1-Wire Communication Library for STM32Fxxx

V1.0.5

Generated by Doxygen 1.8.1.2

Tue Feb 19 2013 13:08:50

Contents

Chapter 1

File Documentation

1.1 example/Example_OneWire.c File Reference

Simple code example for 1-Wire library.

```
#include "OneWire.h"
```

Macros

- #define **MAX_DEVICES** 8
- #define **SOME_COMMAND** 0xAA

Functions

- int **main** (void)

1.1.1 Detailed Description

Simple code example for 1-Wire library.

Author

Vojtěch Vigner `vojtech.vigner@gmail.com`

Date

18-February-2013

Definition in file Example_OneWire.c.

1.2 Example_OneWire.c

```

#include "OneWire.h"

#define MAX_DEVICES      8
#define SOME_COMMAND     0xAA

int main(void) {

    uint64_t Addresses[MAX_DEVICES];
    uint64_t iAddress;
    int iCount = 0;
    int i;

    /* Bus initialization */
    OW_Init();

    /* Ready bus for communcation */
    OW_WeakPullUp();

    /* Search for first 1-Wire device */
    iAddress = OW_SearchFirst(0);

    /* Store all device addresses into a array */
    while ((iAddress) && (iCount < MAX_DEVICES)) {
        iCount++;
        Addresses[iCount - 1] = iAddress;
        iAddress = OW_SearchNext();
    }

    if (iCount == 0) {
        printf("No devices found.\r\n");
        return 1;
    }

    /* Reset communication because the last device remained selected */
    OW_Reset();

    for (i = 0; i < iCount; i++) {
        OW_Reset();
        OW_ByteWrite(SOME_COMMAND);
        printf("Device %d response = %d.\r\n", i, OW_ByteRead());
    }

    printf("Finished.\r\n");

    return 0;
}
```

1.3 inc/OneWire.h File Reference

Provides 1-Wire bus support for STM32Fxxx devices.

```
#include "stdint.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_usart.h"
#include "stm32f4xx_rcc.h"
```

Macros

- #define **OW_TX_PIN_PORT** GPIOD
- #define **OW_TX_PIN_PIN** GPIO_Pin_5
- #define **OW_USART** USART2
- #define **OW_USART_AF** GPIO_AF_USART2
- #define **OW_GPIO_CLOCK**() RCC_APB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE)
- #define **OW_USART_CLOCK**() RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE)
- #define **OW_PARASITE_POWERED** 1
- #define **OW_ADDRESS_ALL** 0

Typedefs

- typedef enum _OW_State **OW_State**

Enumerations

- enum **_OW_State** { **OW_OK** = 0, **OW_PRESENT**, **OW_NO_DEV**, **OW_CRC_ERROR** }

Functions

- void OW_Init (void)
- OW_State OW_Reset (void)
- uint8_t OW_BitRead (void)
- uint8_t OW_ByteRead (void)
- void OW_BitWrite (const uint8_t bBit)
- void OW_ByteWrite (const uint8_t bByte)
- uint8_t OW_CRCCalculate (uint8_t iCRC, uint8_t iValue)
- void OW_FamilySkipSetup (void)
- uint64_t OW_SearchFirst (uint8_t iFamilyCode)
- uint64_t OW_SearchNext (void)
- void OW_StrongPullUp (void)
- void OW_WeakPullUp (void)
- uint64_t OW_ROMRead (void)
- OW_State OW_ROMMatch (uint64_t iAddress)
- OW_State OW_ROMSkip (void)

1.3.1 Detailed Description

Provides 1-Wire bus support for STM32Fxxx devices.

Author

Vojtěch Vigner vojtech.vigner@gmail.com

Version

V1.0.5

Date

12-February-2013

See Also

OneWire.c documentation

Copyright

The BSD 3-Clause License.

1.3.2 License

Definition in file OneWire.h.

1.3.3 Function Documentation

1.3.3.1 uint8_t OW_BitRead (void)

Read one bit.

Returns

0 or 1.

Definition at line 170 of file OneWire.c.

```
                                {
    /* Make sure that all communication is done and receive buffer is cleared
      */
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);

    /* Send byte */
    USART_SendData(OW_USART, OW_1);

    /* Wait for response */
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);

    /* Receive data */
    if (USART_ReceiveData(OW_USART) != OW_1) return 0;

    return 1;
}
```

1.3.3.2 void OW_BitWrite (const uint8_t *bBit*)

Write one bit.

Parameters

bBit	0 or 1.

Definition at line 208 of file OneWire.c.

```
                                            {
    uint8_t bData = OW_0;

    if (bBit) bData = OW_1;

    /* Make sure that all communication is done */
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);
    /* Send byte */
    USART_SendData(OW_USART, bData);
}
```

1.3.3.3 uint8_t OW_ByteRead (void)

Read one byte.

Returns

Received byte.

Definition at line 192 of file OneWire.c.

```
                        {
    int i;
    uint8_t iRet = 0;

    /* Read 8 bits */
    for (i = 0; i < 8; i++) {
        if (OW_BitRead()) iRet |= (1 << i);
    }

    return iRet;
}
```

1.3.3.4 void OW_ByteWrite (const uint8_t *bByte*)

Write one byte.

Parameters

bByte	Byte to be transmited.

Definition at line 226 of file OneWire.c.

```
                                     {
    uint8_t i;

    /* Write 8 bits */
    for (i = 0; i < 8; i++) {
        OW_BitWrite(bByte & (1 << i));
    }
}
```

1.3.3.5 uint8_t OW_CRCCalculate (uint8_t *iCRC*, uint8_t *iValue*)

Calculate the 1-Wire specific CRC.

Parameters

iCRC	Input CRC value.
iValue	Value to be added to CRC.

Returns

Resulting CRC value.

Definition at line 308 of file OneWire.c.

```
                                                    {
```

```
    return CRCTable[iCRC ^ iValue];
}
```

1.3.3.6 void OW_FamilySkipSetup (void)

Setup the search to skip the current device type on the next call of OW_SearchNext function.

Definition at line 293 of file OneWire.c.

```
                              {
    /* Set the last discrepancy to last family discrepancy */
    stSearch.iLastDiscrepancy = stSearch.iLastFamilyDiscrepancy;
    stSearch.iLastFamilyDiscrepancy = 0;

    /* Check for end of list */
    if (stSearch.iLastDiscrepancy == 0) stSearch.iLastDeviceFlag = 1;
}
```

1.3.3.7 void OW_Init (void)

Hardware initialization.

Definition at line 122 of file OneWire.c.

```
                   {
    GPIO_InitTypeDef GPIO_InitStruct;
    USART_InitTypeDef USART_InitStructure;

    /* Enable clock for periphetials */
    OW_GPIO_CLOCK();
    OW_USART_CLOCK();

    /* Alternate function config on TX pin */
    GPIO_PinAFConfig(OW_TX_PIN_PORT, OW_TX_PIN_PIN, OW_USART_AF);

    /* TX pin configuration */
    GPIO_InitStruct.GPIO_Pin = OW_TX_PIN_PIN;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(OW_TX_PIN_PORT, &GPIO_InitStruct);

    /* USART configuration */
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
      USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
    USART_Init(OW_USART, &USART_InitStructure);

    /* BRR register backup for 115200 Baud */
    iUSART115200 = OW_USART->BRR;

    /* BRR register backup for 9600 Baud */
    USART_StructInit(&USART_InitStructure);
    USART_InitStructure.USART_BaudRate = 9600;
    USART_Init(OW_USART, &USART_InitStructure);
    iUSART9600 = OW_USART->BRR;

    /* Half duplex enable, for single pin communication */
    USART_HalfDuplexCmd(OW_USART, ENABLE);

    /* USART enable */
    USART_Cmd(OW_USART, ENABLE);
}
```

1.3.3.8 OW_State OW_Reset (void)

Communication reset and device presence detection.

Returns

OW_PRESENT if device found or OW_NO_DEV if not.

Definition at line 261 of file OneWire.c.

```
                        {
    uint8_t iPresence;

    /* Set USART baudrate to 9600 Baud */
    OW_USART->BRR = iUSART9600;

    /* Make sure that all communication is done and receive buffer is cleared
      */
    USART_ClearFlag(OW_USART, USART_FLAG_TC);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);

    /* Write special byte on USART */
    USART_SendData(OW_USART, OW_R);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);

    /* Receive data from USART */
    iPresence = USART_ReceiveData(OW_USART);

    /* Set USART baudrate to 115200 Baud */
    OW_USART->BRR = iUSART115200;

    /* If received data is equal to data transmitted means that there in no

     device present on the bus. Return value equal to 0 means bus error. */
    if ((iPresence != OW_R) && ((iPresence != 0x00))) return OW_PRESENT;

    return OW_NO_DEV;
}
```

1.3.3.9 OW_State OW_ROMMatch (uint64_t *iAddress*)

Issue ROM match command.

Parameters

iAddress	64-bit device address.

Returns

OW_OK if device is present or OW_NO_DEV if not.

Definition at line 466 of file OneWire.c.

```
                                        {
    int i;

    if (iAddress == OW_ADDRESS_ALL) return OW_ROMSkip();

    if (OW_Reset() == OW_NO_DEV) return OW_NO_DEV;

    OW_ByteWrite(OW_ROM_MATCH);
    for (i = 0; i < 8; i++)
        OW_ByteWrite(((uint8_t*) & iAddress)[i]);

    return OW_OK;
}
```

1.3.3.10 uint64_t OW_ROMRead (void)

Read ROM address of device, works only for one device on the bus.

Returns

64-bit device address.

Definition at line 448 of file OneWire.c.

```
                              {
    uint64_t iRes = 0;
    int i;

    if (OW_Reset() == OW_NO_DEV) return 0;

    OW_ByteWrite(OW_ROM_READ);
    for (i = 0; i < 8; i++)
        ((uint8_t*) & iRes)[i] = OW_ByteRead();

    return iRes;
}
```

1.3.3.11 OW_State OW_ROMSkip (void)

Issue ROM skip command.

Returns

OW_OK if some device is present or OW_NO_DEV if not.

Definition at line 484 of file OneWire.c.

```
                              {
    if (OW_Reset() == OW_NO_DEV) return OW_NO_DEV;

    OW_ByteWrite(OW_ROM_SKIP);

    return OW_OK;
}
```

1.3.3.12 uint64_t OW_SearchFirst (uint8_t *iFamilyCode*)

Find the 'first' devices on the 1-Wire bus.

Parameters

iFamilyCode	Select family code filter or 0 for all.

Returns

64-bit device address or 0 if no device found.

Definition at line 317 of file OneWire.c.

```
                                                  {
    if (iFamilyCode) {
        stSearch.ROM = (uint64_t) iFamilyCode;

        stSearch.iLastDiscrepancy = 64;
        stSearch.iLastFamilyDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 1;
    } else {
        stSearch.ROM = 0;
        stSearch.iLastDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 0;
        stSearch.iLastFamilyDiscrepancy = 0;
    }

    return OW_SearchNext();
}
```

1.3.3.13 uint64_t OW_SearchNext (void)

Perform the 1-Wire Search Algorithm on the 1-Wire bus using the existing search state.

Returns

64-bit device address or 0 if no device found.

Definition at line 339 of file OneWire.c.

```
                             {
    uint8_t iSearchDirection;
    int iIDBit, iCmpIDBit;

    /* Initialize for search */
    uint8_t iROMByteMask = 1;
    uint8_t iCRC = 0;
    int iIDBitNumber = 1;
    int iLastZero = 0;
    int iROMByteNumber = 0;
    int iSearchResult = 0;

    /* If the last call was not the last one */
    if (!stSearch.iLastDeviceFlag) {
        /* 1-Wire reset */
        if (!OW_Reset()) {
            /* Reset the search */
            stSearch.iLastDiscrepancy = 0;
            stSearch.iLastDeviceFlag = 0;
            stSearch.iLastFamilyDiscrepancy = 0;
            return 0;
        }

        /* Issue the search command */
        OW_ByteWrite(OW_ROM_SEARCH);

        /* Loop to do the search */
        do {
            /* Read a bit and its complement */
            iIDBit = OW_BitRead();
            iCmpIDBit = OW_BitRead();

            /* Check for no devices on 1-wire */
            if ((iIDBit == 1) && (iCmpIDBit == 1))
                break;
            else {
                /* All devices coupled have 0 or 1 */
                if (iIDBit != iCmpIDBit)
                    /* Bit write value for search */
                    iSearchDirection = iIDBit;
                else {
                    /* if this discrepancy if before the Last Discrepancy

                    on a previous next then pick the same as last time */
                    if (iIDBitNumber < stSearch.iLastDiscrepancy)
                        iSearchDirection = ((((uint8_t*) & stSearch.ROM)[
      iROMByteNumber] & iROMByteMask) > 0);
                    else
                        /* If equal to last pick 1, if not then pick 0 */
                        iSearchDirection = (iIDBitNumber == stSearch.
      iLastDiscrepancy);

                    /* If 0 was picked then record its position in iLastZero */
                    if (iSearchDirection == 0) {
                        iLastZero = iIDBitNumber;

                        /* Check for Last discrepancy in family */
                        if (iLastZero < 9)
                            stSearch.iLastFamilyDiscrepancy = iLastZero;
                    }
                }

                /* Set or clear the bit in the ROM byte with mask rom_byte_mask
       */
                if (iSearchDirection == 1)
                    ((uint8_t*) & stSearch.ROM)[iROMByteNumber] |= iROMByteMask
      ;
                else
                    ((uint8_t*) & stSearch.ROM)[iROMByteNumber] &= ~
      iROMByteMask;

                /* Set serial number search direction */
                OW_BitWrite(iSearchDirection);
                /* Increment the byte counter and shift the mask */
                iIDBitNumber++;
                iROMByteMask <<= 1;

                /* If the mask is 0 then go to new ROM byte number and reset
       mask */
                if (iROMByteMask == 0) {
                    /* Accumulate the CRC */
                    iCRC = OW_CRCCalculate(iCRC, ((uint8_t*) &
      stSearch.ROM)[iROMByteNumber]);
                    iROMByteNumber++;
                    iROMByteMask = 1;
                }
            }
        } while (iROMByteNumber < 8); /* Loop until through all ROM bytes 0-7
      */

        /* If the search was successful then */
        if (!((iIDBitNumber < 65) || (iCRC != 0))) {
            stSearch.iLastDiscrepancy = iLastZero;

            /* Check for last device */
            if (stSearch.iLastDiscrepancy == 0)
                stSearch.iLastDeviceFlag = 1;

            iSearchResult = 1;
        }
    }

    /* If no device found then reset counters so next 'search' will be like a
       first */
    if (!iSearchResult || !((uint8_t*) & stSearch.ROM)[0]) {
        stSearch.iLastDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 0;
        stSearch.iLastFamilyDiscrepancy = 0;
        return 0;
    }

    return stSearch.ROM;
}
```

1.3.3.14 void OW_StrongPullUp (void)

Set RX/TX pin into strong pull-up state.

Definition at line 238 of file OneWire.c.

```
                           {
#if OW_PARASITE_POWERED

    USART_HalfDuplexCmd(OW_USART, DISABLE);
    GPIO_SetBits(OW_TX_PIN_PORT, OW_TX_PIN_PIN);
    OW_TX_PIN_PORT->OTYPER &= ~(OW_TX_PIN_PIN);
#endif

}
```

1.3.3.15 void OW_WeakPullUp (void)

Set RX/TX pin into weak pull-up state.

Definition at line 249 of file OneWire.c.

```
                         {
#if OW_PARASITE_POWERED

    GPIO_SetBits(OW_TX_PIN_PORT, OW_TX_PIN_PIN);
    OW_TX_PIN_PORT->OTYPER |= OW_TX_PIN_PIN;
    USART_HalfDuplexCmd(OW_USART, ENABLE);
#endif

}
```

1.4 OneWire.h

```

#ifndef ONEWIRE_H
#define ONEWIRE_H

#ifdef  __cplusplus
extern "C" {
#endif

#include "stdint.h"

    /*************************************************************************
*/
    /* Hardware specific configuration */
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_usart.h"
#include "stm32f4xx_rcc.h"

#define OW_TX_PIN_PORT          GPIOD
#define OW_TX_PIN_PIN           GPIO_Pin_5

#define OW_USART                USART2
#define OW_USART_AF             GPIO_AF_USART2

#define OW_GPIO_CLOCK()         RCC_APB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE)
#define OW_USART_CLOCK()        RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE)
    /*************************************************************************
*/

    /* Enables parasite powered device support */
#define OW_PARASITE_POWERED     1

    /* Public defines */
#define OW_ADDRESS_ALL          0

    typedef enum _OW_State {
        OW_OK = 0,
        OW_PRESENT,
        OW_NO_DEV,
        OW_CRC_ERROR
    } OW_State;


    /* Hardware initialization */
    void OW_Init(void);

    /* Communication functions */
    OW_State OW_Reset(void);
    uint8_t OW_BitRead(void);
    uint8_t OW_ByteRead(void);
    void OW_BitWrite(const uint8_t bBit);
    void OW_ByteWrite(const uint8_t bByte);

    /* Utilities */
    uint8_t OW_CRCCalculate(uint8_t iCRC, uint8_t iValue);

    /* 1-Wire search */
    void OW_FamilySkipSetup(void);
    uint64_t OW_SearchFirst(uint8_t iFamilyCode);
    uint64_t OW_SearchNext(void);

    /* Parasite powered devices support */
    void OW_StrongPullUp(void);
    void OW_WeakPullUp(void);

    /* ROM operations */
    uint64_t OW_ROMRead(void);
    OW_State OW_ROMMatch(uint64_t iAddress);
    OW_State OW_ROMSkip(void);

#endif //ONEWIRE_H
```

1.5 src/OneWire.c File Reference

Provides 1-Wire bus support for STM32Fxxx devices.

`#include "OneWire.h"`

Macros

- #define **OW_ROM_READ** 0x33
- #define **OW_ROM_MATCH** 0x55
- #define **OW_ROM_SKIP** 0xCC
- #define **OW_ROM_SEARCH** 0xF0
- #define **OW_ALARM_SEARCH** 0xEC
- #define **OW_R** 0xF0
- #define **OW_0** 0x00
- #define **OW_1** 0xFF

Functions

- void OW_Init (void)
- uint8_t OW_BitRead (void)
- uint8_t OW_ByteRead (void)
- void OW_BitWrite (const uint8_t bBit)
- void OW_ByteWrite (const uint8_t bByte)
- void OW_StrongPullUp (void)
- void OW_WeakPullUp (void)
- OW_State OW_Reset (void)
- void OW_FamilySkipSetup (void)
- uint8_t OW_CRCCalculate (uint8_t iCRC, uint8_t iValue)
- uint64_t OW_SearchFirst (uint8_t iFamilyCode)
- uint64_t OW_SearchNext (void)
- uint64_t OW_ROMRead (void)
- OW_State OW_ROMMatch (uint64_t iAddress)
- OW_State OW_ROMSkip (void)

Variables

- struct {
    uint8_t **iLastDeviceFlag**
    uint8_t **iLastDiscrepancy**
    uint8_t **iLastFamilyDiscrepancy**
    uint64_t **ROM**
  } **stSearch**
- uint16_t **iUSART9600**
- uint16_t **iUSART115200**

1.5.1 Detailed Description

Provides 1-Wire bus support for STM32Fxxx devices.

Author

Vojtěch Vigner `vojtech.vigner@gmail.com`

Version

V1.0.5

Date

12-February-2013

1.5.2 Additional Information

This library provides functions to manage the following functionalities of the 1-Wire bus from MAXIM:

- Initialization and configuration.
- Low level communication functions.
- 1-Wire specific CRC calculation.
- Device address operations.
- Parasite powered device support.
- 1-Wire advanced device search, based on MAXIM App. Note 126.

Library requires one USART for communication with 1-Wire devices. Current implementation used Half Duplex USART mode. This means that only one pin is used for communication.

Currently supports and has been tested on STM32F2xx and STM32F4xx devices.

1.5.3 How to use this library

1. Modify hardware specific section in OneWire.h file according to your HW. Decide if you will be using parasite powered device/s and enable or disable this support.

1. Initialize bus using OW_Init().

1. Now you can use all communication functions.

1. See Example_OneWire.c for simple example.

Copyright

The BSD 3-Clause License.

1.5.4 License

Definition in file OneWire.c.

1.5.5 Function Documentation

1.5.5.1 uint8_t OW_BitRead (void)

Read one bit.

Returns

0 or 1.

Definition at line 170 of file OneWire.c.

```
                         {
    /* Make sure that all communication is done and receive buffer is cleared
      */
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);

    /* Send byte */
    USART_SendData(OW_USART, OW_1);

    /* Wait for response */
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);

    /* Receive data */
    if (USART_ReceiveData(OW_USART) != OW_1) return 0;

    return 1;
}
```

1.5.5.2 void OW_BitWrite (const uint8_t *bBit*)

Write one bit.

Parameters

bBit	0 or 1.

Definition at line 208 of file OneWire.c.

```
                                     {
    uint8_t bData = OW_0;

    if (bBit) bData = OW_1;

    /* Make sure that all communication is done */
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);

    /* Send byte */
    USART_SendData(OW_USART, bData);
}
```

1.5.5.3 uint8_t OW_ByteRead (void)

Read one byte.

Returns

Received byte.

Definition at line 192 of file OneWire.c.

```
                              {
    int i;
    uint8_t iRet = 0;

    /* Read 8 bits */
    for (i = 0; i < 8; i++) {
        if (OW_BitRead()) iRet |= (1 << i);
    }

    return iRet;
}
```

1.5.5.4 void OW_ByteWrite (const uint8_t *bByte*)

Write one byte.

Parameters

bByte	Byte to be transmited.

Definition at line 226 of file OneWire.c.

```
                                           {
    uint8_t i;

    /* Write 8 bits */
    for (i = 0; i < 8; i++) {
        OW_BitWrite(bByte & (1 << i));
    }
}
```

1.5.5.5 uint8_t OW_CRCCalculate (uint8_t *iCRC,* uint8_t *iValue*)

Calculate the 1-Wire specific CRC.

Parameters

iCRC	Input CRC value.
iValue	Value to be added to CRC.

Returns

Resulting CRC value.

Definition at line 308 of file OneWire.c.

```
                                                          {
    return CRCTable[iCRC ^ iValue];
}
```

1.5.5.6 void OW_FamilySkipSetup (void)

Setup the search to skip the current device type on the next call of OW_SearchNext function.

Definition at line 293 of file OneWire.c.

```
                              {
    /* Set the last discrepancy to last family discrepancy */
    stSearch.iLastDiscrepancy = stSearch.iLastFamilyDiscrepancy;
    stSearch.iLastFamilyDiscrepancy = 0;

    /* Check for end of list */
    if (stSearch.iLastDiscrepancy == 0) stSearch.iLastDeviceFlag = 1;
}
```

1.5.5.7 void OW_Init (void)

Hardware initialization.

Definition at line 122 of file OneWire.c.

```
                   {
    GPIO_InitTypeDef GPIO_InitStruct;
    USART_InitTypeDef USART_InitStructure;

    /* Enable clock for periphetials */
    OW_GPIO_CLOCK();
    OW_USART_CLOCK();

    /* Alternate function config on TX pin */
    GPIO_PinAFConfig(OW_TX_PIN_PORT, OW_TX_PIN_PIN, OW_USART_AF);

    /* TX pin configuration */
    GPIO_InitStruct.GPIO_Pin = OW_TX_PIN_PIN;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(OW_TX_PIN_PORT, &GPIO_InitStruct);

    /* USART configuration */
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
      USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
    USART_Init(OW_USART, &USART_InitStructure);

    /* BRR register backup for 115200 Baud */
    iUSART115200 = OW_USART->BRR;

    /* BRR register backup for 9600 Baud */
    USART_StructInit(&USART_InitStructure);
    USART_InitStructure.USART_BaudRate = 9600;
    USART_Init(OW_USART, &USART_InitStructure);
    iUSART9600 = OW_USART->BRR;

    /* Half duplex enable, for single pin communication */
    USART_HalfDuplexCmd(OW_USART, ENABLE);

    /* USART enable */
    USART_Cmd(OW_USART, ENABLE);
}
```

1.5.5.8 OW_State OW_Reset (void)

Communication reset and device presence detection.

Returns

OW_PRESENT if device found or OW_NO_DEV if not.

Definition at line 261 of file OneWire.c.

```
                        {
    uint8_t iPresence;

    /* Set USART baudrate to 9600 Baud */
    OW_USART->BRR = iUSART9600;

    /* Make sure that all communication is done and receive buffer is cleared
      */
    USART_ClearFlag(OW_USART, USART_FLAG_TC);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);

    /* Write special byte on USART */
    USART_SendData(OW_USART, OW_R);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);

    /* Receive data from USART */
    iPresence = USART_ReceiveData(OW_USART);

    /* Set USART baudrate to 115200 Baud */
    OW_USART->BRR = iUSART115200;

    /* If received data is equal to data transmitted means that there in no

     device present on the bus. Return value equal to 0 means bus error. */
    if ((iPresence != OW_R) && ((iPresence != 0x00))) return OW_PRESENT;

    return OW_NO_DEV;
}
```

1.5.5.9 OW_State OW_ROMMatch (uint64_t *iAddress*)

Issue ROM match command.

Parameters

iAddress	64-bit device address.

Returns

OW_OK if device is present or OW_NO_DEV if not.

Definition at line 466 of file OneWire.c.

```
                                    {
    int i;

    if (iAddress == OW_ADDRESS_ALL) return OW_ROMSkip();

    if (OW_Reset() == OW_NO_DEV) return OW_NO_DEV;

    OW_ByteWrite(OW_ROM_MATCH);
    for (i = 0; i < 8; i++)
        OW_ByteWrite(((uint8_t*) & iAddress)[i]);

    return OW_OK;
}
```

1.5.5.10 uint64_t OW_ROMRead (void)

Read ROM address of device, works only for one device on the bus.

Returns

64-bit device address.

Definition at line 448 of file OneWire.c.

```
                      {
    uint64_t iRes = 0;
    int i;

    if (OW_Reset() == OW_NO_DEV) return 0;
    OW_ByteWrite(OW_ROM_READ);
    for (i = 0; i < 8; i++)
        ((uint8_t*) & iRes)[i] = OW_ByteRead();

    return iRes;
}
```

1.5.5.11 OW_State OW_ROMSkip (void)

Issue ROM skip command.

Returns

OW_OK if some device is present or OW_NO_DEV if not.

Definition at line 484 of file OneWire.c.

```
                              {
    if (OW_Reset() == OW_NO_DEV) return OW_NO_DEV;

    OW_ByteWrite(OW_ROM_SKIP);

    return OW_OK;
}
```

1.5.5.12 uint64_t OW_SearchFirst (uint8_t *iFamilyCode*)

Find the ’first’ devices on the 1-Wire bus.

Parameters

iFamilyCode	Select family code filter or 0 for all.

Returns

64-bit device address or 0 if no device found.

Definition at line 317 of file OneWire.c.

```
                                                {
    if (iFamilyCode) {
        stSearch.ROM = (uint64_t) iFamilyCode;

        stSearch.iLastDiscrepancy = 64;
        stSearch.iLastFamilyDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 1;
    } else {
        stSearch.ROM = 0;
        stSearch.iLastDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 0;
        stSearch.iLastFamilyDiscrepancy = 0;
    }

    return OW_SearchNext();
}
```

1.5.5.13 uint64_t OW_SearchNext (void)

Perform the 1-Wire Search Algorithm on the 1-Wire bus using the existing search state.

Returns

64-bit device address or 0 if no device found.

Definition at line 339 of file OneWire.c.

```
                                 {
    uint8_t iSearchDirection;
    int iIDBit, iCmpIDBit;

    /* Initialize for search */
    uint8_t iROMByteMask = 1;
    uint8_t iCRC = 0;
    int iIDBitNumber = 1;
    int iLastZero = 0;
    int iROMByteNumber = 0;
    int iSearchResult = 0;

    /* If the last call was not the last one */
    if (!stSearch.iLastDeviceFlag) {
        /* 1-Wire reset */
        if (!OW_Reset()) {
            /* Reset the search */
            stSearch.iLastDiscrepancy = 0;
            stSearch.iLastDeviceFlag = 0;
            stSearch.iLastFamilyDiscrepancy = 0;
            return 0;
        }

        /* Issue the search command */
        OW_ByteWrite(OW_ROM_SEARCH);

        /* Loop to do the search */
        do {
            /* Read a bit and its complement */
            iIDBit = OW_BitRead();
            iCmpIDBit = OW_BitRead();

            /* Check for no devices on 1-wire */
            if ((iIDBit == 1) && (iCmpIDBit == 1))
                break;
            else {
                /* All devices coupled have 0 or 1 */
                if (iIDBit != iCmpIDBit)
                    /* Bit write value for search */
                    iSearchDirection = iIDBit;
                else {
                    /* if this discrepancy if before the Last Discrepancy

                    on a previous next then pick the same as last time */
                    if (iIDBitNumber < stSearch.iLastDiscrepancy)
                        iSearchDirection = ((((uint8_t*) & stSearch.ROM)[
      iROMByteNumber] & iROMByteMask) > 0);
                    else
                        /* If equal to last pick 1, if not then pick 0 */
                        iSearchDirection = (iIDBitNumber == stSearch.
      iLastDiscrepancy);

                    /* If 0 was picked then record its position in iLastZero */
                    if (iSearchDirection == 0) {
                        iLastZero = iIDBitNumber;

                        /* Check for Last discrepancy in family */
                        if (iLastZero < 9)
                            stSearch.iLastFamilyDiscrepancy = iLastZero;
                    }
                }

                /* Set or clear the bit in the ROM byte with mask rom_byte_mask
       */
                if (iSearchDirection == 1)
                    ((uint8_t*) & stSearch.ROM)[iROMByteNumber] |= iROMByteMask
      ;
                else
                    ((uint8_t*) & stSearch.ROM)[iROMByteNumber] &= ~
      iROMByteMask;

                /* Set serial number search direction */
                OW_BitWrite(iSearchDirection);

                /* Increment the byte counter and shift the mask */
                iIDBitNumber++;
                iROMByteMask <<= 1;

                /* If the mask is 0 then go to new ROM byte number and reset
       mask */
                if (iROMByteMask == 0) {
                    /* Accumulate the CRC */
                    iCRC = OW_CRCCalculate(iCRC, ((uint8_t*) &
      stSearch.ROM)[iROMByteNumber]);
                    iROMByteNumber++;
                    iROMByteMask = 1;
                }
            }
        } while (iROMByteNumber < 8); /* Loop until through all ROM bytes 0-7
      */

        /* If the search was successful then */
        if (!((iIDBitNumber < 65) || (iCRC != 0))) {
            stSearch.iLastDiscrepancy = iLastZero;

            /* Check for last device */
            if (stSearch.iLastDiscrepancy == 0)
                stSearch.iLastDeviceFlag = 1;

            iSearchResult = 1;
        }
    }

    /* If no device found then reset counters so next 'search' will be like a
       first */
    if (!iSearchResult || !((uint8_t*) & stSearch.ROM)[0]) {
        stSearch.iLastDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 0;
        stSearch.iLastFamilyDiscrepancy = 0;
        return 0;
    }

    return stSearch.ROM;
}
```

1.5.5.14 void OW_StrongPullUp (void)

Set RX/TX pin into strong pull-up state.

Definition at line 238 of file OneWire.c.

```
                            {
#if OW_PARASITE_POWERED

    USART_HalfDuplexCmd(OW_USART, DISABLE);
    GPIO_SetBits(OW_TX_PIN_PORT, OW_TX_PIN_PIN);
    OW_TX_PIN_PORT->OTYPER &= ~(OW_TX_PIN_PIN);
#endif

}
```

1.5.5.15 void OW_WeakPullUp (void)

Set RX/TX pin into weak pull-up state.

Definition at line 249 of file OneWire.c.

```
                          {
#if OW_PARASITE_POWERED

    GPIO_SetBits(OW_TX_PIN_PORT, OW_TX_PIN_PIN);
    OW_TX_PIN_PORT->OTYPER |= OW_TX_PIN_PIN;
    USART_HalfDuplexCmd(OW_USART, ENABLE);
#endif

}
```

1.6 OneWire.c

```
00001
00073 #include "OneWire.h"
```

```c

/* Bus specific commands */
#define OW_ROM_READ         0x33
#define OW_ROM_MATCH        0x55
#define OW_ROM_SKIP         0xCC
#define OW_ROM_SEARCH       0xF0
#define OW_ALARM_SEARCH     0xEC

/* Private defines */
#define OW_R                0xF0
#define OW_0                0x00
#define OW_1                0xFF

/* Search related variables */
struct {
    uint8_t iLastDeviceFlag;
    uint8_t iLastDiscrepancy;
    uint8_t iLastFamilyDiscrepancy;
    uint64_t ROM;
} stSearch;

/* Backup of BRR register for different communication speeds*/
uint16_t iUSART9600;
uint16_t iUSART115200;

/* CRC calculation table */
static uint8_t CRCTable[] = {
    0, 94, 188, 226, 97, 63, 221, 131, 194, 156, 126, 32, 163, 253, 31, 65,
    157, 195, 33, 127, 252, 162, 64, 30, 95, 1, 227, 189, 62, 96, 130, 220,
    35, 125, 159, 193, 66, 28, 254, 160, 225, 191, 93, 3, 128, 222, 60, 98,
    190, 224, 2, 92, 223, 129, 99, 61, 124, 34, 192, 158, 29, 67, 161, 255,
    70, 24, 250, 164, 39, 121, 155, 197, 132, 218, 56, 102, 229, 187, 89, 7,
    219, 133, 103, 57, 186, 228, 6, 88, 25, 71, 165, 251, 120, 38, 196, 154,
    101, 59, 217, 135, 4, 90, 184, 230, 167, 249, 27, 69, 198, 152, 122, 36,
    248, 166, 68, 26, 153, 199, 37, 123, 58, 100, 134, 216, 91, 5, 231, 185,
    140, 210, 48, 110, 237, 179, 81, 15, 78, 16, 242, 172, 47, 113, 147, 205,
    17, 79, 173, 243, 112, 46, 204, 146, 211, 141, 111, 49, 178, 236, 14, 80,
    175, 241, 19, 77, 206, 144, 114, 44, 109, 51, 209, 143, 12, 82, 176, 238,
    50, 108, 142, 208, 83, 13, 239, 177, 240, 174, 76, 18, 145, 207, 45, 115,
    202, 148, 118, 40, 171, 245, 23, 73, 8, 86, 180, 234, 105, 55, 213, 139,
    87, 9, 235, 181, 54, 104, 138, 212, 149, 203, 41, 119, 244, 170, 72, 22,
    233, 183, 85, 11, 136, 214, 52, 106, 43, 117, 151, 201, 74, 20, 246, 168,
    116, 42, 200, 150, 21, 75, 169, 247, 182, 232, 10, 84, 215, 137, 107, 53
};

void OW_Init(void) {
    GPIO_InitTypeDef GPIO_InitStruct;
    USART_InitTypeDef USART_InitStructure;

    /* Enable clock for periphetials */
    OW_GPIO_CLOCK();
    OW_USART_CLOCK();

    /* Alternate function config on TX pin */
    GPIO_PinAFConfig(OW_TX_PIN_PORT, OW_TX_PIN_PIN, OW_USART_AF);

    /* TX pin configuration */
    GPIO_InitStruct.GPIO_Pin = OW_TX_PIN_PIN;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(OW_TX_PIN_PORT, &GPIO_InitStruct);

    /* USART configuration */
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
    USART_Init(OW_USART, &USART_InitStructure);

    /* BRR register backup for 115200 Baud */
    iUSART115200 = OW_USART->BRR;

    /* BRR register backup for 9600 Baud */
    USART_StructInit(&USART_InitStructure);
    USART_InitStructure.USART_BaudRate = 9600;
    USART_Init(OW_USART, &USART_InitStructure);
    iUSART9600 = OW_USART->BRR;

    /* Half duplex enable, for single pin communication */
    USART_HalfDuplexCmd(OW_USART, ENABLE);

    /* USART enable */
    USART_Cmd(OW_USART, ENABLE);
}

uint8_t OW_BitRead(void) {
    /* Make sure that all communication is done and receive buffer is cleared
*/
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);

    /* Send byte */
    USART_SendData(OW_USART, OW_1);

    /* Wait for response */
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);

    /* Receive data */
    if (USART_ReceiveData(OW_USART) != OW_1) return 0;

    return 1;
}

uint8_t OW_ByteRead(void) {
    int i;
    uint8_t iRet = 0;

    /* Read 8 bits */
    for (i = 0; i < 8; i++) {
        if (OW_BitRead()) iRet |= (1 << i);
    }

    return iRet;
}

void OW_BitWrite(const uint8_t bBit) {
    uint8_t bData = OW_0;

    if (bBit) bData = OW_1;

    /* Make sure that all communication is done */
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);

    /* Send byte */
    USART_SendData(OW_USART, bData);
}

void OW_ByteWrite(const uint8_t bByte) {
    uint8_t i;

    /* Write 8 bits */
    for (i = 0; i < 8; i++) {
        OW_BitWrite(bByte & (1 << i));
    }
}

void OW_StrongPullUp(void) {
#if OW_PARASITE_POWERED
    USART_HalfDuplexCmd(OW_USART, DISABLE);
    GPIO_SetBits(OW_TX_PIN_PORT, OW_TX_PIN_PIN);
    OW_TX_PIN_PORT->OTYPER &= ~(OW_TX_PIN_PIN);
#endif
}

void OW_WeakPullUp(void) {
#if OW_PARASITE_POWERED
    GPIO_SetBits(OW_TX_PIN_PORT, OW_TX_PIN_PIN);
    OW_TX_PIN_PORT->OTYPER |= OW_TX_PIN_PIN;
    USART_HalfDuplexCmd(OW_USART, ENABLE);
#endif
}

OW_State OW_Reset(void) {
    uint8_t iPresence;

    /* Set USART baudrate to 9600 Baud */
    OW_USART->BRR = iUSART9600;

    /* Make sure that all communication is done and receive buffer is cleared
*/
    USART_ClearFlag(OW_USART, USART_FLAG_TC);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
        USART_ReceiveData(OW_USART);

    /* Write special byte on USART */
    USART_SendData(OW_USART, OW_R);
    while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);

    /* Receive data from USART */
    iPresence = USART_ReceiveData(OW_USART);

    /* Set USART baudrate to 115200 Baud */
    OW_USART->BRR = iUSART115200;

    /* If received data is equal to data transmitted means that there in no
     device present on the bus. Return value equal to 0 means bus error. */
    if ((iPresence != OW_R) && ((iPresence != 0x00))) return OW_PRESENT;

    return OW_NO_DEV;
}

void OW_FamilySkipSetup(void) {
    /* Set the last discrepancy to last family discrepancy */
    stSearch.iLastDiscrepancy = stSearch.iLastFamilyDiscrepancy;
    stSearch.iLastFamilyDiscrepancy = 0;

    /* Check for end of list */
    if (stSearch.iLastDiscrepancy == 0) stSearch.iLastDeviceFlag = 1;
}

uint8_t OW_CRCCalculate(uint8_t iCRC, uint8_t iValue) {
    return CRCTable[iCRC ^ iValue];
}

uint64_t OW_SearchFirst(uint8_t iFamilyCode) {
    if (iFamilyCode) {
        stSearch.ROM = (uint64_t) iFamilyCode;

        stSearch.iLastDiscrepancy = 64;
        stSearch.iLastFamilyDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 1;
    } else {
        stSearch.ROM = 0;
        stSearch.iLastDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 0;
        stSearch.iLastFamilyDiscrepancy = 0;
    }

    return OW_SearchNext();
}

uint64_t OW_SearchNext(void) {
    uint8_t iSearchDirection;
    int iIDBit, iCmpIDBit;

    /* Initialize for search */
    uint8_t iROMByteMask = 1;
    uint8_t iCRC = 0;
    int iIDBitNumber = 1;
    int iLastZero = 0;
    int iROMByteNumber = 0;
    int iSearchResult = 0;

    /* If the last call was not the last one */
    if (!stSearch.iLastDeviceFlag) {
        /* 1-Wire reset */
        if (!OW_Reset()) {
            /* Reset the search */
            stSearch.iLastDiscrepancy = 0;
            stSearch.iLastDeviceFlag = 0;
            stSearch.iLastFamilyDiscrepancy = 0;
            return 0;
        }

        /* Issue the search command */
        OW_ByteWrite(OW_ROM_SEARCH);

        /* Loop to do the search */
        do {
            /* Read a bit and its complement */
            iIDBit = OW_BitRead();
            iCmpIDBit = OW_BitRead();

            /* Check for no devices on 1-wire */
            if ((iIDBit == 1) && (iCmpIDBit == 1))
                break;
            else {
                /* All devices coupled have 0 or 1 */
                if (iIDBit != iCmpIDBit)
                    /* Bit write value for search */
                    iSearchDirection = iIDBit;
                else {
                    /* if this discrepancy if before the Last Discrepancy
                    on a previous next then pick the same as last time */
                    if (iIDBitNumber < stSearch.iLastDiscrepancy)
                        iSearchDirection = ((((uint8_t*) & stSearch.ROM)[iROMByteNumber] & iROMByteMask) > 0);
                    else
                        /* If equal to last pick 1, if not then pick 0 */
                        iSearchDirection = (iIDBitNumber == stSearch.iLastDiscrepancy);

                    /* If 0 was picked then record its position in iLastZero */
                    if (iSearchDirection == 0) {
                        iLastZero = iIDBitNumber;

                        /* Check for Last discrepancy in family */
                        if (iLastZero < 9)
                            stSearch.iLastFamilyDiscrepancy = iLastZero;
                    }
                }

                /* Set or clear the bit in the ROM byte with mask rom_byte_mask */
                if (iSearchDirection == 1)
                    ((uint8_t*) & stSearch.ROM)[iROMByteNumber] |= iROMByteMask;
                else
                    ((uint8_t*) & stSearch.ROM)[iROMByteNumber] &= ~iROMByteMask;

                /* Set serial number search direction */
                OW_BitWrite(iSearchDirection);

                /* Increment the byte counter and shift the mask */
                iIDBitNumber++;
                iROMByteMask <<= 1;

                /* If the mask is 0 then go to new ROM byte number and reset mask */
                if (iROMByteMask == 0) {
                    /* Accumulate the CRC */
                    iCRC = OW_CRCCalculate(iCRC, ((uint8_t*) & stSearch.ROM)[iROMByteNumber]);
                    iROMByteNumber++;
                    iROMByteMask = 1;
                }
            }
        } while (iROMByteNumber < 8); /* Loop until through all ROM bytes 0-7 */

        /* If the search was successful then */
        if (!((iIDBitNumber < 65) || (iCRC != 0))) {
            stSearch.iLastDiscrepancy = iLastZero;

            /* Check for last device */
            if (stSearch.iLastDiscrepancy == 0)
                stSearch.iLastDeviceFlag = 1;

            iSearchResult = 1;
        }
    }

    /* If no device found then reset counters so next 'search' will be like a first */
    if (!iSearchResult || !((uint8_t*) & stSearch.ROM)[0]) {
        stSearch.iLastDiscrepancy = 0;
        stSearch.iLastDeviceFlag = 0;
        stSearch.iLastFamilyDiscrepancy = 0;
        return 0;
    }

    return stSearch.ROM;
}

uint64_t OW_ROMRead(void) {
    uint64_t iRes = 0;
    int i;

    if (OW_Reset() == OW_NO_DEV) return 0;

    OW_ByteWrite(OW_ROM_READ);
    for (i = 0; i < 8; i++)
        ((uint8_t*) & iRes)[i] = OW_ByteRead();

    return iRes;
}

OW_State OW_ROMMatch(uint64_t iAddress) {
    int i;

    if (iAddress == OW_ADDRESS_ALL) return OW_ROMSkip();

    if (OW_Reset() == OW_NO_DEV) return OW_NO_DEV;

    OW_ByteWrite(OW_ROM_MATCH);
    for (i = 0; i < 8; i++)
        OW_ByteWrite(((uint8_t*) & iAddress)[i]);

    return OW_OK;
}

OW_State OW_ROMSkip(void) {
    if (OW_Reset() == OW_NO_DEV) return OW_NO_DEV;

    OW_ByteWrite(OW_ROM_SKIP);

    return OW_OK;
}
```

Index