低功耗系统设计

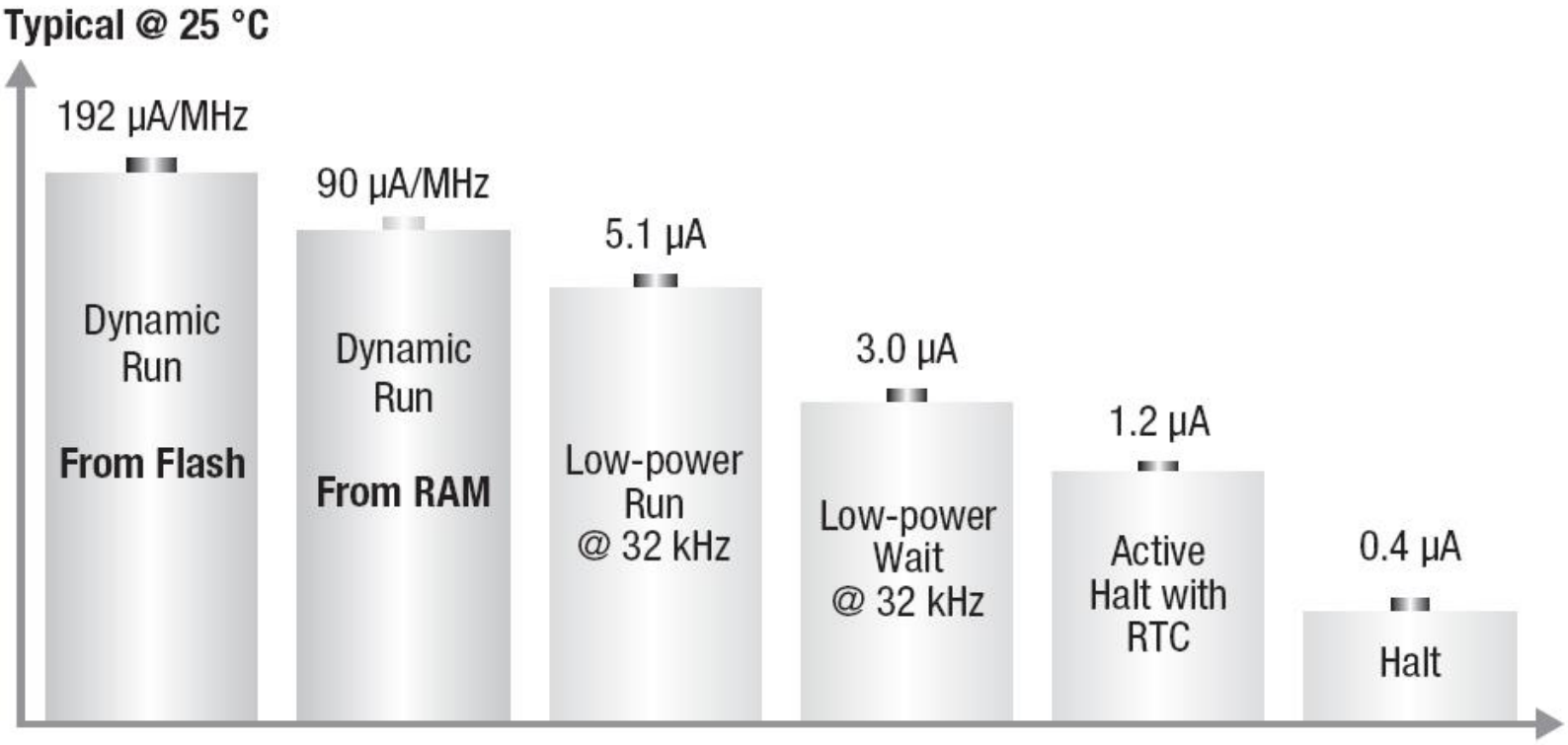
life.augmented

- ST超低功耗MCU特性
- 功耗分类

ST超低功耗MCU特性

- STM32L和STM8L对比
- STM32L和其他MCU对比

STM8L 功耗



Notes:
- POR/PDR on
- RAM content preserved
- BOR option at 2.4 µA
- Startup time from active Halt 5 µs
- Run and Wait consumption values are independent of V_{DD}
- Active Halt and Halt values measured at V_{DD} = 1.8 V

STM32L0 功耗

典型电流

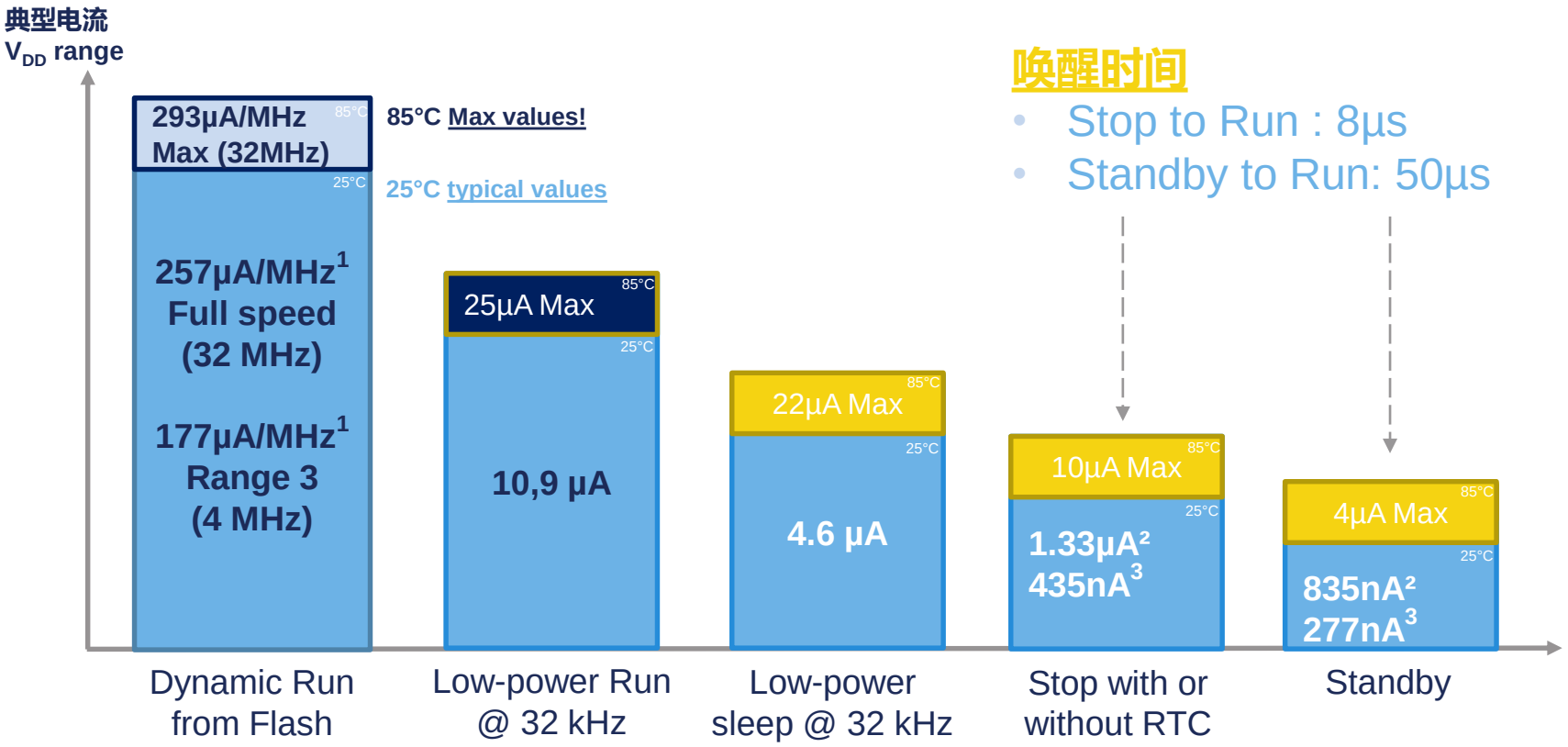
V_{DD} range

唤醒时间

- 停止模式（STOP）到运行：5µs
- 待机模式（Standby）到运行: 600µs

life.augmented

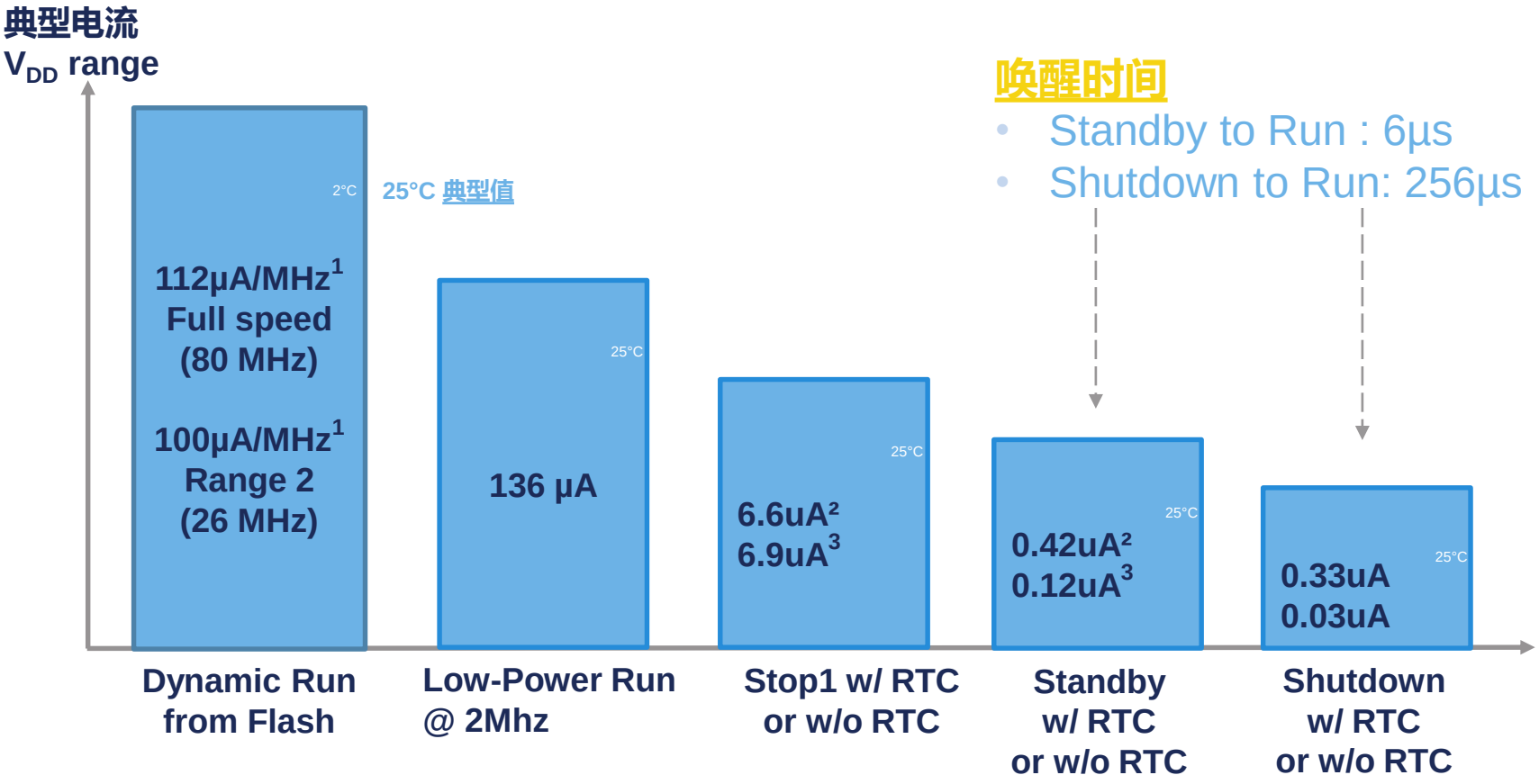
STM32L1 功耗



1. Dhrystone power consumption value executed from Flash with V_{DD} = 3V
2. Stop and Standby with RTC given with V_{DD} = 1.8V
3. Stop and Standby without RTC given with V_{DD} = 3V

life.augmented

STM32L4 功耗



唤醒时间

- Standby to Run : 6µs
- Shutdown to Run: 256µs

1. Dhrystone power consumption value executed from Flash with code is while (1) and Art disable

ST超低功耗MCU特性

STM32L和STM8L对比

STM32L和其他MCU对比

EEMBC ULP score

Clear	Device	EEMBC CERTIFIED	ULPMark™-CP	Active Mode Clock Config	Inactive Power Mode	External DC-DC	Compiler	Revision	Prod. Silicon	Core	Date
☐	Ambiq Micro APOLLO512-KBR	✓	377.50	CPU: 24 MHz, RTC: 32.768 kHz	Deep Sleep		ARM GCC 4.8.3 20140228	Rev A3	✓	Cortex-M4	11/07/15
☐	Analog Devices ADuCM302x	✓	245.50	CPU: 26 MHz, RTC: 32 kHz	Hibernate		IAR EWARM 7.50.2.10505	Rev 1.0	✓	Cortex-M3	02/05/16
☐	STMicroelectronics STM32L433		204.90	CPU: 24MHz (MSI), RTC: 32.768kHz (LSE)	Standby with RTC and SRAM2	ST1PS01EJR Rel 1.1	IAR C/C++ compiler for ARM v6.60.1.5097	Rev 1		Cortex-M4	03/14/16
☐	Texas Instruments MSP432P401R Rev. C	✓	192.30	CPU: 16MHz, RTC: 32KHz	LPM3		IAR EWARM v7.50.3	Rev C	✓	Cortex-M4	02/19/16
☐	STMicroelectronics STM32L476RG		187.70	CPU: 24MHz (MSI), RTC: 32.768kHz (LSE)	Standby with SRAM2 retention	ST1PS01EJR rel 1.1	IAR C/C++ compiler for ARM v6.60.1.5097			Cortex-M4	11/02/15
☐	Atmel SAML21J18A-UES RevA - DC1506	✓	185.80	CPU : 12MHz, RTC 32,768kHz	STANDBY, PD0,PD1,PD2 in retention state		IAR EWARM 7.30.3.802	RevA		Cortex-M0+	02/03/15
☐	STMicroelectronics STM32L433	✓	176.70	CPU: 24MHz (MSI), RTC: 32.768kHz (LSE)	Standby with RTC and SRAM2		IAR C/C++ compiler for ARM v6.60.1.5097	Rev 1	✓	Cortex-M4	12/18/15
☐	STMicroelectronics STM32L011K4		161.00	CPU: 4MHz (MSI), RTC: 32.768kHz (LSE)	Stop with RTC	ST1PS01EJR Rel 1.1	IAR C/C++ compiler for ARM v6.60.1.5097			Cortex-M0+	03/15/16
☐	STMicroelectronics STM32L476RG	✓	153.00	CPU: 24MHz (MSI), RTC: 32.768kHz (LSE)	Standby with SRAM2 retention		IAR C/C++ compiler for ARM v6.60.1.5097		✓	Cortex-M4	05/26/15
☐	Texas Instruments CC2650F128RGZ		143.60	CPU: 48MHz, RTC: 32768Hz	Standby		IAR 7.30.4.8186	V2.2	✓	Cortex-M3	02/19/15

功耗概念

life.augmented

低功耗

- **功耗**即功率的损耗，指设备、器件等输入功率和输出功率的**差额**
- 基于数据手册列出的电流消耗数据比较来选择低功耗**单片机**(MCU)是一项比较困难的任务
- 在大多数情况下，选择MCU的开发人员会先初步看看数据手册第一页，作为快速获得器件信息的参考点，其中包括外设、运行速度、封装信息、GPIO引脚数量和供电特性等。这种方法对于获得器件的整体性能很有效，但是在评估低功耗特性时却不实用
- 如何加快功耗评估呢？

为什么要做低功耗

- 以手机/可穿戴设备为代表的电池供电电路的兴起，为便携式仪表开创了一个新纪元
- 对于网络时代（尤其是IOT），越来越多的设备是需要电池来供电
- 系统是220V供电，就不用在乎功耗问题了？
  - 低功耗设计并不仅仅是为了省电，更多的好处在于降低了电源模块及散热系统的成本、由于电流的减小也减少了电磁辐射和热噪声的干扰。随着设备温度的降低，器件寿命则相应延长（半导体器件的工作温度每提高10度，寿命则缩短一半）

如何评估功耗(1)

- 传统方式
  - 参考数据手册，估算大概的功耗
  - 使用评估板或者是自己做板子，使用范例程序来测试每种模式下的功耗电流
  - 根据实际的使用各种模式的频率来估算平均功耗
  - 最后根据平均功耗和电池的电量来评估这款MCU是否可以使用
  - 备注：其中最占用时间的是打板与各种模式的测试

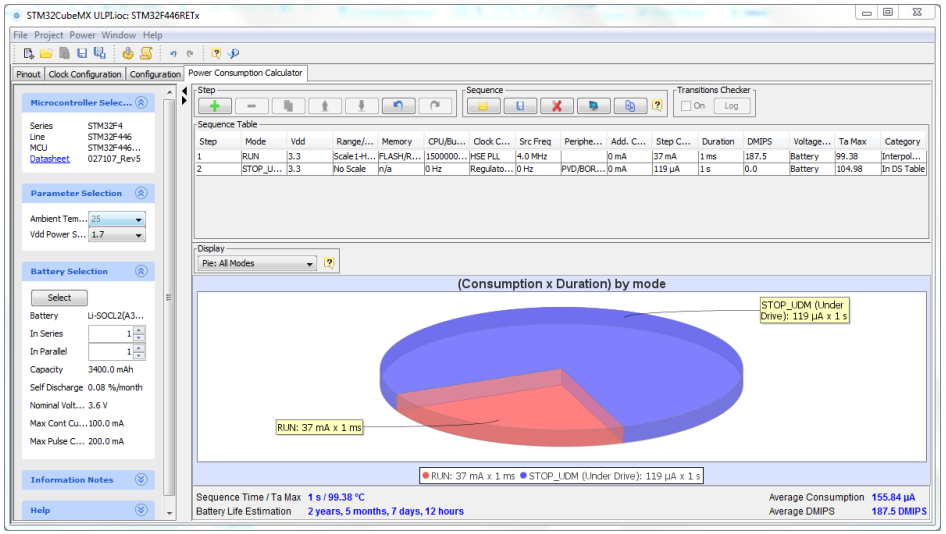
STM32L0-Dsicovery

STM8L-Dsicovery

STM32L1-Dsicovery

如何评估功耗(2)

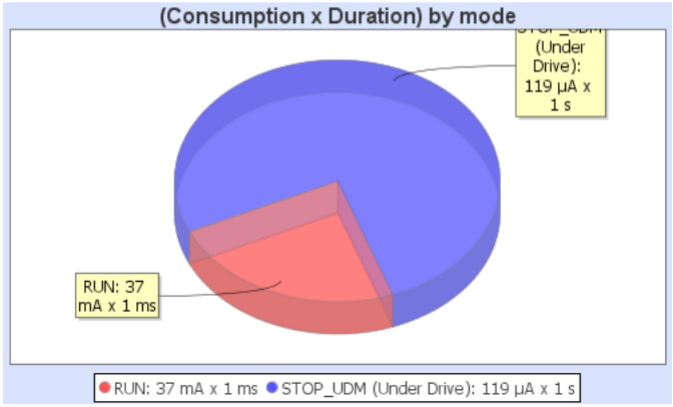
- ST的新方式
  - 首先是建立数学模型，估计产品在各种模式下所用的时间
  - 使用STM32CubeMX中自带的计算器，将数据填写在其中，记得将电池的参数一并输入，或者是在其提供的列表中，选择参数最接近的一种电池
  - 直接生成功耗报告，得到在指定的电池供电情况下，产品能工作多长时间



7.5. RESULTS

Sequence Time	1 s	Average Current	155.84 µA
Battery Life	2 years, 5 months, 7 days, 12 hours	Average DMIPS	187.5 DMIPS

7.6. Chart



如何比较电气规格

- 电气性能规格罗列在数据手册中，通过仔细研究才能判断哪种规格更加重要。一般情况下，定义一系列操作条件并对应到一种电源模式更有意义。例如，开发人员可能会定义下面一组操作条件：
  - 状态保持和RAM保持条件下的休眠模式电流消耗
  - 所有其他外设禁用
  - RTC启用且状态保持和RAM保持条件下的休眠模式电流消耗
  - RTC启用，所有其他外设禁用
- 唤醒时间参数
- 供电电压范围

一旦操作条件有明确的定义，那么就很容易判定属于何种电源模式

不可忽略的额外低功耗特性

- 第二部分是低功耗特性，其散布在数据手册和参考手册中。低功耗功能的示例包括：
  - 可用的唤醒源
  - 如何恢复代码执行
  - 在休眠模式下可操作的外设
  - ……

一旦操作模式明确定义之后，开发人员就开始研究文档中的更多细节。

功耗分类

功耗评估

ST超低功耗产品参数

功耗分类（1）

RUN

- 全速运行模式
- 此时的动态功耗占主导地位

STOP

- 低速半休眠模式
- 动态功耗和静态功耗取决于运行时间

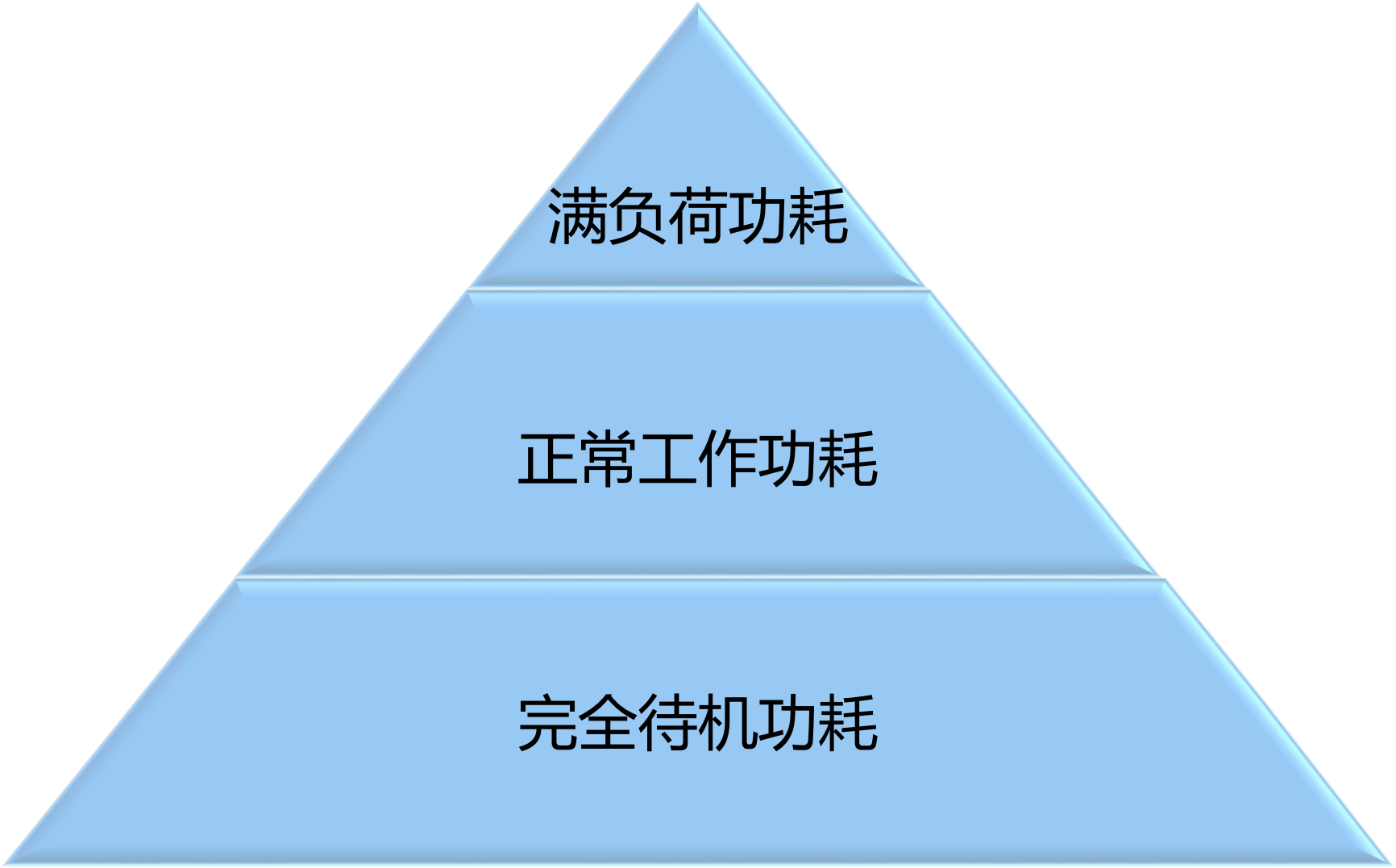
Shut Down

- 停止模式
- 静态功耗占主导地位

根据芯片工作状态

- 一般可分为工作模式和低功耗模式
- 工作模式下主要看动态功耗，我们一般就只是看能效这个参数（mA/MHz）
- 在低功耗模式下，我们就需要看静态功耗与可工作的外设功耗
  - ADC
  - 比较器
  - 内部集成运放
  - 内部时钟
  - ……..

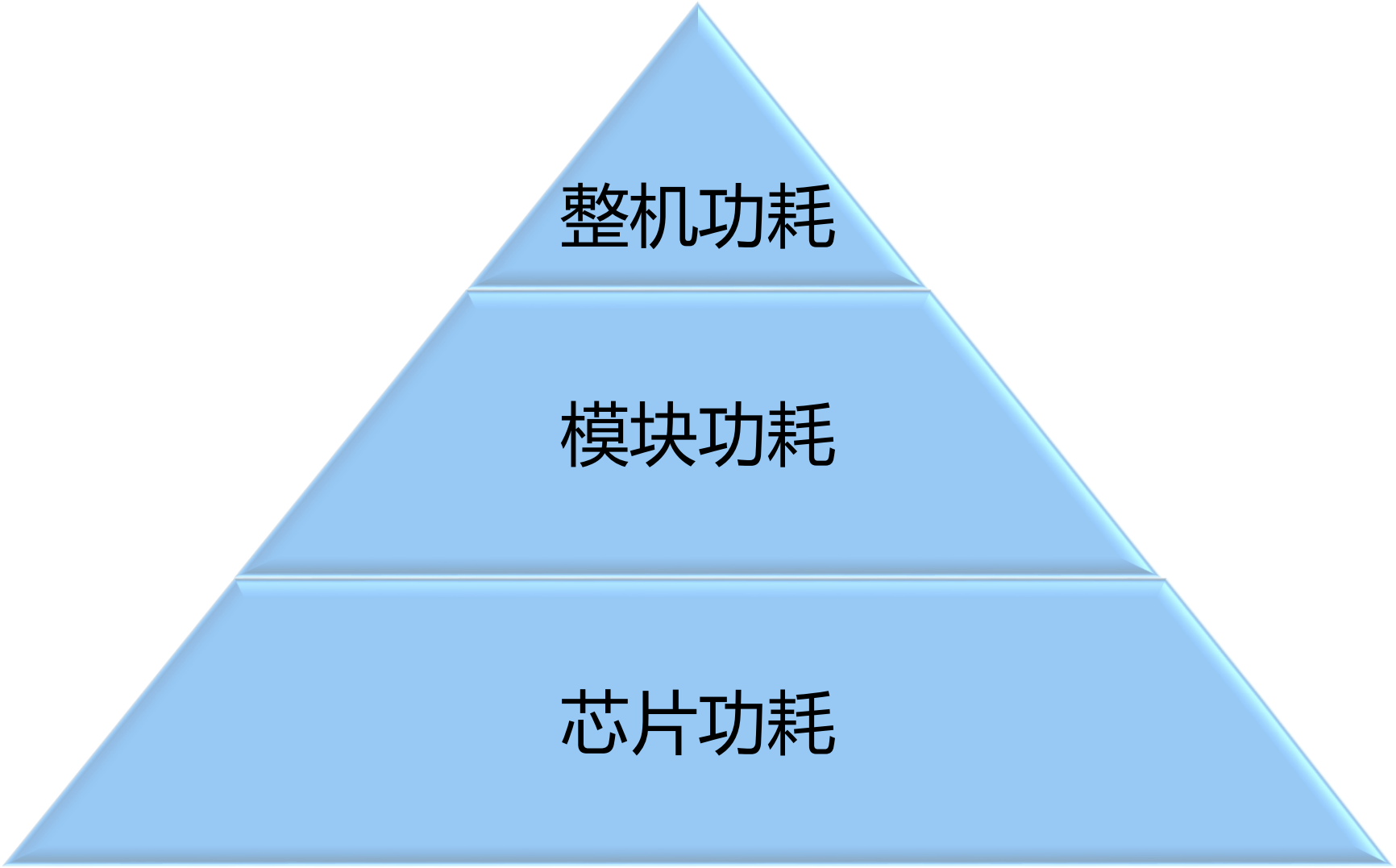
功耗分类（2）



根据产品工作状态划分

- 产品工作状态一般可以分为以下几种情况
  - 采集数据阶段
  - 数据解析阶段
  - 数据通讯阶段
- 一般的数据采集和数据解析中，我们都是希望越快越好，所以一般都是工作在高主频阶段
- 在数据通讯阶段，我们一般希望通讯的次数越少越好，每次的数据通讯所占的时间越少越好，那么MCU就有空闲的时间进入到低功耗模式了

功耗分类（3）



根据产品组成划分

- 最终产品的功耗并不是单独一方面所决定的，一般产品的功耗可分成：
- 模拟功耗，数字功耗
- 采集部分功耗，处理部分功耗
- 驱动部分功耗，控制部分功耗
- ……

芯片内部功耗产生

CHMOS 或 CMOS 电路的功耗特性一般可以表示为：

P=PD+PA

式中， P--总功耗

PD--静态功耗，

PD=VDD • IDD　　(1)

PA--动态功耗，

PA=PTC+PC＝VDD • ITC+Fc1v2dd (2)

PTC --瞬时导通功耗

PC--输出电容充放电功耗

VDD--工作电源电压

IDD--静态时由电源流向电路内部的电流

ITC--脉冲电流的时间平均值

f--输入脉冲重复频率

CL--电路输出端的负载电容

硬件选型/设计

软件设计

其他

基本原则

- 选用尽量简单的CPU内核
  - 在满足功能的情况下，越简单的MCU越省电
- 选择低电压供电的系统
  - 电压与功耗有着2次方的关系
- 选择带有低功耗模式的处理器+外设
  - 灵活切换低功耗模式，能省下一点就是一点
- 选择合适的时钟方案
  - 大部分功耗都来源于时钟震荡
- 选择与MCU电压匹配的外围器件
  - 电源与软硬件配合才是低功耗设计的根本原则

低功耗器件选型

- 并非所有的低功耗芯片组成的就一定是最低功耗产品
- 电源的选型是低功耗产品设计的关键
- MCU的功耗和软件设计关系较大，选用专门为低功耗产品设计的MCU会减少很多软件设计上的困难，比如其会增加更多的低功耗特性与低功耗外设（比如STM32L系列的低功耗串口、低功耗定时器）
- 最耗电的外设，一定要有使能的管脚（可关断最好）
- 供电电压会对你的产品功耗影响较大（尽量选用低电压器件）
- 开关首选MOSFET

影响功耗的因素

- 静态功耗
  - 通常所说的最低功耗
  - 有时候也指待机功耗（低功耗运行状态）
- 电源部分：
  - LDO的选择
  - DCDC的选择
- PCB板的注意事宜
  - 正常工作的情况下尽量增大各个通路的阻抗
  - 最小化高速开关路径中的阻抗
- 最小化工作时间占空比
  - 工作时间越短，功耗越低

外设接口设计

- 我们经常会做如下的“惯性设计”

  - **存储器有这么多控制信号，我这块板子只需要用OE和WE信号就可以了，片选就接地吧，这样读操作时数据出来得快多了？**

    - 大部分存储器的功耗在片选有效时（不论OE和WE如何）将比片选无效时大100倍以上，所以应尽可能使用CS来控制芯片，并且在满足其它要求的情况下尽可能缩短片选脉冲的宽度。

  - **这些总线信号都用电阻拉一下，感觉放心些？**

    - 信号需要上下拉的原因很多，但也不是个个都要拉。上下拉电阻拉一个单纯的输入信号，电流也就几十微安以下，但拉一个被驱动了的信号，其电流将达毫安级，现在的系统常常是地址数据各32位，可能还有244/245隔离后的总线及其它信号，都上拉的话，几瓦的功耗就耗在这些电阻上了

电源设计中的注意事项

- 静态功耗Iq
- LDO的选择
  - 输入电压×电流-输出电压×电流为这个芯片的工作功耗+静态功耗，如果有使能端的LDO，那么在不需要的时候可以将其禁止掉（通过MCU的GPIO即可实现自锁功能），这样就会将其后边的电路都断电，达到较好的功耗
- DCDC的选择
  - 转换效率与设计负载
- LDO具有较小的文波，是数据采集系统的首选；DCDC的效率更高，是对于电源文波不敏感的系统首先。

未使用的管脚应该如何处理

- **“CPU的这些不用的I/O口怎么处理呢？先让它空着吧，以后再说？”这种想法对吗？**
  - 根据外设的电平状态来设置连接到外设的GPIO
  - 软件设置为模拟输入（参考AN4555）
  - 没有使用的管脚建议都连接到一个固定的电平上，设置为输出状态*
  - 如果是模拟管脚，关毕数字时钟部分

电池电路设计

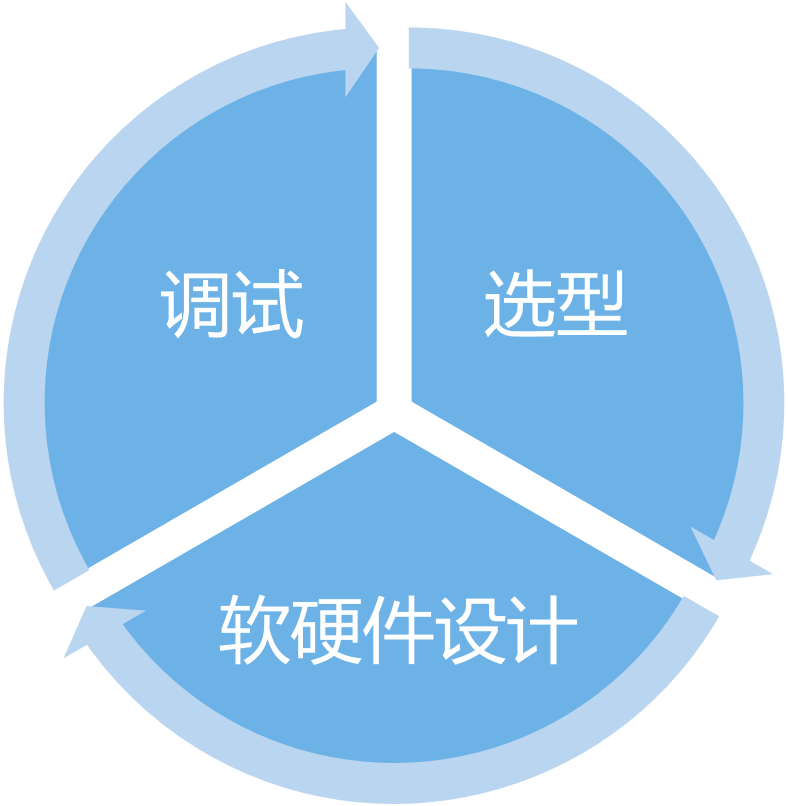
- 对于电池供电的系统，电源中可能完全不需要使用稳压器
- DCDC对于电池供电来说并不是一个最好的选择
- 两个权衡因素
  - 电池电压的下降
  - 所有其他器件都需电压

低功耗产品软件设计

life.augmented

低功耗产品软件设计相关概念

- **“降低功耗都是硬件人员的事，与软件没关系”。这样对吗？**
  - 硬件是搭个舞台，唱戏的却是软件，总线上几乎每一个芯片的访问、每一个信号的翻转差不多都由软件控制的，如果软件能减少外存的访问次数（多使用寄存器变量、多使用内部Cache等）、及时响应中断（中断往往是低电平有效并带有上拉电阻）及其它针对具体单板的特定措施都将对降低功耗作出很大的贡献



RTOS对功耗的影响

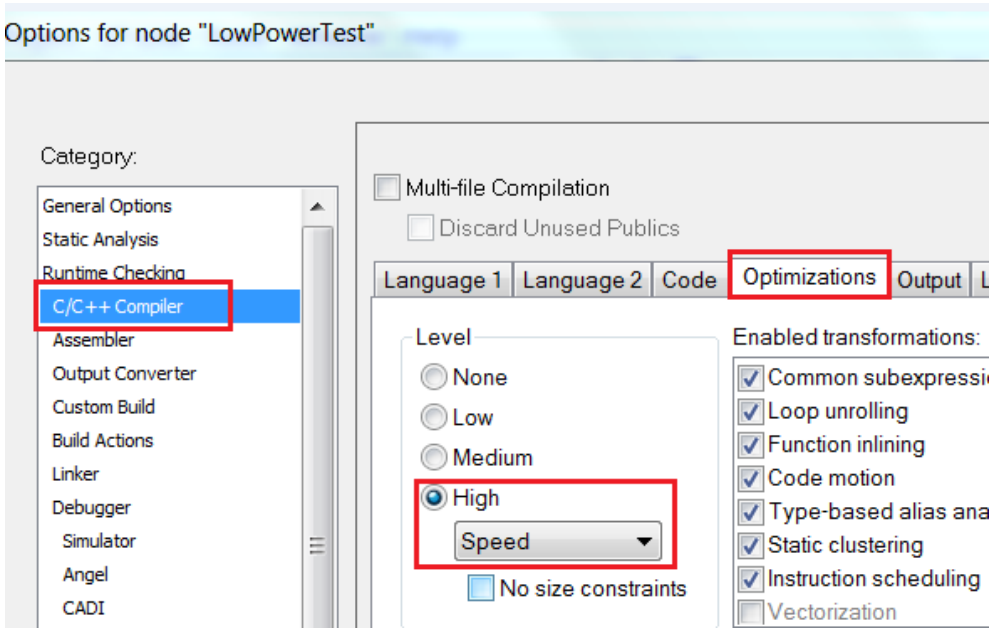
- 使用了RTOS对于功耗的整体是会增加的
  - 在时间上：会有系统时间的开销
  - 在空间上：会有代码空间的增加
- 但对需要一直循环工作的低功耗产品中，我们也是可以进行优化的
  - 在空闲任务中修改RTOS的系统时钟
  - 加入STOP低功耗模式

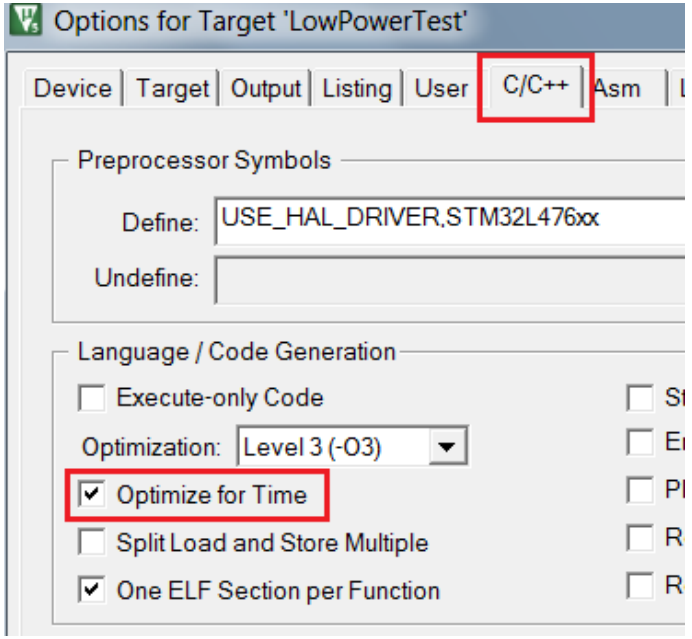
编译优化等级(1)

- 减少整体运行时间（但是也不能增加并联的相关操作）

IAR:
Options → C/C++ Compiler
→Optimization → High “Speed”

MDK:
Options → C/C++
→Optimize for time



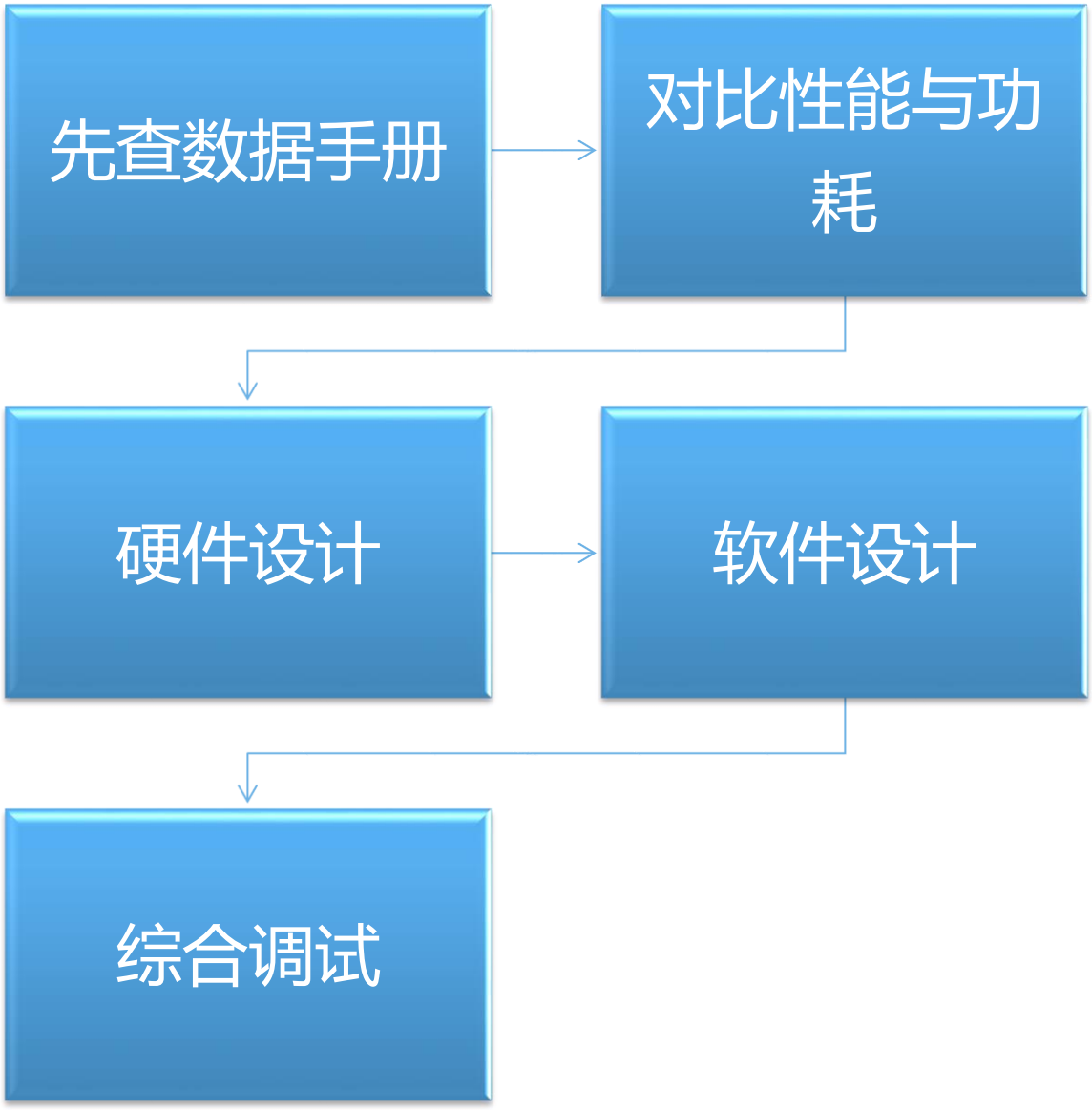


编译优化等级(2)

- 使用查表的方式替换MCU的计算
- 汇编到C语言的区别
  - 汇编不会比C语言更节省功耗
  - C语言在操作寄存器上没有汇编简洁，但是执行后的结果是相同的

低功耗软件设计原则

- 用“中断”代替“查询”
  - 使用中断方式，CPU可以什么都不做，甚至 可以进入等待模式或停止模式；而查询方式下，CPU必须不停地访问I / O寄存器，这会带来很多额外的功耗。
- 用“宏”代替“子程序”
  - 子程序调用的入栈出栈操作，要对RAM进行两次操作，会带来更大的功耗。宏在编译时展开，CPU按顺序执行指令。使用宏，会增加程序的代码量，但对不在乎程序代码量大的应用，使用宏无疑会降低系统 的功耗
- 尽量减少CPU的运算量
- 让I/O模块间歇运行
- 忙时多用，闲时少用，不用关闭

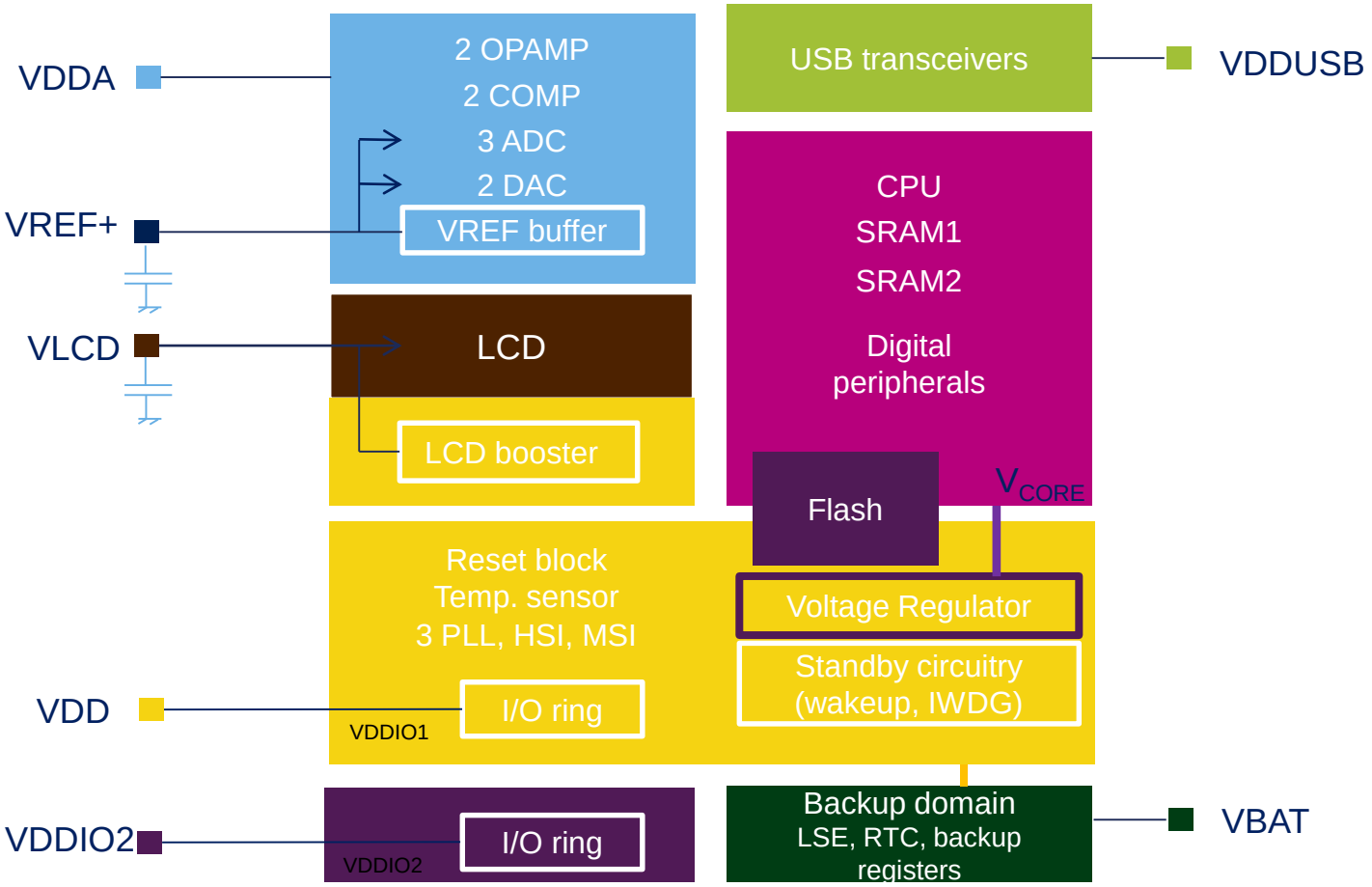

先查数据手册
对比性能与功耗
硬件设计
软件设计
综合调试

STM32L产品的功耗实现

life.augmented

ARM

电源框架图


VDDA
2 OPAMP
2 COMP
3 ADC
2 DAC
VREF buffer
USB transceivers
VDDUSB
VREF+
CPU
SRAM1
SRAM2
Digital peripherals
VLCD
LCD
LCD booster
Flash
VCORE
Reset block
Temp. sensor
3 PLL, HSI, MSI
Voltage Regulator
Standby circuitry (wakeup, IWDG)
VDD
I/O ring
VDDIO1
VDDIO2
I/O ring
VDDIO2
Backup domain
LSE, RTC, backup registers
VBAT

电源框图区别

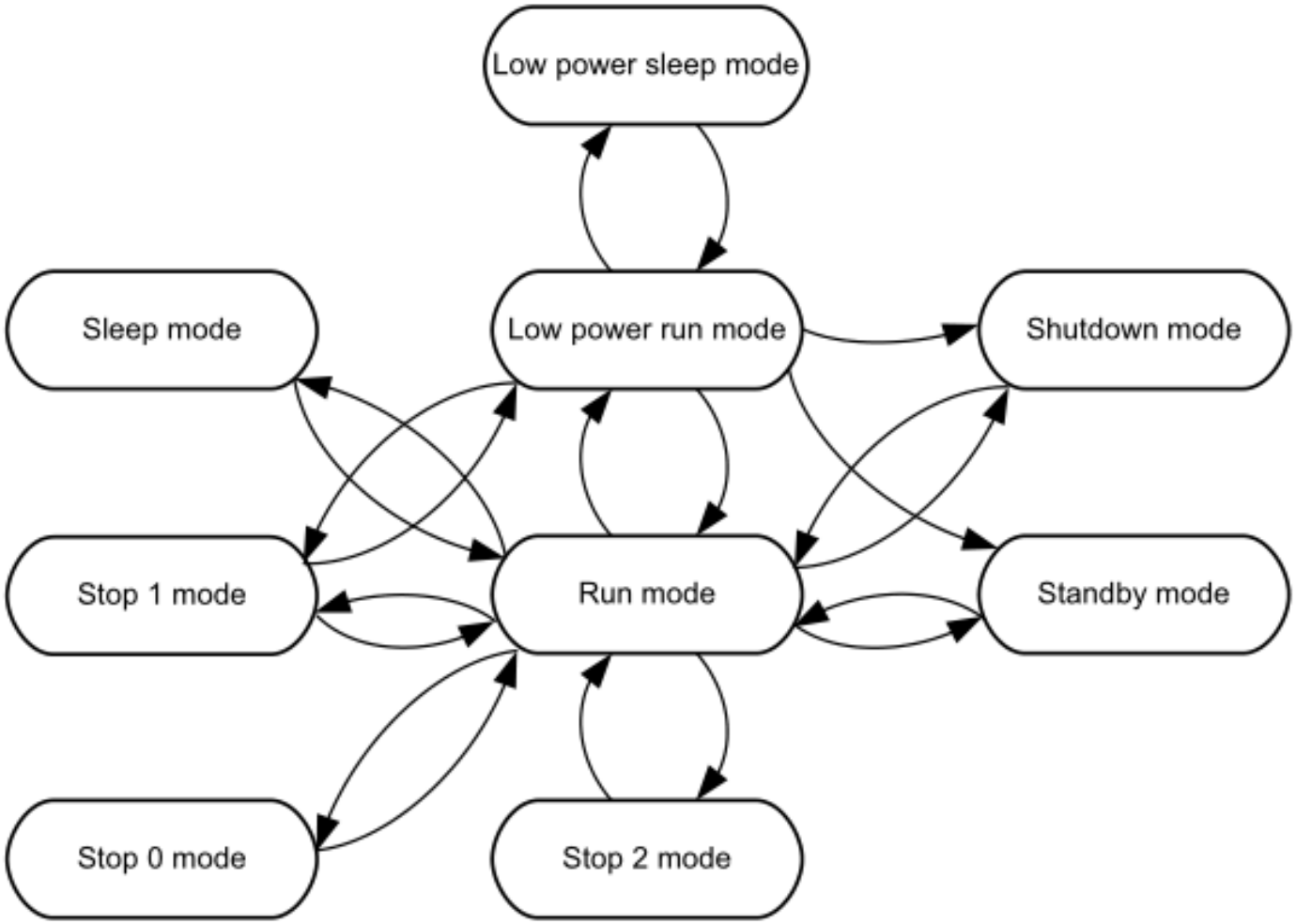
- VDDIO2*
  - 单独给一组GPIO PG[15:2]来供电，其GPIO电压由VDDIO2决定
  - 供电电压：
    - 0-3.6V　　其GPIO都没有用到时
    - 1.08V-3.6V 只要有其中一个GPIO被指用
- VDDUSB
  - 单独给USB外设供电管脚
  - 供电电压：
    - 3.0V--3.6V　　当USB外设使用
    - 连到VDD　　当USB外设不使用
- Vref+
  - 这是一个双向的管脚，当作内部参考电压的输出，当作外部参考电压的输入
  - 输入电压：
    - 2V – VDDA 当VDDA 大于等于2V
    - VDDA　　当VDDA小于2V
  - 输出电压：
    - 2.048V 当VDDA大于2.4V
    - 2.5V　　当VDDA大于2.8V

低功耗模式总结（1）

Mode	Regulator	CPU	Flash	SRAM	Clocks	Peripherals	Consumption @ 1.8V	Wakeup time
Run	R1	**Yes**	**ON**(1)	**ON**	Any	All	127 µA/MHz	N/A
	R2					All except OTG, RNG	111 µA/MHz	
LPRun	LPR	**Yes**	**ON**(1)	**ON**	Any except PLL	All except OTG, RNG	136 µA/MHz	
Sleep	R1	No	**ON**(1)	**ON**(2)	Any	All Any IT or event	37 µA/MHz	6 cycles
	R2						35 µA/MHz	
LPSleep	LPR	No	**ON**(1)	**ON**(2)	Any except PLL	All except OTG, RNG Any IT or event	40 µA/MHz	6 cycles
Stop 0	MR	No	OFF	**ON**	LSE/LSI	Reset pin, all I/Os BOR,PVD,PVM,RTC,LCD,IWDG, COMPx,DACx,OPAMPx,USARTx, LPUART,I2Cx,LPTIMx,OTG_FS,SWPMI	110µA	0.7 µA RAM 5µA Flash memory
Stop 1	LPR						6.6 µA w/o RTC 6.9 µA w/RTC	4 µA RAM 6 µA Flash memory
Stop 2	LPR	No	OFF	**ON**	LSE/LSI	Reset pin, all I/Os BOR,PVD,PVM,RTC,LCD,IWDG, COMPx,LPUART,I2C3,LPTIM1	1.1 µA w/o RTC 1.4 µA w/RTC	5 µA RAM 7 µA Flash memory
Standby	LPR	DOWN	OFF	**SRAM2 ON**	LSE/LSI	Reset pin, 5 WKUPx pins BOR, RTC, IWDG	+ 235 nA	14 µs
	OFF			DOWN			115 nA w/o RTC 415 nA w/RTC	
Shutdown	OFF	DOWN	OFF	DOWN	LSE	Reset pin, 5 WKUPx pins RTC	30 nA w/o RTC 330 nA w/RTC	250 µs

1. Can be put in power-down and clock can be gated off
2. SRAM1 and SRAM2 can be gated off independently

低功耗模式总结（2）


Low power sleep mode
Sleep mode
Low power run mode
Shutdown mode
Stop 1 mode
Run mode
Standby mode
Stop 0 mode
Stop 2 mode

如何进入低功耗模式

- 通过设置寄存器
  - F0/F1/F2/F3/F4/xx：PDDS+LPDS+SLEEPDEEP+WFI/WFE
  - F7xx: PDDS+SLEEPDEEP+(LPDS, MRUDS, LPUDS and UDEN)+WFI/WFE
  - L0/L1:PDDS+LPSDSR+SLEEPDEEP+LPRUN+WFI/WFE
  - L4: LPMS+ RRS+ SLEEPDEEP+ LPR +WFI/WFE
- 通过调用API
  - HAL_PWR_EnterSLEEPMode()
  - HAL_PWR_EnterSTANDBYMode()
  - HAL_PWR_EnterSTOPMode()
  - HAL_PWREx_EnterSTOP0/1/2Mode()
  - HAL_PWREx_EnterSHUTDOWNMode()

Mode name	Entry	Wakeup source(1)	Wakeup system clock	Effect on clocks	Voltage regulators	
					MR	LPR
Sleep (Sleep-now or Sleep-on-exit)	WFI or Return from ISR	Any interrupt	Same as before entering Sleep mode	CPU clock OFF no effect on other clocks or analog clock sources	ON	ON
	WFE	Wakeup event				
Low-power run	Set LPR bit	Clear LPR bit	Same as Low-power run clock	None	OFF	ON
Low-power sleep	Set LPR bit + WFI or Return from ISR	Any interrupt	Same as before entering Low-power sleep mode	CPU clock OFF no effect on other clocks or analog clock sources	OFF	ON
	Set LPR bit + WFE	Wakeup event			OFF	ON
Stop 0	LPMS="000" + SLEEPDEEP bit + WFI or Return from ISR or WFE	Any EXTI line (configured in the EXTI registers) Specific peripherals events	HSI16 when STOPWUCK=1 in RCC_CFGR MSI with the frequency before entering the Stop mode when STOPWUCK=0.	All clocks OFF except LSI and LSE	ON	ON
Stop 1	LPMS="001" + SLEEPDEEP bit + WFI or Return from ISR or WFE				OFF	
Stop 2	LPMS="010" + SLEEPDEEP bit + WFI or Return from ISR or WFE					
Standby with SRAM2	LPMS="011"+ Set RRS bit + SLEEPDEEP bit + WFI or Return from ISR or WFE	WKUP pin edge, RTC event, external reset in NRST pin, IWDG reset	MSI from 1 MHz up to 8 MHz			
Standby	LPMS="011" + Clear RRS bit + SLEEPDEEP bit + WFI or Return from ISR or WFE	WKUP pin edge, RTC event, external reset in NRST pin, IWDG reset			OFF	OFF
Shutdown	LPMS="1--" + SLEEPDEEP bit + WFI or Return from ISR or WFE	WKUP pin edge, RTC event, external reset in NRST pin	MSI 4 MHz	All clocks OFF except LSE	OFF	OFF

STM32L进入低功耗模式

- STM32L进入到低功耗
- 进入低功耗的一般流程总结
  - 第一步：设置相关的管脚状态与唤醒方式的配置
  - 第二步：关闭与低功耗模式不相关的外设（比如时钟源）
  - 第三步：运行对应低功耗模式的操作序列或者API函数

退出低功耗模式（1）

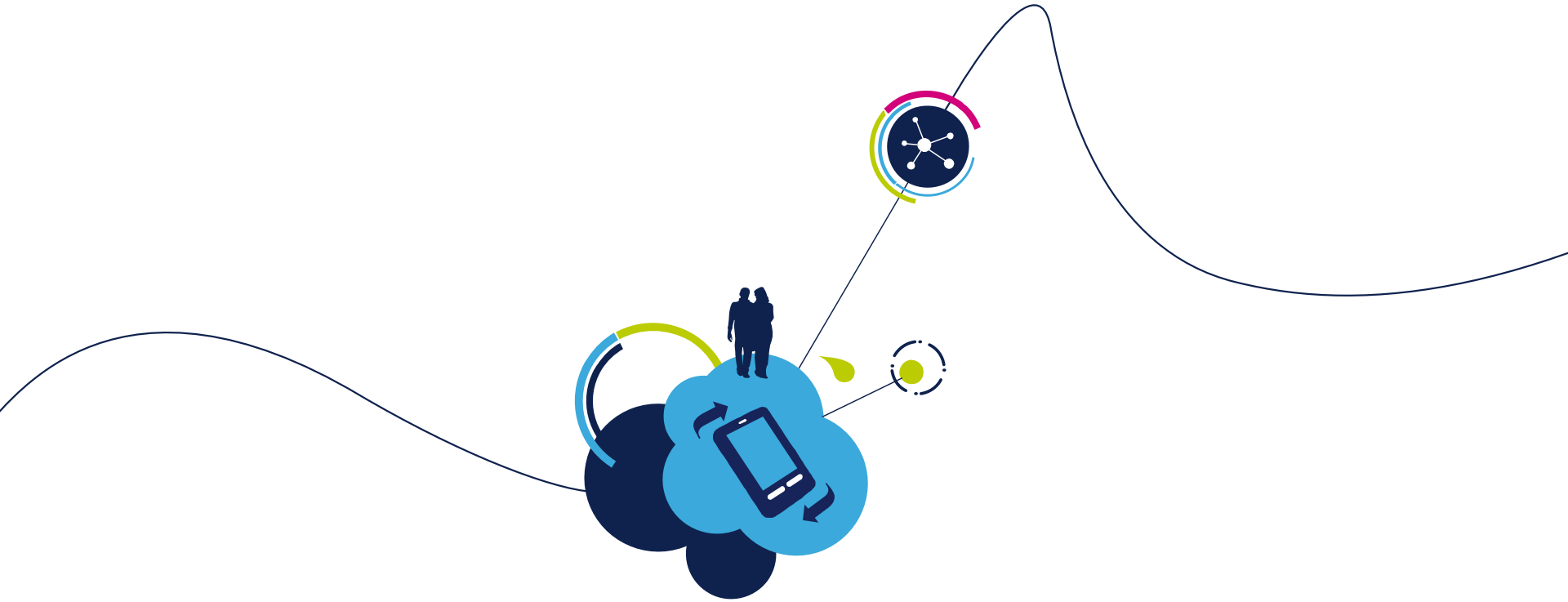
- STM32L退出低功耗
- 退出低功耗模式的一般流程总结
  - 第一步：根据对应的低功耗模式，做时钟恢复的操作
  - 第二步（可选）：是否记录进入低功耗的时间等相关信息
  - 第三步：将外设恢复到正常的运行模式状态，恢复到进入低功耗模式之前的状态。

进入退出低功耗模式需要注意事情（1）

- 时钟
  - 在STOP/STANDBY模式中退出后，时钟都会发生改变，所以在退出低功耗模式之后，一般都需要将时钟配置到之前的状态，即时钟的恢复。
  - 在一些低功耗模式中，降低时钟频率之后才能够进入；唤醒之后，时钟也是一步一步的提升上去的，而不是一次就可以恢复到最高的时钟频率。
- GPIO的状态
  - 进入低功耗模式之前，一般都需要将GPIO设置成特殊状态（根据实际连接的外设与状态决定），在退出低功耗之后，需要恢复成运行状态的设置
  - 对于不使用的GPIO，建议都连接到一个固定电压上。
- 外设的重新初始化
  - 有时外设在进入低功耗之前与进入到低功耗模式之后，外设的状态发生改变，如果想恢复，必须重新初始化，比如说锁相环（PLL）的配置。

实际案例讲解

- 下边我以STM32L476-Nucleo板子为例，讲解具体配置功耗模式的方式方法



谢谢

www.stmcu.com.cn

life.augmented