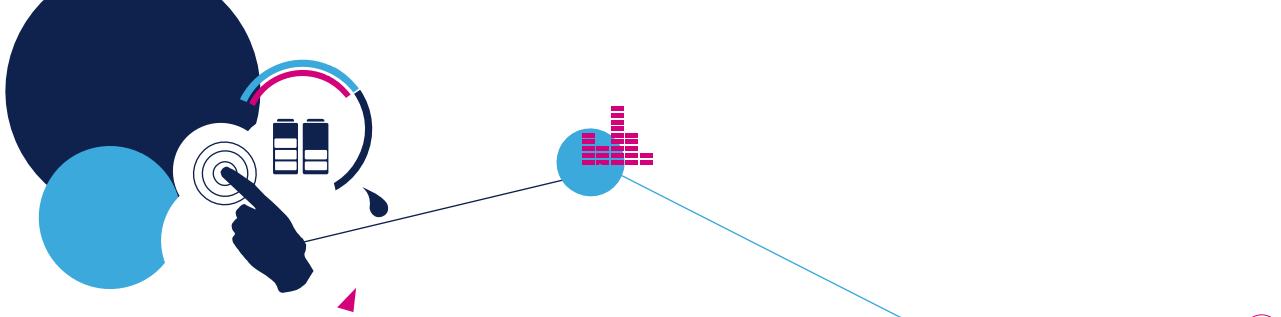


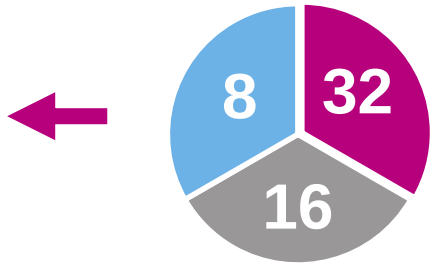
STM32L4 Tips & Tricks

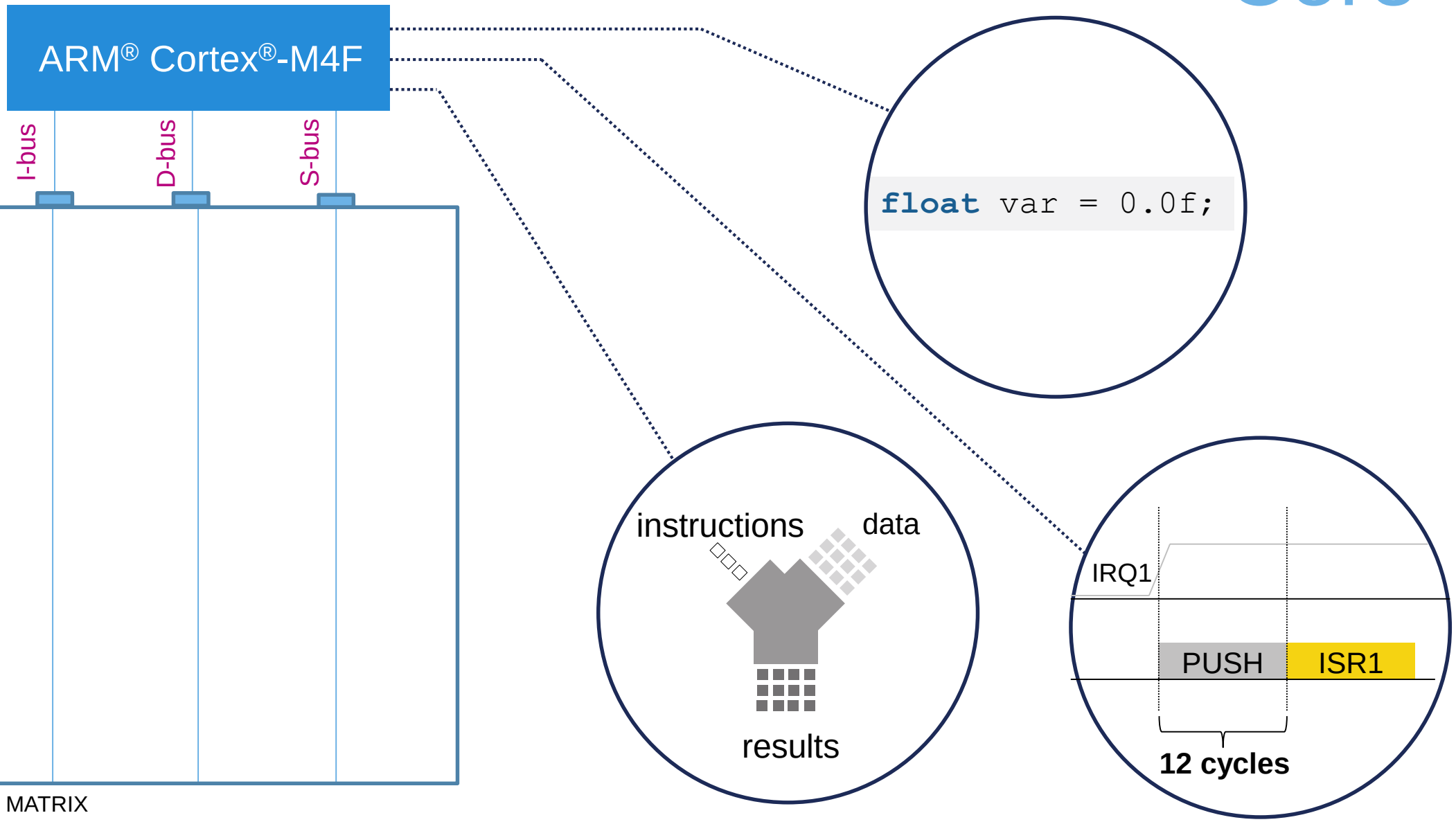
STM32L4 workshop

life.augmented

Squeeze the maximum

❶ From the architecture




ARM® Cortex®-M4F
I-bus
D-bus
S-bus
BUS MATRIX
float var = 0.0f;
instructions
data
results
IRQ1
PUSH
ISR1
12 cycles

Overview

Typ @ VDD =1.8 V @ 25 °C

** : with RTC*

*** : from SRAM1*

FlexPowerControl

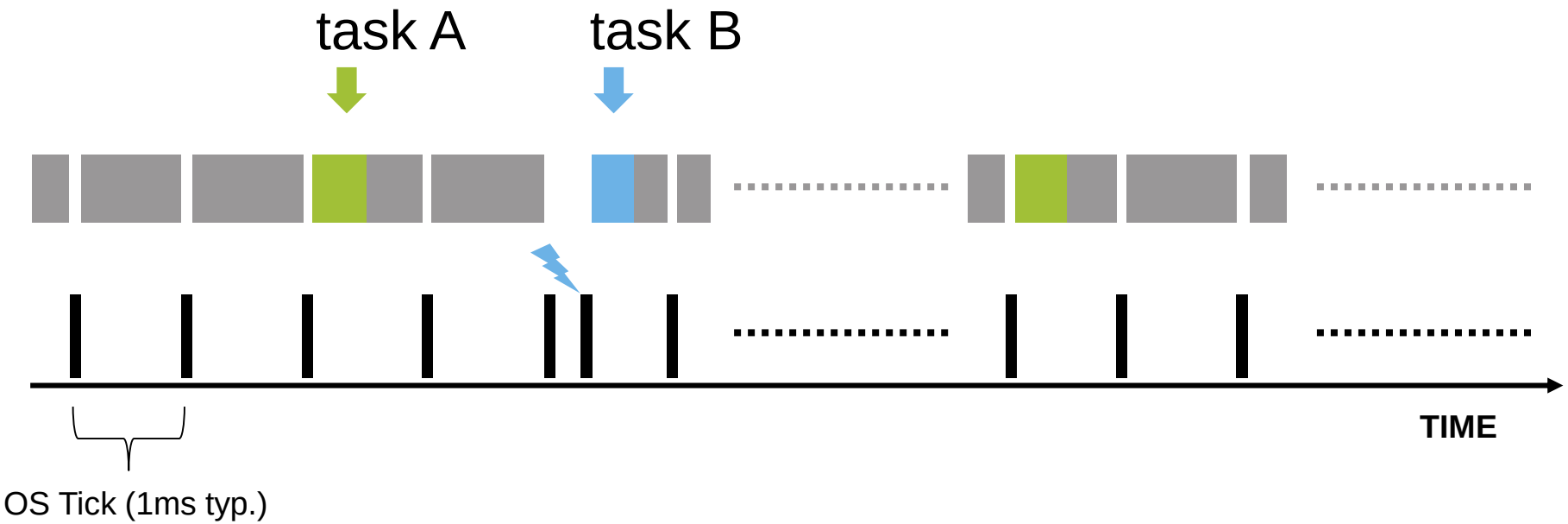
- Efficient running
- 7 low-power modes, several sub-modes
- High flexibility

Application benefits

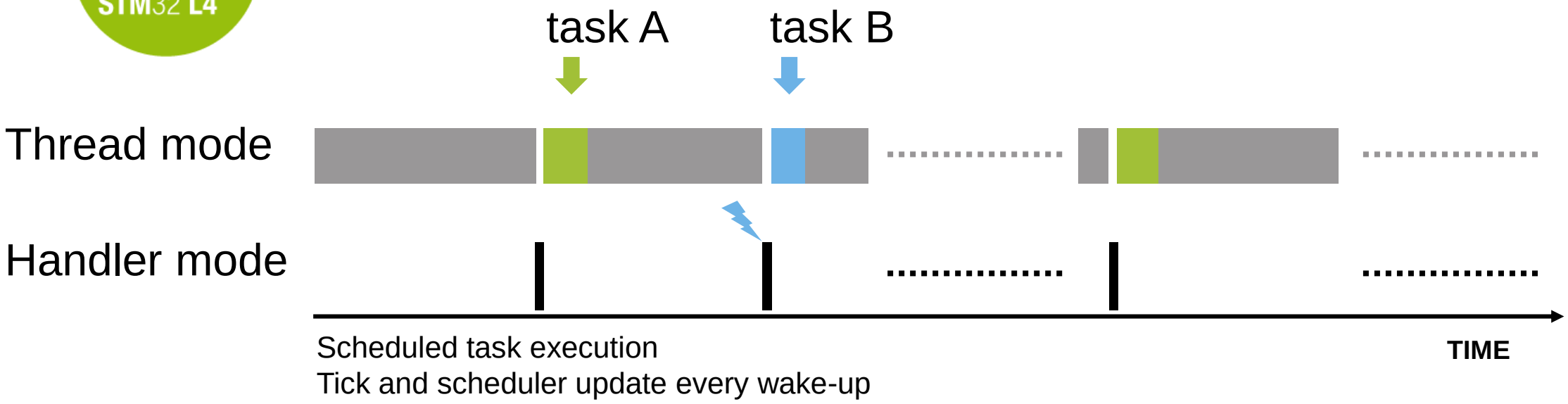
- High performance
  - ➔ CoreMark score = 273
- Outstanding power efficiency
  - ➔ ULPBbench score = 150

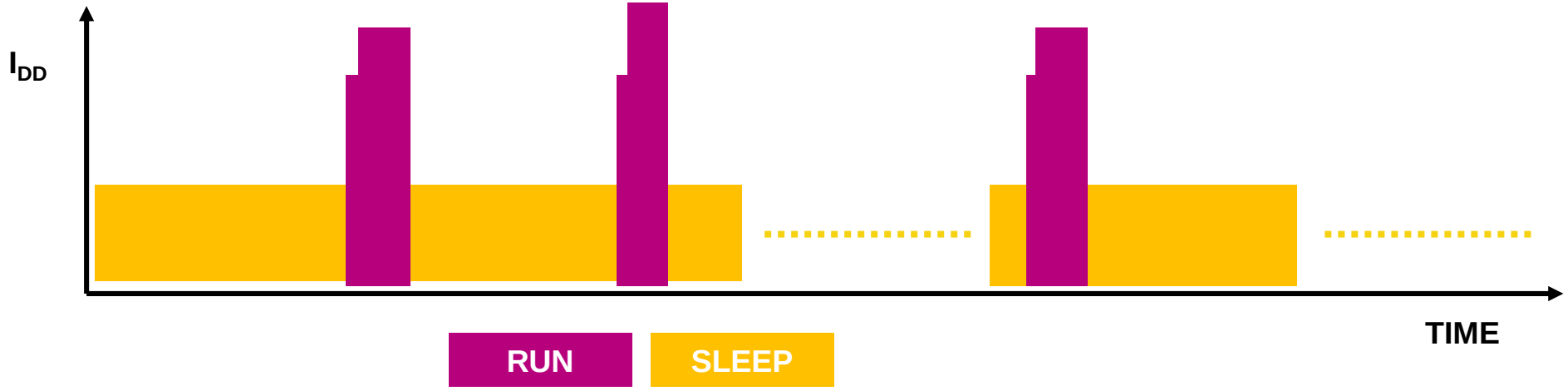
life.augmented

RTOS normal mode

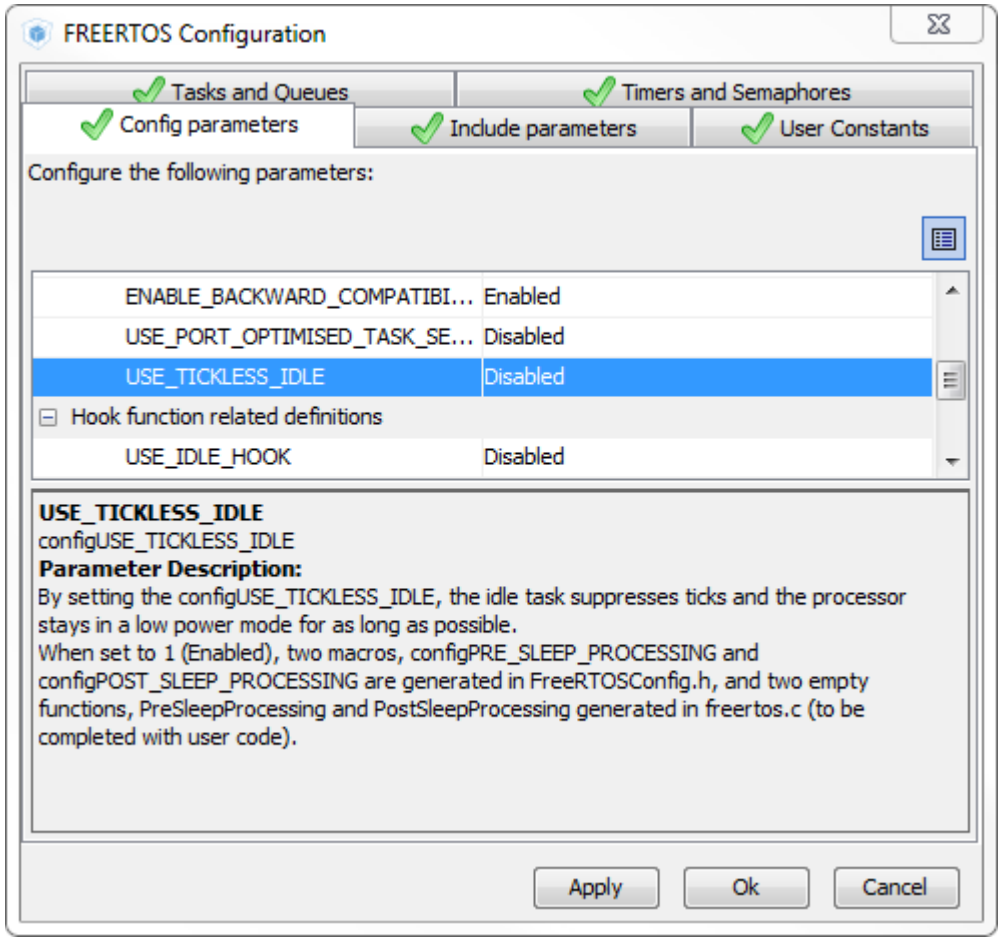


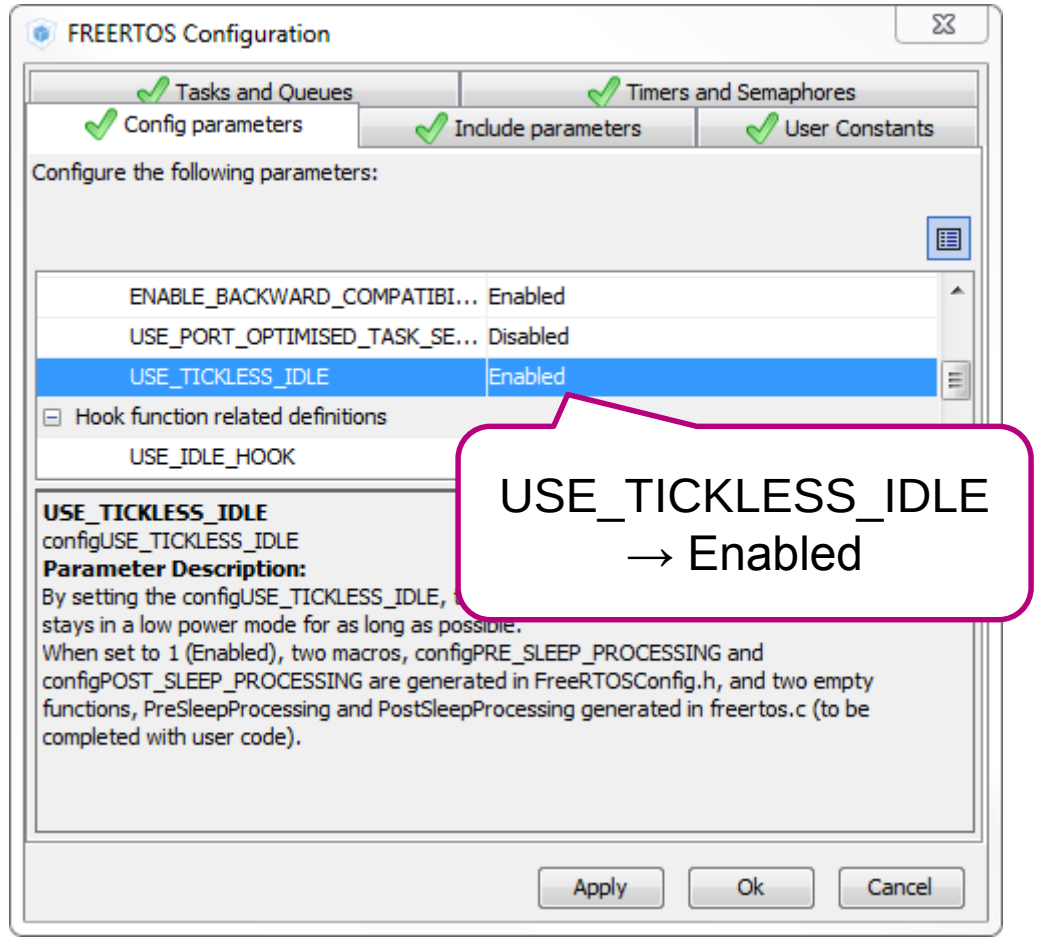
RTOS Tickless mode





How to enable Tickless mode?



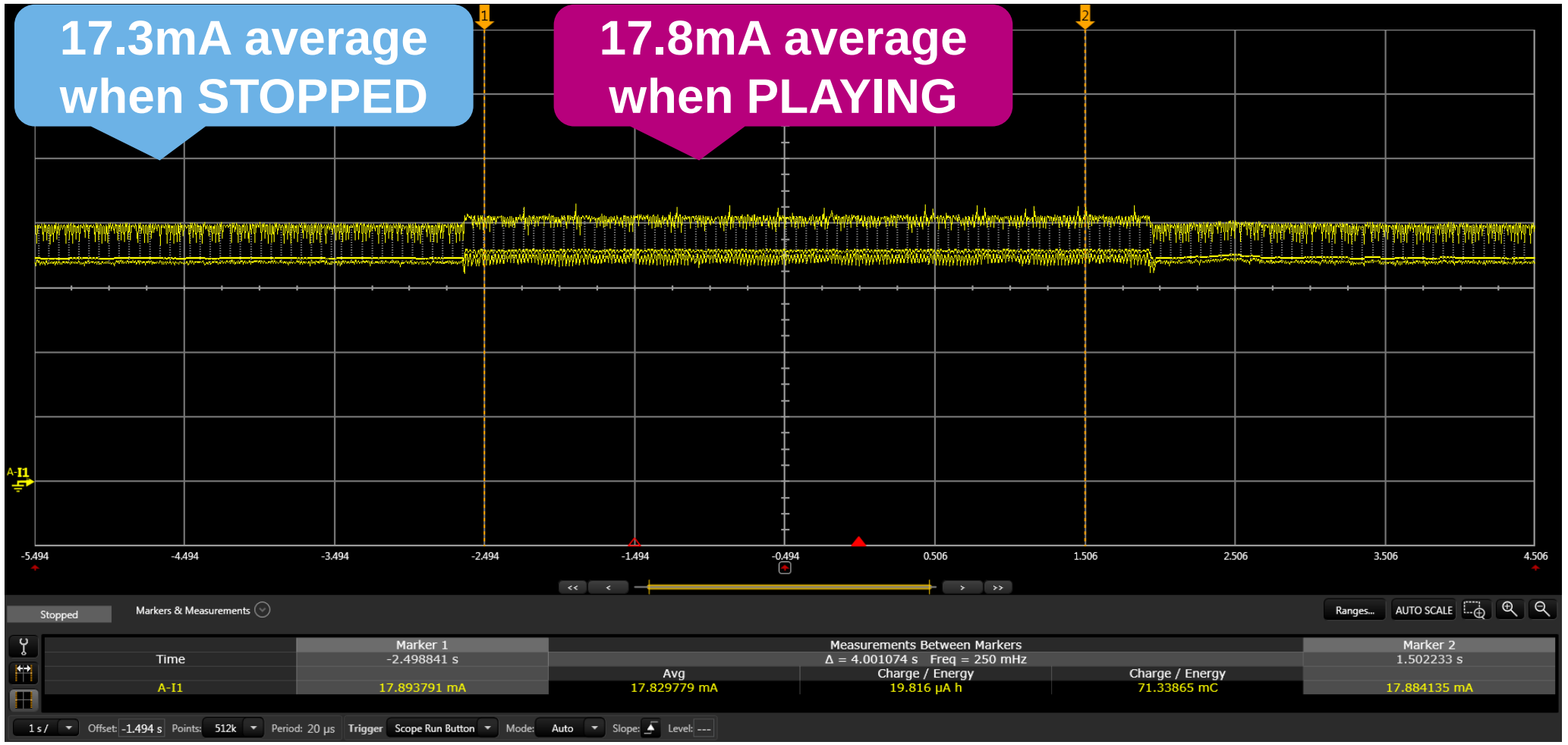


Tickless Mode in FreeRTOS

- Kernel can stop system tick interrupt and place MCU in low power mode, on exit from this mode systick counter is updated
- Enabled when setting configUSE_TICKLESS_IDLE as 1
- The kernel will call a macro portSUPPRESS_TICKS_AND_SLEEP() when the Idle task is the only task able to run (and no other task is scheduled to exit from blocked state after n ticks)
  - n value is defined in FreeRTOSconf.h file
- FreeRTOS implementation of portSUPRESS_TICKS_AND_SLEEP for cortexM3/M4 enters MCU in sleep low power mode
- Wakeup from sleep mode can be from a system interrupt/event

AudioPlayer consumption

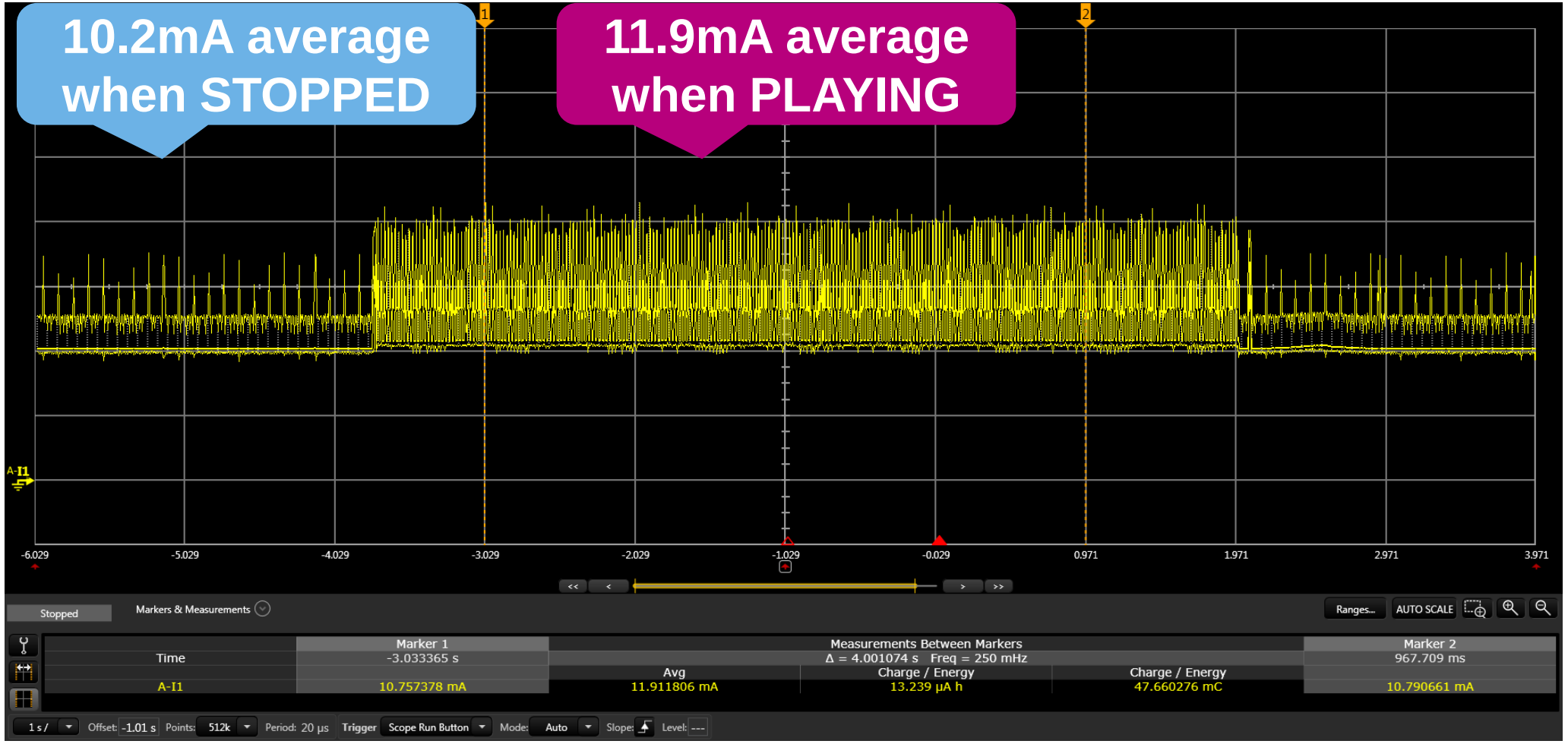
Original state (STEP6)



life.augmented

AudioPlayer optimization 1/6

RTOS Tickless Mode used



life.augmented

AudioPlayer optimization summary

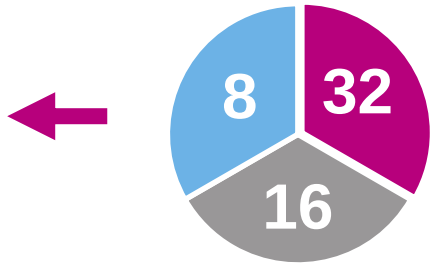
V_{DD}=3.0V @25°C

Optimization applied	STOPPED		PLAYING	
	I_{DD}	Difference	I_{DD}	Difference
Original state (STEP6)	17.3 mA	-	17.8 mA	-
Tickless Mode	10.2 mA	-41%	11.9 mA	-33.1%

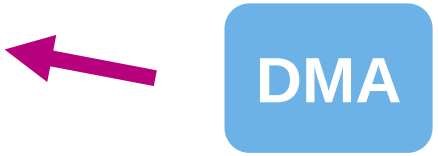
Just the MCU consumption considered
(including circuitries supplied from GPIOs)

Squeeze the maximum

1. **From the architecture**



2. **From the peripherals**



Peripheral Clock Gating

- The clock tree toward each register increases consumption, when the Bus clock is running…
- So clock toward each peripheral register group can be gated (Default is gated)

IP	range 1	range 2	
GPIOA	4.8	3.8	
GPIOB	4.8	4.0	
CRC	0.4	0.2	
DMA1	1.4	1.3	
FMC	8.9	7.5	
SYSCFG	0.6	0.4	
TIM1	8.3	6.9	
TIM17	3.0	2.4	µA/Mhz
SAI1	4.7	3.9	
QUADSPI	7.8	6.7	
LCD	1.0	0.8	
WWDG	3	2.5	
USB	39.6	N/A	
PWR	0.5	0.5	
DAC1	2.4	1.9	
…….			
ALL	256.8	189.6	

More information can be found in datasheet (Table 39: Peripheral current consumption)

IP Behavior when clock is gated 1/2

IP	Clock gated	Available interrupt
LCD	Display is running	
OpAmp	Output ok	
ADC	Conversion on going but data not stored	
DAC	Output ok, but value stays still	
COMP	Output ok	COMP1, COMP2
GPIO	Output stay still, analog input ok	EXTIs
RTC	running	Tamper/Timestamp Wake-Up/Alarm
IWDG	running	Reset
SYSCFG	Stays still,except pending request reg	PVD

IP Behavior when clock is gated 2/2

IP	Clock gated	Available interrupt
WWDG	stopped	
Comm	stopped	USB_FS_WKUP
TIMER	stopped	
AES	stopped	
CRC	stopped	
DMA	stopped	

Clocks gating and AudioPlayer

Enabled peripherals analysis

```
AHB1ENR      = 0x00000102
  TSCEN      = 0
  CRCEN      = 0
  FLASHEN    = 1
  DMA2EN     = 1
  DMA1EN     = 0
```

```
AHB2ENR      = 0x0000101F
  RNGEN      = 0
  AESEN      = 0
  ADCEN      = 0
  OTGFSEN    = 1
  GPIOHEN    = 0
  GPIOGEN    = 0
  GPIOFEN    = 0
  GPIOEEN    = 1
  GPIODEN    = 1
  GPIOCEN    = 1
  GPIOBEN    = 1
  GPIOAEN    = 1
```

```
AHB3ENR      = 0x00000100
  QSPIEN     = 1
  FMCEN      = 0
```

```
APB1ENR1     = 0x10200200
  LPTIM1EN   = 0
  OPAMPEN    = 0
  DAC1EN     = 0
  PWREN      = 1
  CAN1EN     = 0
  I2C3EN     = 0
  I2C2EN     = 0
  I2C1EN     = 1
  UART5EN    = 0
  UART4EN    = 0
  USART3EN   = 0
  USART2EN   = 0
  SP3EN      = 0
  SPI2EN     = 0
  WWDGEN     = 0
  LCDEN      = 1
  TIM7EN     = 0
  TIM6EN     = 0
  TIM5EN     = 0
  TIM4EN     = 0
  TIM3EN     = 0
  TIM2EN     = 0
```

```
APB1ENR2     = 0x00000000
  LPTIM2EN   = 0
  SWPMI1EN   = 0
  LPUART1EN  = 0
```

```
APB2ENR      = 0x00200001
  DFSDMEN    = 0
  SAI2EN     = 0
  SAI1EN     = 1
  TIM17EN    = 0
  TIM16EN    = 0
  TIM15EN    = 0
  USART1EN   = 0
  TIM8EN     = 0
  SPI1EN     = 0
  TIM1EN     = 0
  SDMMCEN    = 0
  FIREWALLEN = 0
```

Clocks gating and AudioPlayer

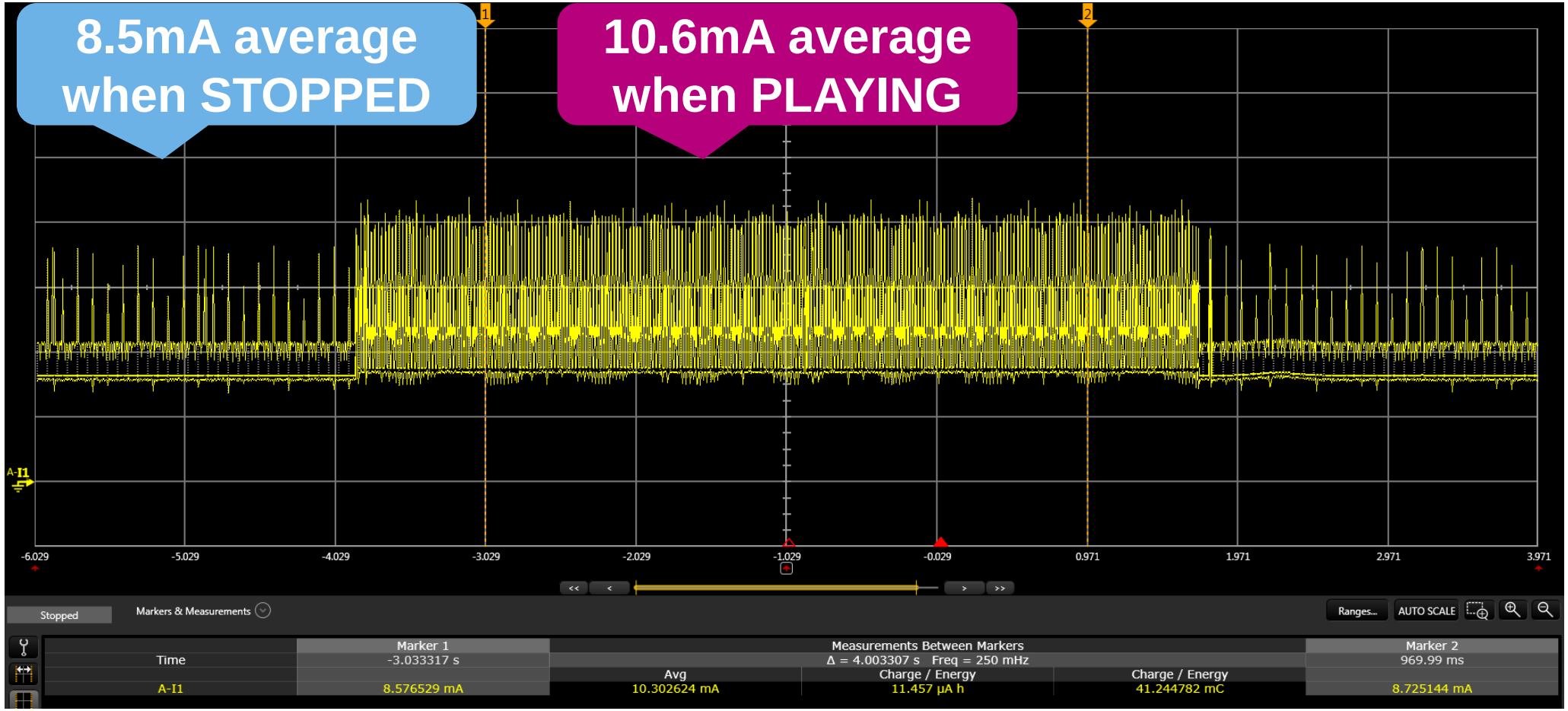
Automatic clocks gating in SLEEP mode

- Let's just make the GPIO ports, I2C1 and PWR to be automatically disabled in SLEEP mode
  - we don't need access to these peripherals in SLEEP mode
- Following code sequence to be added after configuration of all the peripherals:

```
__GPIOA_CLK_SLEEP_DISABLE();
__GPIOB_CLK_SLEEP_DISABLE();
__GPIOC_CLK_SLEEP_DISABLE();
__GPIOD_CLK_SLEEP_DISABLE();
__GPIOE_CLK_SLEEP_DISABLE();
__I2C1_CLK_SLEEP_DISABLE();
__PWR_CLK_SLEEP_DISABLE();
```

AudioPlayer optimization 2/5

Clocks gating applied



life.augmented

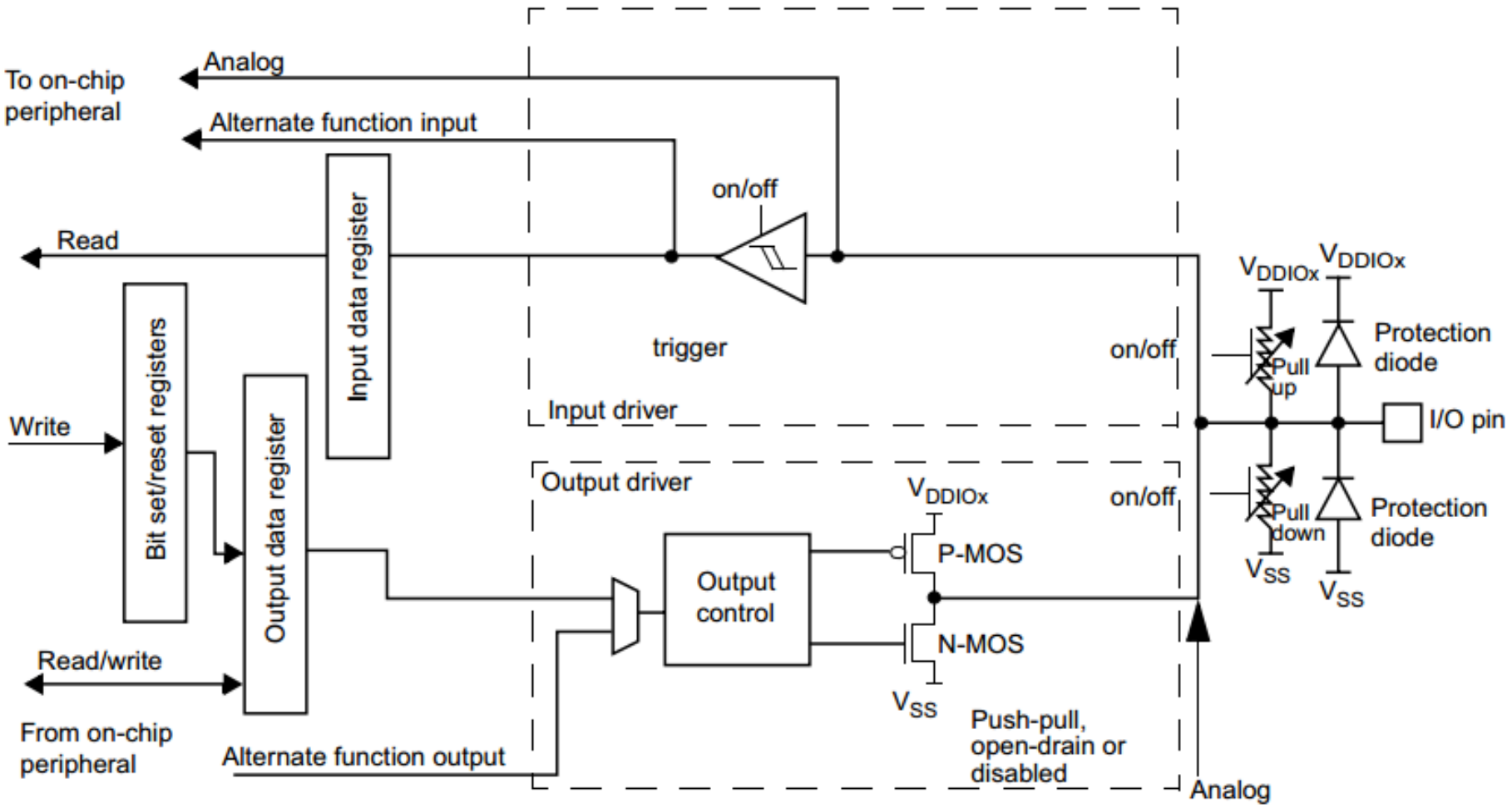
AudioPlayer optimization summary

V_{DD}=3.0V @25°C

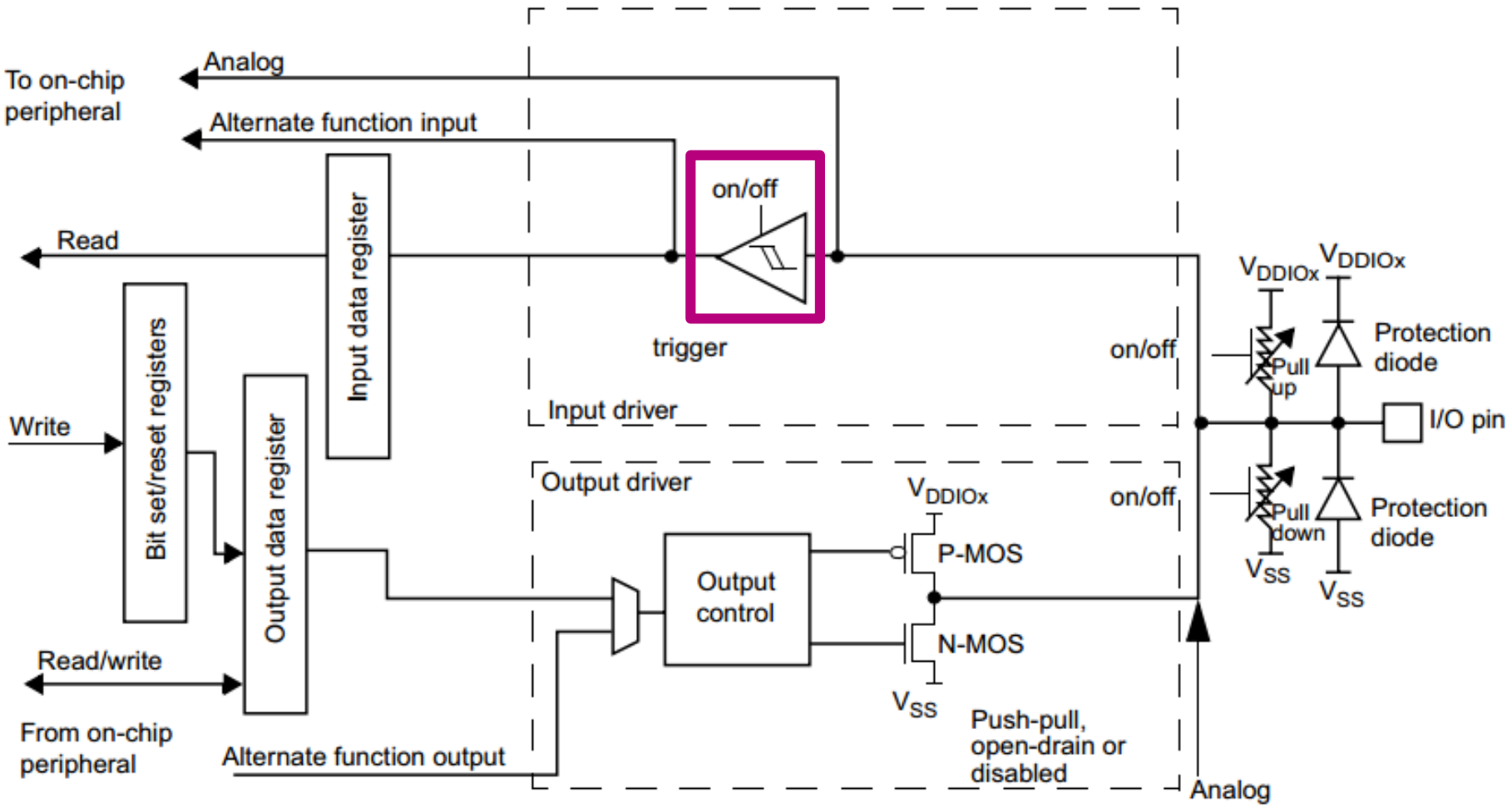
Optimization applied	STOPPED		PLAYING	
	I_{DD}	Difference	I_{DD}	Difference
Original state (STEP6)	17.3 mA	-	17.8 mA	-
Tickless Mode	10.2 mA	-41%	11.9 mA	-33.1%
Clocks gating	8.5 mA	-16.7%	10.6 mA	-10.9%

Just the MCU consumption considered
(including circuitries supplied from GPIOs)

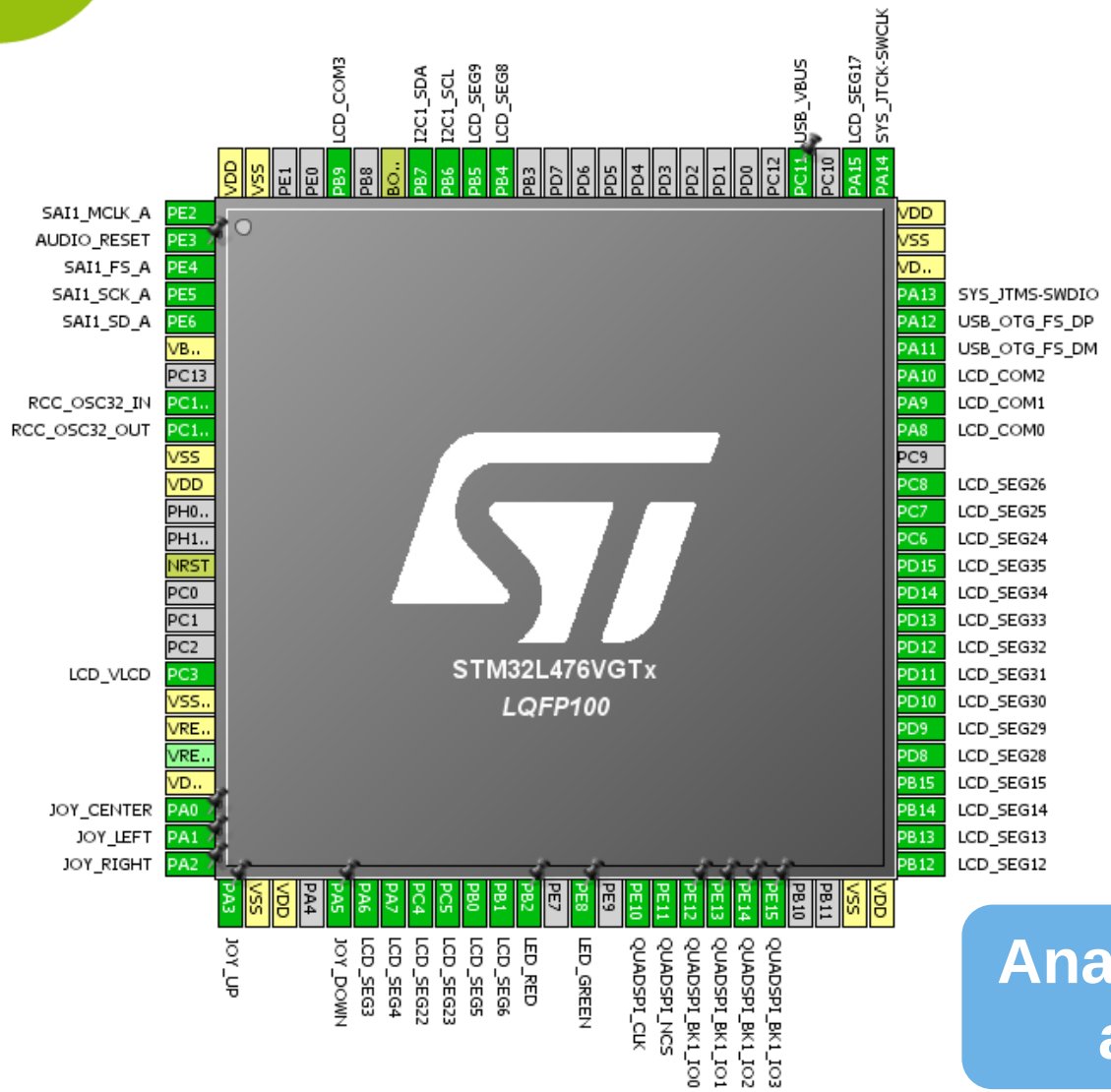
GPIOs configuration



GPIOs configuration



Unused GPIOs



9.4.1 GPIO port mode register (GPIOx_MODER) (x =A..H)

Address offset:0x00

Reset values:

- 0xABFF FFFF for port A
- 0xFFFF FEBF for port B
- 0xFFFF FFFF for ports C..G,
- 0x0000 000F for port H

31	30
MODE15[1:0]	
rw	rw

3	2	1	0
MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw

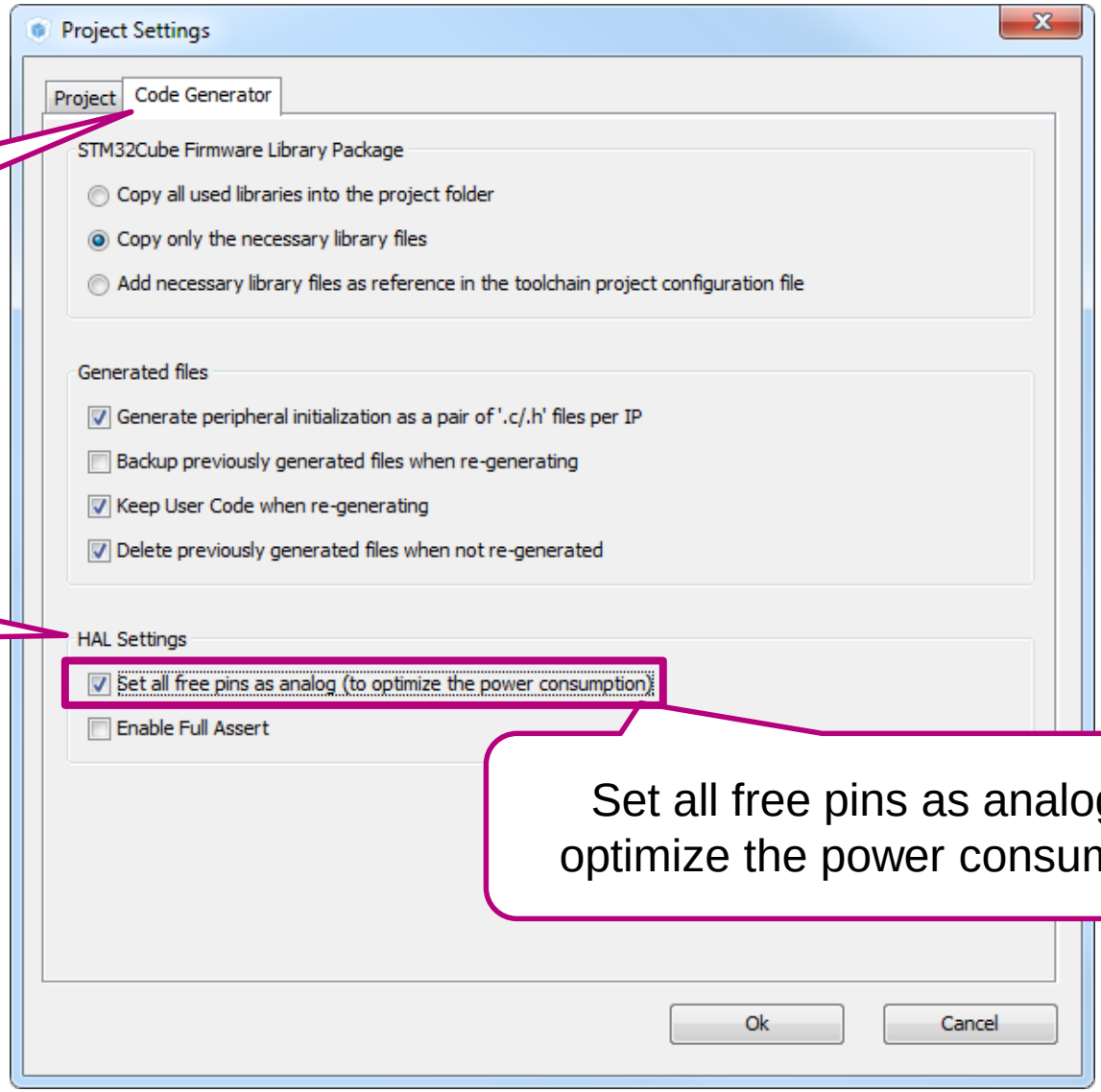
Bits 2y+1:2y **MODEy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode (reset state)

Analog mode is default configuration after reset for most of the pins

life.augmented

Feature of STM32CubeMX

STM32CubeMX
→ Project Settings
→ Code Generator

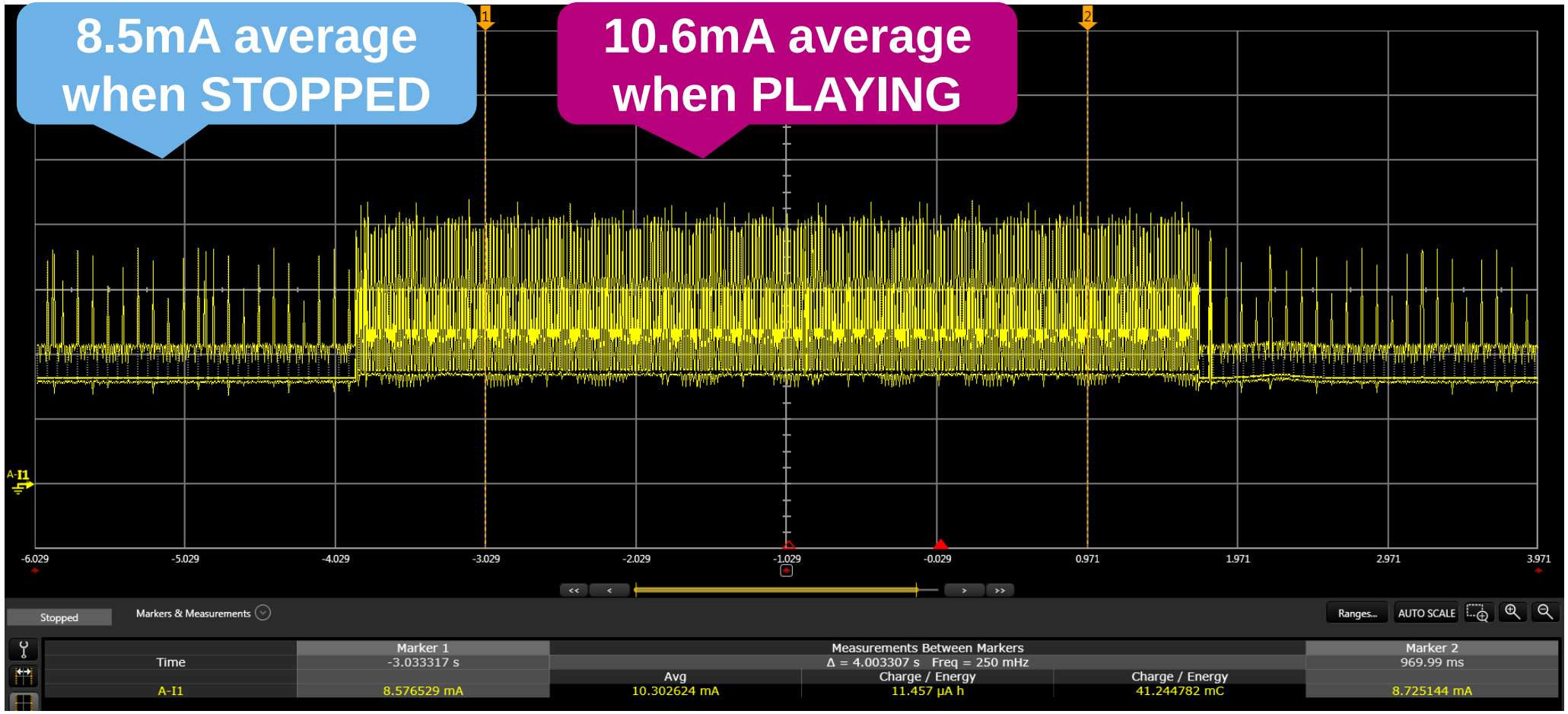


HAL Settings

Set all free pins as analog (to optimize the power consumption

AudioPlayer optimization 3/5

Unused GPIOs handled



AudioPlayer optimization summary

V_{DD}=3.0V @25°C

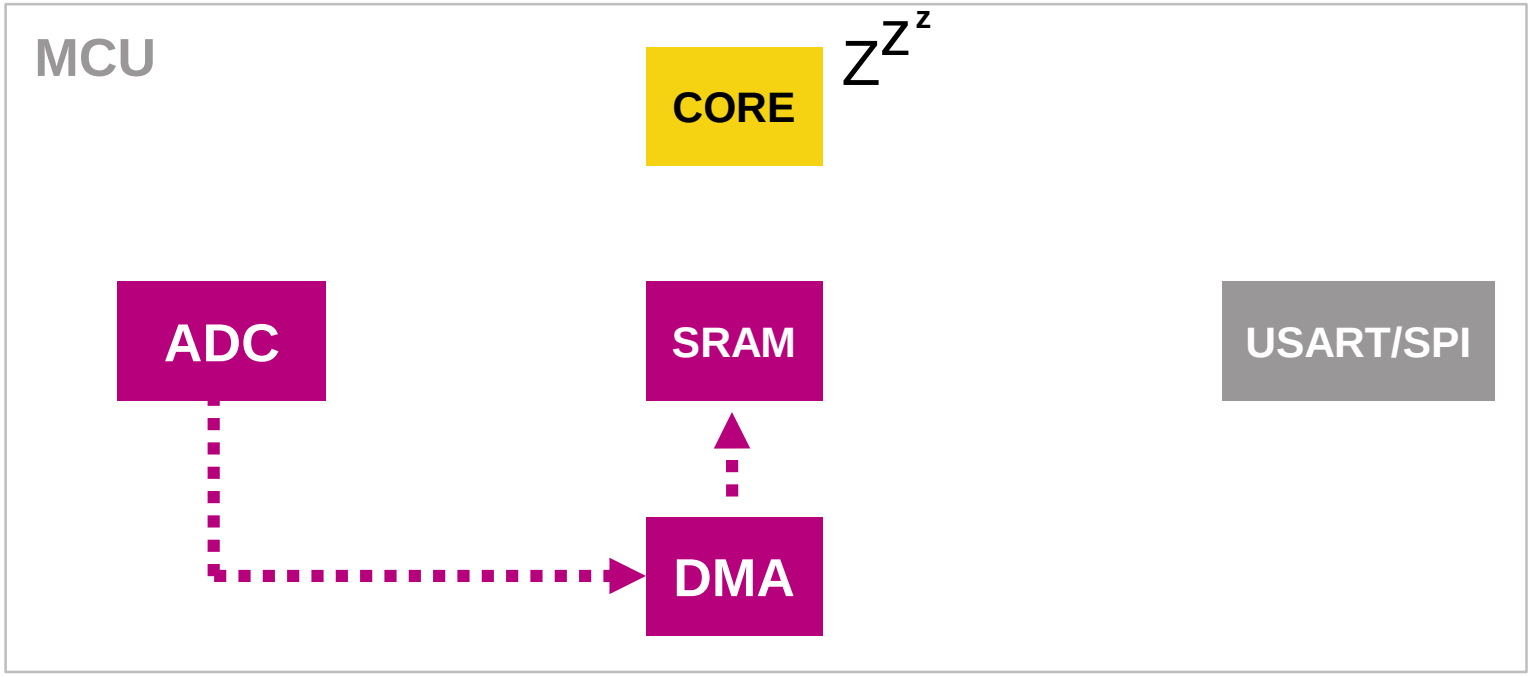
Optimization applied	STOPPED		PLAYING	
	I_{DD}	Difference	I_{DD}	Difference
Original state (STEP6)	17.3 mA	-	17.8 mA	-
Tickless Mode	10.2 mA	-41%	11.9 mA	-33.1%
Clocks gating	8.5 mA	-16.7%	10.6 mA	-10.9%
Unused GPIOs	8.5 mA	~0	10.6 mA	~0

Just the MCU consumption considered
(including circuitries supplied from GPIOs)

Let the peripherals do the job

Scenario #1

1
SAMPLING

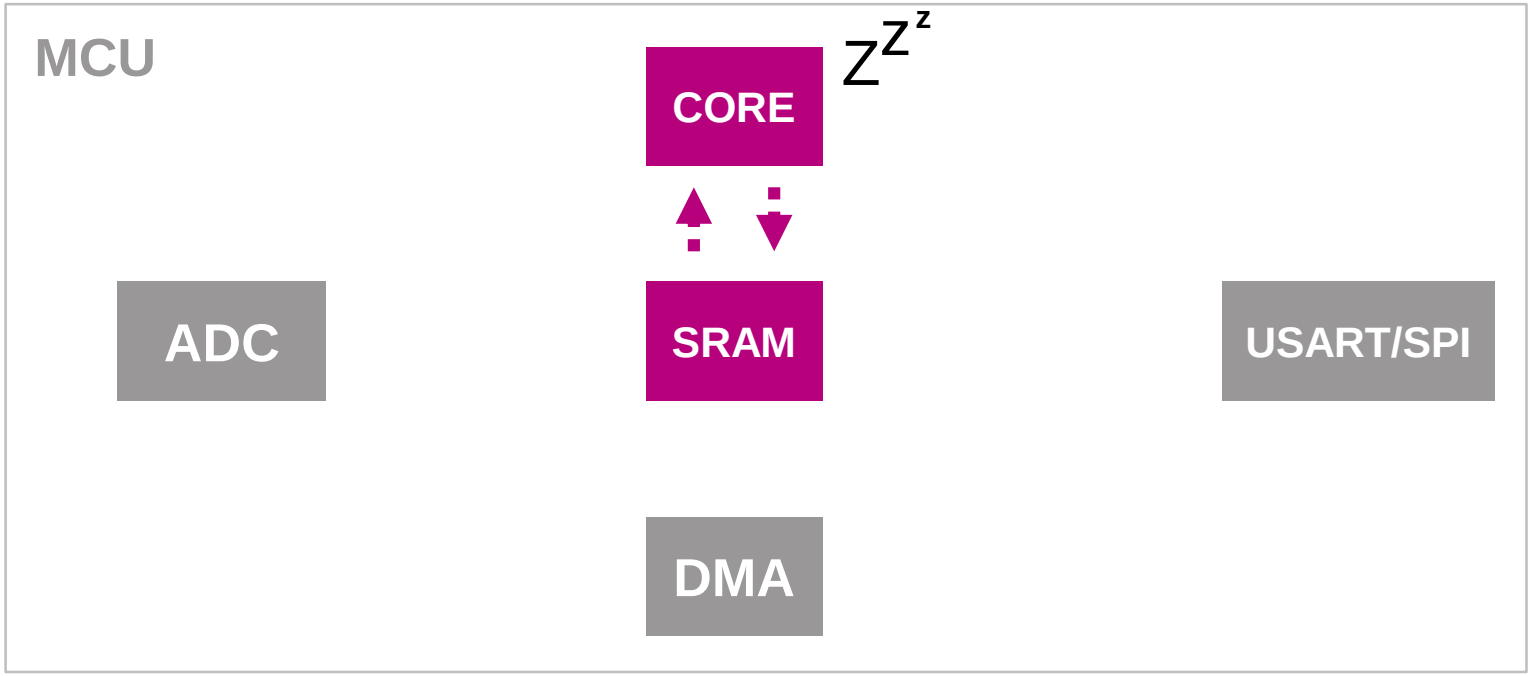


Let the peripherals do the job

Scenario #1

1
SAMPLING

2
PROCESSING



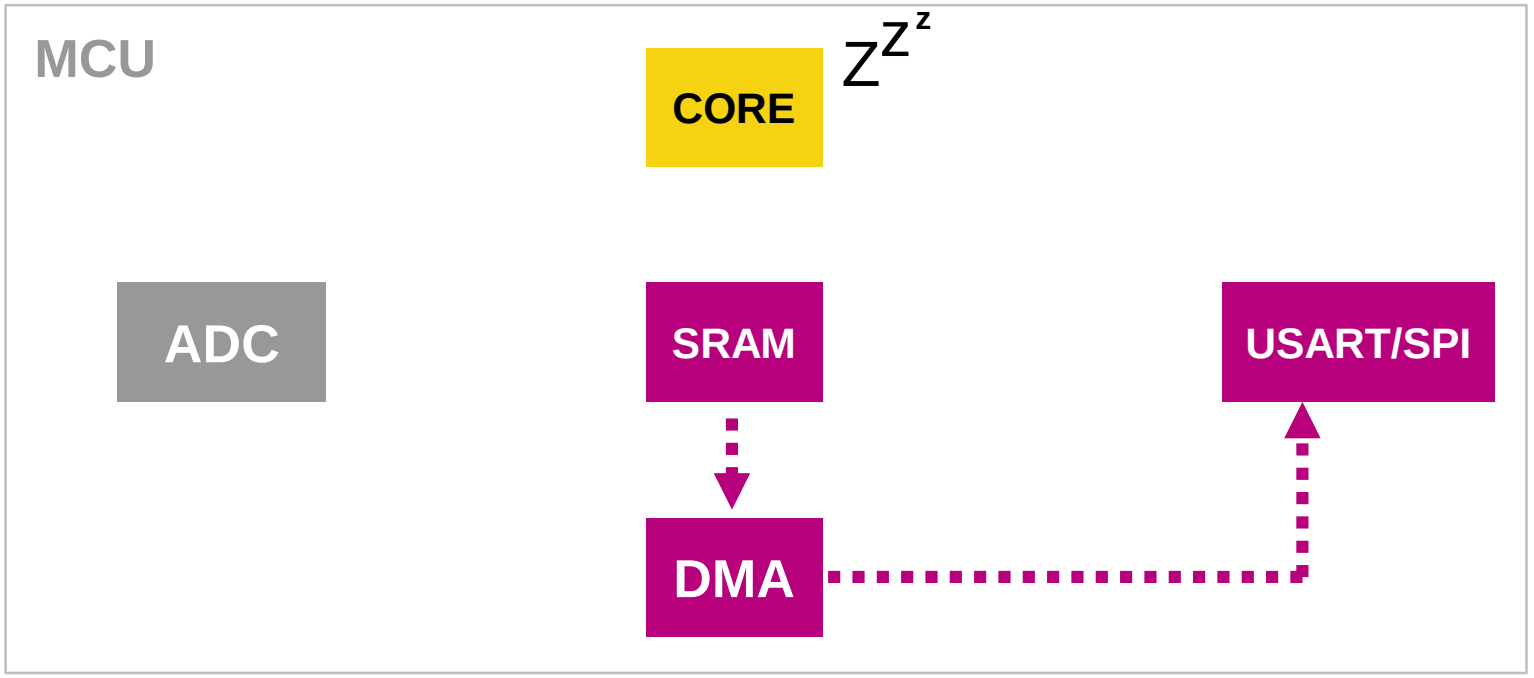
Let the peripherals do the job

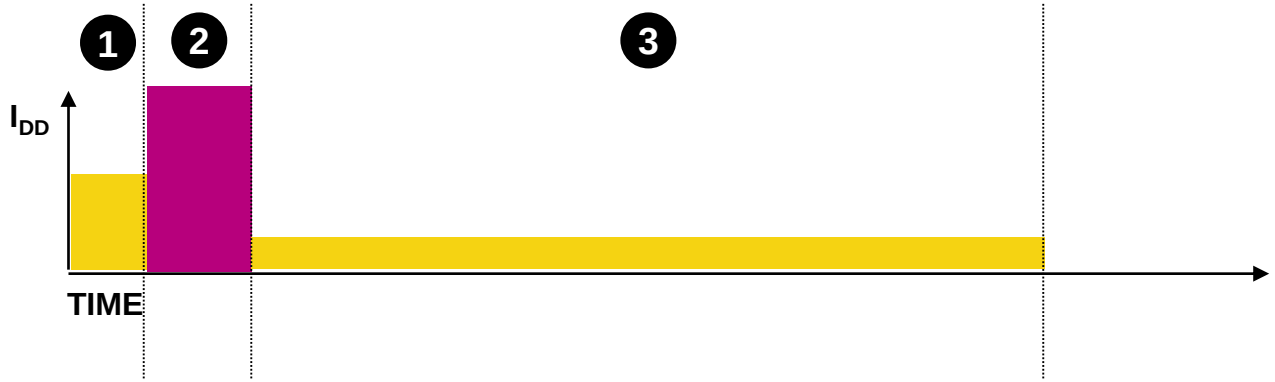
Scenario #1

1 SAMPLING

2 PROCESSING

3 SENDING

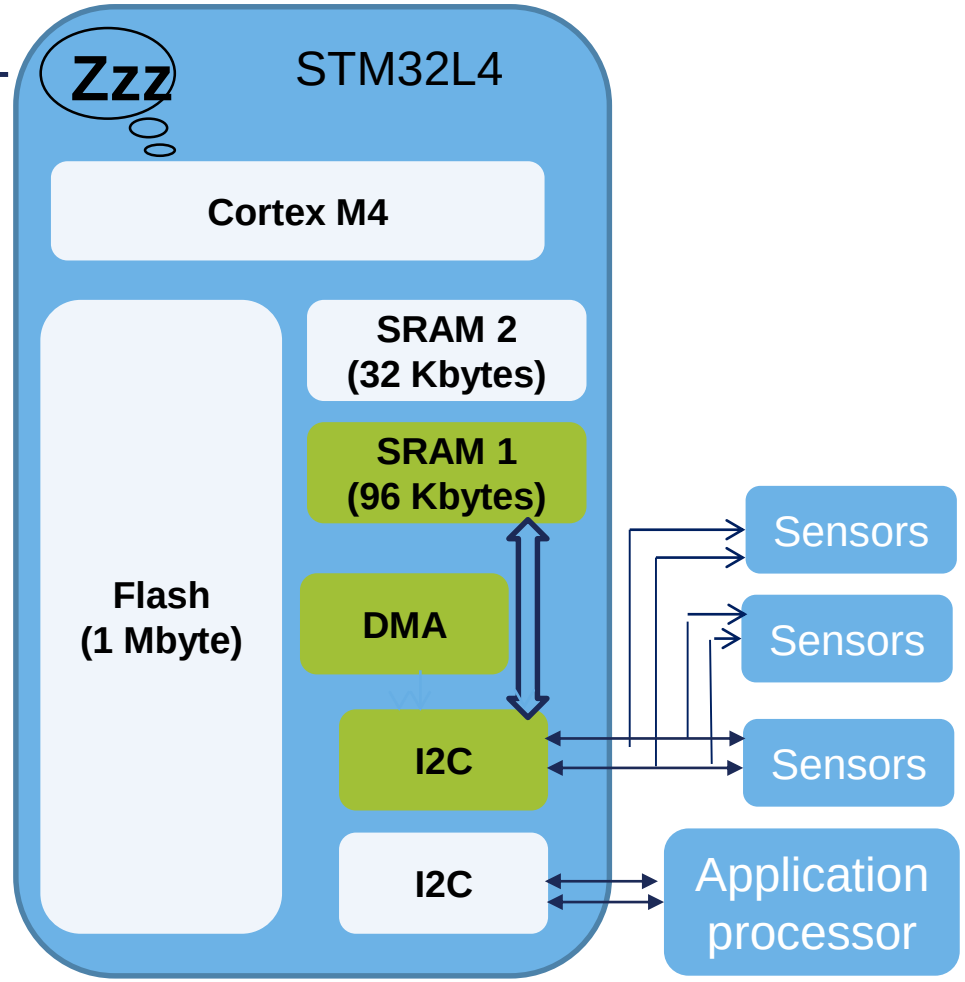




Batch Acquisition mode (BAM)

Optimized mode for transferring data with communication peripherals, while the rest of the device is in low power.

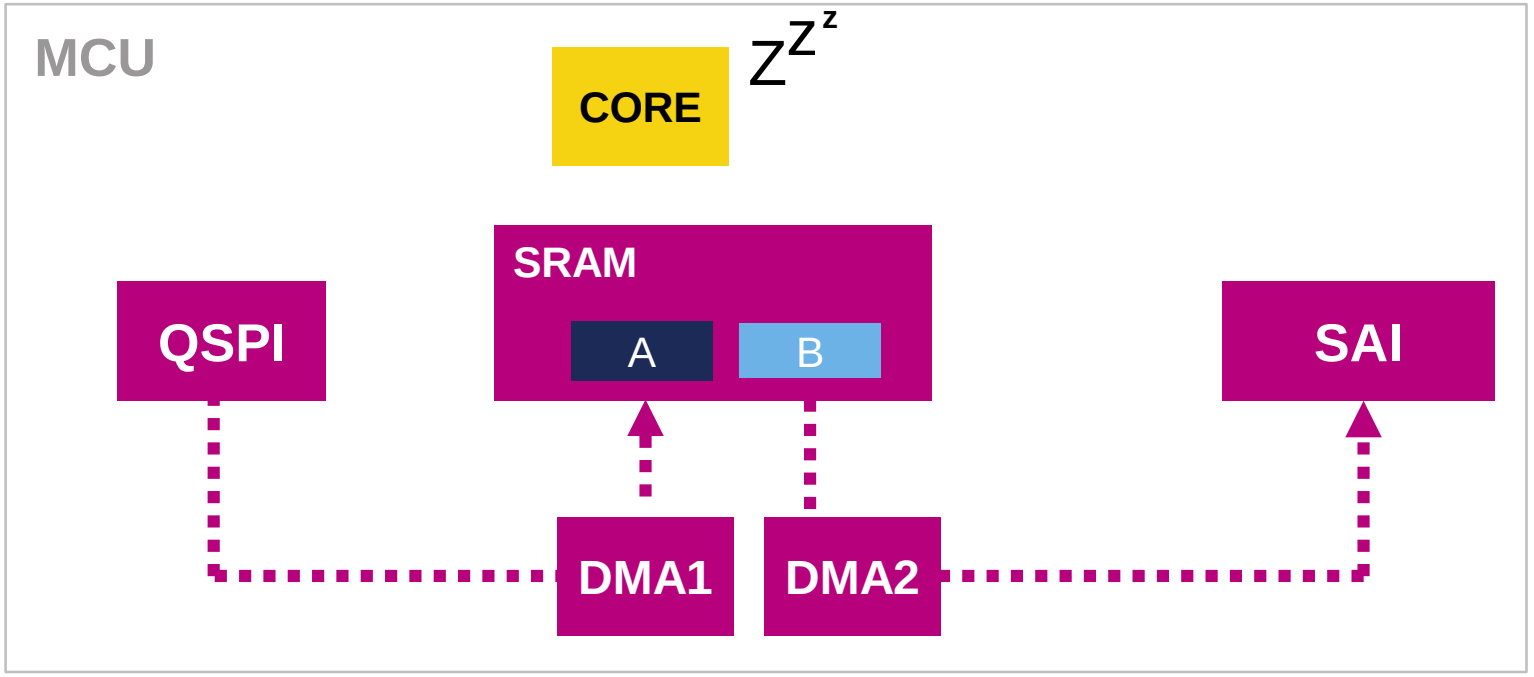
1. Only the needed communication peripheral + 1 DMA + 1 SRAM are configured with clock enabled in Sleep mode
2. Flash memory is put in Power-down mode and Flash clock is gated off during Sleep mode
3. Enter either Sleep or Low-power sleep mode
   - Note that the I2C clock can be at 16 MHz even in Low-power sleep mode, allowing 1 MHz Fast-mode Plus support. U(S)ART/LPUART clock can also be HSI.

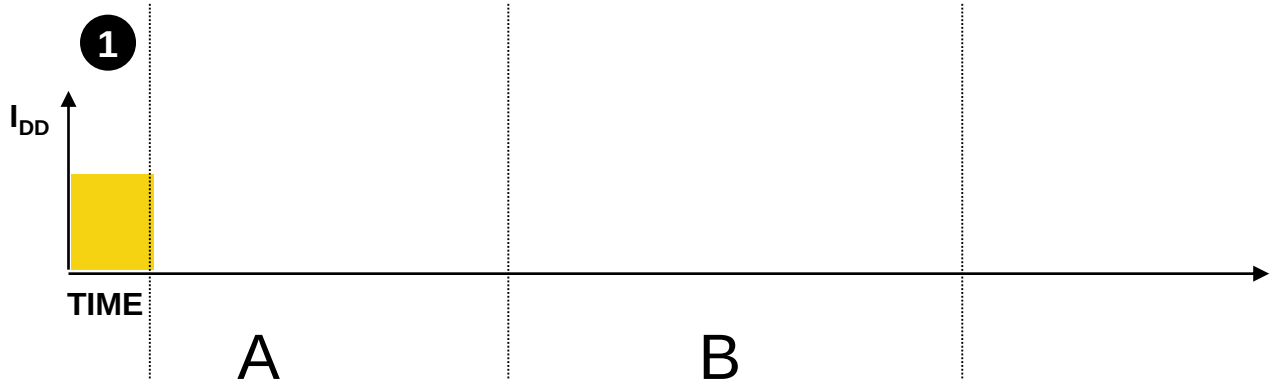


Let the peripherals do the job

Scenario #2

1
Reading new data



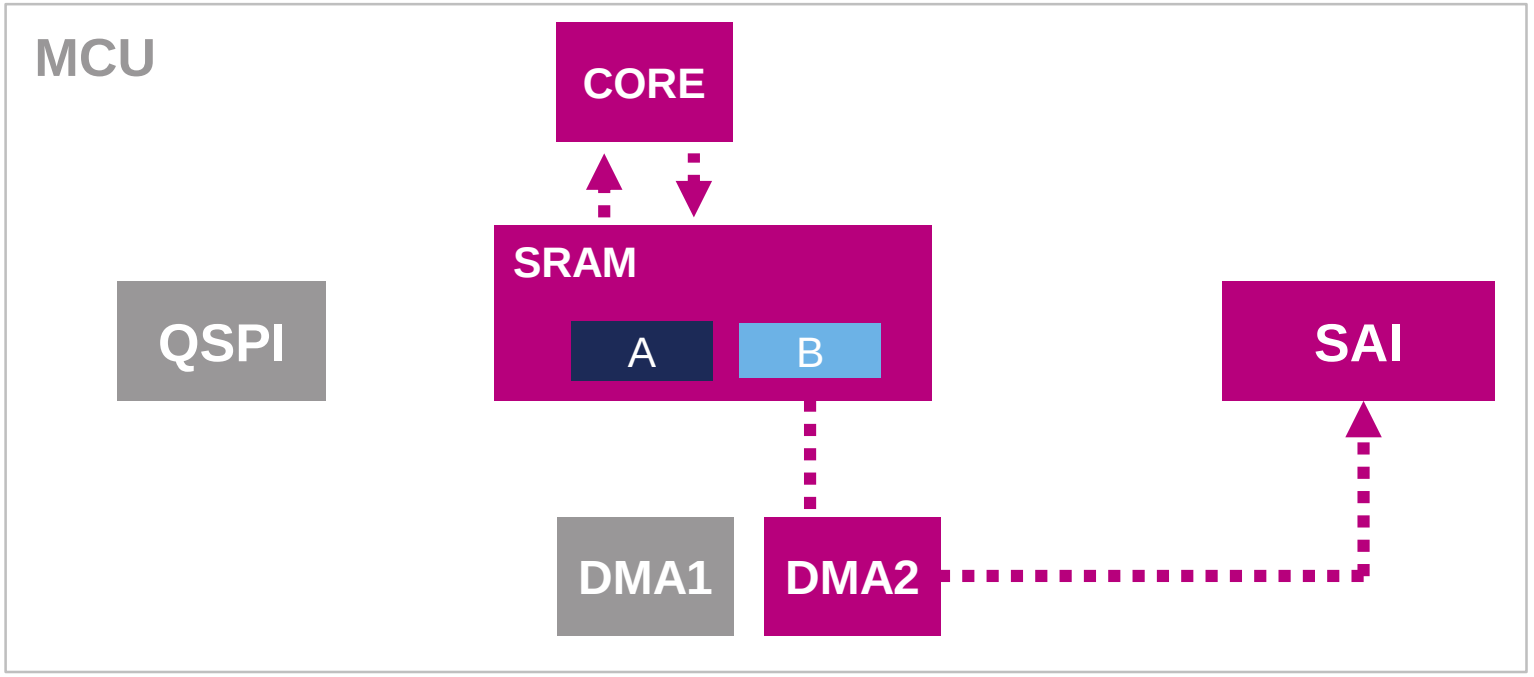


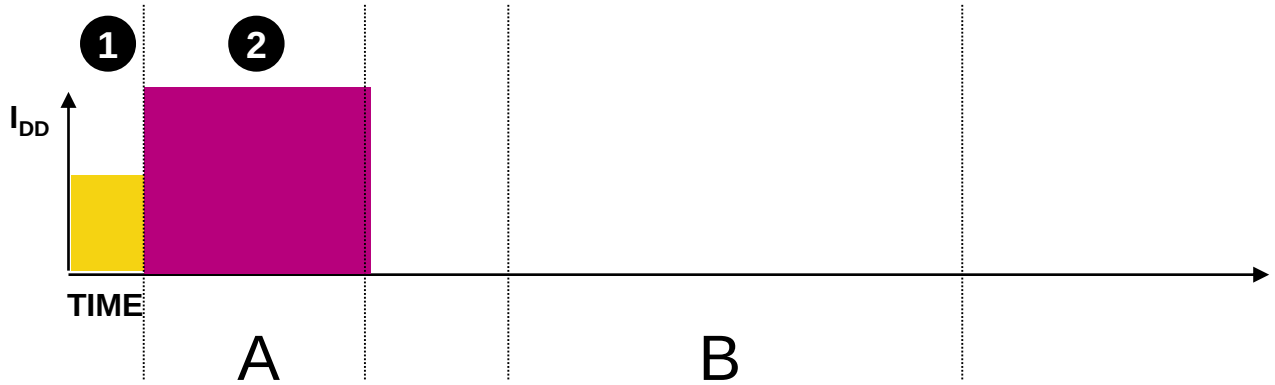
Let the peripherals do the job

Scenario #2

1
Reading new data

2
Processing





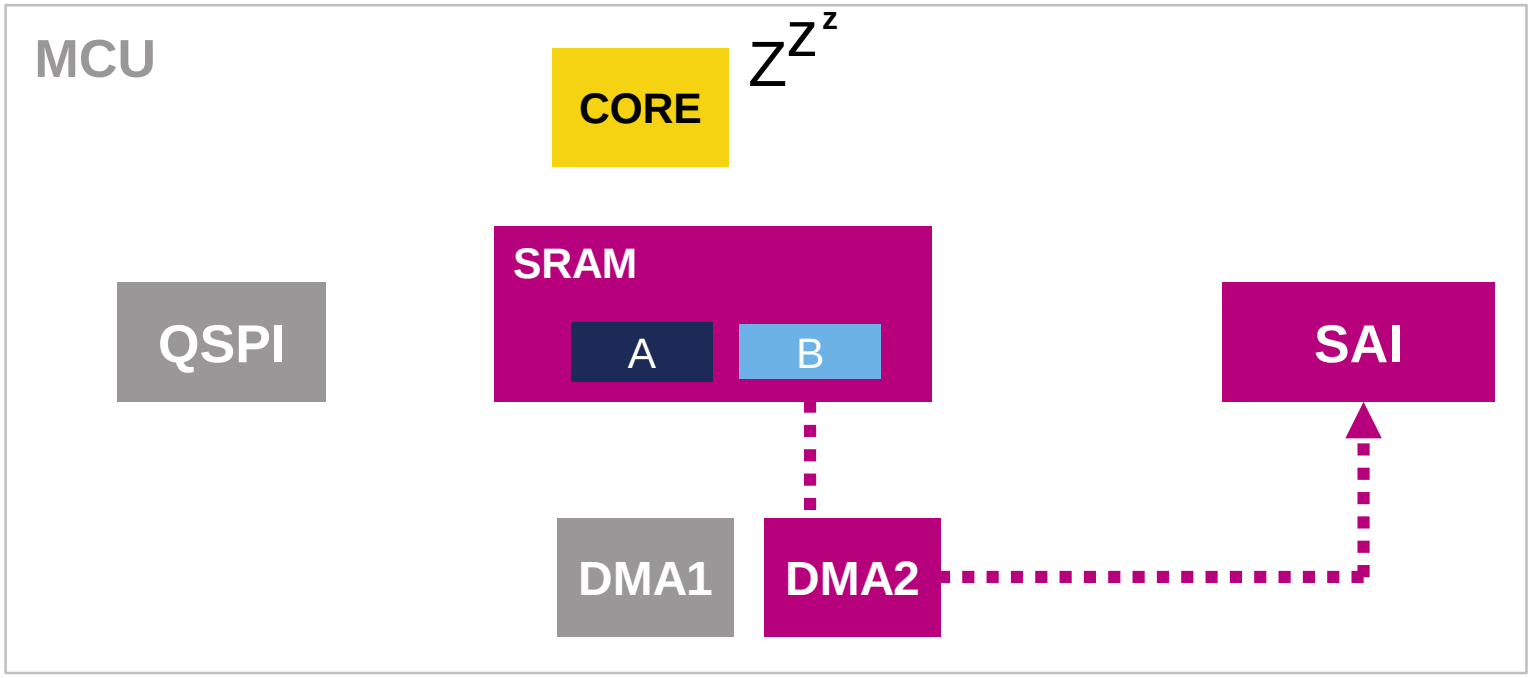
Let the peripherals do the job

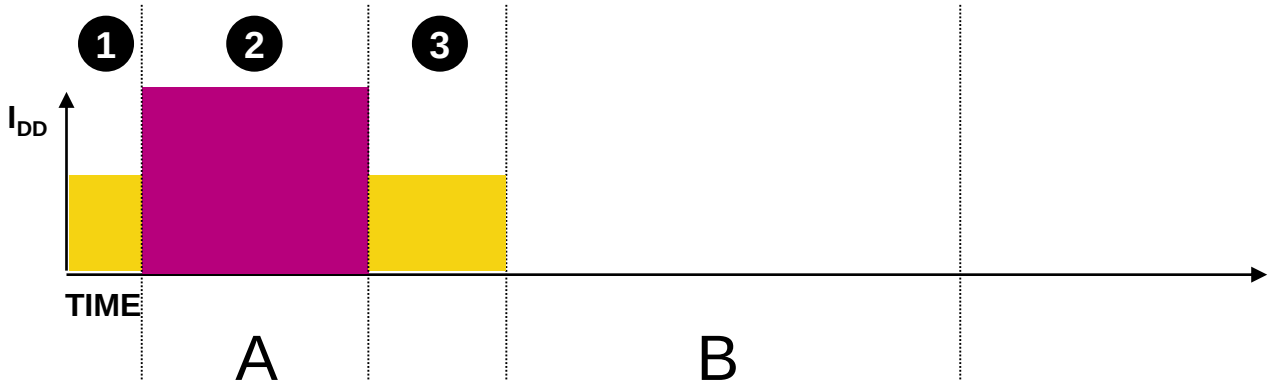
Scenario #2

1 Reading new data

2 Processing

2 Idling



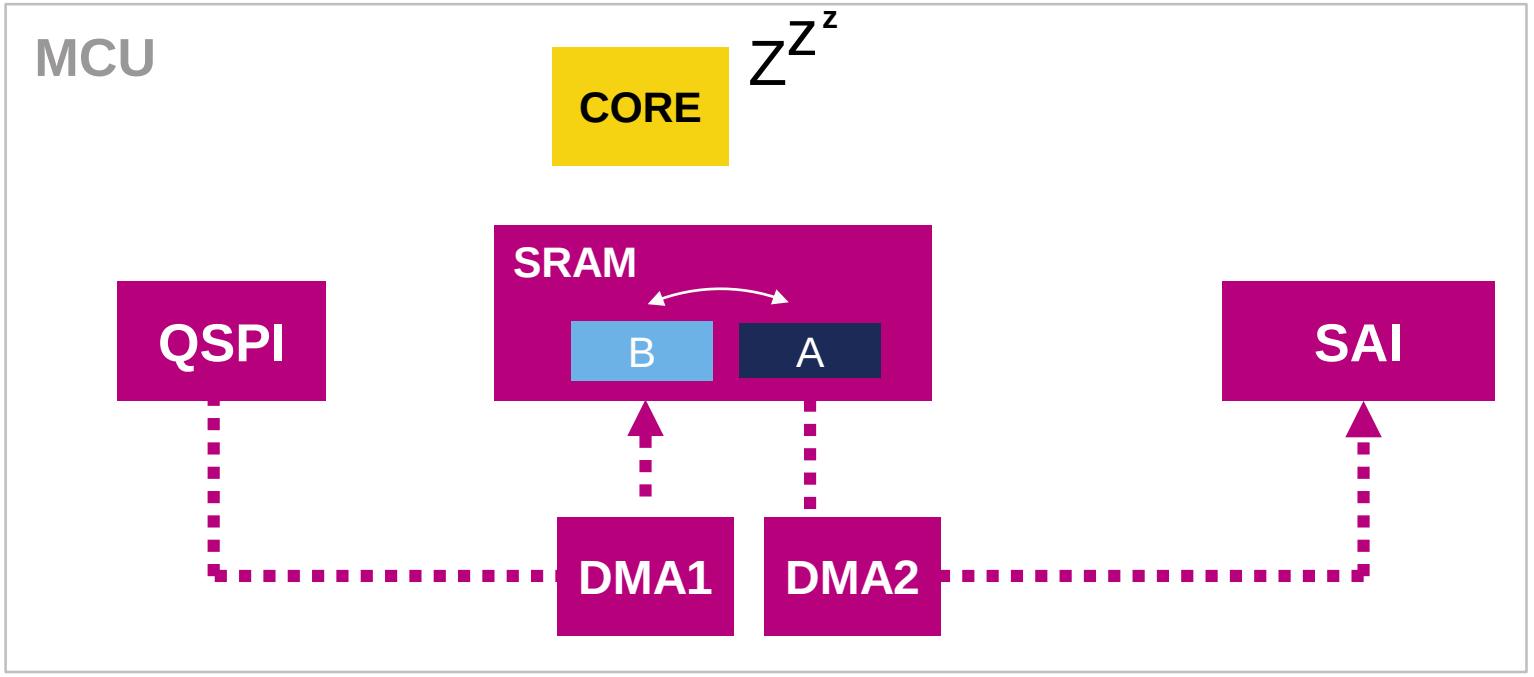


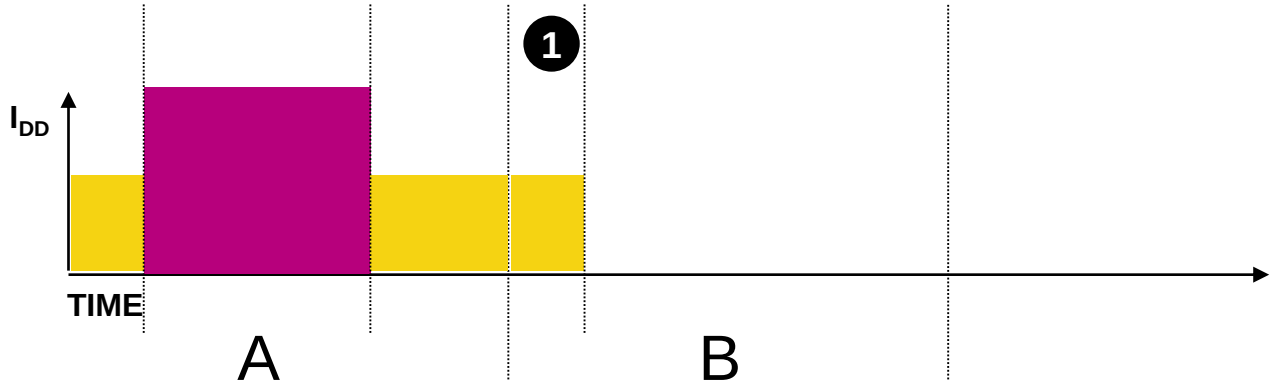
life.augmented

Let the peripherals do the job

Scenario #2

1
Reading new data



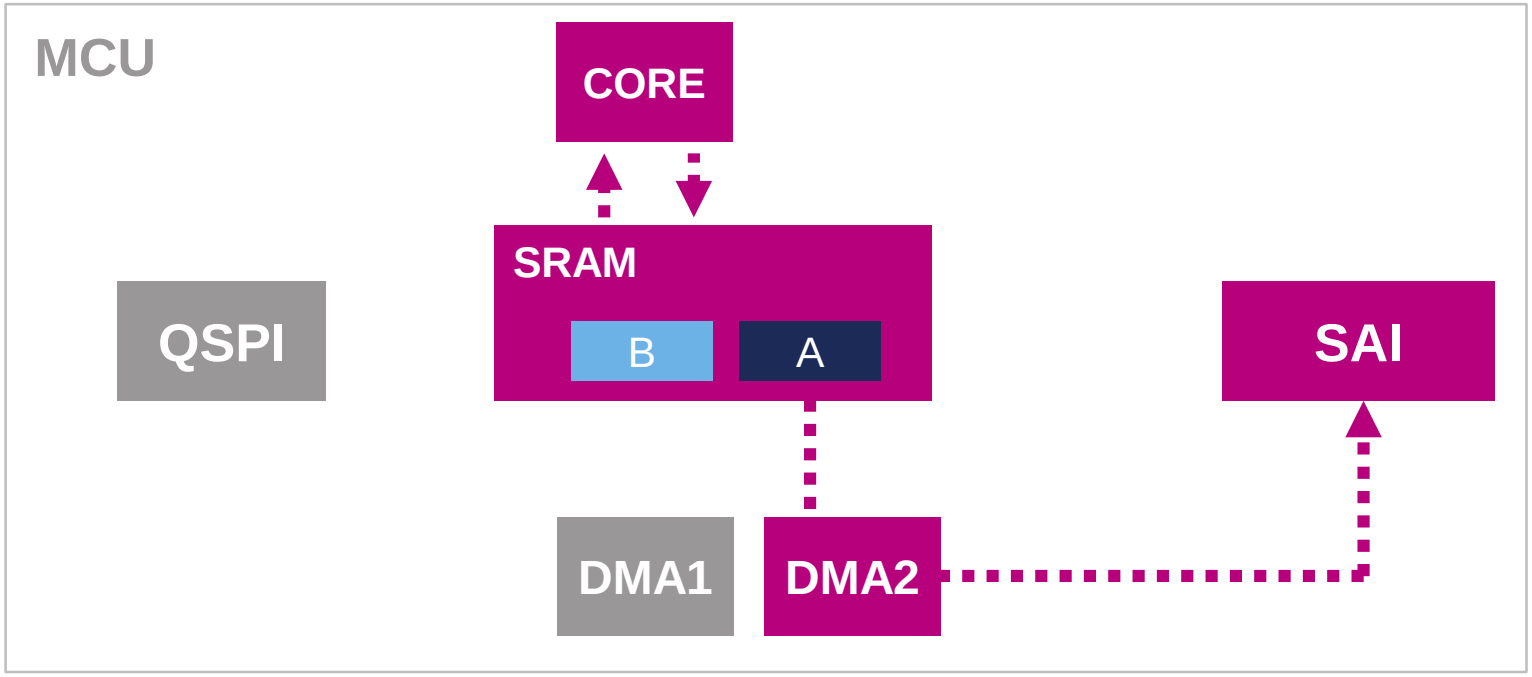


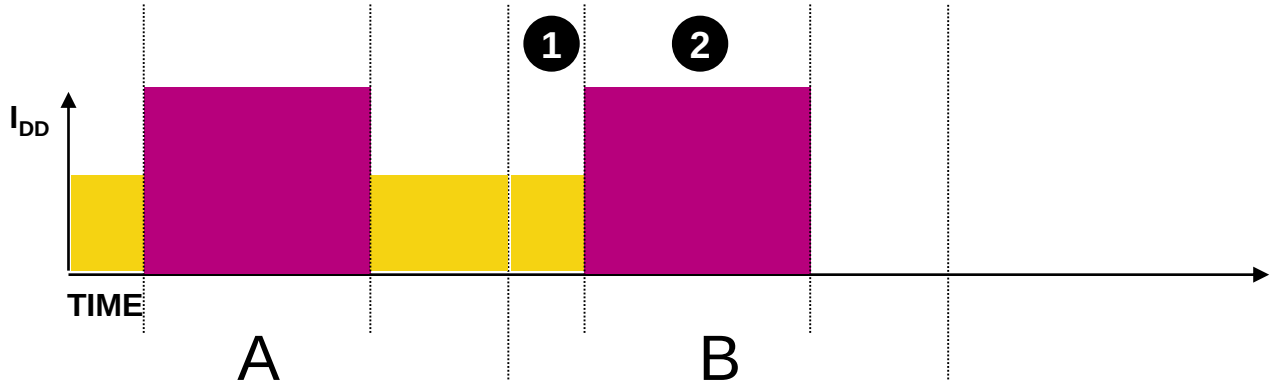
Let the peripherals do the job

Scenario #2

1
Reading new data

2
Processing





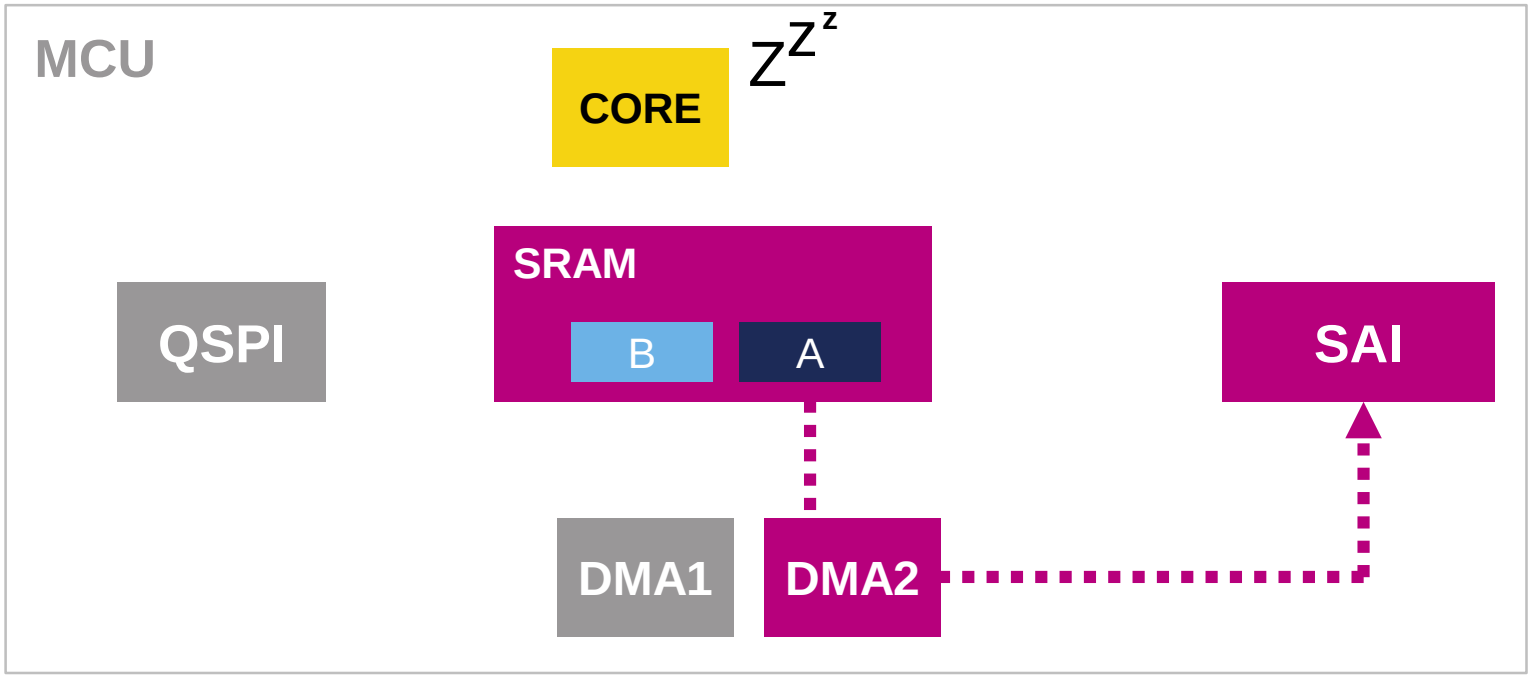
Let the peripherals do the job

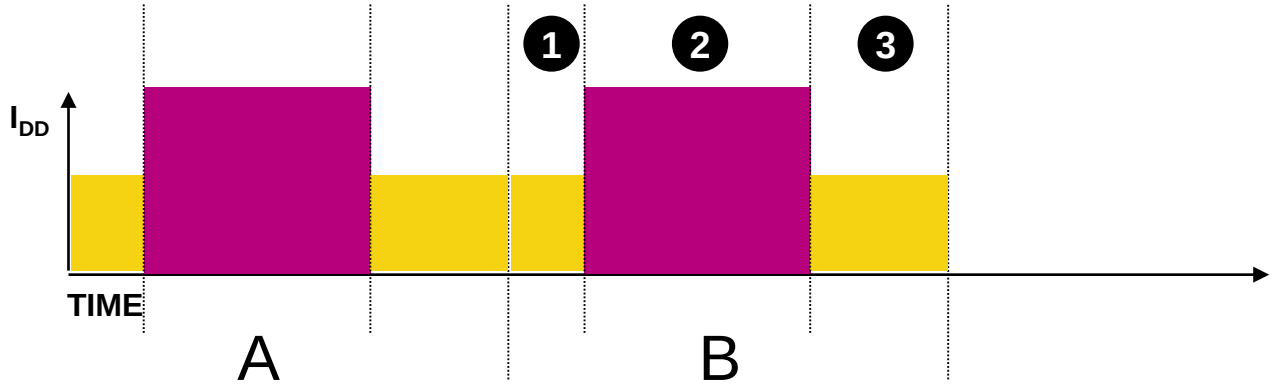
Scenario #2

1 Reading new data

2 Processing

3 Idling

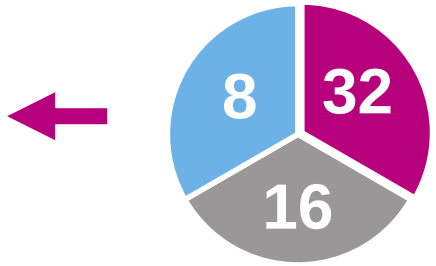




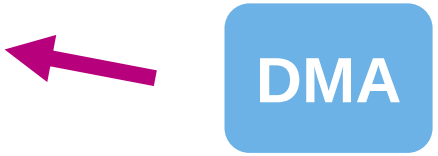
life.augmented

Squeeze the Maximum

1. From the architecture

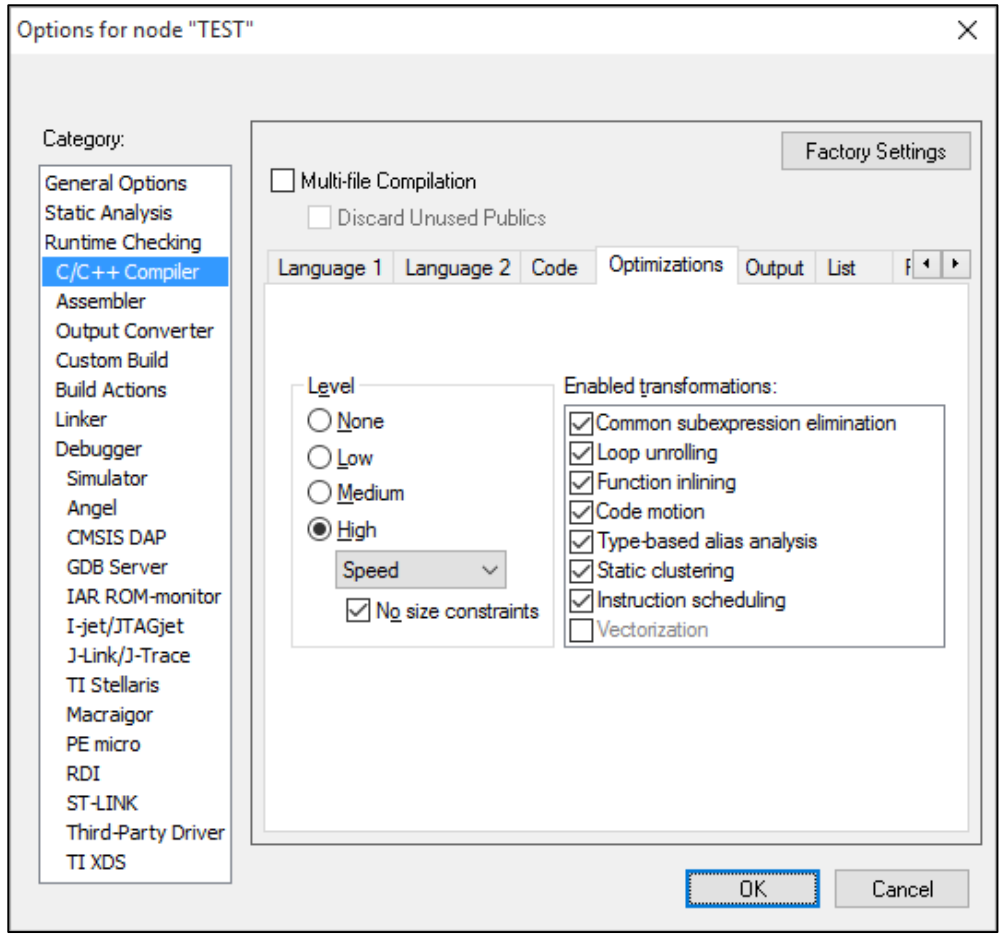


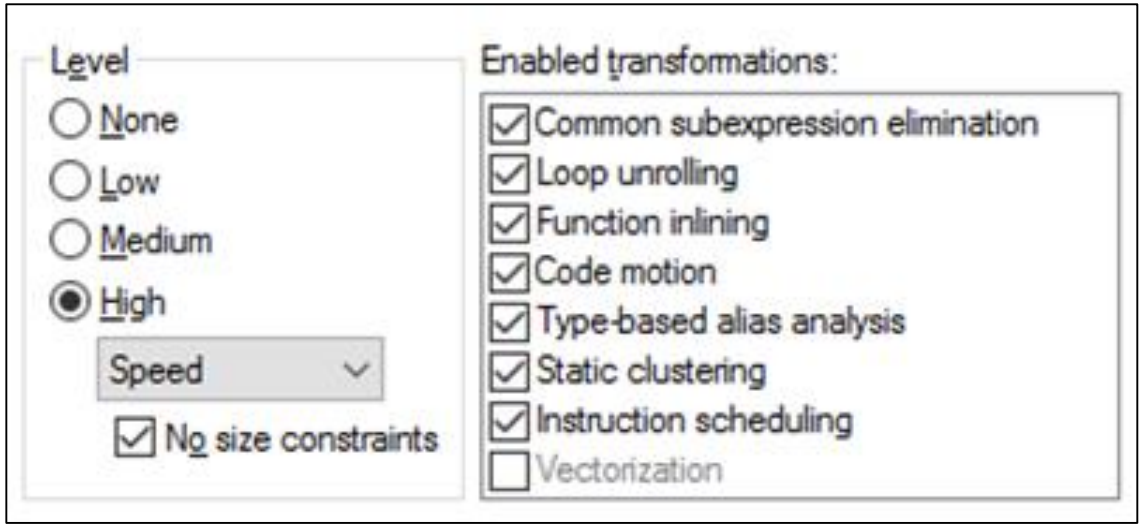
2. From the peripherals



3. From the compiler

IAR optimization options





GCC optimizations options

`-O1`

Optimize. Optimizing compilation takes somewh

With `-O`, the compiler tries to reduce code size an

`-O` turns on the following optimization flags:

```
-fauto-inc-dec
-fbranch-count-reg
-fcombine-stack-adjustments
-fcompare-elim
-fcprop-registers
-fdce
-fdefer-pop
-fdelayed-branch
-fdse
-fforward-propagate
-fguess-branch-probability
-fif-conversion2
-fif-conversion
-finline-functions-called-once
-fipa-pure-const
-fipa-profile
-fipa-reference
-fmerge-constants
-fmove-loop-invariants
-freorder-blocks
-fshrink-wrap
-fsplit-wide-types
-ftree-bit-ccp
-ftree-ccp
-fssa-phiopt
-ftree-ch
-ftree-coalesce-vars
-ftree-copy-prop
-ftree-dce
-ftree-dominator-opts
-ftree-dse
-ftree-forwprop
-ftree-fre
-ftree-phiprop
-ftree-sink
-ftree-slsr
-ftree-sra
-ftree-pta
-ftree-ter
-funit-at-a-time
```

`-O2` turns on all optimization flags specified by `-O`. It also turns on the following optimization flags:

```
-fthread-jumps
-falign-functions  -falign-jumps
-falign-loops  -falign-labels
-fcaller-saves
-fcrossjumping
-fcse-follow-jumps  -fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize -fdevirtualize-speculatively
-fexpensive-optimizations
-fgcse  -fgcse-lm
-fhoist-adjacent-loads
-finline-small-functions
-findirect-inlining
-fipa-cp
-fipa-cp-alignment
-fipa-sra
-fipa-icf
-fisolate-erroneous-paths-dereference
-flra-remat
-foptimize-sibling-calls
-foptimize-strlen
-fpartial-inlining
-fpeephole2
-freorder-blocks-algorithm=stc
-freorder-blocks-and-partition -freorder-functions
-frerun-cse-after-loop
-fsched-interblock  -fsched-spec
-fschedule-insns  -fschedule-insns2
-fstrict-aliasing -fstrict-overflow
-ftree-builtin-call-dce
-ftree-switch-conversion -ftree-tail-merge
-ftree-pre
-ftree-vrp
-fipa-ra
```

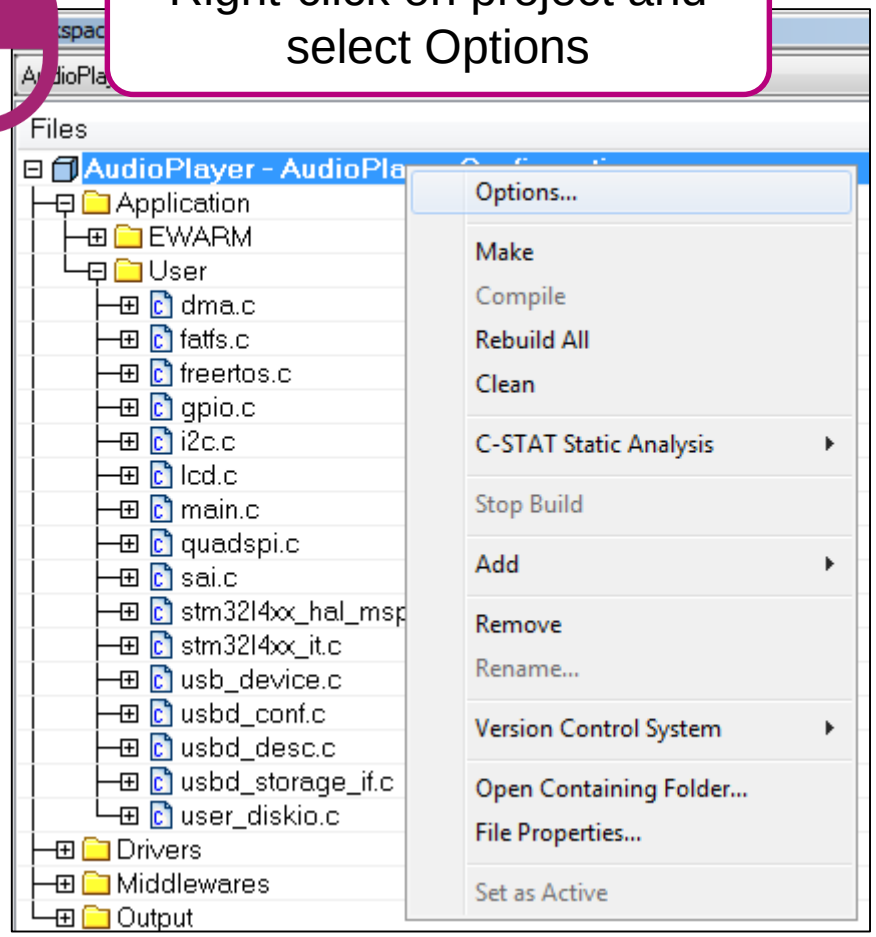
`-O3`

Optimize yet more. `-O3` turns on all optimizations specified by `-O2` and also turns on the `-finline-functions`, `-funswitch-loops`, `-fpredictive-commoning`, `-fgcse-after-reload`, `-ftree-loop-vectorize`, `-ftree-loop-distribute-patterns`, `-ftree-slp-vectorize`, `-fvect-cost-model`, `-ftree-partial-pre` and `-fipa-cp-clone` options.

http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

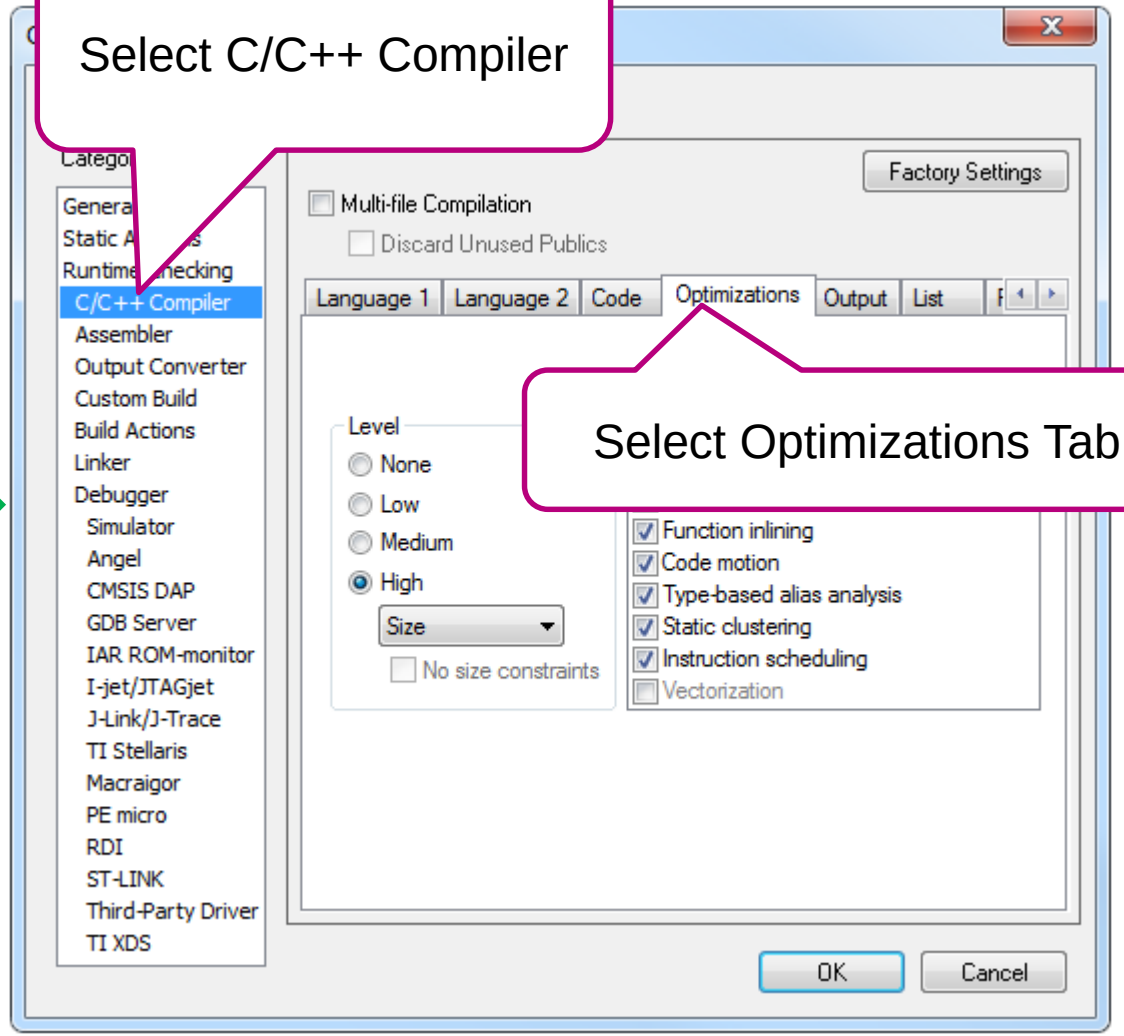
Compiler optimizations and AudioPlayer

Right-click on project and select Options



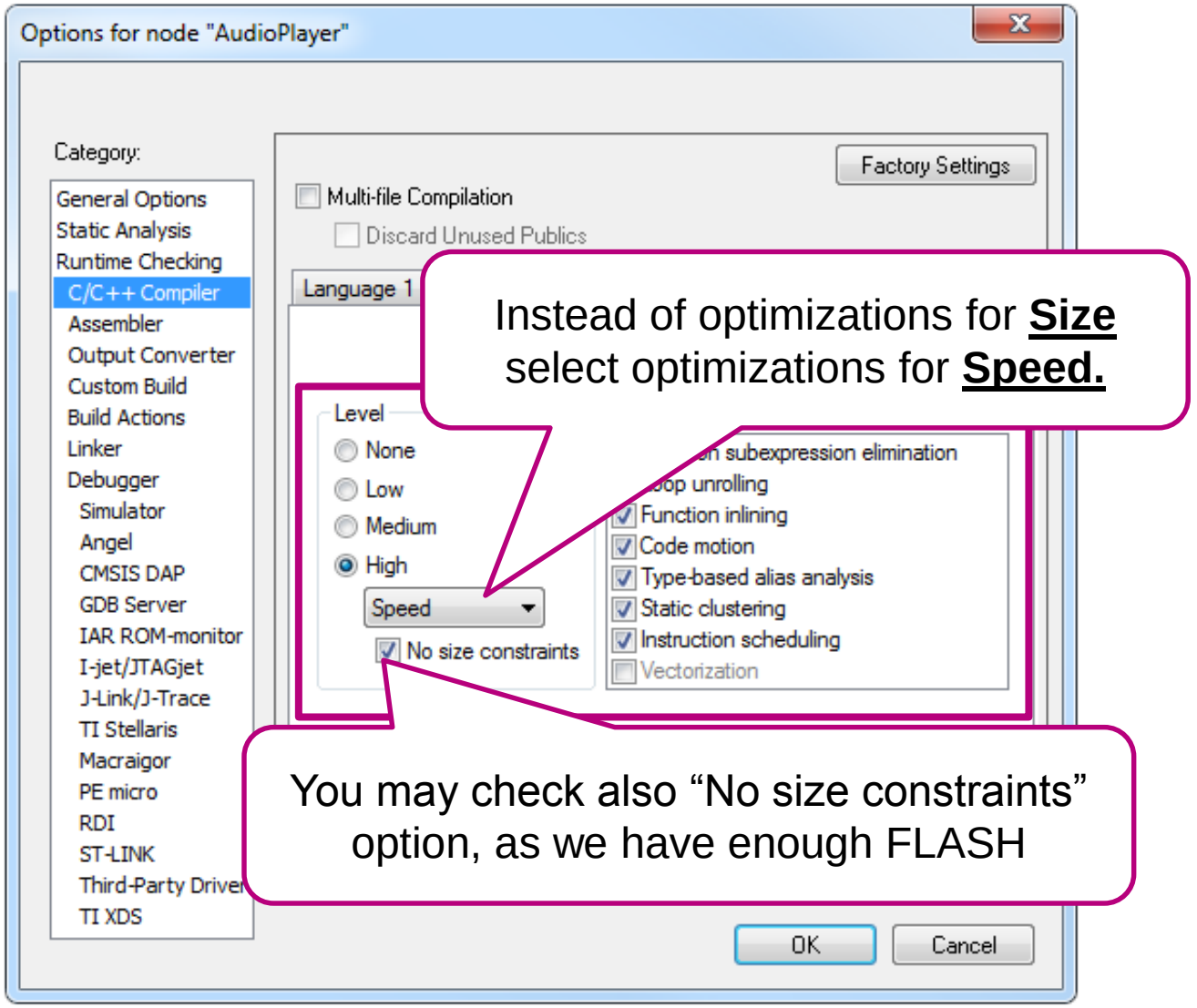
Select C/C++ Compiler

Select Optimizations Tab



life.augmented

Compiler optimizations and AudioPlayer



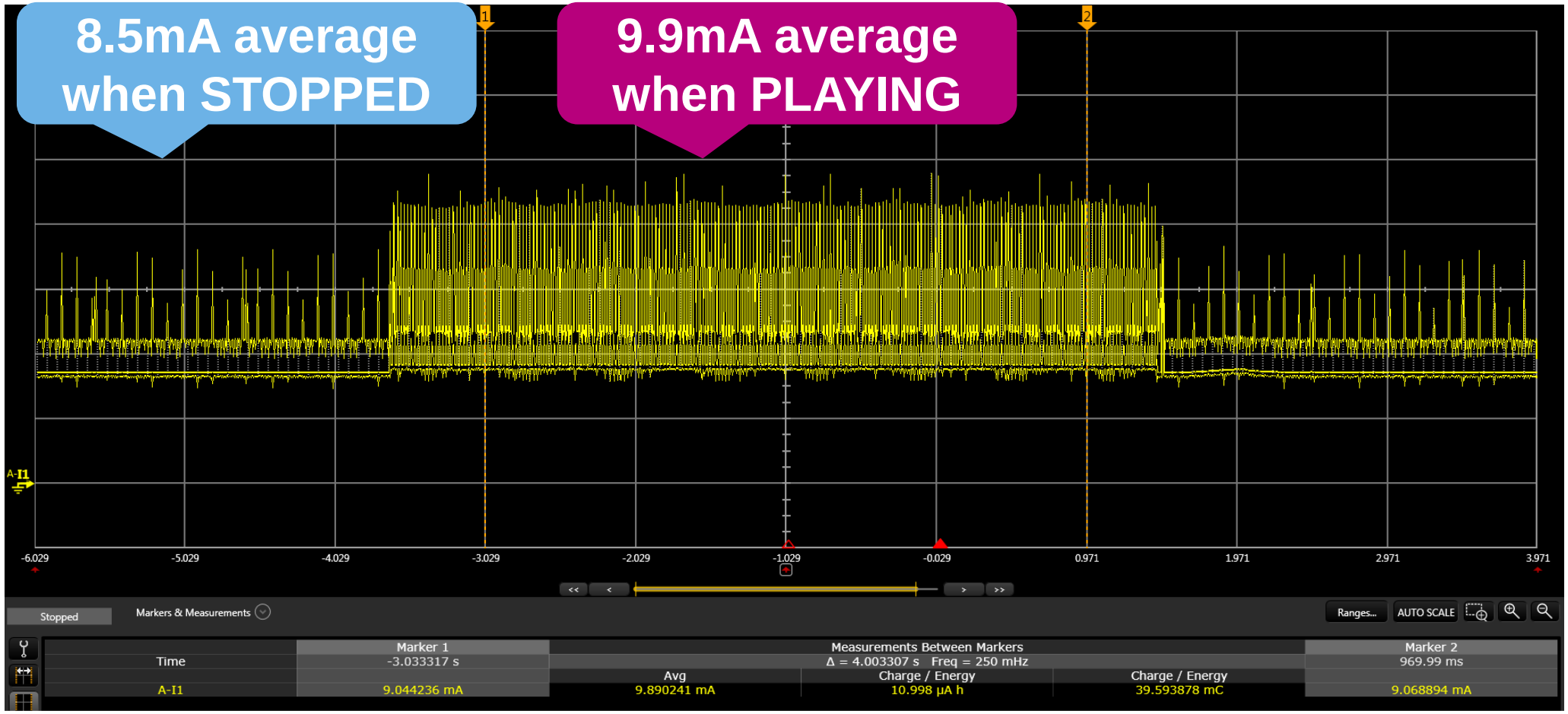
Compiler optimizations and AudioPlayer

Code size impact

optimization	code size
max speed	72 652 B
min size	43 184 B

AudioPlayer optimization 4/5

Compiler settings (optimization for speed selected)



AudioPlayer optimization summary

V_{DD}=3.0V @25°C

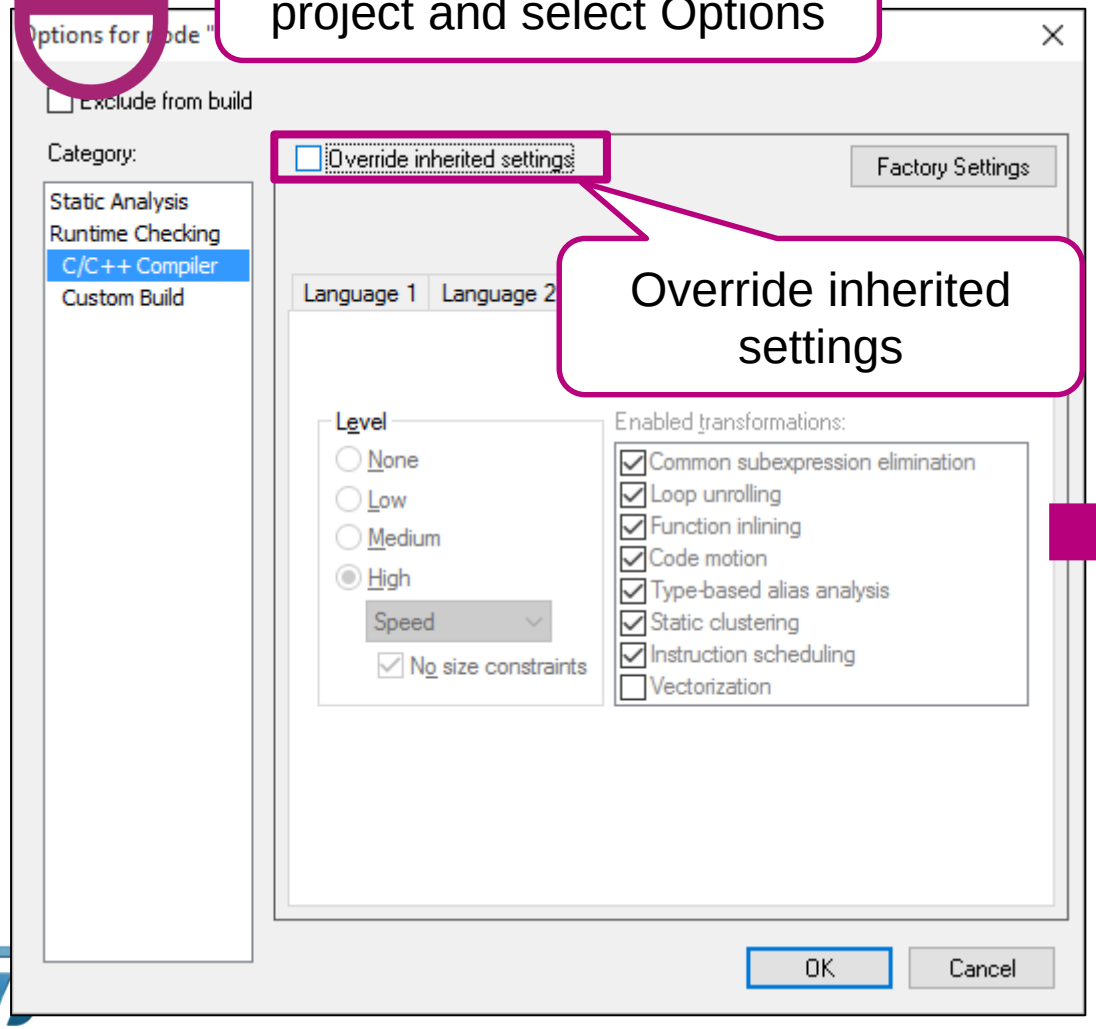
Optimization applied	STOPPED		PLAYING	
	I_{DD}	Difference	I_{DD}	Difference
Original state (STEP6)	17.3 mA	-	17.8 mA	-
Tickless Mode	10.2 mA	-41%	11.9 mA	-33.1%
Clocks gating	8.5 mA	-16.7%	10.6 mA	-10.9%
Unused GPIOs	8.5 mA	~0	10.6 mA	~0
Compiler settings	8.5 mA	~0	9.9 mA	-6.6%

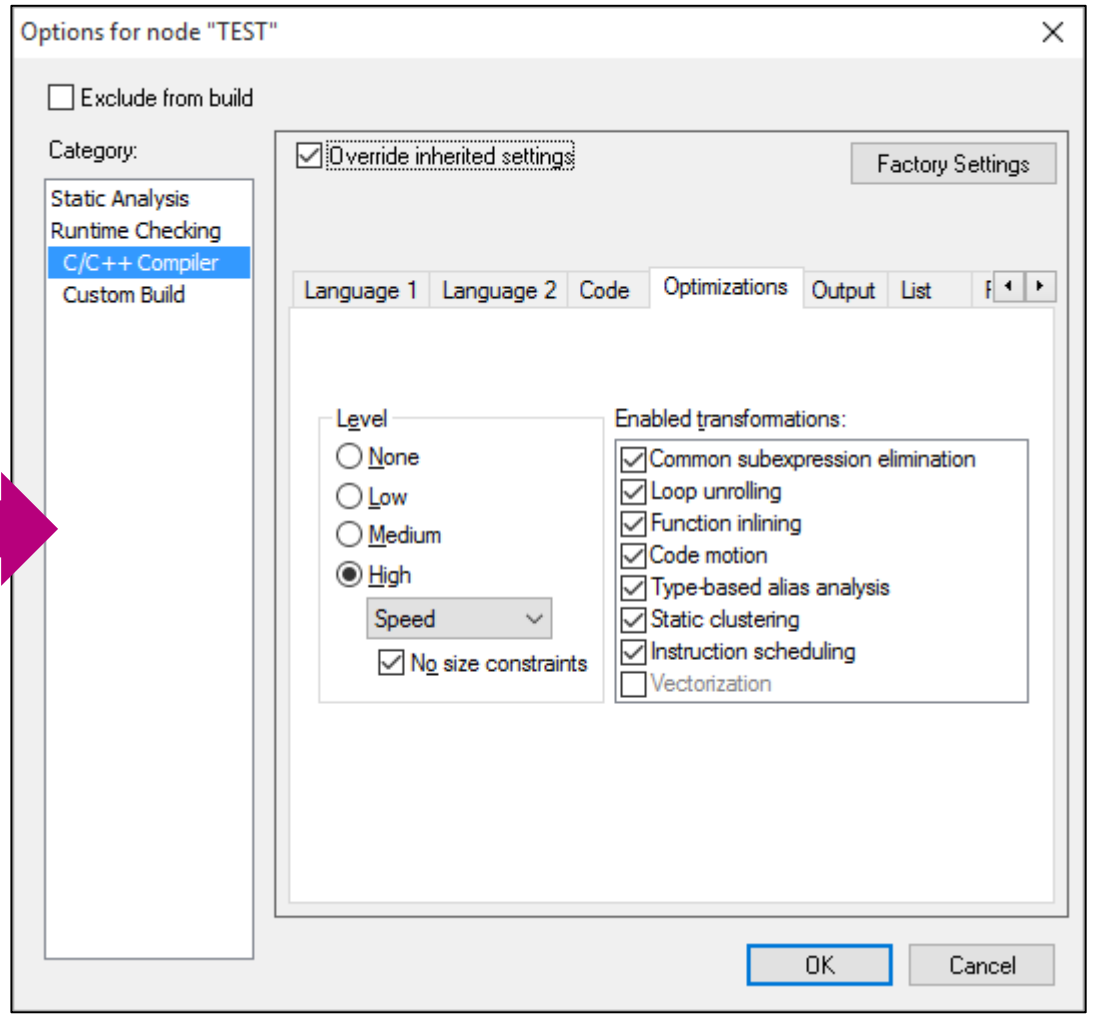
Just the MCU consumption considered
(including circuitries supplied from GPIOs)

Optimizations Level per file

IAR (EXAMPLE)

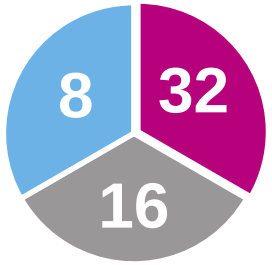
Right-click on desired file in project and select Options



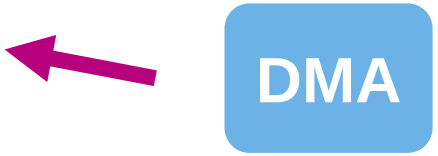


Squeeze the Maximum

1. From the architecture

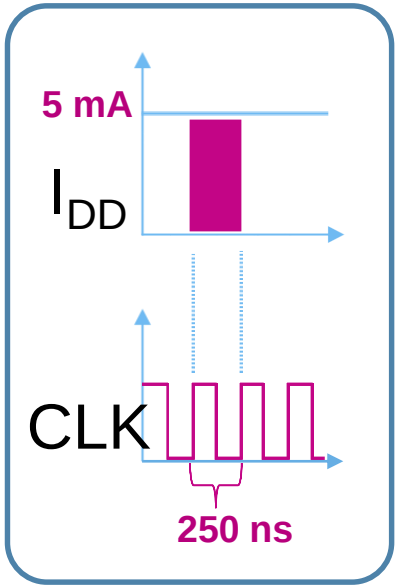


2. From the peripherals

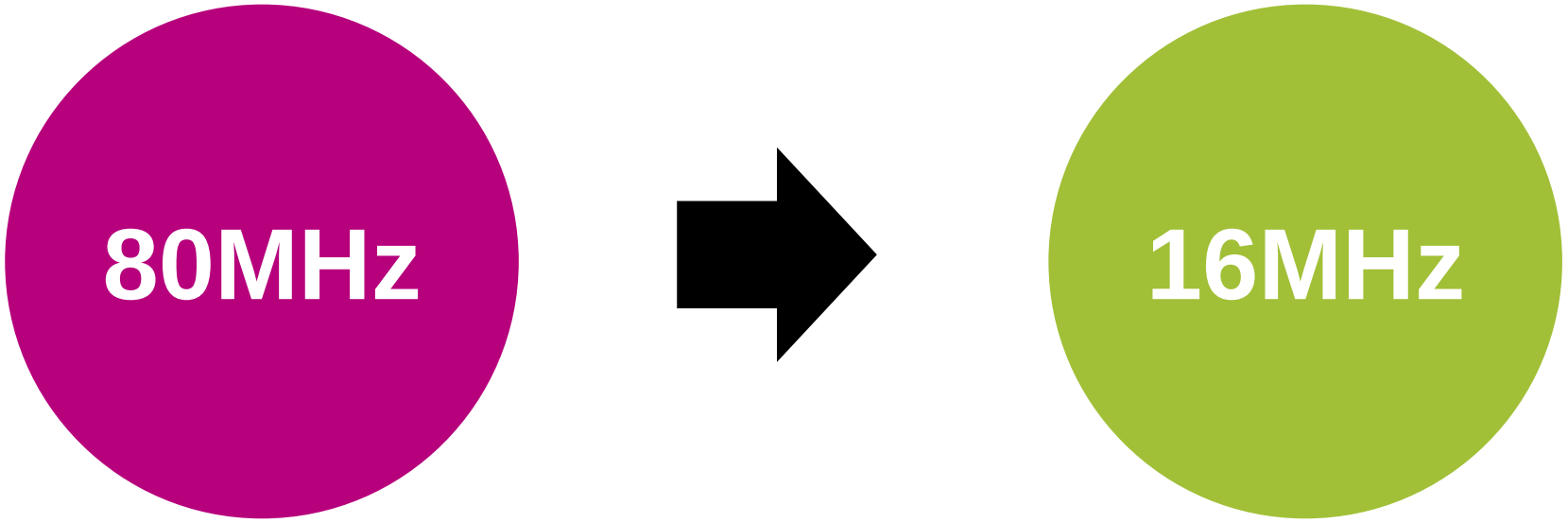


3. From the compiler

4. From every clock cycle



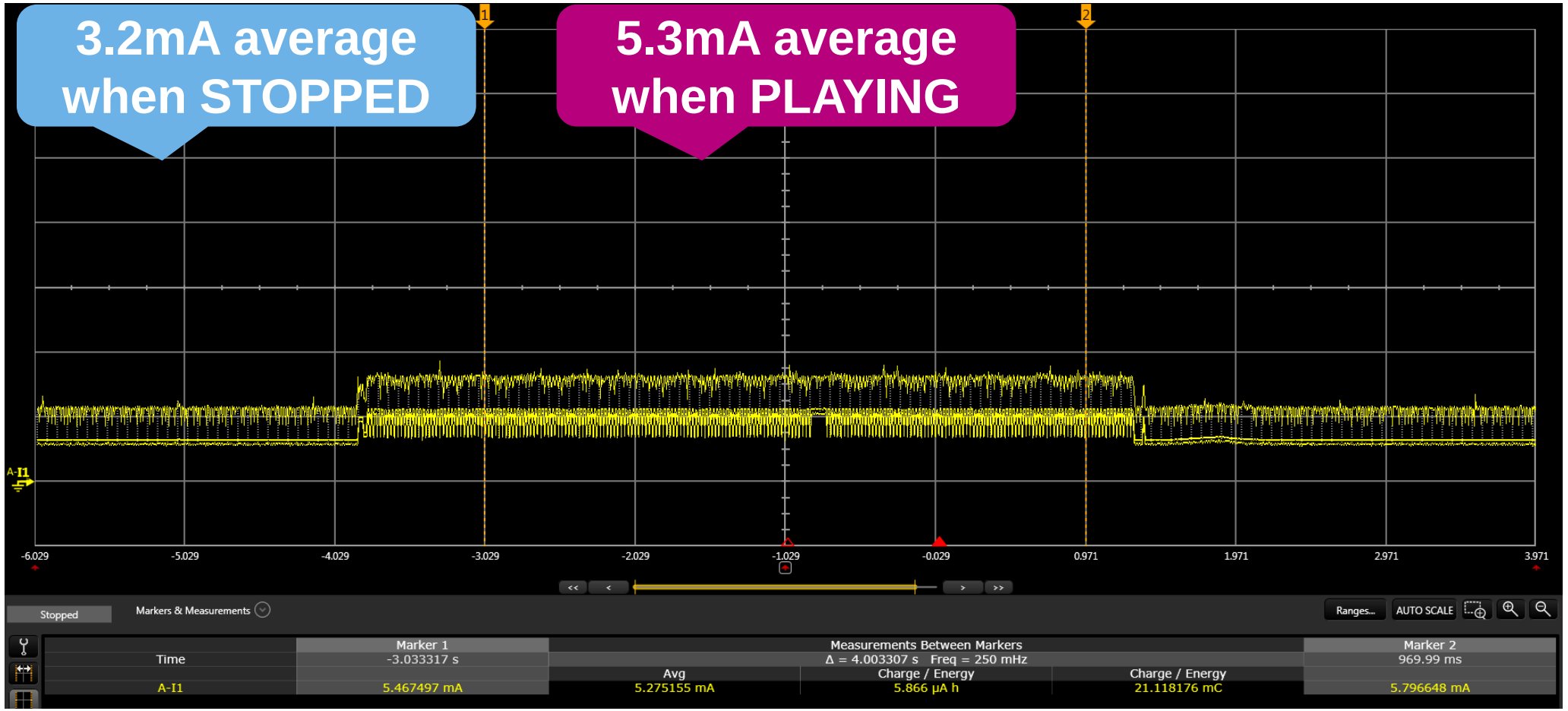
Why do we run @80MHz?



Let’s slow down, just PLL settings to be slightly modified

AudioPlayer optimization 5/5

Slower clocks



AudioPlayer optimization summary

V_{DD}=3.0V @25°C

Optimization applied	STOPPED		PLAYING	
	I_{DD}	Difference	I_{DD}	Difference
Original state (STEP6)	17.3 mA	-	17.8 mA	-
Tickless Mode	10.2 mA	-41%	11.9 mA	-33.1%
Clocks gating	8.5 mA	-16.7%	10.6 mA	-10.9%
Unused GPIOs	8.5 mA	~0	10.6 mA	~0
Compiler settings	8.5 mA	~0	9.9 mA	-6.6%
Slower clocks	3.2 mA	-62.3%	5.3 mA	-46.5%

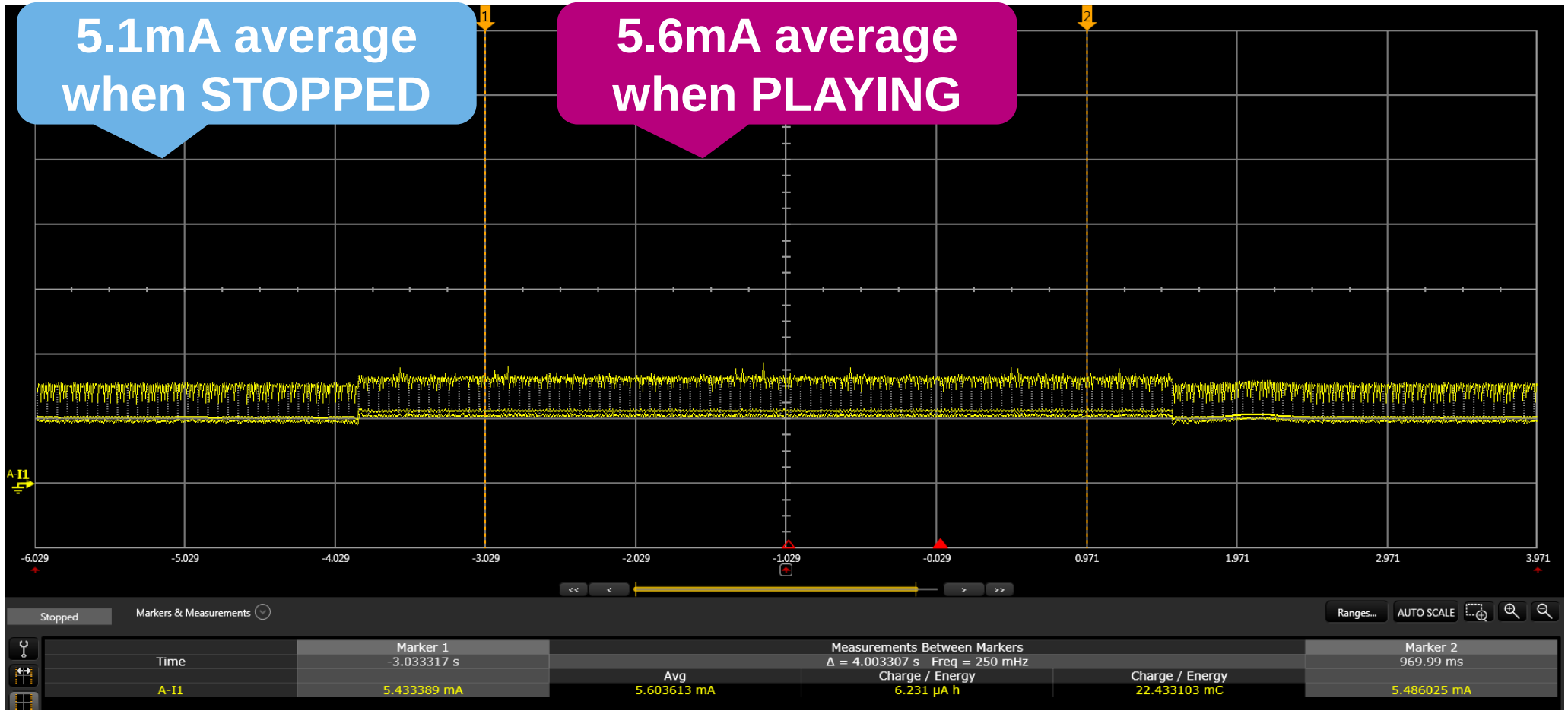
Just the MCU consumption considered
(including circuitries supplied from GPIOs)

AudioPlayer optimization 5/5

Slower clocks (without Tickless Mode)

5.1mA average when STOPPED

5.6mA average when PLAYING



life.augmented

AudioPlayer optimization summary

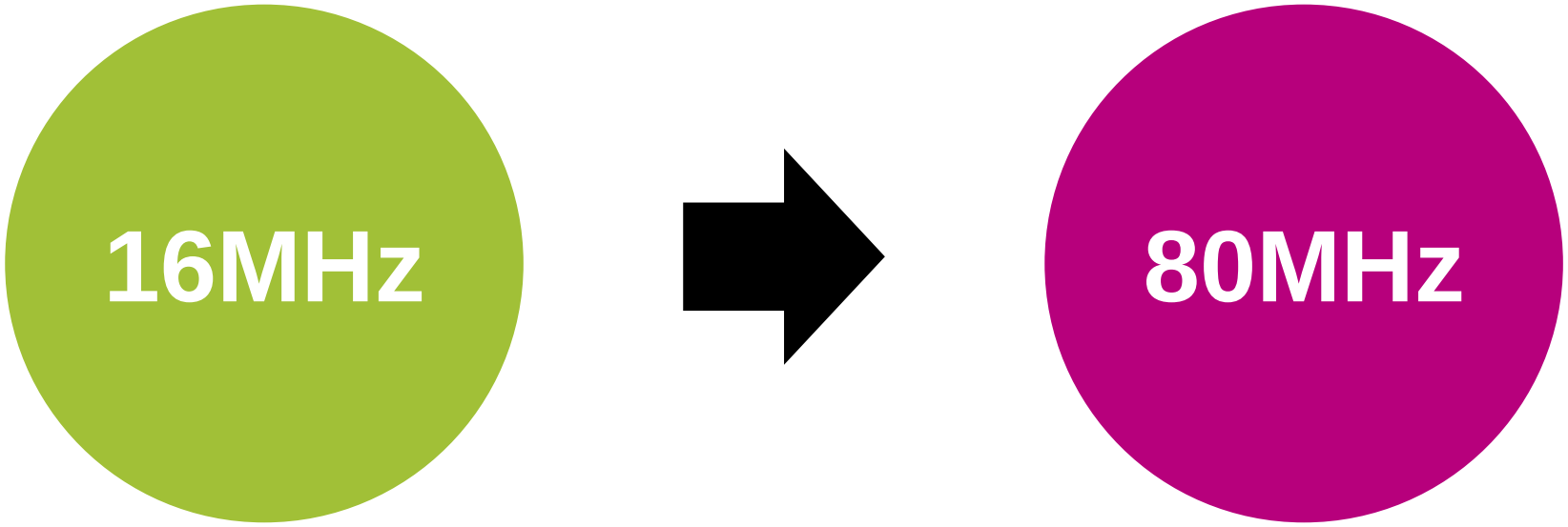
V_{DD}=3.0V @25°C

Optimization applied	STOPPED		PLAYING	
	I_{DD}	Difference	I_{DD}	Difference
Original state (STEP6)	17.3 mA	-	17.8 mA	-
Tickless Mode	10.2 mA	-41%	11.9 mA	-33.1%
Clocks gating	8.5 mA	-16.7%	10.6 mA	-10.9%
Unused GPIOs	8.5 mA	~0	10.6 mA	~0
Compiler settings	8.5 mA	~0	9.9 mA	-6.6%
Slower clocks	3.2 mA	-62.3%	5.3 mA	-46.5%
	5.1 mA	+37.3%	5.6mA	+5.4%

Tickless Mode disabled

High performance on demand

If we would need higher performance (MP3, WMA decoding, DSP post processing, equalizer)

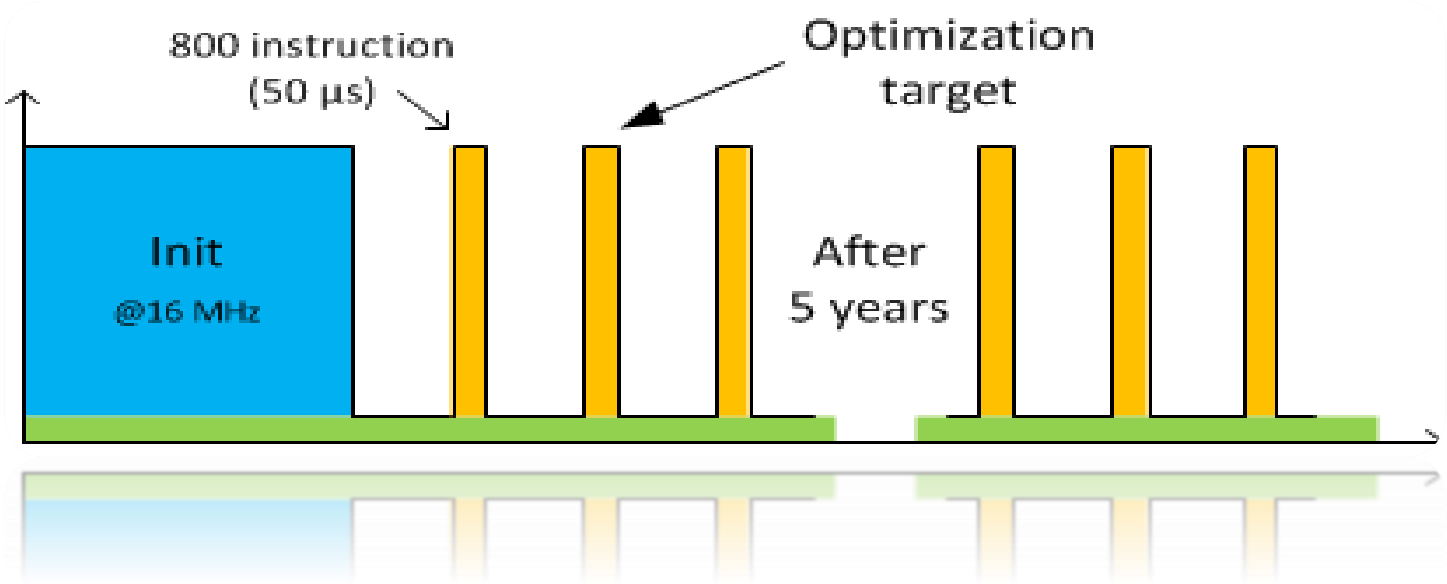


We can switch smoothly in runtime.

Analyze your application!

- Each single instruction is executed 158.10^6 times over the 5years
- Removing only 1 instruction will give **2.3days** time reserve over 5y.

1 instruction removed = **2.3 days saving**



Focus on details

- even 1 extra instruction if optimized (removed) can save a lot of energy

- **Optimized code = Reduced code** with the same functionality

➢ Gain: LOW power consumption + HIGH performance

OPTIMAL CASE !!!

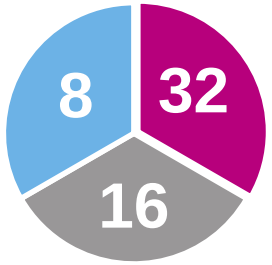
➢ x% of extra instruction = x% waste of core performance

Optimization flow for STM32L4

1. Use low-power modes (at least SLEEP)
2. Use compiler optimizations
3. Re-think use of peripherals
4. Execute from SRAM (real 0WS)
5. Use STM32CubeL4 LL drivers
6. Low level optimizations
7. Increase system frequency only when necessary

Squeeze the Maximum

1. From the architecture

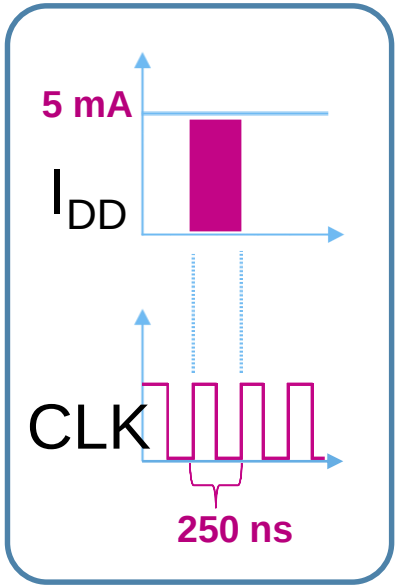


2. From the peripherals

DMA

3. From the compiler

4. From every clock cycle



LOWEST power consumption

HIGHEST performance

Enjoy!

www.st.com/stm32l4

MCU Shootout

@ **4.2**MHz
ARM™ Cortex-**M0+**™
(**0.95**DMIPS/MHz)

	MCU A	MCU B
Active mode	0.62 mA	2.4 mA
Low power mode	0.9 uA ✓	1.87 uA

@ **24**MHz
ARM™ Cortex-**M4**™
(**1.25**DMIPS/MHz)

Current or energy? And what about power profile?

V_{DD}	3.0V
T_A	25°C

	MCU A	MCU B
Energy (10% active in 1s interval)	188 uJ ✓	725 uJ **(~3.9x more)**
Energy (0.1% active in 1s interval)	4.56 uJ ✓	12.8 uJ **(~2.8x more)**

~7.5x faster?

life.augmented

Power versus Energy

- What if the MCUs have different architectures, speeds and Dhrystone/CoreMark scores?
  - The time spent in active mode will not be equal.

	MCU A	MCU B
Active mode time	10 %	1.33 %
Energy (in 1s)	188 uJ **(~1.9x more)**	101.3 uJ ✓

- Less time spent on the task ➔ lower the total energy consumed ➔ more CPU time we have for other activities

Conclusion

- **MCU A** might be better for simple tasks where simple peripherals are more important than the core and the code executed is simple too

- **MCU B** better for complex tasks with advanced peripherals where the CPU is used more

BUT, more advanced peripherals can perform better sometimes

Use the memory, if available 1/2

- **EXAMPLE 1** (short variables on 32-bit architecture)

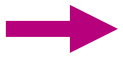
```
unsigned short int
counter = 10;

while (counter)
{


// do something
  counter--;
}
```

```
unsigned int
counter = 10;

while (counter)
{


// do something
  counter--;
}
```

Use the memory, if available 2/2

- **EXAMPLE 1** (short variables on 32-bit architecture)

```
;unsigned short int
counter = 10;

;while (counter)
;{
  MOVS   r0,#0x0A
loop:
;// do something
;  counter--;
;}
  SUBS   r0,r0,#1
  LSLS   r0,r0,#16
  LSRS   r0,r0,#16
  BNE    loop
```



Two data bytes more, but 2 instructions less !

```
;unsigned int
counter = 10;

;while (counter)
;{
  MOVS   r0,#0x0A
loop:
;// do something
;  counter--;
;}
  SUBS   r0,r0,#1
  BNE    loop
```

! usually NOT done by the compiler !

Unroll the whole loop, if you can

- **EXAMPLE 2**

! usually done by the compiler, if enabled

But which level of unrolling ?

```
unsigned int i = 64;

while (i != 0)
{
  i--; A[i] = B[i];
}
```

→

```
unsigned int i = 64;

while (i != 0)
{
  i--; A[i] = B[i];
  …
  i--; A[i] = B[i];
}
```

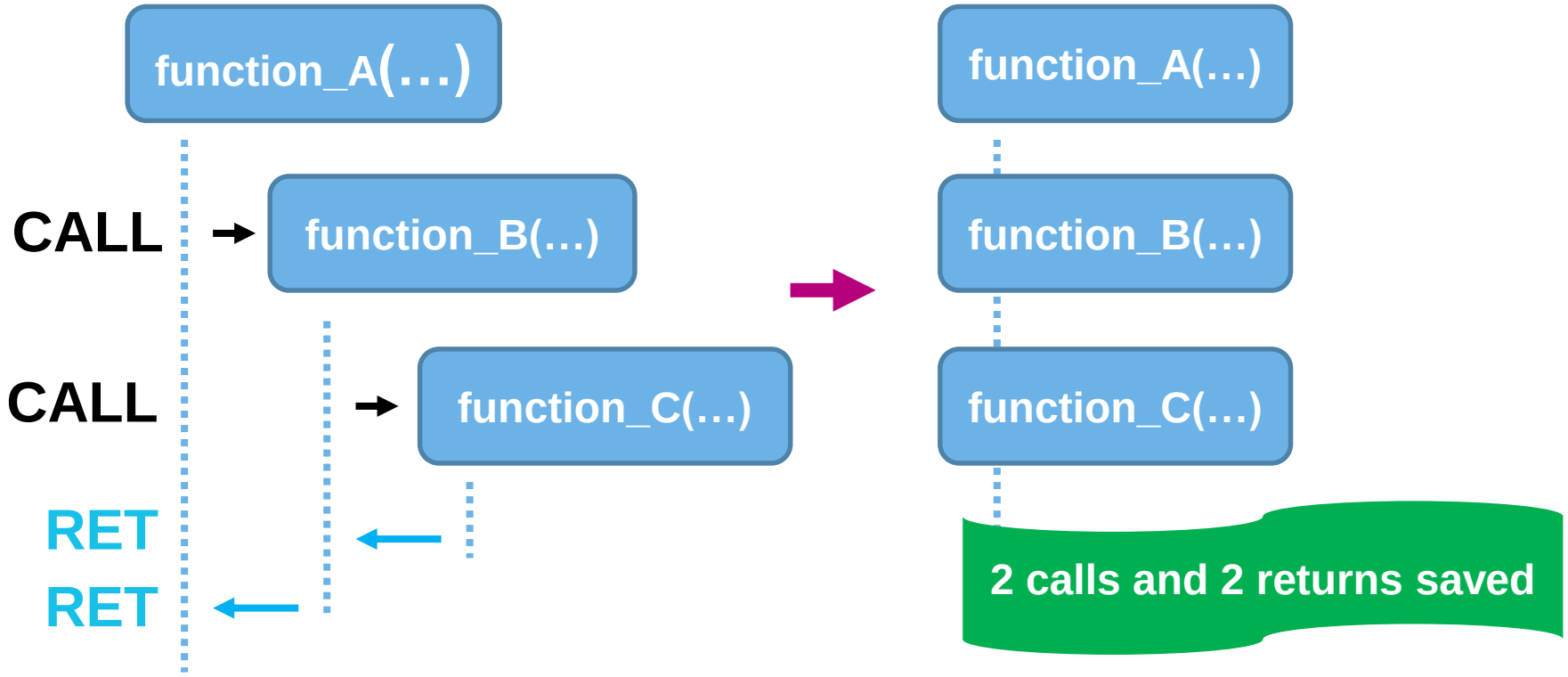
8x in total

56 conditional jumps saved

Use in-lining for time critical code

- **EXAMPLE 3**

usually done by the compiler, if enabled



Sometimes better to use simple instructions

- **EXAMPLE 4** (modulo)

! **usually NOT done by the compiler**

```
if ((i % 5) == 0)
{
   // do something
}
…
if (i == 100)
{
   // do something
   i = 0;
}
else {
   i++;
}
```

```
if (counter == 5)
{
   // do something
   counter = 0;
}
else {
   counter++;
}
…
if (i == 100)
{
   // do something
   i = 0;
}
else {
   i++;
}
```

Modulo is DIV instruction (2~12 clock cycles)

One data byte more, but simple!

Check the use of arithmetic

- Reuse the intermediate results – **usually done by the compiler** !
- **EXAMPLE 5** (subexpression elimination)

```
x = a + b + c
y = a + b + d
```

```
temp = a + b
x = temp + c
y = temp + d
```

Use the optimized math 1/3

- (STEP 1) – Do the math
  - (STEP 2) – Write down the algorithm
- **EXAMPLE 6** (geometric mean)

$$A = \sqrt[6]{x_1 * x_2 * x_3 * x_4 * x_5 * x_6} \rightarrow A = \left[(x_1 * x_2 * x_3 * x_4 * x_5 * x_6)^{\frac{1}{3}}\right]^{\frac{1}{2}}$$

in single-precision floating point (even in integer)

Use the optimized math 2/3

- (STEP 1) – Do the math
  - (STEP 2) – Write down the algorithm
- **EXAMPLE 6** (geometric mean)

```
float a,b,c,d,e,f,result;

...initialization of the variables

temp = a * b * c * d * e * f;
result = powf(temp, (1.0f / 6.0f));
```

```
float a,b,c,d,e,f,result;

...initialization of the variables

temp = a * b * c * d * e * f;
result = cbrtf(temp);
result = sqrtf(result);
```

Use the optimized math 3/3

- (STEP 1) – Do the math
  - (STEP 2) – Write down the algorithm
- **EXAMPLE 6** (geometric mean)

! Could be done by the compiler, but you need to check that !