TA0342
Technical article

Accurate power consumption estimation for STM32L1 series of ultra-low-power microcontrollers

1 Overview

STMicrolectronics' STM32L1 series of ultra-low-power microcontrollers include a number of system features intended to reduce the energy consumption, such as a programmable LDO and multiple internal clock sources. They offer multiple configurations to support applications that have conflicting requirements, for example either running continuously with a tight current budget or requesting complex computational tasks to be executed as fast as possible on waking from the deepest sleep modes.

This article explains how to get the best balanced consumption ratio (energy per software task) for the most efficient processing. Factors that influence energy consumption such as voltage scaling techniques, ADC conversions and wake-up events will be examined and used to build an accurate estimate of the total power consumption.

Contents

2 Introduction

STM32L1 ultra-low-power microcontrollers are designed to maximize battery life cycle by reducing the overall system power consumption. However, applications in markets such as consumer, general portable devices, and portable medical or metering devices have conflicting requirements. On one hand, these applications require very low-power consumption to extend battery life, and on the other hand they require high performance for computational tasks, which leads to higher power consumption.

A typical low-power application spends most of its time in sleep/deep sleep mode waiting for external events or interrupts to wake-up and process software tasks.

The conflicting requirements for low-power consumption and the occasional need for high performance present a sharp contrast that has to be carefully studied by developers who are choosing the most suitable MCU for their application requirements by analyzing the various components of the overall power consumption. Power consumption calculation and analysis is mainly dependant on how the developer uses the selected MCU's low-power features while taking care of all the other factors that can influence the overall power consumption.

With careful design, the overall power consumption can be significantly improved. Some of the more obvious steps that developers can take to reduce the power are slowing down the system frequency, gating clocks of unused peripherals, keeping the system longer in deep sleep modes, and waking-up the MCU only for short periods as necessary for computational tasks.

In this case, the more often the system wakes-up, the less the system stays in deep sleep modes. With many low-power MCUs when waking-up from the deepest sleep modes, the system resumes as if reset and requires time to re-configure the system clocks. Therefore, while keeping the low-power MCU in deep sleep mode for as long as possible is important, it is not enough to fully optimize the current consumption. Developers also need to be aware of the other factors which affect power consumption and take advantage of all hardware features available in the device.

Thus, when we are discussing energy efficient microcontroller applications, the power consumption of the MCU in the deepest sleep modes is not enough to estimate the current consumption budget. Other factors to consider when estimating power consumption in low-power applications include the MCU performance and dedicated low-power MCU hardware features for reducing the overall current consumption.

In this document, different factors which can affect the current consumption of a low-power MCU design are described, and then give some practical examples of how to build an accurate overall current consumption estimate while taking advantage of all the low-power MCU hardware configured features.

To give a sufficiently reliable overall picture of the power consumption, main factors influencing the consumption during low-power modes were focused as well as Run modes. The following topics are discussed:

- Dynamic voltage scaling techniques
- MCU hardware architecture and practical examples of power estimation
- Low-power analog peripherals

3 Dynamic voltage scaling techniques

Power consumption can be expressed as the operating voltage (VDD) multiplied by current consumption (I_{DD}):

$$P = V_{DD} \times I_{DD}$$

The operating supply voltage has a direct impact on the overall power consumed by the MCU. As can be seen from the equation above, decreasing the operating supply voltage decreases the power consumption. The MCU current consumption is affected by the operating system clock frequency.

Therefore, the two major factors affecting the MCU dynamic power consumption are:

- The operating supply voltage
- The system clock frequency

The faster the MCU is running, the higher the power consumption is. Decreasing the operating frequency and the operating supply voltage together makes a significant reduction in the dynamic current consumption. The challenge is to identify the best combination for efficient power consumption while meeting the performance needs of the application.

Dynamic voltage scaling is a very effective technique to achieve the best ratio between power consumption and performance. It allows modification of the MCU operating supply voltage range and system frequency during runtime through periods when the application does not need significant processing.

Dynamic voltage scaling to decrease the core voltage (Vcore) is known as undervolting. It is performed to save power, particularly in laptops and other mobile devices, where the energy comes from a battery and is thus limited.

Although at first glance, it is not obvious that high performance correlates with low MCU current consumption, it is a great advantage for many low-power applications to wake-up very quickly, execute software tasks at a high speed and then go back to sleep again as quickly as possible.

The MCU performance depends on the type of processor core as well as the operating frequency. When discussing the ultra-low-power and performance efficiency of low-power MCUs, quite often values are provided in µA/MHz, which does not tell the full story. A better measure of the MCU performance would be the number of executed instructions per second instead of the system clock frequency.

Let's examine the example of the ARM Cortex-M3 core which delivers outstanding computational performance and exceptional system response to events while meeting the challenges of low dynamic and static power constraints. If we compare the Cortex-M3 core to a typical 16-bit core found in many low-power MCUs, the Cortex-M3 with higher processing performance spends less time in Run mode thus maximizing the time in deep sleep mode. If we look at mA/DMIPS (DMIPS standing for Dhrystone MIPS measured using the public benchmark Rev 2.0), the performance of the Cortex M3 is significantly better than the typical 16-bit MCU core and others.

4 MCU hardware architecture

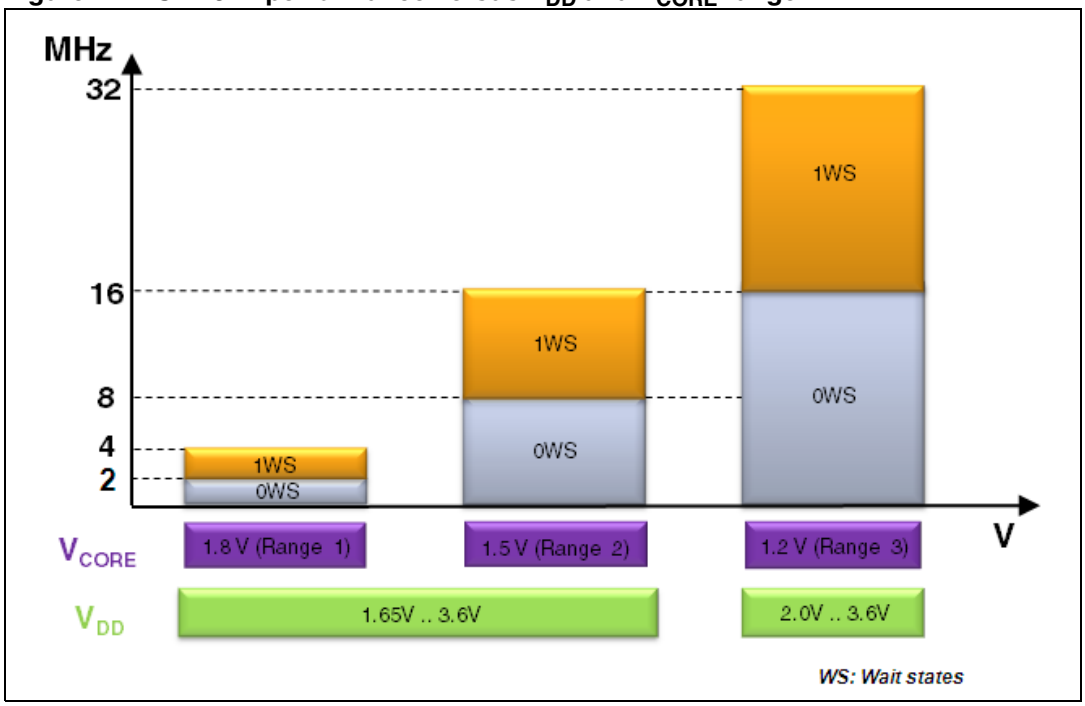
4.1 Programmable LDO (low drop-output)

However, to dynamically modify the voltage and the operating frequency, the device architecture has to allow this practice. Most low-power MCUs have an internal regulator providing the voltage to the core, digital peripherals as well as the memories. The output of the internal regulator voltage usually has a fixed output voltage allowing the entire system to operate at full speed. However, as this output voltage is set for best performance, it doesn't allow the lowest current consumption if the system is running at low speed and from a low-power supply voltage. Some low-power MCUs therefore add an external low-current voltage regulator, and bypass the internal one, in order to provide a lower voltage to the digital peripherals and the core. This solution can be effective in lowering the current consumption but is not cost effective.

A better solution would be a programmable on-chip low drop-out (LDO) linear voltage regulator. This would give the best efficiency for low-power and system cost.

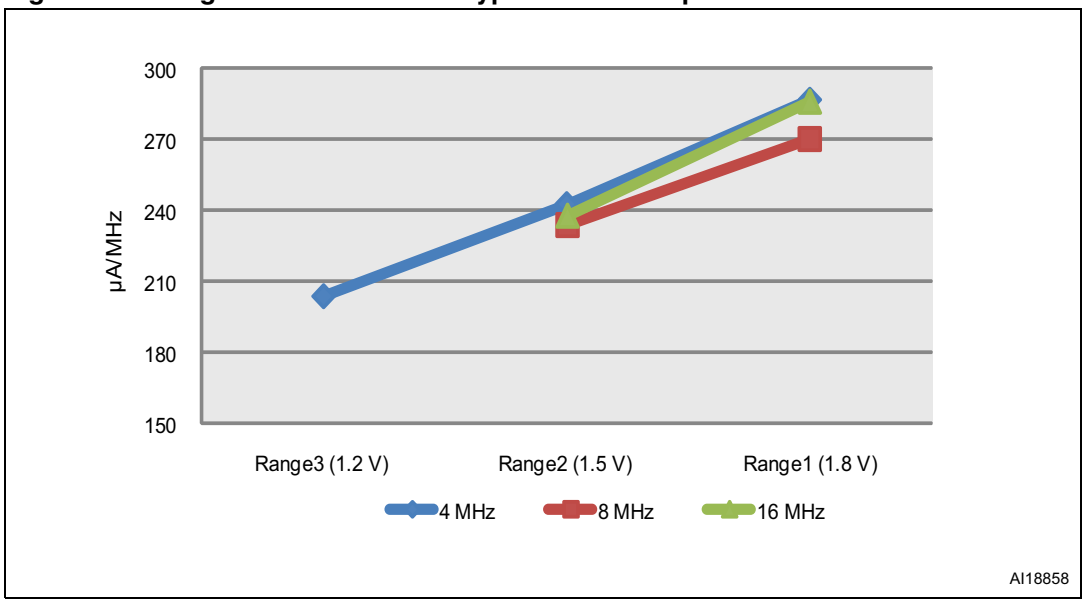
Let's examine an example of an example of a Cortex-M3 based ultra-low-power MCU with a programmable on-chip low drop-out (LDO) linear voltage regulator – the STM32L. It offers three dynamically selectable voltage ranges, from 1.8 V (range 1) down to 1.2 V (range 3).

Figure 1. STM32L performance versus V_{DD} and V_{CORE} range



To illustrate the advantage of a programmable LDO regulator, let's look at the STM32L device running at one operating supply voltage (V_{DD}) and at a given frequency. The power saving gain is up to 30 % when the LDO regulator is down to 1.2 V versus 1.8 V.

Figure 2. Programmable LDO and typical values of µA/MHz



Operating from a single supply voltage anywhere in the range from 1.65 V to 3.6 V, an important power saving can be yielded thanks to the LDO.

4.2 Performance analysis

Let's examine a comparison of STM32L and another low-power MCU "A" and start with datasheet values at the maximum speed of 32 MHz:

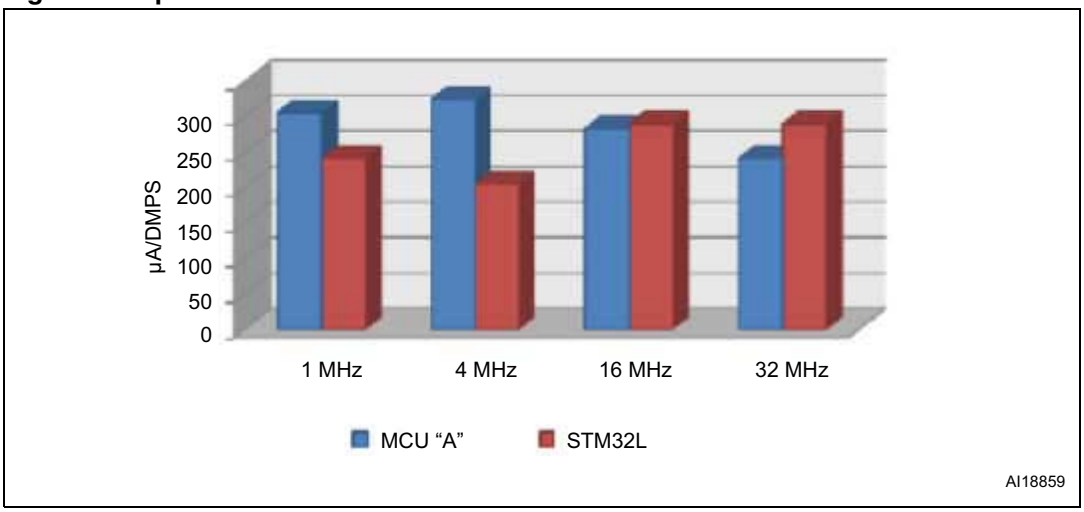
- Low-power MCU "A": 180 µA/MHz
- STM32L: 296 µA/MHz

So if we compare products in terms of µA/MHz at maximum speed, the low-power MCU A" would seem to be better than STM32L.

However, the picture changes when looking at performance analysis based on µA/DMIPS at all operating frequencies. The performance is analyzed with both MCUs running from Flash. For our experiments, the Dhrystone benchmark is used for power consumption comparison versus datasheet values. This is representative of most real-world low-power applications rather than execute irrelevant code with only a limited number of instructions available.

- Dhrystone results, in Run mode at Frequency of 32 MHz at 3.0 Volts (Fresh Coin cell battery)
  - Low-power MCU "A": 0.97 DMIPS/MHz at 1 wait state,
  - STM32L: 1.05 DMIPS/MHz at 1 wait state.
- Comparing the STM32L with low-power MCU "A" in terms of energy efficiency (µA per DMIPS), at high frequencies both MCUs are similar in efficiency while at low and medium frequencies the STM32L is more efficient. For example the STM32L is up to 25% more efficient at 4 MHz. Comparing both products only at a maximum or a given frequency is not enough for low-power applications.

Figure 3. µA/DMIPS at 3.0 Volts



Taking the advantage of the dynamic voltage scaling and the embedded programmable LDO of the STM32L, the overall current consumption is significantly improved for all operating frequencies. The Vcore voltage can be dynamically adjusted to range 3, range 2 or range 1 along with the operating frequency thus optimizing the µA/MHz and µA/DMIPS ratios.

In addition to dynamic voltage scaling and the internal low regulator, developers can take benefit from the low-power MCU system architecture, and if available, the DMA [Direct Memory Access] controller.

The use of DMA not only decreases the CPU load but also dramatically reduces the current consumption. The MCU can stay in low-power mode while the DMA is transferring data and wake-up only when the application is ready to process data, lowering overall current consumption and extending battery life.

These are the major factors to consider when estimating dynamic power consumption when the MCU is in Run mode; however, to estimate the overall power consumption we also need to factor in the power consumption while using low-power.

5 Power consumption estimation

Low-power applications often wake-up periodically from low-power modes to process data. Estimating the overall system power consumption requires looking at the low-power modes in conjunction with the MCU wake-up time and the dynamic power consumption discussed above. Faster wake times combined with the most energy efficient dynamic power consumption helps significantly in applications where the CPU is required to wake-up frequently.

Most low-power MCUs embed a low-power internal oscillator with a fast start up time. The internal oscillator is usually selected to wake-up from the deepest low-power modes and has the advantage of providing a low-cost (no external components) low-power clock source to reduce power consumption and wake-up time. The common practice is to wake-up (or start) on the slower low-power internal oscillator until a higher speed clock is ready.

Wake-up on a high-speed external clock results in higher power consumption and longer startup times whereas the internal oscillator has low-power consumption and fast start-up time and the wake-up depends on internal oscillator clock speed.

How can the power consumption budget be estimated when various internal clock speeds are offered for wake-up from low-power modes? Which one to choose to reach a high performance and low-power consumption?

This question is widely asked, and it is hard to provide common answer. However, an efficient low-power MCU should offer the ability to adapt according to the low-power application requirements.

One approach would be a configurable internal clock which can be adjusted depending on application requirements. The device could be configured to start either at low frequency or at high frequency, and then the system clock could also be slowed down or increased depending on operating tasks.

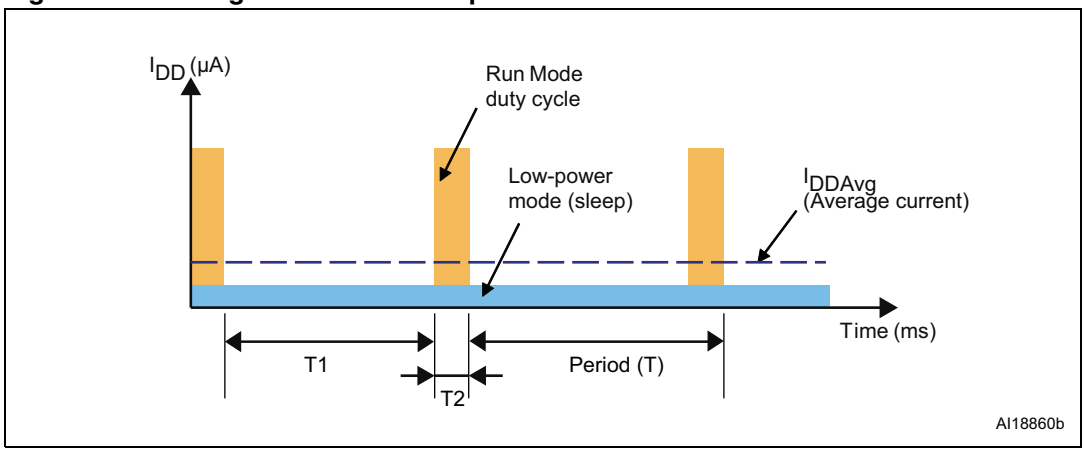
The power consumption mainly depends on two factors: power supply voltage and current consumption. The dynamic voltage scaling affects the first parameter, see *Section 3: Dynamic voltage scaling techniques*; next section is focused on the current consumption and factors that influence power consumption.

The average current consumption is affected by two major factors: The time spent during Run mode / Sleep mode and the operating frequency.

The MCU power consumption can be defined as the sum of the following:

$$TotalPowerConsumption = RunModePower + Sleep \times deepSleepModePower$$

Figure 4. Average current consumption



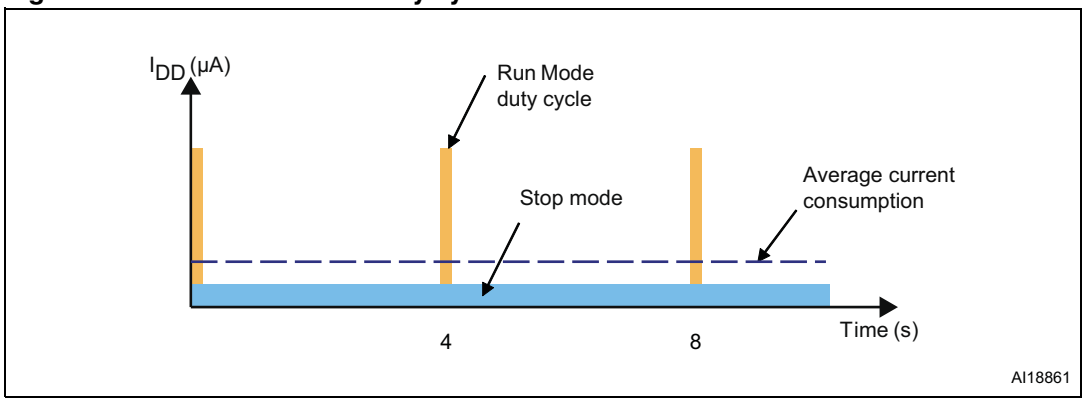
Equation 1:

$$I_{DD} = \frac{1}{T} \times \int_0^T i(t)dt = \frac{1}{T} \times \int_0^{T1} I_{DD(Sleep)}dt + \frac{1}{T} \times \int_{T1}^{T} I_{DD(Run)}dt = \frac{T1}{T} \times I_{DD(Sleep)} + \frac{T2}{T} \times I_{DD(Run)}$$

Let's take an example of an application which wakes up periodically from deep sleep mode. Two cases are examined:

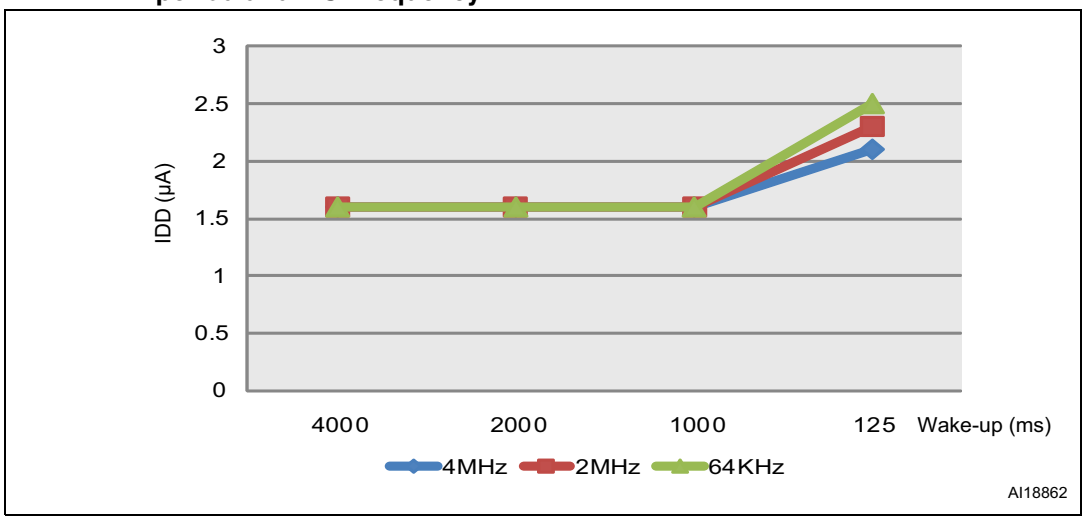
The examples are based on STM32L. The wake-up clock from Stop mode is derived from a programmable Multi-Speed Internal RC oscillator (MSI).

Figure 5. Short Run mode duty cycle



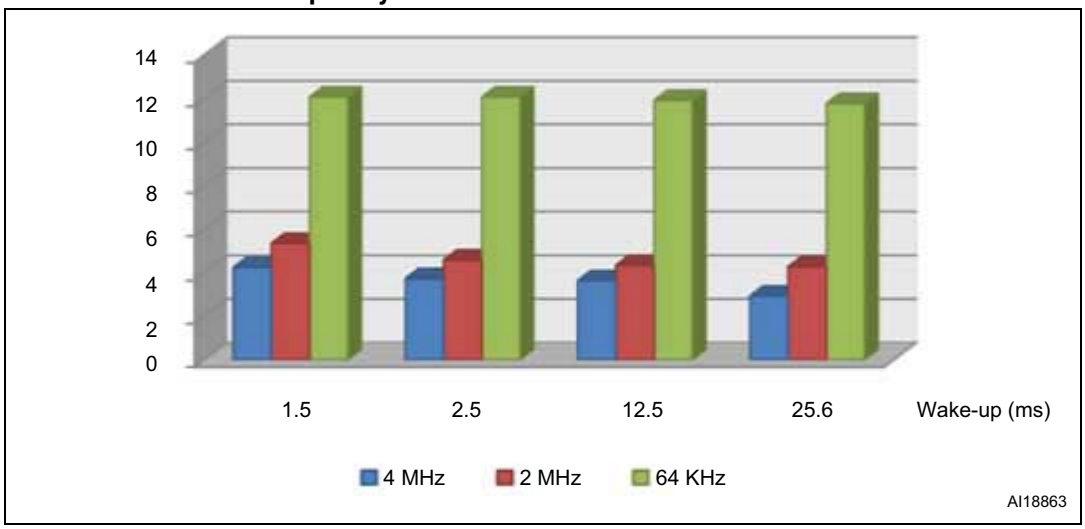
– Case 1: A short Run mode duty cycle versus the time spent during Stop mode. In this case the overall current consumption is closed to the Stop mode current consumption.

Figure 6. Short Run mode duty cycle: Typical current consumption versus wake-up period and MSI frequency



– Case 2: Increasing the Run mode duty cycle (60 % of the total period) while keeping the other conditions to be the same as the above experiments. In this case it is important to look at the overall energy instead of current consumption as the time spent in Run mode is increased / decreased depending on the operating frequency.

Figure 7. Case 2 increased Run mode duty cycle: Energy versus wake-up period and MSI frequency



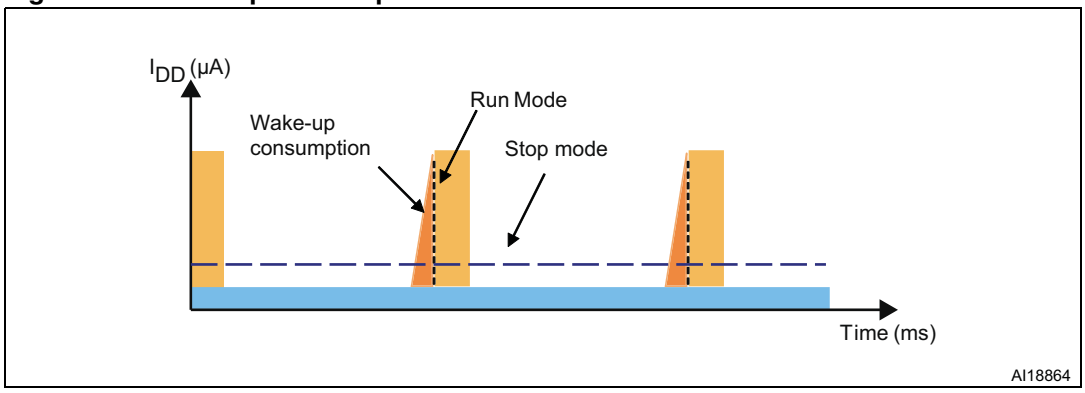
The above experiment shows the significance of a faster CPU clock for heavy software tasks with a higher Run mode duty cycle. At 4 MHz the overall energy spent is less than the 64 KHz or 2 MHz for all wake-up periods. The faster the CPU clock the faster the software processing, allowing the MCU to return to low-power mode more quickly. For lighter software tasks (short Run mode duty cycle), the faster clock 4 MHz results in a lower current consumption for applications with frequent wake-up events. Hence, the operating frequency or wake-up frequency can be adapted depending on the application requirements.

5.1 Wake-up energy

The wake-up time and clock selection for system wake-up are important parameters that contribute to the overall MCU current consumption. This is wasted energy, since during the wake-up time the CPU is just waiting for internal regulator restarts and clock start until it is ready to fetch the first instruction.

The wake-up energy is more critical for applications that wake-up more frequently.

Figure 8. Wake-up consumption illustration



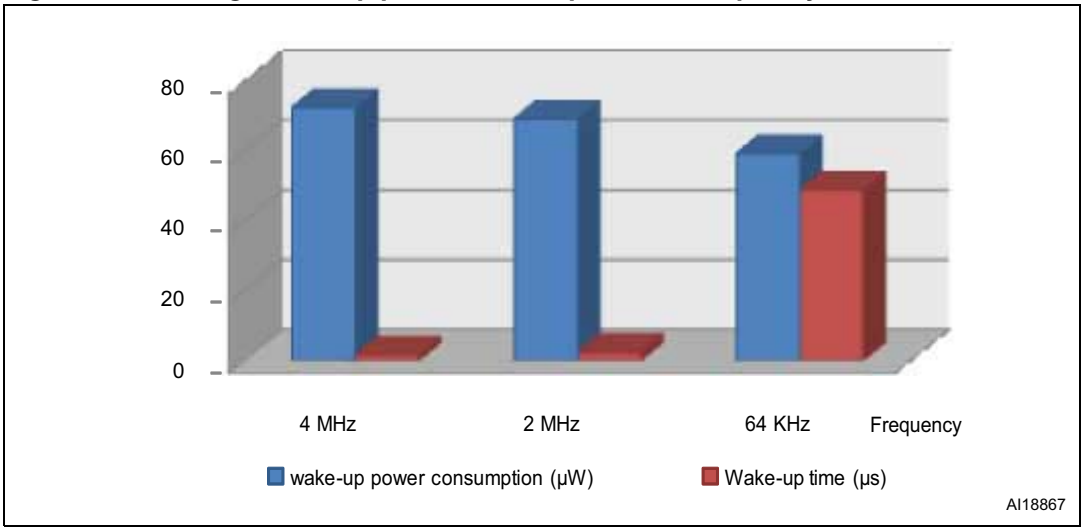
For many applications that wake-up periodically, the MCU consumes more energy when exiting low-power mode than when processing the software task. Therefore, it is important to choose a suitable low-power mode to wake-up as fast as possible in order to reduce the wasted energy during the wake-up phase.

The standby mode provides the lowest power consumption but has a longer wake-up time as the system restarts as if out of reset when exiting from standby. This therefore may not be the best choice for applications that wake-up frequently.

As shown in the earlier sections the 4 MHz wake-up clock source when exiting Stop mode gives the best overall power consumption for applications with a higher Run mode duty cycle. To analyze the contribution of the wake-up energy and its dependence on clock source, the wake-up time and wake-up power consumption with 4 MHz, 2 MHz and 64 kHz clock sources are compared.

This chart shows the wake-up power consumption from Stop mode while system wakes-up periodically every 2.5 ms for only a few software tasks in Run mode. Conditions of the experiments are the same as in the previous chapter.

Figure 9. Average wake-up power consumption vs. Frequency



Therefore, to build an accurate estimation of the overall power consumption, the wake-up power consumption has to be added into the total power estimation.

Let's look at an example to estimate the overall application current consumption. Assuming a run time period of 20 µs and Stop mode period of 10 s, Run mode represents 0.0002 % of time.

- Stop mode with RTC running: 1.3 µA
- Run mode: 420 µA @ 2 MHz
- Wake-up at 2 MHz with average current consumption of 21 µA

In this case, the total current consumption estimate is similar to the Stop mode with RTC current at about 1.3 µA. If the Stop mode period is reduced to 1 ms instead of 10 s, the total current consumption estimate is about 9.7 µA.

5.2 Low-power mode analysis versus datasheet values

Table 1. Low-power modes analysis versus datasheet value

Product/Low-power mode	Stop mode with RTC	Stop mode without RTC	Lowest power mode
MCU "A"	1.5 µA @2V	0.6 µA	20 nA
STM32L	3 µA @2V	0.5 µA	270 nA

Looking at the datasheet values, the STM32L has a better current consumption values except for the lowest power mode. However, the 20 nA needs special attention because this particular low-power mode of MCU "A" could not be used in most low-power applications as it needs power-on cycle or a reset to restart the device. It is important to consider the wake-up events from deep sleep mode. Many applications require wake-up from external events e.g. keyboard / buttons.

6 Ultra low-power mode considerations

The MCU hardware architecture makes the difference in ultra low-power applications. In addition to the sleep/deep sleep mode of the low-power MCU, other factors need special care when computing the power consumption. These factors include the analog peripherals and static power consumption.

The static power consumption is almost all attributed to the memory and can be reduced during Sleep mode or during Run mode by switching OFF the memory and executing code from RAM for a few software tasks.

The dynamic and static power consumption can be reduced progressively up to the point where most of the MCU is powered down:

- The non-volatile memory can be switched off
- The internal voltage regulator can be put in low-power (LP) mode
- The internal reference voltage consumption is not negligible, in particular in Stop and Standby mode. It can be switched off if it is possible with the chosen MCU.

Several low-power MCUs shutdown almost all peripherals and claim a few nA current consumption. The drawback of such a system is the long startup after wake-up to get the clocks and the internal reference voltage stabilized for analog peripherals. However, as demonstrated in the previous chapter, the energy wake-up has to be carefully studied in this case depending on the system wake-up frequency and wake-up period.

7 Low-power analog peripherals

In addition to the advanced low-power modes, several peripherals, specifically the analog peripherals, require a special focus as they have a major contribution in the overall consumption.

For example ADCs (Analog to Digital Converters) are widely used in low-power applications for data acquisition. Those applications require a very fast and accurate ADC which requires a fast clock to reach high conversion rates. The challenge is to keep a fast ADC conversion and the lowest current consumption.

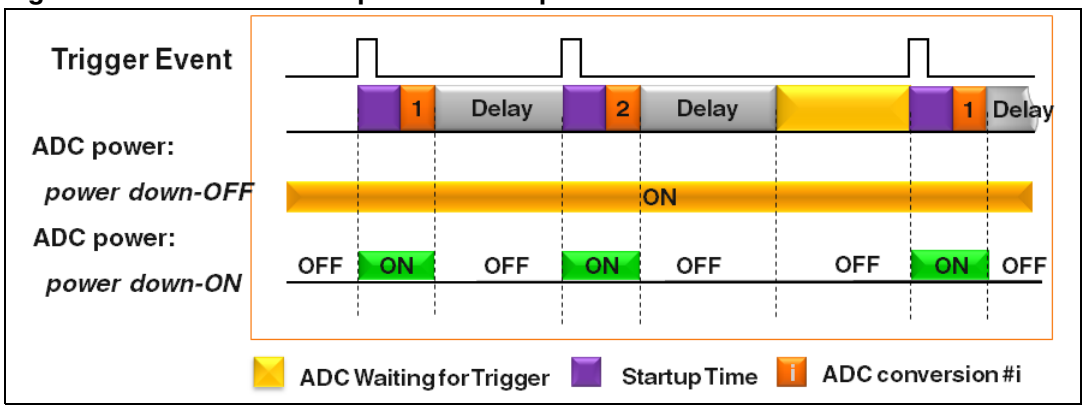
Most of the digital peripherals are switched off during the sleep/deep sleep modes, whereas the analog peripherals such as ADCs may remain powered so the application can continue to acquire data.

The ADC has to convert data whenever it is available. However once finished it is wasting energy while waiting for a trigger event to convert the next data. One possible scenario is to shutoff the ADC after each conversion and then re-enable for the next data conversion. However this requires the CPU to be in Run mode which also adds current consumption.

The time required to switch on and off a peripheral can also become relevant and critical for low operating CPU frequencies. At 1 MHz, each instruction may last as long as the ADC conversion itself.

A few low-power MCUs are starting to offer the ability to auto shutdown the ADC when it is not converting. One of the innovative features supported in STM32L ADC peripheral (12-bit / 1 MSps ADC with a fast power-up time of 3.5 µs) is a programmable auto shutdown period. It can be extended by adding a configurable delay or can be automatically adjusted along with the CPU operating frequency in order to have the best performance combined with very low-power consumption.

Figure 10. ADC Automatic power-on and power-off control



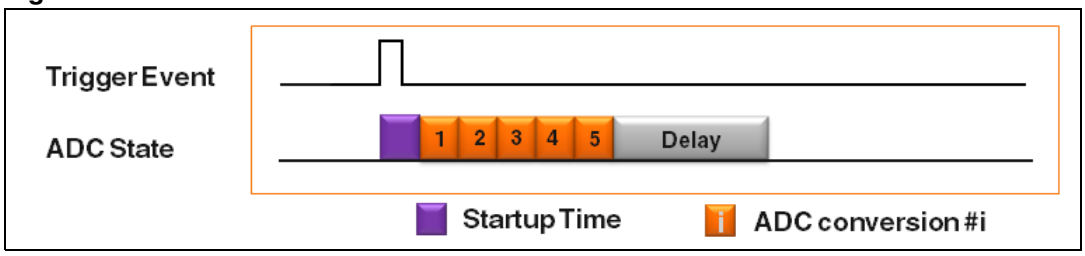
The chart in *Figure 10: ADC Automatic power-on and power-off control* illustrates the ADC automatic power management with and without the auto power down/shutdown mode activated. The ADC is powered down while there is no conversion and while waiting for a trigger event to start a new conversion to optimize the ADC current consumption as much as possible.

Also, a few low-power MCUs support the ADC burst conversion. To achieve the best ADC performance, you can launch a burst of up to 5 conversions that can be performed as a

response to a trigger conversion. It is also more efficient in terms of power consumption as the ADC needs a certain time to start up before a conversion can actually be launched.

Assume the system clock frequency is slowed down to 32 KHz and 1 ADC conversion lasts 1 µs while one instruction can be 31 µs. A burst of conversions can be launched between two cycles, with the ADC powered down automatically, so that the ADC extra consumption can be limited to the necessary time only (for 5 conversions it is 8.5 µs out of the 31 µs CPU period).

Figure 11. ADC in Burst Mode

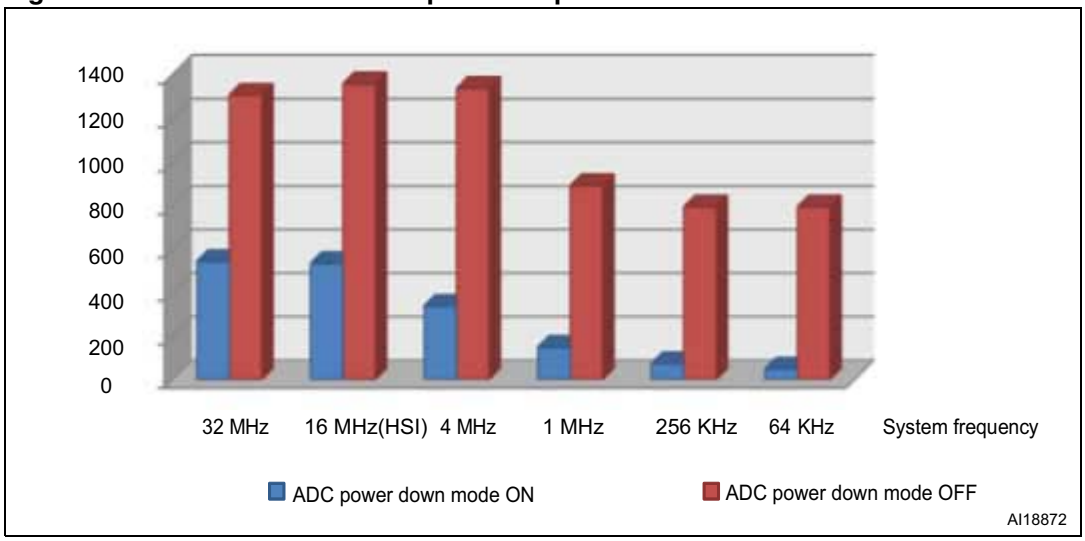


To demonstrate the ADC current consumption contribution with and without the power down saving feature, experiments on ADC current consumption were done under the following conditions:

Example 1: based on STM32L ADC:

- ADC in continuous mode
- A **Delay = 15 cycles** inserted between each channel conversion
- The conversion lasts 1 µs (16 cycles at 16 MHz). The ADC clock is independent from system clock.

Figure 12. ADC Current consumption comparison



7.1 Discussion

An accurate power consumption budget needs an efficient study of all MCU parameters. Factors to be examined include operating frequency, supply voltage, processing efficiency µA/MIPS, wake-up time and wake-up energy, low-power modes, hardware architecture, and analog peripherals.

This is not a complete list, but these are the major factors that influence the MCU current consumption and should help you when computing the power budget for your application. Ignoring these factors and referring only to current consumption in a datasheet can result in an inaccurate power estimation.

Selecting an ultra-low-power MCU which offers the low-power architecture and flexibility to adjust consumption to match the application needs should enable lower power consumption for extended battery life and enabling today's, and tomorrow's, greener applications.

8 Revision history

Table 2. Document revision history

Date	Revision	Changes
08-Apr-2011	1	Initial release.
21-May-2013	2	Updated T period in *Figure 4*.