.h.

Referenced by isca_can_transmit_frame().

A.14.2.3 ISCA_CAN_INV_REQ_TYPE

```
#define ISCA_CAN_INV_REQ_TYPE (-2)
```

CAN invalid request type return value

Definition at line 142 of file ISCA_CAN.h.

Referenced by isca_can_receive_frame(), and isca_can_transmit_frame().

A.14.2.4 ISCA_CAN_RX_FIFO_EMPTY

```
#define ISCA_CAN_RX_FIFO_EMPTY (-3)
```

CAN RX FIFO empty return value

Definition at line 143 of file ISCA_CAN.h.

Referenced by isca_can_receive_frame(), and isca_can_receive_pkt_irq().

A.14.2.5 ISCA_CAN_INV_ID_TYPE

```
#define ISCA_CAN_INV_ID_TYPE (-4)
```

CAN invalid can_frame_s ID type return value

Definition at line 144 of file ISCA_CAN.h.

A.14.2.6 ISCA_CAN_INV_DLC

```
#define ISCA_CAN_INV_DLC (-5)
```

CAN invalid can_frame_s DLC type return value

Definition at line 145 of file ISCA_CAN.h.

A.14.2.7 ISCA_CAN_INV_RESET_MODE

```
#define ISCA_CAN_INV_RESET_MODE (-6)
```

CAN invalid reset mode

See also

ISCA_CAN_MODE_RESET_OFF
ISCA_CAN_MODE_RESET_ON

Definition at line 146 of file ISCA_CAN.h.

A.14.2.8 ISCA_CAN_QUEUES_OCCUPIED

```
#define ISCA_CAN_QUEUES_OCCUPIED (-7)
```

CAN all SW RX queues occupied

Definition at line 147 of file ISCA_CAN.h.

Referenced by lbr_isca_can_init().

A.14.2.9 ISCA_CAN_INV_RST_MODE

```
#define ISCA_CAN_INV_RST_MODE (-8)
```

CAN invalid reset mode ordered

Definition at line 148 of file ISCA_CAN.h.

Referenced by isca_can_switch_mode(), lbr_isca_can_start(), and lbr_isca_↩ can_stop().

A.14.2.10 ISCA_CAN_ERROR

```
#define ISCA_CAN_ERROR (-9)
```

CAN error

Definition at line 149 of file ISCA_CAN.h.

Referenced by main().

A.14.2.11 CAN_REQ_BLOCKING

```
#define CAN_REQ_BLOCKING (0U)
```

Keep polling the controller until request satisfied

Definition at line 154 of file ISCA_CAN.h.

Referenced by isca_can_receive_frame(), isca_can_transmit_frame(), lbr_isca↩ _can_receive_pkt(), and lbr_isca_can_transmit_pkt().

A.14.2.12 CAN_REQ_NONBLOCKING

```
#define CAN_REQ_NONBLOCKING (1U)
```

Send request to the controller only once

Definition at line 155 of file ISCA_CAN.h.

Referenced by isca_can_receive_frame(), isca_can_receive_pkt_irq(), and isca↩
_can_transmit_frame().

A.14.2.13 ISCA_CAN_MODE_RESET_OFF

```
#define ISCA_CAN_MODE_RESET_OFF (0U)
```

Switch CAN SW controlled reset mode off

Definition at line 160 of file ISCA_CAN.h.

Referenced by isca_can_ack_irq_reboot(), isca_can_init(), and lbr_isca_can_↩
start().

A.14.2.14 ISCA_CAN_MODE_RESET_ON

```
#define ISCA_CAN_MODE_RESET_ON (1U)
```

Switch CAN SW controlled reset mode on

Definition at line 161 of file ISCA_CAN.h.

Referenced by isca_can_ack_irq_reboot(), isca_can_init(), and lbr_isca_can_↩
stop().

A.14.2.15 CAN_IRQ_OFF

```
#define CAN_IRQ_OFF (0U)
```

Switch CAN interrupts off

Definition at line 166 of file ISCA_CAN.h.

Referenced by config_can().

A.14.2.16 CAN_IRQ_ON

```
#define CAN_IRQ_ON (1U)
```

Switch CAN interrupts on

Definition at line 167 of file ISCA_CAN.h.

Referenced by config_can(), isca_can_init(), lbr_isca_can_init(), and lbr_isca_↩ can_receive_pkt().

A.14.2.17 CAN_FRAME_STD

```
#define CAN_FRAME_STD (0U)
```

CAN extended mode, basic frame (11-bit header)

Definition at line 174 of file ISCA_CAN.h.

Referenced by isca_can_set_filter().

A.14.2.18 CAN_FRAME_EXT

```
#define CAN_FRAME_EXT (1U)
```

CAN extended mode, extended frame (29-bit header)

Definition at line 175 of file ISCA_CAN.h.

Referenced by can_example(), config_can(), isca_can_receive_frame(), isca_↩ can_set_filter(), and isca_can_transmit_frame().

A.14.3 Typedef Documentation

A.14.3.1 can_frame_s

```
typedef struct can_frame_s can_frame_s
```

Extended mode CAN frame description structure.

A.14.3.2 irq_en

```
typedef union can_irq_en_u irq_en
```

CAN controller IRQ enable union.

A.14.3.3 can_ctrl_s

```
typedef struct can_controller_s can_ctrl_s
```

CAN controller description structure.

A.14.4 Function Documentation

A.14.4.1 isca_can_init()

```
int isca_can_init (
          can_ctrl_s * can_ctrl )
```

CAN controller initialize function.

Parameters

in	*can_ctrl*	CAN controller instance pointer

Returns

ISCA_CAN_OK

See also

CAN_MODE0_REG
CAN_IRQS_EN_REG

Definition at line 1013 of file ISCA_CAN.c.

References can_controller_s::addr, can_controller_s::brp, CAN_BTR0_REG, CAN_BTR1_REG, CAN_CDR_REG, CAN_CLK_DIV, CAN_CLK_O_OFF, CAN_IRQ_ON, CAN_IRQS_EN_REG, CAN_MODE0_REG, CAN_TRIPLE_SAM, can_irq_en_u::err0, can_irq_en_u::err1, can_irq_en_u::err2, can_irq_en_u::err3, can_irq_en_u::err4, can_controller_s::irq, can_controller_s::irqs_en, ISCA_CAN_MODE_RESET_OFF, ISCA_CAN_MODE_RESET_ON, ISCA_CAN_OK, isca_can_set_filter(), ISCA_FPGA_Write8Bit(), can_irq_en_u::rx, can_controller_s::sjw, can_controller_s::tseg1, can_controller_s::tseg2, and can_irq_en_u::tx.

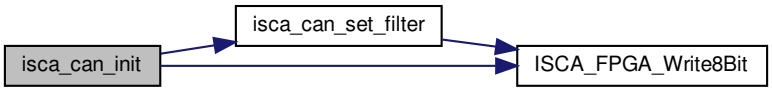
Referenced by lbr_isca_can_init().

```
1014 {
1015
1016 #if (CAN_MODE == CAN_2B)
1017     uint8_t irqs;
1018 #endif // (CAN_MODE == CAN_2B)
```

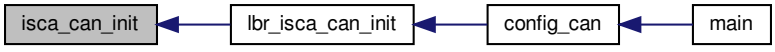
```
1019
1020     uint8_t btr0;
1021     uint8_t btr1;
1022     uint8_t cdr;
1023     uint8_t cfg0;
1024     size_t  addr = can_ctrl->addr;
1025
1026     // Switch on can controller's reset mode, and clear ext_mode values
1027     ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG,
      ISCA_CAN_MODE_RESET_ON);
1028
1029     /* Set bus timing register 1*/
1030     btr0  = 0x0U;
1031     btr0 |= (can_ctrl->sjw << 6U);
1032     btr0 |= (can_ctrl->brp & 0x3F);
1033     ISCA_FPGA_Write8Bit(addr+CAN_BTR0_REG, btr0);
1034
1035     /* Set bus timing register 2*/
1036     btr1  = 0x0U;
1037     btr1 |= (CAN_TRIPLE_SAM << 7U);
1038     btr1 |= ((can_ctrl->tseg2 & 0x7U) << 4U);
1039     btr1 |=  (can_ctrl->tseg1 & 0xFU);
1040
1041     ISCA_FPGA_Write8Bit(addr+CAN_BTR1_REG, btr1);
1042
1043     /* Set bus timing register 1*/
1044     cdr  = 0x0U;
1045     cdr |= (CAN_CLK_O_OFF  << 3U);
1046     cdr |= (CAN_CLK_DIV    &  7U);
1047 #if (CAN_MODE == CAN_2B)
1048     // Enable CAN 2B (extended) mode*/
1049     cdr |= (0x1U << 7U);
1050 #endif // !(CAN_MODE == CAN_2B)
1051
1052     ISCA_FPGA_Write8Bit(addr+CAN_CDR_REG, cdr);
1053
1054     /* Set up can filter*/
1055 #if !(CAN_MODE == CAN_2B)
1056     isca_can_set_filter(can_ctrl);
1057 #else
1058     isca_can_set_filter(can_ctrl);
1059 #endif // !(CAN_MODE == CAN_2B)
1060
1061 #if !(CAN_MODE == CAN_2B)
1062     /* Set CAN mode register*/
1063     cfg0  = ISCA_CAN_MODE_RESET_OFF;
1064     if ( can_ctrl->irq == CAN_IRQ_ON ) {
1065         cfg0 |= ((can_ctrl->irqs_en.err0&1U) << 0x4U); // overrun IRQ
1066         cfg0 |= ((can_ctrl->irqs_en.err1&1U) << 0x3U); // error IRQ
1067         cfg0 |= ((can_ctrl->irqs_en.tx&1U)   << 0x2U); // transmit IRQ
1068         cfg0 |= ((can_ctrl->irqs_en.rx&1U)   << 0x1U); // receive IRQ
1069     }
1071     ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG, cfg0);
1072
1073 #else
1074
1075     /* Set CAN mode register*/
1076     cfg0  = ISCA_CAN_MODE_RESET_OFF;
```

```
1077     cfg0 |= 0x8U; //dual filtering not supported in current version
1078     ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG, cfg0|0x8U);
1079
1080     irqs = 0U;
1081     if ( can_ctrl->irq == CAN_IRQ_ON ) {
1082
1083         /* Activate IRQs as ordered*/
1084         irqs |= ((can_ctrl->irqs_en.err4&1U) << 0x7U); /* bus error IRQ */
1085         irqs |= ((can_ctrl->irqs_en.err3&1U) << 0x6U); /* arbitration lost IRQ */
1086         irqs |= ((can_ctrl->irqs_en.err2&1U) << 0x5U); /* error passive IRQ */
1087         irqs |= ((can_ctrl->irqs_en.err1&1U) << 0x3U); /* overrun IRQ */
1088         irqs |= ((can_ctrl->irqs_en.err0&1U) << 0x2U); /* error IRQ */
1089         irqs |= ((can_ctrl->irqs_en.tx&1U)   << 0x1U); /* transmit IRQ */
1090         irqs |= ((can_ctrl->irqs_en.rx&1U)   << 0x0U); /* receive IRQ */
1091
1092     }
1094     ISCA_FPGA_Write8Bit(addr+CAN_IRQS_EN_REG, irqs);
1095
1096 #endif // !(CAN_MODE == CAN_2B)
1097
1098     return ISCA_CAN_OK;
1099 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.14.4.2 isca_can_transmit_frame()

```
int isca_can_transmit_frame (
            can_ctrl_s * can_ctrl,
```

```
can_frame_s * tx_frame,
uint8_t req_type )
```

CAN controller transmit frame.

Precondition

CAN controller initialization

Parameters

in	*can_ctrl*	CAN controller struct
in	*tx_frame*	CAN frame struct pointer
in	*req_type*	/ref CAN_REF_BLOCKING or CAN_REF_NONBLOCKING

Returns

ISCA_CAN_BUSY
ISCA_CAN_INV_IO_TYPE
ISCA_CAN_OK

<

Todo Sleep while TX requested and controller is busy

Definition at line 1111 of file ISCA_CAN.c.

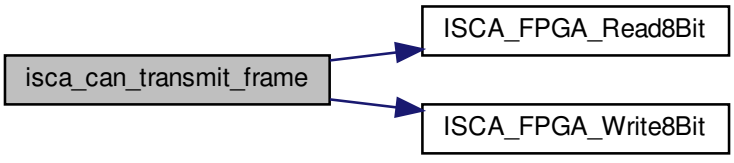
References can_controller_s::addr, CAN_CMNT_TRIGGER_TX, CAN_COMM↩
AND_TX_REG, CAN_FRAME_EXT, CAN_Q_TX_BUF_STATUS, CAN_REQ_↩
BLOCKING, CAN_REQ_NONBLOCKING, CAN_STATUS_REG, CAN_TX0_R↩
EG, CAN_TX1_REG, CAN_TX2_REG, CAN_TX3_REG, CAN_TX4_REG, can↩
_tx_ext_payload_addr, can_tx_payload_addr, can_frame_s::DATA, can_frame↩
_s::DLC, can_frame_s::ID, can_frame_s::IDE, ISCA_CAN_BUSY, ISCA_CAN↩
_INV_REQ_TYPE, ISCA_CAN_OK, ISCA_FPGA_Read8Bit(), ISCA_FPGA_↩
Write8Bit(), and can_frame_s::RTR.

Referenced by lbr_isca_can_transmit_pkt().

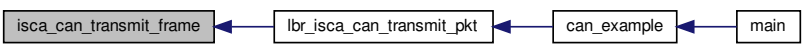
```
1112 {
1113
1114 #if !(CAN_MODE == CAN_2B)
1115     uint8_t tx_header[2] = {0x0U, 0x0U};
1116 #else
1117     uint8_t  tx_header[5] = {0x0U, 0x0U, 0x0U, 0x0U, 0x0U};
1118 #endif // !(CAN_MODE == CAN_2B)
1119
1120     uint8_t i;
1121     size_t  addr = can_ctrl->addr;
1122
1123     if (req_type == CAN_REQ_NONBLOCKING) {
1124         if (CAN_Q_TX_BUF_STATUS(ISCA_FPGA_Read8Bit(addr+
      CAN_STATUS_REG))) {
1125             return ISCA_CAN_BUSY;
1126         }
1127     }
1128     else if (req_type == CAN_REQ_BLOCKING) {
1129         while(CAN_Q_TX_BUF_STATUS(ISCA_FPGA_Read8Bit(addr+
      CAN_STATUS_REG))) {
1130             //TODO: sleep
1132         }; /*tx busy*/
1133     }
1134     else {
1135         return ISCA_CAN_INV_REQ_TYPE;
1136     }
1139 #if !(CAN_MODE == CAN_2B)
1140     tx_header[0] |= (tx_frame->ID  & 0x7F8U) >> 3U; //ID[10:3]
1141     ISCA_FPGA_Write8Bit(addr+CAN_TX0_REG, tx_header[0]);
1142
1143     tx_header[1] |= (tx_frame->ID  & 0x7U)   << 5U; //ID[2:0]
1144     tx_header[1] |= (tx_frame->RTR & 0x1U)   << 4U;
1145     tx_header[1] |= (tx_frame->DLC & 0xFU);
1146     ISCA_FPGA_Write8Bit(addr+CAN_TX1_REG, tx_header[1]);
1147 #else
1148     //tx_header[0] |= (tx_frame->IDE & 0x4U) << 5U; // SCAN
1149     tx_header[0] |= (tx_frame->IDE & 0x1U) << 7U;
1150     tx_header[0] |= (tx_frame->RTR & 0x1U) << 6U;
1151     tx_header[0] |= (tx_frame->DLC & 0xFU);
1152     ISCA_FPGA_Write8Bit(addr+CAN_TX0_REG, tx_header[0]);
1153
1154     if (tx_frame->IDE == CAN_FRAME_EXT) {
1155         tx_header[1] = (tx_frame->ID & 0x1FE00000U) >> 21U;
1156         ISCA_FPGA_Write8Bit(addr+CAN_TX1_REG, tx_header[1]);
1157
1158         tx_header[2] = (tx_frame->ID & 0x1FE000U) >> 13U;
1159         ISCA_FPGA_Write8Bit(addr+CAN_TX2_REG, tx_header[2]);
1160
1161         tx_header[3] = (tx_frame->ID & 0x1FE0U) >> 5U;
1162         ISCA_FPGA_Write8Bit(addr+CAN_TX3_REG, tx_header[3]);
1163
1164         tx_header[4] = (tx_frame->ID & 0x1F) << 3U;
1165         ISCA_FPGA_Write8Bit(addr+CAN_TX4_REG, tx_header[4]);
1166     }
1167     else {
1168         tx_header[1] = (tx_frame->ID & 0x7F8) >> 3U;
1169         ISCA_FPGA_Write8Bit(addr+CAN_TX1_REG, tx_header[1]);
1170
```

```
1171         tx_header[2] = (tx_frame->ID & 0x7U) << 5U;
1172         ISCA_FPGA_Write8Bit(addr+CAN_TX2_REG, tx_header[2]);
1173     }
1174 #endif // !(CAN_MODE == CAN_2B)
1175 
1176 #if !(CAN_MODE == CAN_2B)
1177     // Frame payload may be of variable size
1178     for (i = 0; i < tx_frame->DLC; i++) {
1179         ISCA_FPGA_Write8Bit(addr+can_tx_payload_addr[i], tx_frame->
      DATA[i]);
1180     }
1181 #else
1182 
1183     if (tx_frame->IDE == CAN_FRAME_EXT) { /*CAN_2B extended frame ID*/
1184         /*Frame payload may be of variable size*/
1185         for (i = 0; i < tx_frame->DLC; i++) {
1186             ISCA_FPGA_Write8Bit(addr+can_tx_ext_payload_addr[i],
      tx_frame->DATA[i]);
1187         }
1188     } else { /*CAN_2B basic frame ID*/
1189         /*Frame payload may be of variable size*/
1190         for (i = 0; i < tx_frame->DLC; i++) {
1191             ISCA_FPGA_Write8Bit(addr+can_tx_payload_addr[i], tx_frame
      ->DATA[i]);
1192         }
1193     }
1194 #endif // !(CAN_MODE == CAN_2B)
1195 
1196     /*trigger TX*/
1197     ISCA_FPGA_Write8Bit(addr+CAN_COMMAND_TX_REG,
      CAN_CMNT_TRIGGER_TX);
1198 
1199     return ISCA_CAN_OK;
1200 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.14.4.3 isca_can_receive_frame()

```
int isca_can_receive_frame (
          can_ctrl_s * can_ctrl,
          can_frame_s * rx_frame,
          uint8_t io_type )
```

CAN controller receive frame.

Precondition

CAN controller initialization

Parameters

in	*can_ctrl*	CAN controller struct pointer
out	*rx_frame*	CAN frame struct pointer
in	*req_type*	CAN_IO_BLOCKING or CAN_IO_NONBLOCKING

Returns

ISCA_CAN_RX_FIFO_EMPTY
ISCA_CAN_INV_IO_TYPE
ISCA_CAN_OK

<

Todo Sleep while waiting for a frame to arrive

<

Todo Notify user if during CAN 2B mode and filtering is set to basic, an extended frame is has been received or vice versa

Definition at line 1212 of file ISCA_CAN.c.

References can_controller_s::addr, CAN_CMNT_RX_ACK, CAN_COMMAND↩
_RX_REG, CAN_FRAME_EXT, CAN_INVALID_RX_ACK_REG, CAN_Q_RX_↩
OVERRUN, CAN_REQ_BLOCKING, CAN_REQ_NONBLOCKING, CAN_RX0↩
_REG, CAN_RX1_REG, CAN_RX2_REG, CAN_RX3_REG, CAN_RX4_REG,

CAN_RX_COUNTER_REG, can_rx_ext_payload_addr, can_rx_payload_addr, CAN_STATUS_REG, can_frame_s::DATA, can_controller_s::frm_md, can_↩ frame_s::IDE, ISCA_CAN_INV_REQ_TYPE, ISCA_CAN_RX_FIFO_EMPTY, I↩ SCA_FPGA_Read8Bit(), and ISCA_FPGA_Write8Bit().

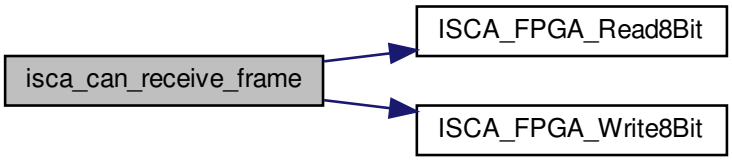
Referenced by isca_can_receive_pkt_irq(), and lbr_isca_can_receive_pkt().

```
1213 {
1214
1215 #if !(CAN_MODE == CAN_2B)
1216     uint8_t rx_header[2] = {0x0U, 0x0U};
1217 #else
1218     uint8_t rx_header[5] = {0x0U, 0x0U, 0x0U, 0x0U, 0x0U};
1219     uint8_t ide = 0x0U;
1220 #endif // !(CAN_MODE == CAN_2B)
1221
1222     int32_t  remain_frames = 0x0U;
1223     size_t   addr  = can_ctrl->addr;
1224     uint32_t id    = 0x0U;
1225     uint8_t  rtr   = 0x0U;
1226     uint8_t  dlc   = 0x0U;
1227     uint8_t  i;
1228
1229     if ( CAN_Q_RX_OVERRUN(ISCA_FPGA_Read8Bit(addr+
      CAN_STATUS_REG)) ) {
1230
1231         while (ISCA_FPGA_Read8Bit(addr+CAN_INVALID_RX_ACK_REG) != 0
      ) {
1232             /*Acknowledge RX buff, should decrease frames counter*/
1233             ISCA_FPGA_Write8Bit(addr+CAN_COMMAND_RX_REG,
      CAN_CMNT_RX_ACK);
1234         }
1236     }
1238     remain_frames = ISCA_FPGA_Read8Bit(addr+CAN_RX_COUNTER_REG);
1239
1240     if ( io_type == CAN_REQ_NONBLOCKING ) {
1241
1242         if ( remain_frames == 0x0U ) {
1243             return ISCA_CAN_RX_FIFO_EMPTY;
1244         }
1246     }
1247     else if (io_type == CAN_REQ_BLOCKING) {
1248
1249         while (remain_frames == 0x0U) {
1250             remain_frames = ISCA_FPGA_Read8Bit(addr+
      CAN_RX_COUNTER_REG);
1251             /*TODO: sleep*/
1253         }; /*wait for a frame to arrive*/
1254
1255     }
1256     else {
1257
1258         return ISCA_CAN_INV_REQ_TYPE;
1259
1260     } /*Invalid io_type*/
```

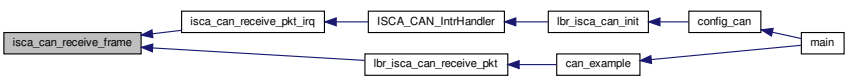
```
1261
1262 #if !(CAN_MODE == CAN_2B)
1263     // read frame header
1264     rx_header[0] = ISCA_FPGA_Read8Bit(addr+CAN_RX0_REG);
1265     rx_header[1] = ISCA_FPGA_Read8Bit(addr+CAN_RX1_REG);
1266
1268     id  |= ( (uint32_t)rx_header[0] << 3);
1269     id  |= ((rx_header[1] >> 5) & 0x7U);
1270     rtr  = ((rx_header[1] >> 4) & 0x1U);
1271     dlc  = ( rx_header[1]       & 0xFU);
1272
1273     // Frame payload may be of variable size
1274     dlc = (dlc==0x8U) ? 0x8U : (dlc & 0x7U); //SCAN
1275     for (i = 0; i < dlc; i++) {
1276         rx_frame-> DATA[i] = ISCA_FPGA_Read8Bit(addr+
      can_rx_payload_addr[i]);
1277     } /*Read frame payload*/
1278
1279 #else
1280     /*Read frame header*/
1281     rx_header[0] = ISCA_FPGA_Read8Bit(addr+CAN_RX0_REG);
1282
1283     // Decode CAN frame header
1284     //ide = ((rx_header[0] & 0x80U) >> 5U);  // SCAN
1285     ide = ((rx_header[0] & 0x80U) >> 7U);
1286     rtr = ((rx_header[0] & 0x40U) >> 6U);
1287     dlc =  (rx_header[0] & 0x0FU);
1288     //dlc_l = (dlc>0x8U) ? 0x8U : dlc; // support SCAN
1289
1290     rx_frame->IDE = ide;
1291
1292     if (ide != can_ctrl->frm_md) {
1293         /* CAN frame received header with mode
1294          * different than the one CAN controller
1295          * is initialized to filter (basic <-> extended)
1296          * */
1300     }
1301
1302     // Read part of CAN header ID
1303     rx_header[1] = ISCA_FPGA_Read8Bit(addr+CAN_RX1_REG);
1304     rx_header[2] = ISCA_FPGA_Read8Bit(addr+CAN_RX2_REG);
1305     if (ide == CAN_FRAME_EXT) {
1306         // Read rest of CAN header ID
1307         rx_header[3] = ISCA_FPGA_Read8Bit(addr+CAN_RX3_REG);
1308         rx_header[4] = ISCA_FPGA_Read8Bit(addr+CAN_RX4_REG);
1309
1310         // Decode CAN frame header ID
1311         id |= (rx_header[1] << 21U);
1312         id |= (rx_header[2] << 13U);
1313         id |= (rx_header[3] << 5U);
1314         id |= ((rx_header[4] & 0xF8U) >> 3U);
1315
1316         // Frame payload may be of variable size
1317         for (i = 0; i < dlc; i++) {
1318             rx_frame-> DATA[i] = ISCA_FPGA_Read8Bit(addr+
      can_rx_ext_payload_addr[i]);
1319         }
1320     }
```

```
1321     else {
1322 
1323         id |= (rx_header[1] << 3U);
1324         id |= ((rx_header[2] & 0xE0U) >> 5U);
1325 
1326         // Frame payload may be of variable size
1327         for (i = 0; i < dlc; i++) {
1328             rx_frame->DATA[i] = ISCA_FPGA_Read8Bit(addr+
      can_rx_payload_addr[i]);
1329         }
1330 
1331     }
1332 #endif // !(CAN_MODE == CAN_2B)
1333 
1334     rx_frame-> ID  = id;
1335     rx_frame-> RTR = rtr;
1336     rx_frame-> DLC = dlc;
1337 
1338     //Acknowledge RX buff read, should decrease frames counter
1339     ISCA_FPGA_Write8Bit(addr+CAN_COMMAND_RX_REG,
      CAN_CMNT_RX_ACK);
1340 
1341     //Return remain frames
1342     return (remain_frames-1);
1343 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.14.4.4 isca_can_set_filter()

```
void isca_can_set_filter (
          can_ctrl_s * can_ctrl )
```

CAN controller set filter function.

Parameters

in	*can_ctrl*	32bit mask

Returns

None At least can_ctrl->addr, can_ctrl->mask and can_ctrl->code should have been set

Definition at line 1384 of file ISCA_CAN.c.

References can_controller_s::addr, CAN_CODE_REG0, CAN_CODE_REG1, CAN_CODE_REG2, CAN_CODE_REG3, CAN_FILTER_MODE_REG, CAN_FRAME_EXT, CAN_FRAME_STD, CAN_MASK_REG0, CAN_MASK_REG1, CAN_MASK_REG2, CAN_MASK_REG3, can_controller_s::code, can_controller_s::frm_md, ISCA_FPGA_Write8Bit(), and can_controller_s::mask.

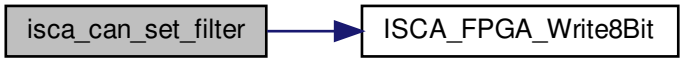
Referenced by isca_can_init().

```
1385 {
1386
1387     size_t    addr = can_ctrl->addr;
1388     uint32_t  code = can_ctrl->code,
1389               mask = can_ctrl->mask;
1390
1391     // Need to enter filter mode since can version 2.2
1392     // Enable filter mode
1393     ISCA_FPGA_Write8Bit(addr+CAN_FILTER_MODE_REG, 0x1U);
1394
1395     // Set mask and code registers
1396 #if !(CAN_MODE == CAN_2B) /*Standard mode*/
1397     // CAN_CODE_REG0[7:0] -> CODE(ID)[10:3] */
1398     ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG0, (uint8_t)((code & 0x7F8U) >> 3U
      ));
1399
1400     // CAN_CODE_REG1[2:0] -> CODE(ID)[2:0] */
1401     ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG1, (uint8_t)(code & 0x7U));
1402
1403     // CAN_MASK_REG0[7:0] -> MASK(ID)[10:3] */
1404     ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG0, (uint8_t)((mask & 0x7F8U) >> 3U
      ));
```

```
1405
1406     // CAN_MASK_REG1[2:0] -> MASK(ID)[2:0] */
1407     ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG1, (uint8_t)(mask & 0x7U));
1408
1409 #else /*Extended mode*/
1410     switch (can_ctrl->frm_md) {
1411         case CAN_FRAME_EXT /*extended ID single filtering*/ :
1412             //CAN_CODE_REG0[7:0] = ID[28:21]
1413             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG0, (uint8_t)(code >>21U));
1414             //CAN_CODE_REG1[7:0] = ID[20:13]
1415             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG1, (uint8_t)(code >>13U));
1416             //CAN_CODE_REG2[7:0] = ID[12:5]
1417             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG2, (uint8_t)(code >> 5U));
1418             //CAN_CODE_REG3[7:3] = ID[4:0], CAN_CODE_REG3[2] = rtr2
1419             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG3, (uint8_t)((code&0x1FU)
    << 3U));
1420
1421             //CAN_MASK_REG0[7:0] = ID[28:21]
1422             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG0, (uint8_t)(mask >>21U));
1423             //CAN_MASK_REG1[7:0] = ID[20:13]
1424             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG1, (uint8_t)(mask >>13U));
1425             //CAN_MASK_REG2[7:0] = ID[12:5]
1426             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG2, (uint8_t)(mask >> 5U));
1427             //CAN_MASK_REG3[7:3] = ID[4:0], CAN_MASK_REG3[2] = rtr2 (='1' always allow)
1428             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG3, (uint8_t)((mask&0x1FU)
    << 3U)|0x4U);
1429
1430             break;
1431
1432         case CAN_FRAME_STD /*standard ID single filtering*/ :
1433             //CAN_CODE_REG0[7:0] = ID[10:3]
1434             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG0, (uint8_t)(code >> 3U));
1435             //CAN_CODE_REG1[7:5] = ID[2:0]
1436             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG1, (uint8_t)((code&0x7U)
    << 5U));
1437
1438             //CAN_MASK_REG0[7:0] = ID[10:3]
1439             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG0, (uint8_t)(mask >> 3U));
1440             //CAN_MASK_REG1[7:5] = ID[2:0], set never check RTR (CAN_MASK_REG1[4])
1441             ISCA_FPGA_Write8Bit(addr+CAN_MASK_REG1, (uint8_t)(((mask&0x7U)
    << 5U)) | 0x10U);
1442
1443             /*Never check for data fields (i.e.payload); set mask to '1'*/
1444             //CAN_MASK_REG2[7:0] = DATA0[7:0]
1445             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG2, (uint8_t)(0xFFU));
1446             //CAN_MASK_REG3[7:0] = DATA1[7:0]
1447             ISCA_FPGA_Write8Bit(addr+CAN_CODE_REG3, (uint8_t)(0xFFU));
1448
1449             break;
1450
1451         default:
1452             break;
1453     }
1454
1455     // Disable filter mode
1456     ISCA_FPGA_Write8Bit(addr+CAN_FILTER_MODE_REG, 0x0U);
1457
1458 #endif // !(CAN_MODE == CAN_2B)
```

```
1459
1460 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

A.14.4.5 isca_can_switch_mode()

```
int isca_can_switch_mode (
            can_ctrl_s * can_ctrl,
            uint8_t reset_mode )
```

Switch can controller's reset mode on/off.

Parameters

in	*can_ctrl*	CAN controller struct pointer
in	*reset*	mode ISCA_CAN_MODE_RESET_OFF or ISCA_CAN_MODE_RESET_ON

Returns

Controller's previous CAN_MODE0_REG value
ISCA_CAN_INV_RST_MODE

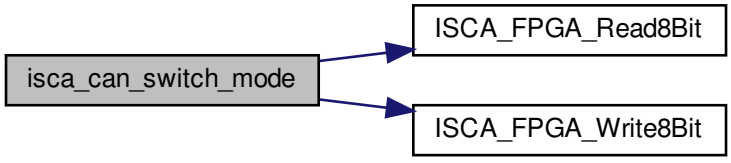
Definition at line 1353 of file ISCA_CAN.c.

References can_controller_s::addr, CAN_MODE0_REG, ISCA_CAN_INV_RS↩ T_MODE, ISCA_FPGA_Read8Bit(), and ISCA_FPGA_Write8Bit().

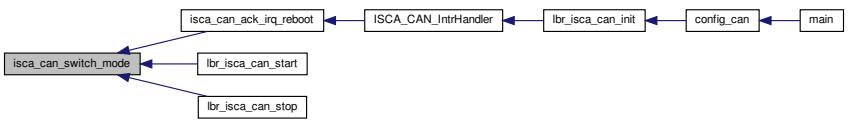
Referenced by isca_can_ack_irq_reboot(), lbr_isca_can_start(), and lbr_isca_↩ can_stop().

```
1354 {
1355
1356     uint8_t prev_mode;
1357     size_t  addr = can_ctrl->addr;
1358
1359     prev_mode = ISCA_FPGA_Read8Bit(addr+CAN_MODE0_REG);
1360
1361     if ((prev_mode & 0x1U) != (reset_mode & 0x1U)) {
1362 #if !(CAN_MODE == CAN_2B)
1363         /*mode register also includes the IRQs mode in 2A mode*/
1364         ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG, (prev_mode&0x01EU)|(
      reset_mode&0x1U));
1365 #else
1366         ISCA_FPGA_Write8Bit(addr+CAN_MODE0_REG, (prev_mode&0x0EU)|(
      reset_mode&0x1U));
1367 #endif // !(CAN_MODE == CAN_2B)
1368     }
1369     else {
1370         return ISCA_CAN_INV_RST_MODE;
1371     }
1373     //return the CAN_MODE0_REG value before update, if any
1374     return prev_mode;
1375 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

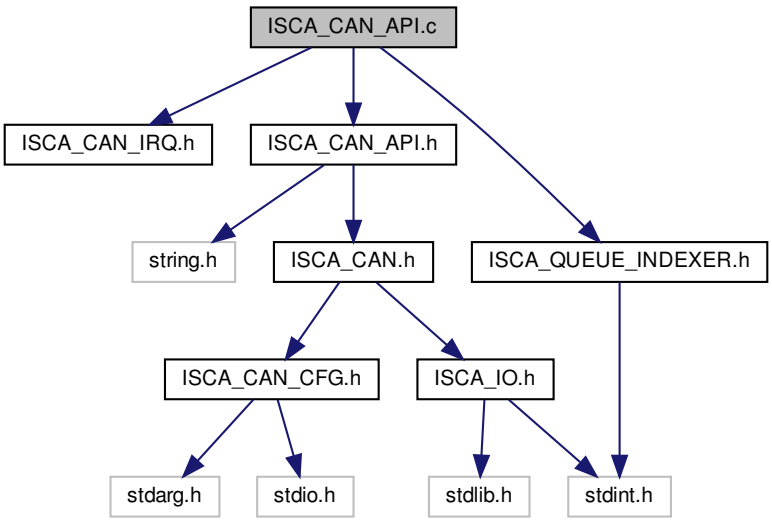


A.15 ISCA_CAN_API.c File Reference

Provides an application programming interface.

```
#include "ISCA_CAN_IRQ.h"
#include "ISCA_CAN_API.h"
#include "ISCA_QUEUE_INDEXER.h"
```

Include dependency graph for ISCA_CAN_API.c:



Functions

- int lbr_isca_can_init (can_ctrl_s *can_ctrl, int queue_slots)
  Initialize the CAN controller.
- int lbr_isca_can_receive_pkt (can_ctrl_s *can_ctrl, can_frame_s *rx_↩ frame)
- int lbr_isca_can_transmit_pkt (can_ctrl_s *can_ctrl, can_frame_s *tx_↩ frame)
- int lbr_isca_can_start (can_ctrl_s *can_ctrl)
- int lbr_isca_can_stop (can_ctrl_s *can_ctrl)

A.15.1 Detailed Description

Provides an application programming interface.

Version

1.0

A.15.2 Function Documentation

A.15.2.1 lbr_isca_can_init()

```
int lbr_isca_can_init (
          can_ctrl_s * can_ctrl,
          int queue_slots )
```

Initialize the CAN controller.

Parameters

in,out	*can_ctrl*	CAN controller instance pointer
in	*queue_slots*	Amount of slots to allocate for the SW RX queue

Returns

ISCA_CAN_OK
ISCA_CAN_QUEUES_OCCUPIED

Precondition

The following can_ctrl 's fields should have been initialized before calling this function

```
can_ctrl.addr
can_ctrl.brp
can_ctrl.tseg1
can_ctrl.tseg2
can_ctrl.sjw
can_ctrl.irq
can_ctrl.mask
can_ctrl.code
can_ctrl.frm_md //(if CAN_MODE==CAN_2B)
```

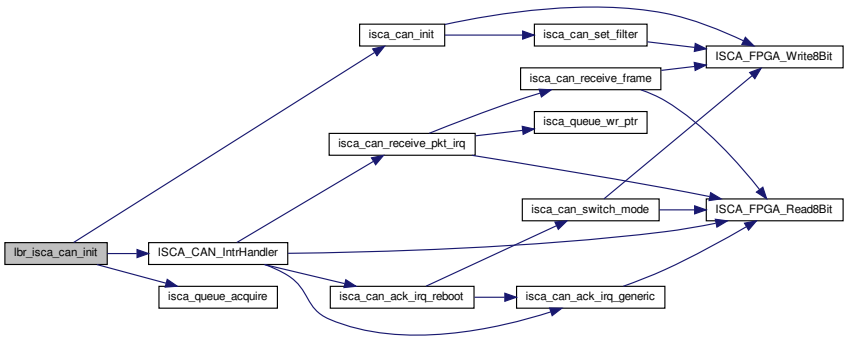
Definition at line 70 of file ISCA_CAN_API.c.

References CAN_IRQ_ON, DQ_OCCUPIED, can_controller_s::InterruptHandler, can_controller_s::irq, isca_can_init(), ISCA_CAN_IntrHandler(), ISCA_CAN↩ _QUEUES_OCCUPIED, isca_queue_acquire(), can_controller_s::q_id, can_↩ controller_s::q_ptr, can_controller_s::q_size, and queue_slots.

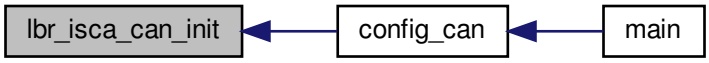
Referenced by config_can().

```
70                                                              {
71
72     int         can_rx_queue_id;
73     can_ctrl_s *can_controller_l;
74
75     /*Point local CAN controller to the passed; cast to can_ctrl_s**/
76     can_controller_l = (can_ctrl_s *)can_ctrl;
77
78     /*Create a CAN frames queue*/
79     can_rx_queue_id = isca_queue_acquire(queue_slots);
80
81     //if ( __builtin_expect(can_rx_queue_id == DQ_OCCUPIED, 0) ) {
82     if ( can_rx_queue_id == DQ_OCCUPIED ) {
83         return ISCA_CAN_QUEUES_OCCUPIED;
84     } /*No queue available*/
85
86     /*Update can controller struct*/
87     can_controller_l->q_ptr  = (can_frame_s *)malloc(sizeof(
       can_frame_s)*queue_slots);
88     can_controller_l->q_id   = can_rx_queue_id;
89     can_controller_l->q_size = queue_slots;
90
91     /*Set CAN controllers default interrupt callback function*/
92     if ( can_controller_l->irq == CAN_IRQ_ON ) {
93         can_controller_l->InterruptHandler = &
       ISCA_CAN_IntrHandler;
94     }
95
96     /*Setup CAN controller and return*/
97     return isca_can_init(can_ctrl);
98
99 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.15.2.2 lbr_isca_can_receive_pkt()

```
int lbr_isca_can_receive_pkt (
            can_ctrl_s * can_ctrl,
            can_frame_s * rx_frame )
```

Definition at line 109 of file ISCA_CAN_API.c.

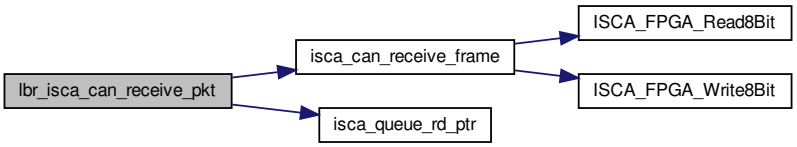
References CAN_IRQ_ON, CAN_REQ_BLOCKING, DQ_EMPTY, can_↩ controller_s::irq, ISCA_CAN_OK, isca_can_receive_frame(), isca_queue_rd↩ _ptr(), and can_controller_s::q_ptr.

Referenced by can_example().

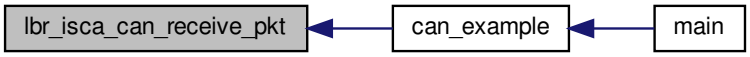
```
109                                                                    {
110
111     uint8_t queue_rd_index;
112     int q_idx;
113     int ret_val = ISCA_CAN_OK;
114
115     if (can_ctrl->irq == CAN_IRQ_ON) {
116
117         do {
118             q_idx = isca_queue_rd_ptr(((can_ctrl_s*)can_ctrl)->q_id, &
      queue_rd_index);
119         } while( q_idx == DQ_EMPTY ); /*Wait for the interrupt handler to push a frame*/
120
121         /*Interrupt routines use the created during initialization queue.

122          *Copy frame from  to user*/
123         memmove(rx_frame, &can_ctrl->q_ptr[queue_rd_index], sizeof(
      can_frame_s));
124     } /*Interrupt mode*/
125     else {
126
```

```
127         /*Poll until frame is available*/
128         ret_val = isca_can_receive_frame(can_ctrl, rx_frame,
      CAN_REQ_BLOCKING);
129
130     } /*Polling mode*/
131
132     return ret_val;
133
134 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.15.2.3 lbr_isca_can_transmit_pkt()

```
int lbr_isca_can_transmit_pkt (
            can_ctrl_s * can_ctrl,
            can_frame_s * tx_frame )
```

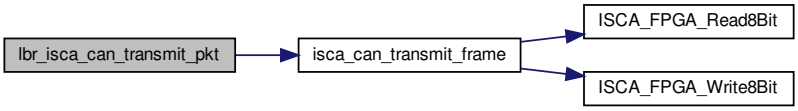
Definition at line 144 of file ISCA_CAN_API.c.

References CAN_REQ_BLOCKING, and isca_can_transmit_frame().

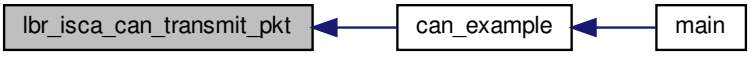
Referenced by can_example().

```
144                                                                    {
145
146     int ret_val;
147
148     ret_val = isca_can_transmit_frame(can_ctrl, tx_frame,
      CAN_REQ_BLOCKING);
149     return ret_val;
150
151 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.15.2.4 lbr_isca_can_start()

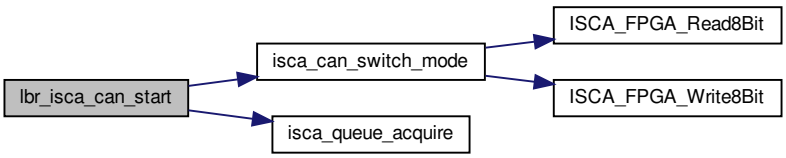
```
int lbr_isca_can_start (
          can_ctrl_s * can_ctrl )
```

Definition at line 160 of file ISCA_CAN_API.c.

References ISCA_CAN_INV_RST_MODE, ISCA_CAN_MODE_RESET_OFF, isca_can_switch_mode(), isca_queue_acquire(), can_controller_s::q_id, and can_controller_s::q_size.

```
160                                                 {
161
162     int ret_val;
163
164     //Switch off can controller's reset mode
165     ret_val = isca_can_switch_mode(can_ctrl,
      ISCA_CAN_MODE_RESET_OFF);
166
167     if ( ret_val != ISCA_CAN_INV_RST_MODE ) {
168         //Acquire queue
169         can_ctrl->q_id = isca_queue_acquire(can_ctrl->
      q_size);
170     }
171
172     return ret_val;
173 }
```

Here is the call graph for this function:



A.15.2.5 lbr_isca_can_stop()

```
int lbr_isca_can_stop (
            can_ctrl_s * can_ctrl )
```

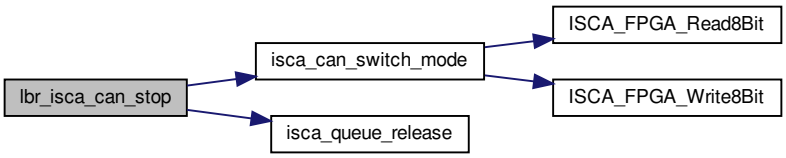
Definition at line 182 of file ISCA_CAN_API.c.

References ISCA_CAN_INV_RST_MODE, ISCA_CAN_MODE_RESET_ON, isca_can_switch_mode(), isca_queue_release(), and can_controller_s::q_id.

```
182                                                 {
183
184     int ret_val;
185
186     //Switch on can controller's reset mode
187     ret_val = isca_can_switch_mode(can_ctrl,
      ISCA_CAN_MODE_RESET_ON);
188
```

```
189     if ( ret_val != ISCA_CAN_INV_RST_MODE ) {
190         //Release queue
191         isca_queue_release(can_ctrl->q_id);
192     }
193 
194     return ret_val;
195 
196 }
```

Here is the call graph for this function:

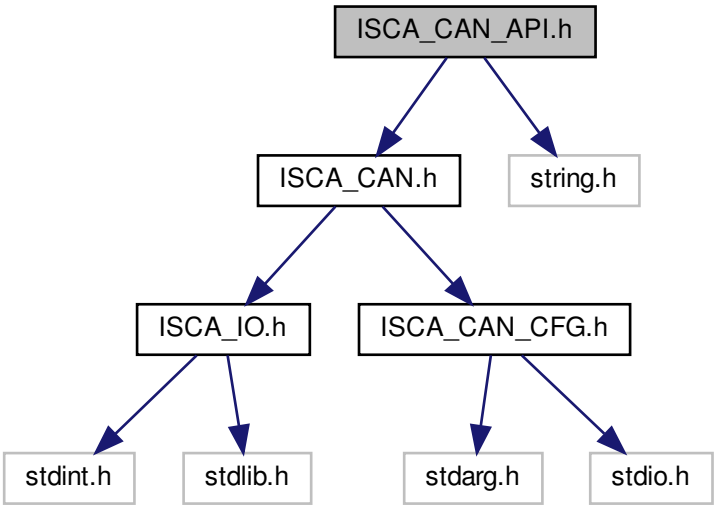


A.16 ISCA_CAN_API.h File Reference

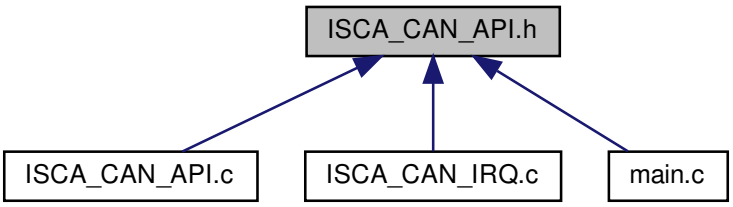
Provides an application programming interface.

```
#include "ISCA_CAN.h"
#include <string.h>
```

Include dependency graph for ISCA_CAN_API.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define LOCAL_NODE_ID (0xACU)
- #define ISCA_CAN_API_ENC (0U)
- #define ISCA_CAN_API_DEC (1U)

Functions

- int lbr_isca_can_init (can_ctrl_s ∗can_ctrl, int queue_slots)
  Initialize the CAN controller.
- int lbr_isca_can_start (can_ctrl_s ∗can_ctrl)
- int lbr_isca_can_stop (can_ctrl_s ∗can_ctrl)
- int lbr_isca_can_receive_pkt (can_ctrl_s ∗can_ctrl, can_frame_s ∗rx_↩ frame)
- int lbr_isca_can_transmit_pkt (can_ctrl_s ∗can_ctrl, can_frame_s ∗tx_↩ frame)

A.16.1 Detailed Description

Provides an application programming interface.

Version

1.0

A.16.2 Macro Definition Documentation

A.16.2.1 LOCAL_NODE_ID

```
#define LOCAL_NODE_ID (0xACU)
```

Local node ID

Definition at line 52 of file ISCA_CAN_API.h.

A.16.2.2 ISCA_CAN_API_ENC

```
#define ISCA_CAN_API_ENC (0U)
```

Encode constant

Definition at line 57 of file ISCA_CAN_API.h.

A.16.2.3 ISCA_CAN_API_DEC

```
#define ISCA_CAN_API_DEC (1U)
```

Decode constant

Definition at line 58 of file ISCA_CAN_API.h.

A.16.3 Function Documentation

A.16.3.1 lbr_isca_can_init()

```
int lbr_isca_can_init (
          can_ctrl_s * can_ctrl,
          int queue_slots )
```

Initialize the CAN controller.

Parameters

`in,out`	*can_ctrl*	CAN controller instance pointer
`in`	*queue_slots*	Amount of slots to allocate for the SW RX queue

Returns

ISCA_CAN_OK
ISCA_CAN_QUEUES_OCCUPIED

Precondition

The following can_ctrl 's fields should have been initialized before calling this function

```
can_ctrl.addr
can_ctrl.brp
can_ctrl.tseg1
can_ctrl.tseg2
can_ctrl.sjw
can_ctrl.irq
can_ctrl.mask
can_ctrl.code
can_ctrl.frm_md //(if CAN_MODE==CAN_2B)
```

Definition at line 70 of file ISCA_CAN_API.c.

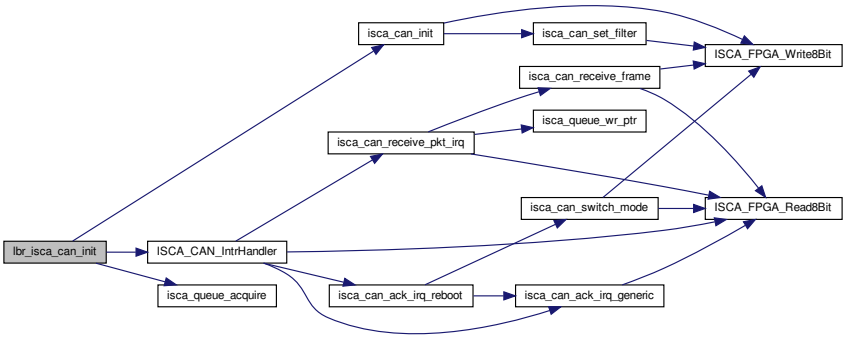
References CAN_IRQ_ON, DQ_OCCUPIED, can_controller_s::InterruptHandler, can_controller_s::irq, isca_can_init(), ISCA_CAN_IntrHandler(), ISCA_CAN↩ _QUEUES_OCCUPIED, isca_queue_acquire(), can_controller_s::q_id, can_↩ controller_s::q_ptr, can_controller_s::q_size, and queue_slots.

Referenced by config_can().

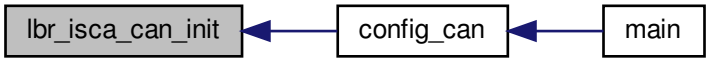
```
70                                                              {
71
72     int         can_rx_queue_id;
73     can_ctrl_s *can_controller_l;
74
75     /*Point local CAN controller to the passed; cast to can_ctrl_s**/
76     can_controller_l = (can_ctrl_s *)can_ctrl;
77
78     /*Create a CAN frames queue*/
79     can_rx_queue_id = isca_queue_acquire(queue_slots);
80
81     //if ( __builtin_expect(can_rx_queue_id == DQ_OCCUPIED, 0) ) {
82     if ( can_rx_queue_id == DQ_OCCUPIED ) {
83         return ISCA_CAN_QUEUES_OCCUPIED;
84     } /*No queue available*/
```

```
85
86      /*Update can controller struct*/
87      can_controller_l->q_ptr  = (can_frame_s *)malloc(sizeof(
       can_frame_s)*queue_slots);
88      can_controller_l->q_id   = can_rx_queue_id;
89      can_controller_l->q_size = queue_slots;
90
91      /*Set CAN controllers default interrupt callback function*/
92      if ( can_controller_l->irq == CAN_IRQ_ON ) {
93          can_controller_l->InterruptHandler = &
       ISCA_CAN_IntrHandler;
94      }
95
96      /*Setup CAN controller and return*/
97      return isca_can_init(can_ctrl);
98
99  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.16.3.2 lbr_isca_can_start()

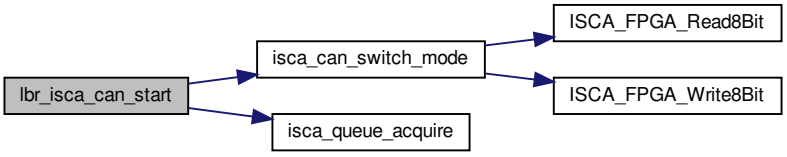
```
int lbr_isca_can_start (
            can_ctrl_s * can_ctrl )
```

Definition at line 160 of file ISCA_CAN_API.c.

References ISCA_CAN_INV_RST_MODE, ISCA_CAN_MODE_RESET_OFF, isca_can_switch_mode(), isca_queue_acquire(), can_controller_s::q_id, and can_controller_s::q_size.

```
160                                             {
161
162     int ret_val;
163
164     //Switch off can controller's reset mode
165     ret_val = isca_can_switch_mode(can_ctrl,
      ISCA_CAN_MODE_RESET_OFF);
166
167     if ( ret_val != ISCA_CAN_INV_RST_MODE ) {
168         //Acquire queue
169         can_ctrl->q_id = isca_queue_acquire(can_ctrl->
      q_size);
170     }
171
172     return ret_val;
173 }
```

Here is the call graph for this function:



A.16.3.3 lbr_isca_can_stop()

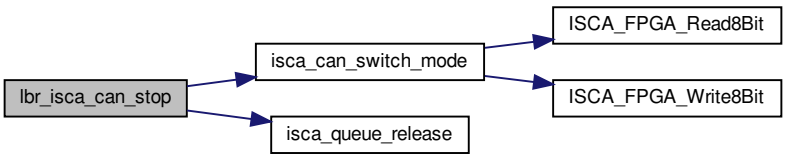
```
int lbr_isca_can_stop (
            can_ctrl_s * can_ctrl )
```

Definition at line 182 of file ISCA_CAN_API.c.

References ISCA_CAN_INV_RST_MODE, ISCA_CAN_MODE_RESET_ON, isca_can_switch_mode(), isca_queue_release(), and can_controller_s::q_id.

```
182                                             {
183
184     int ret_val;
185
186     //Switch on can controller's reset mode
187     ret_val = isca_can_switch_mode(can_ctrl,
      ISCA_CAN_MODE_RESET_ON);
188
189     if ( ret_val != ISCA_CAN_INV_RST_MODE ) {
190         //Release queue
191         isca_queue_release(can_ctrl->q_id);
192     }
193
194     return ret_val;
195
196 }
```

Here is the call graph for this function:



A.16.3.4 lbr_isca_can_receive_pkt()

```
int lbr_isca_can_receive_pkt (
            can_ctrl_s * can_ctrl,
            can_frame_s * rx_frame )
```

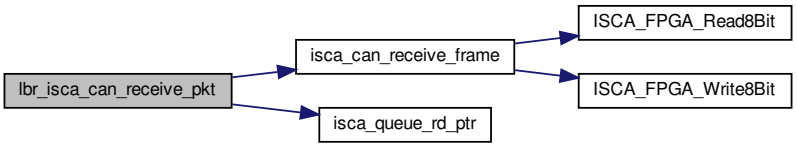
Definition at line 109 of file ISCA_CAN_API.c.

References CAN_IRQ_ON, CAN_REQ_BLOCKING, DQ_EMPTY, can_↩ controller_s::irq, ISCA_CAN_OK, isca_can_receive_frame(), isca_queue_rd↩ _ptr(), and can_controller_s::q_ptr.

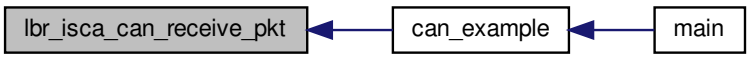
Referenced by can_example().

```
109                                                                    {
110
111     uint8_t queue_rd_index;
112     int q_idx;
113     int ret_val = ISCA_CAN_OK;
114
115     if (can_ctrl->irq == CAN_IRQ_ON) {
116
117         do {
118             q_idx = isca_queue_rd_ptr(((can_ctrl_s*)can_ctrl)->q_id, &
    queue_rd_index);
119         } while( q_idx == DQ_EMPTY ); /*Wait for the interrupt handler to push a frame*/
120
121         /*Interrupt routines use the created during initialization queue.

122          *Copy frame from  to user*/
123         memmove(rx_frame, &can_ctrl->q_ptr[queue_rd_index], sizeof(
    can_frame_s));
124     } /*Interrupt mode*/
125     else {
126
127         /*Poll until frame is available*/
128         ret_val = isca_can_receive_frame(can_ctrl, rx_frame,
    CAN_REQ_BLOCKING);
129
130     } /*Polling mode*/
131
132     return ret_val;
133
134 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.16.3.5 lbr_isca_can_transmit_pkt()

```
int lbr_isca_can_transmit_pkt (
          can_ctrl_s * can_ctrl,
          can_frame_s * tx_frame )
```

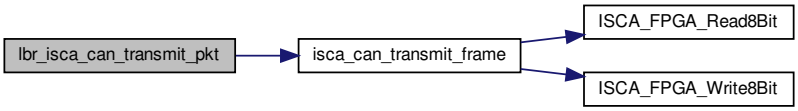
Definition at line 144 of file ISCA_CAN_API.c.

References CAN_REQ_BLOCKING, and isca_can_transmit_frame().

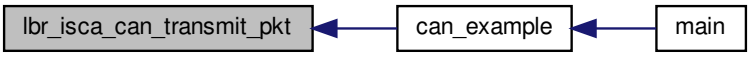
Referenced by can_example().

```
144                                                                    {
145
146     int ret_val;
147
148     ret_val = isca_can_transmit_frame(can_ctrl, tx_frame,
      CAN_REQ_BLOCKING);
149     return ret_val;
150
151 }
```

Here is the call graph for this function:



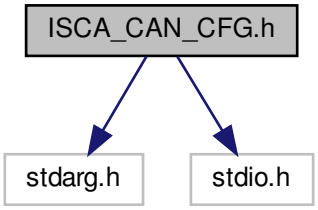
Here is the caller graph for this function:



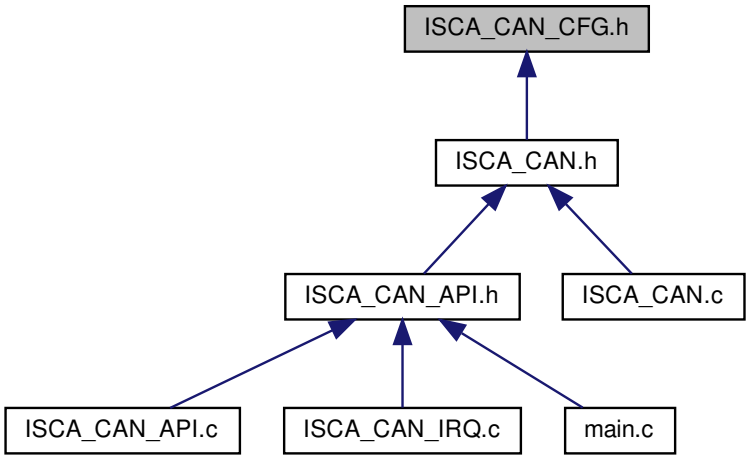
A.17 ISCA_CAN_CFG.h File Reference

```
#include <stdarg.h>
#include <stdio.h>
```

Include dependency graph for include/ISCA_CAN_CFG.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define CAN_2A (0)
- #define CAN_2B (1)
- #define APP_POLL (0)
- #define APP_IRQ (1)
- #define APPRISE_MODE (APP_POLL)
- #define __COUT(str, ...) (printf(str, ##__VA_ARGS__))
  Print to standard output function wrapper.

A.17.1 Macro Definition Documentation

A.17.1.1 CAN_2A

```
#define CAN_2A (0)
```

CAN mode 2A (standard) constant

Definition at line 49 of file include/ISCA_CAN_CFG.h.

A.17.1.2 CAN_2B

```
#define CAN_2B (1)
```

CAN mode 2B (extended) constant

Definition at line 50 of file include/ISCA_CAN_CFG.h.

A.17.1.3 APP_POLL

```
#define APP_POLL (0)
```

CAN apprise in polling mode constant

Definition at line 51 of file include/ISCA_CAN_CFG.h.

A.17.1.4 APP_IRQ

```
#define APP_IRQ (1)
```

CAN apprise in interrupt mode constant

Definition at line 52 of file include/ISCA_CAN_CFG.h.

A.17.1.5 APPRISE_MODE

```
#define APPRISE_MODE (APP_POLL)
```

Apprise mode APP_POLL or APP_IRQ

Definition at line 64 of file include/ISCA_CAN_CFG.h.

A.17.1.6 __COUT

```
#define __COUT(
            str,
            ...  ) (printf(str, ##__VA_ARGS__))
```

Print to standard output function wrapper.

If it happens your board to support some other printf like function, or you have implemented your own modify this definition

Note

You can change standard output function used, by modifying this definition

Definition at line 77 of file include/ISCA_CAN_CFG.h.

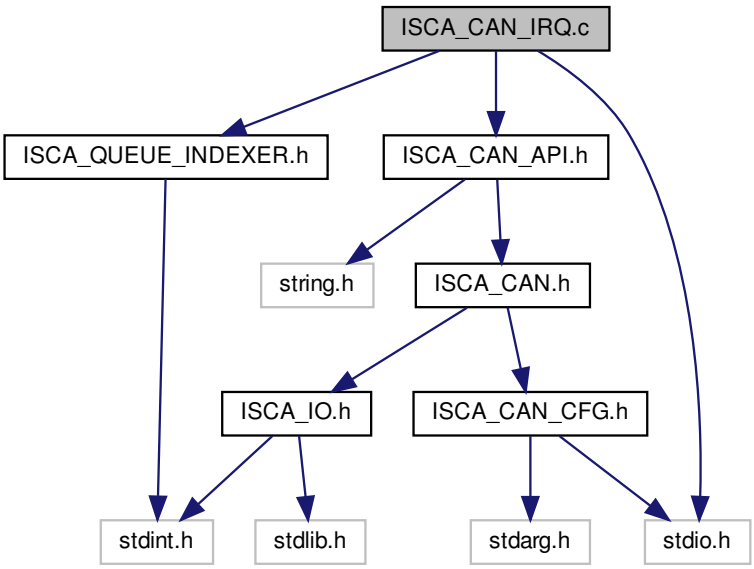
Referenced by ISCA_CAN_IntrHandler(), and isca_can_receive_pkt_irq().

A.18 ISCA_CAN_IRQ.c File Reference

CAN interrupt routines and callback.

```
#include "ISCA_CAN_API.h"
#include "ISCA_QUEUE_INDEXER.h"
#include <stdio.h>
```

Include dependency graph for ISCA_CAN_IRQ.c:



Macros

- #define CAN_IRQS_STATUS_REG (12U)
- #define CAN_IRQ_CLEAR (04U)
- #define BUS_ERROR (0x80U)
- #define ARB_LOST (0x40U)
- #define ERR_P_IRQ (0x20U)
- #define DATA_OVRRUN (0x08U)
- #define ERR_WARN (0x04U)
- #define TX_OK (0x02U)
- #define RX_OK (0x01U)

Functions

- int isca_can_receive_pkt_irq (can_ctrl_s ∗can_ctrl)
- uint8_t isca_can_ack_irq_generic (can_ctrl_s ∗can_ctrl) __attribute__↩ ((always_inline))
- void isca_can_ack_irq_reboot (can_ctrl_s ∗can_ctrl) __attribute__↩ ((always_inline))
- void ISCA_CAN_IntrHandler (void ∗can_ctrl)
  CAN controller interrupt callback.

A.18.1 Detailed Description

CAN interrupt routines and callback.

Version

1.0

Precondition

Fill the "USER CODE" sections with device specific code, as described

A.18.2 Macro Definition Documentation

A.18.2.1 CAN_IRQS_STATUS_REG

```
#define CAN_IRQS_STATUS_REG (12U)
```

IRQs status register offset

Definition at line 52 of file ISCA_CAN_IRQ.c.

Referenced by ISCA_CAN_IntrHandler().

A.18.2.2 CAN_IRQ_CLEAR

```
#define CAN_IRQ_CLEAR (04U)
```

IRQs clear register offset

Definition at line 53 of file ISCA_CAN_IRQ.c.

Referenced by isca_can_ack_irq_generic(), and isca_can_receive_pkt_irq().

A.18.2.3 BUS_ERROR

```
#define BUS_ERROR (0x80U)
```

CAN_IRQS_STATUS_REG[7] -> bus_error_irq_en

Definition at line 59 of file ISCA_CAN_IRQ.c.

Referenced by ISCA_CAN_IntrHandler().

A.18.2.4 ARB_LOST

```
#define ARB_LOST (0x40U)
```

CAN_IRQS_STATUS_REG[6] -> arbitration_lost_irq_en

Definition at line 60 of file ISCA_CAN_IRQ.c.

Referenced by ISCA_CAN_IntrHandler().

A.18.2.5 ERR_P_IRQ

```
#define ERR_P_IRQ (0x20U)
```

CAN_IRQS_STATUS_REG[5] -> error_passive_irq_en

Definition at line 61 of file ISCA_CAN_IRQ.c.

Referenced by ISCA_CAN_IntrHandler().

A.18.2.6 DATA_OVRRUN

```
#define DATA_OVRRUN (0x08U)
```

CAN_IRQS_STATUS_REG[3] -> data_overrun_irq_en_ext

Definition at line 63 of file ISCA_CAN_IRQ.c.

Referenced by ISCA_CAN_IntrHandler().

A.18.2.7 ERR_WARN

```
#define ERR_WARN (0x04U)
```

CAN_IRQS_STATUS_REG[2] -> error_warning_irq_en_ext

Definition at line 64 of file ISCA_CAN_IRQ.c.

Referenced by ISCA_CAN_IntrHandler().

A.18.2.8 TX_OK

```
#define TX_OK (0x02U)
```

CAN_IRQS_STATUS_REG[1] -> transmit_irq_en_ext

Definition at line 65 of file ISCA_CAN_IRQ.c.

Referenced by ISCA_CAN_IntrHandler().

A.18.2.9 RX_OK

```
#define RX_OK (0x01U)
```

CAN_IRQS_STATUS_REG[0] -> receive_irq_en_ext

Definition at line 66 of file ISCA_CAN_IRQ.c.

Referenced by ISCA_CAN_IntrHandler().

A.18.3 Function Documentation

A.18.3.1 isca_can_receive_pkt_irq()

```
int isca_can_receive_pkt_irq (
          can_ctrl_s * can_ctrl )
```

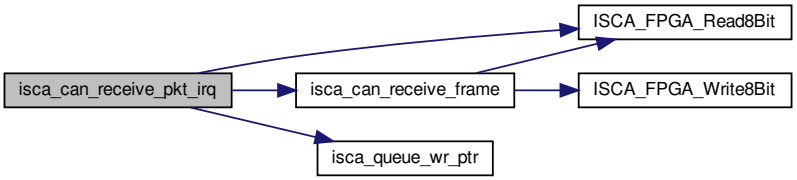
Definition at line 147 of file ISCA_CAN_IRQ.c.

References __COUT, can_controller_s::addr, CAN_IRQ_CLEAR, CAN_REQ_↩ NONBLOCKING, DQ_FULL, isca_can_receive_frame(), ISCA_CAN_RX_FIFO↩ _EMPTY, ISCA_FPGA_Read8Bit(), isca_queue_wr_ptr(), can_controller_s::q_id, and can_controller_s::q_ptr.

Referenced by ISCA_CAN_IntrHandler().

```
147                                                 {
148
149     volatile uint8_t _ack;
150     uint8_t _cpy;
151     int remain_packets;
152     can_frame_s rx_frame;
153     uint8_t queue_wr_index;
154
155     int q_status;
156
157     /*Receive frame, this also releases the RX buffer*/
158     remain_packets = isca_can_receive_frame(can_ctrl, &rx_frame,
    CAN_REQ_NONBLOCKING);
159
160     /*Acknowledge the interrupt*/
161     _ack = ISCA_FPGA_Read8Bit(can_ctrl->addr+CAN_IRQ_CLEAR);
162     /*prevent the compiler from removing the above line; the dummy way*/
163     memmove(&_cpy, (void *)&_ack, sizeof(uint8_t));
164
165     if (remain_packets==ISCA_CAN_RX_FIFO_EMPTY) return 0;
166
167     /*Ask the queue for write pointer*/
168     q_status = isca_queue_wr_ptr(can_ctrl->q_id, &queue_wr_index);
169
170     //if ( __builtin_expect(q_status==DQ_FULL, 0) ) {
171     if ( q_status==DQ_FULL ) {
172         __COUT("\n\rCan queue full, frame rejected");
173         return remain_packets;
174     } /*Queue is full*/
175
176     /*Write frame to the queue*/
177     memmove(&can_ctrl->q_ptr[queue_wr_index], &rx_frame, sizeof(can_frame_s));
178
179     return remain_packets;
180
181 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

A.18.3.2 isca_can_ack_irq_generic()

```
uint8_t isca_can_ack_irq_generic (
          can_ctrl_s * can_ctrl ) [inline]
```

Definition at line 184 of file ISCA_CAN_IRQ.c.

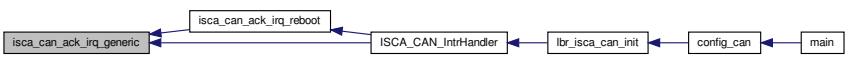
References can_controller_s::addr, CAN_IRQ_CLEAR, and ISCA_FPGA_↩ Read8Bit().

Referenced by isca_can_ack_irq_reboot(), and ISCA_CAN_IntrHandler().

```
184                                                     {
185
186     volatile uint8_t _tmp;
187
188     // acknowledge IRQ.
189     _tmp = ISCA_FPGA_Read8Bit(can_ctrl->addr+CAN_IRQ_CLEAR);
190
191     return _tmp;
192
193 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



A.18.3.3 isca_can_ack_irq_reboot()

```
void isca_can_ack_irq_reboot (
          can_ctrl_s * can_ctrl ) [inline]
```

<

Precondition

Fill the "USER CODE" sections with device specific code, as described

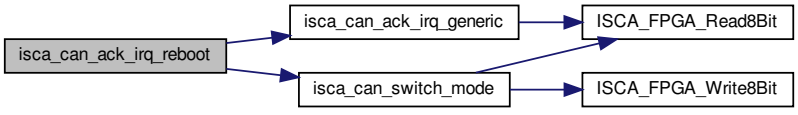
Definition at line 199 of file ISCA_CAN_IRQ.c.

References isca_can_ack_irq_generic(), ISCA_CAN_MODE_RESET_OFF, ISCA_CAN_MODE_RESET_ON, and isca_can_switch_mode().

Referenced by ISCA_CAN_IntrHandler().

```
199                                                 {
200
201     // acknowledge IRQ.
202     isca_can_ack_irq_generic(can_ctrl);
203
204     // Soft reset CAN controller
205     isca_can_switch_mode(can_ctrl, ISCA_CAN_MODE_RESET_ON);
206     isca_can_switch_mode(can_ctrl, ISCA_CAN_MODE_RESET_OFF);
207
208     /*
209      * Place code to reset the FPGA or the entire system
210      * If such functionality not available you can:
211      * A. call exit(0) from stdlib.h
212      * B. Soft reset the can controller with
213      *    isca_can_switch_mode(ISCA_CAN_MODE_RESET_ON);
214      *    isca_can_switch_mode(ISCA_CAN_MODE_RESET_OFF);
215      */
216
217     /*USER CODE*/
218     /*END USER CODE*/
219
220 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

A.18.3.4 ISCA_CAN_IntrHandler()

```
void ISCA_CAN_IntrHandler (
          void * can_ctrl )
```

CAN controller interrupt callback.

Precondition

Fill the "USER CODE" sections with device specific code, as described

Parameters

in	*can_ctrl*	CAN controllers instance pointer

Returns

None

Definition at line 85 of file ISCA_CAN_IRQ.c.

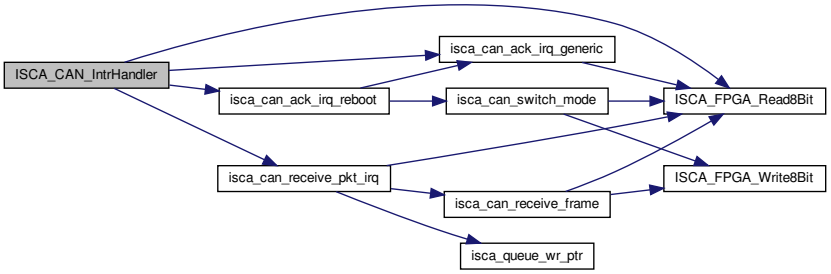
References __COUT, can_controller_s::addr, ARB_LOST, BUS_ERROR, CAN_IRQS_STATUS_REG, DATA_OVRRUN, ERR_P_IRQ, ERR_WARN, isca_can_ack_irq_generic(), isca_can_ack_irq_reboot(), isca_can_receive_pkt_irq(), ISCA_FPGA_Read8Bit(), RX_OK, and TX_OK.

Referenced by lbr_isca_can_init().

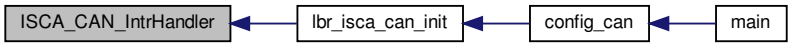
```
85                                          {
86
87     uint8_t irq_rd;
88     can_ctrl_s *can_ctrl_l;
89
90     /* USER CODE
```

```
91      * Place code to apply the proper negotiations with the Interrupt controller
92      * CAN interrupt is connected with, e.g. disable IRQ's or acknowledge it
93      */
94
95     /* END USER_CODE*/
96
97     // Cast & link can controller
98     can_ctrl_l = (can_ctrl_s *) can_ctrl;
99     /* Reading the IRQ status register*/
100     irq_rd = ISCA_FPGA_Read8Bit(can_ctrl_l->addr+
      CAN_IRQS_STATUS_REG);
101
102     /*Check read register's every bit*/
103     if ( (irq_rd&RX_OK) == RX_OK ) {
104         isca_can_receive_pkt_irq(can_ctrl_l);
105     } /*IRQ: RX*/
106     else if ( (irq_rd&TX_OK) == TX_OK ) {
107         isca_can_ack_irq_generic(can_ctrl_l);
108     } /*IRQ: TX*/
109     else if ( (irq_rd&ERR_WARN) == ERR_WARN ) {
110         __COUT("\n\rError IRQ!\n\r");
111         isca_can_ack_irq_reboot(can_ctrl_l);
112     } /*IRQ: Error*/
113     else if ( (irq_rd&DATA_OVRRUN) == DATA_OVRRUN ) {
114         __COUT("\n\rRX FIFO overrun IRQ!\n\r");
115         /*Maybe start reading some frames out of RX FIFO*/
116         isca_can_ack_irq_generic(can_ctrl_l);
117     } /*IRQ: RX FIFO IRQ*/
118 #if (CAN_MODE == CAN_2B)
119     else if ( (irq_rd&ERR_P_IRQ) == ERR_P_IRQ ) {
120         __COUT("\n\rError passive IRQ!\n\r");
121         isca_can_ack_irq_reboot(can_ctrl_l);
122     } /*IRQ: Error passive IRQ*/
123     else if ( (irq_rd&ARB_LOST) == ARB_LOST ) {
124         __COUT("\n\rArbitration lost IRQ!\n\r");
125         isca_can_ack_irq_reboot(can_ctrl_l);
126     } /*IRQ: Arbitration lost*/
127     else if ( (irq_rd&BUS_ERROR) == BUS_ERROR ) {
128         __COUT("\n\rBus error IRQ!\n\r");
129         isca_can_ack_irq_reboot(can_ctrl_l);
130     } /*IRQ: Bus error*/
131 #endif // !(CAN_MODE == CAN_2A)
132
133     /* USER CODE
134      * Place code to apply the proper negotiations with the Interrupt controller
135      * CAN interrupt is connected with, e.g. re-enable IRQ's
136      */
137
138     /* END USER_CODE*/
139
140 }
```

Here is the call graph for this function:



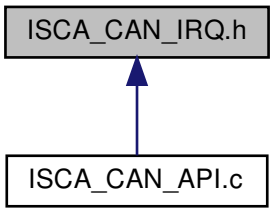
Here is the caller graph for this function:



A.19 ISCA_CAN_IRQ.h File Reference

CAN interrupt routines and callback.

This graph shows which files directly or indirectly include this file:



Functions

- void ISCA_CAN_IntrHandler (void ∗can_ctrl)
  CAN controller interrupt callback.

A.19.1 Detailed Description

CAN interrupt routines and callback.

Version

1.0

A.19.2 Function Documentation

A.19.2.1 ISCA_CAN_IntrHandler()

```
void ISCA_CAN_IntrHandler (
           void * can_ctrl )
```

CAN controller interrupt callback.

Precondition

Fill the "USER CODE" sections with device specific code, as described

Parameters

in	*can_ctrl*	CAN controllers instance pointer

Returns

None

Definition at line 85 of file ISCA_CAN_IRQ.c.

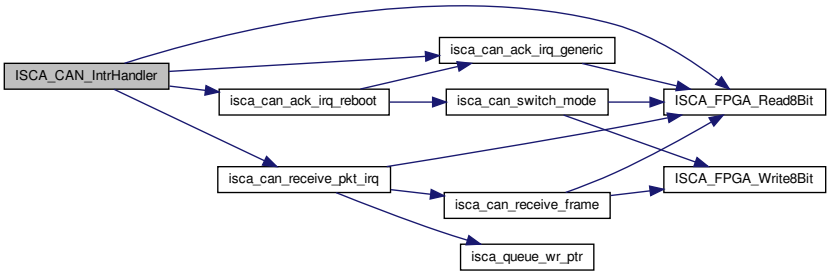
References __COUT, can_controller_s::addr, ARB_LOST, BUS_ERROR, CAN_IRQS_STATUS_REG, DATA_OVRRUN, ERR_P_IRQ, ERR_WARN, isca_can_ack_irq_generic(), isca_can_ack_irq_reboot(), isca_can_receive_pkt_irq(), ISCA_FPGA_Read8Bit(), RX_OK, and TX_OK.

Referenced by lbr_isca_can_init().

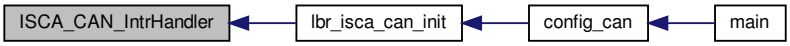
```
85                                          {
86
87     uint8_t irq_rd;
```

```
88      can_ctrl_s *can_ctrl_l;
89
90      /* USER CODE
91       * Place code to apply the proper negotiations with the Interrupt controller
92       * CAN interrupt is connected with, e.g. disable IRQ's or acknowledge it
93       */
94
95      /* END USER_CODE*/
96
97      // Cast & link can controller
98      can_ctrl_l = (can_ctrl_s *) can_ctrl;
99      /* Reading the IRQ status register*/
100      irq_rd = ISCA_FPGA_Read8Bit(can_ctrl_l->addr+
      CAN_IRQS_STATUS_REG);
101
102     /*Check read register's every bit*/
103     if ( (irq_rd&RX_OK) == RX_OK ) {
104         isca_can_receive_pkt_irq(can_ctrl_l);
105     } /*IRQ: RX*/
106     else if ( (irq_rd&TX_OK) == TX_OK ) {
107         isca_can_ack_irq_generic(can_ctrl_l);
108     } /*IRQ: TX*/
109     else if ( (irq_rd&ERR_WARN) == ERR_WARN ) {
110         __COUT("\n\rError IRQ!\n\r");
111         isca_can_ack_irq_reboot(can_ctrl_l);
112     } /*IRQ: Error*/
113     else if ( (irq_rd&DATA_OVRRUN) == DATA_OVRRUN ) {
114         __COUT("\n\rRX FIFO overrun IRQ!\n\r");
115         /*Maybe start reading some frames out of RX FIFO*/
116         isca_can_ack_irq_generic(can_ctrl_l);
117     } /*IRQ: RX FIFO IRQ*/
118 #if (CAN_MODE == CAN_2B)
119     else if ( (irq_rd&ERR_P_IRQ) == ERR_P_IRQ ) {
120         __COUT("\n\rError passive IRQ!\n\r");
121         isca_can_ack_irq_reboot(can_ctrl_l);
122     } /*IRQ: Error passive IRQ*/
123     else if ( (irq_rd&ARB_LOST) == ARB_LOST ) {
124         __COUT("\n\rArbitration lost IRQ!\n\r");
125         isca_can_ack_irq_reboot(can_ctrl_l);
126     } /*IRQ: Arbitration lost*/
127     else if ( (irq_rd&BUS_ERROR) == BUS_ERROR ) {
128         __COUT("\n\rBus error IRQ!\n\r");
129         isca_can_ack_irq_reboot(can_ctrl_l);
130     } /*IRQ: Bus error*/
131 #endif // !(CAN_MODE == CAN_2A)
132
133     /* USER CODE
134      * Place code to apply the proper negotiations with the Interrupt controller
135      * CAN interrupt is connected with, e.g. re-enable IRQ's
136      */
137
138     /* END USER_CODE*/
139
140 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

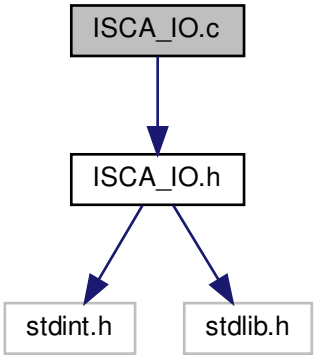


A.20 ISCA_IO.c File Reference

Provides tools to Read/Write 8bit from/to the FPGA.

```
#include "ISCA_IO.h"
```

Include dependency graph for ISCA_IO.c:



Functions

- void __attribute__ ((always_inline))

A.20.1 Detailed Description

Provides tools to Read/Write 8bit from/to the FPGA.

Version

1.0

Todo Implement read/write for more data types

A.20.2 Function Documentation

A.20.2.1 __attribute__()

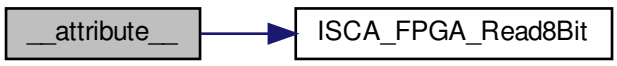
```
uint8_t __attribute__ (
            (always_inline)  )
```

Definition at line 50 of file ISCA_IO.c.

References ISCA_FPGA_Read8Bit().

```
52 {
53     *((volatile uint8_t *)address_ptr) = data;
54 }
```

Here is the call graph for this function:

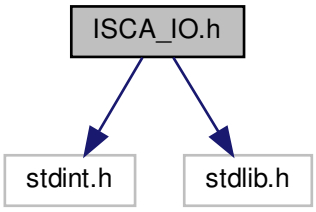


A.21 ISCA_IO.h File Reference

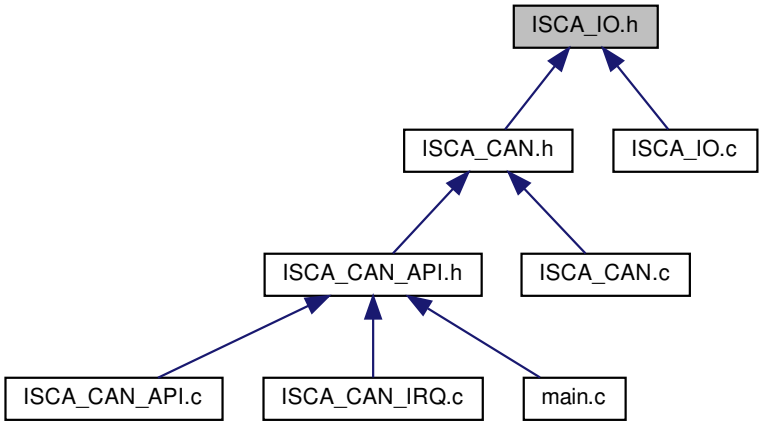
Provides tools to Read/Write 8bit from/to the FPGA.

```
#include "stdint.h"
#include "stdlib.h"
```

Include dependency graph for ISCA_IO.h:



This graph shows which files directly or indirectly include this file:



Functions

- void ISCA_FPGA_Write8Bit (size_t const address_ptr, uint8_t const data)
- uint8_t ISCA_FPGA_Read8Bit (size_t const address_ptr)

A.21.1 Detailed Description

Provides tools to Read/Write 8bit from/to the FPGA.

Version

1.0

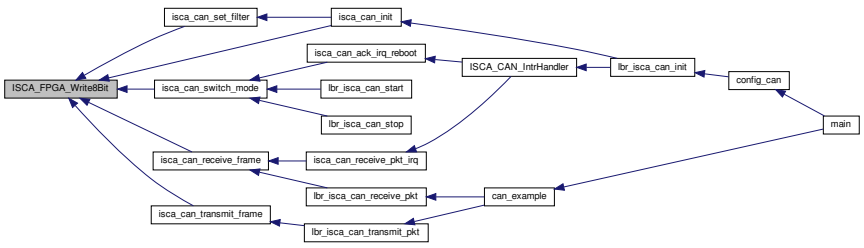
A.21.2 Function Documentation

A.21.2.1 ISCA_FPGA_Write8Bit()

```
void ISCA_FPGA_Write8Bit (
          size_t const address_ptr,
          uint8_t const data )
```

Referenced by isca_can_init(), isca_can_receive_frame(), isca_can_set_filter(), isca_can_switch_mode(), and isca_can_transmit_frame().

Here is the caller graph for this function:

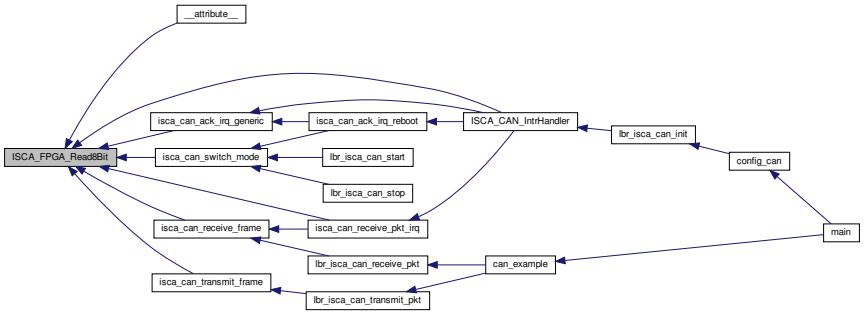


A.21.2.2 ISCA_FPGA_Read8Bit()

```
uint8_t ISCA_FPGA_Read8Bit (
          size_t const address_ptr )
```

Referenced by __attribute__(), isca_can_ack_irq_generic(), ISCA_CAN_Intr↩ Handler(), isca_can_receive_frame(), isca_can_receive_pkt_irq(), isca_can_↩ switch_mode(), and isca_can_transmit_frame().

Here is the caller graph for this function:

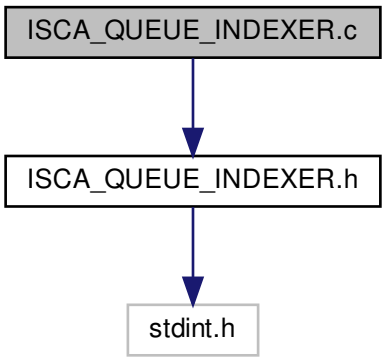


A.22 ISCA_QUEUE_INDEXER.c File Reference

Provides tools to initialize and use a preallocated circular queue pointers.

```
#include "ISCA_QUEUE_INDEXER.h"
```

Include dependency graph for ISCA_QUEUE_INDEXER.c:



Data Structures

- struct isca_data_queue_indexer

Functions

- void queue_lock_acquire ()
- void queue_lock_release ()
- int isca_queue_acquire (uint8_t const q_slots)
  Generic queue (ring buffer) indexer.
- int isca_queue_release (uint8_t queue_index)
- int isca_queue_rd_ptr (uint8_t q_index, uint8_t *queue_rd_ptr)
- int isca_queue_wr_ptr (uint8_t q_index, uint8_t *queue_wr_index)

Variables

- static volatile uint32_t available_queues = 0U
- static volatile uint8_t queue_slots [MAX_QUEUES]
- static volatile isca_data_queue_indexer queue [MAX_QUEUES]

A.22.1 Detailed Description

Provides tools to initialize and use a preallocated circular queue pointers.

Version

1.0

Todo Implement lock mechanisms for multi-threaded environments

A.22.2 Function Documentation

A.22.2.1 isca_queue_acquire()

```
int isca_queue_acquire (
          uint8_t const q_slots )
```

Generic queue (ring buffer) indexer.

Note

The queue to store data is not allocated here; only queue indexing is provided

Parameters

in	*q_slots*	the amount of slots the new queue shall have

Return values

Acquired	queue index

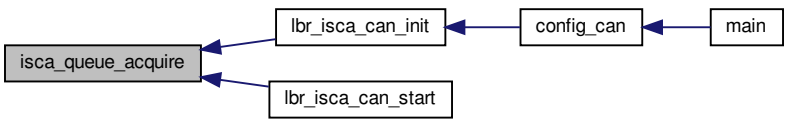
Definition at line 81 of file ISCA_QUEUE_INDEXER.c.

References available_queues, DQ_OCCUPIED, MAX_QUEUES, queue_slots, isca_data_queue_indexer::rd_ptr, and isca_data_queue_indexer::wr_ptr.

Referenced by lbr_isca_can_init(), and lbr_isca_can_start().

```
81                                                  {
82
83     int i;
84
85     //find an available queue indexer
86     for ( i = 0; i < MAX_QUEUES; i++ ) {
87         if ( ( ( available_queues >> i ) & 1U ) == 0U ) {
88             available_queues |= ( 1U << i ); //reserve the queue
89             /*Update acquired queue info*/
90             queue_slots[i]    = q_slots;
91             queue[i].rd_ptr   = 0U;
92             queue[i].wr_ptr   = 0U;
93             return i;
94         } /*queue occupied*/
95     }
96
97     //TODO: return a hash of the queue and match it internally with the queue's index
98     //      in this way only
99     /*return the acquired queue*/
100     return DQ_OCCUPIED;
101 }
```

Here is the caller graph for this function:



A.22.2.2 isca_queue_release()

```
int isca_queue_release (
            uint8_t queue_index )
```

Definition at line 103 of file ISCA_QUEUE_INDEXER.c.

References available_queues, DQ_AVAILABLE, DQ_OK, queue_slots, isca_↩ data_queue_indexer::rd_ptr, and isca_data_queue_indexer::wr_ptr.

Referenced by lbr_isca_can_stop().

```
103                                             {
104
105     if ( ( available_queues >> queue_index ) & 1U ) {
106         queue_slots[queue_index] = 0U;
107         queue[queue_index].rd_ptr = 0U;
108         queue[queue_index].wr_ptr = 0U;
109         available_queues ^= ((uint32_t) 1U << queue_index); //release queue
110     } /*queue occupied*/
111     else {
112         return DQ_AVAILABLE;
113     } /*queue is not occupied*/
114
115     return DQ_OK;
116 }
```

Here is the caller graph for this function:

A.22.2.3 isca_queue_rd_ptr()

```
int isca_queue_rd_ptr (
            uint8_t q_index,
            uint8_t * queue_rd_ptr )
```

<

Bug When asking from RX queue with isca_queue_rd_ptr the library advances the read pointer; it is possible that the slot (read pointer) might be overwritten by a newly arrived frame.

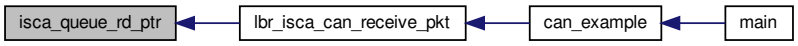
Definition at line 123 of file ISCA_QUEUE_INDEXER.c.

References DQ_EMPTY, DQ_OK, queue_slots, and isca_data_queue_indexer::rd_ptr.

Referenced by lbr_isca_can_receive_pkt().

```
123                                                                 {
124
125     if ( queue[q_index].rd_ptr == queue[q_index].wr_ptr ) {
126         return DQ_EMPTY;
127     } /*Queue empty*/
128
129     *queue_rd_ptr = (uint8_t) queue[q_index].rd_ptr;
130     queue[q_index].rd_ptr = (queue[q_index].rd_ptr + 1) %
      queue_slots[q_index];
131
132     return DQ_OK;
133 }
```

Here is the caller graph for this function:



A.22.2.4 isca_queue_wr_ptr()

```
int isca_queue_wr_ptr (
          uint8_t q_index,
          uint8_t * queue_wr_index )
```

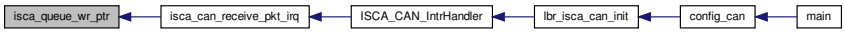
Definition at line 135 of file ISCA_QUEUE_INDEXER.c.

References DQ_FULL, DQ_OK, queue_slots, and isca_data_queue_indexer::wr_ptr.

Referenced by isca_can_receive_pkt_irq().

```
135                                                                 {
136
137     uint8_t next_wr_index;
138
139     next_wr_index = (queue[q_index].wr_ptr + 1) % queue_slots[q_index];
140
141     if ( next_wr_index == queue[q_index].rd_ptr ) {
142         return DQ_FULL;
143     } /*Queue full*/
144
145     *queue_wr_index = (uint8_t) (queue[q_index].wr_ptr);
146     queue[q_index].wr_ptr = next_wr_index;
147
148     return DQ_OK;
149 }
```

Here is the caller graph for this function:



A.22.3 Variable Documentation

A.22.3.1 available_queues

`volatile uint32_t available_queues = 0U [static]`

Definition at line 60 of file ISCA_QUEUE_INDEXER.c.

Referenced by isca_queue_acquire(), and isca_queue_release().

A.22.3.2 queue_slots

`volatile uint8_t queue_slots[MAX_QUEUES] [static]`

Definition at line 63 of file ISCA_QUEUE_INDEXER.c.

Referenced by isca_queue_acquire(), isca_queue_rd_ptr(), isca_queue_release(), isca_queue_wr_ptr(), and lbr_isca_can_init().

A.22.3.3 queue

```
volatile isca_data_queue_indexer queue[MAX_QUEUES] [static]
```

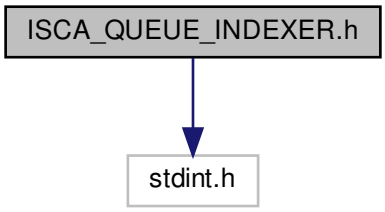
Definition at line 64 of file ISCA_QUEUE_INDEXER.c.

A.23 ISCA_QUEUE_INDEXER.h File Reference

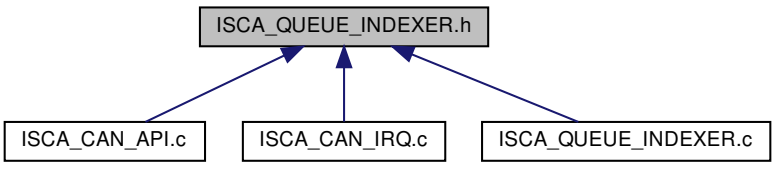
Provides tools to initialize and use a preallocated circular queue pointers.

```
#include "stdint.h"
```

Include dependency graph for ISCA_QUEUE_INDEXER.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define MAX_QUEUES (8)
- #define DQ_OK (0)
- #define DQ_AVAILABLE (1)
- #define DQ_OCCUPIED (-1)
- #define DQ_FULL (-2)
- #define DQ_EMPTY (-3)

Functions

- int isca_queue_acquire (uint8_t const q_slots)
  Generic queue (ring buffer) indexer.
- int isca_queue_release (uint8_t queue_index)
- int isca_queue_rd_ptr (uint8_t q_index, uint8_t *queue_rd_ptr)
- int isca_queue_wr_ptr (uint8_t q_index, uint8_t *queue_wr_index)

A.23.1 Detailed Description

Provides tools to initialize and use a preallocated circular queue pointers.

Version

1.0

A.23.2 Macro Definition Documentation

A.23.2.1 MAX_QUEUES

```
#define MAX_QUEUES (8)
```

Definition at line 52 of file ISCA_QUEUE_INDEXER.h.

Referenced by isca_queue_acquire().

A.23.2.2 DQ_OK

```
#define DQ_OK (0)
```

Definition at line 53 of file ISCA_QUEUE_INDEXER.h.

Referenced by isca_queue_rd_ptr(), isca_queue_release(), and isca_queue_↩ wr_ptr().

A.23.2.3 DQ_AVAILABLE

```
#define DQ_AVAILABLE (1)
```

Definition at line 54 of file ISCA_QUEUE_INDEXER.h.

Referenced by isca_queue_release().

A.23.2.4 DQ_OCCUPIED

```
#define DQ_OCCUPIED (-1)
```

Definition at line 55 of file ISCA_QUEUE_INDEXER.h.

Referenced by isca_queue_acquire(), and lbr_isca_can_init().

A.23.2.5 DQ_FULL

```
#define DQ_FULL (-2)
```

Definition at line 56 of file ISCA_QUEUE_INDEXER.h.

Referenced by isca_can_receive_pkt_irq(), and isca_queue_wr_ptr().

A.23.2.6 DQ_EMPTY

```
#define DQ_EMPTY (-3)
```

Definition at line 57 of file ISCA_QUEUE_INDEXER.h.

Referenced by isca_queue_rd_ptr(), and lbr_isca_can_receive_pkt().

A.23.3 Function Documentation

A.23.3.1 isca_queue_acquire()

```
int isca_queue_acquire (
          uint8_t const q_slots )
```

Generic queue (ring buffer) indexer.

Note

The queue to store data is not allocated here; only queue indexing is provided

Parameters

in	*q_slots*	the amount of slots the new queue shall have

Return values

Acquired	queue index

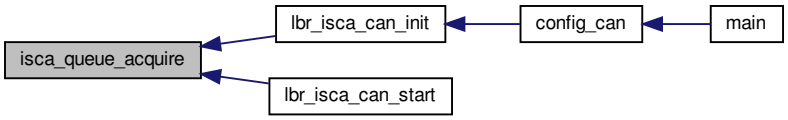
Definition at line 81 of file ISCA_QUEUE_INDEXER.c.

References available_queues, DQ_OCCUPIED, MAX_QUEUES, queue_slots, isca_data_queue_indexer::rd_ptr, and isca_data_queue_indexer::wr_ptr.

Referenced by lbr_isca_can_init(), and lbr_isca_can_start().

```
81                                                  {
82
83     int i;
84
85     //find an available queue indexer
86     for ( i = 0; i < MAX_QUEUES; i++ ) {
87         if ( ( ( available_queues >> i ) & 1U ) == 0U ) {
88             available_queues |= ( 1U << i ); //reserve the queue
89             /*Update acquired queue info*/
90             queue_slots[i]    = q_slots;
91             queue[i].rd_ptr   = 0U;
92             queue[i].wr_ptr   = 0U;
93             return i;
94         } /*queue occupied*/
95     }
96
97     //TODO: return a hash of the queue and match it internally with the queue's index
98     //      in this way only
99     /*return the acquired queue*/
100     return DQ_OCCUPIED;
101 }
```

Here is the caller graph for this function:



A.23.3.2 isca_queue_release()

```
int isca_queue_release (
            uint8_t queue_index )
```

Definition at line 103 of file ISCA_QUEUE_INDEXER.c.

References available_queues, DQ_AVAILABLE, DQ_OK, queue_slots, isca_↩ data_queue_indexer::rd_ptr, and isca_data_queue_indexer::wr_ptr.

Referenced by lbr_isca_can_stop().

```
103                                             {
104
105     if ( ( available_queues >> queue_index ) & 1U ) {
106         queue_slots[queue_index] = 0U;
107         queue[queue_index].rd_ptr = 0U;
108         queue[queue_index].wr_ptr = 0U;
109         available_queues ^= ((uint32_t) 1U << queue_index); //release queue
110     } /*queue occupied*/
111     else {
112         return DQ_AVAILABLE;
113     } /*queue is not occupied*/
114
115     return DQ_OK;
116 }
```

Here is the caller graph for this function:

A.23.3.3 isca_queue_rd_ptr()

```
int isca_queue_rd_ptr (
          uint8_t q_index,
          uint8_t * queue_rd_ptr )
```

<

Bug When asking from RX queue with isca_queue_rd_ptr the library advances the read pointer; it is possible that the slot (read pointer) might be overwritten by a newly arrived frame.

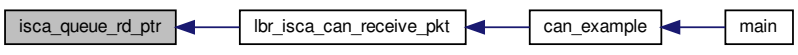
Definition at line 123 of file ISCA_QUEUE_INDEXER.c.

References DQ_EMPTY, DQ_OK, queue_slots, and isca_data_queue_indexer↩ ::rd_ptr.

Referenced by lbr_isca_can_receive_pkt().

```
123                                                                 {
124
125     if ( queue[q_index].rd_ptr == queue[q_index].wr_ptr ) {
126         return DQ_EMPTY;
127     } /*Queue empty*/
128
129     *queue_rd_ptr = (uint8_t) queue[q_index].rd_ptr;
130     queue[q_index].rd_ptr = (queue[q_index].rd_ptr + 1) %
      queue_slots[q_index];
131
132     return DQ_OK;
133 }
```

Here is the caller graph for this function:



A.23.3.4 isca_queue_wr_ptr()

```
int isca_queue_wr_ptr (
          uint8_t q_index,
          uint8_t * queue_wr_index )
```

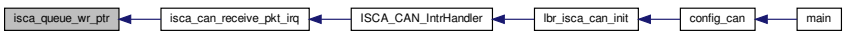
Definition at line 135 of file ISCA_QUEUE_INDEXER.c.

References DQ_FULL, DQ_OK, queue_slots, and isca_data_queue_indexer↩ ::wr_ptr.

Referenced by isca_can_receive_pkt_irq().

```
135                                                                 {
136
137     uint8_t next_wr_index;
138
139     next_wr_index = (queue[q_index].wr_ptr + 1) % queue_slots[q_index];
140
141     if ( next_wr_index == queue[q_index].rd_ptr ) {
142         return DQ_FULL;
143     } /*Queue full*/
144
145     *queue_wr_index = (uint8_t) (queue[q_index].wr_ptr);
146     queue[q_index].wr_ptr = next_wr_index;
147
148     return DQ_OK;
149 }
```

Here is the caller graph for this function:

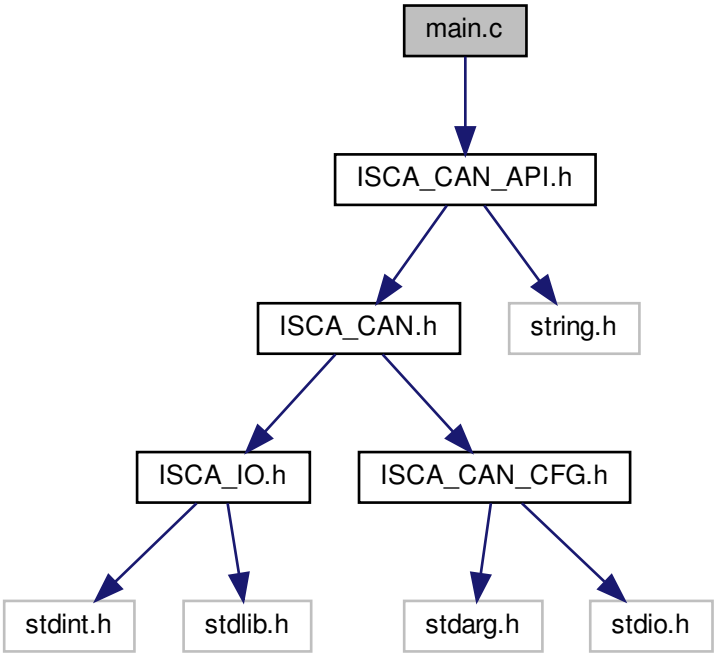


A.24 main.c File Reference

Provides tools to initialize and use a preallocated circular queue pointers.

```
#include "ISCA_CAN_API.h"
```

Include dependency graph for main.c:



Macros

- #define CAN_TX (0U)
- #define CAN_RX (1U)
- #define CAN0_ROLE CAN_TX
- #define CAN0_BASEADDR (0xA0000000U)
- #define RX_QUEUE_SIZE (15U)

Functions

- int config_can (can_ctrl_s ∗CanInstancePtr)
- void can_example (can_ctrl_s ∗CanInstancePtr, can_frame_s ∗CanFrm)
- int main (void)

Variables

- static can_ctrl_s can_ctrl
- static can_frame_s can_frame

A.24.1 Detailed Description

Provides tools to initialize and use a preallocated circular queue pointers.

Version

1.0

A.24.2 Macro Definition Documentation

A.24.2.1 CAN_TX

```
#define CAN_TX (0U)
```

Definition at line 64 of file main.c.

A.24.2.2 CAN_RX

```
#define CAN_RX (1U)
```

Definition at line 65 of file main.c.

A.24.2.3 CAN0_ROLE

```
#define CAN0_ROLE CAN_TX
```

Definition at line 66 of file main.c.

A.24.2.4 CAN0_BASEADDR

```
#define CAN0_BASEADDR (0xA0000000U)
```

Definition at line 67 of file main.c.

Referenced by config_can().

A.24.2.5 RX_QUEUE_SIZE

```
#define RX_QUEUE_SIZE (15U)
```

Definition at line 68 of file main.c.

Referenced by config_can().

A.24.3 Function Documentation

A.24.3.1 config_can()

```
int config_can (
          can_ctrl_s * CanInstancePtr )
```

Initialize can structure

- Operate at 1000kbp (if input clock 100MHz)
- Drop any frame if header MSB is equal to '0'
- Enable interrupts
- Enable extended frame format (29-bit header) if CAN_MODE=CAN_2B

Enable RX interrupt

Disable TX interrupt

Enable Error warning interrupt

Enable RX FIFO overrun interrupt

Enable Error passive IRQ interrupt

Enable Arbitration lost interrupt

Enable Bus error interrupt

Definition at line 110 of file main.c.

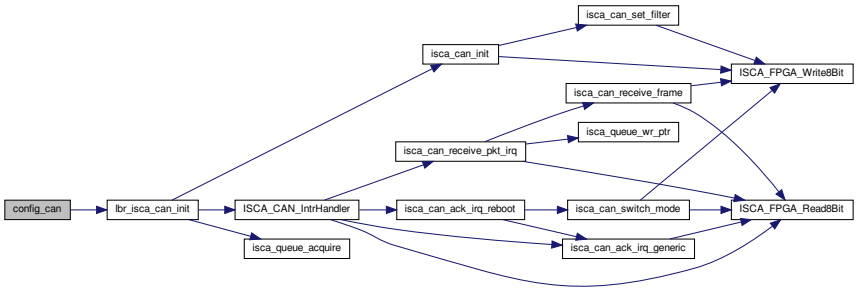
References can_controller_s::addr, can_controller_s::brp, CAN0_BASEADDR, CAN_FRAME_EXT, CAN_IRQ_OFF, CAN_IRQ_ON, can_controller_s::code, can_irq_en_u::err0, can_irq_en_u::err1, can_irq_en_u::err2, can_irq_en_u::err3, can_irq_en_u::err4, can_controller_s::frm_md, can_controller_s::irq, can_controller_s::irqs_en, lbr_isca_can_init(), can_controller_s::mask, can_irq_en_u::rx, RX_QUEUE_SIZE, can_controller_s::sjw, can_controller_s::tseg1, can_controller_s::tseg2, and can_irq_en_u::tx.

Referenced by main().

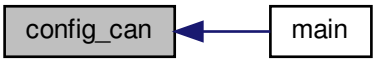
```
110                                         {
118     CanInstancePtr->addr          = CAN0_BASEADDR;
119     CanInstancePtr->brp           = 9U;
120     CanInstancePtr->tseg1         = 1U;
121     CanInstancePtr->tseg2         = 1U;
122     CanInstancePtr->sjw           = 1U;
123     CanInstancePtr->irqs_en.rx    = CAN_IRQ_ON;
124     CanInstancePtr->irqs_en.tx    = CAN_IRQ_OFF;
125 #if !(CAN_MODE == CAN_2B)
126     CanInstancePtr->mask          = 0x3FF;
127     CanInstancePtr->code          = 0x400;
128     CanInstancePtr->irqs_en.err0  = IRQ_ON;
129     CanInstancePtr->irqs_en.err1  = IRQ_ON;
130 #else
131     CanInstancePtr->frm_md        = CAN_FRAME_EXT;
132     CanInstancePtr->mask          = 0x0FFFFFFF;  // check only ID[28]
133     CanInstancePtr->code          = 0x10000000;  // check if ID[28]='1'
134     CanInstancePtr->irqs_en.err0  = CAN_IRQ_ON;
135     CanInstancePtr->irqs_en.err1  = CAN_IRQ_ON;
136     CanInstancePtr->irqs_en.err2  = CAN_IRQ_ON;
```

```
137     CanInstancePtr->irqs_en.err3  = CAN_IRQ_ON;
138     CanInstancePtr->irqs_en.err4  = CAN_IRQ_ON;
139 #endif // !(CAN_MODE == CAN_2B)
140
141 #if !(APPRISE_MODE==APP_IRQ)
142     CanInstancePtr->irq         = CAN_IRQ_OFF;
143 #else
144     CanInstancePtr->irq         = CAN_IRQ_ON;
145 #endif // !(APPRISE_MODE==APP_IRQ)
146
147
148     /*Initialize the controller and exit*/
149     return lbr_isca_can_init(CanInstancePtr, RX_QUEUE_SIZE);
150 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.24.3.2 can_example()

```
void can_example (
          can_ctrl_s * CanInstancePtr,
          can_frame_s * CanFrm )
```

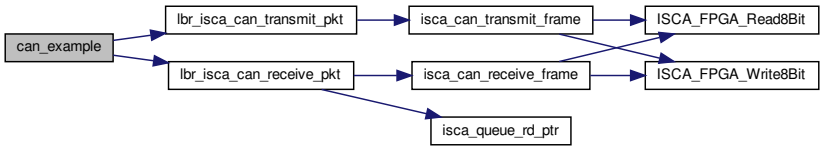
Definition at line 152 of file main.c.

References CAN_FRAME_EXT, can_frame_s::DATA, can_frame_s::DLC, can_↩ frame_s::ID, can_frame_s::IDE, lbr_isca_can_receive_pkt(), and lbr_isca_can_↩ transmit_pkt().

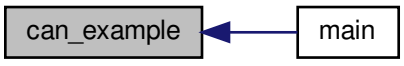
Referenced by main().

```
152                                                                    {
153 #if !(CAN0_ROLE==CAN_RX)
154     /*TX*/
155     CanFrm->ID      = 0x10AB0003;
156 #if (CAN_MODE == CAN_2B)
157     CanFrm->IDE     = CAN_FRAME_EXT;
158 #endif // (CAN_MODE == CAN_2B)
159     CanFrm->DLC     = 2U;
160     CanFrm->DATA[0] = 0xAA;
161     CanFrm->DATA[1] = 0xBB;
162     lbr_isca_can_transmit_pkt(CanInstancePtr, CanFrm);
163
164 #else
165     /*RX*/
166     lbr_isca_can_receive_pkt(CanInstancePtr, CanFrm);
167 #endif // !(CAN0_ROLE==RX)
168 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



A.24.3.3 main()

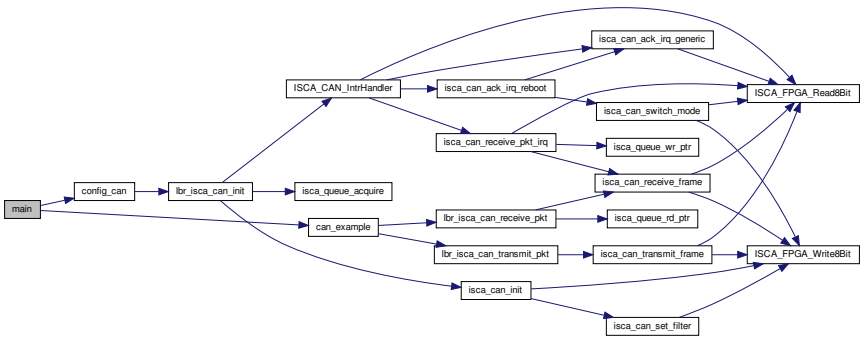
```
int main (
           void  )
```

Definition at line 87 of file main.c.

References can_example(), config_can(), ISCA_CAN_ERROR, and ISCA_CA↩
N_OK.

```
87              {
88
89     int status;
90
91     /*
92      * configure CAN controller
93      */
94     status = config_can(&can_ctrl);
95     if ( status != ISCA_CAN_OK) {
96         return ISCA_CAN_ERROR;
97     }
98
99     /*
100      * Run application
101      */
102     can_example(&can_ctrl, &can_frame);
103
104     return 0;
105 }
```

Here is the call graph for this function:



A.24.4 Variable Documentation

A.24.4.1 can_ctrl

```
can_ctrl_s can_ctrl [static]
```

Definition at line 74 of file main.c.

A.24.4.2 can_frame

```
can_frame_s can_frame [static]
```

Definition at line 75 of file main.c.

Bibliography

[1] Wikipedia. (2019). Can bus, [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus (visited on 11/07/2019).

[2] G. Kornaros, O. Tomoutzoglou, and M. Coppola, "Hardware-assisted security in electronic control units: Secure automotive communications by utilizing one-time-programmable network on chip and firewalls", *IEEE Micro*, vol. 38, no. 5, pp. 63–74, 2018. DOI: 10.1109/MM.2018.053631143. [Online]. Available: https://ieeexplore.ieee.org/document/8474944.

[3] ACTEL. (2010). Can/canbus and can protocol licensing, [Online]. Available: http://www.actel.com/ipdocs/CANbus_lic_RS.pdf (visited on 11/07/2019).

[4] I. Mohor. (2017). Can protocol controller, [Online]. Available: https://opencores.org/projects/can.

[5] A. Byszuk and W. Zabolotny. (2016). Simple axi4-lite bridges for ipbus and wishbone, [Online]. Available: https://opencores.org/projects/ax4lbr.