



## Introduction

This document is intended for touch sensing application designers who require an overview of low power modes in the STM8TL5xxx devices. It describes how to use the general features of these devices in low power modes by explaining the differences between the various modes. It focuses on how to reduce consumption when using the ProxSense peripheral and demonstrates how this is managed by the STM8TL5x STMTouch Library in addition to giving some code examples.

This application note is not intended to replace the STM8TL5xxx datasheet. All values given in this document are for guidance only. For guaranteed values, please refer to the STM8TL5xxx datasheet.

**Table 1. Applicable products**

Type	Product sub-class
Microcontroller	STM8TL5xxx

# Contents

- 1 Power consumption factors . . . . . 4**
- 2 Power supply . . . . . 6**
  - 2.1 Internal supply structure . . . . . 7
- 3 Clock management . . . . . 8**
  - 3.1 Clock system overview . . . . . 8
  - 3.2 Default clock source . . . . . 8
  - 3.3 Clock configuration and power management . . . . . 8
  - 3.4 Clock selection versus power consumption . . . . . 8
- 4 Low power modes . . . . . 10**
  - 4.1 Flash memory . . . . . 10
  - 4.2 Overview of low power modes . . . . . 10
  - 4.3 Slowing down the clock frequency . . . . . 11
  - 4.4 Peripheral clock gating (PCG) . . . . . 11
  - 4.5 Execution from RAM . . . . . 11
  - 4.6 Wait mode . . . . . 12
    - 4.6.1 Entering Wait mode . . . . . 12
    - 4.6.2 Exiting Wait for interrupt mode . . . . . 13
    - 4.6.3 Exiting Wait for event mode . . . . . 13
  - 4.7 Halt mode . . . . . 13
    - 4.7.1 Entering Halt mode . . . . . 14
    - 4.7.2 Exiting Halt mode . . . . . 14
  - 4.8 Active-halt . . . . . 14
    - 4.8.1 Entering Active-halt mode . . . . . 14
    - 4.8.2 Exiting Active-halt mode . . . . . 14
  - 4.9 Activation level control . . . . . 15
- 5 General power management tips . . . . . 16**
  - 5.1 Choosing the optimal low power mode for your application . . . . . 16
  - 5.2 GPIO initialization . . . . . 16
  - 5.3 Dynamic control of pull-up resistor . . . . . 17

---

5.4	Waiting loops/delays . . . . .	17
5.5	Minimizing power consumption . . . . .	18
<b>6</b>	<b>ProxSense and low power modes . . . . .</b>	<b>19</b>
6.1	Possible CPU low power modes combined with ProxSense . . . . .	19
6.2	Main factor of the ProxSense acquisition consumption . . . . .	19
6.3	Low power features in the ProxSense peripheral . . . . .	20
6.3.1	LOW_POWER bit . . . . .	20
6.3.2	Stabilization time . . . . .	20
6.3.3	Bias parameter . . . . .	20
6.3.4	Inactive state . . . . .	21
6.3.5	Receiver disabling . . . . .	21
<b>7</b>	<b>Low power mode management by the STM8TL5x STMTouch Library</b>	<b>22</b>
7.1	Configuration available in the stm8_tsl_conf.h file . . . . .	22
7.1.1	Acquisition time . . . . .	22
7.1.2	LOW_POWER bit . . . . .	23
7.1.3	Stabilization time . . . . .	23
7.1.4	Bias parameter . . . . .	23
7.1.5	Receiver configuration when disabled . . . . .	24
7.1.6	Transmitter configuration when disabled . . . . .	24
7.2	Practical code example . . . . .	24
7.2.1	Low power management with all acquisition banks . . . . .	24
7.2.2	Very low power management with proximity detection . . . . .	25
<b>8</b>	<b>Conclusion . . . . .</b>	<b>28</b>
<b>9</b>	<b>Revision history . . . . .</b>	<b>29</b>

# 1 Power consumption factors

The STM8TL5xxx microcontrollers are digital logic devices using the complementary metal oxide semiconductor (CMOS) technology. In these type of devices, power consumption is a sum of:

- Static power (mainly caused by transistor polarization and leakage)
- Dynamic power which depends on the supply voltage and the clock frequency

Dynamic power is calculated using [Equation 1](#).

## Equation 1

$$\text{Dynamic power} = C \times V^2 \times f$$

where:

- C is the CMOS load capacitance
- V is the supply voltage
- f is the clock frequency

Static consumption is negligible compared to dynamic consumption when the clock is running. In some low power modes, when no clock is running, static consumption is the main consumption source.

Total consumption is a sum of static and dynamic consumption as given by [Equation 2](#).

## Equation 2

$$I_{DD} = f \times I_{\text{DynamicRun}}[\mu\text{A}/(\text{MHz})] + I_{\text{Static}}[\mu\text{A}]$$

where:

- $I_{DD}$  is the supply current
- $I_{\text{DynamicRun}}$  is the current consumption dependent on the CPU frequency
- $I_{\text{Static}}$  is the current consumption independent on the CPU frequency

Consequently, power consumption depends on:

- **The microcontroller unit (MCU) chip size**  
This includes the technology used, the number of transistors, and the analog features/peripherals embedded and used in the application.
- **The MCU supply voltage**  
The amount of current used in CMOS logic is directly proportional to the square of the power supply voltage ( $V^2$ ). Thus, power consumption may be reduced by lowering the MCU supply voltage. This is less critical for STM8TL5xxx devices than for other microcontrollers, as an internal voltage regulator is used. However, the MCU supply voltage could have an impact on the remaining components on the board.
- **The clock frequency**  
Power consumption may be reduced by decreasing the clock frequency when fast processing is not required by the application.
- **The number of active peripherals or MCU features used (such as timers, communication peripherals, watchdogs, ProxSense, etc.)**  
The greater the number of active peripherals or features, the greater the amount of power consumed.
- **The operating mode**  
Power consumption depends on which mode a particular application is running (example: central processing unit (CPU) on/off, oscillator on/off). For an application powered by a battery, the consumption is very important. Usually, the average consumption should be below a certain target value to ensure an optimum battery lifetime. This means that an application can consume more for short periods of time and keep its average current consumption below the target value.



## 2.1 Internal supply structure

STM8TL5xxx devices operate from 1.65 V up to 3.6 V, when connected to one pair of supply pins. There are no dedicated supply pins for the analog voltage domain. It is recommended to use a decoupling ceramic capacitor placed close to the supply pins.

- In Run and Wait modes, both MVR and LPVR provide the  $V_{CORE}$
- In Halt/Active-halt modes, the LPVR is automatically used while the MVR is switched off by the system in order to reduce current consumption.

## 3 Clock management

### 3.1 Clock system overview

The 16 MHz high-speed internal RC oscillator (HSI) is the only clock source that can be used to drive the system clock. The 38 kHz low-speed internal RC oscillator (LSI) is only used to supply the auto-wakeup unit (AWU) and watchdog.

Each peripheral clock can be switched on or off independently, in order to optimize power consumption when the peripheral is not used. This is done by using the peripheral clock gating (PCG) feature. See the “Clock control (CLK)” section of the STM8TL5xxx microcontroller family reference manual (RM0312) for more details.

### 3.2 Default clock source

The default clock source after reset is HSI/8. The user can then switch the clock to different frequencies by choosing the prescaler (/1, /2, /4 or /8) for the internal RC 16 MHz (HSI) clock through the HSIDIV[1:0] bits in the Clock divider (CLK\_CKDIVR) register.

### 3.3 Clock configuration and power management

In addition to the flexibility of the clock sources, different complementary clock configurations and features are available to optimize the power consumption of the device:

- Each peripheral clock can be switched on/off through the Peripheral clock gating registers 1 and 2 (CLK\_PCKENRx).
- System clock dividers from 1 to 8 (HSIDIV[1:0] bits in the CLK\_CKDIVR register) are available.

*Note:* System clocks are used to supply both CPU and peripherals.

The STM8TL5xxx is focused on low consumption. This is why all peripheral clocks are gated by default. Before accessing any peripheral register, it is mandatory to enable the clock for the given peripheral.

### 3.4 Clock selection versus power consumption

The selected clock type and speed is one of the major factors influencing power consumption of the MCU (see [Section 1: Power consumption factors](#)). Total consumption for the STM8TL5xxx devices is given by [Equation 3](#).

#### Equation 3

$$I_{DD(STM8TL5x)} = f \times 150[(\mu A)/MHz] + 215[\mu A]$$

*Note:* The values given in [Equation 3](#) are measured with all peripherals disabled.



Slowing down the clock decreases the immediate consumption but, often this is not the ideal solution. By slowing down the clock, the CPU performance is also reduced and a longer time is required to perform an action or computation. If we consider the average consumption, it might be better to use the highest available clock speed to perform the required operation, and then force the MCU into one of the low power modes (like Active-halt mode) for the remaining time frame. This should be taken into account during the design of application flowcharts.

## 4 Low power modes

By default, after a reset, the microcontroller is in Run mode. The default CPU clock is 2 MHz (HSI/8). Several low power modes are available to save power when the CPU is not used for a standard operation (for instance: waiting for an external event). It is up to the user to select the mode that gives the best compromise between low power consumption, short startup time, good peripheral functionality, and availability of wakeup sources. Those power modes are listed in [Section 4.2: Overview of low power modes](#).

Power consumption in Run and Wait modes can be reduced by one of the following means:

- Slowing down the system clocks
- Executing code from RAM
- Gating the clocks to the peripherals when they are not used

### 4.1 Flash memory

On STM8TL5xxx devices, the Flash is automatically switched off when Halt or Active-halt mode is entered.

### 4.2 Overview of low power modes

The STM8TL5xxx devices feature the following main low power modes:

- Wait mode: CPU stopped and peripherals kept running
- Active-halt mode: CPU stopped; ProxSense (PXS), AWU, and independent watchdog (IWDG) kept running if they are already enabled.
- Halt mode: CPU and peripheral clocks stopped

In all low power modes, the general purpose I/Os continue to actively drive outputs.

The MCU can be woken up from these low power modes by specific interrupts, including through the ProxSense and incoming communications on the serial peripheral interface (SPI), inter-integrated circuit (I2C), and/or universal synchronous/asynchronous receiver transmitter (USART). Refer to the interrupt mapping table in the STM8TL5xxx datasheet.

[Table 2](#) lists the STM8TL5xxx low power modes, shows the basic behavior of STM8TL5xxx devices, and demonstrates the influence of different low power modes on the CPU, peripherals and consumption.

**Table 2. Low power mode summary**

Low power modes	Entry instruction	Functions						Low power mode consumptions
		CPU	Peripherals	HSI	LSI	Flash	RAM	STM8TL5xxx typical values @ 3 V/25°C
Wait	WFE or WFI	Off	Can be enabled	On	On	On	On	500 $\mu$ A
Active-halt with ProxSense	Halt	Off			On	Off	On	600 $\mu$ A <sup>(1)</sup>
Active-halt with AWU					On	Off	On	1.0 $\mu$ A
Halt					Off	Off	On	0.4 $\mu$ A

1. Value during an acquisition (see the STM8TL5xxx datasheet) with 1 transmitter and 1 receiver.

### 4.3 Slowing down the clock frequency

In Run mode, choosing the clock frequency is very important to ensure the best compromise between performance and consumption. The selection is done by programming the prescaler registers. These registers can also be used to slow down the peripherals before entering a low power mode.

### 4.4 Peripheral clock gating (PCG)

Peripheral clock gating (PCG) can be used for additional power saving. This can be done at any time by selectively enabling or disabling the clock connection to individual peripherals through the CLK\_PCKENRx registers. These settings are effective both in Run and Wait modes.

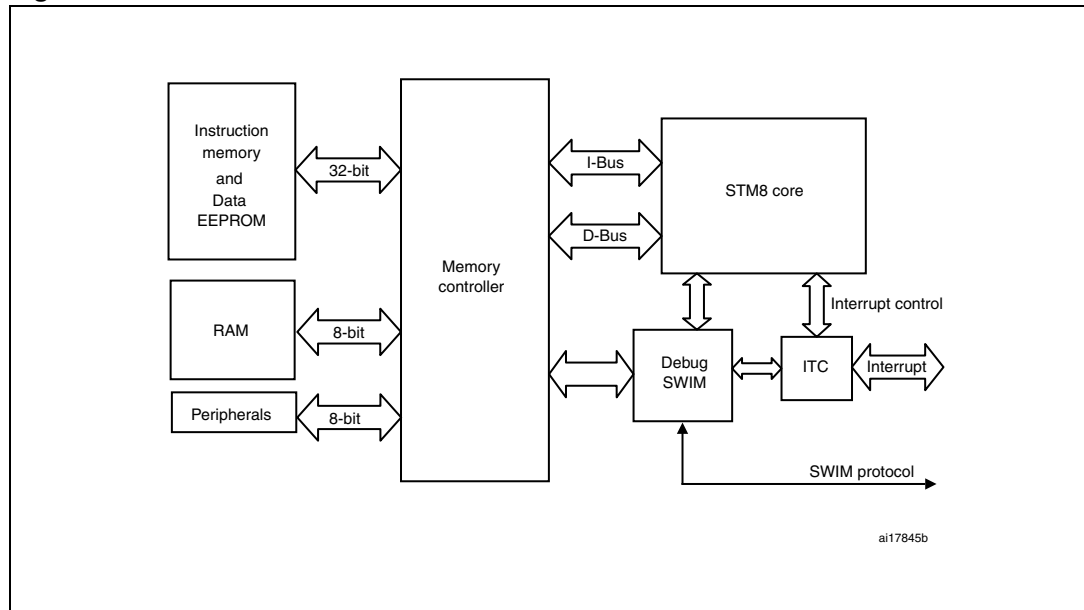
*Note:* As all peripheral clocks are gated by default after reset on STM8TL5xxx devices, it is mandatory to enable the clock for a given peripheral before accessing any peripheral register.

### 4.5 Execution from RAM

The code can be executed from RAM in order to save power consumption. However, due to the difference between the size of the data and the instruction bus from Flash (32-bit) and RAM (8-bit), performance is degraded when executed from RAM. It is important to take into account the performance and the ratio between Run/Halt operations.

*Figure 2* shows the different bus widths for Flash and RAM memories.

**Figure 2. STM8TL5xxx architecture**



1. Legend: I-Bus = instruction bus, D-Bus = databus, SWIM = single wire interface module; ITC = interrupt controller

## 4.6 Wait mode

The STM8TL5xxx devices support two different Wait modes: Wait for interrupt (WFI) and Wait for event (WFE). Both modes are designed to reduce STM8TL5xxx device power consumption by switching off the core when it is not used. Wait mode is mainly used when the STM8TL5xxx is waiting for an external or internal event so that the program execution can continue.

The polling loop on an associated peripheral flag can be efficiently replaced by a wait instruction (WFI() or WFE()) after having correctly configured the interrupt related to this flag or the event in the WFE register.

Wait mode can be combined with peripheral clock gating and a low-speed clock source to further reduce the power consumption of the device.

Wait mode also offers the lowest wakeup time which is interesting for applications requesting a fast response time.

### 4.6.1 Entering Wait mode

Wait mode is entered by executing the WFI or WFE assembly instruction. This stops the CPU, but other peripherals and the interrupt controller can continue to run. When entering Wait mode, the global interrupts are automatically enabled.

- Before entering WFI mode, at least one interrupt must be enabled.
- Before entering WFE mode, at least one event source must be enabled.

### 4.6.2 Exiting Wait for interrupt mode

When an internal or external interrupt request occurs, the CPU wakes up from Wait mode, serves the interrupt routine and resumes processing.

The following list gives examples of peripherals or features with interrupts having exit-from-wait capability:

- ProxSense
- I2C
- USART
- SPI
- AWU
- External interrupt
- Timers

Refer to the STM8TL5xxx reference manual (RM0312) for more details.

### 4.6.3 Exiting Wait for event mode

When an internal or external event request occurs, the CPU wakes up from Wait mode and resumes processing.

The following list gives examples of peripherals or features with events having exit-from-wait capability:

- ProxSense
- I2C
- USART
- SPI
- External interrupt
- Timers

If an interrupt occurs during Wait for event mode, the related interrupt service routine is executed. After this routine, the MCU goes back to Wait for event mode.

Refer to the STM8TL5xxx reference manual (RM0312) for more details.

## 4.7 Halt mode

In this mode the system clock is stopped. This means that the CPU and all the peripherals requiring clocks are disabled, except for the following cases:

- The HSI clock is not stopped if used by the SWIM
- The system clock source is not stopped if a Flash/Data EEPROM write operation is in progress.
- The LSI clock is not stopped if used by the SWIM, by the IWDG or if the "IWDG\_HALT" option bit is disabled.

In Halt mode, none of the peripherals is clocked and the digital part of the MCU consumes almost no power.

### 4.7.1 Entering Halt mode

The MCU enters Halt mode when a HALT instruction is executed. Before executing a HALT instruction, the application must clear any pending peripheral interrupt by clearing the interrupt pending bit in the corresponding peripheral configuration register. Otherwise, the HALT instruction is executed but the MCU wakes up immediately and the program execution continues.

### 4.7.2 Exiting Halt mode

Wakeup from Halt mode is triggered by an external interrupt. This wakeup is sourced by a general purpose I/O port configured as an interrupt input or by an alternate function pin capable of triggering a peripheral interrupt.

The system clock is then restarted at the last selected clock source before the system enters Halt mode.

In an interrupt-based application, where most of the processing is done through the interrupt routines, the main program may be suspended by setting the activation level bit (AL) in the Global configuration (CFG\_GCR) register. Refer to [Section 4.9: Activation level control on page 15](#).

## 4.8 Active-halt

In Active-halt mode, the main oscillator, the CPU, and almost all peripherals are stopped. Only the LSI RC oscillator runs to drive the SWIM and IWDG, if enabled.

### 4.8.1 Entering Active-halt mode

The user can enter this mode by a HALT instruction, once auto-wakeup (AWU) is enabled or a ProxSense acquisition is ongoing.

ProxSense acquisition must have started or the AWU must have been correctly configured and enabled, before executing the HALT instruction.

### 4.8.2 Exiting Active-halt mode

Wakeup from Active-halt mode is triggered by an external interrupt, a ProxSense interrupt or an AWU interrupt.

The system clock is then restarted at the last selected clock source before the system enters Halt mode.

In an interrupt-based application, where most of the processing is done through the interrupt routines, the main program may be suspended by setting the activation level bit (AL) in the Global configuration (CFG\_GCR) register. Refer to [Section 4.9: Activation level control](#).

## 4.9 Activation level control

STM8TL5xxx devices support an automatic activation level control feature. Consequently, the user can configure the MCU so that it directly returns to a low power mode after it has been woken up from such a mode through interrupts.

In an interrupt-based application, where most of the processing is done through the interrupt routines, the main program may be suspended by setting the activation level bit (AL) in the CFG\_GCR register. Setting the AL bit in the CFG\_GCR register (see [Section 4.9: Activation level control on page 15](#)) causes the CPU to return to Low power mode after exiting an interrupt service routine without restoring the main execution context when the IRET instruction is executed. This saves power by removing both the save/restore context activity and the need for a main software loop execution for power management (in order to return to low power mode).

To return to the main loop, the AL bit has to be cleared by software. This must be done at least two clock cycles before the IRET instruction is executed.

## 5 General power management tips

### 5.1 Choosing the optimal low power mode for your application

Different low power modes can be selected depending on your application:

- Applications powered by a battery where the MCU is in sleep mode most of the time:
  - If the MCU is woken up due to external events, no time tracking is necessary and power consumption has to be as low as possible. Halt mode is advised to extend battery life as much as possible.
  - Active-halt mode with AWU clocked by the LSI is advised if the application does not only depend on external events, but needs a non accurate periodic wakeup.
- Applications powered by a battery where the MCU is awake most of the time:
  - Active-halt mode is advised if the MCU has to perform a few periodic actions using no peripheral except the ProxSense.
  - Wait mode is advised if at least one peripheral should be enabled all the time and an interrupt or event can wake up the MCU.
- Applications supplied by mains but where consumption is critical:
  - Run mode, with a clock prescaler adapted to the application requirement, is advised if the MCU has to run all the time and low power modes cannot be used.

### 5.2 GPIO initialization

By default, the I/Os are configured as floating input.

It is important to change the configuration of all I/Os that are not connected to defined logic signals so as to obtain one of the following configurations:

- Input configuration with pull-up
- Output configuration with a low (or high) logic level

Otherwise, an increased consumption is generated by noise, as the internal Schmitt triggers detecting this noise are toggling.

Floating I/Os could generate additional consumption in the range of a few 10  $\mu$ A.

In 28-pin packages, the unbounded I/Os are connected to a defined level by factory option configuration.

In 20-pin packages, the following pins PA6, PA7, PD2, PD3 and PD7 must be configured in output push-pull low level by the application code in order to avoid extra consumption on these unbounded pins.



### 5.3 Dynamic control of pull-up resistor

Many applications have buttons which are used as a user interface. These buttons are connected to I/Os that are connected to  $V_{SS}$  once they are pressed. An internal pull-up is used in order to have a defined logic level on these I/Os when the button is off (not pressed).

Once the button is pressed, the I/O is grounded and it generates an additional current of ~40 – 70  $\mu\text{A}$ , drawn from the power supply. This current is negligible if the application is in Run mode, but becomes very important for applications which run mainly in low power modes.

For such low-power applications, the pull-up can be dynamically controlled. Once a button is pressed, the related service routine is executed. This routine disables the pull-up and the interrupt, in order to save consumption while the button is being pressed.

As the application has to check the button status regularly, the pull-up is enabled again before any checks are made. If the button is continuously pressed, the process is repeated. Once the button release is detected, the pull-up remains on.

For this task, an I/O can be left floating for a short time frame, given by a check period, generating an extra consumption in the range of ~10  $\mu\text{A}$ . Therefore, the total current consumption for a button-press operation is lower than without such dynamic control.

### 5.4 Waiting loops/delays

In some cases, the application has to wait for an input from the user, communication peripheral, or another external unit.

If there is no useful code to execute while waiting for an input, it is recommended to avoid using either a typical delay loop or a polling loop.

- Typical delay loop:

```
(delay = 0; delay < 0xFFFF; delay++)
_asm ("nop");
```

- Polling loop:

```
SPI_DR = data; // start communication
while (!(SPI_SR & SPI_SR_TXE)); // wait for TXE bit set
```

A Wait mode woken up by an interrupt on the SPI transmitter empty (SPI\_SR\_TXE flag) is advised by the following sequence:

- Enable TXE interrupt:

```
SPI_ICR |= SPI_TXIE; // enable interrupt for Tx empty
```

An event is then generated after each byte is sent. The next data can be loaded in the Tx buffer as follows:

```
SPI_DR = data; // start communication
_asm ("wfi"); //or wfi() if you include STM8TL5x.h file in your
project
```

The sequence described above uses WFI mode to wait for Tx buffer empty. A delay loop can be replaced by a Wait mode woken up by a programmed timer interrupt or better an Active-halt with AWU if no other peripheral is running. This can also be used for other communication lines. For external sources (like interrupt from an I/O port), Halt mode can also be considered.

## 5.5 Minimizing power consumption

The following recommendations can help the user choose the right configurations to minimize power consumption of the device in the application:

- Switch off all unused peripherals (peripherals are switched off by default) and use the peripheral clock gating (PCG) feature (through the CLK\_PCKENRx registers) to disable the clock provided to the peripherals when not in use (these clocks are disabled by default).
- All unused pins must be connected to a defined logic level. One option is to configure them as output low level. Make sure there are no unused I/O pins configured as a floating input. This needlessly leads to high consumption.
- Use Wait mode if external wakeup capability is needed in low power mode and if some peripherals have to remain active.
- Use the appropriate  $V_{DD}$  value for the application. The higher the  $V_{DD}$  value, the more power is consumed. Thanks to the internal regulator, which supplies internal structures of the MCU, there is no major difference in the MCU consumption but, the difference could be visible at the application level, for example, due to the current flowing through the pull-up resistors out of the MCU.
- Use Active-halt and Halt modes as much as possible. To achieve this, shorten the time spent in Run mode, for example, by using the highest clock speed in Run mode.
- When a low power mode cannot be used, use the minimum possible frequency for the application. Select the frequency value that meets requirements.
- Use the dynamic control pull-up configuration if possible (see [Section 5.3: Dynamic control of pull-up resistor on page 17](#)).

## 6 ProxSense and low power modes

### 6.1 Possible CPU low power modes combined with ProxSense

Once a ProxSense acquisition is launched, the CPU and its clock are not needed to perform the acquisition. Wait modes, entered by a WFE or a WFI instruction, can be used as well as Active-halt mode by executing a HALT instruction. If WFE is used, either a PXS interrupt or the PXS event must be enabled but, if WFI or HALT is used then a PXS interrupt must be enabled. In most cases, the PXS end of completion interrupt is used.

In Active-halt, it is possible to combine the auto-wakeup with the ProxSense. The acquisition can be time limited by configuring the maximum counter enable register (PXS\_MAXENR) and maximum counter value register (PXS\_MAXR).

**Table 3. ProxSense compatibility with low power modes**

Low power mode	Instruction	Exit by PXS	CPU	HSI	HSI_PXS	SYNCHRO <sup>(1)</sup>
Wait	WFE	Yes	Off	Off	On	Yes
	WFI	Yes	Off	Off	On	Yes
Active-halt	HALT	Yes	Off	Off	On	No <sup>(2)</sup>
Halt	HALT	No	Off	Off	Off	No

1. 'SYNCHRO' indicates compatibility with the synchronization feature.
2. Once in Active-halt, the synchronization edge cannot be detected by the ProxSense peripheral but, Active-halt mode can be entered once the acquisition is started. To synchronize with an external signal, the acquisition must be launched from an external interrupt routine.

Activation level (AL bit set in CFG\_GCR register) can be used in the case of successive acquisitions. The AL bit should be reset once the last acquisition has been performed in order to return to the main processing and take the appropriate action according to the acquisition results.

### 6.2 Main factor of the ProxSense acquisition consumption

The best way to reduce consumption of the ProxSense acquisition itself is to decrease the acquisition time. This can be done by setting the ProxSense in order to get a lower count when the key is not touched. In the STM8TL5x STMTouch Library, this is done by reducing the KEY\_TARGET\_REFERENCE parameter in the touch sensing configuration file. This must be done carefully as reducing this count also reduces the sensitivity of the touchkey. A trade-off must be found between the consumption and the sensitivity.

## 6.3 Low power features in the ProxSense peripheral

Power consumption can be reduced in different ways between ProxSense acquisitions. The ProxSense can be switched off and its clock gated by resetting the PCKENR17 bit in the CLK\_PCKENR1. Consequently, the ProxSense has the following special low-power features.

### 6.3.1 LOW\_POWER bit

The most efficient way to reduce power consumption between two acquisitions is to set the LOW\_POWER bit in the PXS control register 1 (PXS\_CR1). In this case, the ProxSense clock (HSI\_PXS) and the ProxSense regulator are switched off as soon as an acquisition is completed and the results are still available in the PXS counter register of receiver channel n (PXS\_RXnCNTR).

By switching off the ProxSense regulator, a stabilization delay is introduced at the beginning of each new acquisition. This delay time depends on the number of enabled PXS\_RX pins. For a few receivers, a 1.7 ms delay must be selected.

For successive acquisitions, it is recommended to reset the LOW\_POWER bit and to set the ProxSense interrupt to a high priority. This saves the stabilization time on the one hand and greatly reduces the time between the acquisitions while the ProxSense clock and regulator are set. The LOW\_POWER bit must be set again once the last acquisition has been launched. In this way, the regulator and the clock are switched off as soon as the last acquisition is completed.

### 6.3.2 Stabilization time

The stabilization time is needed to ensure the regulator is fully operational at start-up. This delay can be set at different values in the PXS control register 3 (PXS\_CR3). The values are: 1.7 ms (default), 500  $\mu$ s or 120  $\mu$ s.

The more the regulator is loaded, the faster the stabilization. In other words, launching an acquisition with many receivers enabled speeds up the regulator stabilization. But, it is the impedance of the receivers that is more important than their number. The best way to check that the stabilization time is long enough, is to perform two successive acquisitions without switching off the regulator between them, in ensuring that the ProxSense is off before the first acquisition. If the results of the two acquisitions are similar, this means the regulator is correctly stabilized before the first acquisition.

In case of successive acquisitions, in order to select the lowest stabilization time, the acquisition with the maximum number of receivers must be launched first.

### 6.3.3 Bias parameter

This is the nominal bias current for the OpAmp which maintains the PXS\_RXn pins at the final (discharged) voltage while the transmit lines are transitioning (back) low. This function is valid only for projected capacitance measurements. A higher bias is needed when the projected capacitance (the capacitance between each Rx line and each Tx line) is high.

This parameter is set in PXS\_CR3. The lower its value, the less the consumption.

### 6.3.4 Inactive state

It is recommended to set the inactive state of the receivers and transmitters to  $V_{SS}$  in order to avoid driving noise on these lines. This also allows consumption to be reduced through the PXS\_RX and PXS\_TX pins compared to if they were configured in floating state.

### 6.3.5 Receiver disabling

Between acquisitions, if the ProxSense is not switched off by resetting the PXS\_EN bit in the PXS control register 1 (PXS\_CR1) or by setting the LOW\_POWER bit, there is a static consumption which depends on the number of enabled receivers. To reduce this consumption, it is worth disabling all the receivers by resetting the PXS receiver enable register (PXS\_RXENR).

This is particularly relevant for applications that need a fast response time and which cannot use the LOW\_POWER bit option due to the stabilization time.

## 7 Low power mode management by the STM8TL5x STMTouch Library

### 7.1 Configuration available in the `stm8_tsl_conf.h` file

This section presents the parameters of the library, corresponding to the features presented in [Section 6: ProxSense and low power modes](#).

#### 7.1.1 Acquisition time

The acquisition time depends on the:

1. Frequency of the acquisition
2. Up and pass length
3. Average value targeted when there is no touch

##### Frequency of the acquisition

The frequency of the acquisition is set in the `TSLPRM_PXS_HSI` (the high-speed internal clock that is dedicated to the analog ProxSense system). The possible frequencies are:

- 16 MHz (`TSLPRM_PXS_HSI 16000`)
- 8 MHz (`TSLPRM_PXS_HSI 8000`)
- 4 MHz (`TSLPRM_PXS_HSI 4000`)
- 2 MHz (`TSLPRM_PXS_HSI 2000`)
- 1 MHz (`TSLPRM_PXS_HSI 1000`)
- 500 kHz (`TSLPRM_PXS_HSI 500`)
- 250 kHz (`TSLPRM_PXS_HSI 250`)
- 125 kHz (`TSLPRM_PXS_HSI 125`)

##### Up and pass length

Up and Pass are the two active phases of a ProxSense acquisition cycle. Their length must be selected in order to ensure a correct charge and discharge of the projected capacitance between each receiver/transmitter pair.

If the Up and Pass lengths are set for a value of less than five by `TSLPRM_PXS_UP_LENGTH` and `TSLPRM_PXS_PASS_LENGTH` definitions, each phase lasts this value (`TSLPRM_PXS_UP_LENGTH/ TSLPRM_PXS_PASS_LENGTH`) + 0.5 cycles. For a value greater than or equal to five, each Up and Pass length is  $2^{(TSLPRM\_PXS\_UP\_LENGTH-2)+0.5}$  cycles and  $2^{(TSLPRM\_PXS\_PASS\_LENGTH-2)+0.5}$  cycles.

If the `TSLPRM_PXS_UP_LENGTH` and `TSLPRM_PXS_PASS_LENGTH` are equal, [Table 4](#) gives the correspondence between the Up and Pass length value and the number of cycles of a complete phase (Up + Pass + deadtimes).

**Table 4. ProxSense cycle length**

TSLPRM_PXS_UP_LENGTH and TSLPRM_PXS_PASS_LENGTH value	ProxSense cycle length (in HSI_PXS period)
1	4
2	6
3	8
4	10
5	18
6	34
7	66

**Average value targeted when there is no touch**

The average value targeted when there is no touch is set in the TSLPRM\_KEY\_TARGET\_REFERENCE definition. This value defines how long the acquisition lasts. If this value is set to 1000, the acquisition of a receiver set lasts approximately 1000 times the ProxSense cycle length. The lower this value, the lower the consumption but, the lower the key sensitivity.

**7.1.2 LOW\_POWER bit**

The LOW\_POWER bit is set if TSLPRM\_PXS\_LOW\_POWER\_MODE is defined as 1.

The firmware library detects if several acquisitions are launched successively. In this case, the LOW\_POWER bit is reset until the last acquisition is launched. This avoids having to wait for the stabilization time at each acquisition.

**7.1.3 Stabilization time**

The stabilization time is defined by the TSLPRM\_PXS\_STAB definition. It can be set to:

- LONG\_STAB (default stabilization time) for 1.7 ms
- MEDIUM\_STAB for 500  $\mu$ s
- SHORT\_STAB for 250  $\mu$ s

**7.1.4 Bias parameter**

The bias parameter is defined by the TSLPRM\_PXS\_BIAS definition. It can be set to:

- HIGH\_BIAS for 10  $\mu$ A (default bias)
- MEDIUM\_BIAS for 7.5  $\mu$ A
- LOW\_BIAS for 5  $\mu$ A
- VERY\_LOW\_BIAS for 2.5  $\mu$ A

### 7.1.5 Receiver configuration when disabled

The receivers are configured, when disabled, according to the TSLPRM\_PXS\_INACTIVE\_RX definition. The respective bit is reset in the receive enable register (PXS\_RXENR). The TSLPRM\_PXS\_INACTIVE\_RX definition can be defined as 0 to drive the receivers to ground or it can be set to 1 to keep them floating.

When the receiver group A is selected, all group B receivers are configured according to the TSLPRM\_PXS\_INACTIVE\_RX definition.

The receiver PXS\_RX pins are fully dedicated to ProxSense acquisition, i.e. they cannot be used for general purposes and are configured and driven according to the ProxSense configuration even if they are not used by any of the acquisition banks.

### 7.1.6 Transmitter configuration when disabled

The transmitters, which are defined in an acquisition bank and not enabled in the Transmit enable register (PXS\_TXENR), are configured according to the TSLPRM\_PXS\_INACTIVE\_TX definition. The TSLPRM\_PXS\_INACTIVE\_TX definition can be defined as 0 to drive the transmitters to ground or it can be set to 1 to keep them floating.

The GPIOs which support the transmitter (PXS\_TX) alternate function, but which do not belong to any acquisition bank or acquisition transmitter definition (example, BANK\_x\_TX) are not configured by the STM8TL5x STMTouch Library. It is the responsibility of the application code developer to configure the GPIOs which are not used for ProxSense acquisition.

## 7.2 Practical code example

[Section 7.2](#) presents two code examples. Both are based on the STM8TL5x STMTouch Library. The first example (see [Section 7.2.1: Low power management with all acquisition banks](#)) is a general case using all the acquisition banks as defined by default. The second example (see [Section 7.2.2: Very low power management with proximity detection](#)) configures the system in a very low power mode using Active-halt mode, with the AWU and ProxSense waiting for a proximity detection.

### 7.2.1 Low power management with all acquisition banks

This example shows how to perform all bank acquisitions with minimum code executed by the application and a maximum time spent in wait mode.



**Code example 1**

```
// Configure the Zone
TSL_acq_ZoneConfig(&MyZone, 0);
// Set the AL bit to exit from WFI mode only on PXS interrupt
CFG->GCR |= (uint8_t)CFG_GCR_AL;
// Start Bank acquisition
TSL_acq_BankStartAcq();
wfi();
// Process Objects
TSL_obj_GroupProcess(&MyObjGroup);

// Dxs processing
TSL_dxs_FirstObj(&MyObjGroup);
```

With MyZone being declared as followed:

```
TSL_Zone_T MyZone = {MyBankSorting, 0, TSLPRM_TOTAL_BANKS};
```

And MyBankSorting as followed:

```
TSL_tIndex_T MyBankSorting[TSLPRM_TOTAL_BANKS] = {0, 1, 2, 3};
```

MyBankSorting is a simple array of index referring to an acquisition bank number.

The second element of MyZone is the index to an element in MyBankSorting used as the first bank to be acquired.

The third element of MyZone is the number of banks belonging to this Zone.

In this simple example, all the banks are defined in MyZone and sorted from the first to the last.

User can imagine to declare many zones from a single bank sorting table:

```
TSL_Zone_T MyZone = {MyBankSorting, 0, TSLPRM_TOTAL_BANKS};
```

```
TSL_Zone_T MyZone1 = {MyBankSorting, 1, 3};
```

```
TSL_Zone_T MyZone2 = {MyBankSorting, 3, 2};
```

And MyBankSorting as followed:

```
TSL_tIndex_T MyBankSorting[TSLPRM_TOTAL_BANKS] = { 0, 1, 2, 3, 6, 5, 4};
```

With such declaration, the acquisition on MyZone will acquire all acquisition banks, while MyZone1 acquires only bank 1, 2, 3 and MyZone2 acquires only bank 3 and 6.

**7.2.2 Very low power management with proximity detection**

This example shows how to manage a proximity detection by scanning the key each 512 ms. This is done by alternating Active-halt mode with the AWU and Active-halt mode with ProxSense. Once a proximity is detected, a scan of the touch keys is performed. This last

step is almost identical to the previous example in [Section 7.2.1: Low power management with all acquisition banks](#).

Entry into Proximity detection mode is conditioned by the value of the “proxy\_mode” variable which is managed in a user function. This mode is exited by checking the state of the POWER touchkey.

The response time depends on the scanning frequency but, also on the debounce value which is defined by the CounterDebProx parameter in the touchkey TSL\_TouchKeyParam\_T structure. If needed, this variable can be modified by the user application when entering Proximity detection mode and it can be restored when exiting it (this variable is not modified in the example given below).

### Code example 2

```
// Executive low power loop, runs continually till POWER key detects a proximity
AWU_Init(AWU_Timebase_512ms);
ECS_last_tick = 40;

while (proxy_mode)
{
    TSL_user_Action_LowPower();

    if (MyTKeys[POWER].p_Data->StateId == TSL_STATEID_DEB_PROX)
    {
        AWU_Init(AWU_Timebase_16ms); //reduce the period between acquisition in order
to have a faster response time.
    }
    else
    {
        if (TEST_PROX(POWER))
        {
            proxy_mode = FALSE;
            POWER_LED = 1;
            LED_ManageLighting();
        }
        else
        {
            AWU_Init(AWU_Timebase_512ms);
        }
    }

    disablePXS();

    // Prepare AWU
```

```
    AWU_Cmd(ENABLE);  
    halt();  
    enablePXS();  
    AWU_Cmd(DISABLE);  
}
```

With :

```
/**  
 * @brief Execute STMTouch Driver main State machine when in Low Power mode.  
 * @param None  
 * @retval None  
 */  
void TSL_user_Action_LowPower(void)  
{  
    // Configure Bank 2 which the POWER touchkey belongs to  
    TSL_acq_BankConfig(2);  
  
    CFG->GCR |= CFG_GCR_AL;  
  
    // Start Bank acquisition  
    TSL_acq_BankStartAcq();  
  
    halt();  
  
    // Process Objects of the Low Power group  
    TSL_obj_GroupProcess(&MyObjGroupLowPower);  
  
    // ECS  
    if (--ECS_last_tick == 0)  
    {  
        TSL_ecs_Process(&MyObjGroupLowPower);  
        ECS_last_tick = 40;  
    }  
}
```

## 8 Conclusion

The STM8TL5xxx devices offer many possibilities for developing low consumption applications. The user can take advantage of various low power modes like Wait, Active-halt or Halt modes. He can also reduce consumption by switching off peripherals when they are not used. Another option for reducing consumption is to optimize the ratio between Run and Halt modes due to the good performance of the CPU itself.

The ProxSense can be used in Wait or Active-halt modes where its consumption depends mainly on the acquisition duration and on the number of enabled receivers.

The most important principles of low power mode management are described in this application note, including hints on how and when to use it.

## 9 Revision history

**Table 5. Document revision history**

Date	Revision	Changes
08-Jul-2011	1	Initial release.
04-Nov-2011	2	STM8TL53xx product name update Changed references to associated documents Updated <a href="#">Figure 1: Power supply overview on page 6</a>
10-Sep-2012	3	STM8TL5xxx product name update Replaced all footnotes under <a href="#">Figure 1: Power supply overview on page 6</a> Updated <a href="#">Section 5.2: GPIO initialization on page 16</a> Updated <a href="#">Section 7: Low power mode management by the STM8TL5x STMTouch Library on page 22</a>

**Please Read Carefully:**

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

**UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.**

**UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.**

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2012 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

[www.st.com](http://www.st.com)