NXP Semiconductors
Example Code

Rev. 0, 05/2016

MPC5xxx I2C communication driver

by Petr Stancik

1. Introduction

This document summarizes simple I2C driver implementation for MPC5xxx devices. The driver was written in the way the blocking or non-blocking communication on the I2C bus can be done.

The first one uses a SW poll to wait till a byte is transmitted, so the code stalls within transmit or receive function until the full I2C frame is finished.

In non-blocking communication the I2C interrupt is used so the CPU does not wait in the loops.

The code follows Reference Manual's Flow-Chart of Typical I2C Interrupt Routine.

Only I2C master is implemented in this driver version.

2. Functions description

Below tables show a meaning of input arguments and return values that is common for all below *write* and *read* functions

Arguments	
dev_addr	address selecting particular I2C device
reg_addr	address selecting register address or memory item's address
reg_addr_nBytes	address mode 1=8bit, 2=16bit, rest is invalid
*pData	pointer to the first item of input/output data buffer
nBytes	number of bytes to be written or read

Return	
uint8_t	0 … OK 1 … FRAME NO ENDED 2 … NO ACK 3 … BUS BUSY 4 … ARBITRATION LOST

OK is returned if the full I2C frame is completed without error. Thus transmitted data was accepted by the slave device and receive data from the slave is valid.

FRAME NO ENDED value indicates the I2C frame is not finished still. This is only returned in interrupt driven mode.

NO ACK value is returned if slave does not acknowledges the byte transmitted. A STOP bit is generated, thus user should repeat frame transmission/reception.

BUS BUSY is set when attempting to send new message over the I2C bus, but the bus is in the busy state.

ARBITRATION LOST is returned if IBAL flag is detected. This is set by hardware when the arbitration procedure is lost. I2C module immediately switch over to slave receive mode and stop driving the SDA output.

Similarly to *NO ACK* , if *BUS BUSY* and *ARBITRATION LOST* is returned user should repeat frame transmission/reception.

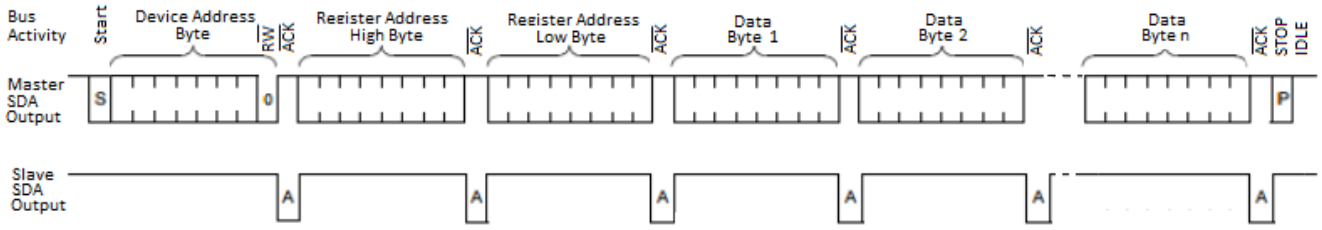
uint8_t I2C_0_WriteBlock(uint8_t dev_addr, uint16_t reg_addr, uint8_t reg_addr_nBytes, uint8_t *pData, uint8_t nBytes);

The function is used to write *nBytes* of input data buffer into defined *reg_addr* of the selected slave device.

For the polled mode it waits till all bytes are sent or an error appears.

For the interrupt mode the function just initiates the transmission of the first byte of the I2C write message, so should be called repeatedly until 0 is returned, meaning whole frame is transferred without errors.

The I2C frame then looks in the following way. A 16-bit register address is considered in this case.



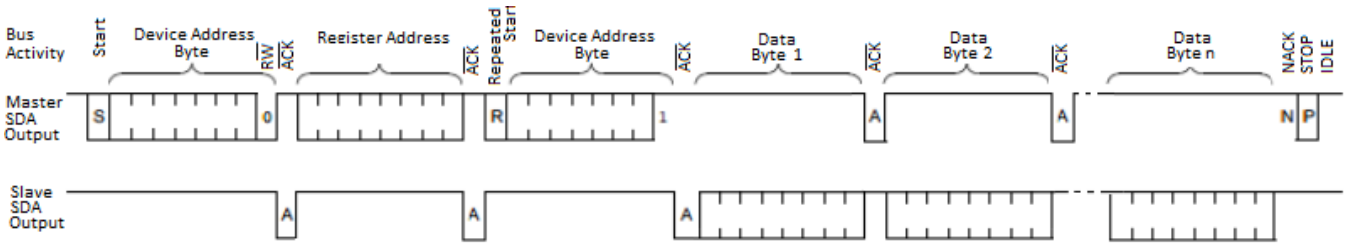
uint8_t I2C_0_ReadBlock_defined_addr(uint8_t dev_addr, uint16_t reg_addr, uint8_t reg_addr_nBytes, uint8_t *pData, uint8_t nBytes);

The function is used to read *nBytes* from defined *reg_addr* of the selected slave device.

For the polled mode it waits till all bytes are sent or an error appears.

For the interrupt mode the function just initiates the transmission of the first byte of the I2C read message, so should be called repeatedly until 0 is returned, meaning whole frame is transferred without errors.

The I2C frame then looks in the following way. An 8-bit register address is considered in this case.



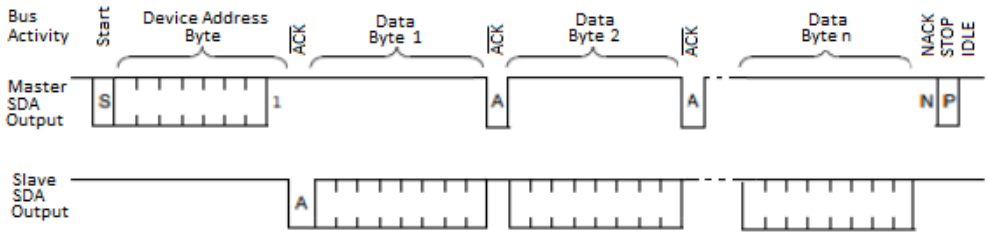
*uint8_t I2C_0_ReadBlock_preset_addr(uint8_t dev_addr, uint8_t *pData, uint8_t nBytes);*

The function is used to read *nBytes* from preset slave's register/memory address.

For the polled mode it waits till all bytes are sent or an error appears.

For the interrupt mode the function just initiates the transmission of the first byte of the I2C read message, so should be called repeatedly until 0 is returned, meaning whole frame is transferred without errors.

The I2C frame then looks in the following way.



void I2C_0_Init(uint8_t divider);

Performs initialization of the I2C module. Set transmission frequency based on given prescale divider. Consult device Reference Manual for the proper divider value.

If the I2C interrupt is enabled, assign I2C_0_Callback function to proper interrupt vector.

void I2C_0_Callback(void);

Follows RM's Flow-Chart of Typical I2C Interrupt Routine. Called from above three functions if SW poll mode is used or it should be assigned as interrupt routine if interrupt mode is used.

3. Other implementation hints

- Include a proper device header file within an I2C_0.h file

- If another I2C module should be used, the easiest way is to find all “I2C_0” occurrences in both *.c and *.h files and replace all with desired module name; e.g. “I2C_2”.

- I2C SDA and SCL pins have to be properly initialized. As both are bidirectional type, pins must be configured to select I2C functionality, enable output and input buffers and select open drain mode. Enable weak pull ups if there are no external pull ups connected.

- If switching between SW poll and interrupt mode, this must be always done when communication is already finished.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP and the NXP logo are trademarks of NXP B.V. All other product or service names are the property of their respective owners. All rights reserved.

Rev. 0
05/2016