APPLICATION NOTE

Atmel AVR3003: Driving AT42QT1110 and AT42QT1111

Atmel QTouch

Features

- Overview of the Atmel® AT42QT1110 and Atmel AT42QT1111
- Circuit Configuration with Host MCU
- SPI Communication
- Demonstration Program

Introduction

This application note explains the communication of Master SPI controller with AT42QT1110 or AT42QT1111 as a slave device. It demonstrates configuring and controlling various parameters of these devices.

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

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

Table of Contents

1. Overview of the Atmel AT42QT1110 and AT42QT1111

1.1 Introduction

The AT42QT1110 and AT42QT1111 are devices based on the Atmel QTouch® technology designed for capacitive touch key applications. Both the devices are designed to work in two configurations:

- 7 – Key Mode

  The device can support up to seven keys and drive matching Detect Outputs to a user configurable PWM.
- 11 – Key Mode

  The device can support up to 11 keys and can send key status information over SPI interface to a host microcontroller.

1.2 Host Interface

The AT42QT1110 and AT42QT1111 communicate with the host microcontroller using SPI interface. This bus protocol takes care of all addressing functions and bidirectional data transfer.

In addition, the devices features a $\overline{\text{CHANGE}}$ pin, which can be used either to wake the host (useful in a battery powered application) or as an interrupt signal to inform the host when the QT™ device has detected a touch. The $\overline{\text{CHANGE}}$ pin can be left unconnected in systems where the QT device is polled on a regular basis.

2. Circuit Configuration with Host microcontroller

Following are the connections used in the demonstration program.

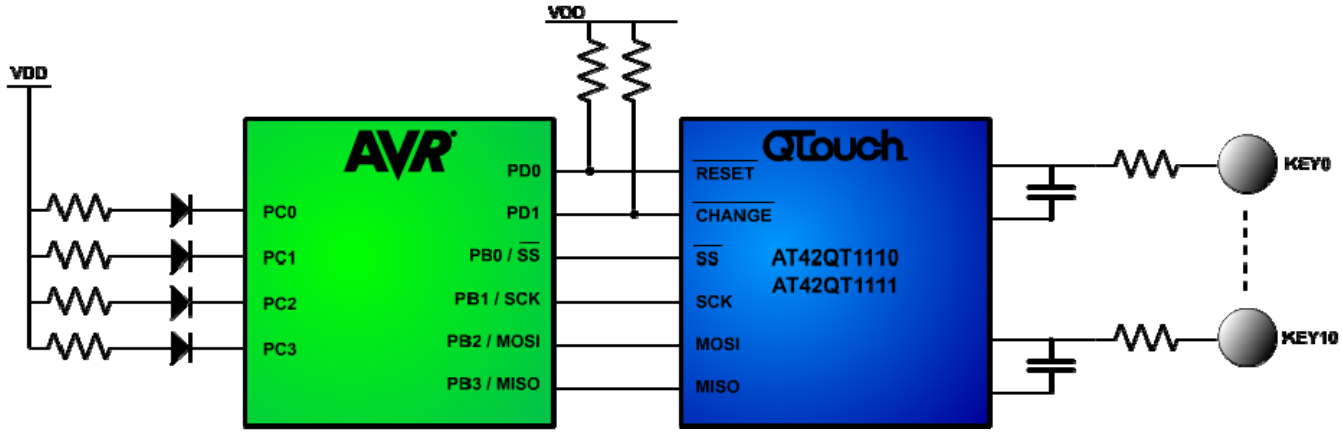
Table 2-1. Connection between Host microcontroller and QT device.

Atmel ATmega2560	AT42QT1110 and AT42QT1111
PD0 (Pin 43)	$\overline{\text{CHANGE}}$
PD1 (Pin 44)	$\overline{\text{RESET}}$
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.

External pull-up resistors of 10KΩ on the $\overline{\text{RESET}}$ and the $\overline{\text{CHANGE}}$ lines are needed.

Figure 2-1. Circuit configuration with Host microcontroller.



The demonstration program provides the visual feedback of the key touch status using LEDs. In the provided code, the PORT C of the Host microcontroller has been configured for this purpose.

3. SPI Communication

3.1 Specifications for the Atmel AT42QT1110 and AT42QT1111

The host communicates with the AT42QT1110 and AT42QT1111 over SPI using master-slave relationship, with QT devices acting in slave mode. There are a few specifications with regards to communication. These are mentioned below.

- The AT42QT1110 SPI Interface can operate at speed of up to 1.5MHz
- The AT42QT1111 SPI Interface can operate at speed of up to 750kHz
- The most significant bit (MSB) is the first byte in a byte of transmitted data
- Both master and slave set up data on the leading edge and read on the trailing edge
- The AT42QT1110 and AT42QT1111 require that the clock idles “high”
- In AT42QT1110 a minimum delay of 150µS is required between transmissions of each byte in case of multi-byte communication
- In AT42QT1111 a minimum delay of 300µS is required between transmissions of each byte in case of multi-byte communication
- If there is a delay of more than 100ms between two bytes in a multi-byte communication, then such a transmission is discarded

Note: Refer to respective device datasheets 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	NA
`uint8_t SPI_TransferByte (uint8_t DataOut)`	Simultaneously transmits and receives an 8-bit data	
	Input	`DataOut` - Data to be transmitted by master (unsigned 8-bit)
	Output	`DataIn` - Data received from slave (unsigned 8-bit)

3.3 SPI Command types for the Atmel AT42QT1110 and AT42QT1111

There are three types of communication commands between Host microcontroller and AT42QT1110 and AT42QT1111. Please refer to the device datasheet Section 4.1.4 for details of these commands.

- **Control Commands**

  A control command is an instruction sent to the QT devices that controls the operations of the device and for which no response is required.

  All control commands, without CRC enabled, require a single byte exchange.

  “Send Setups” (0x01) is an exception although, and needs to transmit 42 bytes of data for the setup block. In the demonstration program function `ReadSetupBlock(void)` is written separately for this special instruction.

- **Report Requests**

  A report request is an instruction sent to the QT devices to return status information.

  All report request commands are multi-byte exchanges. Each command has its own expected number of return bytes. “Null” command (0x00) has to be transmitted by the host device to receive data.

- **Setup commands**

  These commands are used to access and modify single bytes in the register map of the QT devices.

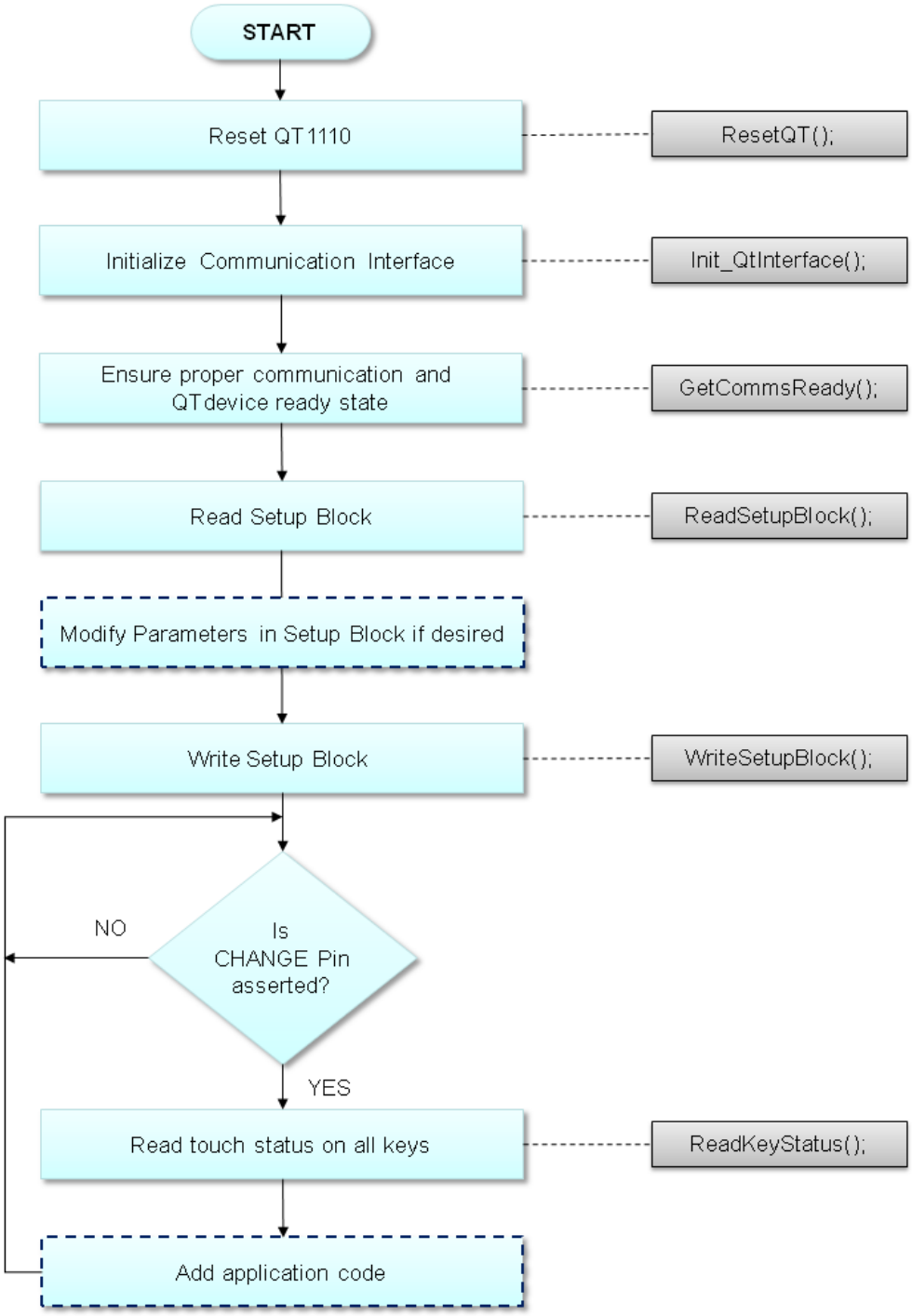
  - Set Instruction
  - Get Instruction

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 read and write the setup block which helps to tune the operating parameters of the device.

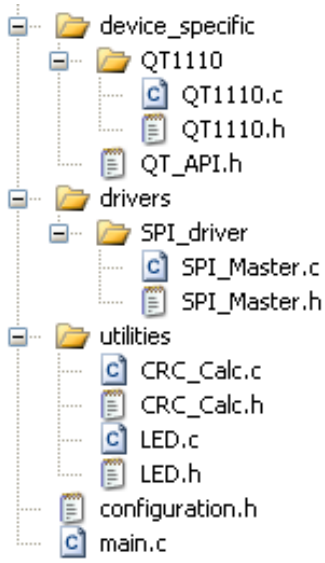
The source code has flexible implementation of the CRC based communications feature. The CRC feature can be enabled by ENABLE_CRC macro available in the configuration.h file.

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	Application code to be written here
configuration.h	Device selection settings to be done here
QT1110.c	Application interfaces to be used to drive the QT device
QT1110.h	Setup Block structure and function prototypes
SPI_Master.c	SPI driver code
SPI_Master.h	Header file for the SPI driver
CRC_Calc.c	Contains algorithm for 8-bit CRC checksum calculation
CRC_Calc.h	Header file for CRC
LED.c	Handling LED update based on touch key status
LED.h	Header file for LED source file

4.3 Functions

Function	Description	
`void ResetQT(void)`	Performs a hardware reset of the QT device by pulling down the RESET pin of the QT device	
	Input	NA
	Output	NA
`void InitQtInterface(void)`	Performs SPI Master initialization and configures a selected GPIO as input to detect the state of CHANGE pin of the QT device	
	Input	NA
	Output	NA
`void GetCommsReady(void)`	Uses the SPI command to read the Device ID to ensure proper communication	
	Input	NA
	Output	NA
`uint8_t ControlCommand (uint8_t Command)`	Sends Control Command instruction	
	Input	`Command` - control command (unsigned 8-bit)
	Output	Returns TRUE if successful or FALSE otherwise
`uint8_t ReportRequest (uint8_t Command, uint8_t NumBytes, uint8_t *DataAddr)`	Sends Report Request instruction	
	Input	`Command` - report request command (unsigned 8-bit) `NumBytes` - number of bytes of return data (unsigned 8-bit) `*DataAddr` - pointer to byte array for returned data
	Output	Returns TRUE if successful or FALSE otherwise
`uint8_t SetInstruction(uint8_t Command, uint8_t SetData)`	Sends Set Command instruction	
	Input	`Command` - set command (unsigned 8-bit) `SetData` - data to be set (unsigned 8-bit)
	Output	Returns TRUE if successful or FALSE otherwise
`uint8_t GetInstruction(uint8_t Command, uint8_t *ReadPtr)`	This function is used to send Get Command instruction to QT device	
	Input	`Command` - Get command (unsigned 8-bit) `*ReadPtr` - Pointer to byte for received data
	Output	Returns TRUE if successful or FALSE otherwise
`uint8_t ReadSetupBlock(uint8_t ReadLength, uint8_t *ReadPtr)`	Reads the entire setup block	
	Input	`ReadLength` - Length of the setup block. Only allowable input is the macro `SETUP_BLOCK_LENGTH` `*ReadPtr` - Pointer to byte array for read data
	Output	Returns TRUE if successful or FALSE otherwise

`uint8_t WriteSetupBlock(uint8_t WriteLength, uint8_t *WritePtr)`	Writes the entire setup block	
	Input	`ReadLength` - Length of the setup block. Only allowable input is the macro `SETUP_BLOCK_LENGTH` `*WritePtr` - Pointer to byte array containing write data
	Output	Returns TRUE if successful or FALSE otherwise
`uint8_t ReadKeyStatus(uint8_t ReadLength, uint8_t *ReadPtr)`	Read status of all keys	
	Input	`ReadLength` - Number of bytes to read `*ReadPtr` - Pointer to byte array for read data
	Output	Returns TRUE if successful or FALSE otherwise
`void InitTouchStatusPorts(void)`	Configure the PORTC pins 0 to 3 for displaying touch status	
	Input	NA
	Output	NA
`void UpdateLedStatus(uint8_t *QtStatusPtr)`	update touch key status through LED indications	
	Input	`QtStatusPtr : Pointer to byte array for Qtouch-data`
	Output	NA

5. Porting code to other platforms

This section 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     0       // 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. The configuration of the MCU pin is done in the function `ResetQT(void)` in the `QT1110.c` file.

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

```
// RESET port and pin configuration
#define RESET_PORT D      // PORT
#define RESET_PIN  1      // 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   DDB0
#define SPI_SCK  DDB1
#define SPI_MOSI DDB2
#define SPI_MISO DDB3

#define SS PB0
```

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. One byte of data (unsigned 8-bit) is transmitted on MOSI and received in MISO line simultaneously. The data to be transmitted is passed as an argument to the function and returns the received data.

6. Revision History

Doc. Rev.	Date	Comments
42040A	10/2012	Initial document release

Enabling Unlimited Possibilities®

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 Bldg.
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370