This document describes the ‘C’ demo code for the QT60xx0 24-key QMatrix Touch IC.
## Contents

1 Introduction .................................................. 3
2 Structure ...................................................... 3
3 Code Modules ................................................... 4
4 QT60xx0 Driver Functions ................................. 4
5 Function Description ......................................... 4
  5.1 Introduction .............................................. 4
  5.2 SetQtAddress () ........................................ 4
  5.3 ReadQtKeys () ........................................ 5
  5.4 GetQtKeys () ........................................ 5

5.5 CalibrateQt () ............................................. 5
5.6 ReadQtSetup () ........................................... 5
5.7 GetQtSetup () ........................................... 6
5.8 SetQtSetup () ........................................... 7
5.9 RestoreDefaultQtSetup () .............................. 8

6 Sample Host Application ................................. 9
7 I2C Driver .................................................... 9
8 Hardware Configuration ................................. 10
1 Introduction
This document describes the ‘C’ demo code for the QT60xx0 24-Key QMatrix Touch IC. The code is intended purely as an example of how driver code could be structured; it definitely does not indicate how an application should be written. The target processor for this example is a Silicon Labs C8051F310 device, and uses a bit-banged I^2C interface. If the demo is ported to other target devices, certain portions of the code will need to be rewritten.

2 Structure
Structure of the example project is shown in Figure 2.1. Arrows indicate signal flow.

Figure 2.1  Example Project Structure

The driver code is presented in module qt60xx0.c and its associated header file (qt60xx0.h). A very simple host program is included in main.c. This is by no means exhaustive, but does illustrate use of the driver functions. A further module, i2c.c provides a generic, bit-banged I^2C driver, which is used for communication with the QT60xx0.
3 Code Modules
Table 3.1 contains the modules which comprise the demo project.

<table>
<thead>
<tr>
<th>Module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qt60xx0.c</td>
<td>QT60xx0 Driver functions</td>
</tr>
<tr>
<td>Qt60xx0.h</td>
<td>Sample host application</td>
</tr>
<tr>
<td>main.c</td>
<td>Sample I2C communications driver</td>
</tr>
<tr>
<td>I2c.c I2c.h</td>
<td>Standard definitions etc.</td>
</tr>
<tr>
<td>common.h</td>
<td>Support functions for C8051F310-based host example.</td>
</tr>
<tr>
<td>timer.c timer.h c8051f310.h</td>
<td></td>
</tr>
</tbody>
</table>

All code relating to handling of the QT60xx0 is contained in Qt60xx0.c.

4 QT60xx0 Driver Functions
Table 4.1 contains the public driver functions which are included in qt60xx0.c.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetQtAddress</td>
<td>Allows the control program to specify the Qt device address</td>
</tr>
<tr>
<td>ReadQtKeys</td>
<td>Performs a read of the QT60xx0 key-status registers</td>
</tr>
<tr>
<td>GetQtKeys</td>
<td>Helper function, which returns the latest key states</td>
</tr>
<tr>
<td>CalibrateQt</td>
<td>Sends the Calibrate command sequence</td>
</tr>
<tr>
<td>ReadQtSetup</td>
<td>Reads the entire setup block from the QT60xx0 and stores it within the driver</td>
</tr>
<tr>
<td>GetQtSetup</td>
<td>Helper function, which returns a specified setup field</td>
</tr>
<tr>
<td>SetQtSetup</td>
<td>Assigns a value to a specified setup field within the driver and optionally writes the entire setup block to the QT60xx0</td>
</tr>
<tr>
<td>RestoreDefaultQtSetup</td>
<td>Resets the entire setup block to the factory default values</td>
</tr>
</tbody>
</table>

5 Function Description

5.1 Introduction
This section provides details regarding operation of the driver functions. It should be read in conjunction with the source code and the QT60xx0 datasheet.

5.2 SetQtAddress ()
Function: Allows the control program to specify the Qt device address.

Code: void SetQtAddress ( UINT8 Address )
{
    QtAddress = Address;
}

This function should be called once, at the start of the control program. If it is not called, the driver assumes the default address; 117.
5.3 ReadQtKeys ()

**Function:** Performs a read of the QT60xx0 key-status registers.

**Code:**
```c
UINT8 ReadQtKeys ( void )
{
    return (I2cRead( QtAddress, QT_KEYS_OFFSET, QtKeys, NUM_QT_KEY_BYTES) == 0) ? 1 : 0;
}
```

This function calls the I2cRead() function to read the three key-state bytes from addresses 1...3 within the QT60xx0. The values returned are saved in the array 'QtKeys[]' and can be accessed by the control program by calling GetQtKeys().

If the QT60xx0 responds correctly to the I2C bus transaction, the function returns the value ‘1’. If the transaction is NACK’ed the function returns ‘0’. This function can therefore be used to identify the presence of the QT60xx0 at the specified device address, often referred to a device polling.

5.4 GetQtKeys ()

**Function:** Helper function, which returns the latest key states.

**Code:**
```c
UINT8 GetQtKeys ( UINT8 Index )
{
    return ( QtKeys[Index] );
}
```

This function simply returns the latest key-state values from array ‘QtKeys[]’.

5.5 CalibrateQt ()

**Function:** Sends the Calibrate command sequence.

**Code:**
```c
UINT8 CalibrateQt ( void )
{
    /* Send calibrate command to QT via I2c */
    return (I2cWrite( QtAddress, QT_CALIBRATE_OFFSET, &QtCommand, 1) == 0) ? 1 : 0;
}
```

This function calls the I2cWrite() function to write the command code 0x55 to address 125 within the QT60xx0. This then activates the calibration sequence.

5.6 ReadQtSetup ()

**Function:** Reads the entire setup block from the QT60xx0 and stores it within the driver.

**Code:**
```c
UINT8 ReadQtSetup ( void )
{
    return (I2cRead( QtAddress, QT_SETUP_OFFSET, QtSetupBlock, LEN_QT_SETUP) == 0) ? 1 : 0;
}
```

This function calls the I2cRead() function to read the 123-byte setup block from the QT60xx0 to the array QtSetupBlock[] within the driver.

If the QT60xx0 responds correctly to the I2C bus transaction, the function returns the value ‘1’. If the transaction is NACK’ed the function returns ‘0’.
5.7 GetQtSetup ()

Function: Helper function, which returns a specified setup field.

Code:  

```c
UINT8 GetQtSetup ( UINT8 Key, UINT8 Param )
{
    UINT8 Index;

    /* calculate index into Setup image array */
    Index = ParamDef[Param].Offset;
    if ( Param <= QT_PARAM_SSYNC )
        Index += Key;

    /* access required value */
    return (QtSetupBlock[Index] >> ParamDef[Param].Position) & ParamDef[Param].Mask;
}
```

This function simply returns the requested setup field from array ‘QtKeys[]’. As some fields are stored as bit-fields within the setup block, GetQtSetup() extracts and returns just the relevant field.

The field is specified by the input parameters Key and Param, where:

- **Key** specifies the key in the range 0...23.
- **Param** specifies the associated parameter and can be one of the following as enumerated in qt60xx0.h:
  - QT_PARAM_NTHR,
  - QT_PARAM_NDRIFT,
  - QT_PARAM_PDRIFT,
  - QT_PARAM_NDIL,
  - QT_PARAM_FDIL,
  - QT_PARAM_NRD,
  - QT_PARAM_WAKE,
  - QT_PARAM_BL,
  - QT_PARAM_AKS,
  - QT_PARAM_SSYNC,
  - QT_PARAM_SLEEP,
  - QT_PARAM_MSYNC,
  - QT_PARAM_AWAKE,
  - QT_PARAM_DHT

For the global parameters, the key should be specified as 0.
5.8 SetQtSetup()

**Function:** Assigns a value to a specified setup field within the driver and optionally writes the entire setup block to the QT60xx0.

**Code:**

```c
UINT8 SetQtSetup ( UINT8 Key, UINT8 Param, UINT8 Value, UINT8 WriteFlag )
{
    UINT8 RetVal = true; /* function return value */
    UINT8 Temp;
    UINT8 Index;

    /* calculate index into Setup image array */
    Index = ParamDef[Param].Offset;
    if ( Param <= QT_PARAM_SSYNC )
        Index += Key;

    /* only process data if a valid index has been specified */
    if ( (Index < LEN_QT_SETUP) && (Key < NUM_QT_KEYS) )
    {
        /* overwrite relevant bit-field within Setup image using ParamDef array */
        Temp = (Value & ParamDef[Param].Mask) << ParamDef[Param].Position;
        QtSetupBlock[Index] &= ~(ParamDef[Param].Mask << ParamDef[Param].Position);
        QtSetupBlock[Index] += Temp;

        if ( WriteFlag ) /* if required, write complete Setup image to QT device */
        {
            /* first, unlock the setup data area for writing */
            if ( I2cWrite (QtAddress, QT_UNLOCK_OFFSET, &QtCommand, 1) )
                RetVal = false; /* fail if I2C returned non-zero status */
            else
                /* if the unlock was successful, write the setup image */
                if ( I2cWrite (QtAddress, QT_SETUP_OFFSET, QtSetupBlock, LEN_QT_SETUP) )
                    RetVal = false; /* fail if I2C returned non-zero status */
        }
    }
    else
    {
        RetVal = false; /* fail if illegal parameter was specified */
        return RetVal; /* completed */
    }
}
```

SetQtSetup() performs the opposite function to calling ReadQtSetup() and GetQtSetup(). It allows a setup field within the array ‘QtKeys[]’ to be updated. Optionally, the entire array can be written back to the setup block within the QT60xx0. The write-process comprises a write of the control code (0x55) to location 130, followed by a write of the 123-byte setup block to locations 131 onwards.
The function requires four input parameters:

- **Key** specifies the key in the range 0...23.
- **Param** specifies the associated parameter as enumerated in qt60xx0.h.
- **Value** specifies the new value to be written to the parameter.
- **WriteFlag** controls the write to the QT60xx0:
  - 0 = don’t write,
  - 1 = Write.

If the QT60xx0 responds correctly to the I²C bus transaction, the function returns the value ‘1’. If the transaction fails at any point, the function returns ‘0’.

### 5.9 RestoreDefaultQtSetup()

**Function:** Resets the entire setup block to the factory default values.

**Code:**
```c
UINT8 RestoreDefaultQtSetup ( void )
{
    UINT8 i;

    /* write default values to key parameters for all keys */
    for (i = 0; i < NUM_QT_KEYS; i++)
    {
        QtSetupBlock[i] = QT_DEF_NTHR + (QT_DEF_NDRIFT << 4);
        QtSetupBlock[i + 24] = QT_DEF_PDRIFT;
        QtSetupBlock[i + 48] = QT_DEF_NDIL + (QT_DEF_FDIL << 4);
        QtSetupBlock[i + 72] = QT_DEF_NRD;
        QtSetupBlock[i + 96] = QT_DEF_BL << 4;
    }

    /* write global parameters */
    QtSetupBlock[120] = QT_DEF_SLEEP;
    QtSetupBlock[121] = QT_DEF_AWAKE;

    /* set DHT default by calling SetQtSetup() also writes device */
    return ( SetQtSetup(0, QT_PARAM_DHT, QT_DEF_DHT, true) );
}
```

RestoreDefaultQtSetup() resets the entire setup block within the driver and QT60xx0 to the factory preset values as defined in the datasheet and qt60xx0.h.

If the QT60xx0 responds correctly to the I²C bus transaction, the function returns the value ‘1’. If the transaction fails at any point, the function returns ‘0’.
6 Sample Host Application
The sample host application is presented in module main.c.

Code: `InitI2c (); /* initialise I2c interface */`

```c
/* Specify QT60xx0 I2C device address to QT driver */
SetQtAddress( 117 );

/* wait for QT60xx0 to respond to I2C read transfers */
while (!ReadQtKeys());

/* read the QT60xx0 setup block */
ReadQtSetup ();

/* loop forever */
while (1) {
    if (CheckTimer (Time, 20) )
    {
        Time = StartTimer();
        if ( ReadQtKeys() ) /* poll QT60xx0 for key states */
        {
            /* get key states for onward processing.... */
            Key[0] = GetQtKeys(0);
            Key[1] = GetQtKeys(1);
            Key[2] = GetQtKeys(2);
        }
    }
}
```

The sample host shown in main.c does the following:

1. **SetQtAddress( 117 );** Sends I2C device address to the QT60xx0 driver.
2. **while (!ReadQtKeys());** Waits for the QT60xx0 to complete its internal power-up sequence and become active on the I2C bus. The function ReadQtKeys() returns the value '0' if the device NACK's its slave address.
3. **ReadQtSetup ();** Reads the QT60xx0 setup block into the driver. This completes the initialisation sequence.
4. The program then enters a loop, which polls the QT60xx0 key states every 20ms. If the poll is successful, the key states are fetched for onward processing.

The timer functions implemented in timer.c provide a 1ms timebase using a hardware timer.
The modules: "C8051F310.H","common.h","timer.h" all relate to the C8051F310 implementation.

7 I²C Driver
The wide variety of hardware I²C interfaces implemented in different microprocessor families rules out a generic software driver function. For this reason, the sample I²C driver provided in i2c.c is a bit-banged interface using a pair of general port-pins for the SCL and SDA signals. This is necessarily complex due to the need to cope with clock-stretching and NACK’ed transactions, which are part of the QT60xx0 comms protocol. The code is not described fully here, but is written so that it should be easily portable to alternative processor platforms.
8 Hardware Configuration

Figure 8.2 illustrates the connections between QT60xx0 and C8051F310 used by the example code.

Figure 8.2 Connections Between QT60xx0 and C8051F310

Qt60xx0 Demo Configuration