AN4777

Application note

How to optimize power consumption on STM32 MCUs

Introduction

This application note applies to the X-CUBE-REF-PM expansion package for STM32Cube, which includes power-mode examples for STM32G0 series, STM32L0 series, STM32L1 series, and STM32L4 series microcontrollers.

The power consumption is the biggest advantage of low-power STM32 microcontrollers. The firmware example in this application note provides helpful hints on achieving the datasheet levels of power consumption and a simple framework to ease further experimentation with different configurations.

The low-power STM32 microcontrollers have a rich variety of configuration options for the flash memory interface.

While STM32G0 is not labeled as a low-power series, the feature set is similar and STM32G0 devices are small and have low power consumption.

This application note showcases the different settings in various test conditions, providing guidelines for the optimization of power efficiency, and is particularly focused on the influence of memory subsystem settings on the execution efficiency. This application note covers the subject with the same detail level as the product datasheets.

Referenced documents

Table 1. **Referenced documents**

Reference	Document number	Document title	Document type
[1]	RM0038	*STM32L100xx, STM32L151xx, STM32L152xx and STM32L162xx advanced Arm®-based 32-bit MCUs*	Reference manual
[2]	RM0351	*STM32L47xxx, STM32L48xxx, STM32L49xxx and STM32L4Axxx advanced Arm®-based 32-bit MCUs*	Reference manual
[3]	RM0367	*Ultra-low-power STM32L0x3 advanced Arm®-based 32-bit MCUs*	Reference manual
[4]	RM0444	*STM32G0x1 advanced Arm®-based 32-bit MCUs*	Reference manual
[5]	UM1724	*STM32 Nucleo-64 boards (MB1136)*	User manual
[6]	ES0548	*STM32G0B1xB/xC/xE device errata*	Errata sheet
[7]	ES0549	*STM32G0C1xC/xE device errata*	Errata sheet

All referenced documents are available on st.com.

1 General information

This document applies to Arm®-based devices.

Note: *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

Definitions

Table 2. **List of acronyms**

Term	Description
NV	Nonvolatile (memory), also referred to as flash memory
HSI	High-speed internal clock
SPI	Serial peripheral interface bus
MCU	Microcontroller
CPU	Central processing unit (part of the MCU)
NVIC	Nested vector interrupt controller
DMA	Direct memory access
SWD	Single wire debug interface

2 System architecture

The memory interface manages the read and write accesses from the core/bus matrix towards the nonvolatile memory. This holds for both the instruction and data access.

For configuring the nonvolatile memory read access during the program execution, the configuration flags are accessible in the access control register.

The latency serves the purpose of reducing the rate at which the NVM is read. An extra wait cycle must be enabled for a system clock higher than 16 MHz for the highest voltage regulator range. For lower core voltages, this threshold frequency is lower.

To compensate this bandwidth deficiency, a prefetch can be configured. The memory controller then attempts to have the next instruction ready before the core requests it.

The STM32L1 flash memory interface can use a 64-bit read access internally to be able to serve the core with data and instruction close to its own space. The extra 32 bits are used by the prefetch to load the next instruction and provide it to the core immediately when needed.

The STM32L0 flash memory interface does not have the 64-bit wide bus, but the memory controller is capable of data preread. This simple buffer is similar to the prefetch, but works also for data.

The STM32L4 flash memory interface has a full 64-bit wide (plus 8-bit ECC) connection to the bus matrix, shared between data and instruction. The flash memory interface incorporates an ART Accelerator, a prefetch mechanism and a cache designed to minimize the effect of memory latency. The flash memory interface is then capable of transferring data and instruction simultaneously, under the condition that they are ready in the cache.

The STM32G0 flash memory interface features prefetch and instruction cache, though smaller than on the L4. No cache is available for data read. It handles one or two banks of flash memory very similar to the situation found in the STM32L4. Native word width is 64-bit plus 8-bit of ECC.

All performance improvements resulting from the memory interface settings come at the cost of increased power consumption. Access with no latency, no preread, no cache, and no prefetch is used in the low-power mode. The following section sheds light on the kind of tradeoffs that the performance improvements represent.

3 Low-power modes

The main focus points of this application note are the run modes and efficiency of the code execution, which are not covered by the datasheets.

For the sake of completeness, the low-power modes must be mentioned. It means the states in which the CPU core cannot execute any code and only the selected subset of peripherals are active.

The following table compares the low-power modes across the MCU series covered by this application note:

Table 3. **Low-power mode brief comparison**

Low-power mode	MCU series		
	STM32L0, STM32L1	STM32L4	STM32G0
Sleep modes	Either main or low-power regulator, flash memory clock off with low-power sleep	Low-power regulator on, main regulator configurable, flash memory clock configurable	Either main or low-power regulator, flash memory state in low power mode configurable
Stop modes	Single stop mode	Stop0, Stop1, and Stop2 steps	Stop0 and Stop1
Standby	Available	Available and also special shutdown mode implemented	Available and shutdown mode as well

For more details about the listed low-power modes, refer to the product reference manuals and datasheets.

4 Operation modes

The different operation modes are used to assess the impact of the memory interface settings on the performance and power consumption. All measurements have been done using V_{CC} = 3.3 V and the voltage regulator range 1. The speed and consumption would be lower using lower regulator levels, but linearly lower relative to the range 1 measurements. For example, with the voltage regulator range 3 and the system clock speed at 2 MHz (from MSI), the power consumption would be roughly ten times lower for all the measurements and the performance roughly ten times lower for all the measured configurations. There is no point in repeating the measurement for all the configuration combinations.

4.1 STM32G0 series device options

Up to two wait states may be configured on the STM32G0 series. Operation with zero latency is permitted up to 24 MHz in main regulator range 1 and to 8 MHz in range 2.

Table 4. **The options in voltage regulator range 1**

Frequency	≤ 24 MHz		≤ 48 MHz				≤ 64 MHz			
Latency	0	0	1	1	1	1	2	2	2	2
Instruction cache	0	1	0	0	1	1	0	0	1	1
Prefetch	0	0	0	1	0	1	0	1	0	1

While it is possible to enable prefetch regardless of latency setting, it makes no sense when number of wait states is zero. In range 2 the system clock is capped at 16 MHz, which is achieved with 1 wait state. For more details, refer to chapter 3.3.4 in document [4].

4.2 STM32L1 series device options

Table 5 lists a short summary of the device options for the STM32L1 series. For a detailed description, refer to the "read interface" section of document [1].

Table 5. **Configurations available on STM32L1 series devices with regulator range 1**

Frequency	<16 MHz				>16 MHz	
Latency	0	0	1	1	1	1
64-bit	0	1	1	1	1	1
Prefetch	0	0	0	1	0	1

The table of valid configurations clearly demonstrates the following rules:

- Wait states are inevitable when exceeding 16 MHz.
- When the latency is set to 1, the 64-bit access is mandatory.
- The prefetch is impossible without the 64-bit access.

4.3 STM32L0 series device options

Table 6 lists a short summary of the device options. For a detailed description, refer to reading the NVM section of document [3].

Table 6. **Configurations available on STM32L0 series devices with regulator range 1**

Frequency	<16 MHz								>16 MHz				
Latency	0	1	0	1	1	1	0	1	1	1	1	1	1
Preread	0	0	1	1	1	0	X	X	0	1	1	0	X
Prefetch	0	0	0	0	1	1	X	X	0	0	1	1	X
Buffer disable	0	0	0	0	0	0	1	1	0	0	0	0	1

The table of valid configurations clearly demonstrates the following rules:

- The latency cannot be zero with clock speeds exceeding 16 MHz.
- When the buffer is disabled, it cannot be configured.
- Prefetch and preread configure the usage of the six words in the internal buffer, not their total amount.

4.4 STM32L4 series device options

Table 7 lists a short summary of the device options. For a detailed description, refer to the "reading the NVM" section of document [4].

Table 7. **STM32L4 series device option summary**

Frequency	<16 MHz (at V_{CORE} range 1)				>16 MHz (at V_{CORE} range 2)							
Latency	0				>1							
Data cache	0	0	1	1	0	0	0	1	0	1	1	1
Instruction cache	0	1	0	1	0	0	1	0	1	1	0	1
Prefetch	0	0	0	0	0	1	0	0	1	0	1	1

The prefetch, data cache, and instruction cache settings are independent of each other. Each of these three features can be enabled or disabled independently of the frequency or any other setting. However, some settings make less sense than others, especially with zero wait states, prefetch is definitely not recommended.

The settings are only simple when the voltage regulator settings are disregarded. But the read access latency strongly depends on the voltage regulator settings. For example, at a 16-MHz speed, while with range 1 the latency on a flash memory read is 1 CPU cycle, with range 2 the latency on the same core frequency increases to 3 CPU cycles.

For more details, refer to the "read access latency" section in document [4].

4.5 Execution from a volatile memory

The intuitive way to avoid the flash memory speed issues would be to use the RAM for selected portions of code. There are several reasons not to do that.

1. The RAM is a scarce resource on small devices.
2. Most of data are likely to be placed in the RAM, accessing the code in the RAM eliminates the advantage of Harvard architecture (separate data and instruction buses) approach in the STM32L1 and STM32L4 series.
3. To switch off the flash memory and conserve more energy, the interrupt table and interrupt handlers also need to be in the RAM.

In the case of a typical microcontroller application, the overall energy budget of the RAM execution is roughly the same as the execution on the 32-MHz system clock with the flash memory latency set. Which means that if the flash memory can run without the latency enabled, it is a better option most of the time. In other words, the RAM execution tends to be about 30% slower than the execution of the same code from the flash memory and the current consumption does not decrease more than the same 30% range.

Note: When the decision is taken to use the RAM for code storage, the address on which the code is stored within RAM may play a significant role in the power consumption figures. This note is not only relevant for the STM32L4 series. Because the principle behind this behavior cannot be generally described for every configuration and use case, it is best to figure out the optimal placement by experimenting with the application during development, especially if the product features several separate sections of RAM with different properties.

5 Reproducing the measurements to get datasheet values

The STM32Cube Expansion Package (X-CUBE-REF-PM) related to this application note is intended for use with cheap and widespread STM32 Nucleo application boards. With some effort, the examples can be adapted for other hardware boards. The descriptions in this chapter refer to Nucleo boards.

5.1 Hardware and prerequisites

For simplicity sake the examples are using the VCP UART embedded in the STLINK for the UI and controls. Only a single USB cable is used to power and control the tested Nucleo board. A terminal emulator program is necessary on the PC to connect to the virtual COM port. Tera Term 4.84 is used in the testing.

The Nucleo board is not equipped with any power sensing capability. However, it is equipped with a JP6 jumper that can be replaced with an ampere meter or any other current sensing device. For details refer to document [5].

The X-NUCLEO-LPM01 energy monitor device used in this example development is an ideal choice of current and energy monitor device for the Nucleo board.

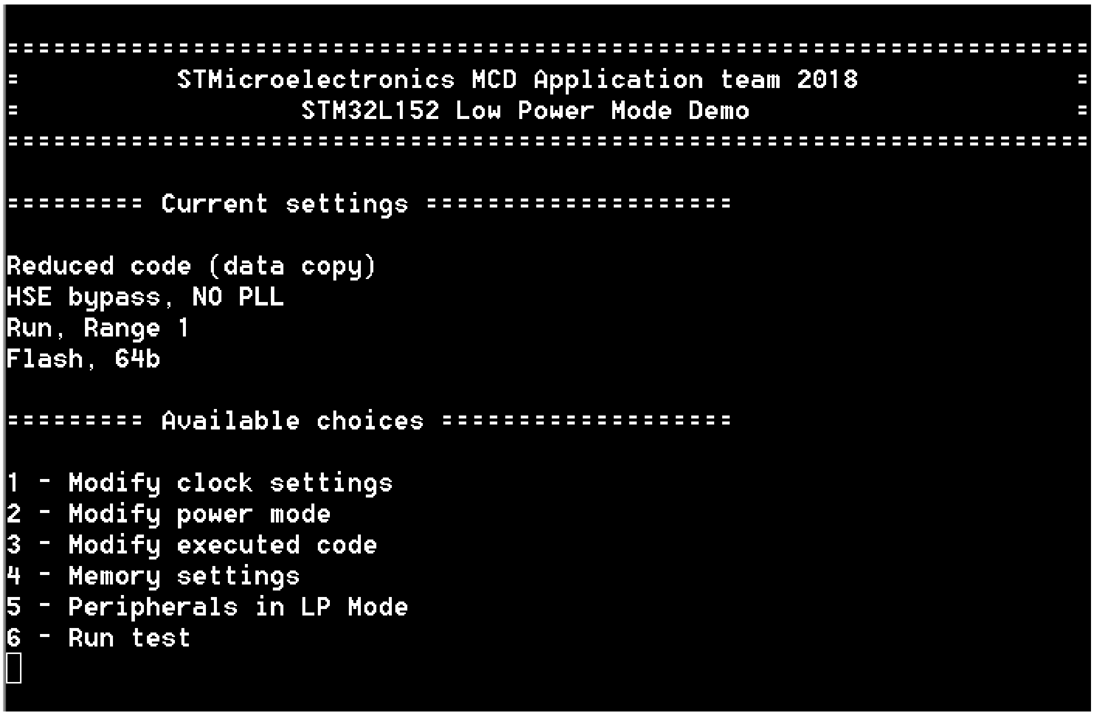
For measurements involving the HSE bypass, a clock source is necessary. Nucleo boards however provide an option to use 8 MHz MCO from STLINK as the HSE clock. Some solder bridges may need to be modified for this. See the relevant Nucleo user manual for details.

5.2 Example operation

Configure the terminal emulator application to 9600Bd, 8-bit, no parity.

The firmware is loaded and executed, and the terminal displays the following screen:

Figure 1. Terminal screen



```
======================================================================
=          STMicroelectronics MCD Application team 2018              =
=                  STM32L152 Low Power Mode Demo                     =
======================================================================

========= Current settings ====================

Reduced code (data copy)
HSE bypass, NO PLL
Run, Range 1
Flash, 64b

========= Available choices ===================

1 - Modify clock settings
2 - Modify power mode
3 - Modify executed code
4 - Memory settings
5 - Peripherals in LP Mode
6 - Run test
```

All controls are implemented as number key press inputs, with choices listed on the bottom of the screen. The choices are not available at all times.

The control firmware deliberately tries to hide settings that are not applicable. For example, when a low-power mode is selected, the executed code selection is hidden as not relevant.

Enter the number corresponding to the available choices (selections 1-5).

In case of another selection, the terminal asks for a new value. Once the choice is made, updated settings are listed. For example, when the low-power run mode is selected, the oscillator settings are adjusted to produce a compatible system clock.

To execute a test, first set the power mode: it determines the available system clock settings and the test availability. For low-power mode the active peripherals may be selected, for run modes the executed code may be selected.

The firmware tries to limit the access to some of the setting combinations, that would obviously lead to failure. However, especially when using the HSE clock source it is still possible to leave the operating conditions envelope defined in the datasheets. The correct operation is then not guaranteed.

To start the test execution, enter ‘6’ in the root menu. In case of failure, the firmware activates the on-board LED.

5.3 Test configurations explained

The example firmware may be built with several different options.

Table 8. **The example build options**

Define	Active	Not active
EXTI_BUTTON	Blue button on the Nucleo board abort most of the tests, returning into the root menu, retaining settings. May cause additional current consumption.	Reset button on the Nucleo board is used to return into the root menu. Settings are however reset to default values.
FINITE_LOOP	Relevant computational tests are limited to LOOP_COUNT cycles. Measuring the time to complete the task is used to compute the execution efficiency.	Tests run until aborted by reset, power off, debugger or EXTI (list depending on other options).
DEBUG_ON	Debug interface is active during the test. Useful to review the settings and check the functionality.	Debug interface is in high-Z during the test. Only this code must be ever used for measurement.

The default setting ‘with all three define switches not active’, is the configuration, which allows the user to obtain the datasheet values.

The datasheet includes the power consumption measurements for several different codes executed. These are Dhrystone, CoreMark, Fibonacci, and while(1) loop. The CoreMark is not included in the published example code for licensing reasons. But the example includes two additional test codes instead. The “Reduced code” and “Memory read stress test” are focusing directly on the memory interface settings and their influence on the execution efficiency.

The flash memory interface efficiency focused tests are not present in the datasheet. The results of their execution are analyzed in the following pages.

6 Power consumption and performance comparison using STM32L1 series devices

To assess the performance of the MCU with different memory controller settings, several benchmark tests have been used. All tests have been executed on a NUCLEO-L152RE board using all available memory interface settings, listed in Section 4.2: STM32L1 series device options. All tests have been executed both in standalone and in parallel with a DMA transfer, constantly reading from the program NV memory. The DMA channel was directed to the SPI output configured to the highest available speed ($f_{PCLK}/2$) and low priority.

Three clock configurations have been used in the measurements. One with the plain 16 MHz HSI clock as the system clock and no latency set, another with the same clock but the flash memory latency configured (flash memory running effectively on lower clock) and the third with the PLL set to produce the 32 MHz system clock.

All the measurements are taken on a single sample of NUCLEO-L152RE board at ambient temperature. The values provided are an arithmetic mean from several measurements.

6.1 STM32L1 Dhrystone benchmark

Although the Dhrystone benchmark is often deemed outdated, it is still somewhat representative of many microcontroller applications.

Table 9. **Dhrystone results with no background transfer**

Frequency	16 MHz					32 MHz	
Latency	0	0	0	1	1	1	1
64-bit	0	1	1	1	1	1	1
Prefetch	0	0	1	0	1	0	1
Timing for 50000 cycles [s]	2.57	2.57	2.57	3.05	2.86	1.52	1.46
Average current [mA]	5.75	5.78	6.11	5.13	5.62	10.42	11.08
Energy [mJ]	48.77	49.02	51.82	51.63	53.04	52.27	53.38

Figure 2. Dhrystone results with no background transfer

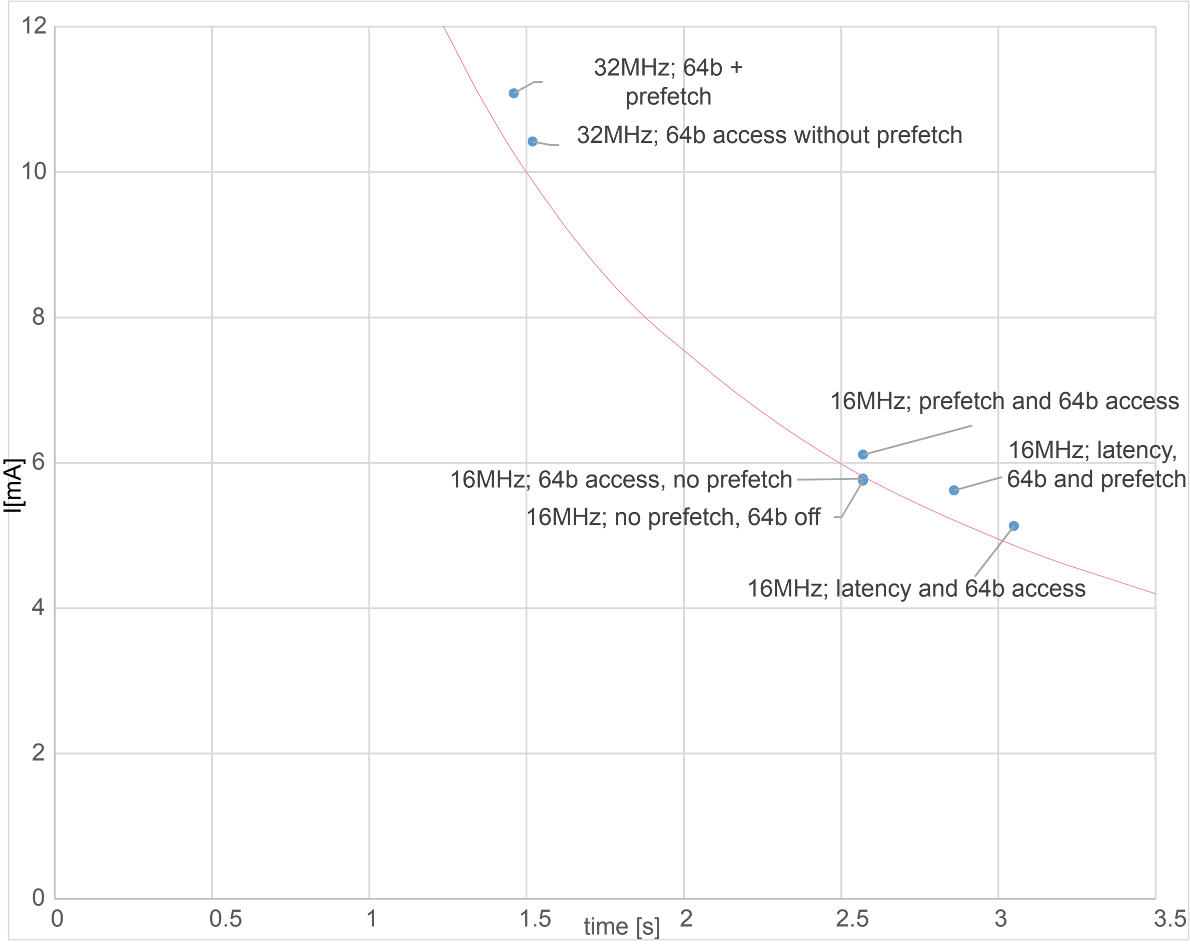
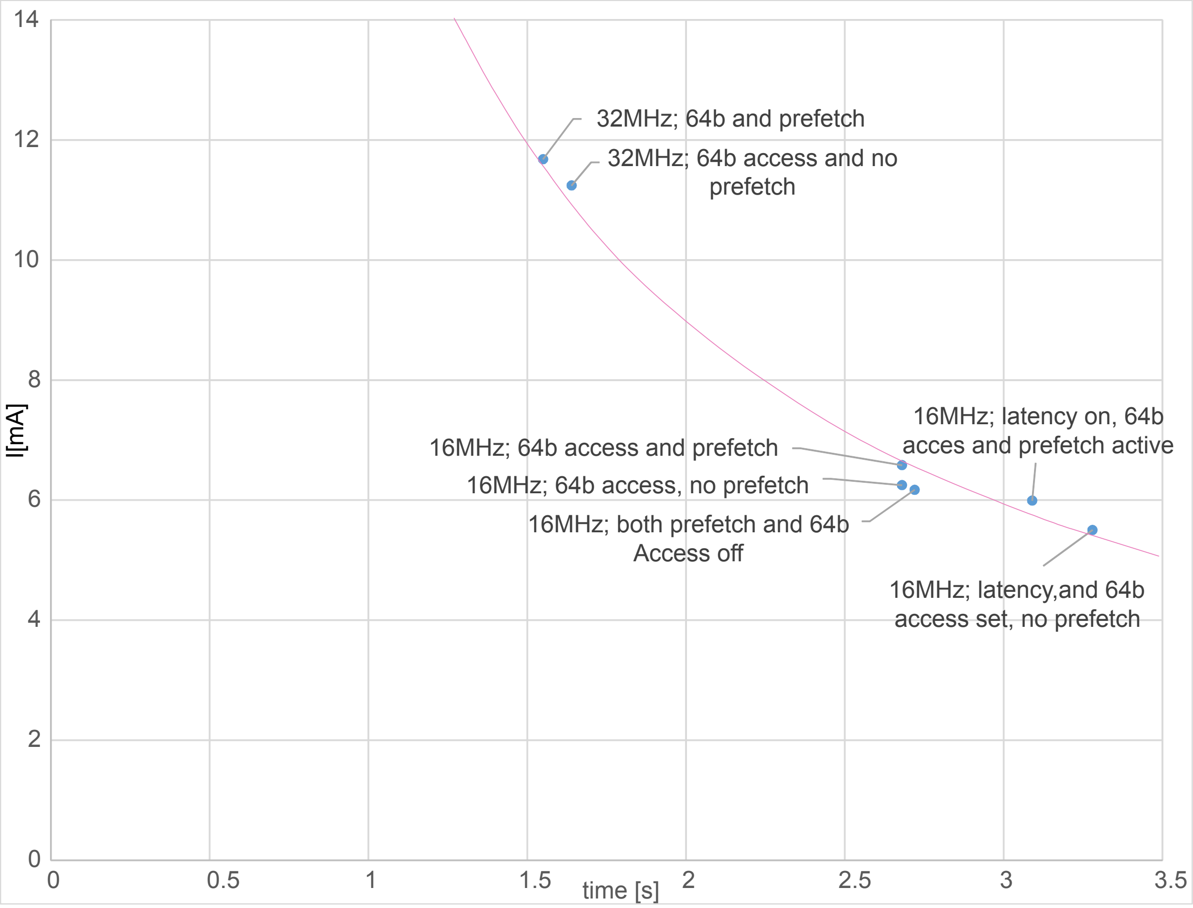


Table 10. Dhrystone results with DMA simultaneously reading data from the flash memory

Frequency	16 MHz					32 MHz	
Latency	0	0	0	1	1	1	1
64-bit	0	1	1	1	1	1	1
Prefetch	0	0	1	0	1	0	1
Timing for 50000 cycles [s]	2.72	2.68	2.68	3.28	3.09	1.64	1.55
Average current [mA]	6.17	6.25	6.58	5.50	5.99	11.24	11.68
Energy [mJ]	55.38	55.28	58.19	59.53	61.08	60.83	59.74

Figure 3. Dhrystone results with DMA simultaneously reading data from the flash memory



Configuring a 64-bit access or a prefetch makes a very small difference on a low clock speed where the latency can be avoided. On the contrary, setting the latency may lead to a lower power consumption in situations where the speed is not critical. At higher speeds the efficiency of the prefetch is situational, leading to ultimate performance but the gain in speed may be lower than the consumption increase.

6.2 32-bit instruction code

A stress test consists of executing 12 aligned 32-bit instructions manipulating data in registers in a loop of 500000 cycles. The code with a higher ratio of 32-bit instructions is more likely to find a bottleneck in the memory interface than a typical Thumb code with prevalent 16-bit instructions.

Table 11. 32-bit code result with no background transfer

Frequency	16 MHz					32 MHz	
Latency	0	0	0	1	1	1	1
64-bit	0	1	1	1	1	1	1
Prefetch	0	0	1	0	1	0	1
Timing for 500000 cycles [s]	0.9	0.9	0.9	1.06	0.964	0.59	0.497
Average current [mA]	5.25	5.41	5.63	4.82	5.11	9.09	9.78
Energy [mJ]	15.59	16.07	16.72	16.86	16.26	17.70	16.04

Figure 4. **32-bit code result with no background transfer**

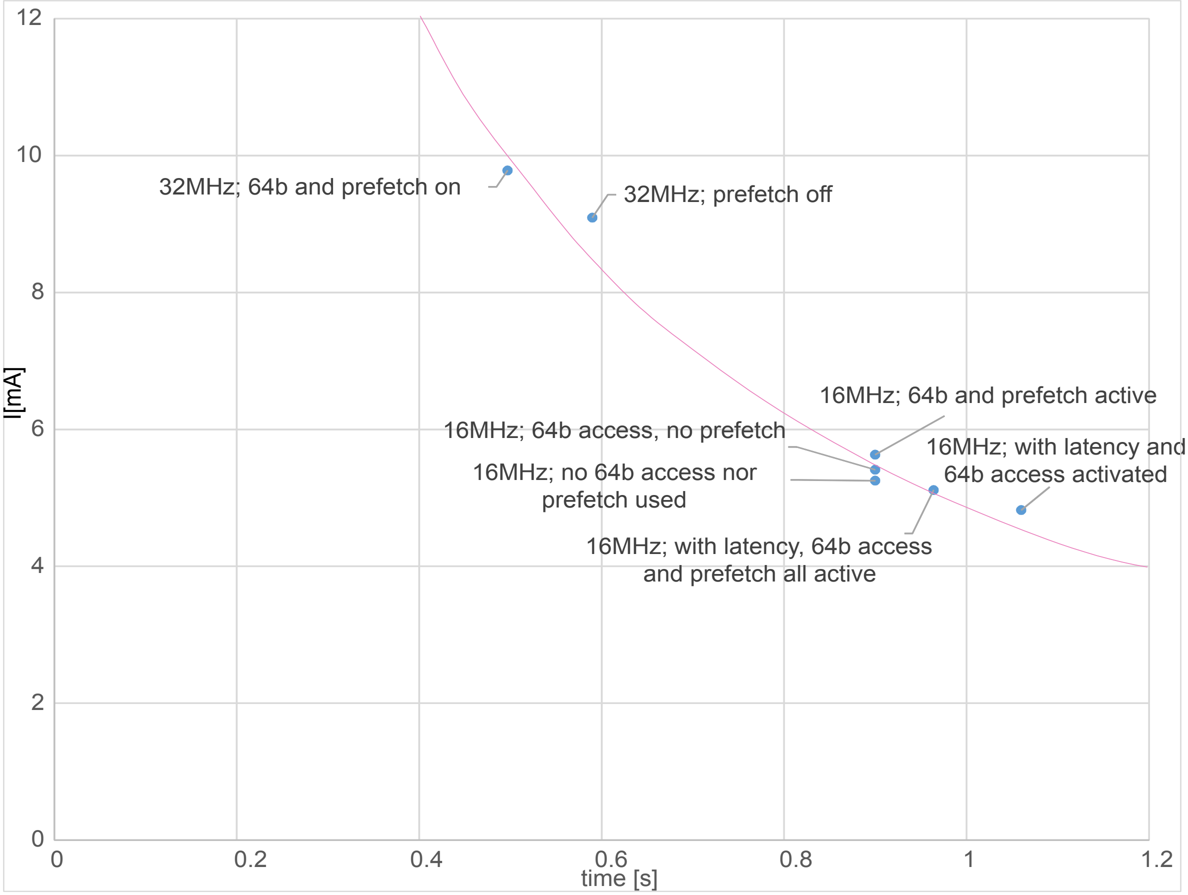
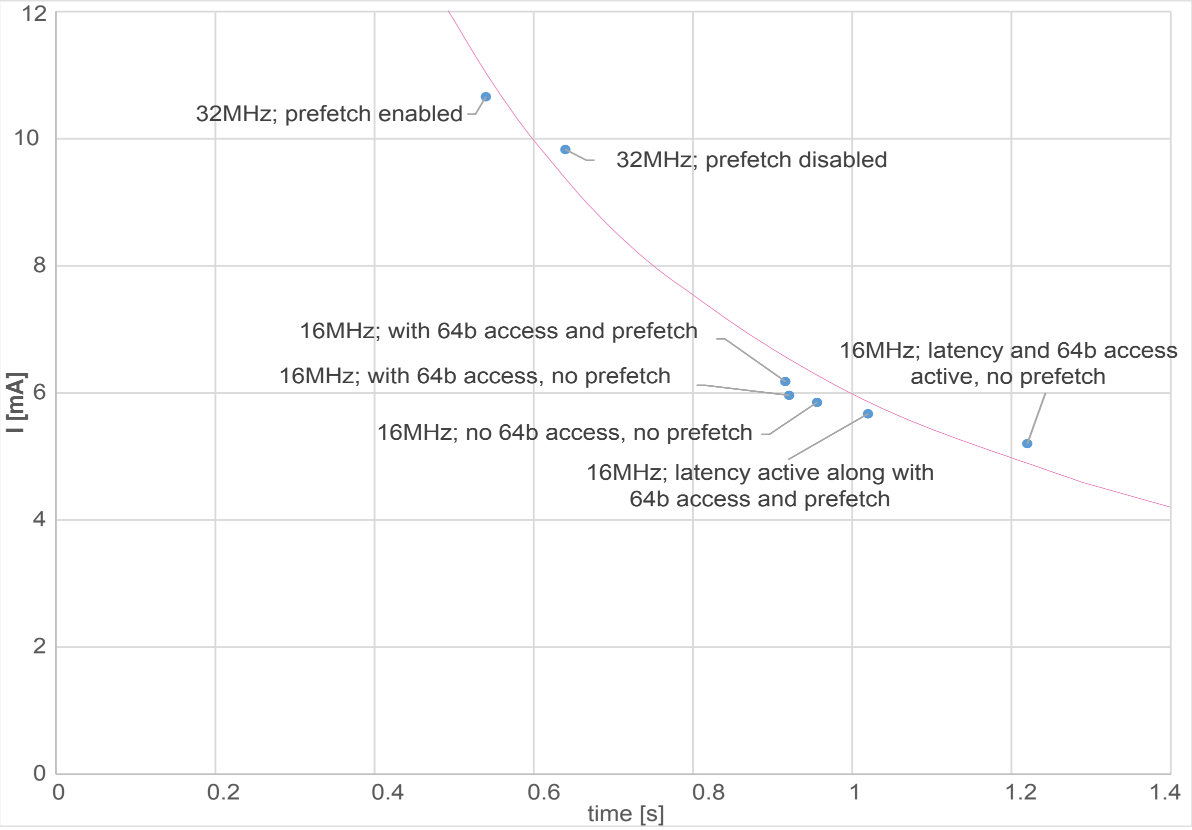


Table 12. **32-bit code result with DMA simultaneously reading data from the flash memory**

Frequency	16 MHz					32 MHz	
Latency	0	0	0	1	1	1	1
64bit	0	1	1	1	1	1	1
Prefetch	0	0	1	0	1	0	1
Timing for 500000 cycles [s]	0.956	0.921	0.916	1.22	1.02	0.64	0.54
Average current [mA]	5.85	5.96	6.18	5.20	5.67	9.83	10.66
Energy [mJ]	18.46	18.11	18.68	20.94	19.09	20.76	19.00

Figure 5. **32-bit code result with DMA simultaneously reading data from the flash memory**



The findings are in line with the expectations: a code with high share of 32-bit instructions benefits a lot from the prefetch once the memory latency is in place. But with zero latency the extra bandwidth is likely to be useless.

6.3 STM32L1 memory read stress test

A stress test consists of executing 20 LDR instructions fetching data from the program NV memory to the CPU core registers in a loop of 500000 cycles. This way, not only the instructions are fetched from the memory but another read access is generated during the instruction execution, again creating a choke point at the memory interface. Fetching of subsequent instruction is then likely to be delayed. The code simulates a case when a heavy load of literal pools (string constants) like for example predefined messages, is read from a non-volatile memory very often.

Note: The memory reading by LDM instructions is not used as it is not demonstrating limits of the memory interface, only the memory itself.

Table 13. **Literal pool with no additional data read from the flash memory**

Frequency	16 MHz					32 MHz	
Latency	0	0	0	1	1	1	1
64-bit	0	1	1	1	1	1	1
Prefetch	0	0	1	0	1	0	1
Timing for 500000 cycles [s]	3.66	2.73	2.72	3.38	3.32	1.69	1.66
Average current [mA]	5.44	5.58	6.12	4.85	5.33	9.78	10.73
Energy [mJ]	65.70	50.27	54.93	54.10	58.40	54.54	58.78

Figure 6. Literal pool reading with no additional data read from the flash memory

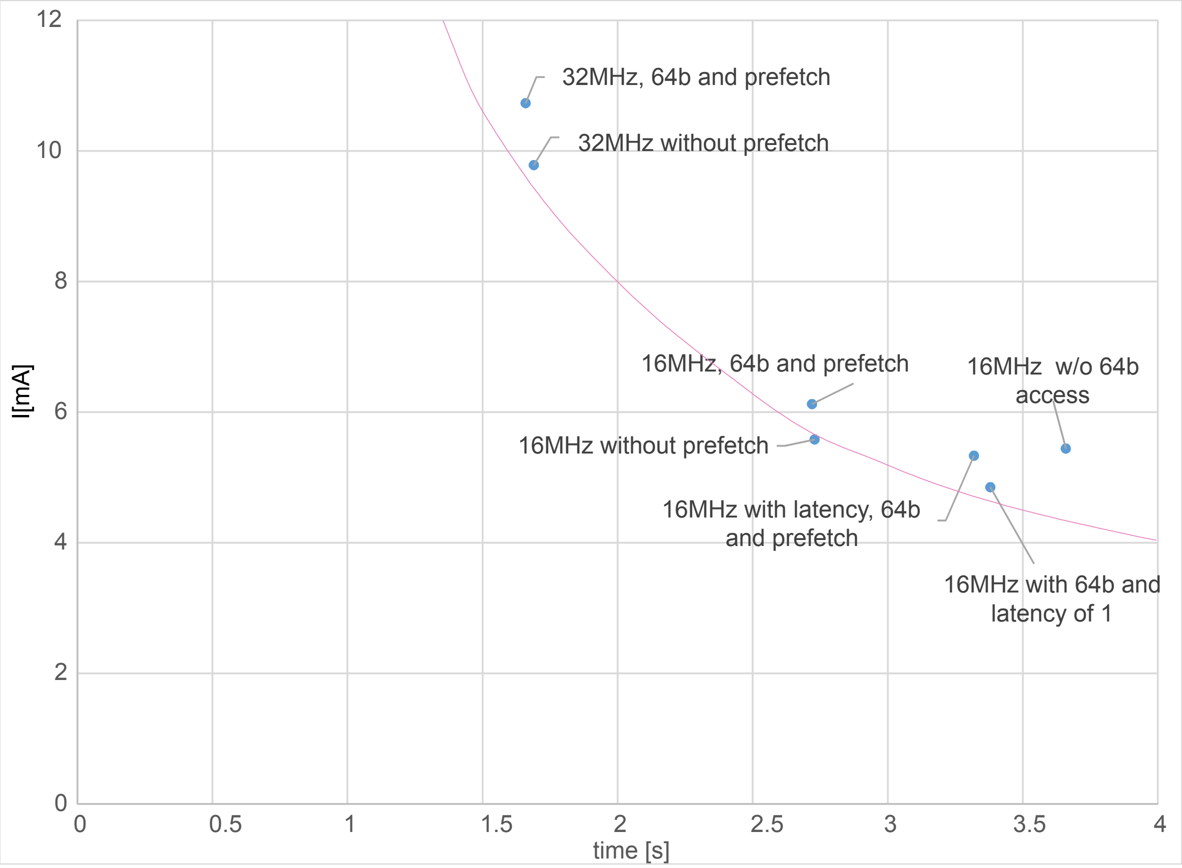
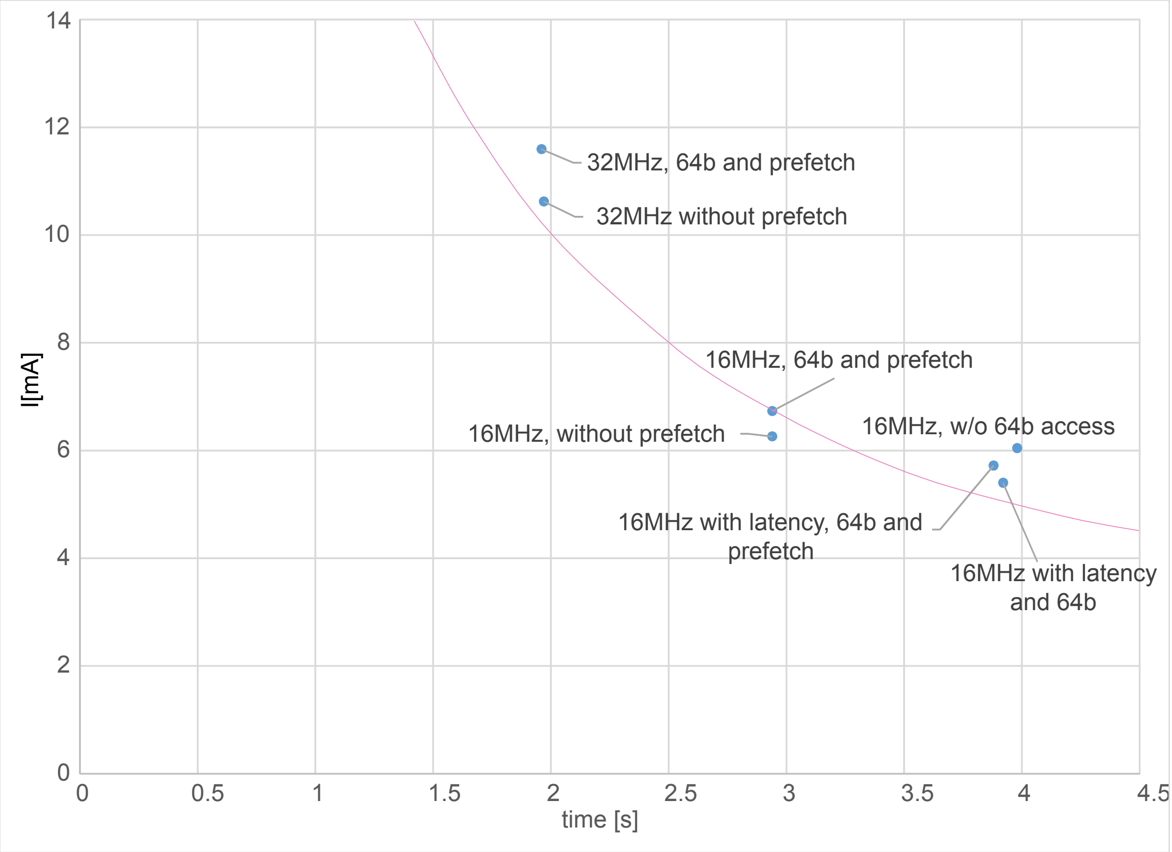


Table 14. Literal pool reading with DMA simultaneously reading the Flash memory

Frequency	16 MHz					32 MHz	
Latency	0	0	0	1	1	1	1
64-bit	0	1	1	1	1	1	1
Prefetch	0	0	1	0	1	0	1
Timing for 500000 cycles [s]	3.98	2.94	2.94	3.92	3.88	1.97	1.96
Average current [mA]	6.04	6.26	6.73	5.40	5.72	10.62	11.59
Energy [mJ]	79.33	60.73	65.29	69.85	73.24	69.04	74.96

Figure 7. **Literal pool reading with DMA simultaneously reading data from the Flash memory**



As expected, mostly in case of a data read transfer the effect of the prefetch is lower, but a 64-bit memory access makes a significant difference even with zero memory latency.

7 Power consumption and performance comparison using STM32L0 series devices

The Cortex®-M0+ core is much simpler compared to the Cortex®-M3 used in the STM32L1 series. The 32-bit instruction benchmark is dropped as the Thumb-2 instruction set support in the Cortex®-M0+ core is very limited and an extensive usage of 32-bit code is not realistic with a code compiled for the STM32L0 Series.

The remaining tests have been executed on a NUCLEO-L073RZ board using all available memory interface settings, listed in Section 4.3: STM32L0 series device options. All the tests have been executed both standalone and in parallel with a DMA transfer constantly reading from the program NV memory. The DMA channel was directed to the SPI output configured to the highest available speed ($f_{PCLK}/2$), but low priority.

Two clock configurations have been used in the measurements. One with the plain 16-MHz HSI clock as the system clock and no latency set, the other with the PLL set to produce the 32-MHz system clock and the flash memory latency set to 1.

All measurements are taken on a single sample of NUCLEO-L073RZ board at ambient temperature. The values provided are an arithmetic mean from several measurements.

7.1 STM32L0 Dhrystone benchmark

The Dhrystone code is executed and the task consists of processing 50000 cycles of the test code.

Table 15. **Dhrystone with no additional data read from the flash memory**

Frequency	16 MHz					32 MHz				
Latency	0	0	0	0	0	1	1	1	1	1
Prefetch	1	0	0	1	0	1	0	0	1	0
Preread	1	1	0	0	0	1	1	0	0	0
Disabled buffer	0	0	1	0	0	0	0	1	0	0
Time [ms]	3769	3766	3771	3769	3769	2139	2667	2720	2130	2667
Average current [mA]	4.32	4.42	4.54	4.40	4.39	8.14	7.52	7.52	8.04	7.43
Energy [mJ]	53.73	54.93	56.49	54.72	54.60	57.46	66.20	67.49	56.51	65.40

Figure 8. Dhrystone with no additional data read from the flash memory

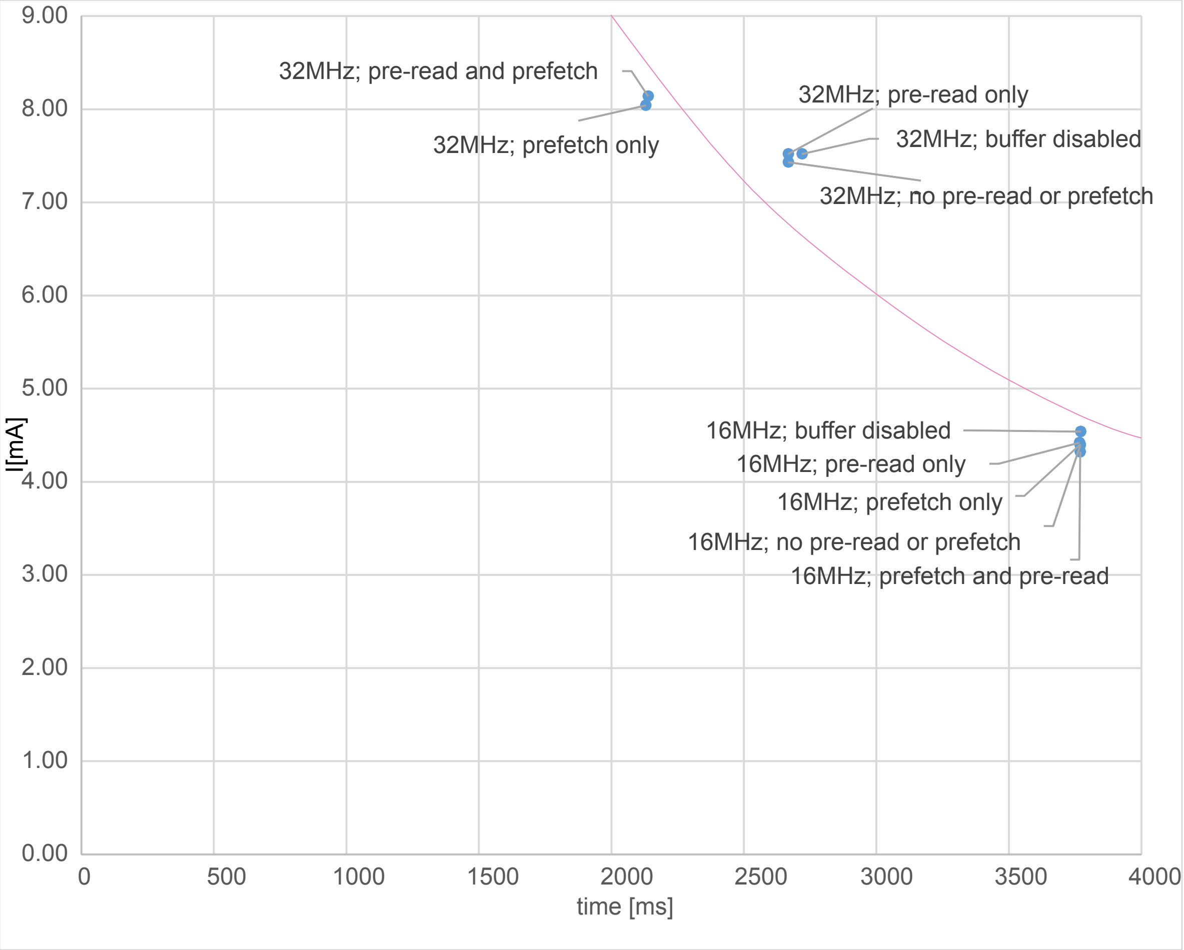
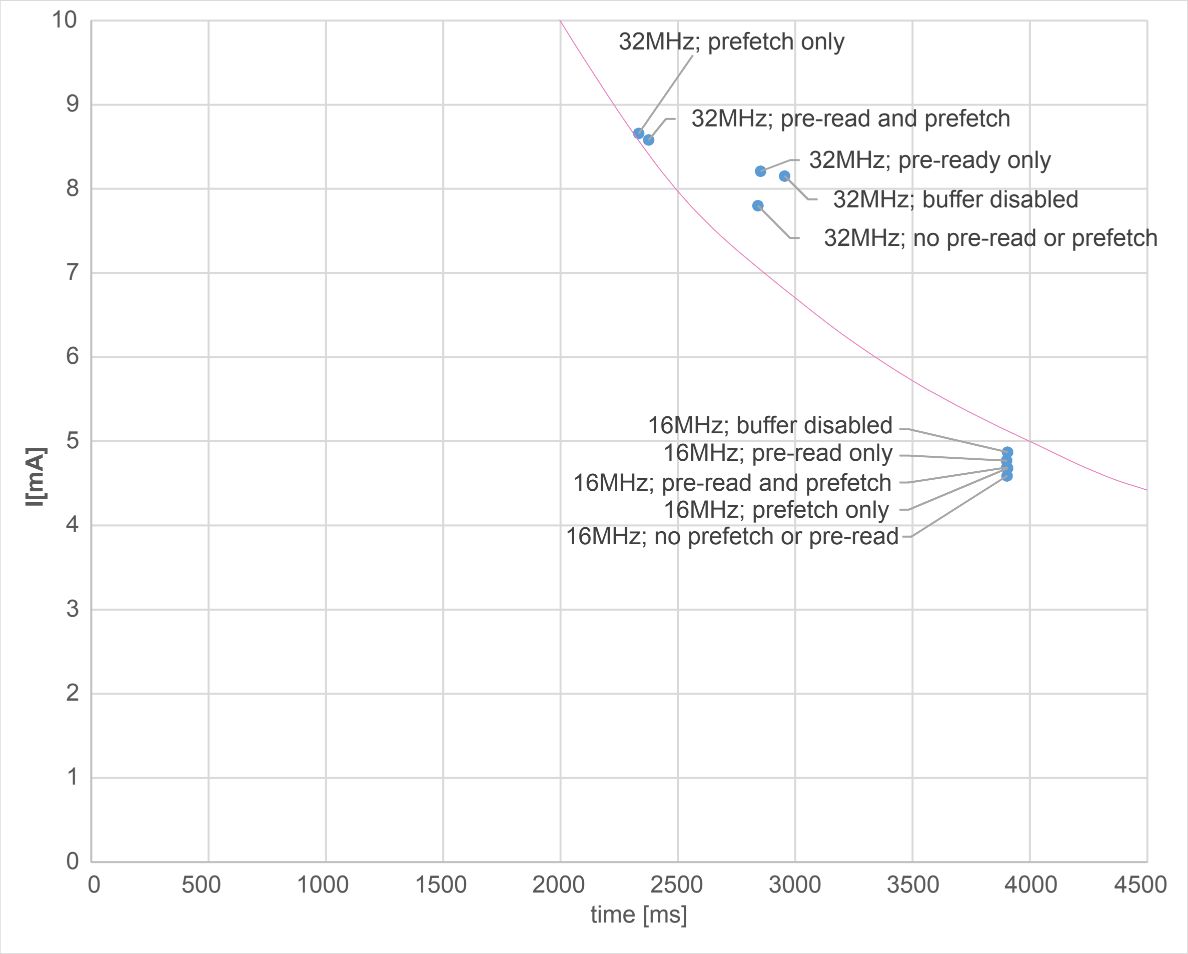


Table 16. Dhrystone with DMA simultaneously reading data from the flash memory

Frequency	16 MHz					32 MHz				
Latency	0	0	0	0	0	1	1	1	1	1
Prefetch	1	0	0	1	0	1	0	0	1	0
Preread	1	1	0	0	0	1	1	0	0	0
Disabled buffer	0	0	1	0	0	0	0	1	0	0
Time [ms]	3903	3901	3906	3906	3904	2377	2853	2956	2334	2843
Average current [mA]	4.69	4.77	4.87	4.68	4.59	8.58	8.21	8.15	8.66	7.80
Energy [mJ]	69.40	61.41	62.77	60.32	59.13	67.29	77.31	79.31	66.70	73.17

Figure 9. **Dhrystone with DMA simultaneously reading data from the flash memory**



This example clearly shows that the internal six word buffer improves the energy efficiency even if it is not well utilized, like in case of zero latency. The best option is to keep it on, but to disable the prefetch and preread.

In case of the configuration with the latency is enabled, the prefetch is probably worth using. The preread is obviously not used by the DMA channel and does not represent an improvement in this particular scenario.

7.2 STM32L0 memory read stress test

A stress test consists of executing 20 LDR instructions fetching data from program NV memory to CPU core registers in a loop of 500000 cycles. This way, not only the instructions are fetched from the memory but another read access is generated during the instruction execution, again creating a choke point at the memory interface. Fetching of subsequent instruction is then likely to be delayed. The code simulates a case when a heavy load of literal pools, like for example predefined messages, is read from a non-volatile memory very often.

Note: *The memory reading by LDM instructions are not used as it is not demonstrating limits of the memory interface, only the memory itself.*

Table 17. Literal pool with no additional data read from the flash memory

Frequency	16 MHz					32 MHz				
Latency	0	0	0	0	0	1	1	1	1	1
Prefetch	1	0	0	1	0	1	0	0	1	0
Pre-read	1	1	0	0	0	1	1	0	0	0
Disabled buffer	0	0	1	0	0	0	0	1	0	0
Time [ms]	2402.5	2401.5	2403	2403	2399.5	2009	2058.5	2091	1817	1819
Average current [mA]	3.4	3.42	3.36	3.14	3.19	6.03	6.05	5.94	5.83	5.73
Energy [mJ]	26.95	27.10	26.64	24.89	25.25	39.97	41.09	40.98	34.95	34.39

Figure 10. Literal pool with no additional data read from the flash memory

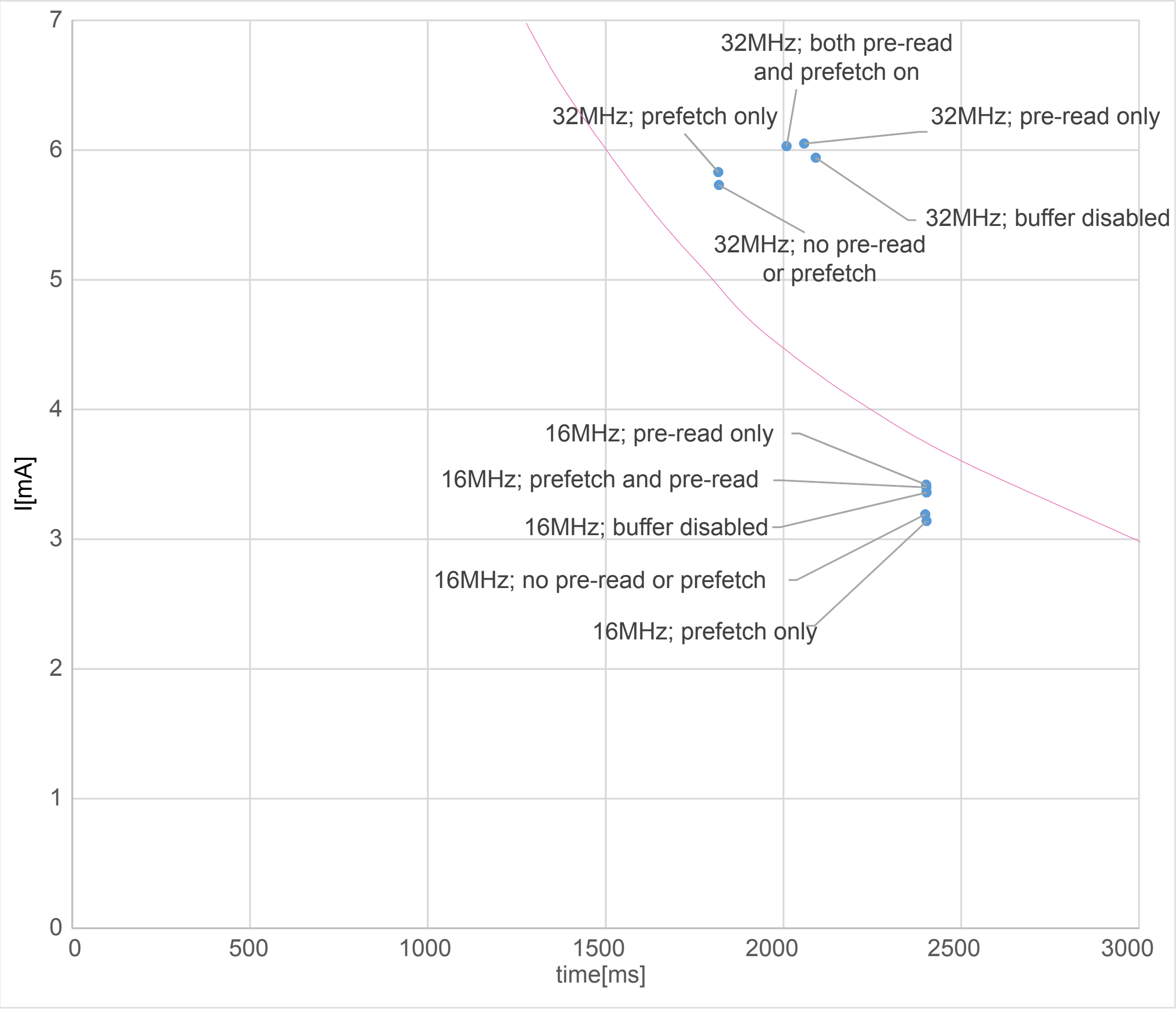
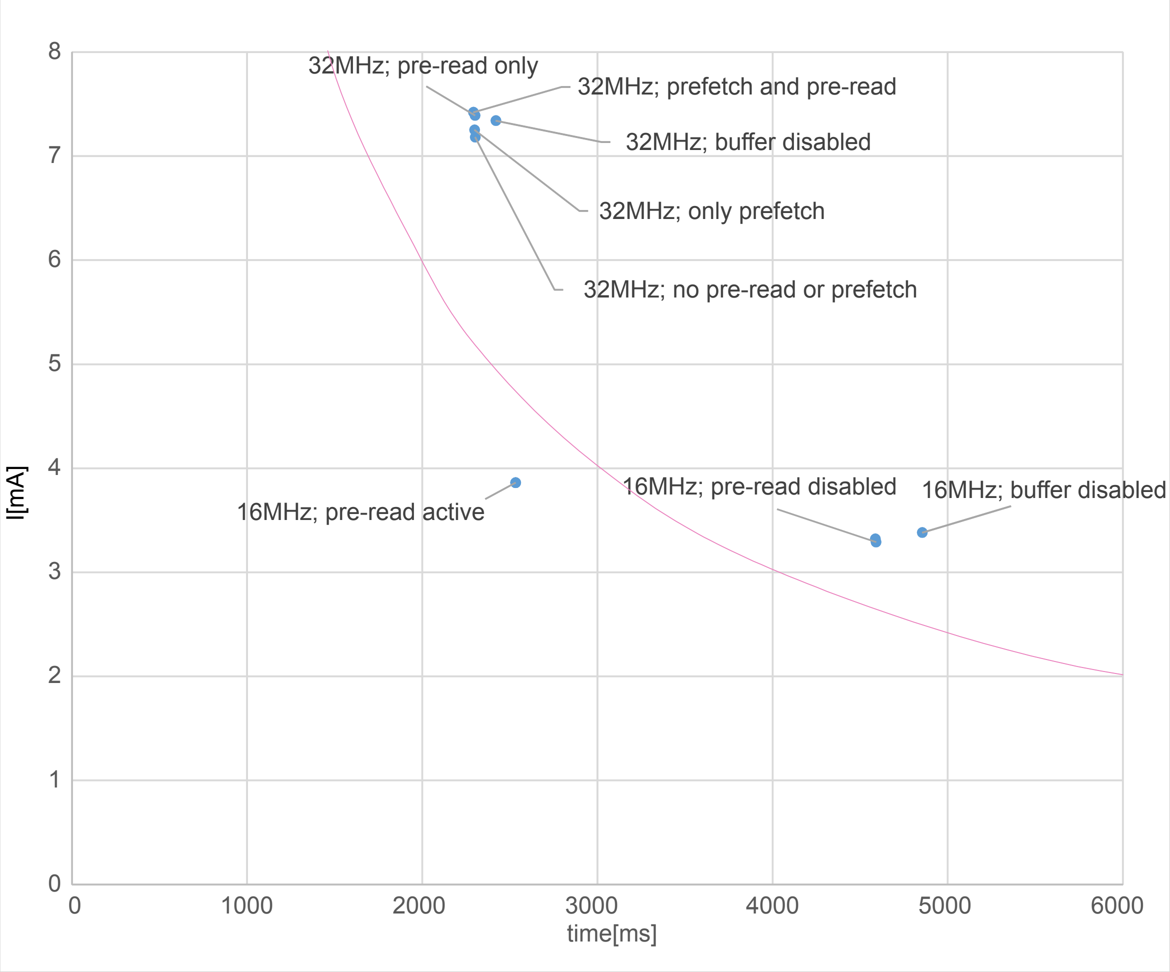


Table 18. **Literal pool with DMA simultaneously reading data from the flash memory**

Frequency	16 MHz					32 MHz				
Latency	0	0	0	0	0	1	1	1	1	1
Prefetch	1	0	0	1	0	1	0	0	1	0
Pre-read	1	1	0	0	0	1	1	0	0	0
Disabled buffer	0	0	1	0	0	0	0	1	0	0
Time [ms]	2533.5	2533.5	4854.5	4587	4591	2292.5	2301	2420	2299	2302.5
Average current [mA]	3.86	3.86	3.38	3.32	3.29	7.42	7.39	7.34	7.25	7.18
Energy [mJ]	32.27	32.27	54.15	50.26	49.84	56.13	56.11	58.62	55.00	54.56

Figure 11. **Literal pool with DMA simultaneously reading data from the flash memory**



This example finally demonstrates the advantage of the pre-read setting. It can greatly improve the efficiency when more than one stream of data is read from the flash memory and there is no latency. The prefetch is not useful when dealing mostly with data, that is no surprise. Again it is a good idea to keep the buffer enabled. The only reason to disable the buffer is if the timing needs to be more deterministic, whatever the efficiency cost may be.

8 Power consumption and performance comparison using STM32L4 series devices

The STM32L4 series devices are based on the Arm® Cortex®-M4 core connected to the 32-bit multilayer AHB bus matrix that connects up to six master and eight slave devices supporting concurrent operations as long as the bus masters are accessing different bus slaves.

The tests have been executed on a NUCLEO-L476RG board using all the available memory interface settings, listed in Table 7. The results of execution with a concurrent DMA transfer are not included for the STM32L4 series. The impact of the DMA on timing is minimal and the added current consumption is approximately the same regardless of the flash memory interface configuration, making the results not interesting.

One set of tests has been executed only with V_{CORE} range1 to provide a comparison with other series featured in this overview and to assess the impact of the prefetch and caches.

Other set of measurements has been executed using different latency, frequency, and voltage regulator settings to assess the energy needed for different operations in case of a battery powered application.

All the measurements are taken on a single sample of NUCLEO-L476RG board at ambient temperature. The values provided are an arithmetic mean from several measurements.

8.1 Influence of prefetch and cache with zero flash memory latency

One fact must be clarified before more measurement results presentation. Neither the prefetch or caches have any influence on the execution speed when the flash memory is available with zero latency. But the impact on the power consumption may be significant.

The prefetch actively tries to read the following instruction from the flash memory and the energy used to read the instruction may be wasted in case of branch. In case of a correct instruction prefetch there is no timing advantage, as the instruction is also ready within one clock cycle from the flash memory. It is recommended to disable the prefetch when the latency is zero. The measured input current difference is 10% in case of dhrystone.

On the contrary the caches tend to conserve the energy when they are activated. Both the instruction and data cache are likely to replace an access to the flash memory with an access to the cache, which needs significantly less current. The test have proven that enabling the caches lowers the power consumption by 20%.

With both contributors combined, the STM32L476G in a worst configuration of the flash memory interface, runs at significantly higher current consumption than that using optimal settings (both at 16 MHz, latency 0, V_{CORE} range 1).

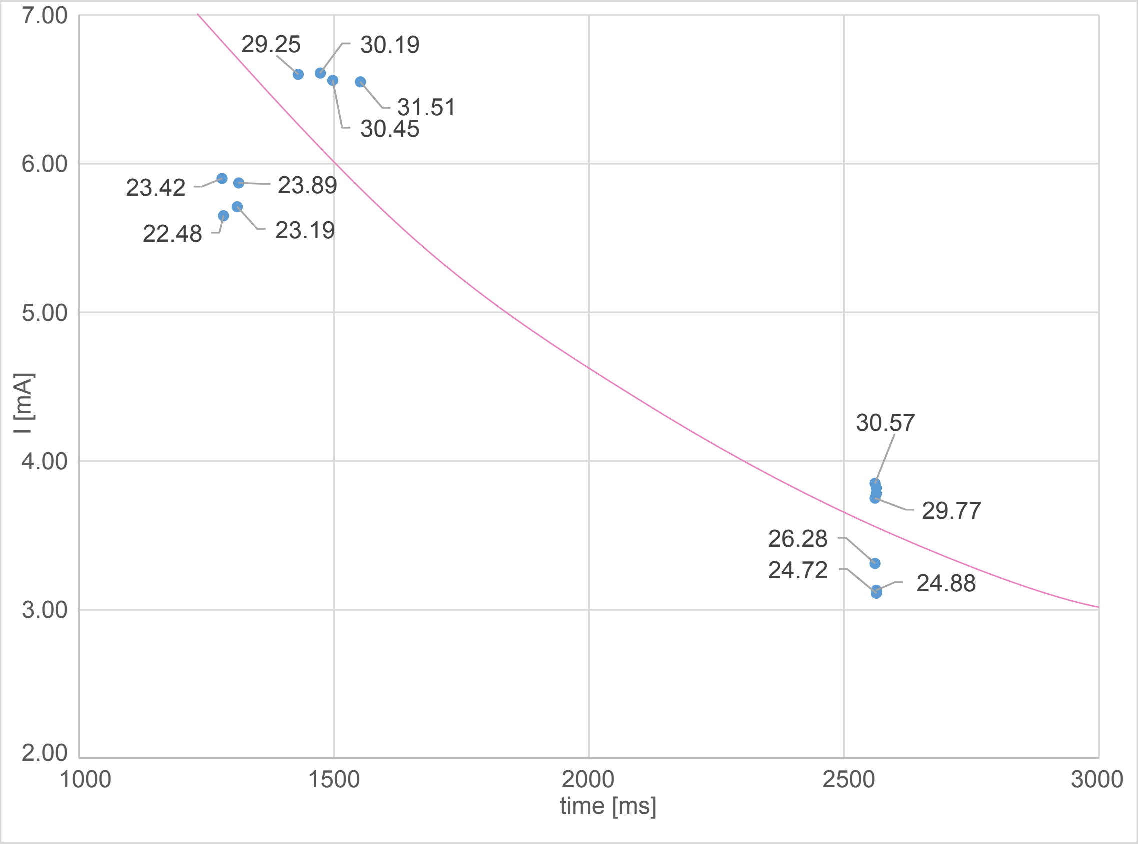
8.2 STM32L4 Dhrystone benchmark

The Dhrystone code is executed only on optimal settings with a zero latency, where the timing is still the same. The task consists of processing 50000 cycles of the test loop, using the HSI or the HSI sourced PLL as the clock source.

Table 19. **Dhrystone test using core voltage range 1 and HSI clock**

Frequency	16 MHz	32 MHz							
Latency	0	1							
D-cache	1	0	0	0	1	1	1	1	0
I-cache	1	0	0	1	1	1	0	0	1
Prefetch	0	0	1	1	1	0	0	1	0
Time [ms]	2561	1552	1473	1313	1281	1283	1498	1430	1310
Average current [mA]	3.12	6.55	6.61	5.87	5.9	5.65	6.56	6.6	5.71
Energy [mJ]	24.80	31.51	30.19	23.89	23.42	22.48	30.45	29.25	23.19

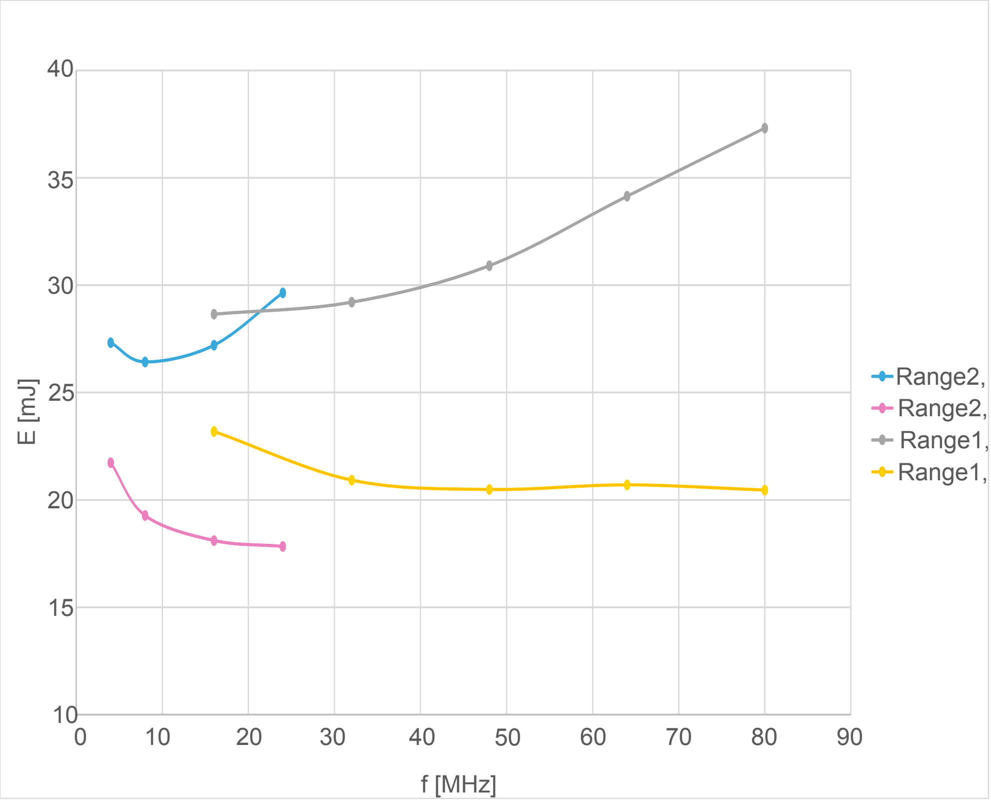
Figure 12. **Dhrystone test plot of energy needed for execution**



This example clearly demonstrates that while the prefetch can lead to an improved performance, especially if the instruction cache is enabled, it does not bring a significant additional advantage in case of the Dhrystone test code. The prefetch complements the caches and helps in the code sections with minimum loops, where the caches cannot help.

The optimal configuration of the flash interface being identified, how the cache behaves using different core clock speeds. A higher clock speed leads to a higher latency, forcing the core to wait for a read access to the fash memory if the instruction and data are not available in the ART cache. The core waiting for the memory still needs energy, reducing the overall efficiency.

Figure 13. Energy cost of the dhrystone test loop



In Figure 13, the same test loop of 50000 Dhrystone tests is executed with different clock settings using either the MSI, or in case of a 64-MHz and a 80-MHz PLL, a module with the MSI as the source clock. The additional power consumption of the PLL causes a slight drop in the efficiency visible on the chart.

Otherwise the chart shows us that at least in case of a Dhrystone test, which includes lot of loops, the ART accelerator cache is able of improving the MCU execution efficiency by increasing the core clock. This is a remarkable feature.

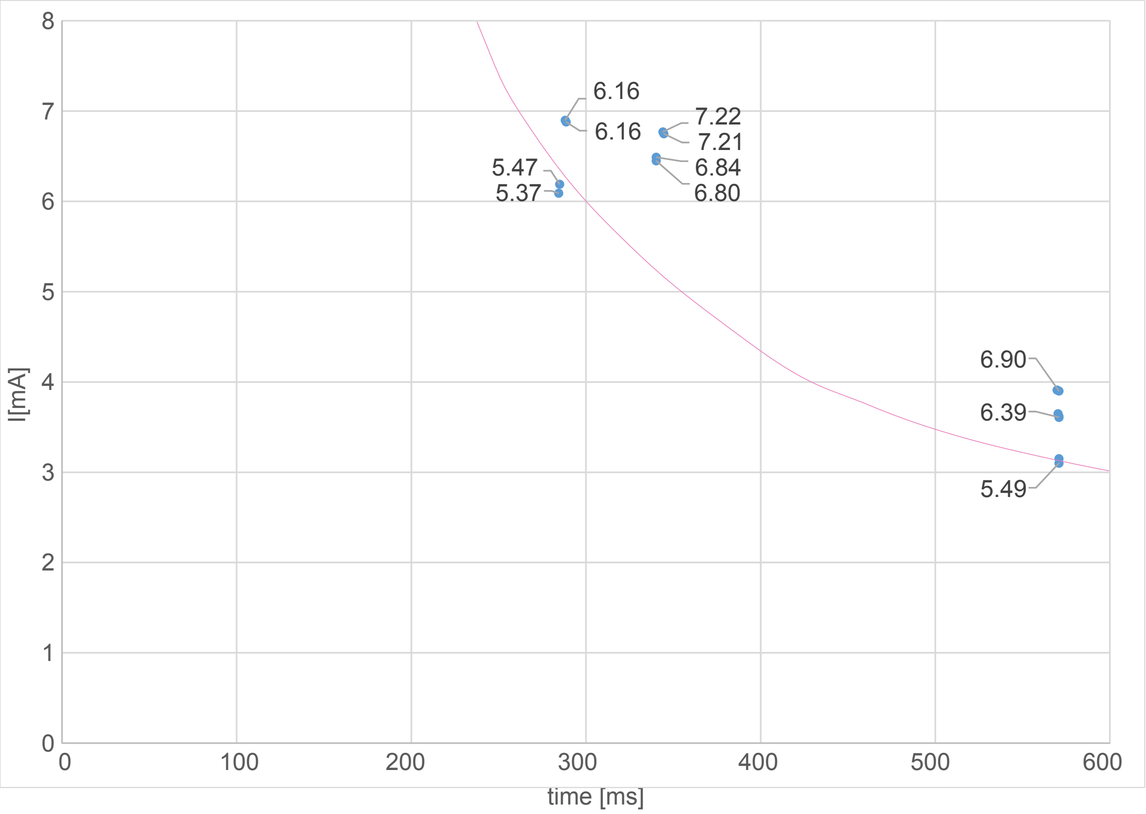
8.3 STM32L4 memory read stress test

A stress test consists of executing 20 LDR instructions fetching data from program NV memory to CPU core registers in a loop of 100000 cycles. This test demonstrates mainly the power of the data cache in such situations.

Table 20. Literal measurements

Frequency	16-MHz	32-MHz							
Latency	0	1							
D-cache	1	0	0	0	1	1	1	1	0
I-cache	1	0	0	1	1	1	0	0	1
Prefetch	0	0	1	1	1	0	0	1	0
Time [ms]	570	344.5	344	340.2	284.9	284.3	288.1	288.7	340.2
Average current [mA]	3.10	6.75	6.77	6.49	6.19	6.09	6.9	6.88	6.45
Energy [mJ]	5.49	7.21	7.22	6.84	5.47	5.37	6.16	6.16	6.80

Figure 14. Literal pool chart plot of energy efficiency



In case of data literal pool loop the data cache tends to improve significantly the execution speed, while the instruction cache tends to rather contribute to the power consumption. What is not visible from the plot is that the efficiency improvement tends to grow slowly with several hundred iterations before reaching a maximum.

9 Power consumption and performance measurements on STM32G0 series device

STM32G0 shares some power saving features with the low power series. STM32G0B1RE, the device used in the measurement, has 512 Kbytes of dual-bank flash memory.

Documents [6] and [7] describe a bug that compromises the prefetch advantage of this device. When the boundary between the two banks is crossed, the prefetch may fail to present the intended instruction, resulting in a possible hard fault. There is no workaround, so disabling prefetch is recommended.

Architecturally, STM32G0 has the same Arm® Cortex®-M0+ CPU core as the STM32L0 series, but with a nonvolatile memory arrangement more similar to the STM32L4 series, with a smaller cache.

The measurements presented in this document are performed on the NUCLEO-G0B1RE board without modifications.

9.1 STM32G0 Dhrystone benchmark

All measurements are made using the 8-MHz HSE supplied by the STLINK. This arrangement is slightly less efficient than HSI at 16 MHz because it needs a PLL to achieve this clock speed. At higher clock rates, where a PLL is needed anyway, the consumption is slightly lower because the oscillator is externalized. In this case, it is a fair comparison of efficiency between different clock speed configurations.

The benefit of both cache and prefetch depends on the code being executed. More loops emphasize the benefit of cache, while branching negates the benefit of prefetch. In general, accessing flash memory consumes more power than accessing RAM. If the cache hits, some energy is saved. If the prefetch fails, energy is wasted.

The following table shows how flash memory interface settings affect both performance and efficiency.

Table 21. **Flash memory interface settings**

Frequency	16 MHz	32 MHz				64 MHz			
Latency	0	1	1	1	1	2	2	2	2
Cache	0	1	1	0	0	1	1	0	0
Prefetch	0	0	1	1	0	0	1	1	0
Time [s]	2.06	1.17	1.09	1.19	1.3	0.66	0.595	0.693	0.789
Average current [mA]	2.56	4.21	4.57	4.84	4.39	7.72	8.4	8.56	7.7
Energy [mJ]	17.67	16.89	17.05	19.73	19.57	17.38	17.08	20.23	20.56

The benchmark shows the advantage of both cache and prefetch. As latency increases, they keep the CPU busy and efficient. But while the cache hits save energy, the prefetch costs energy even if the instruction is not used.

In some cases, such as the Dhrystone running with 1 wait state, prefetching improves performance but decreases the overall power efficiency.

Other methods of assessing performance have been used, with results that differ in absolute terms or even in the order of configurations in terms of efficiency. However, the overall trend is broadly the same, suggesting that both prefetch and cache benefits increase as clock speed (and latency) increases, with cache improving more on the efficiency side, and prefetch providing the greatest benefit at peak performance.

10 General observations on power consumption optimization

The general rule to minimize the power consumption is to perform the task for the shortest possible time, at the lowest possible operating frequency and with the clock enabled to a minimal part of the silicon.

In other words, the goal is to optimize for execution speed and then find an optimal balance between the time and the clock frequency. The speed optimization is mostly a matter of compiler choice. If the user has the opportunity, he must build the reference projects with different development tools and observe the difference in power consumption and execution speed.

Even the best compiler can benefit from some tricks applicable in most C source codes:

1. Where possible, use variables of size that correspond to the CPU register size (32 bits).
2. Use macros instead of simple functions to save on function call overhead.
3. Learn to use keywords like static, restrict, register, inline.
4. Most compilers can be guided using various “#pragma” statements for more optimized results. Check what pragmas are available in your development toolchain.

The memory placement influences also the power consumption. Some microcontrollers embed more than one type of volatile memory. Some may need little more energy than others.

11 Conclusion

Each low-power STM32 microcontroller series requires a slightly different approach to optimize the energy efficiency.

Putting the product in low-power mode during the idle period is best practice, but the wake up time must always be considered. The peripherals left active in low-power mode to trigger the wake up have an impact on the power consumption. This is detailed in the datasheet and can be checked using the firmware examples.

Another set of optimization challenges is presented in relation to the Run mode and the code execution.

The measured results provide the guidance for decision whether or not to enable the different memory interface settings. The features like the prefetch, improving the benchmark result, also lead to a higher power consumption and the overall efficiency is dependent on the task processed by the microcontroller.

There is no significant benefit in tweaking the settings when the flash memory latency is not in place. This makes sense only if the flash memory contains frequently used literal pools (predefined data constants) or if the cache access leads to lower energy consumption.

With the flash memory latency in place, the flash interface must be set up carefully, as the performance difference between the optimal and default configuration may be significant. It is definitely possible to activate some flash interface settings only temporarily for particular operations and disable them afterwards.

It is demonstrated that the erratum present on the dual-bank STM32G0 devices does impact the top performance, but less so the efficiency.

Revision history

Table 22. **Document revision history**

Date	Revision	Changes
19-Jan-2016	1	Initial release.
24-Oct-2016	2	Updated cover adding STM32L4 Series. Updated Section 2: System architecture adding STM32L4 memory interface description. Added Section 4.4: STM32L4 series device options. Added Power consumption and performance comparison using STM32L4 Series devices. Updated Section 11: Conclusion.
21-Aug-2019	3	Added Section 1: General information Added Section 3: Low-power modes. Added Section 5: Reproducing the measurements to get datasheet values. Updated Section 10: General observations on power consumption optimization.
28-Jun-2023	4	Added new sections: • Section 4.1: STM32G0 series device options • Section 9: Power consumption and performance measurements on STM32G0 series device
22-Jan-2024	5	Updated: • Section Introduction • Section Referenced documents • Section 1: General information • Section 3: Low-power modes • Section 4.1: STM32G0 series device options • Section 4.2: STM32L1 series device options • Section 4.3: STM32L0 series device options • Section 4.4: STM32L4 series device options • Section 4.5: Execution from a volatile memory • Section 5.1: Hardware and prerequisites • Section 9: Power consumption and performance measurements on STM32G0 series device General document cleanup.

Contents

List of tables

List of figures

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2024 STMicroelectronics – All rights reserved