

1-Wire Communication Library for STM32Fxxx
V1.0.5

Generated by Doxygen 1.8.1.2

Tue Feb 19 2013 13:08:50

Contents

1	File Documentation	1
1.1	example/Example_OneWire.c File Reference	1
1.1.1	Detailed Description	1
1.2	Example_OneWire.c	1
1.3	inc/OneWire.h File Reference	2
1.3.1	Detailed Description	3
1.3.2	License	3
1.3.3	Function Documentation	4
1.3.3.1	OW_BitRead	4
1.3.3.2	OW_BitWrite	4
1.3.3.3	OW_ByteRead	5
1.3.3.4	OW_ByteWrite	5
1.3.3.5	OW_CRCCalculate	5
1.3.3.6	OW_FamilySkipSetup	6
1.3.3.7	OW_Init	6
1.3.3.8	OW_Reset	6
1.3.3.9	OW_ROMMatch	7
1.3.3.10	OW_ROMRead	7
1.3.3.11	OW_ROMSkip	8
1.3.3.12	OW_SearchFirst	8
1.3.3.13	OW_SearchNext	9
1.3.3.14	OW_StrongPullUp	10
1.3.3.15	OW_WeakPullUp	10
1.4	OneWire.h	11
1.5	src/OneWire.c File Reference	11
1.5.1	Detailed Description	12
1.5.2	Additional Information	13
1.5.3	How to use this library	13
1.5.4	License	13
1.5.5	Function Documentation	14
1.5.5.1	OW_BitRead	14

1.5.5.2	OW_BitWrite	14
1.5.5.3	OW_ByteRead	14
1.5.5.4	OW_ByteWrite	15
1.5.5.5	OW_CRCCalculate	15
1.5.5.6	OW_FamilySkipSetup	15
1.5.5.7	OW_Init	16
1.5.5.8	OW_Reset	16
1.5.5.9	OW_ROMMatch	17
1.5.5.10	OW_ROMRead	17
1.5.5.11	OW_ROMSkip	18
1.5.5.12	OW_SearchFirst	18
1.5.5.13	OW_SearchNext	18
1.5.5.14	OW_StrongPullUp	20
1.5.5.15	OW_WeakPullUp	20
1.6	OneWire.c	20

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

```
00001
00010 #include "OneWire.h"
00011
00012 #define MAX_DEVICES      8
00013 #define SOME_COMMAND     0xAA
00014
00015 int main(void) {
00016
```

```

00017     uint64_t Addresses[MAX_DEVICES];
00018     uint64_t iAddress;
00019     int iCount = 0;
00020     int i;
00021
00022     /* Bus initialization */
00023     OW_Init();
00024
00025     /* Ready bus for communication */
00026     OW_WeakPullUp();
00027
00028     /* Search for first 1-Wire device */
00029     iAddress = OW_SearchFirst(0);
00030
00031     /* Store all device addresses into a array */
00032     while ((iAddress) && (iCount < MAX_DEVICES)) {
00033         iCount++;
00034         Addresses[iCount - 1] = iAddress;
00035         iAddress = OW_SearchNext();
00036     }
00037
00038     if (iCount == 0) {
00039         printf("No devices found.\r\n");
00040         return 1;
00041     }
00042
00043     /* Reset communication because the last device remained selected */
00044     OW_Reset();
00045
00046     for (i = 0; i < iCount; i++) {
00047         OW_Reset();
00048         OW_ByteWrite(SOME_COMMAND);
00049         printf("Device %d response = %d.\r\n", i, OW_ByteRead());
00050     }
00051
00052     printf("Finished.\r\n");
00053
00054     return 0;
00055 }

```

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_GPIO, 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

Copyright (c) 2013, Vojtěch Vigner vojtech.vigner@gmail.com

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- The name of the contributors can not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file [OneWire.h](#).

1.3.3 Function Documentation

1.3.3.1 uint8_t OW_Read (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_Write (const uint8_t bBit)

Write one bit.

Parameters

<i>bBit</i>	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

<i>bByte</i>	Byte to be transmitted.
--------------	-------------------------

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

<i>iCRC</i>	Input CRC value.
<i>iValue</i>	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_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    /* Enable clock for peripherals */
    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_InitStructure.GPIO_Pin = OW_TX_PIN_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(OW_TX_PIN_PORT, &GPIO_InitStructure);

    /* 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 is 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

<i>iAddress</i>	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

<i>iFamilyCode</i>	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](#).

```

{
#ifdef 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](#).

```

{
#ifdef 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

```

00001
00048 #ifndef ONEWIRE_H
00049 #define ONEWIRE_H
00050
00051 #ifdef __cplusplus
00052 extern "C" {
00053 #endif
00054
00055 #include "stdint.h"
00056
00057 /*****
00058 */
00058 /* Hardware specific configuration */
00059 #include "stm32f4xx_gpio.h"
00060 #include "stm32f4xx_usart.h"
00061 #include "stm32f4xx_rcc.h"
00062
00063 #define OW_TX_PIN_PORT      GPIOD
00064 #define OW_TX_PIN_PIN      GPIO_Pin_5
00065
00066 #define OW_USART            USART2
00067 #define OW_USART_AF        GPIO_AF_USART2
00068
00069 #define OW_GPIO_CLOCK()     RCC_APB1PeriphClockCmd(RCC_AHB1Periph_GPIOD,
00070     ENABLE)
00070 #define OW_USART_CLOCK()    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,
00071     ENABLE)
00071 /*****
00072 */
00072 /* Enables parasite powered device support */
00073 #define OW_PARASITE_POWERED 1
00074
00075 /* Public defines */
00076 #define OW_ADDRESS_ALL      0
00077
00078 typedef enum _OW_State {
00079     OW_OK = 0,
00080     OW_PRESENT,
00081     OW_NO_DEV,
00082     OW_CRC_ERROR
00083 } OW_State;
00084
00085
00086 /* Hardware initialization */
00087 void OW_Init(void);
00088
00089 /* Communication functions */
00090 OW_State OW_Reset(void);
00091 uint8_t OW_BitRead(void);
00092 uint8_t OW_ByteRead(void);
00093 void OW_BitWrite(const uint8_t bBit);
00094 void OW_ByteWrite(const uint8_t bByte);
00095
00096 /* Utilities */
00097 uint8_t OW_CRCCalculate(uint8_t iCRC, uint8_t iValue);
00098
00099 /* 1-Wire search */
00100 void OW_FamilySkipSetup(void);
00101 uint64_t OW_SearchFirst(uint8_t iFamilyCode);
00102 uint64_t OW_SearchNext(void);
00103
00104 /* Parasite powered devices support */
00105 void OW_StrongPullUp(void);
00106 void OW_WeakPullUp(void);
00107
00108 /* ROM operations */
00109 uint64_t OW_ROMRead(void);
00110 OW_State OW_ROMMatch(uint64_t iAddress);
00111 OW_State OW_ROMSkip(void);
00112
00113 #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

Copyright (c) 2013, Vojtěch Vigner vojtech.vigner@gmail.com

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of the contributors can not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT

OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

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

<i>bBit</i>	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

<i>bByte</i>	Byte to be transmitted.
--------------	-------------------------

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

<i>iCRC</i>	Input CRC value.
<i>iValue</i>	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_InitStructure;
        USART_InitTypeDef USART_InitStructure;

        /* Enable clock for peripherals */
        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_InitStructure.GPIO_Pin = OW_TX_PIN_PIN;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
        GPIO_Init(OW_TX_PIN_PORT, &GPIO_InitStructure);

        /* 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 is 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

<i>iAddress</i>	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

<i>iFamilyCode</i>	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](#).

```

{
#ifdef 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](#).

```

{
#ifdef 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"

```



```

00074
00075 /* Bus specific commands */
00076 #define OW_ROM_READ      0x33
00077 #define OW_ROM_MATCH     0x55
00078 #define OW_ROM_SKIP      0xCC
00079 #define OW_ROM_SEARCH    0xF0
00080 #define OW_ALARM_SEARCH  0xEC
00081
00082 /* Private defines */
00083 #define OW_R              0xF0
00084 #define OW_0              0x00
00085 #define OW_1              0xFF
00086
00087 /* Search related variables */
00088 struct {
00089     uint8_t iLastDeviceFlag;
00090     uint8_t iLastDiscrepancy;
00091     uint8_t iLastFamilyDiscrepancy;
00092     uint64_t ROM;
00093 } stSearch;
00094
00095 /* Backup of BRR register for different communication speeds*/
00096 uint16_t iUSART9600;
00097 uint16_t iUSART115200;
00098
00099 /* CRC calculation table */
00100 static uint8_t CRCTable[] = {
00101     0, 94, 188, 226, 97, 63, 221, 131, 194, 156, 126, 32, 163, 253, 31, 65,
00102     157, 195, 33, 127, 252, 162, 64, 30, 95, 1, 227, 189, 62, 96, 130, 220,
00103     35, 125, 159, 193, 66, 28, 254, 160, 225, 191, 93, 3, 128, 222, 60, 98,
00104     190, 224, 2, 92, 223, 129, 99, 61, 124, 34, 192, 158, 29, 67, 161, 255,
00105     70, 24, 250, 164, 39, 121, 155, 197, 132, 218, 56, 102, 229, 187, 89, 7,
00106     219, 133, 103, 57, 186, 228, 6, 88, 25, 71, 165, 251, 120, 38, 196, 154,
00107     101, 59, 217, 135, 4, 90, 184, 230, 167, 249, 27, 69, 198, 152, 122, 36,
00108     248, 166, 68, 26, 153, 199, 37, 123, 58, 100, 134, 216, 91, 5, 231, 185,
00109     140, 210, 48, 110, 237, 179, 81, 15, 78, 16, 242, 172, 47, 113, 147, 205,
00110     17, 79, 173, 243, 112, 46, 204, 146, 211, 141, 111, 49, 178, 236, 14, 80,
00111     175, 241, 19, 77, 206, 144, 114, 44, 109, 51, 209, 143, 12, 82, 176, 238,
00112     50, 108, 142, 208, 83, 13, 239, 177, 240, 174, 76, 18, 145, 207, 45, 115,
00113     202, 148, 118, 40, 171, 245, 23, 73, 8, 86, 180, 234, 105, 55, 213, 139,
00114     87, 9, 235, 181, 54, 104, 138, 212, 149, 203, 41, 119, 244, 170, 72, 22,
00115     233, 183, 85, 11, 136, 214, 52, 106, 43, 117, 151, 201, 74, 20, 246, 168,
00116     116, 42, 200, 150, 21, 75, 169, 247, 182, 232, 10, 84, 215, 137, 107, 53
00117 };
00118
00122 void OW_Init(void) {
00123     GPIO_InitTypeDef GPIO_InitStructure;
00124     USART_InitTypeDef USART_InitStructure;
00125
00126     /* Enable clock for peripherals */
00127     OW_GPIO_CLOCK();
00128     OW_USART_CLOCK();
00129
00130     /* Alternate function config on TX pin */
00131     GPIO_PinAFConfig(OW_TX_PIN_PORT, OW_TX_PIN_PIN, OW_USART_AF);
00132
00133     /* TX pin configuration */
00134     GPIO_InitStructure.GPIO_Pin = OW_TX_PIN_PIN;
00135     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
00136     GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
00137     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
00138     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
00139     GPIO_Init(OW_TX_PIN_PORT, &GPIO_InitStructure);
00140
00141     /* USART configuration */
00142     USART_InitStructure.USART_BaudRate = 115200;
00143     USART_InitStructure.USART_WordLength = USART_WordLength_8b;
00144     USART_InitStructure.USART_StopBits = USART_StopBits_1;
00145     USART_InitStructure.USART_Parity = USART_Parity_No;
00146     USART_InitStructure.USART_HardwareFlowControl =
        USART_HardwareFlowControl_None;
00147     USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
00148     USART_Init(OW_USART, &USART_InitStructure);
00149
00150     /* BRR register backup for 115200 Baud */
00151     iUSART115200 = OW_USART->BRR;
00152
00153     /* BRR register backup for 9600 Baud */
00154     USART_StructInit(&USART_InitStructure);
00155     USART_InitStructure.USART_BaudRate = 9600;
00156     USART_Init(OW_USART, &USART_InitStructure);
00157     iUSART9600 = OW_USART->BRR;
00158
00159     /* Half duplex enable, for single pin communication */
00160     USART_HalfDuplexCmd(OW_USART, ENABLE);
00161
00162     /* USART enable */

```

```

00163     USART_Cmd(OW_USART, ENABLE);
00164 }
00165
00170 uint8_t OW_BitRead(void) {
00171     /* Make sure that all communication is done and receive buffer is cleared
00172     */
00172     while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);
00173     while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
00174         USART_ReceiveData(OW_USART);
00175
00176     /* Send byte */
00177     USART_SendData(OW_USART, OW_1);
00178
00179     /* Wait for response */
00180     while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);
00181
00182     /* Receive data */
00183     if (USART_ReceiveData(OW_USART) != OW_1) return 0;
00184
00185     return 1;
00186 }
00187
00192 uint8_t OW_ByteRead(void) {
00193     int i;
00194     uint8_t iRet = 0;
00195
00196     /* Read 8 bits */
00197     for (i = 0; i < 8; i++) {
00198         if (OW_BitRead()) iRet |= (1 << i);
00199     }
00200
00201     return iRet;
00202 }
00203
00208 void OW_BitWrite(const uint8_t bBit) {
00209     uint8_t bData = OW_0;
00210
00211     if (bBit) bData = OW_1;
00212
00213     /* Make sure that all communication is done */
00214     while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
00215         USART_ReceiveData(OW_USART);
00216     while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);
00217
00218     /* Send byte */
00219     USART_SendData(OW_USART, bData);
00220 }
00221
00226 void OW_ByteWrite(const uint8_t bByte) {
00227     uint8_t i;
00228
00229     /* Write 8 bits */
00230     for (i = 0; i < 8; i++) {
00231         OW_BitWrite(bByte & (1 << i));
00232     }
00233 }
00234
00238 void OW_StrongPullUp(void) {
00239     #if OW_PARASITE_POWERED
00240         USART_HalfDuplexCmd(OW_USART, DISABLE);
00241         GPIO_SetBits(OW_TX_PIN_PORT, OW_TX_PIN_PIN);
00242         OW_TX_PIN_PORT->OTYPER &= ~(OW_TX_PIN_PIN);
00243     #endif
00244 }
00245
00249 void OW_WeakPullUp(void) {
00250     #if OW_PARASITE_POWERED
00251         GPIO_SetBits(OW_TX_PIN_PORT, OW_TX_PIN_PIN);
00252         OW_TX_PIN_PORT->OTYPER |= OW_TX_PIN_PIN;
00253         USART_HalfDuplexCmd(OW_USART, ENABLE);
00254     #endif
00255 }
00256
00261 OW_State OW_Reset(void) {
00262     uint8_t iPresence;
00263
00264     /* Set USART baudrate to 9600 Baud */
00265     OW_USART->BRR = iUSART9600;
00266
00267     /* Make sure that all communication is done and receive buffer is cleared
00268     */
00268     USART_ClearFlag(OW_USART, USART_FLAG_TC);
00269     while (USART_GetFlagStatus(OW_USART, USART_FLAG_RXNE) == SET)
00270         USART_ReceiveData(OW_USART);
00271
00272     /* Write special byte on USART */
00273     USART_SendData(OW_USART, OW_R);

```

```

00274     while (USART_GetFlagStatus(OW_USART, USART_FLAG_TC) == RESET);
00275
00276     /* Receive data from USART */
00277     iPresence = USART_ReceiveData(OW_USART);
00278
00279     /* Set USART baudrate to 115200 Baud */
00280     OW_USART->BRR = iUSART115200;
00281
00282     /* If received data is equal to data transmitted means that there is no
00283        device present on the bus. Return value equal to 0 means bus error. */
00284     if ((iPresence != OW_R) && ((iPresence != 0x00))) return OW_PRESENT;
00285
00286     return OW_NO_DEV;
00287 }
00288
00293 void OW_FamilySkipSetup(void) {
00294     /* Set the last discrepancy to last family discrepancy */
00295     stSearch.iLastDiscrepancy = stSearch.iLastFamilyDiscrepancy;
00296     stSearch.iLastFamilyDiscrepancy = 0;
00297
00298     /* Check for end of list */
00299     if (stSearch.iLastDiscrepancy == 0) stSearch.iLastDeviceFlag = 1;
00300 }
00301
00308 uint8_t OW_CRCCalculate(uint8_t iCRC, uint8_t iValue) {
00309     return CRCTable[iCRC ^ iValue];
00310 }
00311
00317 uint64_t OW_SearchFirst(uint8_t iFamilyCode) {
00318     if (iFamilyCode) {
00319         stSearch.ROM = (uint64_t) iFamilyCode;
00320
00321         stSearch.iLastDiscrepancy = 64;
00322         stSearch.iLastFamilyDiscrepancy = 0;
00323         stSearch.iLastDeviceFlag = 1;
00324     } else {
00325         stSearch.ROM = 0;
00326         stSearch.iLastDiscrepancy = 0;
00327         stSearch.iLastDeviceFlag = 0;
00328         stSearch.iLastFamilyDiscrepancy = 0;
00329     }
00330
00331     return OW_SearchNext();
00332 }
00333
00339 uint64_t OW_SearchNext(void) {
00340     uint8_t iSearchDirection;
00341     int iIDBit, iCmpIDBit;
00342
00343     /* Initialize for search */
00344     uint8_t iROMByteMask = 1;
00345     uint8_t iCRC = 0;
00346     int iIDBitNumber = 1;
00347     int iLastZero = 0;
00348     int iROMByteNumber = 0;
00349     int iSearchResult = 0;
00350
00351     /* If the last call was not the last one */
00352     if (!stSearch.iLastDeviceFlag) {
00353         /* 1-Wire reset */
00354         if (!OW_Reset()) {
00355             /* Reset the search */
00356             stSearch.iLastDiscrepancy = 0;
00357             stSearch.iLastDeviceFlag = 0;
00358             stSearch.iLastFamilyDiscrepancy = 0;
00359             return 0;
00360         }
00361
00362         /* Issue the search command */
00363         OW_ByteWrite(OW_ROM_SEARCH);
00364
00365         /* Loop to do the search */
00366         do {
00367             /* Read a bit and its complement */
00368             iIDBit = OW_BitRead();
00369             iCmpIDBit = OW_BitRead();
00370
00371             /* Check for no devices on 1-wire */
00372             if ((iIDBit == 1) && (iCmpIDBit == 1))
00373                 break;
00374             else {
00375                 /* All devices coupled have 0 or 1 */
00376                 if (iIDBit != iCmpIDBit)
00377                     /* Bit write value for search */
00378                     iSearchDirection = iIDBit;
00379                 else {
00380                     /* if this discrepancy is before the Last Discrepancy

```

```

00381             on a previous next then pick the same as last time */
00382             if (iIDBitNumber < stSearch.iLastDiscrepancy)
00383             {
00384                 iROMByteNumber] & iROMByteMask) > 0);
00385             }
00386             /* If equal to last pick 1, if not then pick 0 */
00387             iSearchDirection = (iIDBitNumber == stSearch.
00388             iLastDiscrepancy);
00389
00390             /* If 0 was picked then record its position in iLastZero */
00391             if (iSearchDirection == 0) {
00392                 iLastZero = iIDBitNumber;
00393             }
00394             /* Check for Last discrepancy in family */
00395             if (iLastZero < 9)
00396                 stSearch.iLastFamilyDiscrepancy = iLastZero;
00397             }
00398             /* Set or clear the bit in the ROM byte with mask rom_byte_mask
00399             */
00400             if (iSearchDirection == 1)
00401                 ((uint8_t*) & stSearch.ROM)[iROMByteNumber] |= iROMByteMask
00402             ;
00403             else
00404                 ((uint8_t*) & stSearch.ROM)[iROMByteNumber] &= ~
00405             iROMByteMask;
00406
00407             /* Set serial number search direction */
00408             OW_BitWrite(iSearchDirection);
00409
00410             /* Increment the byte counter and shift the mask */
00411             iIDBitNumber++;
00412             iROMByteMask <<= 1;
00413
00414             /* If the mask is 0 then go to new ROM byte number and reset
00415             mask */
00416             if (iROMByteMask == 0) {
00417                 /* Accumulate the CRC */
00418                 iCRC = OW_CRCCalculate(iCRC, ((uint8_t*) &
00419                 stSearch.ROM)[iROMByteNumber]);
00420                 iROMByteNumber++;
00421                 iROMByteMask = 1;
00422             }
00423             } while (iROMByteNumber < 8); /* Loop until through all ROM bytes 0-7
00424             */
00425
00426             /* If the search was successful then */
00427             if (!(iIDBitNumber < 65) || (iCRC != 0)) {
00428                 stSearch.iLastDiscrepancy = iLastZero;
00429             }
00430             /* Check for last device */
00431             if (stSearch.iLastDiscrepancy == 0)
00432                 stSearch.iLastDeviceFlag = 1;
00433             iSearchResult = 1;
00434             }
00435             }
00436             /* If no device found then reset counters so next 'search' will be like a
00437             first */
00438             if (!iSearchResult || !((uint8_t*) & stSearch.ROM)[0]) {
00439                 stSearch.iLastDiscrepancy = 0;
00440                 stSearch.iLastDeviceFlag = 0;
00441                 stSearch.iLastFamilyDiscrepancy = 0;
00442                 return 0;
00443             }
00444             return stSearch.ROM;
00445         }
00446     }
00447
00448     uint64_t OW_ROMRead(void) {
00449         uint64_t iRes = 0;
00450         int i;
00451
00452         if (OW_Reset() == OW_NO_DEV) return 0;
00453
00454         OW_ByteWrite(OW_ROM_READ);
00455         for (i = 0; i < 8; i++)
00456             ((uint8_t*) & iRes)[i] = OW_ByteRead();
00457
00458         return iRes;
00459     }
00460
00461     OW_State OW_ROMMatch(uint64_t iAddress) {
00462         int i;

```

```
00468
00469     if (iAddress == OW_ADDRESS_ALL) return OW_ROMSkip();
00470
00471     if (OW_Reset() == OW_NO_DEV) return OW_NO_DEV;
00472
00473     OW_ByteWrite(OW_ROM_MATCH);
00474     for (i = 0; i < 8; i++)
00475         OW_ByteWrite(((uint8_t*) & iAddress)[i]);
00476
00477     return OW_OK;
00478 }
00479
00484 OW_State OW_ROMSkip(void) {
00485     if (OW_Reset() == OW_NO_DEV) return OW_NO_DEV;
00486
00487     OW_ByteWrite(OW_ROM_SKIP);
00488
00489     return OW_OK;
00490 }
```

Index

example/Example_OneWire.c, [1](#)

inc/OneWire.h, [2](#), [11](#)

OW_BitRead

OneWire.c, [14](#)

OneWire.h, [4](#)

OW_BitWrite

OneWire.c, [14](#)

OneWire.h, [4](#)

OW_ByteRead

OneWire.c, [14](#)

OneWire.h, [5](#)

OW_ByteWrite

OneWire.c, [15](#)

OneWire.h, [5](#)

OW_CRCCalculate

OneWire.c, [15](#)

OneWire.h, [5](#)

OW_FamilySkipSetup

OneWire.c, [15](#)

OneWire.h, [6](#)

OW_Init

OneWire.c, [16](#)

OneWire.h, [6](#)

OW_ROMMatch

OneWire.c, [17](#)

OneWire.h, [7](#)

OW_ROMRead

OneWire.c, [17](#)

OneWire.h, [7](#)

OW_ROMSkip

OneWire.c, [18](#)

OneWire.h, [8](#)

OW_Reset

OneWire.c, [16](#)

OneWire.h, [6](#)

OW_SearchFirst

OneWire.c, [18](#)

OneWire.h, [8](#)

OW_SearchNext

OneWire.c, [18](#)

OneWire.h, [8](#)

OW_StrongPullUp

OneWire.c, [20](#)

OneWire.h, [10](#)

OW_WeakPullUp

OneWire.c, [20](#)

OneWire.h, [10](#)

OneWire.c

OW_BitRead, [14](#)

OW_BitWrite, [14](#)

OW_ByteRead, [14](#)

OW_ByteWrite, [15](#)

OW_CRCCalculate, [15](#)

OW_FamilySkipSetup, [15](#)

OW_Init, [16](#)

OW_ROMMatch, [17](#)

OW_ROMRead, [17](#)

OW_ROMSkip, [18](#)

OW_Reset, [16](#)

OW_SearchFirst, [18](#)

OW_SearchNext, [18](#)

OW_StrongPullUp, [20](#)

OW_WeakPullUp, [20](#)

OneWire.h

OW_BitRead, [4](#)

OW_BitWrite, [4](#)

OW_ByteRead, [5](#)

OW_ByteWrite, [5](#)

OW_CRCCalculate, [5](#)

OW_FamilySkipSetup, [6](#)

OW_Init, [6](#)

OW_ROMMatch, [7](#)

OW_ROMRead, [7](#)

OW_ROMSkip, [8](#)

OW_Reset, [6](#)

OW_SearchFirst, [8](#)

OW_SearchNext, [8](#)

OW_StrongPullUp, [10](#)

OW_WeakPullUp, [10](#)

src/OneWire.c, [11](#), [20](#)