Atmel

Atmel AT42QT1085

Object Protocol Guide

1. Introduction

This document describes in detail the Object Protocol for the AT42QT1085 (QT1085).

The Object Protocol provides a single common interface across the Atmel® QTouch® touch sensor controllers. This allows the different features in each controller to be configured in a consistent manner. It makes the future expansion of features and simple product upgrades possible, whilst allowing backwards compatibility for the host driver and application code.

2. Overview

2.1 Memory Map Structure

The protocol is designed to control the processing chain in a modular manner. This is achieved by breaking the features of the device into objects that can be controlled individually. Each object represents a certain feature or function of the device, such as a touchscreen or a key array. Objects can be disabled or enabled as needed.

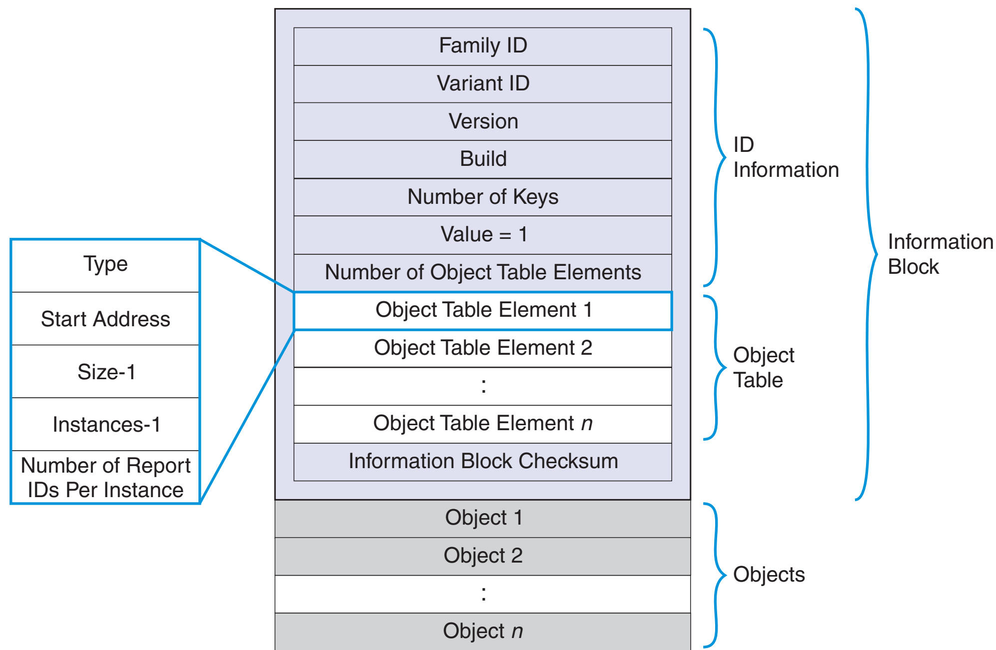
Each object has its own configuration memory. The objects are stacked together to produce an object-based memory map. A generalized structure of this memory map is shown in Figure 2-1.

There are some special objects that can use their memory space for a unique purpose. One example is the Command Processor T6 object, which executes a command when a certain value is written into its memory space. Another example is the Message Processor T5 object, which outputs messages by having the host read its memory space.

From Figure 2-1 on page 2 it can be seen that the memory map contains two main sections:

- An Information Block. This documents which objects are contained in the memory map for the device (see Section 2.2 on page 2). It is further sub-divided into the ID Information Block and the Object Table (see Section 2.4 on page 4).
- The objects themselves (see Section 2.5 on page 5).

Figure 2-1. Generic Memory Map Structure



2.2 Information Block

The Information Block allows the host to read information about the layout of objects in the memory map. It contains a list of all the objects in the memory map. This is used by the host driver to know which objects exist, where they are located in the memory map and their sizes. The host driver can therefore read the device Information Block and gather enough information to be able to communicate with the device.

The Information Block is positioned at the start of the memory map at address zero. This allows it to be read easily by the host as the first operation.

The Information Block contains the following three sections:

- The ID Information fields. These include:
  - Standard ID fields that make up the unique identifier for the device
  - The number of touch sensor channels that the device supports
  - The number of objects in the Object Table
- The Object Table itself. This acts as an *index* to the objects in the memory map.
- A checksum for the Information Block. This allows the host to check that the Information Block has been read correctly over the communications interface. See Appendix A. on page 33 for details on calculating the checksum.

Table 2-1 shows the contents of the Information Block.

Table 2-1. Information Block Layout

<table>
<tr><th>Byte</th><th>Description of Field</th><th></th></tr>
<tr><td>0</td><td>Family ID</td><td rowspan="7">ID Information</td></tr>
<tr><td>1</td><td>Variant ID</td></tr>
<tr><td>2</td><td>Version</td></tr>
<tr><td>3</td><td>Build</td></tr>
<tr><td>4</td><td>X = number of keys</td></tr>
<tr><td>5</td><td>Y = 1</td></tr>
<tr><td>6</td><td>Number of elements in the Object Table</td></tr>
<tr><td>7 – 12</td><td>Object Table element 1 (6 bytes)</td><td rowspan="4">Object Table</td></tr>
<tr><td>13 – 18</td><td>Object Table element 2 (6 bytes)</td></tr>
<tr><td>…</td><td>…</td></tr>
<tr><td>...</td><td>Last Object Table element (6 bytes)</td></tr>
<tr><td>(end-2) – end</td><td>24-bit checksum (3 bytes)</td><td>Checksum Field</td></tr>
</table>

2.3 ID Information

The first four bytes of the Information Block are used to identify the device and its version, as in Table 2-2.

Table 2-2. Device Identifier Fields

Field	Description
Family ID	A unique identifier that indicates the device family. The family ID for the QT1085 is `0x03`.
Variant ID	A unique identifier that indicates the device variant. The variant ID for the QT1085 is `0x00`.
Version	The current major and minor firmware version of the device. The upper nibble contains the major version and the lower nibble contains the minor version. For example, firmware version 0.1 is stored as `0x01`.
Build	The firmware build number.

2.4 Object Table

2.4.1 Introduction

The Object Table is held within the Information Block and contains information on all the objects held within the memory map. It indicates which objects are present and their addresses.

Each element in the Object Table has the fields listed in Table 2-3.

Table 2-3. Format of an Object Table Element

Byte	Description of Field
0	Type
1	Start position LSByte
2	Start position MSByte
3	Size – 1
4	Instances – 1
5	Number of Report IDs per instance

2.4.2 Type

Each type of object has a unique type code to identify it. This is the number after the *_T* suffix at the end of the object internal name as given in Section 3. to Section 7.. For example, the type code for the Command Processor is **6** (from GEN_COMMANDPROCESSOR_T**6**).

2.4.3 Start Position

Bytes 1 and 2 of the Object Table element hold the start location of the object in the memory map (LSByte and MSByte respectively).

The driver code should ALWAYS read these bytes to find out where in the memory map the object is located and use this address to communicate with the device. The driver code should never use hard-coded addresses for the objects, as these may change with firmware updates.

This means that driver code can be written without making assumptions about the addresses of the objects. This ensures that the code is *future-proof* and will work correctly following firmware updates to the device. It also makes it possible to write common driver code for communication with any Atmel touch controller that uses this object-based protocol approach.

2.4.4 Size

Byte 3 of the Object Table element holds the size (minus 1) of the object in the memory map. This is stored as Size–1, so it is effectively the offset to the end of the object.

2.4.5 Number of Instances

Byte 4 of the Object Table element holds the number of instances of the object in the memory map, minus 1. The number of instances can be calculated by adding 1 to this number (see Table 2-4 on page 6). The different instances of an object are arranged consecutively in the memory map.

2.4.6 Report IDs

If an object sends messages, it is necessary to identify the messages from the object so that they can be correctly interpreted. A report ID is therefore used to identify the source object of a message returned in the Message

Processor T5 object (see Section 4.2 on page 10).

Report IDs are numbered sequentially in the order in which the objects are listed in the Object Table, allowing for the appropriate number of instances for each object. Note the following:

- A report ID of zero is a reserved value for use by Atmel. Report IDs from a user's perspective therefore effectively start from 1.
- A report ID value of 255 is reserved to indicate an *invalid message* response.

If an object has report IDs allocated, each instance of the object will have its own block of report IDs.

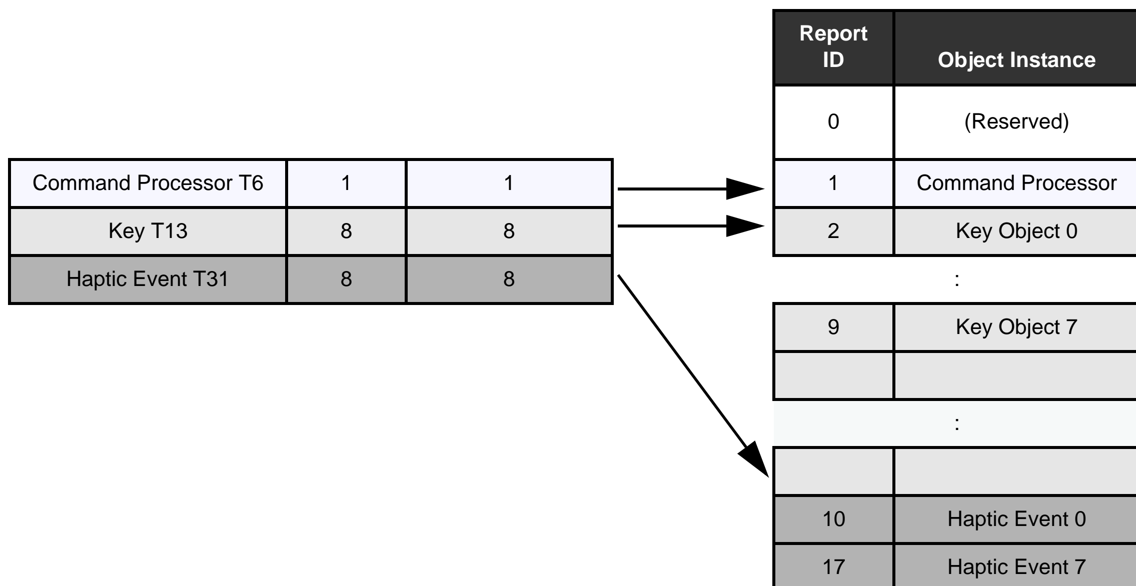
Figure 2-2 on page 5 shows an example of how the number of instances and the Report IDs per instance determine the report IDs for a set of objects. Note that the objects shown in Figure 2-2 are examples only and may not reflect the actual objects present on the QT1085.

The driver code should build up its own in-memory table of object types and associated report IDs during its initialization. It can do this by parsing the object structure given in the Object Table. This in-memory table can then be used to interpret the messages returned by the device.

A typical algorithm to process the report IDs is as follows:

1. For each element in the Object Table:
    1. Read byte 4 to retrieve the number of instances (remember to add 1 to the value retrieved).
    2. Read byte 5 to retrieve the number of report IDs per instance.
    3. Multiply the figures retrieved in steps 1 and 2 together, and then add this number of *object type/report ID* pairings to the table being built. The report IDs should have sequential values starting with 1 (the zero value is reserved for use by Atmel).

Figure 2-2. Example Assignment of Report IDs



2.5 Objects

2.5.1 Classes of Objects

The QT1085 contains the following classes of objects:

- **Debug objects** – provide a raw data output method for development and testing. See Section 3. on page 8.
- **General objects** – required for global configuration, receiving commands and transmitting messages. See Section 4. on page 10.
- **Touch objects** – operate on measured signals from the touch sensor and report touch data. See Section 5. on page 17.
- **Signal processing objects** – process data from other objects (typically signal filtering operations). See Section 6. on page 20.
- **Support objects** – provide additional functionality on the device. See Section 7. on page 23.

2.5.2 Object Instances

Table 2-4 lists the instances of the objects on the QT1085.

Table 2-4. Objects on the QT1085

Object	Number of Instances	Reference
Debug Objects		
Debug Delta T2	1	Section 3.2 on page 8
Debug Reference T3	1	Section 3.3 on page 9
Debug Signals T4	1	Section 3.4 on page 9
General Objects		
Message Processor T5	1	Section 4.2 on page 10
Command Processor T6	1	Section 4.3 on page 11
Power Configuration T7	1	Section 4.4 on page 13
QTouchADC Configuration T49	1	Section 4.5 on page 14
Touch Objects		
Key T13	8	Section 5.2 on page 17
Signal Processing Objects		
Touch Configuration T16	1	Section 6.2 on page 20
Support Objects		
Self Test T25	1	Section 7.2 on page 23
GPIO Configuration T29	16	Section 7.3 on page 25
Haptic Event T31	8	Section 7.4 on page 29

2.5.3 Configuration Defaults

The objects are designed such that a default value of zero in their fields is a *safe* value. For example, a value of zero typically disables functionality.

An object must be configured as required with non-zero values before use. Any unused settings can be left at their default zero values. The settings should also be written to the nonvolatile memory using the Command Processor T6 object (see Section 4.3 on page 11).

2.5.4 Compatibility of Object Versions

The Object Protocol described in this datasheet may document fields that are not present in the memory map as it is implemented on a particular firmware version of the device. Over time newer versions of the objects in the Object Protocol may gain additional fields to implement new features.

New fields are added to the end of an object to allow for this situation. This preserves the order of the fields between old and new object versions. A driver designed to work with an older version of the device can safely set any unknown fields located at the end of the object to zero. The device will then behave in the same manner as the older version of the device without the field.

The host driver must always use the Information Block (see Section 2.2 on page 2) to locate the address of each object and the current size of the object. It must also zero any fields that it does not intend to use. This ensures that the host driver code is compatible with this object expansion scheme.

2.6 Configuration Checks

A configuration check is carried out following any write operation. A configuration check may determine that a configuration error has occurred (for example, if a setting is set outside of its allowed range or a conflict has occurred between two settings). This is signaled to the host (see Section 4.3.2 on page 12), and the device halts until the error has been corrected. To fix the error, the object settings should be checked to verify that they are all within their allowed limits, as stated in the field descriptions.

The device prevents invalid configurations by performing a sequence of checks. These checks are performed on power-up and whenever configuration settings are updated.

If an error is found during the configuration check, the device pauses until the configuration error is resolved. The device also flags the error to the host by setting the CFGERR bit of the Command Processor message data (see Section 4.3.2 on page 12). This message will be sent to the host every 200 ms until the error is corrected.

Backup requests are allowed when an error has been found during a configuration check. This allows a setting to be corrected and backed up to the nonvolatile memory (NVM). The device can then be reset for the setting to take effect, if the setting requires this. The device will not operate until all errors have been corrected.

Possible causes of configuration errors are:

- Multiple objects enabled which use the same pin
- Field settings outside the permitted range
- GPIO or Haptic Events whose source object is not enabled

If a configuration error is encountered during product development, it is sufficient for the designer to check the settings used and correct them. In a working product, the device driver should resend the configuration settings and request a backup. The device driver should be written to handle this.

To find the source of the configuration error, first disable all touch, signal processing and support objects. Then re-enable each object in turn until the configuration error is found. Once the error has been corrected, the other objects can be re-enabled and processing can continue as normal. See Appendix B. on page 37 for a flow chart showing this method for finding configuration errors.

Notice should be taken of any recommendations in this datasheet concerning configuration checks for the offending object.

2.7 Byte Order

The memory map uses a little-endian configuration for its bytes, meaning that all multi-byte fields lead with the least significant byte (LSByte).

3. Debug Objects

3.1 Introduction

Debug objects contain raw data for development and testing purposes. Table 3-1 lists the debug objects on the QT1085.

Table 3-1. Debug Objects

Object and Name	Description
Delta Debug (DEBUG_DELTAS_T2)	Allows the measurement *deltas* to be read from the object memory space. See Section 3.2.
Reference Debug (DEBUG_REFERENCES_T3)	Allows the measurement *references* to be read from the object memory space. See Section 3.3.
Signals Debug (DEBUG_SIGNALS_T4)	Allows the measurement signals to be read from the object memory space. See Section 3.4.

3.2 Delta Debug (DEBUG _DELTAS_T2)

The Debug Delta T2 object contains the raw signed 16-bit delta values received from all the sensing channels. These values are supplied for debugging purposes

The maximum data size that can be held in the Debug Reference T3 object is 256 bytes (that is, the delta values for up to 128 channels). .

Table 3-2. DEBUG _DELTAS_T2

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	CHANNEL[0]	Channel 0 delta LSByte							
1		Channel 0 delta MSByte							
…	…	…							
254	CHANNEL[127]	Channel 127 delta LSByte							
255		Channel 127 delta MSByte							

3.3 Reference Debug (DEBUG_REFERENCES_T3)

The Debug Reference T3 object contains the raw 16-bit reference values received from all the sensing channels. These values are supplied for debugging purposes.

The maximum data size that can be held in the Debug Reference T3 object is 256 bytes (that is, the reference values for up to 128 channels).

Table 3-3. DEBUG _REFERENCES_T3

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	CHANNEL[0]	Channel 0 reference LSByte							
1		Channel 0 reference MSByte							
…	…	…							
254	CHANNEL[127]	Channel 127 reference LSByte							
255		Channel 127 reference MSByte							

3.4 Signals Debug (DEBUG_SIGNALS_T4)

The Debug Signals T4 object contains the raw 16-bit signal values received from all the sensing channels. These values are supplied for debugging purposes.

The maximum data size that can be held in the Debug Signals T4 object is 256 bytes (that is, the signal values for up to 128 channels).

Table 3-4. DEBUG _SIGNALS_T4

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	CHANNEL[0]	Channel 0 signal LSByte							
1		Channel 0 signal MSByte							
254	CHANNEL[127]	Channel 127 signal LSByte							
255		Channel 127 signal MSByte							

Note: All 16-bit fields are *non-atomic* so it is possible for the 2 bytes of a 16-bit value to be updated by the device mid-read. In this case the host may read the LSB of the earlier value and the MSB of the later value. Where the value changes from positive to negative or increases from 255 to 256, this may lead to the 16-bit value read by the host being quantitatively incorrect.

4. General Objects

4.1 Introduction

General objects provide global configuration, such as receiving commands and transmitting messages. Table 4-1 lists the general objects on the QT1085.

Table 4-1. General Objects

Object	Description
Message Processor (GEN_MESSAGEPROCESSOR_T5)	Handles the transmission of messages. This object holds a message in its memory space for the host to read. See Section 4.2.
Command Processor (GEN_COMMANDPROCESSOR_T6)	Performs a command when written to. Commands include reset, calibrate and backup settings. See Section 4.3.
Power Configuration (GEN_POWERCONFIG_T7)	Controls the sleep mode of the device. Current consumption can be lowered by controlling the acquisition frequency and the sleep time between acquisitions. See Section 4.4.
QTouchADC Configuration (GEN_QTOUCHADC_T49)	Configures the settings for the QTouchADC touch detection method. See Section 4.5.

4.2 Message Processor (GEN_MESSAGEPROCESSOR_T5)

The purpose of the Message Processor T5 object is to relay the latest status information to the host. This object contains the message data from those objects in the memory map that generate messages (for example, the touch objects and the Command Processor T6 object). A message is generated whenever an object's status has changed. For this to happen, the object report enable bit must be set.

When a device has data to send, it asserts the $\overline{\text{CHANGE}}$ line to indicate to the host that there is a message. The host should then read the message and use the REPORTID field to determine from which object the message originated. This provides the host with an interrupt-style interface. This has the potential for fast response times and reduces the need for wasteful communications.

The host should ALWAYS use the $\overline{\text{CHANGE}}$ line as an indication that a message is available to read in the Message Processor T5 object. The host should not read the Message Processor T5 object at any other time, such as to poll the device for messages. If the Message Processor T5 object is read when the $\overline{\text{CHANGE}}$ line is not asserted, an *invalid message* report ID is returned in the Message Processor T5 object.

Table 4-2. GEN_MESSAGEPROCESSOR_T5

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	REPORTID	Report ID for source object							
1 – *n* [1]	MESSAGE	Message data from source object							
n + 1		CRC of message data							

1. The size of the MESSAGE field is set to the length of largest message possible. It is dependent on the objects present in the device at a particular revision. The size should be calculated by subtracting 2 from the size of the Message Processor T5 object retrieved from the Object Table entry (see Section 2.4 on page 4). Any unused bytes in a particular message should be treated as reserved bytes.

REPORTID Field

This field contains the report ID for the message. Messages contain report IDs to allow the host to identify the type of message and its originator. Report IDs are assigned to any object that can send messages. See Section 2.4.6 on page 5 for more information on the assignment of report IDs.

MESSAGE Field

This field contains the message data for the object generating the message.

The size of the MESSAGE field is fixed to the size of the message data for the largest object. For compatibility with future firmware updates, this should *always* be calculated by subtracting 2 from the size of the object recorded in the Object Table entry for the Message Processor T5 object (see Section 2.4 on page 4).

For information on the contents of the MESSAGE field, see the descriptions for each object elsewhere in this datasheet.

4.3 Command Processor (GEN_COMMANDPROCESSOR_T6)

The Command Processor T6 object allows commands to be sent to the device. This is done by writing an appropriate value to one of its fields.

4.3.1 Configuration

Table 4-3. GEN_COMMANDPROCESSOR_T6

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	RESET	Reset							
1	BACKUPNV	Backup settings							
2	CALIBRATE	Calibrate							
3	REPORTALL	Report current status							
4 – 5	Reserved	Reserved							

RESET Field

This field forces a reset of the device if a nonzero value is written.

Write value: Nonzero (normal)

BACKUPNV Field

This field backs up settings to the non-volatile memory (NVM). Once the device has processed this command it generates a status message containing the new NVM checksum.

Write value: `0x55`

CALIBRATE Field

This field performs a global recalibration on all channels. If all the channels are disabled, no message is generated. If the device is in Deep Sleep mode (see Section 4.4), a message is generated when the device wakes from sleep.

Write value: Nonzero

REPORTALL Field

This field forces all objects that generate messages to report their current status:

- For optional objects, this applies only if they have their report enable bit set and are currently enabled.
- For objects that are always present and generate messages setting this bit will always cause a status message to be reported.

This field is cleared once the command has been processed.

Write value: Nonzero

4.3.2 Messages

The message data (see Section 4.2) for the Command Processor T6 object is shown in Table 4-4.

Table 4-4. Message Data for GEN_COMMANDPROCESSOR_T6

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	REPORT ID	Report ID							
1	STATUS	RESET	OFL	SIGERR	CAL	CFGERR	COMMSERR	Reserved	
2	CHECKSUM	Configuration settings checksum							

STATUS Field

Reports the current status and flags errors. A bit is set to indicate the corresponding status/error. Note that there may be more than one status/error reported.

CFGERR, CAL, SIGERR and OFL report ongoing status and error conditions, so once these status/error conditions have terminated, a further message is sent with the appropriate bit cleared.

COMSERR and RESET are one-off reports indicating already terminated conditions. These error conditions do not generate a further message with a cleared bit.

COMSERR: There is an error with the communications checksum. This error bit is set when a read or write operation is not completed correctly; that is:

- An $\overline{SS}$ high event has occurred before the requested number of Read or Write bytes has been sent.
- Additional byte exchanges have been attempted after the requested number of Read or Write bytes has been sent, but no $\overline{SS}$ high event has been detected to initiate a new exchange.
- An attempt to write to a read-only field of the memory map.

Note: If there is an error after a write, then the data will still have been written to the device. It is the responsibility of the host to take corrective action. COMSERR is cleared when $\overline{SS}$ is taken high after a correctly completed Read or Write exchange.

CFGERR: There is a configuration error in one or more of the enabled objects. The device pauses its processing and generates a status message every 200 ms. Note that the device will stop scanning for touches while the error persists.

Note: It is possible to execute a backup command while the device is in this error state.

See Section 2.6 on page 7 for more information on configuration checks.

CAL: The device is calibrating.

SIGERR: There was an error in the acquisition.

OFL: The acquisition and processing cycle length has overflowed the desired power mode interval. These are controlled by the IDLEACQINT and ACTVACQINT fields in the Power Configuration T7 object (see Section 4.4). Note that the OFL flag is not updated in Free-run or Deep Sleep modes.

RESET: The device has reset.

CHECKSUM Field

Reports the checksum of the configuration settings held in the nonvolatile memory. See Appendix A. on page 33 for details on how to calculate the checksum.

4.4 Power Configuration (GEN_POWERCONFIG_T7)

The Power Configuration T7 object controls the active and idle (sleep) times of the device. Current consumption can be lowered by controlling the acquisition frequency and sleep times between measurements.

The device operates in two modes: *active* (touch detected) and *idle* (no touches detected).

The normal state for the device is *idle* mode. In this mode the device operates in a series of long burst cycles. Each cycle consists of a short burst (during which measurements are taken to detect a possible touch) followed by an inactive sleep period.

When a touch is detected the device enters a *Fast DI* (free-run) mode. This consists of a series of free-run bursts to confirm that a change in the touch state has indeed occurred. The number of bursts is determined by the TCHDI field in the touch objects). If it is a genuine touch, the device enters active mode. In this mode the device operates in a series of burst cycles that intersperse measurement bursts with very short sleep periods. These sleep periods are typically shorter than those in idle mode.

When the user releases the touch, the device again enters a short Fast DI mode to confirm that a change in the touch state has occurred. Then, if it is a genuine release, the device returns to idle mode after a short timeout period. During this timeout, the device continues to run in active mode to allow further touches to keep the device active.

Note that the changes to the cycle time happen regardless of whether the touch object is reporting or not.

Table 4-5. GEN_POWERCONFIG_T7

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	IDLEACQINT	Idle Acquisition Interval							
1	ACTVACQINT	Active Acquisition Interval							
2	ACTV2IDLETO	Active to Idle Time Out							

IDLEACQINT and ACTVACQINT Fields

The length of the idle and active burst cycles is determined by the Idle Acquisition Interval (IDLEACQINT) and the Active Acquisition Interval (ACTVACQINT) fields respectively.

A setting of 255 forces the device to enter Free-run mode the next time that the appropriate mode (idle or active) is entered. In Free-run mode the device does not sleep between acquisitions. This gives the fastest response time at the expense of power consumption.

A setting of zero forces the device to enter Deep Sleep mode the next time that the appropriate mode (idle or active) is entered. The device remains in Deep Sleep mode until the IDLEACQINT or ACTVACQINT setting is restored. Deep Sleep mode is used to conserve maximum power if the device does not need to be sensing. If Deep Sleep mode is requested, it is advisable to set both IDLEACQINT and ACTVACQINT to zero to avoid indeterminate behavior if one mode is still active. The status flags in the Command Processor (see Section 4.3 on page 11) are not updated when the device is in Deep Sleep mode.

Other values for IDLEACQINT or ACTVACQINT determine the Idle or Active Acquisition Interval in millisec-onds. A high value causes more sleep time between acquisitions. This results in lower power consumption but a slower response time.

Do not set either field to be less than the actual burst time. The device is also designed to sleep as much as possible in order to conserve power. IDLEACQINT should therefore be set longer than ACTVACQINT. Under some circumstances it may be desirable to set IDLEACQINT lower than ACTVACQINT. For example, this might be necessary to minimize the difference between the best-case and the worst-case touchdown latency.

The minimum interval that can be specified is 4 ms. If an interval is required that is shorter than the minimum value allowed, the Free-run mode setting (255) should be used instead.

Range: 0 (Deep Sleep), 4 to 254 (interval in ms), 255 (Free-run)

IDLEACQINT Typical: 32 (32 ms)

ACTVACQINT Typical: 16 (16 ms)

ACTV2IDLETO Field

The device automatically goes into idle mode whenever possible after each scan to conserve power, unless a touch object is being touched.

The device does not go into idle mode immediately. Instead there is a timeout period. The device runs in active mode during this timeout period to allow further touches to keep the device active. This timeout period is determined by the Active to Idle Timeout (ACTV2IDLETO) field. Under normal operation, the device enters idle mode after the expiry of the Active to Idle Timeout and then remains in idle mode until the next touch is detected. If there is more than one touch present, the Active to Idle Timeout applies only after the last touch has been released. This means that once the device has been awakened by a change, the touch response time is fast for as long as the sensor remains in use. Once channel activity lapses for a period longer than the Active to Idle Timeout, the device returns to idle mode.

The Active to Idle Timeout is specified in 200 ms increments, where 0 means 1 cycle.

Range: 0 (1 cycle), 1 to 255 (in 200 ms increments)

Typical: 50 (10 seconds)

4.5 QTouchADC Configuration T49 (GEN_QTOUCHADC_T49)

The Tuning of the T49 settings for each channel allows the QTouchADC measurement procedure to provide excellent results with a wide variety of sensor shapes and sizes and with different cover materials and thickness. The purpose of a sensor can even be changed on the fly, for example a Proximity Sensor may be reconfigured as a Guard Sensor simply by reducing the signal gain. This is done either by reducing the BLEN setting (for faster operation) or increasing SCALE (for greater signal stability) such that the channel goes into detect on contact instead of on approach.

4.5.1 Configuration

Table 4-6. QTouchADC Configuration T49

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	CONFIG_CH[0]	BLEN				SCALE			
:									
n	CONFIG_CH[*n*]	BLEN				SCALE			

CONFIG_CH Field

SCALE: The scale factor (averaging factor) for the accumulated signal is an exponent of 2. Unit = Scaling denominator. (see Table 4-7).

Table 4-7. Scale Factor

Setting	Scale Factor	Setting	Scale Factor
0	1	8	256
1	2	9	512
2	4	10	1024
3	8	11	2048
4	16	12	4096
5	32	13	8192
6	64	14	16384
7	128	15	32768

Note: To ensure correct operation of the QT1085:

BLEN / SCALE combination must never lead to a signal of > 32767 or < 32

BLEN must never exceed SCALE by more than 5

BLEN: This is the number of measurements accumulated and is an exponent of 2. The number of QTouchADC pairs to be generated for each key measurement.

Each pair of pulses give a 10-bit signal value. These signal values are accumulated over the number of QTouchADC pairs and the total is divided by the scale factor. This configuration option allows for a wide configuration of Gain and Averaging on the acquisition signals.

Table 4-8. BLEN (Number of Measurements Accumulated)

Setting	Number of Pulse Pairs	Setting	Number of Pulse Pairs
0	1	8	256
1	2	9	512
2	4	10	1024
3	8	11	2048
4	16	12	4096
5	32	13	8192
6	64	14	16384
7	128	15	32768

BLEN may be considered as a GAIN setting for the key signals, as the measurements taken are accumulated to give the final signal value. For a given SCALE setting, each increment of the BLEN setting doubles the number of measurements and therefore the signal value. Similarly the delta value used for touch detection will doubled for each increment of BLEN.

SCALE is an averaging factor which allows for efficient noise rejection in a noisy environment. For each increment in SCALE the signal value accumulated by BLEN is divided by 2.

For example, a BLEN:SCALE ratio of 6:3 (64 accumulated measurements, divided by 8) will result in signal, reference and delta being of the same magnitude as for a setting of 3:1 (8 accumulated measurements, divided by 1) but will have steadier resultant values in a noisy environment due to averaging.

5. Touch Objects

5.1 Introduction

Touch objects operate on measured signals from the touch sensor and report touch data. For example, a Key T13 object reports when a key is in detect. Table 5-1 lists the touch objects on the QT1085.

Table 5-1. Touch Objects

Object	Description
Key T13 (TOUCH_KEY_T13)	Configures a single key. See Section 5.2.

5.2 Key T13 (TOUCH_KEY_T13)

Each instance of a Key T13 object represents a key. There are 8 instances of the Key T13 object on the QT1085.

5.2.1 Configuration

Table 5-2. TOUCH_KEY_T13

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	CTRL[0 – *n*]	DISPRESS	DISREL	GUARD				RPTEN	ENABLE
1	CFG[0 – *n*]	TCHDI			AKSGRP			HYST	
2	TCHTHR[0 – *n*]	TCHTHR							

CTRL Field

ENABLE: Enables the use of this object. The object is enabled if set to 1, and disabled if set to 0. The object does not scan for touches if it is disabled, in order to conserve power.

RPTEN: Allows the object to send status messages to the host through the message processor. Reporting is enabled if set to 1, and disabled if set to 0. Events must be enabled for this bit to have an effect.

GUARD: This field determines whether the key is a guard key. Set to 1 to make the key a guard key. A guard key differs from a normal key in that detection does not cause the device to change from IDLE to ACTIVE mode. Also, a TCHAUTOCAL (Touch recal delay) is never carried out on a guard key.

DISREL: This field determines whether the key generates a message when the touch is released. Set to 1 to not generate a message on release.

DISPRESS: This field determines whether the key generates a message on touch. Set to 1 to not generate a message on touch.

CFG Field

HYST: This field controls the level of hysteresis applied to touch detections. For a touch to enter detection the touch delta must be greater than the touch threshold (TCHTHR). For a touch to leave detection the touch delta must be less than (TCHTHR – HYST). This field allows a touch to be tuned so that a hovering finger does not cause detection to flicker on and off. See Table 5-3 for the range of hysteresis levels available.

Table 5-3. Hysteresis Levels

Bits	Hysteresis Level
00	50%
01	25%
10	12.5%
11	6.25%

AKSGRP: This field configures Adjacent Key Suppression® (AKS®) between this Key T13 object and any other Key T13 objects.

AKS technology is a patented method used to detect which touch object is touched when objects are located close together. A touch in a group of AKS objects is indicated only on the object with the largest signal. This is assumed to be the intended object. Once an object in an AKS group is in detect, there can be no further detections within that group until the object is released.

Bits 2 – 4 specify which AKS groups this Key T13 object is within. The default value of 0 means that the object is in no AKS groups and the AKS feature is disabled for that particular Key T13.

Range: 0 – 7 (0 = not in an AKS group, 1 – 7 = group number for AKS)

TCHDI: To suppress false detections caused by spurious events like electrical noise, the device incorporates a detection integrator (TCHDI) counter mechanism to provide signal filtering. A per-key counter is incremented each cycle that a touch is detected. When this counter reaches the TCHDI setting limit the touch is finally declared to be present. If on any acquisition a delta is not seen to exceed the threshold level, the counter is cleared and the process has to start from the beginning.

An opposite process is applied when a key leaves detection. The counter is decremented each cycle that the delta does not exceed the threshold level, and incremented again if it does exceed the threshold. When the counter reaches zero, the touch is finally declared to be out of detect. In this case there is an additional extra cycle (that is, the number of cycles is TCHDI + 1).

Range: 0 – 7

TCHTHR Field

TCHTHR: The channel detection Touch Threshold value (TCHTHR) defines how much a channel's touch delta (that is, Signal minus the Reference) must be to qualify as a potential touch detection. The reference level is determined during calibration and adjusted using drift compensation. The final detection confirmation uses the Touch Detect Integration as described in the TCHDI field. Larger values for the threshold desensitize channels, since the signal must change more in order to exceed the threshold level. Conversely, lower thresholds make channels more sensitive.

The setting for TCHTHR for each channel depends on the amount of signal swing that occurs when a channel is touched. Thicker panels or smaller electrode geometries reduce channel sensitivity (that is, signal swing from touch). In this case smaller TCHTHR values are required to detect touch.

Range: 0 to 255

Typical: 30 to 80

5.2.2 Messages

The message data for the Key T13 object is shown in Table 5-2.

Table 5-4. Message Data for TOUCH_KEY_T13

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	REPORT ID	REPORT ID							
1	KEYSTATE								KEYSTATE

REPORT ID Field

This field contains the report ID for the message. Messages contain report IDs to allow the host to identify the type of message and its originator. Report IDs are assigned to any object that can send messages. See Section 2.4.6 on page 5 for more information on the assignment of report IDs.

KEYSTATE Field

This field indicates whether or not a key is in detect. 1 = in detect, 0 = not in detect.

6. Signal Processing Objects

6.1 Introduction

Signal processing objects process the data from other objects, for example to provide filtering operations. Table 6-2 lists the signal processing objects on the QT1085.

Table 6-1. Signal Processing Objects

Object	Description
Touch Configuration (PROCG_TOUCHCONFIG_T16)	Configures the parameters of the object. See Section 6.2.

6.2 Touch Configuration (PROCG_TOUCHCONFIG_T16)

The Touch Configuration T16 object controls how the device processes touch information.

6.2.1 Configuration

Table 6-2. PROCG_TOUCHCONFIG_T16

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	CFG	CHRGTIME				ATCHCALTHR			
1	DRIFTST	Drift suspend time							
2	TCHAUTOCAL	Touch automatic calibration							
3	TCHDRIFT	Reserved	Touch drift						
4	ATCHDRIFT	Reserved	Antitouch drift						

CFG Field

ATCHCALTHR: The standard finger recovery process is intended to allow the sensor to recover when, for example, a finger is present on the sensor during calibration and then subsequently removed resulting in an anti-touch signal delta.

The ATCHCALTHR field sets the antitouch calibration threshold. If this field is set to zero then an anti-touch recalibration will not take place. The *away from touch* signal will be compensated for only through the thermal drift mechanism.

Table 6-3. ATCHCALTHR Settings

Contents	Setting
0	100% of TCHTHR
1	50% of TCHTHR
2	25% of TCHTHR
3	12.5% of TCHTHR
4	6.25% of TCHTHR
5 – 15	Anti-touch delta count

CHRGTIME: prolongs the charge-transfer period of signal acquisition by 1 µs per count
Allows full charge transfer for keys with heavy Rs / Cx loading.

Range: 0 to 15

Typical: 0

DRIFTST Field

DRIFTST: The Drift Suspend Time (DRIFTST) setting controls the time from a touch release until the drift process is re-enabled. DRIFTST is used to restrict drift on all channels while one or more channels are activated. It defines the length of time the drift is halted after a touch detection.

This feature is particularly useful in preventing an actual touch – or simply a hovering finger – from causing untouched channels to drift. Without this feature, a sensitivity shift could be created that would ultimately inhibit any further touch detection.

DRIFTST can be configured to a value of between 0 and 255 in increments of 200 ms, where a value of 0 = disabled (no drift suspend time). This gives a range of 200 ms to 51 s.

Range: 0, 1 to 255 (in 200 ms increments)

Typical: 20

Default: 20

TCHAUTOCAL Field

TCHAUTOCAL: A prolonged (usually unintentional) contact from a foreign object may result in a touch detection for a prolonged interval. It is desirable to perform a recalibration in order to restore a touch object function. This is usually done after a time delay of some seconds.

The Touch Automatic Calibration (TCHAUTOCAL) setting controls the length of time a touch is held until it is considered false and an automatic recalibration is performed to compensate. The TCHAUTOCAL timer monitors touch detections. If a detection event exceeds the timer's setting, an automatic recalibration occurs. After the recalibration has taken place, normal functionality resumes, even if the touch object is still being contacted by the foreign object. This feature is enabled globally, but the exact mechanism depends on the object being touched:

- For a Key T13 object (see Section 5.2 on page 17) there is a counter per key, incremented while a touch remains on the key. An automatic recalibration occurs if the touch is still present at the end of the TCHAUTOCAL period. The automatic recalibration recalibrates all sensors.

TCHAUTOCAL can be disabled by setting it to zero (infinite timeout). In this case the object never autorecalibrates during a continuous detection (but the host could still command it). TCHAUTOCAL above 0 is configured in 200 ms increments.

TCHAUTOCAL is not applied to a guard channel.

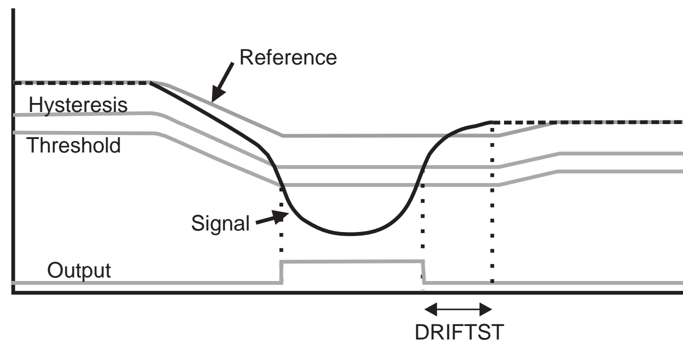
Range: 0 (infinite), 1 to 255 (in 200 ms increments)

TCHDRIFT Field

TCHDRIFT: The Touch Drift (TCHDRIFT) setting controls the ‘towards touch’ drift interval. Signals can drift because of changes in the nature of the components and materials over time and temperature. It is crucial that such drift is compensated for, otherwise false detections and sensitivity shifts can occur.

Drift compensation (see Figure 6-1) is performed by making the reference level track the raw signal at a slow rate, but only while there is no detection in effect. The rate of adjustment must be slow, otherwise legitimate detections could be ignored. The TCHDRIFT field can be configured in increments of 200 ms, where a value of 0 means 10 (2 seconds).

Figure 6-1. Thresholds and Drift Compensation



The device drift compensates using a slew-rate limited change to the reference level. The threshold and hysteresis values are slaved to this reference. Any changes to TCHDRIFT affect the rate.

If the touch is in a Key T13 object, then once the signal has crossed the respective threshold level for that object, the drift compensation mechanism ceases.

Range: 0, 1 to 127 (in 200 ms increments)

Typical: 5 to 20 (1 s to 4 s)

Default: 10 (2 s)

ATCHDRIFT Field

ATCHDRIFT: The Anti-touch Drift (ATCHDRIFT) setting controls the *away from touch* drift interval. Signals can drift because of changes in the nature of the components and materials over time and temperature. It is crucial that such drift is compensated for, otherwise false detections and sensitivity shifts can occur.

Anti-touch drift compensation is performed by making the reference level track the raw signal at a slow rate, but only while there is no detection in effect. The rate of drift should be relatively fast for ATCHDRIFT, in order to ensure that a touch can be detected. The ATCHDRIFT field can be configured in increments of 200 ms, where a value of 0 means 2 (400 ms).

The device anti-touch drift compensates using a slew-rate limited change to the reference level. The threshold and hysteresis values are slaved to this reference. Any changes to ATCHDRIFT affect the rate.

If the touch is in a Key T13 object, then once the signal has crossed the respective threshold level for that object, the drift compensation mechanism ceases.

Range: 0, 1 to 127 (in 200 ms increments)

Typical: 1 to 5

Default: 2 (400 ms)

7. Support Objects

7.1 Introduction

Support objects provide additional functionality on the device. Table 7-1 lists the support objects on the QT1085.

Table 7-1. Support Objects

Object	Description
Self Test (SPT_SELFTEST _T25)	Performs self-test routines to find faults on a touch sensor. See Section 7.2.
GPIO Configuration (SPT_GPIO_T29)	Sets up the general-purpose input/output (I/O) pins See Section 7.3.
Haptic Event (SPT_HAPTICEVENT_T31)	Configures a haptic effect to provide tactile feedback to the user. See Section 7.4.

7.2 Self Test (SPT_SELFTEST_T25)

The Self Test T25 object runs self-test routines in the device to find faults in the sense lines and electrodes. The Self Test T25 object runs a series of test sequences. As soon as the first failure is found, the test run stops and the object reports a message.

The following tests can be run:

- Signal Limit test. These test the signals from each of the touch objects on the device. These tests are run per touch object and check whether signals for enabled keys are all within the range +32 to +32,767.
- Pin Fault test: Checks for *stuck at* faults on enabled touch key pins and GPIOs which are configured as outputs.

To run a test, the CMD field is set to the code of the test to be run. The Self Test T25 object runs the test once and then stops. At the end of the test, the CMD field is cleared. If reporting is enabled (see “CTRL Field” on page 23), the Self Test also sends a report message with the result of the test (see Section 7.2.2 on page 24). The tests can be called at the following times:

- For pin fault testing: when the sense lines are not in use.
- For signal limit testing: at any time after acquisition when the signals are stable (such as not during calibration).

Note: If an error is found, the calibration command in the Command Processor T6 object should be used once the error has been cleared.

7.2.1 Configuration

Table 7-2. SPT_SELFTEST_T25

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	CTRL	Reserved						RPTEN	ENABLE
1	CMD	Test code of test to run							

CTRL Field

ENABLE: Allows the object to run tests. Tests are enabled if set to 1, and disabled if set to 0.

RPTEN: Allows the object to report test messages. Reporting is enabled if set to 1, and disabled if set to 0.

CMD Field

This field is used to send test commands to the device. Valid test commands are listed in Table 7-3.

Table 7-3. Test Commands

Code	Test Description
`0x00`	The CMD field is set to 0x00 after test completed
`0x11`	Run the pin fault test
`0x17`	Run the signal limit test
`0xFE`	Run all the tests

Writing `0x11` and `0x17` to the CMD field causes the device to run a pin fault test or signal limit test (as appropriate).

7.2.2 Messages

The Self Test T25 object reports the test results in its message data. The message data for the Self Test T25 object is shown in Table 7-4.

Table 7-4. Message Data for SPT_SELFTEST_T25

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	STATUS	Result code							
2 – 6	INFO	Result data							

STATUS Field

This field contains a result code that indicates the success or failure of the test. Valid codes are given in Table 7-5.

Note: The device must be recalibrated when all errors have been resolved.

Table 7-5. Result Codes

Code	Test Result
`0xFE`	All tests passed.
`0x11`	The test failed because of a pin fault. The INFO fields indicate the first pin fault that was detected (see Table 7-6).
`0x17`	The test failed because of a signal limit fault.

INFO Field

This field contains the result data of the test. The actual data depends on which test was run, as detailed below. Note that if a test does not generate data, the INFO field consists solely of reserved bytes.

INFO Field – Pin Fault

If the result was a pin fault, the INFO field has the data format shown in Table 7-6.

Table 7-6. Test Result Data for a Pin Fault

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2	SEQ_NUM	Test sequence number							

SEQ_NUM Field

The test sequence number of the test in which a fault is found. The sequence numbers and their meanings are listed in Table 7-7.

Table 7-7. Sequence Numbers

Bit	Description
7	1 = Pin High Fail. All pins are driven to Ground. This test detects shorts to Vdd (that is, they fail high).
6	1 = Pin Low Fail. All pins are driven to Vdd. This test detects shorts to Ground (that is, they fail low)
5	Pin type, 0 = Key, 1 = GPIO
4 – 0	Fail object instance (Key number or GPIO number)

INFO Field – Signal Limit Error

If the result was a signal limit error, the INFO field has the data format shown in Table 7-9.

Table 7-8. Signal Limit Test Result

Bit	Description
7	1 = Signal High Fail
6	1 = Signal Low Fail
5 – 0	Fail object instance (Key number)

7.3 GPIO Configuration (SPT_GPIO_T29)

A GPIO Configuration T29 object sets up the general-purpose input/output (I/O) pins. The pins can be set up as input or output. Output pins can generate a PWM signal depending on the state of the source object. There are twelve dedicated GPIO pins, plus up to four additional pins (replacing keys). There are 16 instances of the GPIO Configuration T29 object on the QT1085, one for each pin.

If a pin is not needed disable the relevant object (via the ENABLE bit in the CTRL field).

The only restriction is where a key and a GPIO share a pin; for example, Key 7 and GPIO 12. If both objects are enabled this will give a configuration error. Otherwise the 16 GPIOs can all drive different PWM signals based on input states.

GPIOs are checked and reported once per cycle, whether an ACTIVE or IDLE cycle is in progress.

An Input GPIO status is updated, and a message generated if enabled, on the following cycle after a change to the logic level applied at the pin. The maximum rate of change which can be reported is therefore equal to the applicable cycle time.

Similarly an Output GPIO status is updated during the cycle following the change in status of its control source. GPIO states are checked in ascending order, so, for example, a cascade effect where GPIOs use other GPIOs as control source will occur within a single cycle if the cascade is upwards, or over n cycles where *n* GPIOs are configured in a downward cascade. A GPIO configured as an output has some side-effects on the device power consumption.

- If any GPIO is enabled and configured as an output, then the device cannot utilize its lowest power sleep mode during LP_MODE as it must retain some high level timing functions for PWM generation.
- If the device is put into Deep Sleep mode with a GPIO configured as an output, the GPIO will retain its latest state on the pin. If the GPIO is running a PWM that is neither 0% nor 100% then the pin's state during deep-sleep is unpredictable as the PWM may be in either the low or high period of its duty cycle at the time when Deep Sleep is initiated.

7.3.1 Configuration

Table 7-9. SPT_GPIO_T29

<table>
<tr><th>Byte</th><th>Field</th><th>Bit 7</th><th>Bit 6</th><th>Bit 5</th><th>Bit 4</th><th>Bit 3</th><th>Bit 2</th><th>Bit 1</th><th>Bit 0</th></tr>
<tr><td>0</td><td>CTRL</td><td>OUTPUT</td><td>INVERT</td><td>DISOFF</td><td>DRIVELVL</td><td>TOGGLE</td><td>FADE</td><td>RPTEN</td><td>ENABLE</td></tr>
<tr><td>1</td><td>PWM</td><td colspan="4">ON PWM</td><td colspan="4">OFF PWM</td></tr>
<tr><td>2</td><td>SOURCE</td><td colspan="3">SOURCE TYPE</td><td colspan="5">SOURCE INSTANCE</td></tr>
</table>

CTRL Field

Note: For a GPIO configured as an input, only the Enable, Invert and reporting settings RPTEN and DISOFF are relevant. All other settings apply to GPIOs in Output mode only.

ENABLE: Enables the use of this object. The object is enabled if set to 1, and disabled if set to 0.

RPTEN: Allows the object to send status messages to the host through the message processor. Reporting is enabled if set to 1, and disabled if set to 0. Events must be enabled for this bit to have an effect.

FADE: If set to 0 the feature is disabled and the duty cycle seen at the GPIO output pin jumps from OFF PWM to ON PWM. If set to 1 the feature is enabled and steps from OFF PWM to ON PWM (and reverse) through intermediate PWM levels at a rate of 1 step per 15 ms.

TOGGLE: If set to 0 the state follows the source. If set to 1 the state toggles with the ON event.

DRIVELVL: This sets the drive level for outputs only. See Table 7-10 for the settings.

Table 7-10. Drive Level

Setting	Output
0	Float high (open drain operation)
1	Drive high (push-pull operation)

DISOFF: Messages can be generated for ON and OFF events. This bit, if set to 1, disables the ability to generate messages for an OFF event.

INVERT: For an input: 0 = Report logic level, 1 = Report inverted logic level. For an output, reverse *ON* and *OFF* conditions.

Table 7-11. INVERT

Setting	Input	Output
0	Report logic level	Source state is not inverted, 0 = Off 1 = On
1	Report inverted logic level	Source state is inverted, 0 = On 1 = Off

OUTPUT: Sets the direction of the GPIO pins:

1 = Output (PWM)

0 = Digital input

PWM Field

A GPIO configured as Output may generate any of 16 duty cycles from 0 to 100% by pulse-width modulation of the digital signal at the pin.

All PWM settings are implemented at a fixed frequency of 66 Hz

Specifies that the GPIO pins are to be used as PWM pins and what the duty cycle is (see Table 7-12 on page 27).

The 4-bit PWM generated by the GPIO in the ON state. The 4-bit PWM generated by the GPIO in OFF state.

Note: There may be some variance in the actual duty cycle and frequency of the PWM as the pin-state updates is given a lower process priority than touch acquisition measurements or communications.

Table 7-12. PWM Settings

Setting	Number	Setting	Number
0	0%	8	53.36%
1	6.67%	9	60.03%
2	13.34%	10	66.70%
3	20.01%	11	73.37%
4	26.68%	12	80.04%
5	33.35%	13	86.71%
6	40.02%	14	93.38%
7	46.69%	15	100%

SOURCE Field

SOURCE TYPE: The type of object which controls whether GPIO in Output mode is delivering ON PWM or OFF PWM.

Selection of an object type that does not exist on the device should generate a Config error.

Table 7-13. Commands

Setting	Control Source	
0	Host Control	Under Host Control, an output GPIO outputs the PWM at 'Off PWM'. To change the PWM seen at the pin, the value of 'Off PWM' is changed. Alternatively, the host may switch from 'Off PWM' to 'On PWM' by setting the 'Invert' bit of the CTRL field
1	Key	If the control source is 'Key' or 'GPIO' then when the source Object is 'ON' then the GPIO outputs its 'ON PWM', when the source object is 'Off' then the GPIO outputs its 'OFF PWM'. 'On' and 'Off' states of control sources may be reversed by setting the GPIO's 'Invert' bit.
4	GPIO	

SOURCE INSTANCE: The instance of an object which controls the GPIO in Output mode.

7.3.2 Messages

The message data for a GPIO Configuration T29 object is shown in Table 7-14.

Table 7-14. Message Data for SPT_GPIO_T29

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	REPORT ID	REPORT ID							
1	GPIOSTATE	Reserved						TYPE	STATE

REPORTID Field

This field contains the report ID for the message. Messages contain report IDs to allow the host to identify the type of message and its originator. Report IDs are assigned to any object that can send messages. See Section 2.4.6 on page 5 for more information on the assignment of report IDs.

GPIOSTATE Field

This field indicates whether the GPIO is an input or an output and also whether it is turned on or off.

STATE: 0 = Off / Low

1 = On / High

TYPE: 0 = Input

1 = Output

7.4 Haptic Event (SPT_HAPTICEVENT_T31)

A Haptic effect is a sequence of motor actions that provide tactile feedback to the user (that is, the feeling of a physical button press may be conveyed when a touch sensor is touched). There are 14 pre-configured effects provided by the QT1085 (see Table 7-16 on page 29).

The Haptic Event T31 object allows for configuration of the effect to be played under a given circumstance (the source condition).

The QT1085 provides eight instances of the Haptic Event T31 object, allowing for different tactile feedback effects to be played, depending on the touched key or other control.

The control source for each enabled GPIO is checked once per cycle (ACTIVE or IDLE) so the effect is started during the following cycle after the control source has made a state transition from OFF to ON.

The effect is started at an OFF to ON transition of the control source state. The effect stops either upon completion of the effect or where the trigger state (that is the On state of the control object) is negated.

7.4.1 Configuration

Table 7-15. SPT_HAPTICEVENT_T31

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	CTRL	DISSTART	DISFINISH	LOOP	–	–	–	RPTEN	ENABLE
1	EFFECT	–	EFFECT TO PLAY						
2	SOURCE	SOURCE TYPE			SOURCE INSTANCE				

CTRL Field

ENABLE: Enables the use of this object. The object is enabled if set to 1, and disabled if set to 0.

RPTEN: Allows the object to send status messages to the host through the message processor. Reporting is enabled if set to 1, and disabled if set to 0. Events must be enabled for this bit to have an effect.

LOOP: This enables the effect to be continuously repeated for a stated trigger condition. When set to a 1 the effect can be repeated for a long as the trigger condition exists.

DISFINISH / DISSTART: You may not want to generate a message at the start or finish of an operation.

- To not generate a message to the host at the finish, set DISFINISH to 1.
- To not generate a message to the host at the start, set DISSTART to 1.

Note: If RPTEN is disabled then the DISFINISH / DISSTART settings have no effect.

EFFECT Field

EFFECT TO PLAY: This designates the ID of the desired effect. See Table 7-16) for a preconfigured selection of haptic effects available on the device.

Table 7-16. Preconfigured Haptic Effects

ID	Type of Effect
1	Strong Click
2	Strong Click at 60% strength
3	Strong Click at 30% strength
4	Sharp Click
5	Sharp Click at 60% strength

Table 7-16. Preconfigured Haptic Effects (Continued)

ID	Type of Effect
6	Sharp Click at 30% strength
7	Soft Bump
8	Soft Bump at 60% strength
9	Soft Bump at 30% strength
10	Double Click
11	Double Click at 60% strength
12	Triple Click
13	Soft Buzz
14	Strong Buzz

SOURCE Field

SOURCE INSTANCE: This is the number of instances of the object referred to in the Source Type. The options for the Source Instance depend on the Source Type (see Table 7-17 on page 31):

- If the Source Type is Touch Key, then Source Instance is 0 to 7 (number of keys).
- If the Source Type is GPIO, then Source Instance is 0 to 15 (number of GPIO lines).

SOURCE TYPE: The type of object which triggers the haptic event to play (see Table 7-17 on page 31).

For a Source type *Key* or *GPIO* the haptic event is triggered by an *Off* to *On* transition of the source object state.

Selection of an object type that does not exist on the device generates a Config error.

To play a haptic under direct host control:

1. Set the Source Type to 0.
2. Set the Source Instance to any non-zero value. The effect will play once, generating Start and Stop messages, if enabled.
3. To play the effect again, reset the haptic event by setting the Source Instance to 0. After a *reset* of the haptic event, the host must wait at least 1 cycle time (Active or Idle) for the reset to be applied before the effect may be replayed.
4. Repeat steps 1 and 2.

Table 7-17. Source Type

Setting	PWM
0	Host Control
1	Key
2 – 3	Reserved
4	GPIO
5 – 7	Reserved

7.4.2 Configuration Checks

A Haptic Event T31 object causes a configuration error in the following circumstances:

- If the selected effect does not exist on the device.
  For the QT1085 this means that if a preconfigured effect number greater than 14 is selected.
- If the selected source object does not exist.
- If the selected source object is not enabled.

A configuration check may determine that a configuration error has occurred (for example, if a setting is set outside of its allowed range or a conflict has occurred between two settings) [1]. This is signaled to the host (see Section 7.4.3 on page 31). The device halts until the error has been corrected. To fix the error, check that all the object settings are within their allowed limits, as stated in the field descriptions.

Checks will be carried out on enabled objects at power-on or reset and when configurations are changed.

Note: On the QT1085, checks are carried out on all enabled objects when any configuration is changed.

7.4.3 Messages

The message data for a Haptic Event T31 object is shown in Table 7-18.

Table 7-18. Message Data for SPT_HAPTICEVENT_T31

Byte	Field	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	REPORT ID	REPORT ID							
1	HAPTICSTATE	–							STATE

1. While the object is disabled, however, potential configuration errors are allowed.

REPORT ID Field

REPORT ID: This field contains the report ID for the message. Messages contain report IDs to allow the host to identify the type of message and its originator. Report IDs are assigned to any object that can send messages. See Section 2.4.6 on page 5 for more information on the assignment of report IDs.

HAPTICSTATE Field

STATE: This indicates whether or not the haptic effect is still playing or has finished.

0 = Inactive (finished)

1 = Active (playing).

Note: If a haptic effect is playing when the device enters Deep Sleep mode, the effect is paused. When Deep Sleep mode is negated (by writing a nonzero setting to the appropriate field in the Power Configuration T7 object) the haptic effect is resumed and continues until either the effect is complete or the control source's ON state is negated.

Appendix A. Checksum Calculation

A.1 24-bit CRC

A.1.1 Introduction

The device uses a 24-bit cyclic redundancy check (CRC) in the following place:

- To check the integrity of the main Information Block for the device

This CRC checksum allows the host to be confident that the memory map layout has been read over the communications bus correctly.

Note: The C code in this appendix uses the type declarations `uint8_t`, `uint16_t` and `uint32_t` for 8-bit, 16-bit and 32-bit integers respectively.

A.1.2 24-bit CRC Algorithm

Each checksum is generated by running an algorithm which takes two new bytes of data and combines them with the current checksum to produce a new checksum value.

The sample code below shows how to calculate the checksum for each 16-bit word in a byte stream.

```
uint32_t crc24(uint32_t crc, uint8_t firstbyte, uint8_t secondbyte)
{

        static const uint32_t crcpoly = 0x80001b;
        uint32_t result;
        uint16_t data_word;

        data_word = (uint16_t)((uint16_t)(secondbyte << 8u) | firstbyte);

        result = ((crc<<1u) ^ (uint32_t)data_word);

        if(result & 0x1000000)  // If bit 25 is set
        {
                result ^= crcpoly;// XOR result with crcpoly
        }

        return result;
}
```

The checksum routine is called iteratively to calculate the CRC two bytes (16-bit word) at a time. This means that two bytes must be read from the device before performing each iteration of the checksum. Therefore, if an odd number of bytes is read from the device, the host checksum code should add a zero byte to the end of the byte stream to make the sequence even. For example, if the following stream of 7 bytes is received from the device:

byte1 — byte2 — byte3 — byte4 — byte5 — byte6 — byte7

The checksum could be calculated as follows:

```
uint32_t CRC = 0;
CRC = crc24(CRC, byte1, byte2)
CRC = crc24(CRC, byte3, byte4)
CRC = crc24(CRC, byte5, byte6)
CRC = crc24(CRC, byte7, 0)        /* <- zero added for the last checksum */

CRC = CRC & 0x00FFFFFF;           /* <- mask the 32-bit result to 24-bit */
```

An alternative method of calling the CRC calculation function is to use a loop, as shown in Section A.1.3

Table A-1 shows an example block of eight bytes and the CRCs these generate. The bytes consist of seven data bytes plus an additional zero byte to even up the count. You can use the data in this table to verify the validity of any coded CRC routine.

Table A-1. Example CRC Calculations for 24-bit CRC

Byte Pairs		CRC Calculation Result
First Byte	Second Byte	
0x0	0xFF	0xFF00 (Intermediate CRC – partial calculation)
0x11	0xEE	0x11011 (Intermediate CRC – partial calculation)
0x22	0xDD	0x2FD00 (Intermediate CRC – partial calculation)
0x33	0xCC	0x53633 (Intermediate CRC – partial calculation)
0x44	0xBB	0xAD722 (Intermediate CRC – partial calculation)
0x55	0xAA	0x150411 (Intermediate CRC – partial calculation)
0x66	0x99	0x2A9144 (Intermediate CRC – partial calculation)
0x77	0x88	0x55AAFF (Intermediate CRC – partial calculation)
0x88	0x77	0xAB2276 (Intermediate CRC – partial calculation)
0x99	0x66	0xD6226E (Intermediate CRC – partial calculation)
0xAA	0x55	0x2C116D (Intermediate CRC – partial calculation)
0xBB	0x44	0x586661 (Intermediate CRC – partial calculation)
0xCC	0x33	0xB0FF0E (Intermediate CRC – partial calculation)
0xDD	0x22	0xE1DCDA (Intermediate CRC – partial calculation)
0xEE	0x11	0x43A841 (Intermediate CRC – partial calculation)
0xFF	0x00	0x87507D (Expected CRC – final calculation)

A.1.3 Information Block Checksum

The checksum for the Information Block should be calculated by the host on start-up when the Information Block is first read. This should be compared to the checksum value at the end of the Information Block. If there is a mismatch, an error has occurred.

The following code shows how to use the example `crc24()` function given in Section A.1.2 on page 33 to calculate the CRC checksum for the Information Block.

```
uint32_t crc = 0;                    /* Calculated CRC */
uint16_t crc_area_size;              /* Size of data for CRC calculation */
uint8_t *mem;                        /* Data buffer */
uint8_t i;
uint8_t status;

/* 7 bytes of version data, 6 * NUM_OF_OBJECTS bytes of object table. */
crc_area_size =                                         ID_INFORMATION_SIZE +
                                                        info_block-
>info_id.num_declared_objects *
                                                        OBJECT_TABLE_ELEMENT_SIZE;

mem = (uint8_t *) malloc(crc_area_size);
if (mem == NULL)
{
       /* Handle error */
}

/*
 * Read the data using a function written for this purpose
 * Here, it is assumed that the function is named read_mem()
 * and that it returns a status code with the value READ_MEM_OK.
 * It is assumed to have the following prototype:
 * uint8_t read_mem( uint16_t Address, uint8_t ByteCount, uint8_t *Data )
*/
status = read_mem(0, crc_area_size, mem);

if (status != READ_MEM_OK)
{
       /* Handle error */
}

/*
 * Call the CRC function crc24() iteratively to calculate the CRC,
 * passing it two bytes at a time.
*/
i = 0;
while (i < (crc_area_size - 1))
{
       crc = crc24(crc, *(mem + i), *(mem + i + 1));
       i += 2;
}

/*
 * Call the crc24() with the final byte,
 * plus an extra 0 value byte to make the sequence even.
*/
crc = crc24(crc, *(mem + i), 0);

free(mem);

/* Final result */
crc = (crc & 0x00FFFFFF);                               /* <- mask the 32-bit
result to 24-bit */
```

A.2 8-bit CRC

A.2.1 Introduction

An 8-bit CRC can be used in the following place:

- To check the integrity of data in messages from the Message Processor when receiving messages (see Section A.2.3 on page 36).

A.2.2 8-bit CRC Algorithm

The sample code below shows how to calculate the 8-bit checksum.

```
uint8_t crc8(unsigned char crc, unsigned char data)
{
        static const uint8_t crcpoly = 0x8c;
        uint8_t index;
        uint8_t fb;
        index = 8;

        do
        {
                fb = (crc ^ data) & 0x01;
                data >>= 1;
                crc >>= 1;
                if (fb)
                        crc ^= crcpoly;
        } while (--index);

        return crc;
}
```

A.2.3 Reading the CRC for the Message Processor Data

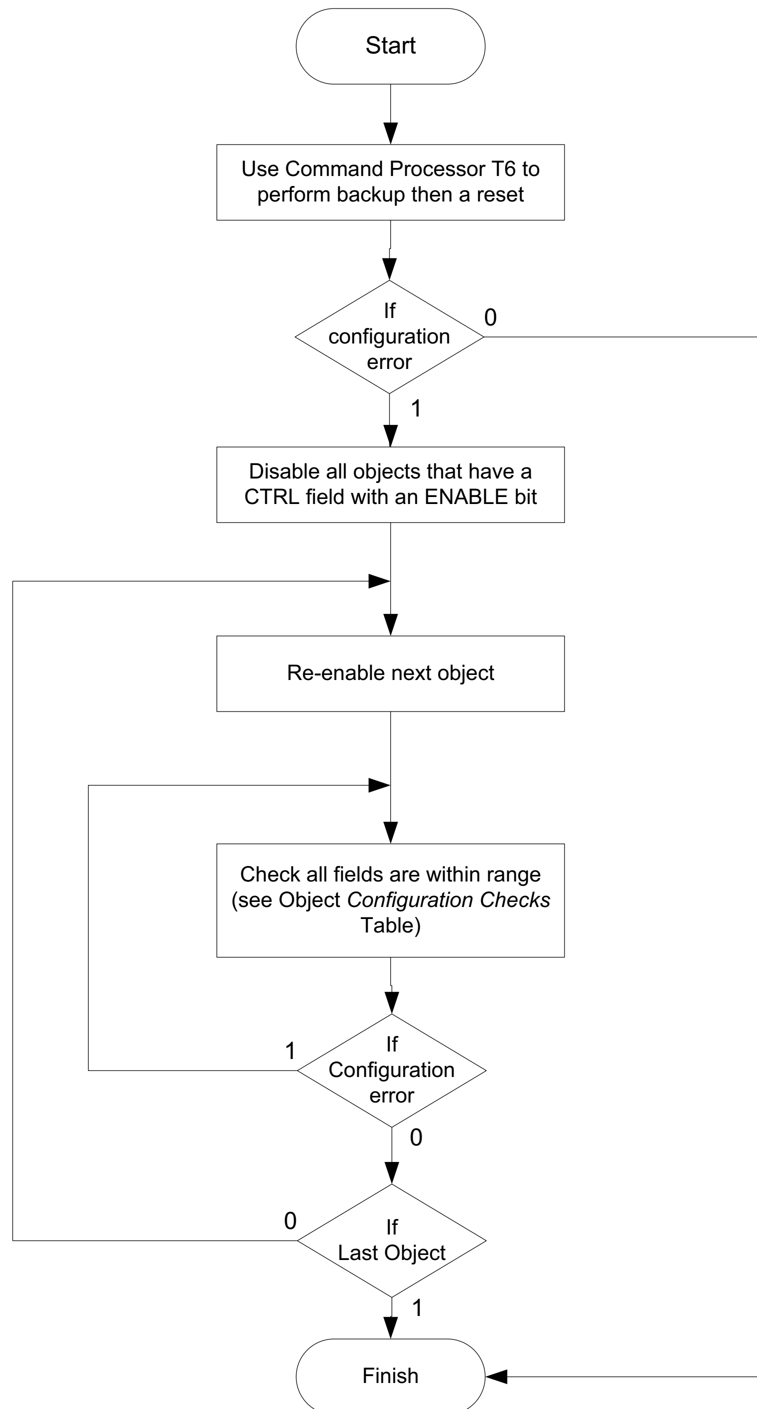
The following code shows how to use the example `crc8()` function given in Section A.2.2 on page 36 to calculate the CRC checksum for the data in the Message Processor T5 object.

```
uint8_t crc = 0;
uint8_t data_in;

for(i=0; i<MESSAGEPROCESSOR_SIZE; i++)
{
        data_in = read_byte();
        crc = crc8(crc, data_in);
}

if(crc == 0)
{
        /* CRC is OK - do something appropriate */
        crc_pass();
}
else
{
        /* CRC failed - handle error */
        crc_fail();
}
```

Appendix B. Locating Configuration Errors

Figure B-1 shows the algorithm for locating configuration errors. See Section 2.6 on page 7 for more information on the configuration checks performed by the device.

Figure B-1. Algorithm for Locating Configuration Errors



Associated Documents

- Datasheet: *AT42QT1085 – QTouch 8-key Touch Sensor IC*

Revision History

Revision Number	History
Revision AX – June 2011	Initial release
Revision BX – December 2011	Firmware revision 10.AA. Various amendments to existing objects and deletion of others.
Revision C – May 2013	Amendments and clarifications to text Applied new template

Notes

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 Roa
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel München GmbH
Business Campus
Parkring 4
D-85748 Garching bei München
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

© 2013 Atmel Corporation. All rights reserved. / Rev.: 9626C–AT42–05/3013

Atmel®, Atmel logo and combinations thereof, Adjacent Key Suppression®, AKS®, QTouch® and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be registered trademarks or 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.