AN3281
Application note

STM8 8-bit MCUs I²C optimized examples

Introduction

This document describes how to use the following I²C optimized examples:

- Hardware configuration example of a common I²C bus
- Master firmware examples in polling mode
- Master firmware examples with interrupt
- Slave firmware examples.

Reference documents

- I²C-bus specification, version 2.1, January 2000, NXP
- *STM8S series and STM8AF series 8-bit microcontrollers reference manual* (RM0016)
- *STM8L051/L052 Value Line, STM8L151/L152, STM8L162, STM8AL31, STM8AL3L MCU lines reference manual* (RM0031)
- *STM8L101xx microcontroller family reference manual* (RM0013)
- *STM8TL5xxx microcontroller family reference manual* (RM0312).

Contents

List of tables

List of figures

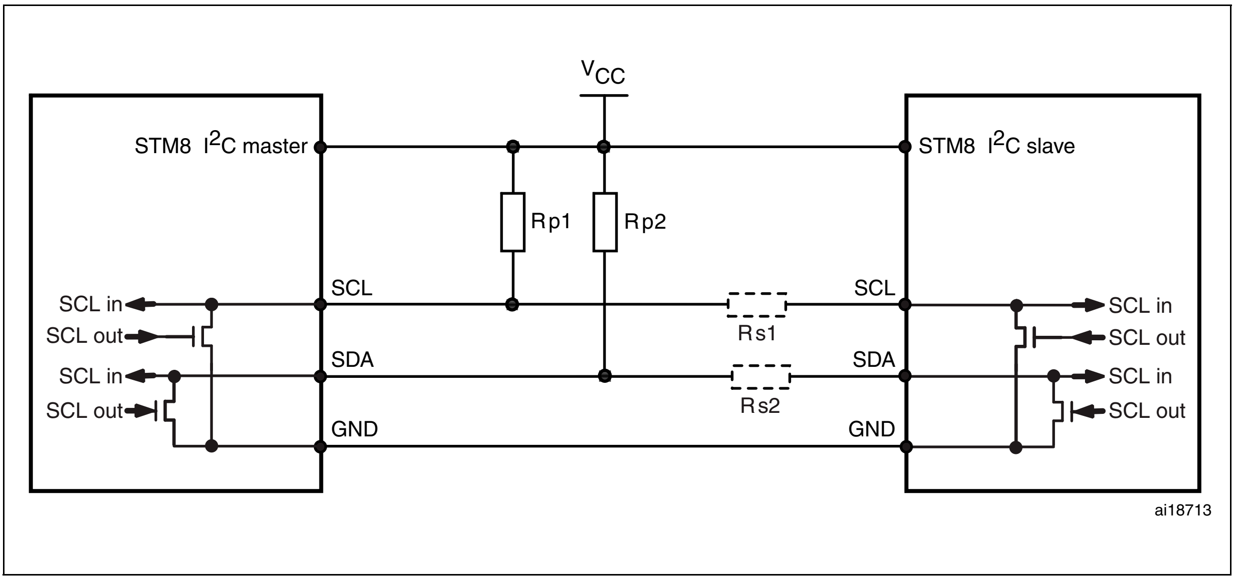
1 Hardware configuration example of a common I^2C bus

The firmware examples provided within this application note illustrate the basics of the I^2C communication protocol on the STM8 microcontrollers. In these examples, the I^2C peripheral is used to communicate between two STM8 devices.

The I^2C can also be reused and customized to fit a specific application which requires an I^2C communication with another device using the I^2C protocol. In these examples, the master and slave work together and transmit data through the bus.

At all times, the I^2C protocol is respected (see the I^2C-bus specification). *Figure 1* shows the hardware configuration that must be followed.

Figure 1. Hardware configuration example of a common I^2C bus



1. Legend:
   V_{CC} = supply voltage, typically ranging from 1.8 V to 5 V
   SDA = Serial data (I^2C data line)
   SCL = Serial clock (I^2C clock line)
   Rp1, Rp2 = Pull-up resistor used to set the bus idle voltage to V_{CC}. Also called the I^2C termination.
   Rs1, Rs2 = Optional 100 Ohm serial resistor used to ease differentiation between master and slave when analyzing communication waveforms on the oscilloscope. These resistors must be placed on one extremity of the bus (on the master or slave side).

2 Software of I^2C firmware examples

The software of all I^2C firmware examples is divided into two basic levels:

- **An application layer** (main.c) - which is an example of how to implement all the I^2C procedures. It must be replaced by the usercode in the final application.
- **A data link layer** (I2C_xxx.c) - which manages the data flow process and hardware control. The user should not change the software at this level. All processes at data link level are managed by a set of predefined functions contained in the data link layer. These functions are called from the application level.

3 Master firmware examples in polling mode

3.1 Application layer example in polling mode

This layer simulates an I^2C memory access with an offset command. It should be used with the example of the provided I^2C slave.

After peripheral initialization, the program runs in a testing loop. This sends a succession of bytes to be stored in the slave memory and then reads them back. After each loop, all sent and received values are compared for integrity checking purposes. The first byte of every message is used as a memory offset command for the data storage register.

All read and write procedures performed on the I^2C bus are managed by calling dedicated functions from the data link layer. Their execution times are guarded by a timeout which is serviced by a dedicated timer. This timeout is reset at the start of every testing loop and is checked at the end of every loop. If the timer counter reaches 0, it means that one I^2C communication is stuck.

3.2 Data link layer example in polling mode

All I^2C activities of the data link layer (except errors) are performed and checked by polling. Errors are handled by the I^2C interrupt service. The specific functions for I^2C flow control are predefined in this part of the firmware. They are called by the application layer to control all I^2C processes and they can be customized by the user according to the application needs.

The address of the slave is fixed in form of a compilation parameter. These procedures follow specific processes which cover all the known I^2C errata issues (see *Figure 4* and *Figure 7*).

3.2.1 Functions predefined to control the data flow on master firmware in polling mode (to be customized)

I2C_WriteRegister function

This function sends an offset/command byte followed by a defined number of data bytes from a specific data field. This function can also be used to send one byte if it is called with zero number of data bytes. In the example in *Figure 2*, the first (command) byte is interpreted as an offset from which data is stored in the slave device.

Prototype

void Function I2C_WriteRegister (u8 offset_command, u8 number_of_data_bytes, u8 *data_field_address).

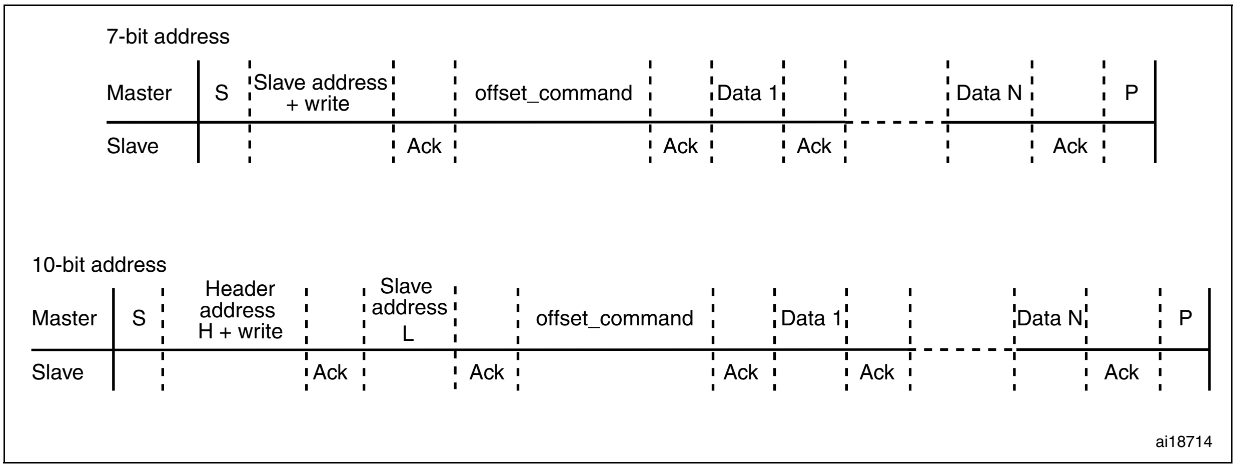
Parameters

- offset_command: first byte to send during communication. Can be used as offset, command, or first datum.
- number_of_data_bytes: number of bytes to be sent. Value from 0 to 255.
- *data_field_address: pointer to first address of data to be sent.

Return value

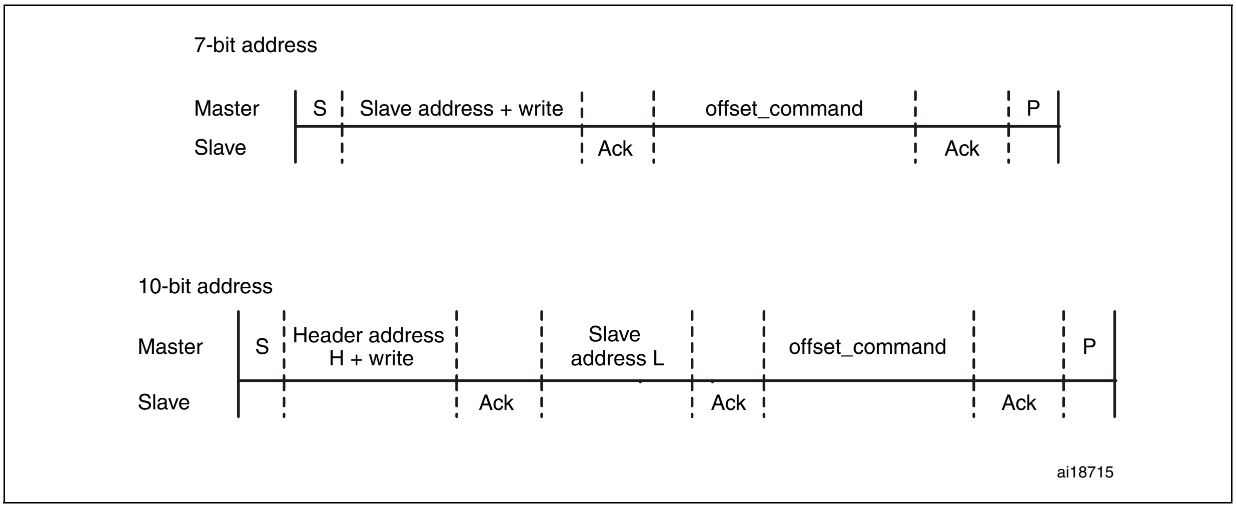
None.

Figure 2. N-data byte write sequences preceded by a one-command byte



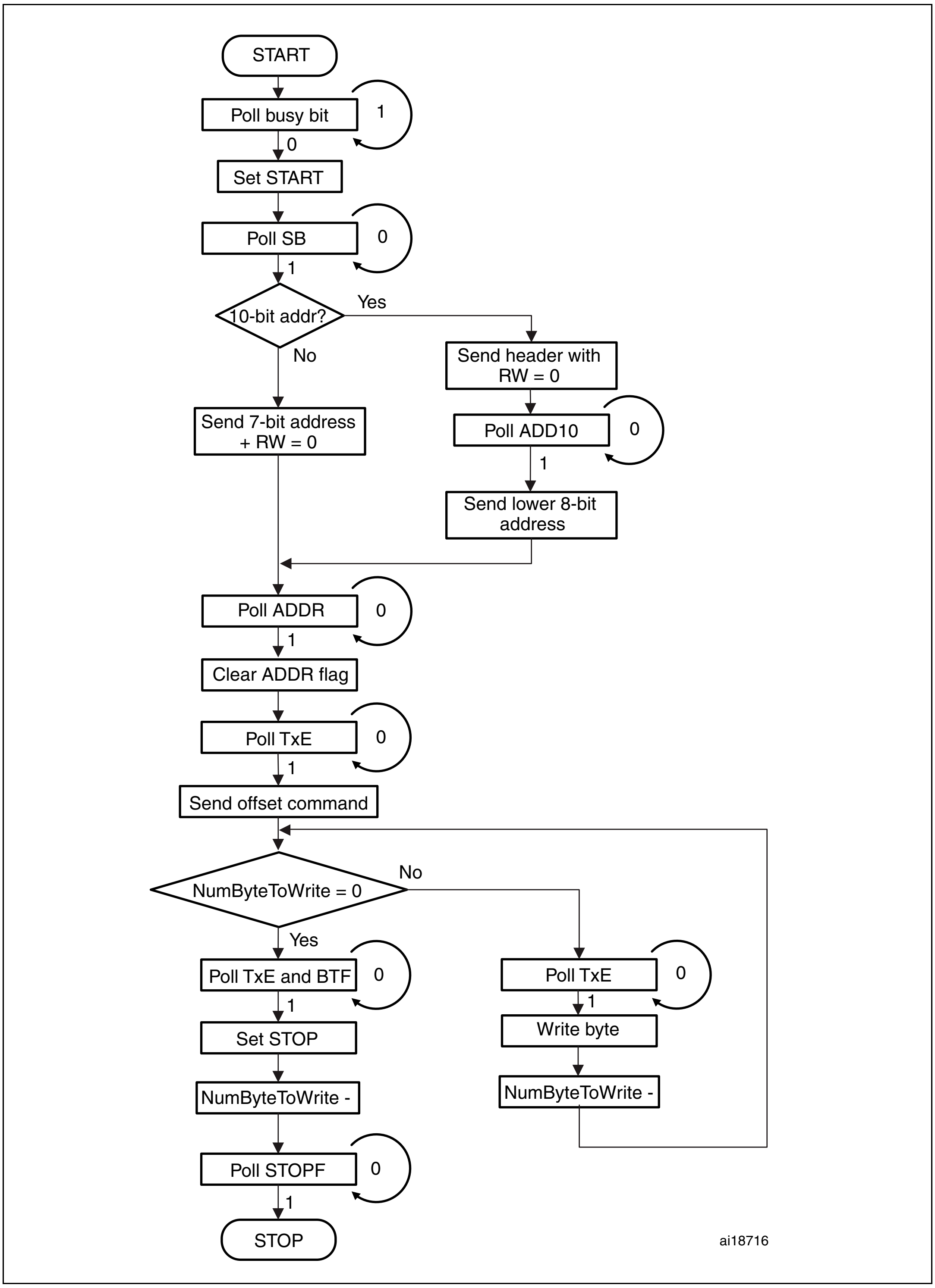
1. Legend: S = start, P = stop, H = high, L = low

Figure 3. One-datum or command byte write sequence



1. Legend: S = start, P = stop, H = high, L = low

Figure 4. Flowchart of data-write sequences made by the I2C_WriteRegister() function



1. Legend: SB = start bit, RW = read/write bit

I2C_ReadRegister function

This function sends one byte (offset_command value) to the slave. It then restarts the bus and continues communication by reading a defined number of data bytes. Bytes are stored in a specific data field starting from a specified address. The offset_command value depends on the slave device interpretation. In this example, it is used as a memory offset. It can be used in other applications as a command for specific I^2C peripherals.

Prototype

void Function I2C_ReadRegister (u8 offset_command, u8 number_of_data_bytes, u8 *data_field_address).

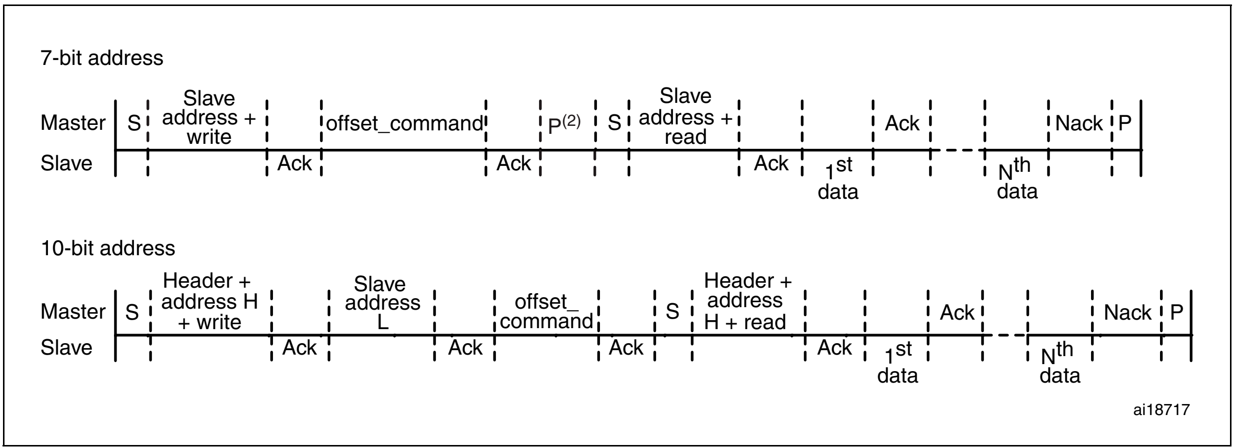
Parameters

- offset_command: first byte to send during communication. Can be used as offset, command, or first datum.
- number_of_data_bytes: number of bytes to read. Value from 0 to 255.
- *data_field_address: pointer to first address of data to store received data.

Return value

None.

Figure 5. N-data byte read sequences preceded by a one-command byte



1. Legend: S = start, P = stop, H = high, L = low
2. Stop is not mandatory in this sequence and can be skipped by defining the “NO_RESTART” constant in the driver header file (I2C_master_poll.h).

I2C_RandomRead function

This function reads directly the requested data from the slave. No offset or command byte is written previously. It can be used as a standard or continuous read (where auto-incrementation of addresses is available on the slave side). Received bytes are stored in data fields starting from specified data field addresses.

Prototype

void Function I2C_RandomRead (u8 number_of_data_bytes, u8 *data_field_address).

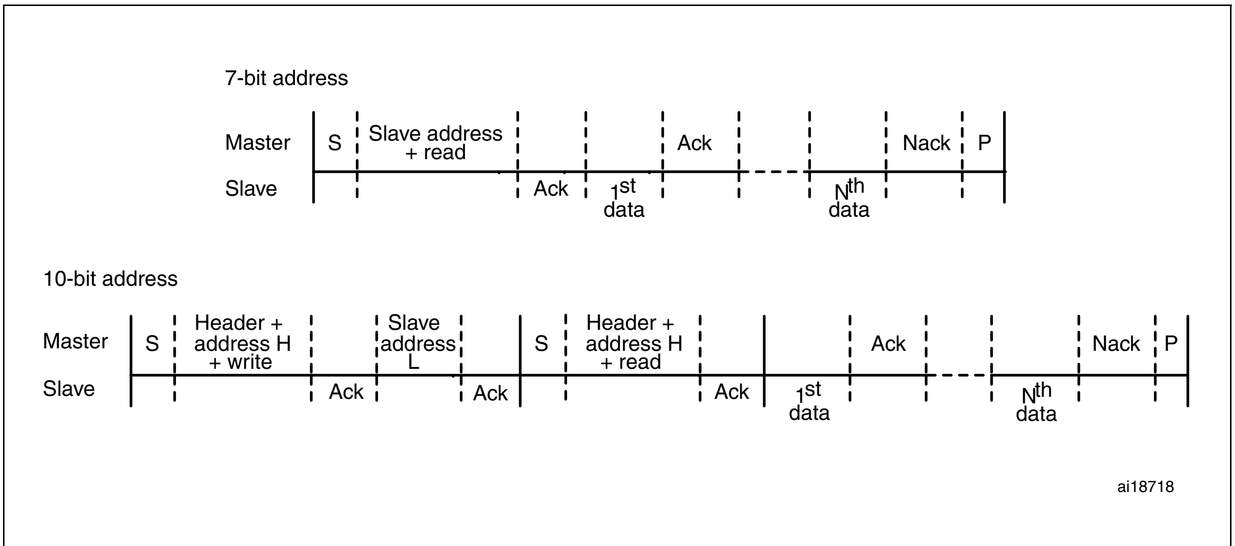
Parameters

- number_of_data_bytes: number of bytes to read. Value from 0 to 255.
- *data_field_address: pointer to first address to store received data.

Return value

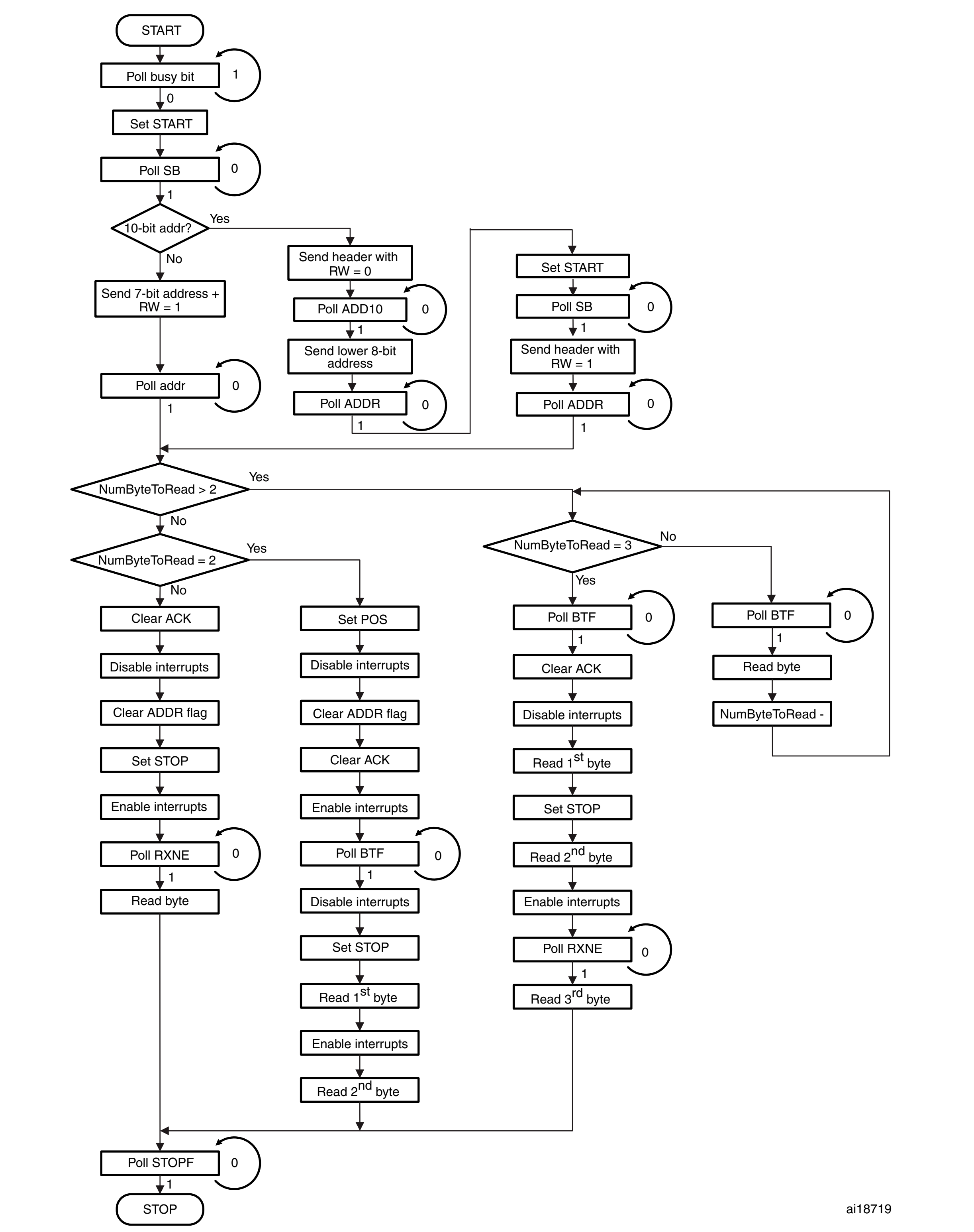
None.

Figure 6. N-data byte random read sequences (without any command)



1. Legend: S = start, P = stop, H = high, L = low

Figure 7. Flowchart of data read sequences made by the I2C_RandomRead() function



1. Legend: SB = start bit, RW = read/write bit

The I^2C read register function is the succession of one I^2C write register function call and one I^2C random read function call (see *Figure 4: Flowchart of data-write sequences made by the I2C_WriteRegister() function* and *Figure 7: Flowchart of data read sequences made by the I2C_RandomRead() function*).

4 Master firmware examples with interrupt

4.1 Master application layer example with interrupt

The function and purpose of this layer is the same as for polling mode. After peripheral initialization, the program stays in a testing loop. This sends a succession of bytes to be stored into the slave memory and then reads back from the slave. After each loop, all sent and received values are compared for integrity checking purposes. For more details, please refer to *Section 3.1: Application layer example in polling mode*.

4.2 Master data link layer example with interrupt

All I^2C activities of the data link layer are handled by the I^2C interrupt service. This interrupt routine is managed by the internal state machine (see *Figure 8 on page 16*). The procedures included in this flowchart follow specific processes which cover all known I^2C errata issues (see *Figure 4 on page 9* and *Figure 7 on page 12*).

It is highly recommended not to change this layer to ensure that the application handles specific states on the I^2C bus. The specific functions for I^2C flow control are predefined in this part of the firmware. They are called by the application layer to control all I^2C processes and can be customized by the user according to the application needs.

4.2.1 Functions predefined to control the data flow on master firmware with interrupt (to be customized)

I2C_WriteRegister function

This function sets up and starts the state machine to perform an I^2C write process. It returns 1 when the process is started or 0 when the peripheral or line is busy.

Prototype

u8 I2C_WriteRegister (u16 SlaveAdd, u8 AddType, u8 NoStop, u8 NumByteToWrite, u8 *DataBuffer).

Parameters

- SlaveAdd: unsigned short number address of the slave.
- AddType: 7-bit (SEV_BIT_ADDRESS) or 10-bit addressing (TEN_BIT_ADDRESS).
- NoStop: stop is/is not performed after the transmission (STOP; NOSTOP).
- NumByteToWrite: number of bytes to be sent.
- DataBuffer: first data buffer address.

Return value

- 0 is returned if the write process is not started due to other I^2C operations.
- 1 is returned if the write process is started.

I2C_ReadRegister function

This function sets up and starts the state machine to perform an I^{2}C read process. It returns 1 when the process is started or 0 when the peripheral or line is busy.

Prototype

u8 I2C_ReadRegister(u16 SlaveAdd, u8 AddType, u8 NoStop, u8 NumByteToRead, u8 *DataBuffer);

Parameters

- SlaveAdd: unsigned short number address of the slave.
- AddType: 7-bit (SEV_BIT_ADDRESS) or 10-bit addressing (TEN_BIT_ADDRESS).
- NoStop: stop is/is not performed before the transmission (used for 10-bit addressing. mode when the complete address or header is sent depending on the STOP or NOSTOP flag).
- NumByteToRead: number of bytes to be received.
- DataBuffer: first data buffer address.

Return value

- 0 is returned if the read process is not started due to other I^{2}C operations.
- 1 is returned if the read process is started.

ErrProc function

This function is called from I^{2}C interrupt routines each time an error is detected. It can be customized according to the application needs.

Prototype

void ErrProc (void).

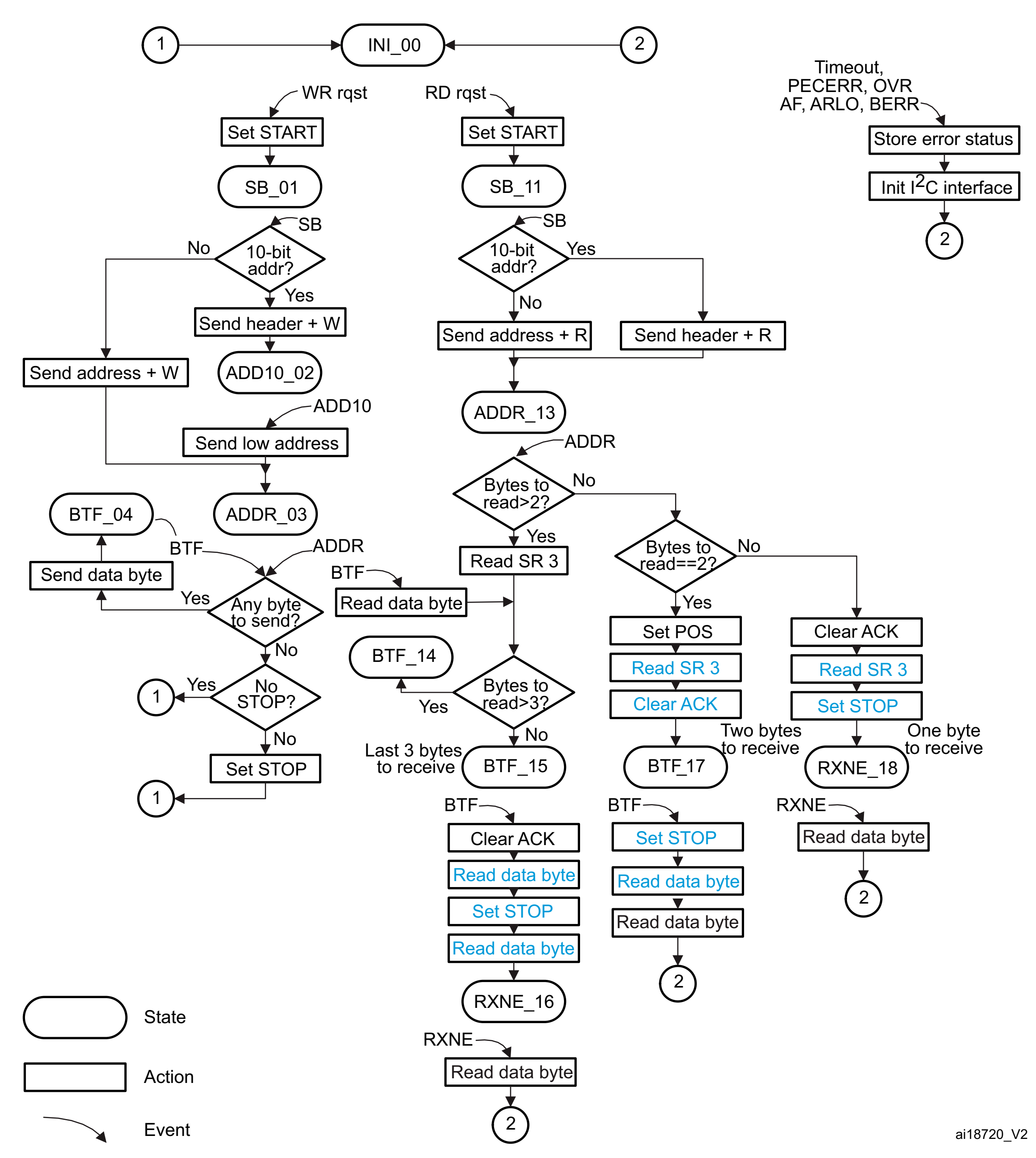
Parameters

None.

Return value

None.

Figure 8. I²C state machine flowchart



1. Legend: SB = start bit, W = write, R = read
2. The text in blue indicates the parts of this State machine which must be protected from interrupt by software disabling (see the device errata sheet).

Note: For a 10-bit address random read, a WriteRegister function call (without data and STOP) should be performed before a ReadRegister function call.

Example

// Send 10-bit slave address

I²C_WriteRegister (0x3F0,TEN_BIT_ADDRESS,NOSTOP,0,Buff);

// Read data from slave

I²C_ReadRegister (0x3F0,TEN_BIT_ADDRESS,STOP,3,Buff);

5 Slave firmware examples with interrupt

5.1 Slave application layer example with interrupt

This layer simulates an I^2C memory with an offset command example. The first datum received is interpreted as a command (memory offset). Interaction with the data link layer is made using specific customizable callback functions (see *Section 5.2.1: Functions predefined to control the data flow on slave firmware with interrupt (to be customized)*). These functions can be modified depending on the application needs.

5.2 Slave data link layer example with interrupt

All I^2C activities of the data link layer are handled by the I^2C interrupt service. All procedures in this interrupt service follow specific processes which cover all known I^2C errata issues (see *Figure 4* and *Figure 7*). It is highly recommended not to change this layer to ensure that the application handles specific states on the I^2C bus.

5.2.1 Functions predefined to control the data flow on slave firmware with interrupt (to be customized)

I2C_transaction_begin

This function is called every time a transaction with the slave begins (slave address recognized).

Prototype

void I2C_transaction_begin (void).

Parameters

None.

Return value

None.

I2C_transaction_end

This function is called every time a transaction with the slave ends (stop or Nack detected).

Prototype

void I2C_transaction_end (void).

Parameters

None.

Return value

None.

I2C_byte_received

This function is called every time a byte is received by the I^2C peripheral.
This example stores data in the memory.

Prototype

void I2C_byte_received (u8 u8_RxData).

Parameters

None.

Return value

None.

I2C_byte_write

This function is called every time a byte needs to be sent. It must return a u8 value which corresponds to the byte to be written on the I^2C line. In this example, the function returns selected stored data from the memory.

Prototype

u8 I2C_byte_write(void).

Parameters

None.

Return value

To be customized according to the application needs. The returned value is the datum which is written on the I^2C line.

ErrProc function

This function is called from the I^2C interrupt routines each time an error is detected. It can be customized according to the application needs.

Prototype

void ErrProc (void).

Parameters

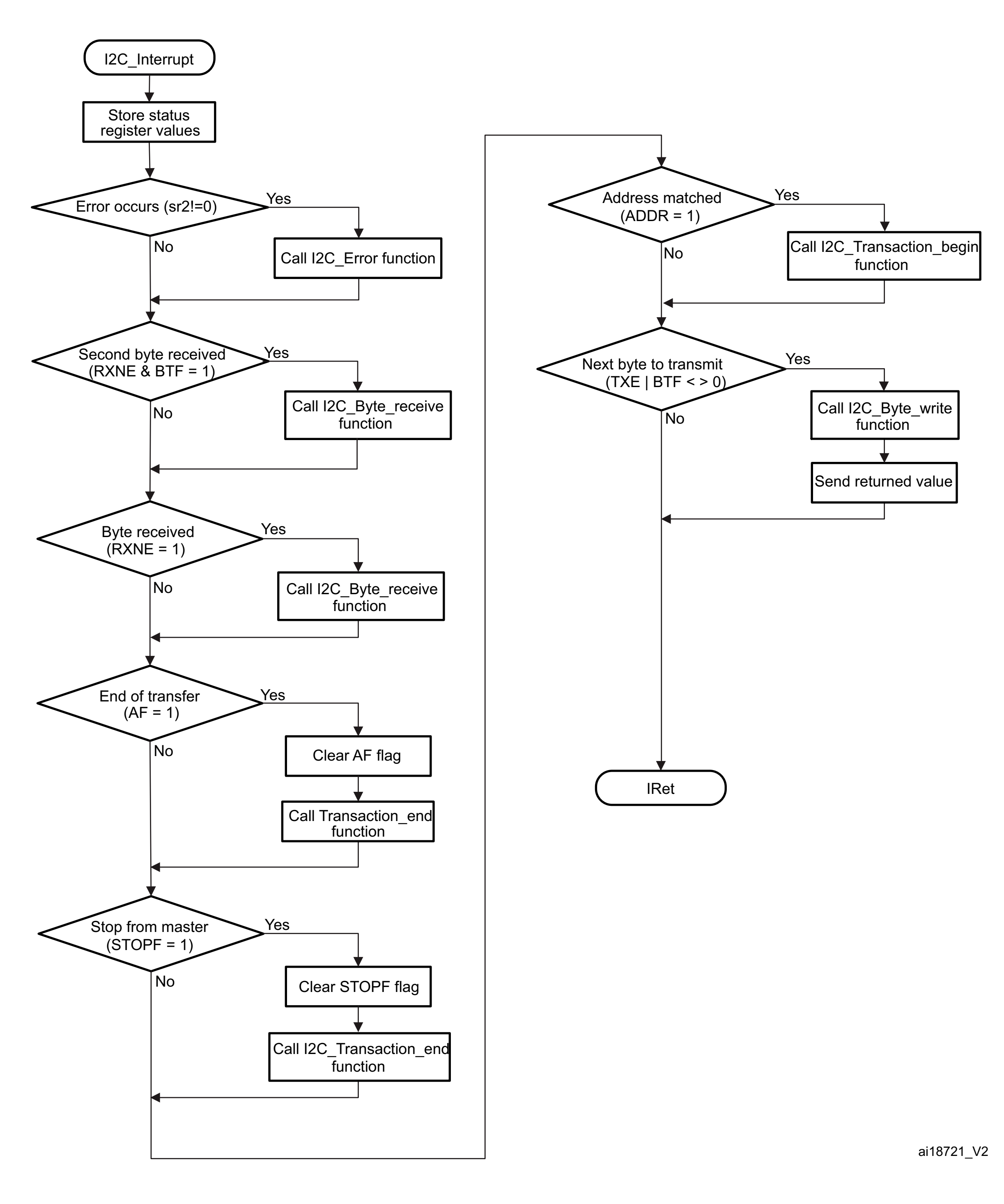
None.

Return value

None.

5.3 Data link layer flowchart

Figure 9. Data link layer flowchart



6 Revision history

Table 1. Document revision history

Date	Revision	Changes
20-Oct-2010	1	Initial release
20-Nov-2012	2	Document updated to include STM8AL and STM8TL5 devices. Added *Table 1: Applicable products*. Updated *Reference documents*.
09-May-2016	3	Modified *Figure 8* and *Figure 9*. Added *Section 5: Slave firmware examples with interrupt*.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved