AN2038
Application note

I2C emulation with ST7MC in slave mode

1 Introduction

This application note describes how to emulate I2C communication using an ST7MC as slave device. Since there is no dedicated I2C peripheral in the ST7MC microcontroller, the protocol is emulated using the SPI peripheral. This peripheral is common to different ST7 products so the software presented can be easily adapted to run in other ST7 microcontrollers. In this document, the hardware and software description refers to the ST7MC microcontroller for motor control applications. See *Section 7: Reference and related materials* to get more information about standard motor control libraries. *Section 4: Software modifications* describes the modification you need to make to implement I2C communication in the current standard libraries for specific types of motor.

Section 3: Hardware modifications describes the required modifications to the ST7MC Starter Kit board (Softec AK-ST7FMC) for implementing I2C communication.

The two modules presented in this application note are C modules (.h and .c file) developed to be compiled with Cosmic compiler v.4.5b (see www.cosmic-software.com for information and free download) and used in Softec STVD7 v.3.10 Integrated Development Environment (see www.softecmicro.com for information and free download). The first module, called **I2C module** implements the low level I2C protocol, the second module, called **ProtoSup module** is an example of a customized protocol based on I2C emulation. This protocol implements a Frame of Communication that allows the master to send a set of typical commands for a motor control application.

You should have a basic knowledge of C programming, motor control drives and basics of I2C protocol in order to use this module. In-depth know-how of ST7MC functions is only required for customizing existing modules or when adding new modules to develop your own application.

Contents

2 I2C emulation

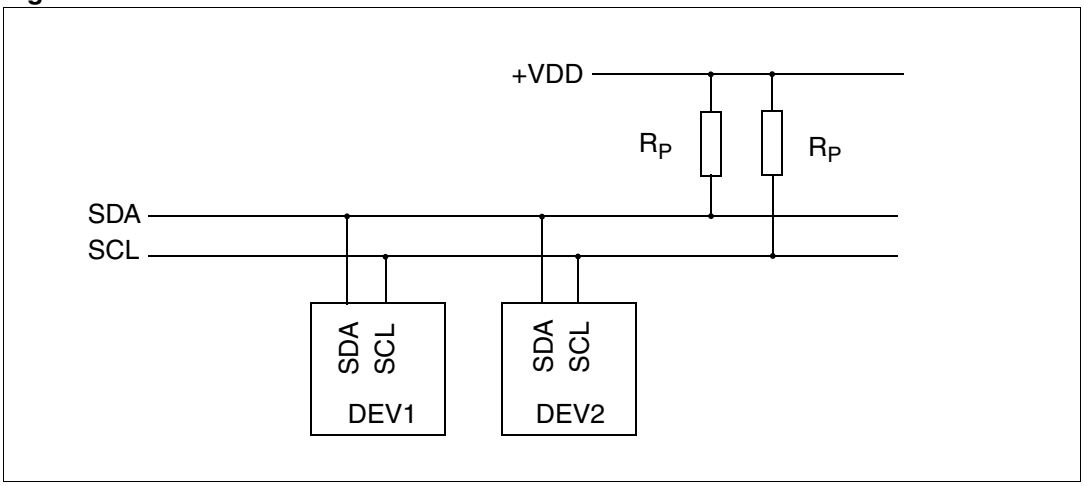
2.1 I2C basics

The I2C-bus has two wires, serial data (SDA) and serial clock (SCL), used to carry information between the devices. Each device is recognized by a unique address and can operate as either a transmitter or receiver, depending on the function of the device.

In addition to transmitters and receivers, devices can also be considered "masters" or "slaves" when performing data transfers. A master is the device which initiates a data transfer on the bus and generates the clock signals. At that time, any device addressed is considered a slave.

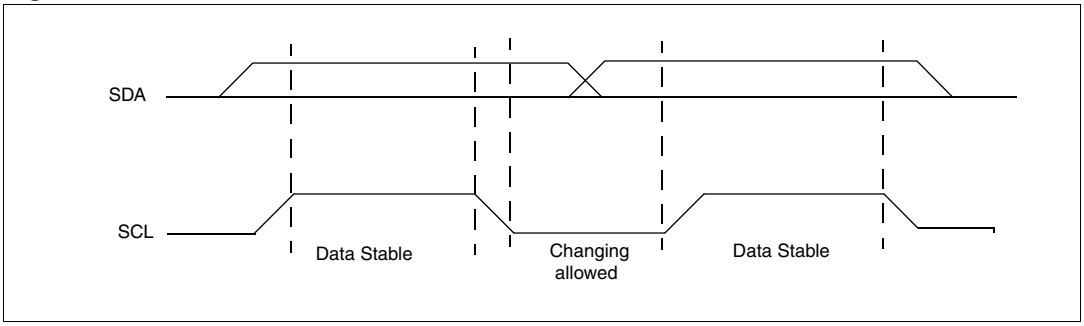
Both SDA and SCL are bi-directional lines, connected to a positive supply voltage via a pull-up resistor (see *Figure 1*). When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the AND function. Since data on the I2C-bus can be transferred at rates more than 100 kb/s in Standard-mode, in this application we reach a speed of 4 kb/s. This is caused by need to emulate the I2C with the SPI peripheral and the presence of a non-interruptible interrupt sub-routine for the motor control application.

Figure 1. I2C bus



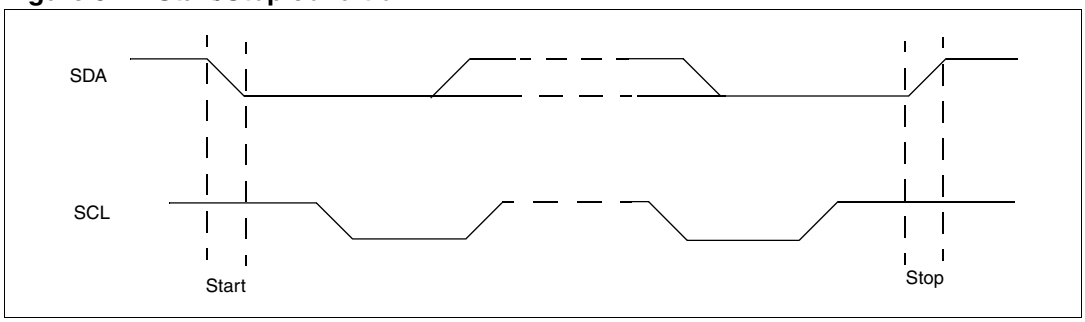
The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see *Figure 2*).

Figure 2. Data Valid



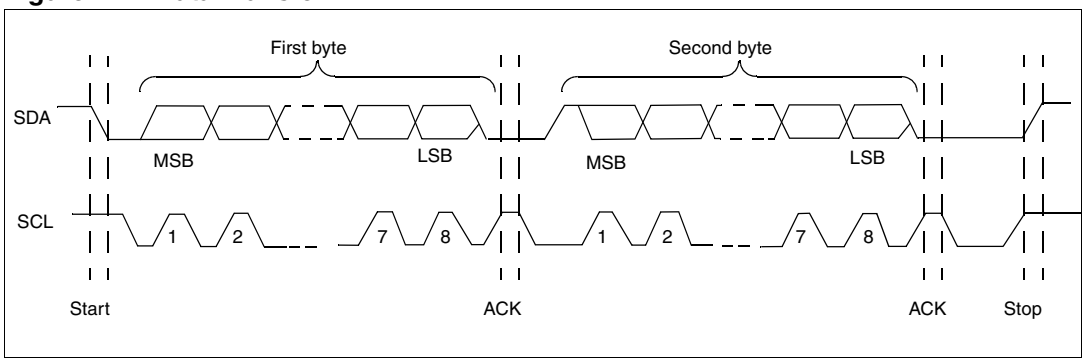
A HIGH to LOW transition on the SDA line while SCL is HIGH indicates a START condition. A LOW to HIGH transition on the SDA line while SCL is HIGH indicates a STOP condition (see *Figure 3*). START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition. The bus is considered to be free again after the STOP condition.

Figure 3. Start/Stop condition



Every byte put on the SDA line must be 8 bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte has to be followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first (see *Figure 4*).

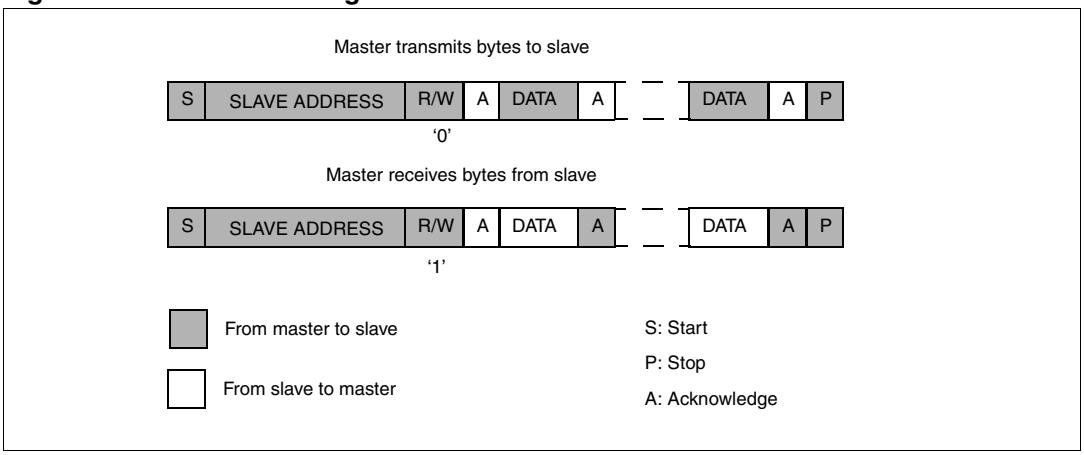
Figure 4. Data Transfer



The acknowledge clock pulse is generated by the master. The transmitter releases the SDA line (HIGH) during the acknowledge clock pulse and the receiver must pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse.

Data transfers follow the format shown in *Figure 5*: after the START condition (S), a slave address is sent. This address is 7 bits long followed by an eighth bit which is a data direction bit (R/W) - a 'zero' indicates a transmission (WRITE), a 'one' indicates a request for data (READ). A data transfer is always terminated by a STOP condition (P) generated by the master.

Figure 5. 7-bit addressing

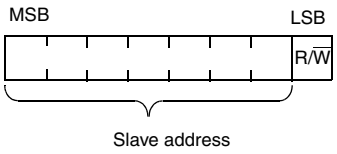


Possible data transfer formats are:

- Master-transmitter transmits to slave-receiver. The transfer direction is not changed (see *Figure 5* "Master transmit bytes to slave").
- Master reads slave immediately after first byte (see *Figure 5* "Master receive bytes from slave"). At the moment of the first acknowledge, the master transmitter becomes a master-receiver and the slave-receiver becomes a slave-transmitter. This first acknowledge is still generated by the slave. The STOP condition is generated by the master.

See *Figure 6* for the format of first byte after the Start condition.

Figure 6. First byte after the Start condition



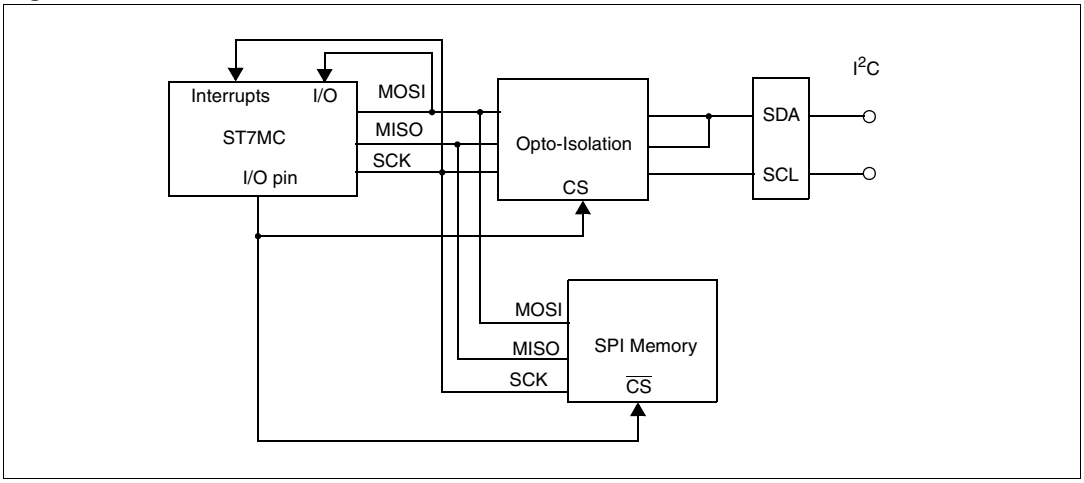
2.2 I2C architecture

In *Figure 7* shows the hardware architecture used to implement I2C emulation using the SPI peripheral.

The ST7MC is connected to the I2C bus through an optoisolator able to isolate the High voltage section from the bus.

By means of one I/O pin the ST7MC is able to select if the SPI bus is used in I2C emulation mode or in SPI mode for communication with the SPI Memory. In the first case the optoisolator is enabled and the SPI memory is disabled, in the second case the optoisolator is disabled so SPI and I2C are unconnected and SPI memory is enabled.

Figure 7. Hardware Architecture



The MOSI and MISO pins are unidirectional lines, the first from master to slave and the second from slave to master. Both are connected together after the optoisolator to become the SDA bidirectional line. The SCK line is driven by the master and is connected to SCL.

Two I/O interrupts are used to sense the edge of the SDA and SCL signal and to signal the various I2C events (Start condition, Stop condition, Acknowledge) to the microcontroller.

2.3 I2C Emulation

When emulating the I2C bus using SPI it is important to manage the following conditions:

- Detection of Start Condition
- Validation of Start Condition
- End of Byte Receive (SPI)
- End of Byte Transmission (SPI)
- Validation of the transmission detecting a master acknowledge
- Time out
- Detection of the Stop Condition

The detection of the start condition is performed using an interrupt port sensitive to the falling edge of the SDA signal. Each time a falling of SDA occurs together with SCL in high state, a Start Condition occurs.

A state machine is implemented to manage the Start Condition. The state machine has the following states:

Table 1. Start Condition management

State	Description
WAIT_START	In this state the micro is waiting for a Start Condition. After a falling edge occurs on SDA when SCL is high, the state is changed to START_VALIDATION
START_VALIDATION	In this state the micro is in waiting for a validation of Start Condition. The Start Condition is validated when the first faling edge on SCL occurs, in this case the state is changed to COMMUNICATION
COMMUNICATION	In this state the micro is exchanging data using the SPI

If a TIMEOUT occurs or a Stop condition is detected the state is forced to WAIT_START.

Each time a rising edge occurs on SDA occurs together with SCL in high state, a Stop Condition occurs. In this case a flag is set indicating that new data has been received or transmitted. This flag is used by the ProtoSup module.

When an SPI interrupt occurs a byte is received or transmitted.

If it is the first byte after the Start Condition, this byte will be analyzed to see if:

- The Slave is addressed, this is the case if the address of the slave is equal to the microcontroller I2C address.
- If the transmission is a write from the master or a read from the slave. If it is the former, the following bytes have to be read else the transmission buffer has to be transmitted using SPI. The corresponding status flag is therefore set.

If the byte received is not the first byte, it is stored in a read buffer.

If an end of transmission interrupt occurs, the microcontroller has to validate the transmission by detecting the master acknowledge.

The detection of master acknowledges will be performed in the SCL interrupt service routine on falling edge.

After the detection of the master acknowledge a new byte in the transmission buffer will be sent. If the buffer is empty, the condition is indicated setting a flag.

3 Hardware modifications

The following modifications should be performed on the Softec Starter Kit to allow the I2C emulation.

Figure 8 shows the I2C module board with the required optical isolation.

J1 is the connector interface to the I2C bus and J2 is the connector interface to the ST7MC Starter Kit.

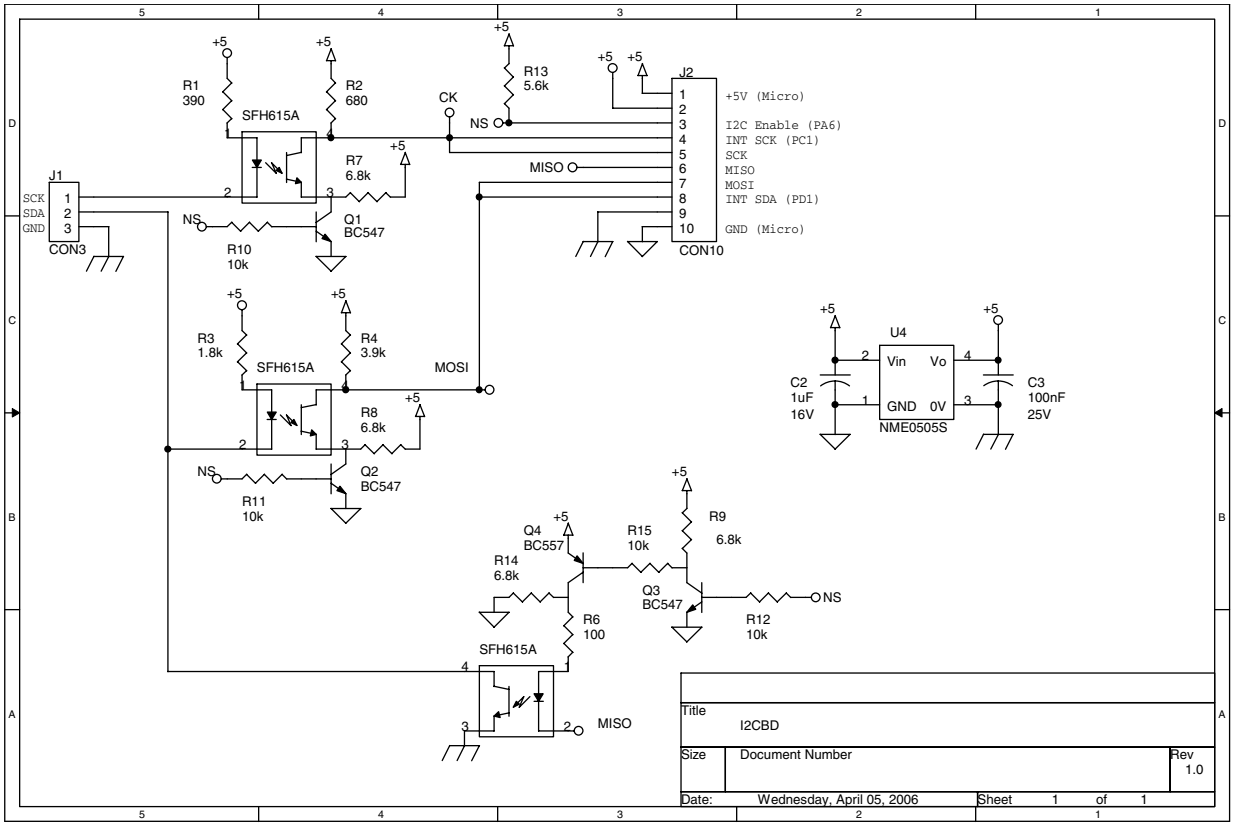
To use the module you must connect J2 to the Starter Kit pins. Since there is no free available pin on the board, you have to to solder the wires directly to the pins listed in *Table 2*.

Table 2. Hardware J2 connection to be soldered in ST7MC Starter Kit

J2 Pin	Starter Kit	J2 Pin	Starter Kit	J2 Pin	Starter Kit
1	U10 Pin 41			3	U10 Pin 20
4	U10 Pin 30	5	U10 Pin 27	6	U10 Pin 25
7	U10 Pin 26	8	U10 Pin 47		
10	U10 Pin 40				

Note: U10 is the ST7MC2 chip.

Figure 8. I2C module schematic



4 Software modifications

To implement the I2C emulation you have to perform the following modifications to the standard motor control software library.

4.1 Standard motor control libraries for ST7MC Starter Kit

At the moment the following standard libraries are released for the ST7MC Starter Kit (Motor Control application), each dedicated to specific kinds of motor. The libraries are listed in *Table 3*. See *Section 7: Reference and related materials* for details on the motor control libraries.

Table 3. Motor Control Standard Libraries

Library Name	Description
AC_1PH_SR_2.0	Induction mono phase motors sensored
AC_2PH_SR_2.0	Induction bi phases motors sensored
AC_3PH_SR_2.0	Induction three phases motor sensored
BLDC_3PH_SL_2.0	Brushless or permanent magnet motors drive in trapezoidal mode (6 Step) sensorless
BLDC_3PH_SR_2.0	Brushless or Permanent Magnet motors drive in trapezoidal mode (6 Step) sensored
PMAC_3PH_SR_2.0	Brushless or Permanent Magnet motors drive in sinusoidal mode sensored

The following steps are required to modify the standard libraries to emulate the I2C using SPI:

- First, configure the libraries using the configuration tools AK-ST7FMC Control Panel. Refer to the related manual.
- Copy the two provided modules (I2C.c, I2C.h, ProtoSup.c, ProtoSup.h) in the project folder.
- Copy the vector.c file provided into the project folder overwriting old vector.c file.
- Copy the main.c file provided into the project folder overwriting old main.c file.

- Open the workspace using Softec STVD7 v.3.10 Integrated Development Environment.
- Add the two files I2C.c and ProtoSup.c in the "Source Files" folder of the workspace and the two files I2C.h and ProtoSup.c in the "Include Files" folder of the workspace.

- Inside the I2C.h file select the kind of motor defining the correspondent MOTOR_TYPE constant PMAC, BLDC or AC:

  #define MOTOR_TYPE PMAC

  or

  #define MOTOR_TYPE BLDC

  or

  #define MOTOR_TYPE AC

- Interrupt priority setup:

  If the AC (or PMAC) motor has been selected, then modify the ST7_IntPrioritySetUp function defined in the st7_misc.c file:

  ISPR0 = MCES_LVL_3 + MCC_SI_LVL_1 + EXT_IT0_LVL_2 + EXT_IT1_LVL_1;

  ISPR1 = EXT_IT2_LVL_2 + MTC_U_CL_SO_LVL_3 + MTC_R_Z_LVL_2 + MTC_C_D_LVL_2;

  ISPR2 = SPI_LVL_2 + TIMER_A_LVL_2 + TIMER_B_LVL_1 + SCI_LVL_1;

  If the BLDC motor has been selected, then modify the ST7_IntPrioritySetUp function defined in the misc.c file:

  ITSPR0 = MCES_LVL_3 + MCC_SI_LVL_1 + EXT_IT0_LVL_2 + EXT_IT1_LVL_1;

  ITSPR1 = EXT_IT2_LVL_2 + MTC_U_CL_SO_LVL_3 + MTC_R_Z_LVL_3 + MTC_C_D_LVL_3;

  ITSPR2 = SPI_LVL_2 + TIMER_A_LVL_2 + TIMER_B_LVL_3 + SCI_LVL_1;

  ITSPR3 = (AVD_LVL_1 + PWMART_LVL_2) | (u8)(0xf0);

Modification related to PMAC Motor

- Modify the PMACparam.h:
- Modify the define of target frequency

  #define TARGET_FREQ_CL 1500

  in

  extern u16 TARGET_FREQ_CL;

Modification related to BLDC Motor

- Modify the mtc.h:

  #define MotorStalled 4

- Inside the mtc.c for BLDC motor
  - Add the following include at the beginning

    #include "I2C.h"

    #include "Protosup.h"
  - Then is necessary to modify the Chk_Motor_Stalled() and MTC_C_D_IT() functions to manage the fault condition.

  Inside Chk_Motor_Stalled() function add the following

```
// Frequency Synchronization
if ((MPRSR & 0x0F) == RATIO_MAX)
   {
      COMMAND = CMD_BRAKE;
      MotorStatus = MOTOR_STALLED;
....
   }
```

  Only for BLDC sensorless, inside MTC_C_D_IT() function find the following block and modify as written.

```
if (RampIndex >= (RAMP_SIZE-1))// If Ramp is finished without success
{
........
      MotorStatus |= START_UP_FAILED;
      COMMAND = CMD_BRAKE;
   ..........
 }
```

5 Command frames

The commands are sent from the master to the slave and are organized as shown in *Figure 9*.

After the Start condition, the first byte addresses the slave and forces a w to send the command to the slave.

STX is a byte that indicates a Start of Transmission and is coded in *Table 4.*

Then is dispatched a byte with the length (n) of the sequence of bytes that will be sent.

D1 is the "Command" byte see *Table 4.* D2 - Dn is an optional sequence of bytes; this is a additional parameter sent together with the command.

The next two bytes CRC1 and CRC2 is the checksum calculated with the following rule:

CRC calculation

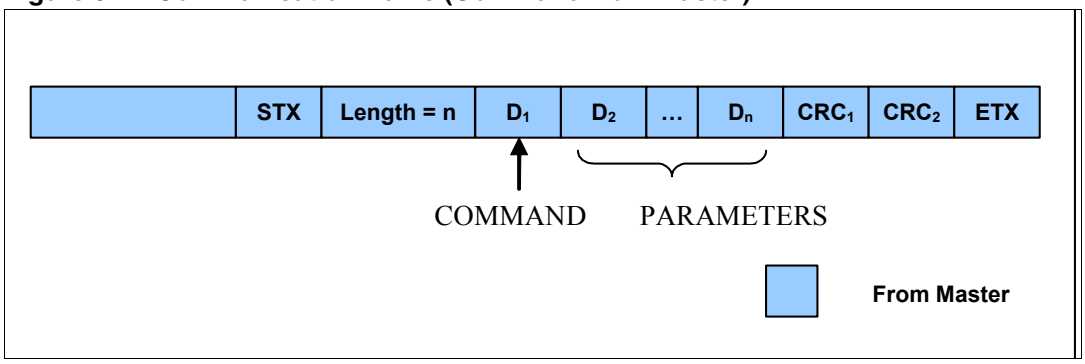
CRC value is a 16-bit value that contains the sum of D1, D2, … , Dn.

CRC_1 is the MSB of CRC value.

CRC_2 is the LSB of CRC value.

The last byte is ETX that indicates the End of Transmission.

Figure 9. Communication frame (Command from Master)



As command the master can ask a request from the slave. The next frame is organized as show in *Figure 10*.

After the ask command of the previous frame the master start a new communication and in the first byte after the Start Condition it address the slave from which it wait the answer and force a r.

The slave sent the following byte:

STX is a byte that indicates a Start of Transmission and is coded in the *Table 4*.

After that it is transmitted a byte with the length (n) of the sequence of bytes that will be sent.

D_1 - D_n is the answer from the slave.

The next two byte CRC_1 and CRC_2 is the checksum calculated with the above described rule.

The last byte is ETX that indicate the End of Transmission.

Figure 10. Communication frame (Response from Slave)

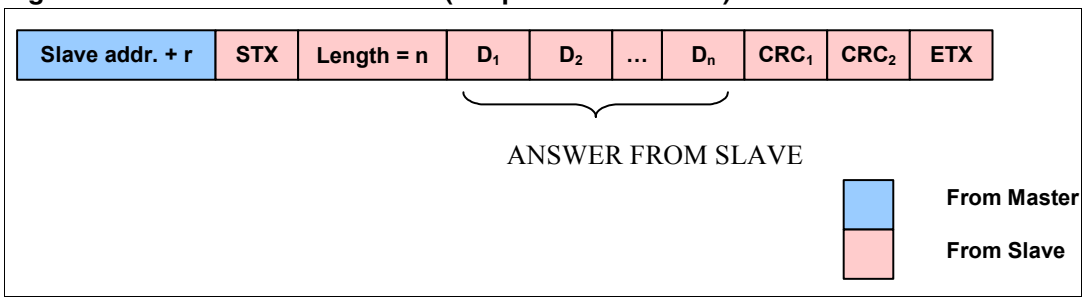


Table 4. I2C Code table

COMMUNICATION CODE		
STX	0x02	Start Transmission
ETX	0x03	End Transmission
ACK	0x1c	Acknowledge
NOT_ACK	0xee	Not Acknowledge
NOT_READY	0xcc	Not Ready
WAIT	0xfc	Wait
MASTER COMMANDS		
CMD_SETSPEED	0x77	Command set speed. D2 and D3 will be respectively MSB and LSB of a word contains speed in rpm.
CMD_BRAKE	0x88	Command brake
CMD_START	0x66	Command start motor
MASTER QUERY		
ASK_STATUS	0x08	Master asks for motor status. D1 and D2 of subsequent answer of the slave is a byte formatted as below. D2 is a checksum byte equal to complemented bit respect D1
ASK_SPEED	0x07	Master asks for motor speed. D1, D2, D3, D4 of subsequent answer of the slave is respectively MSB and LSB of a word contains speed in rpm. D3 and D4 is two checksum byte equal to complemented bit respect D1 and D2

Motor Status Byte

0	0	0	MS	SF	OV	OC	OT

MS: Motor Stalled

SF: Startup Fails

OT: Over Temperature

OV: Over Voltage

OC: Over Current

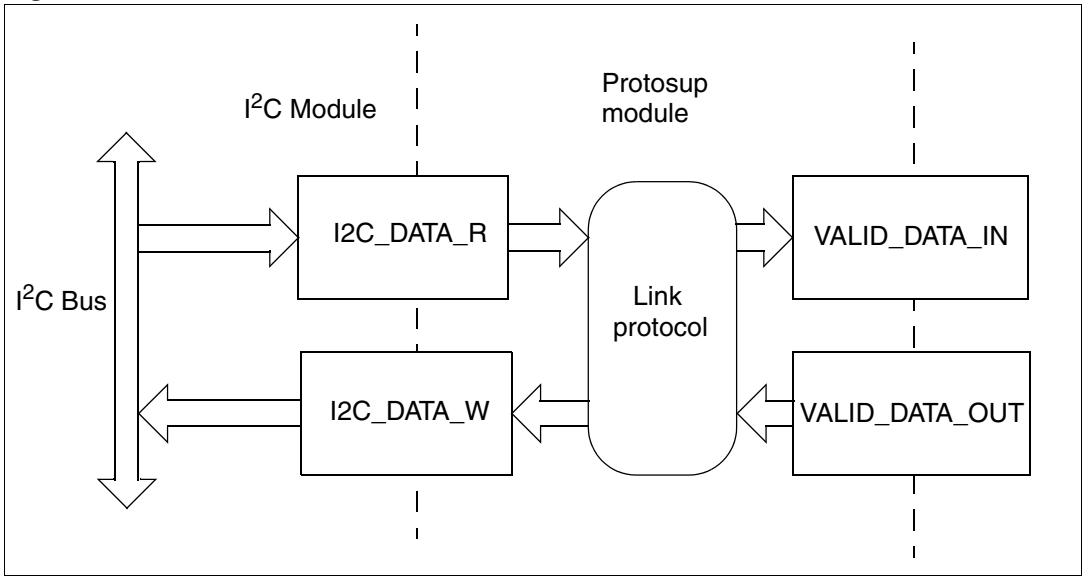
6 Function description

The two I2C modules are organized as shown in *Figure 11*.

I2C Module function is to get or put the raw data from/to the I2C bus into two data buffer I2C_DATA_R and I2C_DATA_W.

ProtoSup Module function is to extract the valid data from the stream coming from I2C_DATA_R buffer into the VALID_DATA_IN and from VALID_DATA_OUT buffer into I2C_DATA_W buffer.

Figure 11. Firmware architecture



6.1 I2C Module

This module contains the low level routine to emulate the I2C communication protocol.

Initialization Routine

Function Name	Input Parameter	Return Value
Init_I2C_Protocol	void	void
Description		
I2C module initialization. This function should be called inside the main, before the main loop, to initialize the I2C emulation. This function calls the following initialization function: – Init_SPI_4_I2C – Init_TIMER_4_I2C – Init_INT_SCK_4_I2C – Init_INT_SDA_4_I2C This function also set the status of the I2C state machine to WAIT_START, it means that the micro is in waiting of a Start Condition. The I2C Slave Acknowledge flag is set false (No ACK from slave in progress). This function also set the PA6 as output push pull and with an high value to select SPI bus for I2C communication.		

Function Name	Input Parameter	Return Value
Init_SPI_4_I2C	void	void
Description		
SPI peripheral initialization. (This function is called by Init_I2C_Protocol) This function performs first the initialization of the I/O of SPI pins calling "Init_SPI_Ports" routine. Then it clears the contingent pending SPI interrupt. Then enable SPI interrupt, disable alternate function that will be enabled further, sets SPI clock edge as rising edge and sets the slave mode.		

Function Name	Input Parameter	Return Value
Init_TIMER_4_I2C	void	void
Description		
Timer (A) initialization. (This function is called by Init_I2C_Protocol) Initialization of Timer A used for Time Out purpose. TIMEOUT defines the number of microseconds it is defined in I2C.h. The timer is freeze and will be started using Start Timer when the Start Condition occurs. After the time out the ISR "TIMER_A_Interrupt_Routine" will be served.		

Function Name	Input Parameter	Return Value
Init_INT_SCK_4_I2C	void	void
Description		
Initialize the external interrupt SCK (ei2 PC1). (This function is called by Init_I2C_Protocol) This function sets PC1 as floating interrupt sensitive to falling edge.		

Function Name	Input Parameter	Return Value
Init_INT_SDA_4_I2C	void	void
Description		
Initialize the external interrupt SDA (ei0 PD1). (This function is called by Init_I2C_Protocol) This function sets PD1 as floating interrupt sensitive to rising and falling edge.		

Low level routine

Function Name	Input Parameter	Return Value
I2C_enable_SPI	void	void
Description		
This function enables the SPI communication. It is enabled after a Start condition and falling of SCK. It clear also the pending bit.		

Function Name	Input Parameter	Return Value
I2C_disable_SPI	void	void
Description		
This function disables the SPI communication. The ports is dedicated to sensing the edge transition using port interrupts.		

Function Name	Input Parameter	Return Value
I2C_Stop_detected	void	void
Description		
This function is called when a Stop condition is detected. It disables the SPI peripheral and reinitialize SDA and SCK interrupts. If the communication is from master to slave (I2C_SLAVE_READ_flag set true) the flag New_Data_Receved is set to true for high level protocol purposes (ProtoSup), Otherwise if the communication is from slave to master (I2C_SLAVE_WRITE_flag set true) the flag New_Data_Sent is set to true.		

Function Name	Input Parameter	Return Value
Reset_TIMER	void	void
Description		
Reset and Freeze the timer. It is called after a validation of Start Condition.		

Function Name	Input Parameter	Return Value
Start_TIMER	void	void
Description		
UnFreeze the Timer. It is called when a falling of SDA occurs when SCL is high (Start Condition)		

Interrupt Service Routine

Function Name	Input Parameter	Return Value
TIMER_A_Interrupt_Routine	void	void
Description		
Timeout interrupt service routine. It is used to reset the state if a Start Condition is not validated. The ISR will be executed TIMEOUT microseconds (defined in I2C.h) after the "Start_TIMER" command; usually the timer start after a falling of SDA while SCL is high (Start Condition event). If TIMEOUT occurs without a valid clock impulse the status came back to WAIT_START and the timer will be reinitialized (freeze) in this case the Start Condition is not validated.		

Function Name	Input Parameter	Return Value
SPI_Interrupt_Routine	void	void
Description		
ISR executed each SPI end of transmission. If the received byte is the first byte after the Start Condition, the slave check if it is addressed (the SLAVE_ADDRESS is defined in the I2C.h). If it is addressed, acknowledge will be sent, in the same time the flags SLAVE_READ or SLAVE_WRITE will be set according the 8th bit (R/W bit). Otherwise, if SLAVE is reading (SLAVE_READ is true) the data is stored in I2C_Datas_R and acknowledge will be sent; if SLAVE is writing (SLAVE_WRITE is true) set SLAVE_WAIT_MSACK flag. SPI is disabled and the micro is waiting for the next clock to validate the master acknowledge.		

Function Name	Input Parameter	Return Value
INT_SDA_I2C	void	void
Description		
ISR generated on each edge of SDA. It is used to detect Start or Stop condition. If a falling edge of SDA occurs when SCL is high a Start Condition (to be validated) occurs. In this case the I2C_First_Byte will be set to indicate that the next byte sent is the first byte after Start Condition, the state is changed in START_VALIDATION and the timer is started calling "Start_TIMER" function. If a rising edge of SDA occurs when SCL is high a Stop Condition occurs. In this case the state is changed in WAIT_START and the "I2C_Stop_detected" function is called.		

Function Name	Input Parameter	Return Value
INT_SCK_I2C	void	void
Description		
ISR generated on each falling edge of the clock. It is used for three purpose: – If status is STARTING_VALIDATION it will be validated and the status is set to COMMUNICATION. – If the slave is acknowledging (I2C_Slave_Ack true) it stop the Slave acknowledge. – If the slave is in waiting of master acknowledge here it is checked and the next byte in I2C_Datas_W will be sent.		

6.2 PROTOSUP module

This module contains the high level routine to dispatch the commands frame.

Function Name	Input Parameter	Return Value
Test_Check_sum	void	True or False
Description		
This function returns true is the checksum received in the read buffer is coherent with the data received, otherwise return false.		

Function Name	Input Parameter	Return Value
Return_CRC	Pointer to data buffer	CRC calculated
Description		
This function returns the CRC of the data present in the buffer. The CRC is the sum of the byte in the buffer.		

Function Name	Input Parameter	Return Value
New_Cde_Detected	void	void
Description		
This function writes in the COMMAND global variable the value of the command sent by the master (D1)		

Function Name	Input Parameter	Return Value
Not_Ready_Function	void	void
Description		
This function writes the output buffer with a frame: STX, 1, NOT_READY, ETX, CRC1, CRC2		

Function Name	Input Parameter	Return Value
Not_ACK_Function	void	void
Description		
This function writes the output buffer with a frame: STX, 1, NOT_ACK, ETX, CRC1, CRC2		

Function Name	Input Parameter	Return Value
ACK_Function	void	void
Description		
This function writes the output buffer with a frame: STX, 1, ACK, ETX, CRC1, CRC2		

Function Name	Input Parameter	Return Value
Master_Querry	void	True or False
Description		
This function returns true if the received command frame is a query of the master (Request of information to the slave, for example the current speed) otherwise it returns false.		

Function Name	Input Parameter	Return Value
Querries_of_Master	void	void
Description		
This function fills the Data_Out buffer according to the master's request.		

Function Name	Input Parameter	Return Value
Test_STX_Presence	void	True or False
Description		
This function returns true if STX byte is present in the read buffer, otherwise it returns false.		

Function Name	Input Parameter	Return Value
Fill_Data_In	void	void
Description		
This function fills the Data_In buffer with the byte present in the read buffer I2C_Datas_R and test the ETX presence.		

Function Name	Input Parameter	Return Value
Fill_Data_valid	void	void
Description		
This function extracts the D1, D2, Dn byte from Data_In buffer and put them into Valid_Data_In buffer.		

Function Name	Input Parameter	Return Value
LINK_protocol	void	void
Description		
This function performs the dispatching of the commands. This function should be called inside the main loop of the application. If new data is received it tests the presence of STX calling "Test_ETX_Presence", if STX is present the Checksum is tested calling "Test_Check_sum()", if the Checksum is correct the following calling will be performed: "ACK_Function" is called "Fill_Data_valid" is called "New_Cde_Detected" is called and if "Master_Querry" calls return true: "Querries_of_Master" is called.		

7 Reference and related materials

In the following AN it is possible to find information about standard motor control libraries:

- AN1904: STMC AC three-phase induction motor control library
- AN1905: STMC brushless permanent magnet DC motor control library
- AN1947: STMC brushless permanent magnet AC sinus motor control library

It is possible to find additional information in the following website:

- Microcontroller at st: mcu.st.com/mcu/
- ST7MC product information: mcu.st.com/mcu/ > downloads
- Forum: mcu.st.com/mcu/ > forum

Datasheet:

- See ST7MC datasheet

Information about Softec ST7MC Starter Kit board:

- See www.softecmicro.com web site

8 Revision history

Table 5. Document revision history

Date	Revision	Changes
13-June-2006	1	Initial release.