LOW POWER MODES

Estimación de la duración de una batería para un sistema basado en microcontrolador.

El caso general, algunos ejemplos y el caso particular del ATmega328.

Recordar que

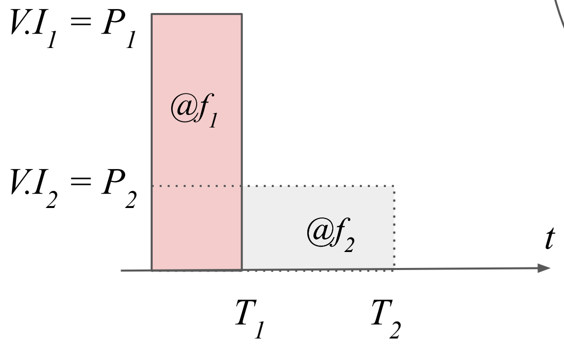
$$P = C\,V^2 f \qquad \text{y} \qquad T = N\,CPI\,/\,f$$

Entonces, la energía necesaria para realizar una determinada tarea (N instrucciones) no depende de la frecuencia de operación del procesador:

$$E = P\,.\,T = C\,V^2\,N\,CPI$$

Por lo tanto **no se ahorra energía disminuyendo la frecuencia**.

En nuestro caso podríamos reducir la tensión de alimentación (*V*), pero hay un límite porque aparecen problemas de ruido (que depende de la tecnología de implementación y de la **frecuencia de operación**). Entonces se ahorra?



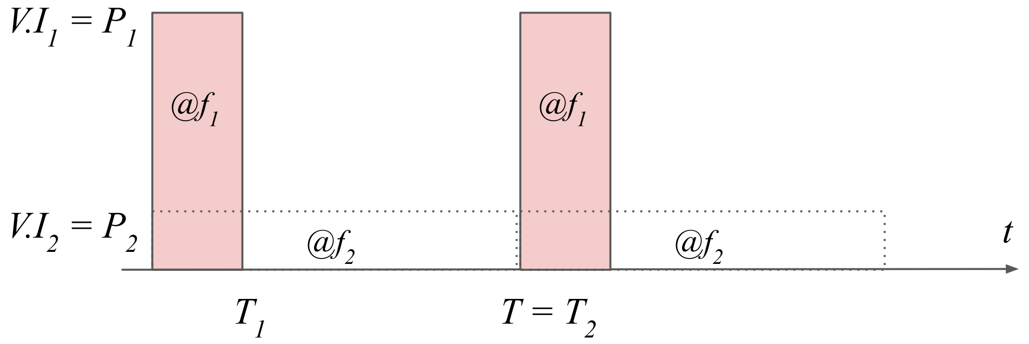
De las páginas 1 y 2 del manual de ATmega328P (el modelo P es low-power, automotive grade):

Operating voltage: 2.7V to 5.5V
Temperature range: Automotive temperature range: –40°C to +125°C
Speed grade:
- 0 to 8MHz at 2.7 to 5.5V
- 0 to 16MHz at 4.5 to 5.5V

Low power consumption:
- Active mode: 1.5mA at 3V - 4MHz
- Power-down mode: 1µA at 3V

Atmel AVR®

Podría reducirse la frecuencia de la CPU hasta que la CPU esté trabajando el 100% del tiempo, pero es difícil de ajustar, la frecuencia inferior tiene un límite, etc.



Lo más razonable es hacer la tarea y apagar todo lo más posible: **low power modes** (stand-by, power-down, off-mode) CPU y módulos ADC, timers, etc.

¿Cómo se entra en el modo de bajo consumo?

¿Cómo se sale? ¿Cuánto tarda en salir?

¿Cuánto consume en bajo consumo? ¿Cómo varía con f y con V?

¿Se pueden apagar los periféricos que no utilizo?

Toda esa información está en la hoja de datos. Lo que no está ahí es cuánto tiempo voy a estar computando (T_1).

Benchmarks

- **DMIPS** = iteraciones Dhrystone/1757 (Dhrystone de la VAX-11/780, primera máquina de 1 MIPS)
- **DMIPS/MHz** Refleja la organización de la máquina
- **(DMIPS/MHz) x (MHz/mWatt)** = **DMIPS/mWatt**

Se puede medir con un benchmark

Clase anterior, AVR:
0.33 DMIPS/MHz

x

Se puede medir con un tester (hoja de datos)

Manual ATmega328P:
Active mode: 1.5mA at 3V - 4MHz
Entonces 4MHz / 4.5mW = **0.88 MHz/mW**

=

Figura de eficiencia de cómputo

0.33 DMIPS/MHz x 0.88 MHz/mW =
0.3 DMIPS/mW

Benchmarks

- **DMIPS** = iteraciones Dhrystone/1757 (Dhrystone de la VAX-11/780, primera máquina de 1 MIPS)
- **DMIPS/MHz** Refleja la organización de la máquina
- **(DMIPS/MHz) x (MHz/mWatt)** = **DMIPS/mWatt**

FALTA CONSIDERAR la eficiencia cuando no está computando: modos de bajo consumo

Se puede medir con un benchmark

Clase anterior, AVR:
0.33 DMIPS/MHz

x

Se puede medir con un tester (hoja de datos)

Manual ATmega328P:
Active mode: 1.5mA at 3V - 4MHz
Entonces 4MHz / 4.5mW = **0.88 MHz/mW**

=

Figura de eficiencia de cómputo

0.33 DMIPS/MHz x 0.88 MHz/mW =
0.3 DMIPS/mW

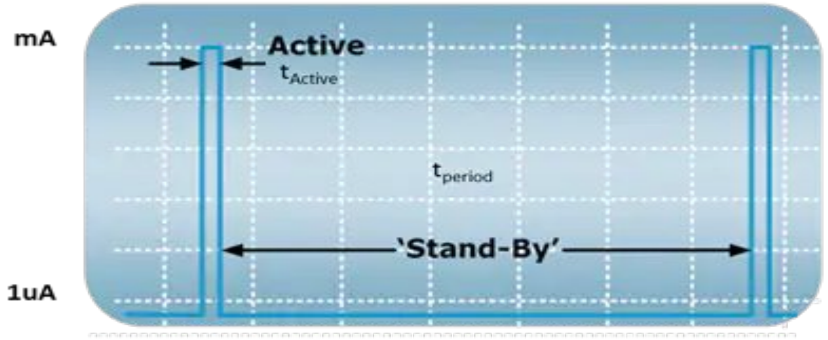
https://www.eembc.org/coremark/scores.php
(no está STM8)

CoreMark®
An EEMBC Benchmark

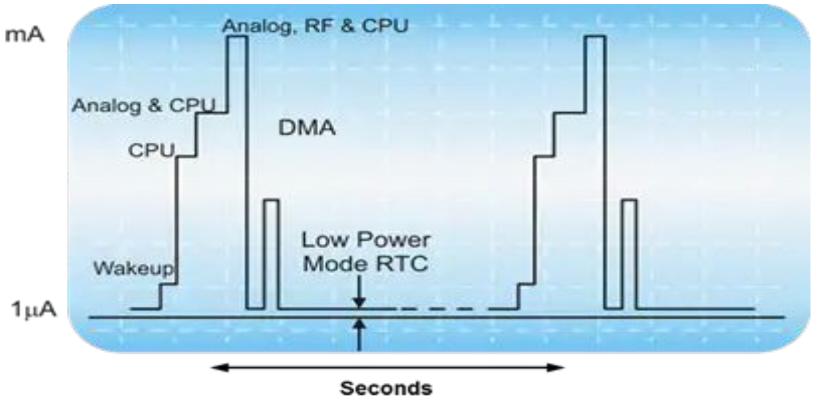
	Compiler	Execution Memory	MHz	CoreMark	CoreMark / MHz↑	DMIPS/MHz
Atmel AT89C51RE2 in X2 mode (6 clocks/machine cycle)	Keil C51 v8.18	Internal RAM & flash (static)	22	2.36	**0.11**	
Microchip PIC18F46K22	Microchip MPLAB XC8 v1.32	Code in Flash, Data in RAM	64	7.23	**0.11**	
Atmel ATmega644	avr-gcc-4.3.2	Flash & SRAM 20 MHz (Static)	20	10.81	**0.54**	**0.33**
Texas Instruments MSP430F5438	Code Composer Studio 4.1.2	Internal flash and RAM (static)	18	11.10	**0.62**	**0.31**
Microchip PIC24FJ64GA004	gcc 4.0.3	Code Flash. Data SRAM (Static)	32	23.87	**0.75**	**0.50**
STMicroelectronics STM32F103	GCC 4.4.1	Internal Flash	72	108.26	**3.32**	**1.25**

Low power modes

https://ar.mouser.com/applications/low-power-ewc-design/
https://microchipdeveloper.com/8avr:low-power-example

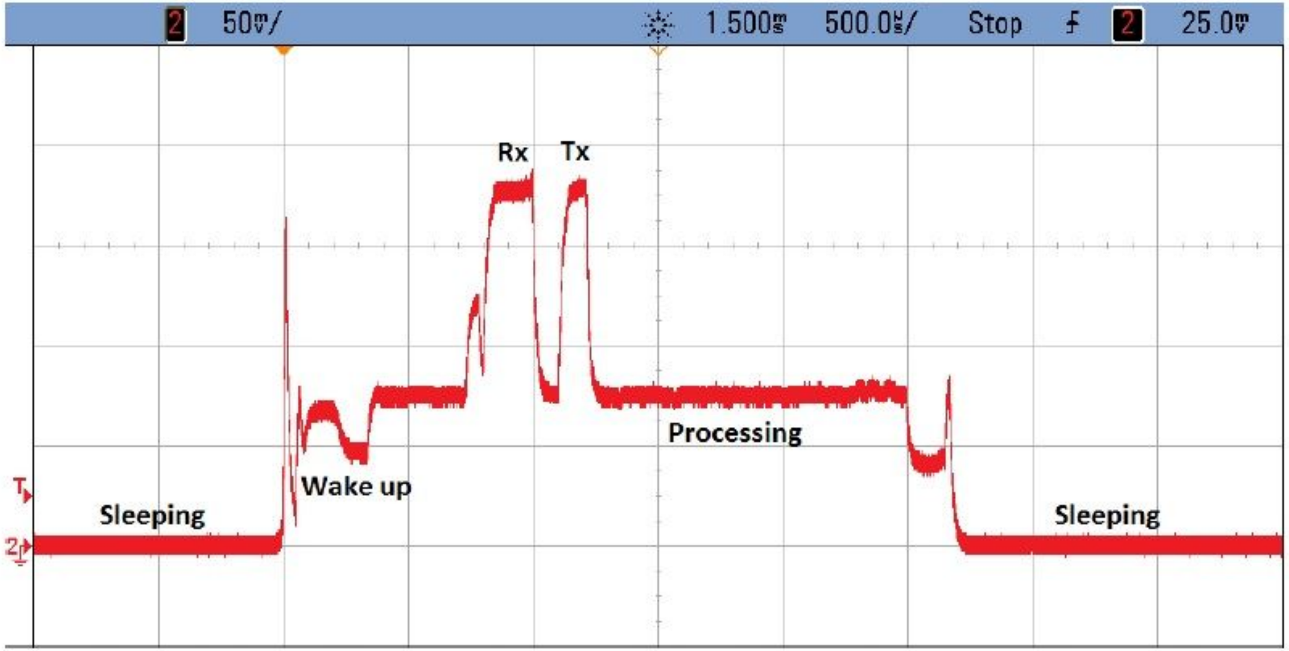


$$I_{AVG} = I_{Stand\text{-}By} + I_{Active} * t_{Active} / t_{period}$$



Power Profile: *puede ser difícil de calcular, si es posible es mejor medirlo, los kits suelen tener la posibilidad (jumper serie)*

Power profile



Perfil de potencia de un posible sensor BLE

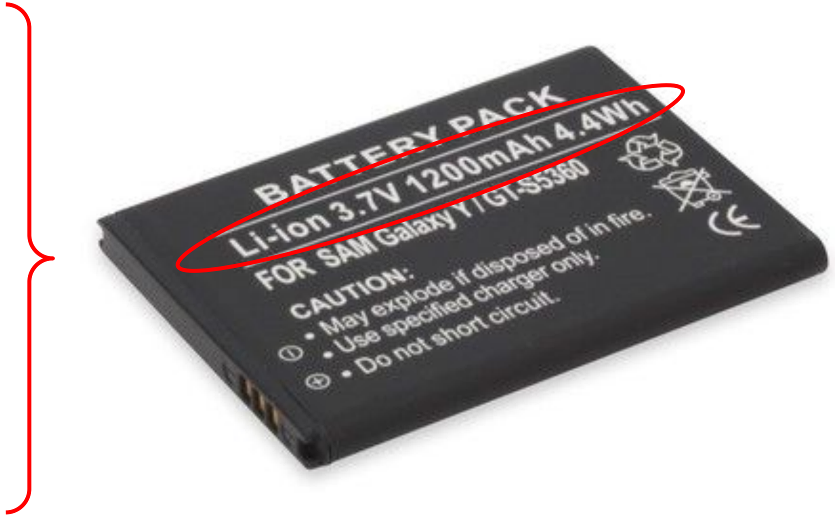
Capacidad de una batería

Carga [mAh]
(Sistema Internacional: Coulomb)
[A] . [s] = [C] … pero suele utilizarse [mAh]

Q = 1200 mAh * 3600 s/h * 0.001 A/mA = 4320 C

Energía [Wh]
(Sistema Internacional: Joule)
[Joule] = [W] . [s] … pero suele utilizarse [Wh]

E = P . t = 1200 mAh * 3.7 V = 4440 mWh = 4.44 Wh
E = 4.44 Wh * 3600 s/h = 16000 J


BATTERY PACK
Li-ion 3.7V 1200mAh 4.4Wh
FOR SAM Galaxy Y / GT-S5360
CAUTION:
• May explode if disposed of in fire.
• Use specified charger only.
• Do not short circuit.
CE

El voltaje está determinado por el **proceso químico**, por ejemplo Li-ion 3.7V, alcalinas 1.5V, mercurio 1.35V, etc.
La capacidad máxima está determinada por el **volumen de electrolito** (tamaño).
Es importante verificar la corriente máxima de carga y descarga (calentamiento debido a la resistencia interna). Ver advertencias.

Si se va a convertir a otro voltaje debe considerarse la **eficiencia del convertidor**.
Por ejemplo, si convierto a 5V con eficiencia 90%, obtengo el equivalente a 4.44Wh / 5V * 0.9 = 800 mAh (energía constante).

Ejemplo muy simplificado CPU08 vs. AVR

Cada 1 minuto el microcontrolador debe realizar un cálculo de aproximadamente un millón de operaciones y luego pasar a stand-by.

a) ¿Cuánto durará una batería de 2400 mAh si se utiliza la siguiente CPU?

0.125 DMIPS/MHz

I_{act} = 1 mA/MHz at 3V

$I_{stand\text{-}by}$ = 16 uA at 3V

f_{max} = 8 MHz

Cada 1 minuto el microcontrolador debe realizar un cálculo de aproximadamente un millón de operaciones y luego pasar a stand-by.

a) ¿Cuánto durará una batería de 2400 mAh si se utiliza la siguiente CPU?

0.125 DMIPS/MHz

I_{act} = 1 mA/MHz at 3V

$I_{stand\text{-}by}$ = 16 uA at 3V

f_{max} = 8 MHz

*@8MHz 8 MHz * 0.125 DMIPS/MHz = 1 DMIPS* $T_{act} = 1\ s$ $I_{act} = 8\ mA$

*@4MHz 4 MHz * 0.125 DMIPS/MHz = 0.5 DMIPS* $T_{act} = 2\ s$ $I_{act} = 4\ mA$

$I_{media} = (1\ s * 8\ mA + 60\ s * 0.016\ mA\ ) / 60 = 0.15\ mA$

Por lo tanto la duración de la batería estará dada por : 2400 mAh / 0.15 mA = 16000 horas = **1.8 años**

Cada 1 minuto el microcontrolador debe realizar un cálculo de aproximadamente un millón de operaciones y luego pasar a stand-by.

a) ¿Cuánto durará una batería de 2400 mAh si se utiliza la siguiente CPU?

0.125 DMIPS/MHz
I_{act} = 1 mA/MHz at 3V
$I_{stand\text{-}by}$ = 16 uA at 3V
f_{max} = 8 MHz

*@8MHz 8 MHz * 0.125 DMIPS/MHz = 1 DMIPS* $T_{act} = 1\ s$ $I_{act} = 8\ mA$
*@4MHz 4 MHz * 0.125 DMIPS/MHz = 0.5 DMIPS* $T_{act} = 2\ s$ $I_{act} = 4\ mA$
$I_{media} = (1\ s * 8\ mA + 60\ s * 0.016\ mA) / 60 = 0.15\ mA$
Por lo tanto la duración de la batería estará dada por : 2400 mAh / 0.15 mA = 16000 horas = **1.8 años**

b) ¿Cuánto durará la misma batería si se utiliza un ATmega328P?
- 0.33 DMIPS/MHz (AVR, clase anterior)
- Active mode: 1.5mA at 3V - 4MHz
- Power-down mode: 1µA at 3V

Cada 1 minuto el microcontrolador debe realizar un cálculo de aproximadamente un millón de operaciones y luego pasar a stand-by.

a) ¿Cuánto durará una batería de 2400 mAh si se utiliza la siguiente CPU?

0.125 DMIPS/MHz
I_{act} = 1 mA/MHz at 3V
$I_{stand\text{-}by}$ = 16 uA at 3V
f_{max} = 8 MHz

*@8MHz 8 MHz * 0.125 DMIPS/MHz = 1 DMIPS* $T_{act} = 1\ s$ $I_{act} = 8\ mA$
*@4MHz 4 MHz * 0.125 DMIPS/MHz = 0.5 DMIPS* $T_{act} = 2\ s$ $I_{act} = 4\ mA$
$I_{media} = (1\ s * 8\ mA + 60\ s * 0.016\ mA\ ) / 60 = 0.15\ mA$
Por lo tanto la duración de la batería estará dada por : 2400 mAh / 0.15 mA = 16000 horas = **1.8 años**

b) ¿Cuánto durará la misma batería si se utiliza un ATmega328P con AVR?
- 0.33 DMIPS/MHz (AVR)
- Active mode: 1.5mA at 3V - 4MHz
- Power-down mode: 1μA at 3V

*@4MHz 4 MHz * 0.33 DMIPS/MHz = 1.32 DMIPS* $T_{act} = 0.75\ s$ $I_{act} = 1.5\ mA$
$I_{media} = (0.75\ s * 1.5\ mA + 60\ s * 0.001\ mA\ ) / 60 = 0.02\ mA$
Duración de la batería: 2400 mAh / 0.02 mA = **14 años**

Cada 1 minuto el microcontrolador debe realizar un cálculo de aproximadamente un millón de operaciones y luego pasar a stand-by.

a) ¿Cuánto durará una batería de 2400 mAh si se utiliza la siguiente CPU?

0.125 DMIPS/MHz
I_{act} = 1 mA/MHz at 3V
$I_{stand\text{-}by}$ = 16 uA at 3V
f_{max} = 8 MHz

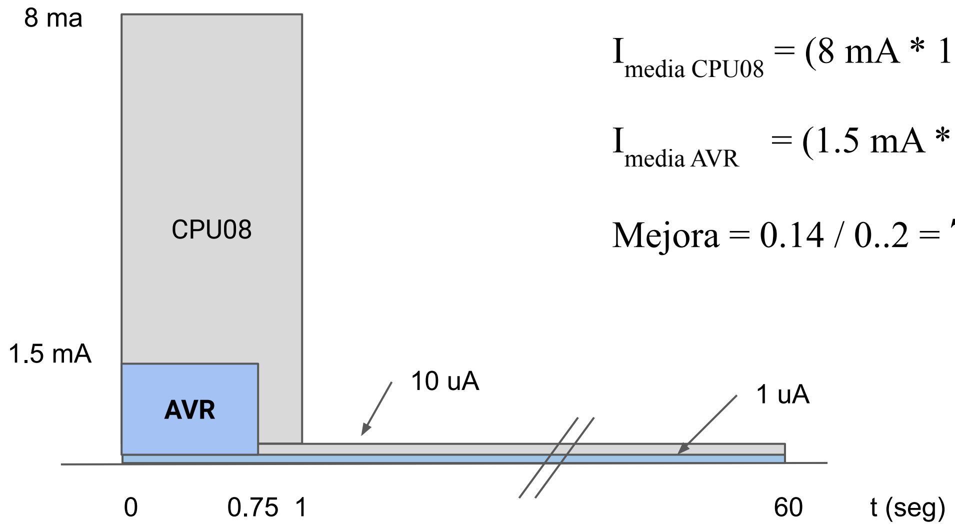
@8MHz $8\ MHz * 0.125\ DMIPS/MHz = 1\ DMIPS$ $T_{act} = 1\ s$ $I_{act} = 8\ mA$
@4MHz $4\ MHz * 0.125\ DMIPS/MHz = 0.5\ DMIPS$ $T_{act} = 2\ s$ $I_{act} = 4\ mA$
$I_{media} = (1\ s * 8\ mA + 60\ s * 0.016\ mA\ ) / 60 = 0.15\ mA$
Por lo tanto la duración de la batería estará dada por : $2400\ mAh / 0.15\ mA = 16000$ *horas* = **1.8 años**

b) ¿Cuánto durará la misma batería si se utiliza un ATmega328P con AVR?
- 0.33 DMIPS/MHz (AVR)
- Active mode: 1.5mA at 3V - 4MHz
- Power-down mode: 1μA at 3V

@4MHz $4\ MHz * 0.33\ DMIPS/MHz = 1.32\ DMIPS$ $T_{act} = 0.75\ s$ $I_{act} = 1.5\ mA$
$I_{media} = (0.75\ s * 1.5\ mA + 60\ s * 0.001\ mA\ ) / 60 = 0.02\ mA$
Duración de la batería: $2400\ mAh / 0.02\ mA$ = **14 años**

c) ¿Cuánto duraría la batería si se mantuviera siempre activo? **d)** ¿Y si se mantuviera siempre en stand-by?

c) I_{media} = *4 (08) ó 1.5 mA (AVR)* *Duración de la batería:* **24 días (08) 66 días (AVR)**
d) I_{media} = *0.016 (08) ó 0.001mA (AVR)* *Duración de la batería:* **17 años (08) 274 años (AVR)** **> RETENCIÓN BATERÍA**



T_{CPU08} = 1/ (0.125 DMIPS/MHz * 8 MHz) = 1 s

T_{AVR} = 1/ (0.33 DMIPS/MHz * 4 MHz) = 0.75 s

$I_{media\ CPU08}$ = (8 mA * 1 s + 10 uA * 60 s) / 60 s = 0.14 mA

$I_{media\ AVR}$ = (1.5 mA * 0.75 s + 1 uA * 60 s) / 60 s = 0.02 mA

Mejora = 0.14 / 0..2 = **7x**

Ejemplo con CoreMark

https://www.eembc.org/coremark/scores.php

Atmel ATmega2560 (AVR 8-bits)
CoreMark @8Mhz: 4.25 - CoreMark/MHz: 0.53
Datasheet: Ultra-Low Power Consumption
– Active Mode: 1MHz, 1.8V: 500µA
– Power-down Mode: 0.1µA at 1.8V

Otro miembro de la familia AVR, con modo ultra low power

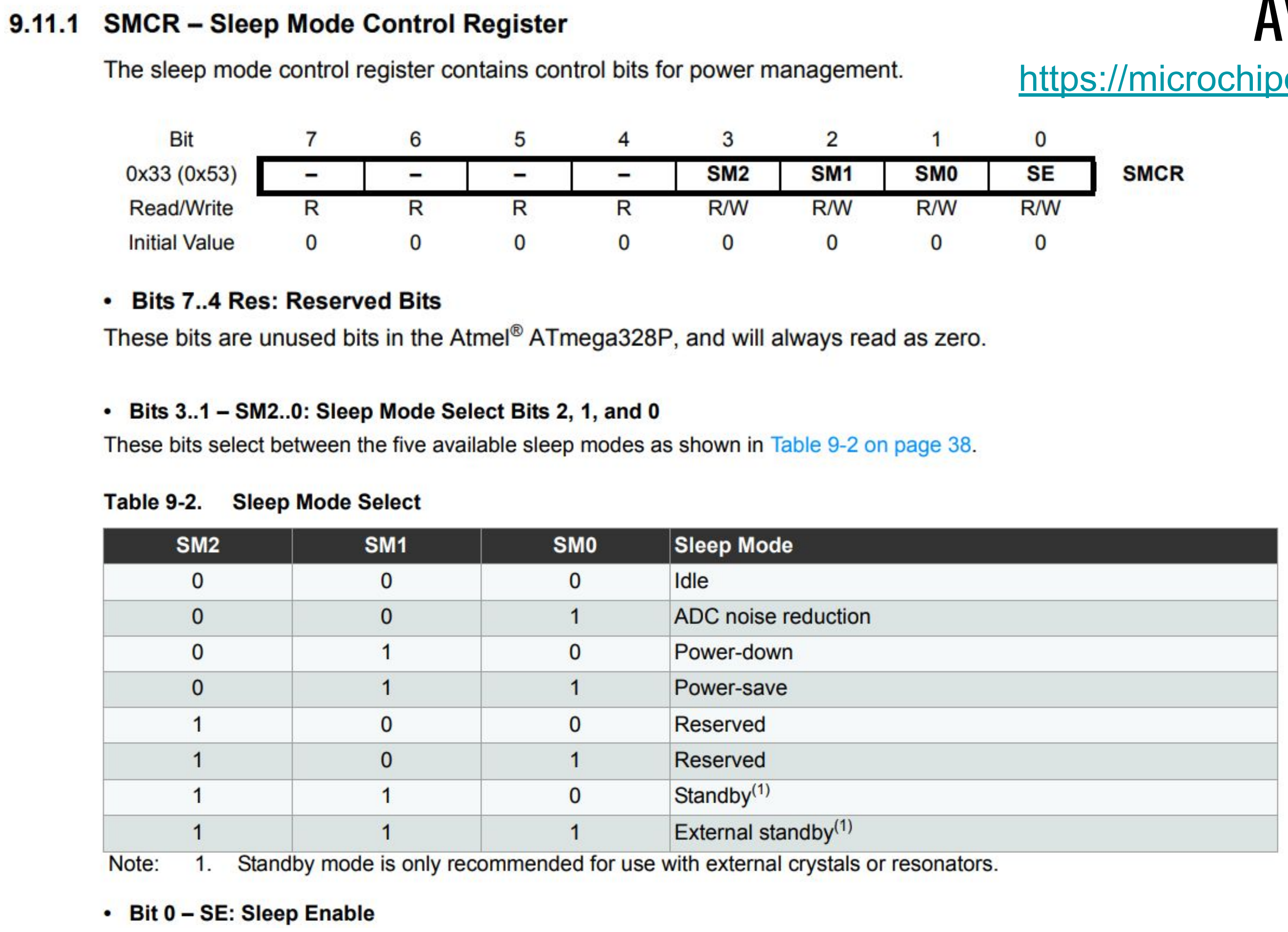
Texas Instruments MSP430FG4618 (MSP430 16-bits)
CoreMark @8Mhz: 5.90 - CoreMark/MHz: 0.74
Datasheet: Ultra-low power consumption
– Active mode: 400 µA at 1 MHz, 2.2 V
– Standby mode: 1.3 µA
– Off mode (RAM retention): 0.22 µA

Mejora de performance: 0.74/0.53 = 1.4x
Leve mejora de potencia en modo activo = (1.8x500)/(2.2x400) = 1.02x
Para una tarea fija la mejora es 1.4x1.02=1.45x

PERO OJO
stand-by, power-down, off-mode?? Puede cambiar la cuenta, dependiendo de los modos y de la proporción de tiempo que pasa en bajo consumo.

AVR low power modes

https://microchipdeveloper.com/8avr:avrsleep



9.11.1 SMCR – Sleep Mode Control Register

The sleep mode control register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	–	–	–	–	SM2	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..4 Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega328P, and will always read as zero.

- **Bits 3..1 – SM2..0: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the five available sleep modes as shown in Table 9-2 on page 38.

Table 9-2. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC noise reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby[1]
1	1	1	External standby[1]

Note: 1. Standby mode is only recommended for use with external crystals or resonators.

- **Bit 0 – SE: Sleep Enable**

Varios modos, ver diferencias

Además los módulos pueden apagarse en forma individual.

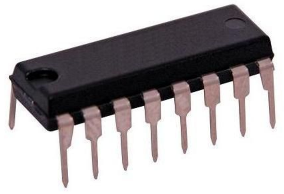
Para entrar en el modo seleccionado:
  SE=1
  instrucción SLEEP

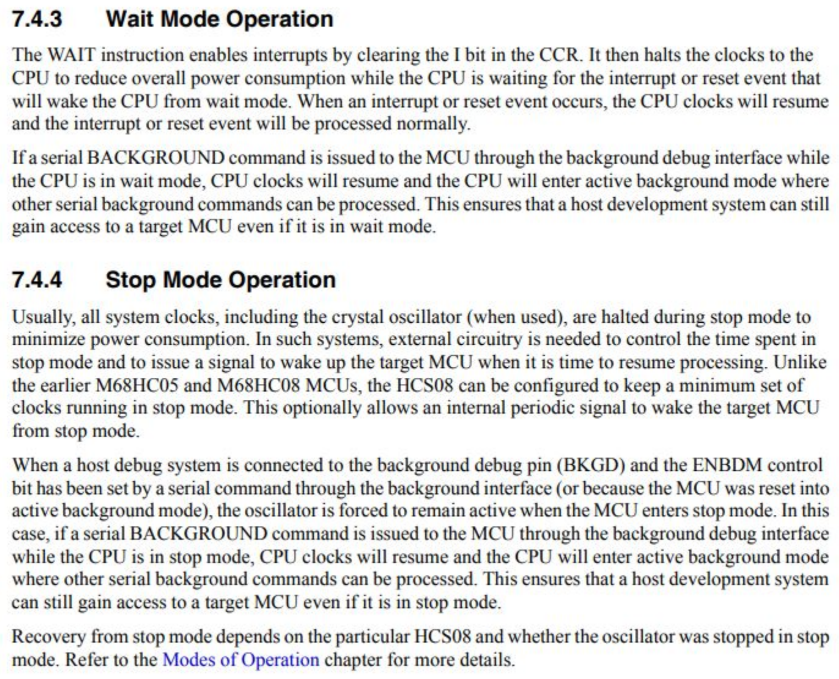
Para salir: interrupción habilitada.
Continua a partir de SLEEP.
O reset.

Ver wake-up time (ciclos)

CPU08 low power modes

The **WAIT** instruction stops only the CPU clock and therefore uses more power than the **STOP** instruction, which stops both the CPU clock and the peripheral clocks. In most modules, clocks can be shut off in wait mode





7.4.3 Wait Mode Operation

The WAIT instruction enables interrupts by clearing the I bit in the CCR. It then halts the clocks to the CPU to reduce overall power consumption while the CPU is waiting for the interrupt or reset event that will wake the CPU from wait mode. When an interrupt or reset event occurs, the CPU clocks will resume and the interrupt or reset event will be processed normally.

If a serial BACKGROUND command is issued to the MCU through the background debug interface while the CPU is in wait mode, CPU clocks will resume and the CPU will enter active background mode where other serial background commands can be processed. This ensures that a host development system can still gain access to a target MCU even if it is in wait mode.

7.4.4 Stop Mode Operation

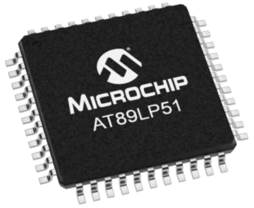
Usually, all system clocks, including the crystal oscillator (when used), are halted during stop mode to minimize power consumption. In such systems, external circuitry is needed to control the time spent in stop mode and to issue a signal to wake up the target MCU when it is time to resume processing. Unlike the earlier M68HC05 and M68HC08 MCUs, the HCS08 can be configured to keep a minimum set of clocks running in stop mode. This optionally allows an internal periodic signal to wake the target MCU from stop mode.

When a host debug system is connected to the background debug pin (BKGD) and the ENBDM control bit has been set by a serial command through the background interface (or because the MCU was reset into active background mode), the oscillator is forced to remain active when the MCU enters stop mode. In this case, if a serial BACKGROUND command is issued to the MCU through the background debug interface while the CPU is in stop mode, CPU clocks will resume and the CPU will enter active background mode where other serial background commands can be processed. This ensures that a host development system can still gain access to a target MCU even if it is in stop mode.

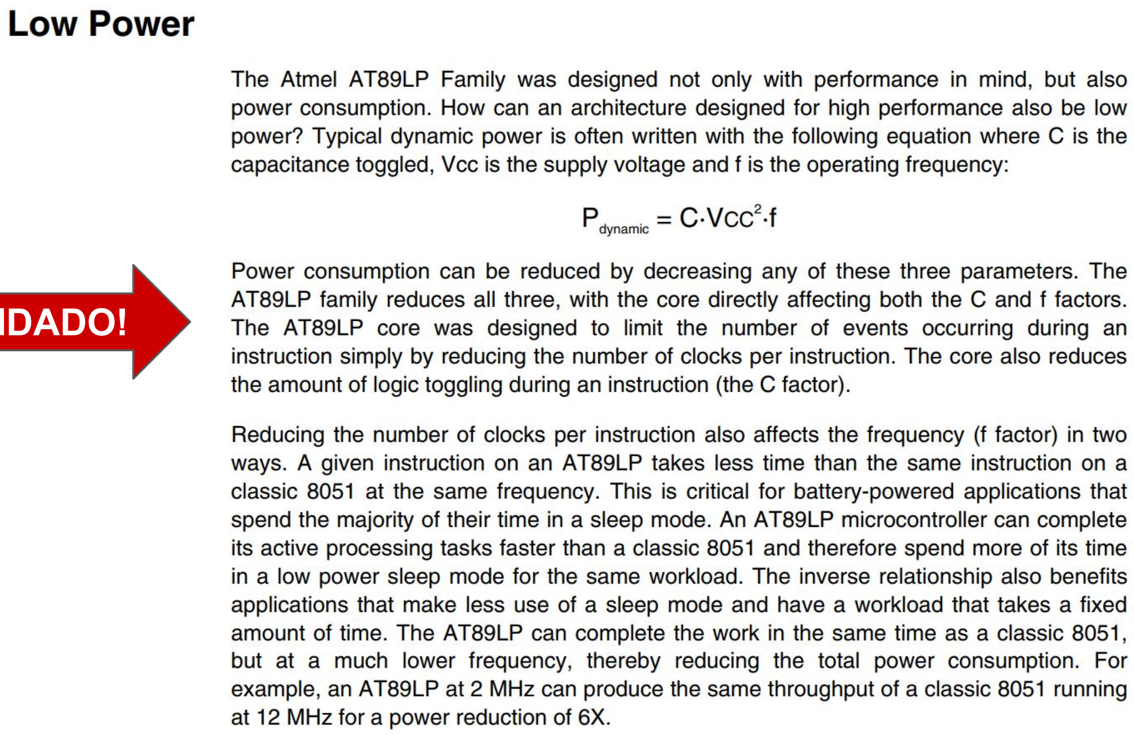
Recovery from stop mode depends on the particular HCS08 and whether the oscillator was stopped in stop mode. Refer to the Modes of Operation chapter for more details.

Intel 8051 - Atmel AT89LP

100% de compatibilidad binaria (ISA)
Poco queda de la organización inicial (ORG)
Implementado con tecnología actual (TEC)



CUIDADO!



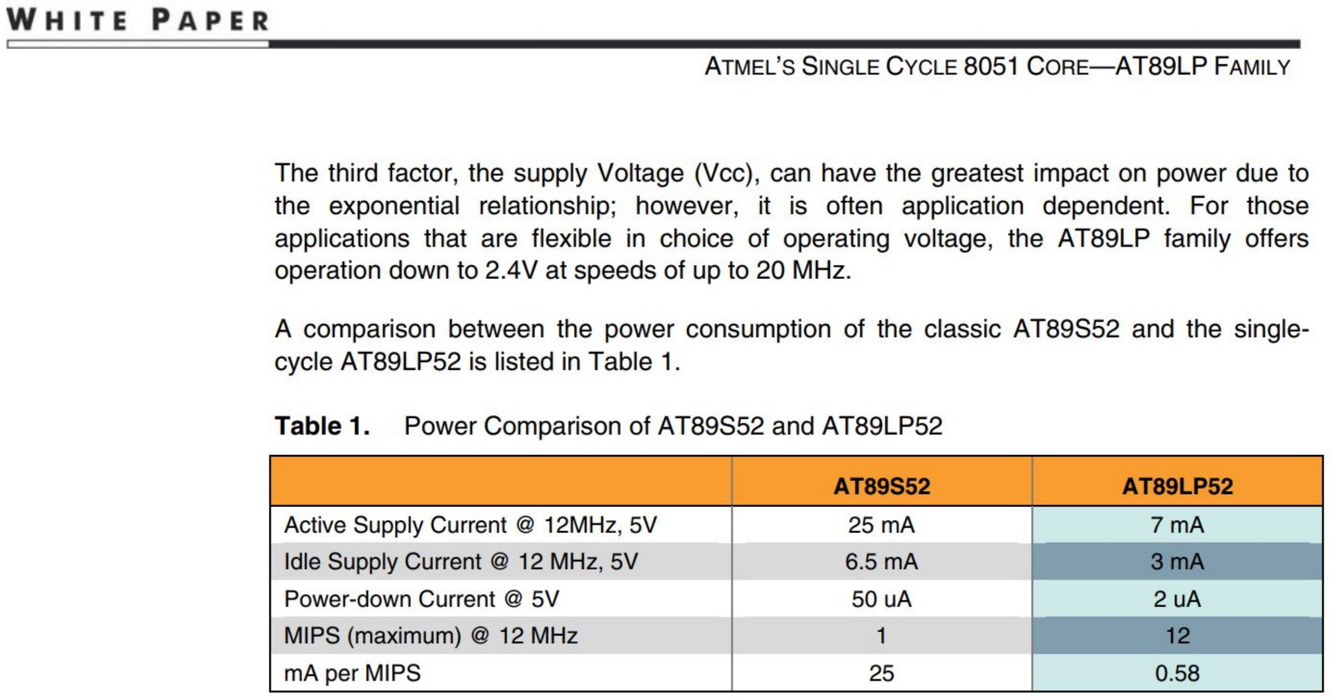
Low Power

The Atmel AT89LP Family was designed not only with performance in mind, but also power consumption. How can an architecture designed for high performance also be low power? Typical dynamic power is often written with the following equation where C is the capacitance toggled, Vcc is the supply voltage and f is the operating frequency:

$$P_{dynamic} = C \cdot Vcc^2 \cdot f$$

Power consumption can be reduced by decreasing any of these three parameters. The AT89LP family reduces all three, with the core directly affecting both the C and f factors. The AT89LP core was designed to limit the number of events occurring during an instruction simply by reducing the number of clocks per instruction. The core also reduces the amount of logic toggling during an instruction (the C factor).

Reducing the number of clocks per instruction also affects the frequency (f factor) in two ways. A given instruction on an AT89LP takes less time than the same instruction on a classic 8051 at the same frequency. This is critical for battery-powered applications that spend the majority of their time in a sleep mode. An AT89LP microcontroller can complete its active processing tasks faster than a classic 8051 and therefore spend more of its time in a low power sleep mode for the same workload. The inverse relationship also benefits applications that make less use of a sleep mode and have a workload that takes a fixed amount of time. The AT89LP can complete the work in the same time as a classic 8051, but at a much lower frequency, thereby reducing the total power consumption. For example, an AT89LP at 2 MHz can produce the same throughput of a classic 8051 running at 12 MHz for a power reduction of 6X.



WHITE PAPER

ATMEL'S SINGLE CYCLE 8051 CORE—AT89LP FAMILY

The third factor, the supply Voltage (Vcc), can have the greatest impact on power due to the exponential relationship; however, it is often application dependent. For those applications that are flexible in choice of operating voltage, the AT89LP family offers operation down to 2.4V at speeds of up to 20 MHz.

A comparison between the power consumption of the classic AT89S52 and the single-cycle AT89LP52 is listed in Table 1.

Table 1. Power Comparison of AT89S52 and AT89LP52

	AT89S52	**AT89LP52**
Active Supply Current @ 12MHz, 5V	25 mA	7 mA
Idle Supply Current @ 12 MHz, 5V	6.5 mA	3 mA
Power-down Current @ 5V	50 uA	2 uA
MIPS (maximum) @ 12 MHz	1	12
mA per MIPS	25	0.58

Hay datos suficientes para recalcular el ejemplo con estos dos procesadores

Anexo: ARM Cortex-M Low Power Modes

Cada microcontrolador basado en el procesador Arm Cortex-M tendrá al menos tres **Power Modes: Run, Sleep y Deep Sleep**. El modo Run es cuando el procesador está completamente encendido. El modo Sleep detendrá el reloj de la CPU pero dejará operativos el reloj del sistema, la memoria flash y los periféricos. El modo Deep Sleep no sólo detendrá el reloj de la CPU, sino que también apagará el reloj del sistema, la memoria flash y el PLL.

El fabricante de microcontroladores tiene la capacidad de personalizar completamente sus módulos de bajo consumo. Por ejemplo, los procesadores NXP Kinetis-L no solo tienen los modos estándar de bajo consumo Cortex-M, sino también modos como Low-Leakage Sleep Mode y Very Low Leakage Sleep Mode, que consumen sólo micro o nanoamperes de corriente.

Es importante tener en cuenta que cuanto más profundamente duerme el microcontrolador, más cerca está de apagarse por completo. Es importante darse cuenta de esto porque a medida que utiliza modos de suspensión cada vez más profundos, el tiempo necesario para volver a encender el microcontrolador y comenzar a ejecutar instrucciones puede aumentar dramáticamente. Algunos procesadores que ofrecen modos de energía muy profundos requieren la misma cantidad de tiempo que la secuencia de inicio del procesador para volver a estar operativos. Dependiendo de su aplicación, esto podría tener efectos significativos en el rendimiento en tiempo real del sistema.

Instrucción WFI (Wait for Interrupt)

When the WFI instruction is encountered, the processor will immediately move into the low power sleep mode that is configured in the **System Control Block (SCB)** unless there is a pending interrupt. Once the processor is asleep, it will wake-up when an interrupt is triggered. Some processors may even have an optional Wakeup-Interrupt-Controller which allows a developer to configure what sources are allowed to wake up the processor. This provides a developer with finer grain control over how long their system can remain in the lowest power sleep mode.

Within the **SCR register** there is a **SLEEPDEEP** bit that if set to 1 will put the processor into a deep sleep mode. If the bit is cleared to 0, the processor will go into the sleep mode. As I mentioned earlier, the sleep modes are often enhanced by the silicon vendor so there will also be additional registers that need to be configured based on who designed your microcontroller.

System Control Register

The SCR controls features of entry to and exit from low power state. See the register summary in Table 4.12 for its attributes. The bit assignments are:

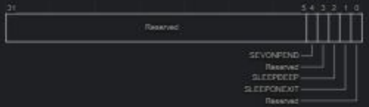


Table 4.19. SCR bit assignments

Bits	Name	Function
[31:5]	-	Reserved.
[4]	SEVONPEND	Send Event on Pending bit: 0 = only enabled interrupts or events can wakeup the processor, disabled interrupts are excluded1 = enabled events and all interrupts, including disabled interrupts, can wakeup the processor. When an event or interrupt enters pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. The processor also wakes up on execution of an `SEV` instruction or an external event.
[3]	-	Reserved.
[2]	SLEEPDEEP	Controls whether the processor uses sleep or deep sleep as its low power mode: 0 = sleep 1 = deep sleep.
[1]	SLEEPONEXIT	Indicates sleep-on-exit when returning from Handler mode to Thread mode: 0 = do not sleep when returning to Thread mode 1 = enter sleep, or deep sleep, on return from an ISR. Setting this bit to 1 enables an interrupt driven application to avoid returning to an empty main application.
[0]	-	Reserved.

Low Power Modes in STM32

https://controllerstech.com/low-power-modes-in-stm32/

Documentado perfectamente en el Reference Manual del STM32

The device requires a 2.0-to-3.6 V operating voltage supply (VDD). An embedded regulator is used to supply the internal 1.8 V digital power.

The real-time clock (RTC) and backup registers can be powered from the VBAT voltage when the main VDD supply is powered off.

By default, the microcontroller is in Run mode after a system or a power Reset. Several low power modes are available to save power when the CPU does not need to be kept running, for example when 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 and available wakeup sources.

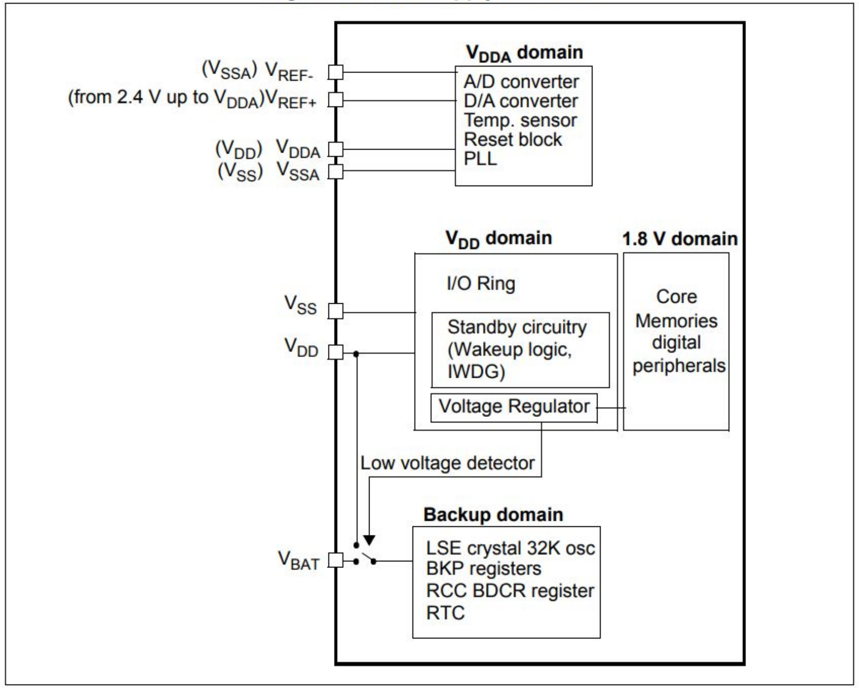
The STM32F10xxx devices feature three low-power modes:

- **Sleep mode** (CPU clock off, all peripherals including Cortex®-M3 core peripherals like NVIC, SysTick, etc. are kept running)
- **Stop mode** (all clocks are stopped)
- **Standby mode** (1.8V domain powered-off)

In addition, the power consumption in Run mode can be reduce by one of the following means:

- Slowing down the system clocks
- Gating the clocks to the APB and AHB peripherals when they are unused.

Figure 4. Power supply overview



1. V_{DDA} and V_{SSA} must be connected to V_{DD} and V_{SS}, respectively.

SLEEP MODE

In this mode, CPU CLK is turned OFF and there is no effect on other clocks or analog clock sources. The current consumption is HIGHEST in this mode, compared to other Low Power Modes. In order to enter the SLEEP MODE, we must disable the systick interrupt first, or else this interrupt will wake the MCU every time the interrupt gets triggered.

```
HAL_SuspendTick();
```

Next, we will enter the sleep mode by executing the WFI (Wait For Interrupt), or WFE (Wait For Event) instructions. If the WFI instruction was used to enter the SLEEP MODE, any peripheral interrupt acknowledged by the NVIC can wake up the device.

```
HAL_PWR_EnterSLEEPMode(PWR_MAINREGULATOR_ON, PWR_SLEEPENTRY_WFI);
```

The device will wakeup whenever any interrupt is triggered.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  HAL_ResumeTick();
}
```

Inside the callback function we can resume the systick, so that we can use the delay function again for the rest of the code.

SLEEPONEXIT: This is another feature available in SLEEP MODE, where the MCU will wake up when the interrupt is triggered, it will process the ISR, and go back to sleep when the ISR exits. This is useful when we want the controller to run only in the interrupt mode.

Ver ejemplo y video con mediciones: Run 6.7 mA -> Sleep 1.8 mA

STOP MODE

In Stop mode, all clocks in the 1.2 V domain are stopped, the PLLs, the HSI and the HSE RC oscillators are disabled. Internal SRAM and register contents are preserved. STOP MODE have may different categories depending on what should be turned off. See STM32F446RE reference manual. Just like sleep mode, in order to enter the STOP MODE, we must disable the systick interrupt, or else this interrupt will wake the MCU every time the interrupt gets triggered.

```
HAL_SuspendTick();
```

Next, we will enter the sleep mode by executing the WFI (Wait For Interrupt), or WFE (Wait For Event) instructions. If the WFI instruction was used to enter the STOP MODE, EXTI, Independent watchdog (IWDG), or RTC can wake up the device. Also I am using the first mode as shown in the picture above i.e. The main Regulator will be turned off and only the LOW Power Regulator will be running.ce.

```
HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON, PWR_STOPENTRY_WFI);
```

We have entered the STOP MODE using the WFI instruction, so the device will wakeup whenever any interrupt is triggered by an EXTI, Independent watchdog (IWDG), or RTC. We must reconfigure the SYSTEM CLOCKS after wakeup, as they were disabled when entering the STOP MODE. Also don’t forget to resume the systick.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  if(GPIO_Pin == GPIO_PIN_13) {
    SystemClock_Config ();
    HAL_ResumeTick();
  }
}
```

SLEEPONEXIT: idem
Ver ejemplo y video con mediciones: Stop xx mA

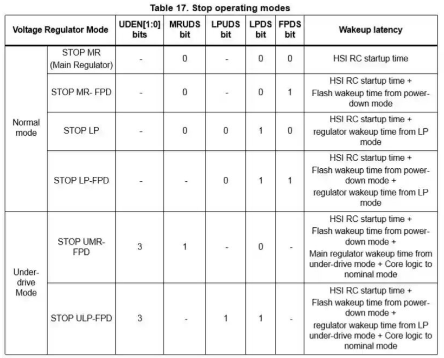


Table 17. Stop operating modes

Voltage Regulator Mode		UDEN[1:0] bits	MRUDS bit	LPUDS bit	LPDS bit	FPDS bit	Wakeup latency
Normal mode	STOP MR (Main Regulator)	-	0	-	0	0	HSI RC startup time
	STOP MR- FPD	-	0	-	0	1	HSI RC startup time + Flash wakeup time from power-down mode
	STOP LP	-	0	0	1	0	HSI RC startup time + regulator wakeup time from LP mode
	STOP LP-FPD	-	-	0	1	1	HSI RC startup time + Flash wakeup time from power-down mode + regulator wakeup time from LP mode
Under-drive Mode	STOP UMR-FPD	3	1	-	0	-	HSI RC startup time + Flash wakeup time from power-down mode + Main regulator wakeup time from under-drive mode + Core logic to nominal mode
	STOP ULP-FPD	3	-	1	1	-	HSI RC startup time + Flash wakeup time from power-down mode + regulator wakeup time from LP under-drive mode + Core logic to nominal mode

STANDBY MODE

The Standby mode allows to achieve the lowest power consumption. It is based on the Cortex®-M4 with FPU deepsleep mode, with the voltage regulator disabled. The 1.2 V domain is consequently powered off. The PLLs, the HSI oscillator and the HSE oscillator are also switched off.

Before entering the STANDBY MODE, we must disable the wakeup flags as shown belo

```
/* Clear the WU FLAG */
__HAL_PWR_CLEAR_FLAG(PWR_FLAG_WU);
/* clear the RTC Wake UP (WU) flag */
__HAL_RTC_WAKEUPTIMER_CLEAR_FLAG(&hrtc, RTC_FLAG_WUTF);
```

Next, enable the wakeup pin, or the RTC periodic wakeup (if you want to use RTC)

```
HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN1);
```

and finally enter the STANDBY MODE

```
HAL_PWR_EnterSTANDBYMode();
```

The standby wakeup is same as a system RESET. The entire code runs from the beginning just as if it was a RESET. The only difference between a reset and a STANDBY wakeup is that, when the MCU wakesup, The SBF status flag in the PWR power control/status register (PWR_CSR) is set. The wakeup can be triggered by WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.

See entire code for the STANDBY MODE
Ver ejemplo y video con mediciones: Standby 2 uA

Low Power Modes in STM32

https://controllerstech.com/low-power-modes-in-stm32/