Access control system panel firmware

1.0

Generated by Doxygen 1.8.16

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

•

Chapter 3

Data Structure Documentation

3.1 acs_msg_data_auth_req_t Struct Reference

Data Fields

- uint32_t **user_id**

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

- acs_can_protocol.h

3.2 acs_msg_data_auth_resp_t Struct Reference

Data Fields

- uint32_t **user_id**

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

- acs_can_protocol.h

3.3 acs_msg_data_door_ctrl_t Struct Reference

Data Fields

- uint8_t **ctrl_command**

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

- acs_can_protocol.h

3.4 acs_msg_head_t Union Reference

Data Fields

- struct {
  uint32_t **src**: ACS_ADDR_BITS
  uint32_t **dst**: ACS_ADDR_BITS
  uint32_t **fc**: ACS_FC_BITS
  uint32_t **prio**: ACS_PRIO_BITS
  uint32_t **flags**: 3
  };
- uint32_t **scalar**

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

- acs_can_protocol.h

3.5 cache_item_t Union Reference

`#include <static_cache.h>`

Data Fields

- struct {
  uint32_t **value**: 2
  uint32_t **key**: 30
  };
- uint32_t **scalar**

3.5.1 Detailed Description

Public types/enumerations/variables

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

- static_cache.h

3.6 reader_conf_t Struct Reference

Data Fields

- TimerHandle_t **timer_open**
- TimerHandle_t **timer_ok**
- uint16_t **open_time_sec**
- uint16_t **gled_time_sec**
- uint8_t **enabled**
- uint8_t **door_open**

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

- reader.h

3.7 reader_wiring_t Struct Reference

Data Fields

- uint8_t **data_port**
- uint8_t **d0_pin**
- uint8_t **d1_pin**
- uint8_t **beep_port**
- uint8_t **beep_pin**
- uint8_t **gled_port**
- uint8_t **gled_pin**
- uint8_t **rled_port**
- uint8_t **rled_pin**
- uint8_t **relay_port**
- uint8_t **relay_pin**
- uint8_t **sensor_port**
- uint8_t **sensor_pin**

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

- reader.h

3.8 weigand26_buff_item_t Struct Reference

Data Fields

- uint8_t **source**
- weigand26_frame_t **frame**

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

- weigand.h

3.9 weigand26_frame_t Union Reference

Data Fields

- struct {
  uint32_t **odd_parity**: 1
  uint32_t **card_number**: 16
  uint32_t **facility_code**: 8
  uint32_t **even_parity**: 1
  uint32_t **__pad0__**: 6
  };
- uint32_t **value**

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

- weigand.h

3.10 weigand26_t Struct Reference

Data Fields

- weigand26_frame_t **frame_buffer**
- StreamBufferHandle_t **consumer_buffer**
- TimerHandle_t **timer**
- uint8_t **frame_buffer_ptr**
- uint8_t **port**
- uint8_t **pin_d0**
- uint8_t **pin_d1**
- uint8_t **id**

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

- weigand.c

Chapter 4

File Documentation

4.1 acs_can_protocol.h File Reference

ACS CAN protocol.

Data Structures

- union acs_msg_head_t
- struct acs_msg_data_auth_req_t
- struct acs_msg_data_auth_resp_t
- struct acs_msg_data_door_ctrl_t

Macros

- #define **ACS_MSGOBJ_SEND_DOOR_A** 0
- #define **ACS_MSGOBJ_SEND_DOOR_B** 1
- #define **ACS_MSGOBJ_RECV_DOOR_A** 2
- #define **ACS_MSGOBJ_RECV_DOOR_B** 3
- #define **ACS_MSGOBJ_RECV_BCAST** 4
- #define **ACS_PRIO_BITS** 3
- #define **ACS_FC_BITS** 6
- #define **ACS_ADDR_BITS** 10
- #define **ACS_BROADCAST_ADDR** ((1 << ACS_ADDR_BITS) - 1)
- #define **ACS_RESERVED_ADDR** 0
- #define **ACS_MSTR_FIRST_ADDR** 1
- #define **ACS_MSTR_LAST_ADDR** 3
- #define **ACS_PNL_FIRST_ADDR** 4
- #define **ACS_PNL_LAST_ADDR** (ACS_BROADCAST_ADDR - 1)
- #define **ACS_SRC_ADDR_OFFSET** 0
- #define **ACS_DST_ADDR_OFFSET** (ACS_SRC_ADDR_OFFSET + ACS_ADDR_BITS)
- #define **ACS_FC_OFFSET** (ACS_DST_ADDR_OFFSET + ACS_ADDR_BITS)
- #define **ACS_PRIO_OFFSET** (ACS_FC_OFFSET + ACS_FC_BITS)
- #define **ACS_SRC_ADDR_MASK** (((1 << ACS_ADDR_BITS) - 1) << ACS_SRC_ADDR_OFFSET)
- #define **ACS_DST_ADDR_MASK** (((1 << ACS_ADDR_BITS) - 1) << ACS_DST_ADDR_OFFSET)
- #define **ACS_FC_MASK** (((1 << ACS_FC_BITS) - 1) << ACS_FC_OFFSET)
- #define **ACS_PRIO_MASK** (((1 << ACS_PRIO_BITS) - 1) << ACS_PRIO_OFFSET)

- #define **FC_RESERVED** 0x0
- #define **FC_USER_AUTH_REQ** 0x1
- #define **FC_USER_NOT_AUTH_RESP** 0x2
- #define **FC_USER_AUTH_RESP** 0x3
- #define **FC_DOOR_CTRL** 0x4
- #define **FC_DOOR_STATUS** 0x5
- #define **FC_ALIVE** 0x6
- #define **ACS_MAX_PRIO** 0
- #define **ACS_LOW_PRIO** ((1 << ACS_PRIO_BITS) - 1)
- #define **PRIO_RESERVED** 0x0
- #define **PRIO_USER_AUTH_REQ** 0x2
- #define **PRIO_USER_AUTH_RESP_FAIL** 0x2
- #define **PRIO_USER_AUTH_RESP_OK** 0x2
- #define **PRIO_DOOR_CTRL** 0x3
- #define **PRIO_DOOR_STATUS** 0x4
- #define **PRIO_ALIVE** 0x1
- #define **DATA_DOOR_CTRL_REMOTE_UNLCK** 0x01
- #define **DATA_DOOR_CTRL_CLR_CACHE** 0x02
- #define **DATA_DOOR_STATUS_CLOSED** 0x01
- #define **DATA_DOOR_STATUS_OPEN** 0x02
- #define **ACS_MASTER_ALIVE_PERIOD_MS** 5000
- #define **ACS_MASTER_ALIVE_TIMEOUT_MS** 10000

4.1.1 Detailed Description

ACS CAN protocol.

Contains defines and data types for the protocol. ACS protocol uses messages with extended ID from CAN 2.0B specification.

Author

Petr Elexa

See also

LICENSE

4.2 brownout.c File Reference

Brown-out detection.

```
#include "brownout.h"
#include "board.h"
```

Functions

- void BOD_IRQHandler (void)

  Brown-out detection interrupt handler.
- void BOD_Init (void)

  Initialize brown-out detection.

4.2.1 Detailed Description

Brown-out detection.

Author

Petr Elexa

See also

LICENSE

4.2.2 Function Documentation

4.2.2.1 BOD_Init()

```
void BOD_Init (
            void  )
```

Initialize brown-out detection.

4.2.2.2 BOD_IRQHandler()

```
void BOD_IRQHandler (
            void  )
```

Brown-out detection interrupt handler.

4.3 brownout.h File Reference

Brown-out detection.

Functions

- void BOD_IRQHandler (void)

  Brown-out detection interrupt handler.
- void BOD_Init (void)

  Initialize brown-out detection.

4.3.1 Detailed Description

Brown-out detection.

Author

Petr Elexa

See also

LICENSE

4.3.2 Function Documentation

4.3.2.1 BOD_Init()

```
void BOD_Init (
            void  )
```

Initialize brown-out detection.

4.3.2.2 BOD_IRQHandler()

```
void BOD_IRQHandler (
            void  )
```

Brown-out detection interrupt handler.

4.4 can_term_driver.c File Reference

Interface for LPC11C24's on-board CCAN driver in ROM.

```
#include "can/can_term_driver.h"
#include <string.h>
```

Macros

- #define **CCAN_BCR_QUANTA**(x) ((x) & 0x3F)
- #define **CCAN_BCR_SJW**(x) (((x) & 0x3) << 6)
- #define **CCAN_BCR_TSEG1**(x) (((x) & 0x0F) << 8)
- #define **CCAN_BCR_TSEG2**(x) (((x) & 0x07) << 12)
- #define **CAN_CALC_SYNC_SEG** 1
- #define **TSEG1_MIN** 1
- #define **TSEG1_MAX** 13
- #define **TSEG2_MIN** 1
- #define **TSEG2_MAX** 8
- #define **SJW_MAX** 4
- #define **BRP_MIN** 1
- #define **BRP_MAX** 32

Functions

- static void _timing_calculate (uint32_t baud_rate, uint32_t *can_api_timing_cfg)
- static void **_timing_calculate_sp** (uint32_t baud_rate, uint32_t *can_api_timing_cfg)
- static void **_125_kbaud_75sp** (uint32_t *can_api_timing_cfg)
- void CAN_init (CCAN_CALLBACKS_T *ptr_callbacks, uint32_t baud_rate)

  Initializes CAN periphery.
- void CAN_recv_filter (uint8_t msgobj_num, uint32_t id, uint32_t mask, bool extended)

  Setup HW filter for received CAN messages.
- void CAN_recv_filter_all_ext (uint8_t msgobj_num)

  Setup HW filter to receive all extended frames ID (0-0x1FFFFFFF).
- void CAN_send_once (uint8_t msgobj_num, uint32_t id, uint8_t *data, uint8_t size)

  Send one time CAN message.
- void CAN_send_test (void)

  Send test message on CAN.
- void CAN_IRQHandler (void)

  CAN interrupt handler.

4.4.1 Detailed Description

Interface for LPC11C24's on-board CCAN driver in ROM.

Based on LPCOpen CCAN on-chip driver example.

Author

Petr Elexa

Note

4.4.2 Function Documentation

4.4.2.1 _timing_calculate()

```
static void _timing_calculate (
            uint32_t baud_rate,
            uint32_t * can_api_timing_cfg )  [static]
```

Private functions

4.4.2.2 CAN_init()

```
void CAN_init (
            CCAN_CALLBACKS_T * ptr_callbacks,
            uint32_t baud_rate )
```

Initializes CAN periphery.

Public functions

4.4.2.3 CAN_IRQHandler()

```
void CAN_IRQHandler (
            void  )
```

CAN interrupt handler.

```
The CAN interrupt handler must be provided by the user application.
It's function is to call the handler located in the ROM.
```

4.4.2.4 CAN_recv_filter()

```
void CAN_recv_filter (
            uint8_t msgobj_num,
            uint32_t id,
            uint32_t mask,
            bool extended )
```

Setup HW filter for received CAN messages.

```
The filter matches when <recieved_id> & mask == id & mask.
Non-matching messages are dropped.
```

Parameters

msgobj_num	... number of message object (0-31) to setup.
id	... CAN message ID
mask	... CAN ID mask
extended	... use extended frame ID (29bit) if true

4.4.2.5 CAN_recv_filter_all_ext()

```
void CAN_recv_filter_all_ext (
             uint8_t msgobj_num )
```

Setup HW filter to receive all extended frames ID (0-0x1FFFFFFF).

```
   This setups HW filter for received CAN message.
   The filter matches when <recieved_id> & mask == id & mask.
   Non-matching messages are dropped.
```

Parameters

msgobj_num	... number of message object (0-31).

4.4.2.6 CAN_send_once()

```
void CAN_send_once (
             uint8_t msgobj_num,
             uint32_t id,
             uint8_t * data,
             uint8_t size )
```

Send one time CAN message.

Parameters

msgobj_num	... number of message object (0-31).
id	... CAN arbitration ID.
data	... pointer to data to send.
size	... size of data.

4.4.2.7 CAN_send_test()

```
void CAN_send_test (
```

```
            void  )
```

Send test message on CAN.

4.5 can_term_driver.h File Reference

Interface for LPC11C24's on-board CCAN driver in ROM.

```
#include "board.h"
#include <stdint.h>
```

Macros

- #define **CCAN_MSG_OBJ_FIRST** 0
- #define **CCAN_MSG_OBJ_LAST** 31
- #define **CAN_EXT_ID_BIT_MASK** 0x1FFFFFFFUL
- #define **CAN_DLC_MAX** 8

Functions

- void CAN_init (CCAN_CALLBACKS_T *ptr_callbacks, uint32_t baud_rate)

  Initializes CAN periphery.
- void CAN_recv_filter (uint8_t msgobj_num, uint32_t id, uint32_t mask, bool extended)

  Setup HW filter for received CAN messages.
- void CAN_recv_filter_all_ext (uint8_t msgobj_num)

  Setup HW filter to receive all extended frames ID (0-0x1FFFFFFF).
- void CAN_send_once (uint8_t msgobj_num, uint32_t id, uint8_t *data, uint8_t size)

  Send one time CAN message.
- void CAN_send_test (void)

  Send test message on CAN.
- void CAN_IRQHandler (void)

  CAN interrupt handler.

4.5.1 Detailed Description

Interface for LPC11C24's on-board CCAN driver in ROM.

Author

Petr Elexa

See also

LICENSE

4.5.2 Function Documentation

4.5.2.1 CAN_init()

```
void CAN_init (
            CCAN_CALLBACKS_T * ptr_callbacks,
            uint32_t baud_rate )
```

Initializes CAN periphery.

```
Function should be executed before using the CAN bus.
Initializes the CAN controller, on-chip drivers.
```

Parameters

ptr_callbacks	... pointer to callback structure
baud_rate	... CAN baud rate to use

Public functions

4.5.2.2 CAN_IRQHandler()

```
void CAN_IRQHandler (
            void  )
```

CAN interrupt handler.

```
The CAN interrupt handler must be provided by the user application.
It's function is to call the handler located in the ROM.
```

4.5.2.3 CAN_recv_filter()

```
void CAN_recv_filter (
            uint8_t msgobj_num,
            uint32_t id,
            uint32_t mask,
            bool extended )
```

Setup HW filter for received CAN messages.

```
The filter matches when <recieved_id> & mask == id & mask.
Non-matching messages are dropped.
```

Parameters

msgobj_num	... number of message object (0-31) to setup.
id	... CAN message ID
mask	... CAN ID mask
extended	... use extended frame ID (29bit) if true

4.5.2.4 CAN_recv_filter_all_ext()

```
void CAN_recv_filter_all_ext (
            uint8_t msgobj_num )
```

Setup HW filter to receive all extended frames ID (0-0x1FFFFFFF).

```
   This setups HW filter for received CAN message.
   The filter matches when <recieved_id> & mask == id & mask.
   Non-matching messages are dropped.
```

Parameters

msgobj_num	... number of message object (0-31).

4.5.2.5 CAN_send_once()

```
void CAN_send_once (
            uint8_t msgobj_num,
            uint32_t id,
            uint8_t * data,
            uint8_t size )
```

Send one time CAN message.

Parameters

msgobj_num	... number of message object (0-31).
id	... CAN arbitration ID.
data	... pointer to data to send.
size	... size of data.

4.5.2.6 CAN_send_test()

```
void CAN_send_test (
```

```
            void  )
```

Send test message on CAN.

4.6 reader.c File Reference

RFID reader driver.

```
#include <reader.h>
```

Macros

- #define **DOOR_SENSOR_VALUE_OPEN** LOG_LOW
- #define **DOOR_OPEN** 1
- #define **DOOR_CLOSED** 0

Functions

- static void **_timer_open_callback** (TimerHandle_t pxTimer)
- static void **_timer_ok_callback** (TimerHandle_t pxTimer)
- void reader_init (uint8_t idx)

  Initialize reader driver.
- void reader_deinit (uint8_t idx)

  Disable reader driver.
- int8_t reader_get_request_from_buffer (uint32_t ∗user_id, uint16_t time_to_wait_ms)

  Disable reader driver.
- void reader_unlock (uint8_t idx, bool with_beep, bool with_ok_led)

  Unlock door belonging to reader.
- bool reader_is_door_open (uint8_t reader_idx)

  Check door status.
- void **reader_sensor_int_handler** (uint8_t port, uint32_t int_states)
- void **PIOINT0_IRQHandler** (void)
- void **PIOINT1_IRQHandler** (void)

Variables

- static StreamBufferHandle_t **_reader_buffer**
- static const reader_wiring_t **_reader_wiring** [ACS_READER_MAXCOUNT]
- reader_conf_t reader_conf [ACS_READER_MAXCOUNT]

  Configuration of readers.

4.6.1 Detailed Description

RFID reader driver.

```
This RFID reader implementation uses Weigand26 card reader protocol.
```

Author

Petr Elexa

See also

LICENSE

4.6.2 Function Documentation

4.6.2.1 reader_deinit()

```
void reader_deinit (
            uint8_t idx )
```

Disable reader driver.

Parameters

idx	... reader index

4.6.2.2 reader_get_request_from_buffer()

```
int8_t reader_get_request_from_buffer (
            uint32_t * user_id,
            uint16_t time_to_wait_ms )
```

Disable reader driver.

Parameters

user↩ _id	... 24-bit number
idx	... reader index
...	time_to_wait_ms ... time to wait for request

4.6.2.3 reader_init()

```
void reader_init (
            uint8_t idx )
```

Initialize reader driver.

4.6.2.4 reader_is_door_open()

```
bool reader_is_door_open (
            uint8_t reader_idx )
```

Check door status.

Parameters

idx	... reader index

Returns

true if door is open

4.6.2.5 reader_unlock()

```
void reader_unlock (
            uint8_t idx,
            bool with_beep,
            bool with_ok_led )
```

Unlock door belonging to reader.

Parameters

idx	... reader index
with_beep	... true for sound signal
with_ok_led	... true for visual signal

4.6.3 Variable Documentation

4.6.3.1 reader_conf

```
reader_conf_t reader_conf[ACS_READER_MAXCOUNT]
```

Initial value:

```
=
{
  {
    .timer_ok = NULL,
    .timer_open = NULL,
    .open_time_sec = ACS_READER_A_OPEN_TIME_MS,
    .gled_time_sec = ACS_READER_A_OK_GLED_TIME_MS,
    .enabled = ACS_READER_A_ENABLED,
    .door_open = DOOR_CLOSED
  },
  {
    .timer_ok = NULL,
    .timer_open = NULL,
    .open_time_sec = ACS_READER_B_OPEN_TIME_MS,
    .gled_time_sec = ACS_READER_B_OK_GLED_TIME_MS,
    .enabled = ACS_READER_B_ENABLED,
    .door_open = DOOR_CLOSED
  }
}
```

Configuration of readers.

4.7 reader.h File Reference

RFID card reader driver.

```
#include <stdint.h>
#include <stdbool.h>
#include "board.h"
#include "FreeRTOS.h"
#include "stream_buffer.h"
#include "timers.h"
#include "weigand.h"
```

Data Structures

- struct reader_conf_t
- struct reader_wiring_t

Enumerations

- enum **reader_mode_t** { **READER_MODE_DEF** = 0, **READER_MODE_LOCKED**, **READER_MODE_LE↩ ARN** }

Functions

- void reader_init (uint8_t idx)

  Initialize reader driver.
- void reader_deinit (uint8_t idx)

  Disable reader driver.
- int8_t reader_get_request_from_buffer (uint32_t *user_id, uint16_t time_to_wait_ms)

  Disable reader driver.
- void reader_unlock (uint8_t idx, bool with_beep, bool with_ok_led)

  Unlock door belonging to reader.
- bool reader_is_door_open (uint8_t reader_idx)

  Check door status.

Variables

- reader_conf_t reader_conf [ACS_READER_COUNT]
  Configuration of readers.

4.7.1 Detailed Description

RFID card reader driver.

Author

Petr Elexa

See also

LICENSE

4.7.2 Function Documentation

4.7.2.1 reader_deinit()

```
void reader_deinit (
            uint8_t idx )
```

Disable reader driver.

Parameters

idx	... reader index

4.7.2.2 reader_get_request_from_buffer()

```
int8_t reader_get_request_from_buffer (
            uint32_t * user_id,
            uint16_t time_to_wait_ms )
```

Disable reader driver.

Parameters

user_id	... 24-bit number
idx	... reader index
...	time_to_wait_ms ... time to wait for request

4.7.2.3 reader_init()

```
void reader_init (
            uint8_t idx )
```

Initialize reader driver.

4.7.2.4 reader_is_door_open()

```
bool reader_is_door_open (
            uint8_t reader_idx )
```

Check door status.

Parameters

idx	... reader index

Returns

true if door is open

4.7.2.5 reader_unlock()

```
void reader_unlock (
            uint8_t idx,
            bool with_beep,
            bool with_ok_led )
```

Unlock door belonging to reader.

Parameters

idx	... reader index
with_beep	... true for sound signal
with_ok_led	... true for visual signal

4.8 start.c File Reference

Main entry point.

```
#include "can/can_term_driver.h"
#include "terminal.h"
#include "board.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "watchdog.h"
#include "brownout.h"
#include "storage.h"
```

Functions

- static void _check_system_stack_size (void)
- int main (void)
- void **vConfigureTimerForRunTimeStats** (void)
- void **vApplicationMallocFailedHook** (void)
- void **vApplicationIdleHook** (void)
- void **vApplicationStackOverflowHook** (TaskHandle_t pxTask, char ∗pcTaskName)
- void **vApplicationTickHook** (void)

4.8.1 Detailed Description

Main entry point.

Contains start-up sequence.

Author

Petr Elexa

See also

LICENSE

4.8.2 Function Documentation

4.8.2.1 _check_system_stack_size()

```
static void _check_system_stack_size (
            void  )  [static]
```

Private types/enumerations/variables Public types/enumerations/variables Private functions

4.8.2.2 main()

```
int main (
            void  )
```

Public functions

4.9 static_cache.c File Reference

Statically allocated cache for user IDs.

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

4.9.1 Detailed Description

Statically allocated cache for user IDs.

It is similar to Set Associative Cache. Place key and value into sets by key hash. Each set is a sorted array.

Author

Petr Elexa

See also

LICENSE

4.10 static_cache.h File Reference

Statically allocated cache for user IDs.

```
#include <stdint.h>
#include <stdbool.h>
#include "terminal_config.h"
```

Data Structures

- union cache_item_t

Macros

- #define STATIC_CACHE_SETS 4
- #define **STATIC_CACHE_SET_CAP** 128
- #define **STATIC_CACHE_CAPACITY** (STATIC_CACHE_SET_CAP ∗ STATIC_CACHE_SETS)

Functions

- bool static_cache_get (cache_item_t *ptr_kv)
  Retrieve item from the cache.
- void static_cache_insert (const cache_item_t kv)
  Insert item to the cache.
- void static_cache_erase (const cache_item_t kv)
  Erase item from the cache.
- void static_cache_reset (void)
  Clear all data in the cache.
- cache_item_t static_cache_convert (uint32_t key, uint32_t value)
  Create cache item from key and value parameters.

4.10.1 Detailed Description

Statically allocated cache for user IDs.

It is similar to Set Associative Cache. Divide key and value into sets by key hash. Each set is a sorted array.

The key in cache can be up to 30 bits in size.

Author

Petr Elexa

See also

LICENSE

4.10.2 Macro Definition Documentation

4.10.2.1 STATIC_CACHE_SETS

```
#define STATIC_CACHE_SETS 4
```

Configuration of the static cache.

4.10.3 Function Documentation

4.10.3.1 static_cache_convert()

```
cache_item_t static_cache_convert (
            uint32_t key,
            uint32_t value )
```

Create cache item from key and value parameters.

Parameters

key	... Key for the value.
value	... Value for key to use.

Returns

Cache item structure.

4.10.3.2 static_cache_erase()

```
void static_cache_erase (
            const cache_item_t kv )
```

Erase item from the cache.

```
Complexity is O(log(STATIC_CACHE_SET_CAP) + 2*(STATIC_CACHE_SET_CAP)).
```

Parameters

kv	... Key containing key to be erased. The item will erased if key is found.

4.10.3.3 static_cache_get()

```
bool static_cache_get (
            cache_item_t * ptr_kv )
```

Retrieve item from the cache.

Public functions

```
Complexity is O(log(STATIC_CACHE_SET_CAP)).
```

Parameters

ptr_kv	... Pointer to cache_item_t containing key. The value will be filled in if key is found.

Returns

true if key is found.

4.10.3.4 static_cache_insert()

```
void static_cache_insert (
            const cache_item_t kv )
```

Insert item to the cache.

Will overwrite items already in cache if the cache is full. Will also update item with same key.

Complexity is O(log(STATIC_CACHE_SET_CAP) + 2*(STATIC_CACHE_SET_CAP)).

Parameters

kv	... Key and value to be inserted.

4.10.3.5 static_cache_reset()

```
void static_cache_reset (
            void  )
```

Clear all data in the cache.

Resets all data in cache to 0s.
Complexity is O(STATIC_CACHE_CAPACITY).

4.11 storage.c File Reference

Storage implementation.

```
#include "storage.h"
#include "FreeRTOS.h"
#include "task.h"
```

Functions

- static void **_init_I2C_pins** (void)
- static void **_wait_for_dev_ready** (void)
- void storage_init (void)

  Initialize I2C bus for storage.
- bool storage_read_word_le (const uint8_t addr, uint16_t *data)

  Read a word from address (little-endian).
- bool storage_write_word_le (const uint8_t addr, const uint16_t data)

  Write a word to address (little-endian).
- bool storage_read_byte (const uint8_t addr, uint8_t *data)

  Read a byte from address.
- bool storage_write_byte (const uint8_t addr, const uint8_t data)

  Write a byte to address.
- void I2C_IRQHandler (void)

  I2C Interrupt Handler.

4.11.1 Detailed Description

Storage implementation.

```
Supports I2C EEPROMs and I/O expanders.
```

Author

Petr Elexa

See also

LICENSE

4.11.2 Function Documentation

4.11.2.1 I2C_IRQHandler()

```
void I2C_IRQHandler (
            void  )
```

I2C Interrupt Handler.

Returns

None

4.11.2.2 storage_init()

```
void storage_init (
            void  )
```

Initialize I2C bus for storage.

4.11.2.3 storage_read_byte()

```
bool storage_read_byte (
            const uint8_t addr,
            uint8_t * data )
```

Read a byte from address.

Returns

true if succeeded

4.11.2.4 storage_read_word_le()

```
bool storage_read_word_le (
            const uint8_t addr,
            uint16_t * data )
```

Read a word from address (little-endian).

Returns

true if succeeded

4.11.2.5 storage_write_byte()

```
bool storage_write_byte (
            const uint8_t addr,
            const uint8_t data )
```

Write a byte to address.

Returns

true if succeeded

4.11.2.6 storage_write_word_le()

```
bool storage_write_word_le (
            const uint8_t addr,
            const uint16_t data )
```

Write a word to address (little-endian).

Returns

true if succeeded

4.12 storage.h File Reference

Storage implementation.

```
#include "board.h"
#include <stdint.h>
#include <stdbool.h>
```

Macros

- #define **STORE_I2C_DEV** I2C0

Functions

- void storage_init (void)

  Initialize I2C bus for storage.
- bool storage_read_word_le (const uint8_t addr, uint16_t *data)

  Read a word from address (little-endian).
- bool storage_write_word_le (const uint8_t addr, const uint16_t data)

  Write a word to address (little-endian).
- bool storage_read_byte (const uint8_t addr, uint8_t *data)

  Read a byte from address.
- bool storage_write_byte (const uint8_t addr, const uint8_t data)

  Write a byte to address.

4.12.1 Detailed Description

Storage implementation.

```
Supports I2C EEPROMs and I/O expanders.
```

Author

Petr Elexa

See also

LICENSE

4.12.2 Function Documentation

4.12.2.1 storage_init()

```
void storage_init (
            void  )
```

Initialize I2C bus for storage.

4.12.2.2 storage_read_byte()

```
bool storage_read_byte (
            const uint8_t addr,
            uint8_t * data )
```

Read a byte from address.

Returns

true if succeeded

4.12.2.3 storage_read_word_le()

```
bool storage_read_word_le (
            const uint8_t addr,
            uint16_t * data )
```

Read a word from address (little-endian).

Returns

true if succeeded

4.12.2.4 storage_write_byte()

```
bool storage_write_byte (
            const uint8_t addr,
            const uint8_t data )
```

Write a byte to address.

Returns

true if succeeded

4.12.2.5 storage_write_word_le()

```
bool storage_write_word_le (
            const uint8_t addr,
            const uint16_t data )
```

Write a word to address (little-endian).

Returns

true if succeeded

4.13 terminal.c File Reference

Terminal client for access control system (ACS).

```
#include "terminal.h"
#include "board.h"
#include "weigand.h"
#include "static_cache.h"
#include "FreeRTOS.h"
#include "task.h"
#include "stream_buffer.h"
#include "can/can_term_driver.h"
#include "acs_can_protocol.h"
#include <stdio.h>
#include <string.h>
```

Typedefs

- typedef cache_item_t **term_cache_item_t**

Enumerations

- enum **term_cache_reader** { **cache_reader_none** = 0, **cache_reader_A** = 1, **cache_reader_B** = 2, **cache_reader_all** = 3 }

Functions

- static uint8_t map_reader_idx_to_cache (uint8_t reader_idx)
- static void **_terminal_user_authorized** (uint8_t reader_idx)
- static void **__terminal_user_not_authorized** (uint8_t reader_idx)
- static void **_timer_callback** (TimerHandle_t pxTimer)
- void term_can_error (uint32_t error_info)

  CAN error callback.
- void term_can_send (uint8_t msg_obj_num)

  CAN transmit callback.
- void term_can_recv (uint8_t msg_obj_num)

  CAN receive callback.
- static void **terminal_send_door_status** (uint8_t reader_idx, bool is_open)
- static void **terminal_request_auth** (uint32_t user_id, uint8_t reader_idx)
- static void **terminal_user_identified** (uint32_t user_id, uint8_t reader_idx)
- static void **terminal_task** (void *pvParameters)
- void terminal_init (void)

  Initialize terminal.
- void terminal_reconfigure (reader_conf_t *reader_cfg, uint8_t reader_idx)

  Reconfigure terminal's reader at runtime.

Variables

- static const uint16_t USER_REQUEST_WAIT_MS = 1500
- static uint16_t **_act_master** = ACS_RESERVED_ADDR
- static bool **_master_timeout** = true
- static TimerHandle_t **_act_timer** = NULL
- static const uint32_t **_act_timer_id** = TERMINAL_TIMER_ID
- static bool **_last_door_state** [ACS_READER_COUNT] = {false, false}

4.13.1 Detailed Description

Terminal client for access control system (ACS).

Author

Petr Elexa

See also

LICENSE

4.13.2 Function Documentation

4.13.2.1 map_reader_idx_to_cache()

```
static uint8_t map_reader_idx_to_cache (
            uint8_t reader_idx )  [inline], [static]
```

Private functions

4.13.2.2 term_can_error()

```
void term_can_error (
            uint32_t error_info )
```

CAN error callback.

Public functions

4.13.2.3 term_can_recv()

```
void term_can_recv (
            uint8_t msg_obj_num )
```

CAN receive callback.

Function is executed by the Callback handler after a CAN message has been received.

Parameters

msg_obj_num	Contains the number of the message object that triggered the CAN receive callback.

4.13.2.4 term_can_send()

```
void term_can_send (
            uint8_t msg_obj_num )
```

CAN transmit callback.

Function is executed by the Callback handler after a CAN message has been transmitted.

Parameters

msg_obj_num	Contains the number of the message object that triggered the CAN transmit callback.

4.13.2.5 terminal_init()

```
void terminal_init (
            void  )
```

Initialize terminal.

Will initialize terminal configuration and all readers and create terminal task.

Parameters

reader_cfg	
id	

4.13.2.6 terminal_reconfigure()

```
void terminal_reconfigure (
            reader_conf_t * reader_cfg,
            uint8_t id )
```

Reconfigure terminal's reader at runtime.

Parameters

reader_cfg	
id	

4.13.3 Variable Documentation

4.13.3.1 USER_REQUEST_WAIT_MS

```
const uint16_t USER_REQUEST_WAIT_MS = 1500  [static]
```

Private types/enumerations/variables

4.14 terminal.h File Reference

Terminal client for access control system (ACS).

```
#include <reader.h>
#include <stdint.h>
```

Functions

- void terminal_init (void)
  Initialize terminal.
- void terminal_reconfigure (reader_conf_t *reader_cfg, uint8_t id)
  Reconfigure terminal's reader at runtime.
- void term_can_recv (uint8_t msg_obj_num)
  CAN receive callback.
- void term_can_send (uint8_t msg_obj_num)
  CAN transmit callback.
- void term_can_error (uint32_t error_info)
  CAN error callback.

4.14.1 Detailed Description

Terminal client for access control system (ACS).

Author

Petr Elexa

See also

LICENSE

4.14.2 Function Documentation

4.14.2.1 term_can_error()

```
void term_can_error (
            uint32_t error_info )
```

CAN error callback.

Function is executed by the Callback handler after an error has occurred on the CAN bus.

Parameters

error_info	Contains the error code that triggered the CAN error callback.

Public functions

4.14.2.2 term_can_recv()

```
void term_can_recv (
            uint8_t msg_obj_num )
```

CAN receive callback.

Function is executed by the Callback handler after a CAN message has been received.

Parameters

msg_obj_num	Contains the number of the message object that triggered the CAN receive callback.

4.14.2.3 term_can_send()

```
void term_can_send (
            uint8_t msg_obj_num )
```

CAN transmit callback.

Function is executed by the Callback handler after a CAN message has been transmitted.

Parameters

msg_obj_num	Contains the number of the message object that triggered the CAN transmit callback.

4.14.2.4 terminal_init()

```
void terminal_init (
            void  )
```

Initialize terminal.

Will initialize terminal configuration and all readers and create terminal task.

Parameters

reader_cfg	
id	

4.14.2.5 terminal_reconfigure()

```
void terminal_reconfigure (
            reader_conf_t * reader_cfg,
            uint8_t id )
```

Reconfigure terminal's reader at runtime.

Parameters

reader_cfg	
id	

4.15 terminal_config.c File Reference

Configuration of hardware and software.

```
#include "terminal_config.h"
#include "storage.h"
#include "acs_can_protocol.h"
```

Macros

- #define **ACS_ADDR_BIT_MASK** ((1 << ACS_ADDR_BITS) - 1)

Functions

- uint16_t get_reader_a_addr (void)

  Get network address of door A.
- uint16_t get_reader_b_addr (void)

  Get network address of door B.
- static bool **_load_acs_addrs_from_ext_stor** (void)
- static bool **_save_acs_addrs_from_ext_stor** (void)
- void set_reader_addr (uint16_t acs_addr)

  Address setter.
- bool terminal_config_init (void)

  Initialize configuration for terminal.

Variables

- uint16_t **_READER_A_ADDR** = 0x4
- uint16_t **_READER_B_ADDR** = 0x5

4.15.1 Detailed Description

Configuration of hardware and software.

Author

Petr Elexa

See also

LICENSE

4.15.2 Function Documentation

4.15.2.1 get_reader_a_addr()

```
uint16_t get_reader_a_addr (
            void  )  [inline]
```

Get network address of door A.

4.15.2.2 get_reader_b_addr()

```
uint16_t get_reader_b_addr (
            void  )  [inline]
```

Get network address of door B.

4.15.2.3 set_reader_addr()

```
void set_reader_addr (
            const uint16_t acs_addr )
```

Address setter.

Parameters

acs_addr	... ACS network address (odd or even).

4.15.2.4 terminal_config_init()

```
bool terminal_config_init (
            void  )
```

Initialize configuration for terminal.

```
Reads address from storage.
```

Returns

true if succeeded

4.16 terminal_config.h File Reference

Terminal client configuration (hardware and software). ∗.

```
#include <stdint.h>
#include <stdbool.h>
```

Macros

- #define **DEVEL_BOARD**
- #define **ENABLE_LOCAL_ACS_ADDR_WRITE** 0
- #define **CACHING_ENABLED** 0
- #define **CAN_BAUD_RATE** 125000
- #define **STORE_I2C_BUS_FREQ** 400000
- #define **STORE_I2C_SLAVE_ADDR** 0x50
- #define **ACS_READER_A_IDX** 0
- #define **ACS_READER_B_IDX** 1
- #define **ACS_READER_COUNT** 2
- #define **BEEP_ON_SUCCESS** false
- #define **OK_LED_ON_SUCCESS** true
- #define **SENSOR_IS_NO** 0
- #define **SENSOR_IS_NC** 1
- #define **DOOR_SENSOR_TYPE** SENSOR_IS_NO
- #define **ACS_COMM_STATUS_LED_PORT** 0
- #define **ACS_COMM_STATUS_LED_PIN** 6
- #define **IAP_READ_UID** 58
- #define **PTR_READER_FIRST_ADDR** 0x0
- #define **STORE_DEV_BUSY_FOR** 50
- #define **ACS_READER_A_ENABLED** true
- #define **ACS_READER_A_DATA_PORT** 3
- #define **ACS_READER_A_D0_PIN** 2
- #define **ACS_READER_A_D1_PIN** 1
- #define **ACS_READER_A_BEEP_PORT** 2
- #define **ACS_READER_A_BEEP_PIN** 7
- #define **ACS_READER_A_GLED_PORT** 2
- #define **ACS_READER_A_GLED_PIN** 1
- #define **ACS_READER_A_RLED_PORT** 2

- #define **ACS_READER_A_RLED_PIN** 0
- #define **ACS_READER_A_RELAY_PORT** 1
- #define **ACS_READER_A_RELAY_PIN** 10
- #define **ACS_READER_A_SENSOR_PORT** 1
- #define **ACS_READER_A_SENSOR_PIN** 8
- #define **ACS_READER_A_OPEN_TIME_MS** 8000
- #define **ACS_READER_A_OK_GLED_TIME_MS** 4000
- #define **ACS_READER_B_ENABLED** true
- #define **ACS_READER_B_DATA_PORT** 2
- #define **ACS_READER_B_D0_PIN** 8
- #define **ACS_READER_B_D1_PIN** 6
- #define **ACS_READER_B_BEEP_PORT** 2
- #define **ACS_READER_B_BEEP_PIN** 10
- #define **ACS_READER_B_GLED_PORT** 2
- #define **ACS_READER_B_GLED_PIN** 3
- #define **ACS_READER_B_RLED_PORT** 2
- #define **ACS_READER_B_RLED_PIN** 2
- #define **ACS_READER_B_RELAY_PORT** 1
- #define **ACS_READER_B_RELAY_PIN** 11
- #define **ACS_READER_B_SENSOR_PORT** 1
- #define **ACS_READER_B_SENSOR_PIN** 5
- #define **ACS_READER_B_OPEN_TIME_MS** 8000
- #define **ACS_READER_B_OK_GLED_TIME_MS** 4000
- #define **WEIGAND_DEVICE_LIMIT** 4
- #define **SERIAL_DEVICE_LIMIT** 0
- #define **ACS_READER_MAXCOUNT** 2
- #define **TERMINAL_TIMER_ID** 15
- #define **HW_WATCHDOG_TIMEOUT** 1
- #define **LOG_HIGH** 1
- #define **LOG_LOW** 0

Functions

- uint16_t get_reader_a_addr (void)

  Get network address of door A.
- uint16_t get_reader_b_addr (void)

  Get network address of door B.
- void set_reader_addr (const uint16_t acs_addr)

  Address setter.
- bool terminal_config_init (void)

  Initialize configuration for terminal.

Variables

- unsigned int **TERMINAL_UID** [5]

4.16.1 Detailed Description

Terminal client configuration (hardware and software). ∗.

Author

Petr Elexa

See also

LICENSE

4.16.2 Function Documentation

4.16.2.1 get_reader_a_addr()

```
uint16_t get_reader_a_addr (
            void  )  [inline]
```

Get network address of door A.

4.16.2.2 get_reader_b_addr()

```
uint16_t get_reader_b_addr (
            void  )  [inline]
```

Get network address of door B.

4.16.2.3 set_reader_addr()

```
void set_reader_addr (
            const uint16_t acs_addr )
```

Address setter.

Parameters

acs_addr	... ACS network address (odd or even).

4.16.2.4 terminal_config_init()

```
bool terminal_config_init (
            void  )
```

Initialize configuration for terminal.

```
Reads address from storage.
```

Returns

true if succeeded

4.17 watchdog.c File Reference

HW watchdog.

```
#include "watchdog.h"
#include "board.h"
```

Functions

- void WDT_IRQHandler (void)
  Watchdog timer interrupt handler.
- void WDT_Init (uint8_t timeout)
  Initialize Watchdog timer.
- void WDT_Feed (void)
  Feed watchdog timer.

4.17.1 Detailed Description

HW watchdog.

Author

Petr Elexa

Note

4.17.2 Function Documentation

4.17.2.1 WDT_Feed()

```
void WDT_Feed (
             void  )
```

Feed watchdog timer.

4.17.2.2 WDT_Init()

```
void WDT_Init (
             uint8_t timeout )
```

Initialize Watchdog timer.

```
   After init watchdog must be reloaded by @ref WDT_Feed.
```

Parameters

timeout	... time to reset or interrupt from watchdog timer.

4.17.2.3 WDT_IRQHandler()

```
void WDT_IRQHandler (
             void  )
```

Watchdog timer interrupt handler.

Public functions

4.18 watchdog.h File Reference

HW watchdog.

```
#include <stdint.h>
```

Functions

- void WDT_IRQHandler (void)

  Watchdog timer interrupt handler.
- void WDT_Init (uint8_t timeout)

  Initialize Watchdog timer.
- void WDT_Feed (void)

  Feed watchdog timer.

4.18.1 Detailed Description

HW watchdog.

Author

Petr Elexa

See also

LICENSE

4.18.2 Function Documentation

4.18.2.1 WDT_Feed()

```
void WDT_Feed (
            void  )
```

Feed watchdog timer.

4.18.2.2 WDT_Init()

```
void WDT_Init (
            uint8_t timeout )
```

Initialize Watchdog timer.

```
After init watchdog must be reloaded by @ref WDT_Feed.
```

Parameters

timeout	... time to reset or interrupt from watchdog timer.

4.18.2.3 WDT_IRQHandler()

```
void WDT_IRQHandler (
            void  )
```

Watchdog timer interrupt handler.

```
Handles watchdog timer timeout events.
Will be called only in debug mode.
```

Public functions

4.19 weigand.c File Reference

Wiegand26 interface driver.

```
#include "weigand.h"
#include "board.h"
#include "timers.h"
#include <limits.h>
```

Data Structures

- struct weigand26_t

Functions

- static void **weigand_frame_timeout** (TimerHandle_t pxTimer)
- void weigand_init (StreamBufferHandle_t buffer, uint8_t id, uint8_t dx_port, uint8_t d0_pin, uint8_t d1_pin)

  Initialize Wiegand driver.
- void weigand_disable (uint8_t dx_port, uint8_t d0_pin, uint8_t d1_pin)

  Disable Wiegand driver.
- bool **weigand_pending_frame** (weigand26_t *device)
- weigand26_frame_t **weigand_get_frame** (weigand26_t *device)
- uint32_t weigand_get_facility (weigand26_frame_t frame)

  Parse facility code from frame.
- uint32_t weigand_get_card (weigand26_frame_t frame)

  Parse card number from frame.
- bool weigand_is_parity_ok (weigand26_frame_t frame)

  Check frame parity.
- static void **_wake_timer_on_frame_start** (weigand26_t *device)
- void **weigand_int_handler** (weigand26_t *device)
- void **PIOINT2_IRQHandler** (void)
- void **PIOINT3_IRQHandler** (void)

Variables

- static weigand26_t **device** [WEIGAND_DEVICE_LIMIT] = {0}

4.19.1 Detailed Description

Wiegand26 interface driver.

500 b/s transfer rate data pulse 40-70us data interval >2ms

Only Wiegand 26bit format Frame: | even parity (1b) | facility code (8b) | card number (16b) | odd parity (1b) | transmission duration ~52ms

Author

Petr Elexa

See also

LICENSE

4.19.2 Function Documentation

4.19.2.1 weigand_disable()

```
void weigand_disable (
            uint8_t dx_port,
            uint8_t d0_pin,
            uint8_t d1_pin )
```

Disable Wiegand driver.

Parameters

dx_port	... Port number for data signals.
d0_pin	... Pin for 0's data signal.
d1_pin	... Pin for 1's data signal.

4.19.2.2 weigand_get_card()

```
uint32_t weigand_get_card (
            weigand26_frame_t frame )
```

Parse card number from frame.

Parameters

frame	... Wiegand26 data frame.

Returns

card number

4.19.2.3 weigand_get_facility()

```
uint32_t weigand_get_facility (
            weigand26_frame_t frame )
```

Parse facility code from frame.

Parameters

frame	... Wiegand26 data frame.

Returns

facility code

4.19.2.4 weigand_init()

```
void weigand_init (
            StreamBufferHandle_t buffer,
            uint8_t id,
            uint8_t dx_port,
            uint8_t d0_pin,
            uint8_t d1_pin )
```

Initialize Wiegand driver.

Note

One buffer for each consumer is preferred.

Parameters

buffer	... Receive buffer for frames from RFID card/tags. See weigand26_buff_item_t
id	... interface identification
dx_port	... Port number for data signals.
d0_pin	... Pin for 0's data signal.
d1_pin	... Pin for 1's data signal.

4.19.2.5 weigand_is_parity_ok()

```
bool weigand_is_parity_ok (
            weigand26_frame_t frame )
```

Check frame parity.

Parameters

frame	... Wiegand26 data frame.

Returns

true if parity is valid

4.20 weigand.h File Reference

Wiegand26 interface driver.

```
#include <stdbool.h>
#include <stdint.h>
#include "FreeRTOS.h"
#include "task.h"
#include "stream_buffer.h"
```

Data Structures

- union weigand26_frame_t
- struct weigand26_buff_item_t

Macros

- #define **WEIGAND26_FRAME_SIZE** 26
- #define **WEIGAND26_FRAME_TIME_LIMIT** 80
- #define **WEIGAND26_BUFF_ITEM_SIZE** sizeof(weigand26_buff_item_t)

Functions

- void weigand_init (StreamBufferHandle_t buffer, uint8_t id, uint8_t dx_port, uint8_t d0_pin, uint8_t d1_pin)

  Initialize Wiegand driver.
- void weigand_disable (uint8_t dx_port, uint8_t d0_pin, uint8_t d1_pin)

  Disable Wiegand driver.
- uint32_t weigand_get_facility (weigand26_frame_t frame)

  Parse facility code from frame.
- uint32_t weigand_get_card (weigand26_frame_t frame)

  Parse card number from frame.
- bool weigand_is_parity_ok (weigand26_frame_t frame)

  Check frame parity.

4.20.1 Detailed Description

Wiegand26 interface driver.

500 b/s transfer rate data pulse 40-70us data interval >2ms

Only Wiegand 26bit format Frame: | even parity (1b) | facility code (8b) | card number (16b) | odd parity (1b) | transmission duration ~52ms

Author

Petr Elexa

See also

LICENSE

4.20.2 Function Documentation

4.20.2.1 weigand_disable()

```
void weigand_disable (
            uint8_t dx_port,
            uint8_t d0_pin,
            uint8_t d1_pin )
```

Disable Wiegand driver.

Parameters

dx_port	... Port number for data signals.
d0_pin	... Pin for 0's data signal.
d1_pin	... Pin for 1's data signal.

4.20.2.2 weigand_get_card()

```
uint32_t weigand_get_card (
            weigand26_frame_t frame )
```

Parse card number from frame.

Parameters

frame	... Wiegand26 data frame.

Returns

card number

4.20.2.3 weigand_get_facility()

```
uint32_t weigand_get_facility (
            weigand26_frame_t frame )
```

Parse facility code from frame.

Parameters

frame	... Wiegand26 data frame.

Returns

facility code

4.20.2.4 weigand_init()

```
void weigand_init (
            StreamBufferHandle_t buffer,
            uint8_t id,
            uint8_t dx_port,
            uint8_t d0_pin,
            uint8_t d1_pin )
```

Initialize Wiegand driver.

Note

One buffer for each consumer is preferred.

Parameters

buffer	... Receive buffer for frames from RFID card/tags. See weigand26_buff_item_t
id	... interface identification
dx_port	... Port number for data signals.
d0_pin	... Pin for 0's data signal.
d1_pin	... Pin for 1's data signal.

4.20.2.5 weigand_is_parity_ok()

```
bool weigand_is_parity_ok (
            weigand26_frame_t frame )
```

Check frame parity.

Parameters

frame	... Wiegand26 data frame.

Returns

true if parity is valid

Index