APPLICATION NOTE

Atmel AT02260: Driving AT42QT1085

Atmel QTouch

Features

- Overview of Atmel® AT42QT1085
- Circuit configuration with Host MCU
- SPI communication
- Demonstration program

Description

This application note explains the communication of Master SPI controller with AT42QT1085 as a slave device. It demonstrates configuring and controlling various parameters of this device.

The host code demonstration program provided has been developed for 8-bit Atmel megaAVR® (Atmel ATmega2560) microcontroller but can be easily adapted for other platforms.

The demonstration program is written in C and supports both GCC (Atmel Studio) and IAR™ (IAR Embedded Workbench®) compiler.

Table of Contents

1. Overview of Atmel AT42QT1085

1.1 Introduction

AT42QT1085 is a device based on the Atmel QTouchADC technology designed for capacitive touch key applications.

This device supports up to eight keys, where four are dedicated channels for keys (Key0 to Key3) and another four channels can be configured either as keys (Key4 to Key7) or as GPIO channels (GPIO_12 to GPIO _15). This device also supports 12 dedicated GPIO channels (GPIO_0 to GPIO_11).

This device has a haptics engine integrated in it, which allows haptic effects to be triggered on key detection or directly controlled by the host microcontroller.

1.2 Host interface

The host microcontroller communicates with the AT42QT1085 over SPI interface using master-slave relationship, with the AT42QT1085 acting in slave mode. This bus protocol takes care of all addressing functions and bidirectional data transfer.

In addition, the device also features a $\overline{\text{CHANGE}}$ signal, which is asserted when there is a message waiting to be read. This can be used either to wake the host or as an interrupt signal to inform the host when the QT™ device has detected a touch. The host should always use $\overline{\text{CHANGE}}$ line as an indication to read messages from QT device and the host should not read messages at any other time, when $\overline{\text{CHANGE}}$ line is not asserted.

2. Circuit Configuration with Host Microcontroller

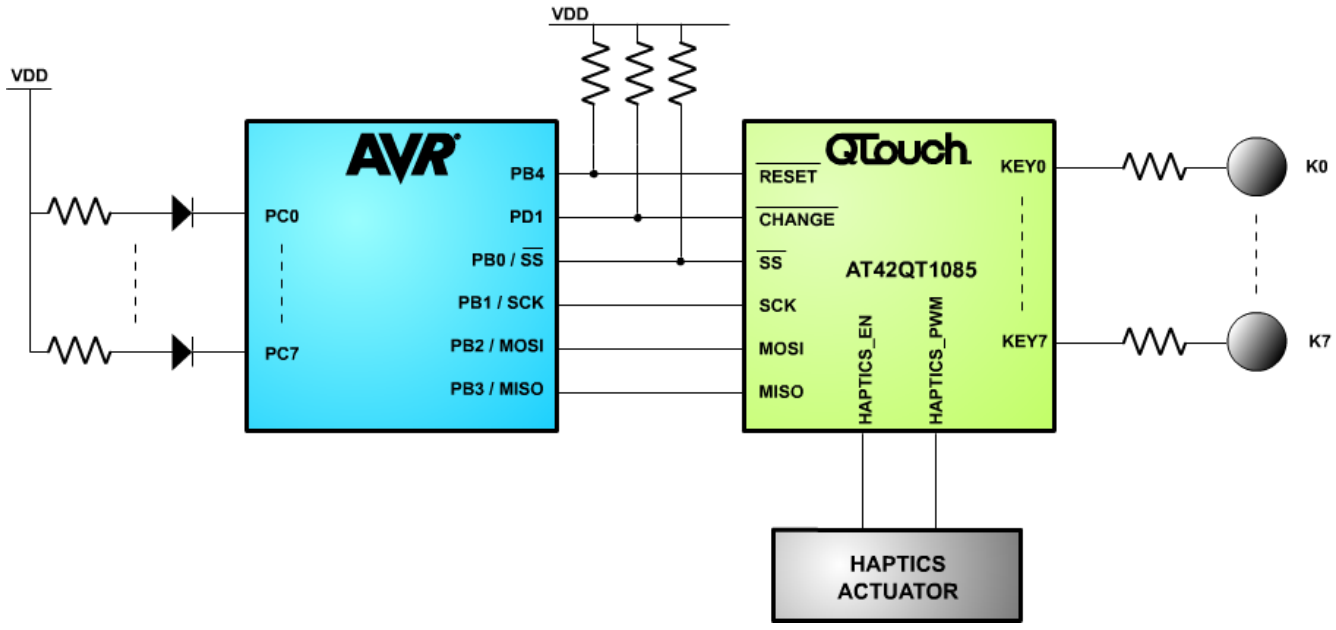
Following are the connections used in the demonstration program.

Table 2-1. Connection between Host microcontroller and QT device.

Atmel ATmega2560	Atmel AT42QT1085
PD1 (Pin 44)	$\overline{\text{CHANGE}}$
PB4 (Pin 23)	$\overline{\text{RESET}}$
PB0 (Pin 19)	$\overline{\text{SS}}$
PB1 (Pin 20)	SCK
PB2 (Pin 21)	MOSI
PB3 (Pin 22)	MISO

The hardware SPI module of the ATmega2560 is used in this demonstration. The SPI module of ATmega2560 is available in the corresponding pins mentioned in the Table 2-1.

Figure 2-1. Circuit configuration with Host microcontroller.



Touch keys (K0 – K7) are connected to the KEY0 to KEY7 sense pins of AT42QT1085.

In the demonstration source code provided, PORT C (PC0 to PC7) of the Host microcontroller has been configured for touch status indication on keys K0 to K7 using LEDs.

Note: External pull-up resistors of 100kΩ are required on the $\overline{\text{SS}}$, $\overline{\text{RESET}}$ and the $\overline{\text{CHANGE}}$ lines.

3. SPI Communication

3.1 Specifications for the Atmel AT42QT1085

The AT42QT1085 communicates with the host over a full-duplex 4-wire (MISO, MOSI, SCK, $\overline{SS}$) SPI interface. The AT42QT1085 supports an object-based protocol that is used to communicate with the device. There are a few specifications with regards to AT42QT1085 communication, which are mentioned below.

- The AT42QT1085 SPI Interface can operate at a speed of up to 750kHz
- The Least Significant Bit (LSB) is the first byte in a multi-byte data transmission (Little-Endian Configuration)
- The AT42QT1085 set up data on the rising edge and replace data on the falling edge
- The AT42QT1085 require that the clock idles to “high” state
- In AT42QT1085 a minimum delay of 100µs is required between transmissions of each byte in case of multi-byte communication
- To begin a new communication exchange with AT42QT1085, $\overline{SS}$ must be pulled high for at least 2ms after performing a read operation or 10ms after performing a write operation

Note: Refer to the AT42QT1085 device datasheet for other timing specifications.

3.2 SPI driver implementation

The SPI driver in this demonstration program is developed for the Atmel ATmega2560. SPI Peripheral for this device is available in PORTB. Refer to Table 2-1 and Figure 2-1 for details.

Following are the functions used to implement the SPI driver:

Function	Description	
`void SPI_MasterInit (uint8_t sck_fosc_div)`	Initializes the SPI module of the Host microcontroller	

	Input	`sck_fosc_div` - Division Factor to generate SCK Frequency. The following macros are the allowable inputs as arguments
		`SCK_FOSC_DIV_2`
		`SCK_FOSC_DIV_4`
		`SCK_FOSC_DIV_8`
		`SCK_FOSC_DIV_16`
		`SCK_FOSC_DIV_32`
		`SCK_FOSC_DIV_64`
		`SCK_FOSC_DIV_128`

	Output	None
`void SPI_Transfer (uint8_t *command, uint8_t command_length, uint8_t *data, uint8_t data_length)`	Sends command and simultaneously transmits and receives the number of data bytes specified	
	Input	*command – Pointer to command array command_length – Number of command bytes to be transmitted by master *data – Pointer to local array to send data from host data_length – Number of data bytes to be written by master or read back from the slave
	Output	*data – Pointer to local array to read data from QT device

3.3 SPI Communication Protocol for the Atmel AT42QT1085

The Object Protocol of the AT42QT1085 device organizes different features of the device into objects that can be controlled individually. The Objects must be configured before use and the modified settings need to be written into the nonvolatile memory using the Command Processor object.

Using the Object Protocol

The host should perform the following initialization to communicate with the AT42QT1085 using SPI:

a. Read the start positions and sizes of all the objects from the Object Table in the AT42QT1085 and build up a list of all these addresses.
b. Use the Object Table to calculate the report IDs so that messages from the QT device can be correctly interpreted.

Host Command

A 3-byte command sequence is transmitted by the host on MOSI, setting the memory map address pointer, a Read / Write indication, and the number of bytes which will be read or written.

Reading from / Writing to the device

The SPI Transfer routine sends the appropriate command to the QT device for reading or writing.

This routine receives a pointer to command array and length of the command array as inputs. The command array byte indicates read or write operation.

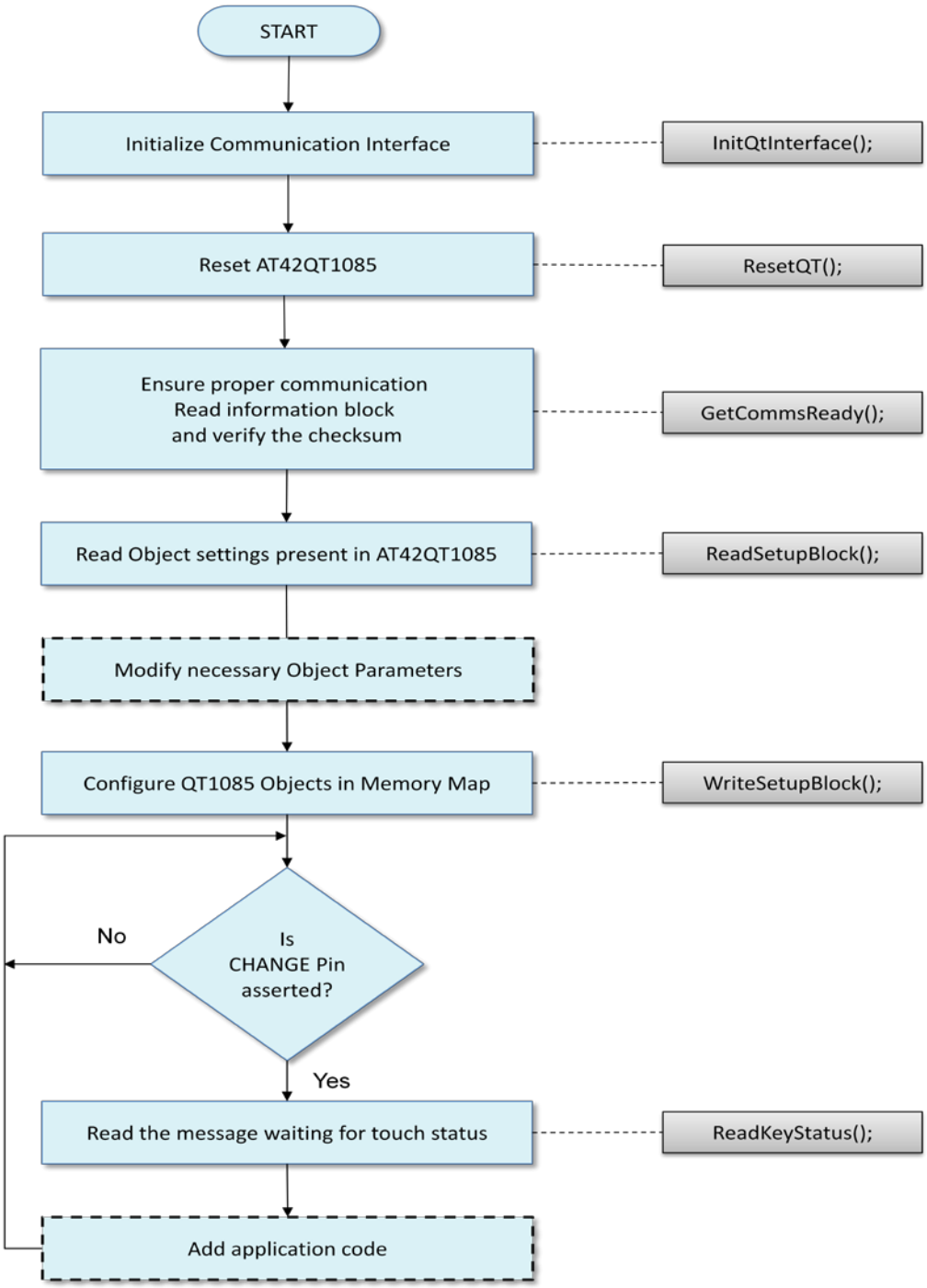
This routine also receives a pointer to local array and number of data bytes to be read from the QT device or written into the QT device. The local array is dumped with the received SPI data during read operation and during write operation, the host controller fills the local array with data to be written into the QT device.

Note: Refer to the AT42QT1085 Protocol Guide for detailed information on configuring objects.

4. Demonstration Program

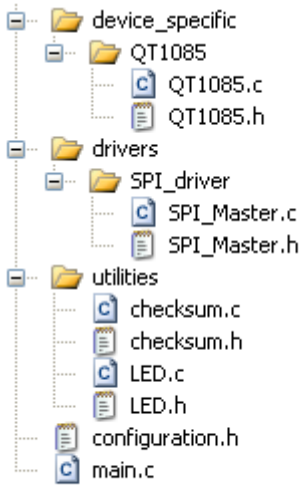
The demonstration program shows how to use the host interface to read real-time touch information from QT devices. It also demonstrates the procedure to perform read and write operations into objects on the Object Protocol memory map structure. This helps in tuning different operating parameters of the device.

4.1 Program flow



4.2 Files

The folder structure for the demonstration program is shown below.



The source code consists of the following files:

File name	Description
main.c	It consists the main() function and the body of the application program
configuration.h	Device selection and port pin selections for $\overline{\text{RESET}}$ and $\overline{\text{CHANGE}}$ pins are configured here
QT1085.c	It consists the application interfaces to drive the QT device
QT1085.h	Consists of function prototypes and memory map structure for object protocol
SPI_Master.c	SPI driver code
SPI_Master.h	Header file for the SPI driver
checksum.c	Contains algorithm for 24-bit CRC checksum calculation and verifies the information block checksum read
checksum.h	Header file for CRC computation
LED.c	Handling LED update based on touch key status
LED.h	Header file for Host Control source file

4.3 Functions

Function	Description	
`void InitQtInterface(void)`	Performs SPI Master initialization, configures a selected GPIO as input for $\overline{\text{CHANGE}}$ pin and configures a GPIO as output for $\overline{\text{RESET}}$ pin	
	Input	None
	Output Return	None None
`void ResetQT(void)`	Performs a hardware reset of the QT device by pulling down the RESET pin of the QT device	
	Input	None
	Output Return	None None
`void GetCommsReady(void)`	Uses the SPI command to ensure proper communication, by verifying information block checksum read	
	Input	None
	Output Return	None None
`void ReadObjectTable(void)`	Reads the information block bytes of the QT device	
	Input	None
	Output Return	None None
`uint32_t InformationBlockCRC (void)`	Computes CRC for the information block bytes	
	Input	None
	Output Return	None Returns the computed CRC
`Uint8_t VerifyInfoBlockChecksum (uint32_t checkcum_calc)`	Verifies the computed CRC of the information block	
	Input	`checkcum_calc` - computed CRC of the information block
	Output Return	None Returns TRUE if successful or FALSE otherwise
`uint8_t ReadSetupBlock (struct QT1085_objects_config_t *objects_config_ptr)`	Reads the configurable Object parameters from memory map structure	
	Input	`*objects_config_ptr`: Pointer to configuration objects structure
	Output Return	`*objects_config_ptr`: Pointer to configuration objects structure, Configured data filled in objects_config structure Returns TRUE if successful or FALSE otherwise
`struct object_table_t* GetObjectInfo (uint8_t type)`	Determines the starting address of the requested Object table element in memory map structure, which matches with the type input	
	Input	`type` - type code for the object details requested
	Output Return	None Returns the starting address of the object table element
`uint8_t WriteSetupBlock (struct QT1085_objects_config_t *objects_config_ptr)`	Writes the modified Object parameters into memory map structure	
	Input	`*objects_config_ptr`: Pointer to configuration objects structure
	Output Return	None Returns TRUE if successful or FALSE otherwise

`void BackupConfigSetting(void)`	Sends command to backup configuration settings in nonvolatile memory	
	Input	None
	Output Return	None None
`uint8_t ReadKeyStatus (uint8_t *msg_ptr)`	Reads the message waiting to be read from QT device	
	Input	`*msg_ptr`: Pointer to message status array
	Output Return	None Returns TRUE if there is a touch status message, else FALSE
`uint8_t ReadMessageProcessorData (uint8_t *msg_ptr)`	Reads the object generated message from message processor	
	Input	`*msg_ptr`: Pointer to message status array
	Output Return	None Returns the report ID of the object that sent the message
`uint8_t ConfigurationCheck(void)`	Checks whether the Configuration Error Bit is set in received message	
	Input	None
	Output Return	None Returns the Configuration Error bit status
`void InitTouchStatusPorts(void)`	Configure the PORTC pins 0 to 7 for displaying touch status of Keys 0 to 7	
	Input	None
	Output Return	None None
`void UpdateLedStatus (uint8_t * QtStatusPtr)`	Update touch key status through LED indications	
	Input	`*QtStatusPtr`: Pointer to QT Touch status message array
	Output Return	None None

5. Porting Code to Other Platforms

This chapter discusses the parts of the demonstration program which needs modification while porting to other MCUs.

5.1 Change pin

The $\overline{\text{CHANGE}}$ pin of the QT device must be connected to a MCU pin which can be configured as an input.

To assign any particular pin the following MACROs declared in configuration.h needs to be modified.

```
// CHANGE Status port and pin configuration
#define CHANGE_STATUS_PORT    D       // PORT
#define CHANGE_STATUS_PIN     1      // Pin Number
```

5.2 Reset pin

The $\overline{\text{RESET}}$ pin of the QT device must be connected to a MCU pin which can be configured as an output.

To assign any particular pin the following MACROs declared in configuration.h needs to be modified.

```
// RESET port and pin configuration
#define RESET_PORT B     // PORT
#define RESET_PIN  4     // Pin Number
```

5.3 SPI driver

The device level drivers for SPI communication will be specific to the MCU platform used. The details of the SPI driver implementation has been provided in Section 3.2.

5.3.1 Pins for SPI communication

Pins for the SPI communication are configured in SPI_Master.h file. For porting to any other Atmel megaAVR or Atmel tinyAVR® one can simply configure the SPI pins as per the device datasheet. In the current demonstration program the following configuration has been made.

```
#define SPI_PORT PORTB
#define SPI_DDR  DDRB

#define SPI_SS   PB0
#define SPI_SCK  PB1
#define SPI_MOSI PB2
#define SPI_MISO PB3
```

5.3.2 SPI Initialization

The initialization routine for SPI master module must be done in `void SPI_MasterInit(uint_8 sck_fosc_div)`.

The implementation can be made for a fixed SCK frequency rather than a configurable one. The SCK frequency should not exceed the maximum speed supported by the QT device.

5.3.3 SPI data transfer

The data transfer routine should be able to handle bidirectional data transfer. This routine sends three command bytes and simultaneously transmits and receives the number of data bytes specified.

This routine should receive a pointer to command array, length of the command array as inputs, a pointer to local array and number of data bytes to be read from QT device or written into QT device.

6. References

- AT42QT1085 Protocol Guide Complete
- AT42QT1085 Complete device datasheet

7. Revision History

Doc. Rev.	Date	Comments
42092A	03/2013	Initial document release

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370

© 2013 Atmel Corporation. All rights reserved. / Rev.: 42092A–AVR–03/2013

Atmel®, Atmel logo and combinations thereof, AVR®, Enabling Unlimited Possibilities®, megaAVR®, QTouch®, tinyAVR®, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.