AN3404
Application note

Low power considerations for STM8TL53xx devices

Introduction

This document is intended for touch sensing application designers who require an overview of low power modes in the STM8TL53xx 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 Touch Sensing Library in addition to giving some code examples.

This application note is not intended to replace the STM8TL53xx datasheet. All values given in this document are for guidance only. For guaranteed values, please refer to the STM8TL53xx datasheet.

Contents

1 Power consumption factors

The STM8TL53xx 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

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

where:

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

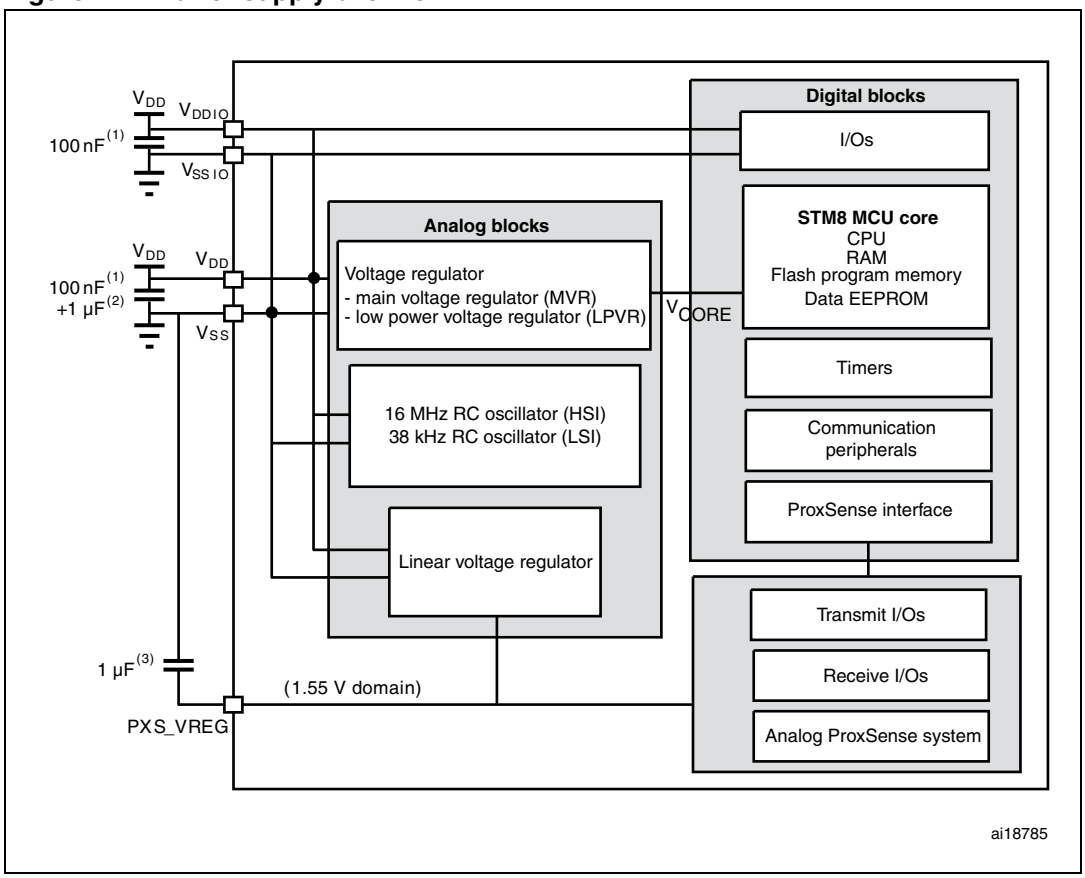
Consequently, power consumption depends on:

- **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 STM8TL53xx 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 Power supply

The STM8TL53xx family embeds two regulators which provide a supply voltage (V_{CORE}) for the core and internal peripherals.

Figure 1. Power supply overview



1. If V_{DD} is lower than 1.8 V, the 1.8 V domain is supplied by the voltage on the V_{DD} input. For low power modes where the low power voltage regulator (LPVR) is used, this domain is supplied by a 1.55 V voltage.

The main voltage regulator (MVR) provides a 1.8 V supply voltage. It has a high current capability, as it can deliver up to 25 mA. However, the consumption of this regulator is higher than the consumption of the LPVR. Consequently, the MVR is used during a standard operation only.

The consumption of the LPVR is very low as required for low power modes. The LPVR can deliver up to 200 µA, providing 1.55 V to the digital part of the MCU.

After reset, the MVR provides a supply voltage (V_{CORE}) to the internal digital parts of the microcontroller. Depending on the functional mode, the MVR can be switched off. In this case, the LPVR continues to provide the V_{CORE} voltage. The power supply is monitored by the power-on reset/power-down reset (POR/PDR). This system ensures a proper startup and reset of the MCU, while V_{DD} rises above the POR threshold. It resets the MCU when V_{DD} falls below the PDR threshold.

2.1 Internal supply structure

STM8TL53xx 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 STM8TL53xx 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 STM8TL53xx 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 STM8TL53xx devices is given by *Equation 3*.

Equation 3

$$I_{DD(STM8TL53xx)} = f \times 150[(\mu A)/MHz] + 215[\mu A]$$

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 STM8TL53xx devices, the Flash is automatically switched off when Halt or Active-halt mode is entered.

4.2 Overview of low power modes

The STM8TL53xx 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 STM8TL53xx datasheet.

Table 1 lists the STM8TL53xx low power modes, shows the basic behavior of STM8TL53xx devices, and demonstrates the influence of different low power modes on the CPU, peripherals and consumption.

Table 1. Low power mode summary

Low power modes	Entry instruction	Functions						Low power mode consumptions
		CPU	Peripherals	HSI	LSI	Flash	RAM	STM8TL53xx typical values @ 3 V/25°C
Wait	WFE or WFI	Off	Can be enabled	On	On	On	On	500 µA
Active-halt with ProxSense	Halt	Off			On	Off	On	600 µA(1)
Active-halt with AWU					On	Off	On	1.0 µA
Halt					Off	Off	On	0.4 µA

1. Value during an acquisition (see the STM8TL53xx 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.

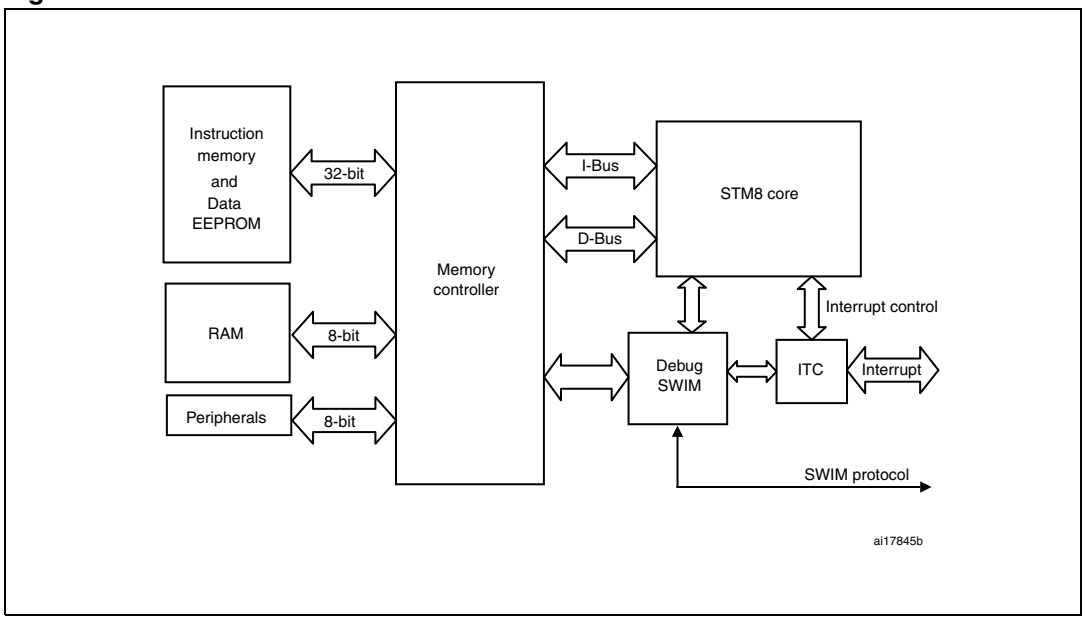
Note: As all peripheral clocks are gated by default after reset on STM8TL53xx devices, it is mandatory to enable the clock for a given peripheral before accessing any peripheral register.

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. STM8TL53xx architecture



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

4.6 Wait mode

The STM8TL53xx devices support two different Wait modes: Wait for interrupt (WFI) and Wait for event (WFE). Both modes are designed to reduce STM8TL53xx device power consumption by switching off the core when it is not used. Wait mode is mainly used when the STM8TL53xx 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 STM8TL53xx 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 STM8TL53xx 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

STM8TL53xx 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 nonaccurate 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 µA.

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

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 µA, 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 regulary, 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 µA. 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 STM8TL53.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 2. ProxSense compatibility with low power modes

Low power mode	Instruction	Exit by PXS	CPU	HSI	HSI_PXS	SYNCHRO(1)
Wait	WFE	Yes	Off	Off	On	Yes
	WFI	Yes	Off	Off	On	Yes
Active-halt	HALT	Yes	Off	Off	On	No(2)
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 appropraite 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 Touch Sensing 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_RX*n*CNTR).

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 µs or 120 µs.

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_RX*n* 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 reseting 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 Touch Sensing Firmware 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 targetted when there is no touch

Frequency of the acquisition

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

- 16 MHz (HSI_PXS 16000)
- 8 MHz (HSI_PXS 8000)
- 4 MHz (HSI_PXS 4000)
- 2 MHz (HSI_PXS 2000)
- 1 MHz (HSI_PXS 1000)
- 500 kHz (HSI_PXS 500)
- 250 kHz (HSI_PXS 250)
- 125kHz (HSI_PXS 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 UP_LENGTH and PASS_LENGTH definitions, each phase lasts this value (UP_LENGTH/ PASS_LENGTH) + 0.5 cycles. For a value greater than or equal to five, each Up and Pass length is 2^(UP_LENGTH-2)+0.5 cycles and 2^(PASS_LENGTH-2)+0.5 cycles.

If the UP_LENGTH and PASS_LENGTH are equal, *Table 3* gives the correspondance between the Up and Pass length value and the number of cycles of a complete phase (Up + Pass + deadtimes).

Table 3. ProxSense cycle length

UP_LENGTH and 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 targetted when there is no touch

The average value targetted when there is no touch is set in the 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 LOW_POWER_MODE is defined as 1.

The firmware library detects if several acquisitions are launched sucessively. 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 STAB definition. It can be set to:

- LONG_STAB (default stabilization time) for 1.7 ms
- MEDIUM_STAB for 500 µs
- SHORT_STAB for 250 µs

7.1.4 Bias parameter

The bias parameter is defined by the BIAS definition. It can be set to:

- HIGH_BIAS for 10 µA (default bias)
- MEDIUM_BIAS for 7.5 µA
- LOW_BIAS for 5µA
- VERY_LOW_BIAS for 2.5 µA

7.1.5 Receiver configuration when disabled

The receivers are configured, when disabled, according to the INACTIVE_RX definition. The respective bit is reset in the receive enable register (PXS_RXENR). The 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 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 INACTIVE_TX definition. The 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, BANKxx_TX) are not configured by the STM8TL5x Touch Sensing 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 Touch Sensing 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 focuses on the TSL_Action management and respects the following two definitions in the stm8_tsl_conf.h file:

- #define AUTOMATIC_CHAINING (1)
- #define FAST_RESPONSE_TIME (1)

This example shows how to perform all bank acquisitions with a minimum number of calls to TSL_Action and where the rest of the time is spent in low power mode. If a communication does occur, the application should call TSL_Action() in order to start the pending acquisition after which the acquisitions chain themselves. In this example, it is assumed that the communication is managed by interrupt, the STM8TL53xx being slave in this communication protocol.

Code example 1

```
Clock_init();//Initialize the CPU clock (user code)
GPIO_Init(); //Initialize the General purpose I/Os(user code)
TSL_Init();  //Initialize the Touch Sensing library

ExtraCode_Init(); // other user initialization code

for (;;) //endless loop
{
   /*
      At this stage TSLState is always TSL_IDLE_STATE.
   */
   TSL_Action();//switch from TSL_IDLE_STATE to TSL_ALLKEYS_PXS_ACQ_STATE
                // with AcqState equals NO_ACQUISITION

   /*
      The following loop is used in case a communication occurs during
      the acquisitions (communication being processed by interrupt)
      If communication occurs, the Activation level bit must be reset
      once finished in order to resume the processing
   */

   do
   {
      TSL_Action();//Launch the acquisitions AcqState equals ACQUISITION_ON_GOING
      CFG->GCR |= CFG_GCR_AL; // set Activation level to stay in low power
                              // till the last acquisition
      wfi();  //halt() can also be used
   }
   while (AcqState != ALL_ACQUISITION_COMPLETED);  // this test allows to relaunch
the acquisition in case a communication has stopped their chaining

   /*
      When the main resumes all the acquisition has been performed
      at this stage, TLSState equals TSL_SCKEY_PXS_PROC_STATE (if SCKey defined)
      or TSL_MCKEY_PXS_PROC_STATE(if only MCKeys are defined)
   */

   TSL_Action();//perform the key processing (switch to TSL_ECS_STATE)
   TSL_Action();// perform the environment change system
                // (switch to TSL_IDLE_STATE)

   ExtraCode_StateMachine(); //application processing
}
```

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 (ExtraCode_StateMachine() in this example). This mode is exited by checking the STM8TL5x Touch Sensing Library global variable “TSL_GlobalState.b.DETECTED”.

The response time depends on the scanning frequency but, also on the debounce value which is defined by the “DetectionIntegrator” global variable. 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

```
   Clock_init();
   GPIO_Init();
   TSL_Init();
   ExtraCode_Init();

   AWU_Init(AWU_Timebase_512ms);
   tick_delay = 512 * 2;
   AcquisitionBankSorting[4] = 0; //Only the first four acquisition banks are
acquired, the fifth one being for proximity

   for (;;) //endless loop
   {
      if (proxy_mode)
      {
         AcquisitionBankSorting[0] = 5; //Configure bank #5 to be acquired first
         AcquisitionBankSorting[1] = 0; //Only the first acquisition is performed

         while (proxy_mode)
         {
            TSL_Action();// switch from TSL_IDLE_STATE to TSL_ALLKEYS_PXS_ACQ_STATE
                         // with AcqState equals NO_ACQUISITION

            //only PXS and TIM4 are clocked
            // This assumes there will be no communication while in low power
            RestorePCKENR = CLK->PCKENR1;
            CLK->PCKENR1 = CLK_PCKENR1_PXS + CLK_PCKENR1_TIM4;

            //Launch the acquisition AcqState equals ACQUISITION_ON_GOING
            TSL_Action();

            halt(); //entering in Active-halt with wake-up by ProxSense

            // PXS acquisition estimated between 2 and 2.5ms with 16MHz
            // and UP/PASS_LENGTH = 3 for 1 bank
            TSL_Timer_Adjust(5);

            CLK->PCKENR1 = RestorePCKENR;

            // When the main resumes all the acquisition has been performed.
            // Here, TLSState equals TSL_SCKEY_PXS_PROC_STATE,if SCKey defined,
            // or TSL_MCKEY_PXS_PROC_STATE, only if MCKeys are defined

            TSL_Action(); //perform the key processing (switch to TSL_ECS_STATE)

            TSL_Action(); //perform the environment change system

            // The following loop is to insure no TSL_state is missed
            // in order not to be stuck in halt()
            while (TSLState != TSL_IDLE_STATE)
            {
               TSL_Action();
            }

            disablePXS(); // ProxSense is off to decrease consumption
            CLK_PCKENR1 &= ~CLK_PCKENR1_PXS;

            if (TSL_GlobalState.b.DETECTED != 0)
            {
               proxy_mode = 0;  //to exit proxy mode as a detection occured
            }
            else
            {
               // Disable the unused peripheral clock,
               // only AWU and TIM4 clock are kept on
               RestorePCKENR = CLK->PCKENR1;
               CLK->PCKENR1 = CLK_PCKENR1_AWU + CLK_PCKENR1_TIM4;
```

```
            // Prepare AWU
            AWU_Cmd(ENABLE);

            halt(); // enter in Active-halt mode with Auto wake-up

            TSL_Timer_Adjust(tick_delay);
            /*Restore the peripheral clocks */
            CLK->PCKENR1 = RestorePCKENR;

            //restart ProxSense
            CLK_PCKENR1 |= CLK_PCKENR1_PXS;

            enablePXS();

            AWU_Cmd(DISABLE);
         }

      }

      //restore the acquisition bank table to perform all the bank processing
      AcquisitionBankSorting[0] = 1;
      AcquisitionBankSorting[1] = 2;
   }

   while (!proxy_mode)//This loop is similar to the first example
                      //some parts can be factorized in order to save code size
   {
   /*
      At this stage TSLState is always TSL_IDLE_STATE.
   */
   TSL_Action();//switch from TSL_IDLE_STATE to TSL_ALLKEYS_PXS_ACQ_STATE
                // with AcqState equals NO_ACQUISITION

   /*
      The following loop is used in case a communication occurs during
      the acquisitions (communication being processed by interrupt)
      If communication occurs, the Activation level bit must be reset
      once finished in order to resume the processing
   */

   do
   {
      TSL_Action();//Launch the acquisitions AcqState equals ACQUISITION_ON_GOING
      CFG->GCR |= CFG_GCR_AL; // set Activation level to stay in low power
                              // till the last acquisition
      wfi();  //halt() can also be used
   }
   while (AcqState != ALL_ACQUISITION_COMPLETED);  // this test allows to relaunch
   the acquisition in case a communication has stopped their chaining

   /*
      When the main resumes all the acquisition has been performed
      at this stage, TLSState equals TSL_SCKEY_PXS_PROC_STATE (if SCKey defined)
      or TSL_MCKEY_PXS_PROC_STATE(if only MCKeys are defined)
   */

   TSL_Action();//perform the key processing (switch to TSL_ECS_STATE)
   TSL_Action();// perform the environment change system
                // (switch to TSL_IDLE_STATE)

   ExtraCode_StateMachine();//application processing
                            //define if the proxy mode must be entered
                            //according to user conditions

   }
}
```

8 Conclusion

The STM8TL53xx 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 4. 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 *Figure 1: Power supply overview on page 6*