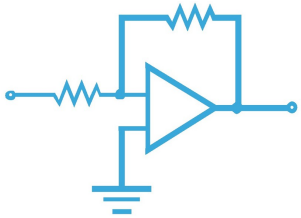
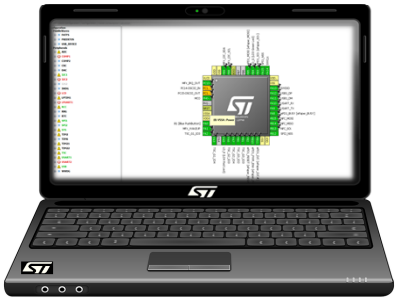
STM32가 제안하는 초저전력 IoT 어플리케이션 구현하기

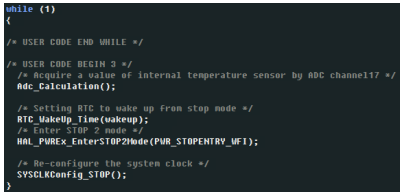
STMicroelectronics Korea - MMS

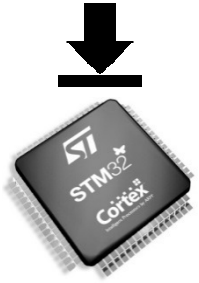
life.augmented

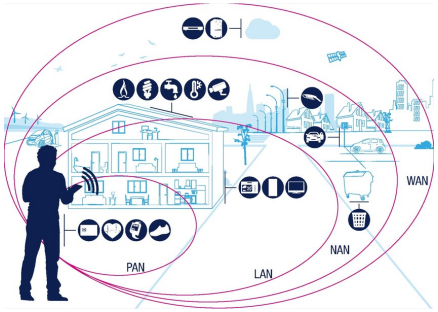
IoT Application design process

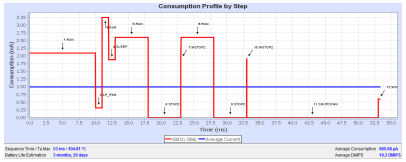




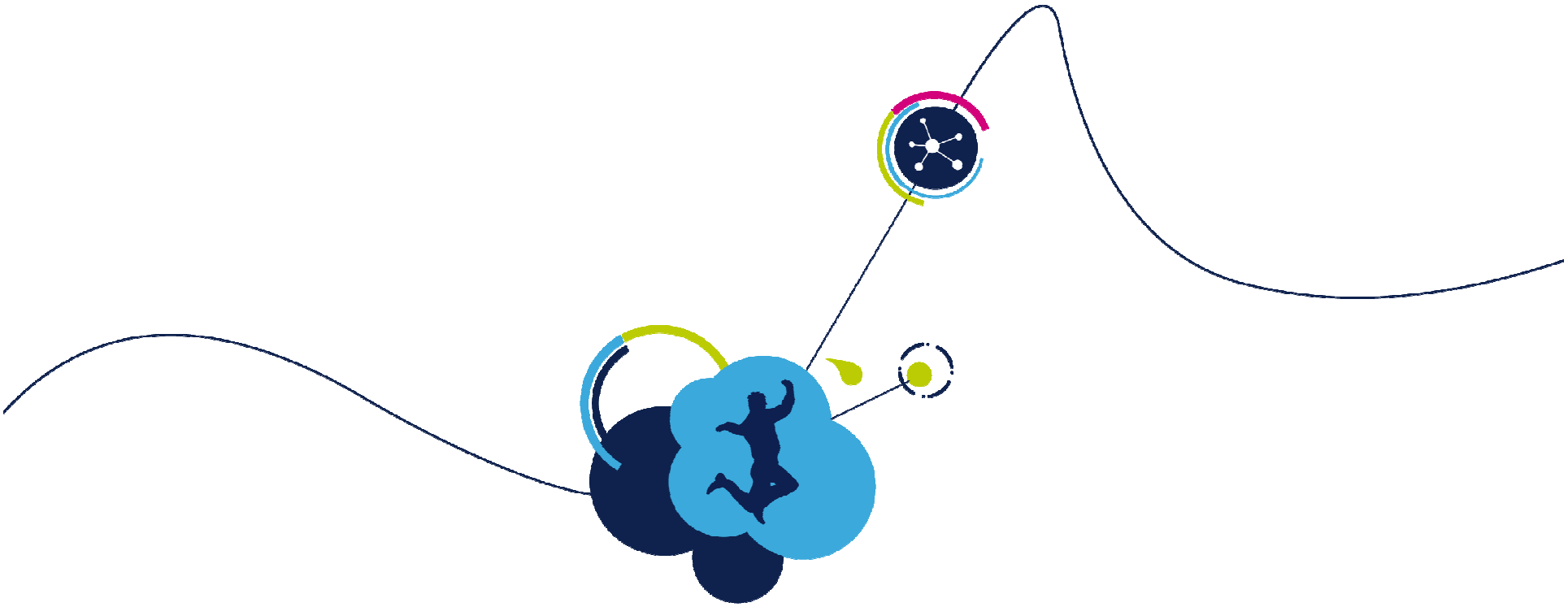








life.augmented

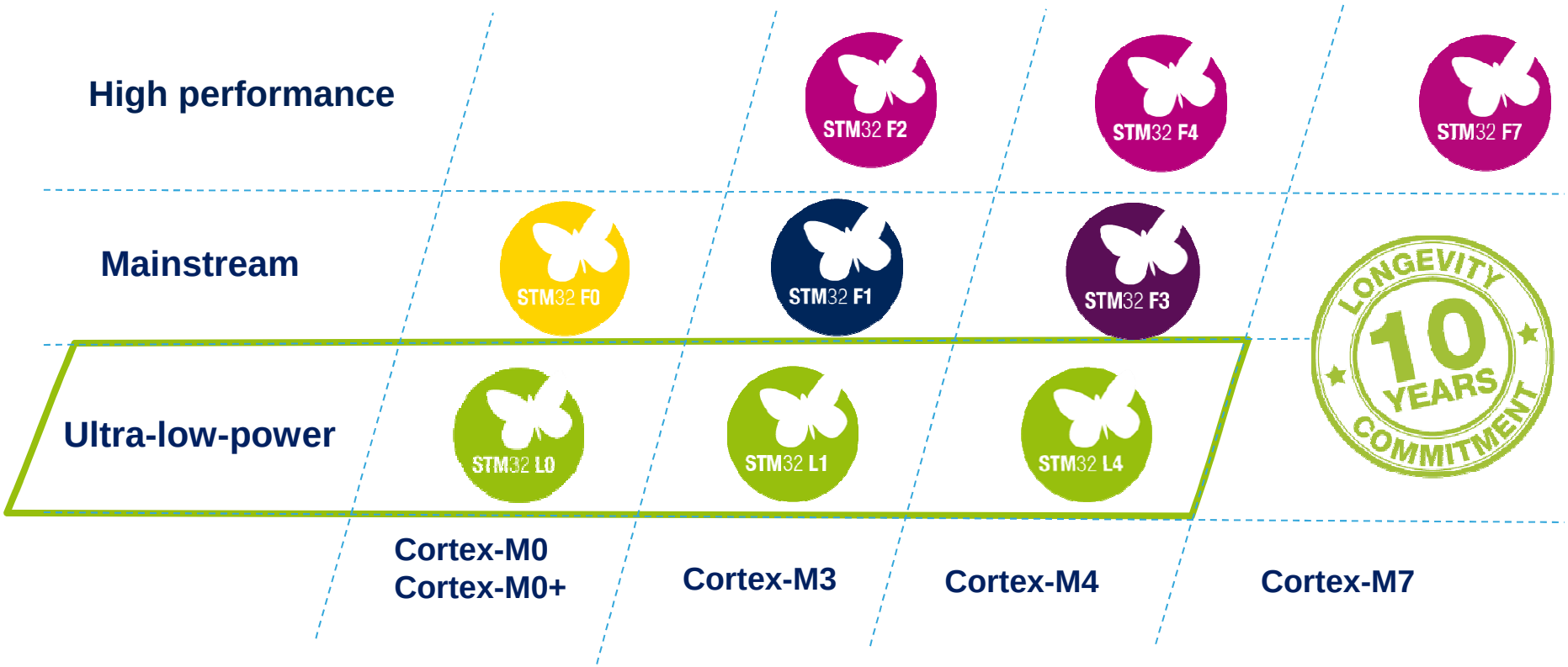


Ultra Low Power MCU Families

life.augmented

Great Investment

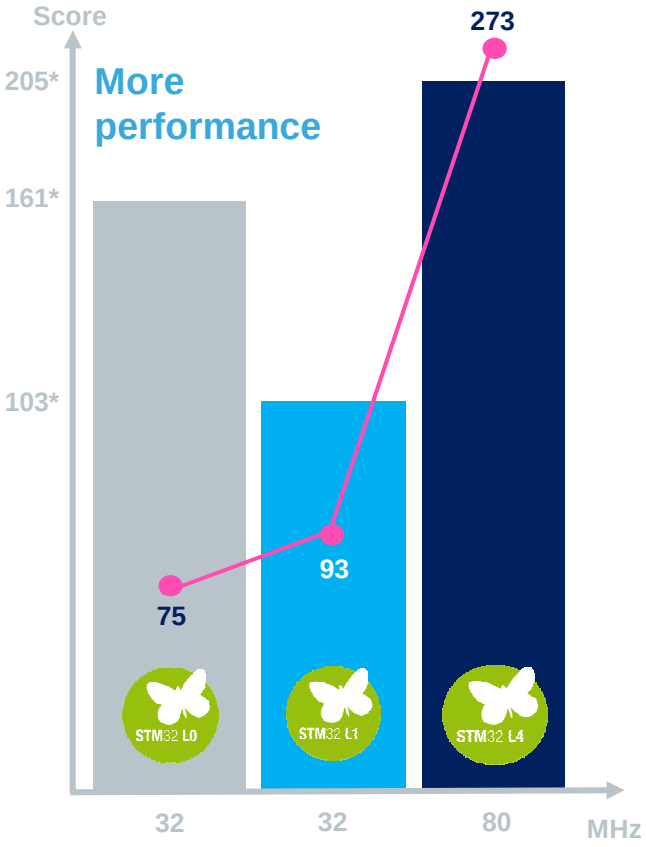
9 product series / 32 product lines / 600 P/N



life.augmented

STM32L ULP offer

Completed the ultra-low-power family

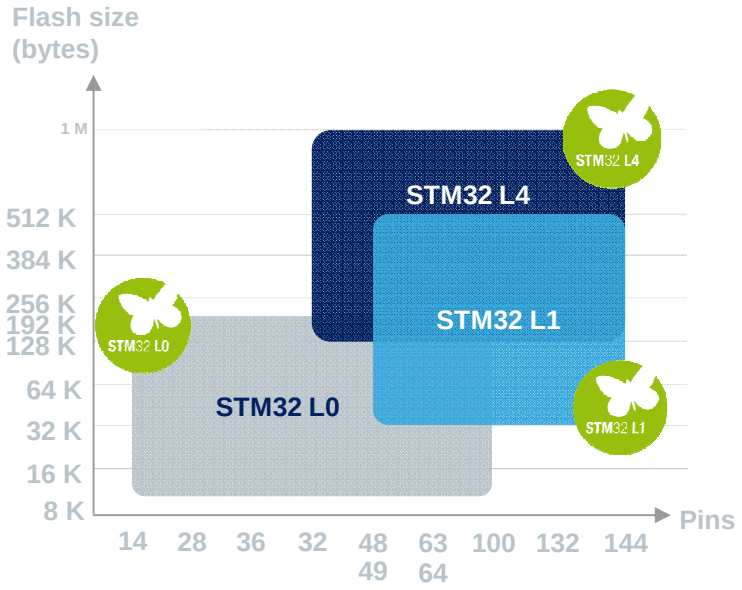


ULPBENCH™ An EEMBC Benchmark

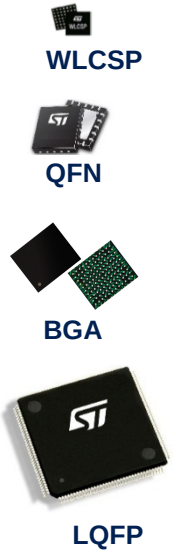
CORE MARK® An EEMBC Benchmark

life.augmented

More memory and pin counts



More packages



STM32L ULP offer

Completed the ultra-low-power family

Cost-smart ULP Champion

Cortex-M0+ @ 32 MHz
Operating range:
1.65 to 3.6V
8/16-bit applications
Numerous pin counts

3 product lines,
Cost-effective,
Smaller packages
USB, LCD, Analog
16 to 192 Kbytes of Flash
Up to 20 Kbytes of SRAM

Broad Range Foundation

Cortex-M3 @ 32 MHz
Operating range:
1.65 to 3.6V
Wide choice of memory sizes

3 product lines,
USB, LCD, AES,
Rich Analog
True EEPROM,
Dual bank Flash (RWW)
32 to 512 Kbytes of Flash
Up to 80 Kbytes of SRAM

ULP with performance

Cortex-M4 w/ FPU @ 80 MHz
Operating range:
1.71 to 3.6V
Advanced Peripheral Performance

3 product lines,
ADC 5 Msps, PGA, Compar.,
DAC, op amp, USB OTG,
LCD, AES
256 Kbytes to 1 Mbyte of Flash
Up to 128 Kbytes of SRAM

life.augmented

STM32L1 – Market proven solution

STM32L1 Product lines

Cortex®-M3 (32 MHz with MPU)

- Low voltage 1.65 to 3.6V
- Dynamic voltage scaling
- 5 clock sources
- Advanced RTC w/ cal.
- Multiple USART, SPI, I²C
- 16- and 32-bit timers
- - 40 to 85 °C oper. temp. Up to 105 °C in LP Modes
- 2 watchdogs
- Brown-out Reset
- Programmable voltage detector (PVD)
- DMA
- Reset circuitry POR/PDR
- 12-bit ADC 1 MSPS
- 12-bit DAC

Product line	Flash memory (KB)	RAM (KB)	EEPROM (KB)	Memory I/F	Op amps	Comp.	Temp. sensor	Capacitive touch	Segment LCD driver	128-bit AES
STM32L100 Value line	32 to 256	4 to 16	2	-	-	-	-	-	Up to 8x28	-
STM32L151 STM32L152	32 to 512	16 to 80	4 to 16	SDIO FSMC	●	●	●	●	- Up to 8x40	-
STM32L162	256 to 512	32 to 80	8 to 16	SDIO FSMC	●	●	●	●	Up to 8x40	●

STM32L1 Ultra-low-power

- ARM® Cortex® -M3 at 32 MHz – 33 DMIPS
- Dynamic run mode: down to 177 µA/MHz
- Stop with Full RAM retention 435 nA (1.3 µA with RTC)
- Standby mode + RTC: 900 nA with backup registers
- Standby mode: 280 nA with backup registers
- Dual-bank Flash memory and True embedded EEPROM
- Operates at up to 105 °C

ULPBENCH™ An EEMBC Benchmark **103**

COREMARK® An EEMBC Benchmark **93**

www.st.com/stm32l1

life.augmented

STM32L0 – Entry level solution

STM32L0 Product lines

Cortex®-M0+ (32 MHz with MPU)

- Low voltage 1.65 to 3.6V
- Dynamic Voltage Scaling
- 5 clock sources
- Advanced RTC w/ calibration
- Multiple USART, SPI, I²C
- Multiple 16-bit timers
- - 40 to 125°C Operating
- 2 watchdogs
- Program Voltage Detector
- Reset circuitry POR/PDR
- Brown Out Reset
- DMA
- Comparators
- Temperature sensor
- AES 128-bit

Product line	FLASH (KB)	RAM (KB)	EEPROM (KB)	12-bit ADC 1.14 Msps	LP[1] UART	LP[1] 16-bit timer	12-bit DAC	Touch sense	True RNG	USB 2.0 FS Crystal-less	Segment LCD Driver
STM32L0x1 Access	Up to 192 (dual bank)	20	6 (rww)[2]	Down to 1,65V	•	•					
STM32L0x2 USB	Up to 192 (dual bank)	20	6 (rww)[2]	Down to 1,65V	•	•	Up to 2	•	•	•	
STM32L0x3 USB & LCD	Up to 192 (dual bank)	20	6 (rww)[2]	Down to 1,65V	•	•	Up to 2	•	•	•	Up to 8x48 Or 4x52

Note 1: Low-power peripherals available in ultra-low-power modes
Note 2 : Read while write from flash to EEPROM

STM32L0 Ultra-low-power

- ARM® Cortex® -M0+ at 32 MHz
- Dynamic run mode down to 87 µA/MHz
- Stop mode with RAM + LTC (low-power time clock): 440 nA
- Wakeup: 3.5 µs (RAM) / 5 µs (Flash memory)
- 12/16-bit ADC: 1 MSPS – 240 µA – 1.65 V capable
- Unique ID / 128-bit AES / Flash Proprietary Stack protection
- Operates at up to 125 °C

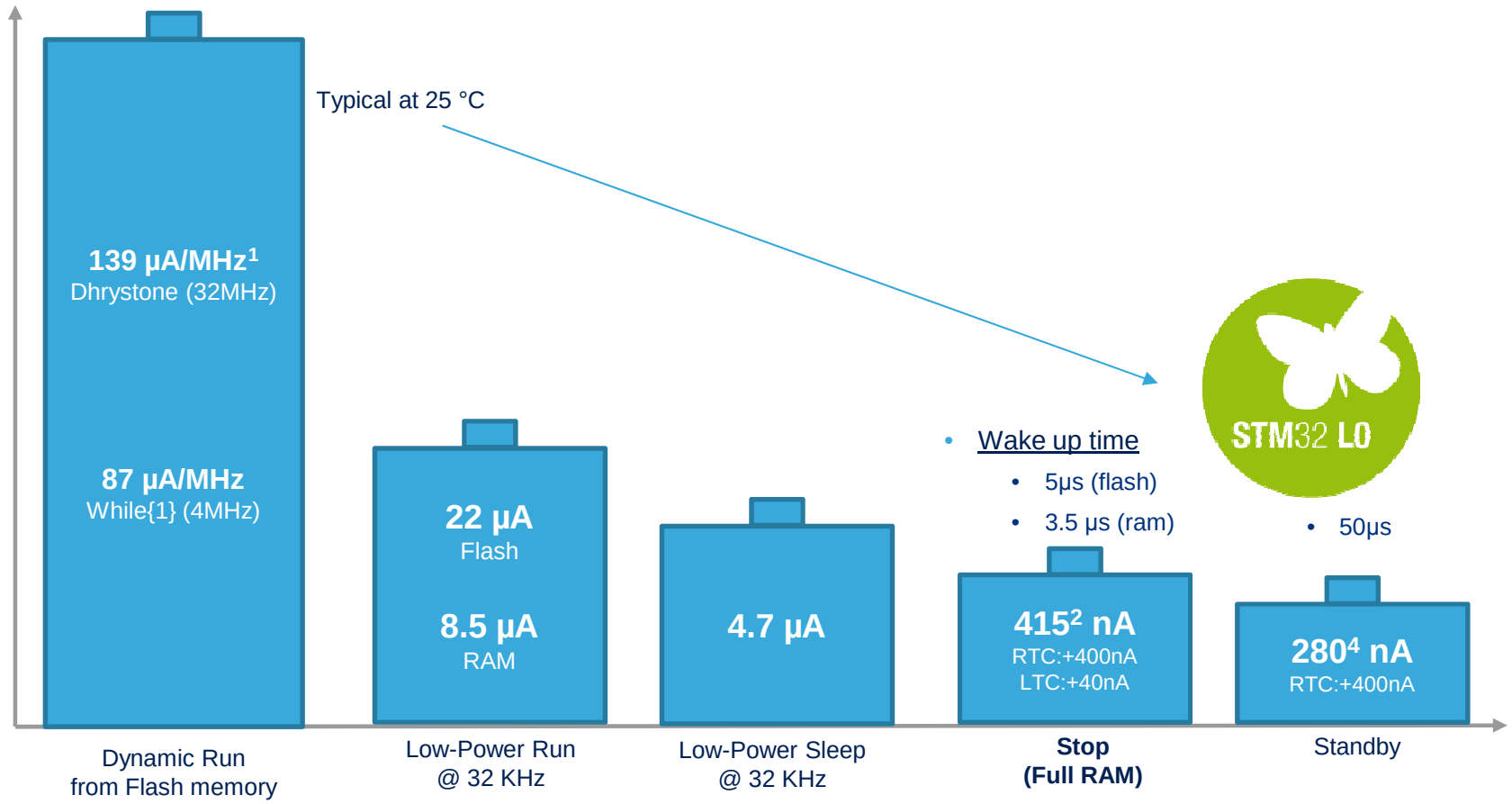
ULPBENCH™ An EEMBC Benchmark **161**

COREMARK® An EEMBC Benchmark **75**

www.st.com/stm32l0

life.augmented

STM32L05* - power consumption



1. Dhrystone power consumption value executed from Flash (Prefetch off) with VDD=3.3V
2. STOP mode consumption with **Full Ram data retention** (RTC value given with LSE low-drive using 32,768kHz crystal)
3. LTC**:** Low-power Time Counter @ 100Hz with external oscillator (LSE)
4. STANDBY mode consumption with **20Byte of backup register and Power supply monitoring**

STM32L4 – Performance solution

STM32L4 Product lines

Cortex®-M4 (DSP + FPU) – 80 MHz

- ART Accelerator™
- USART, SPI, I²C
- QuadSPI
- 16- and 32-bit timers
- SAI + audio PLL
- SWP
- 1x CAN
- 2x 12-bit DAC
- Temperature sensor
- Low voltage 1.71 to 3.6 V
- V_{BAT} mode
- Unique ID
- Capacitive touch sensing

Product line	Flash memory (KB)	RAM (KB)	Memory I/F	2 x Op amps	2 x Comp.	4x / 8ch Sigma Delta Interface	5 MSPS, 12- bit ADC 16-bit HW over sampling	USB 2.0 OTG FS	Segment LCD Driver	128/ 256-bit AES
STM32L471 Access	512 to 1024	128	SDIO FSMC	●	●	●	3			
STM32L475 USB OTG	256 to 1024	128	SDIO FSMC	●	●	●	3	●		
STM32L476 USB OTG & LCD	256 to 1024	128	SDIO FSMC	●	●	●	3	●	Up to 8x40	
STM32L486 USB OTG & LCD & AES	1024	128	SDIO FSMC	●	●	●	3	●	Up to 8x40	●

STM32L4 Ultra-low-power

- ARM® Cortex® -M4 at 80 MHz with DSP + FPU – 100 DMIPS
- Dynamic run mode at 100 µA/MHz
- Down to 600 nA with 32 kHz RTC + 32 Kbytes of RAM + I/Os
- Down to 265 nA with 32 kHz RTC or 44 nA without RTC
- 12/16-bit ADC with 5 MSPS – 240 µA – 1.65 V capable
- Operates at up to 125 °C

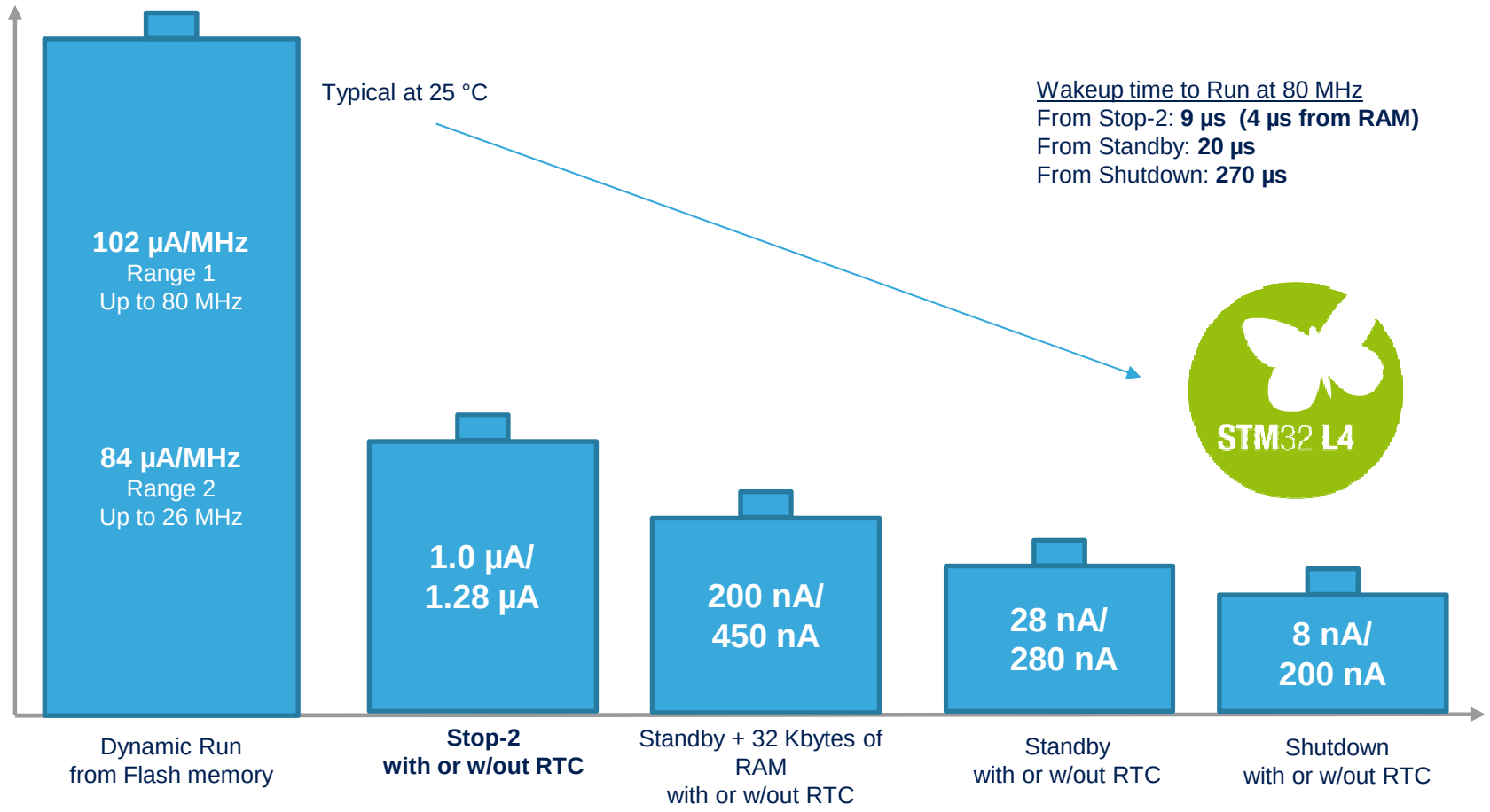
ULPBENCH™ An EEMBC Benchmark 204

CORE MARK® An EEMBC Benchmark 273

www.st.com/stm32l4

life.augmented

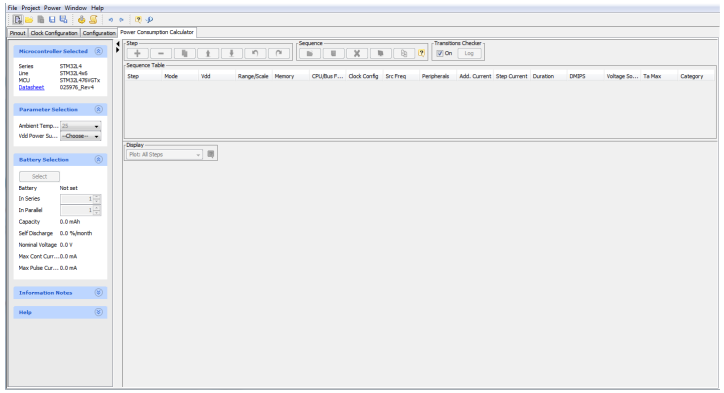
STM32L4 - power consumption



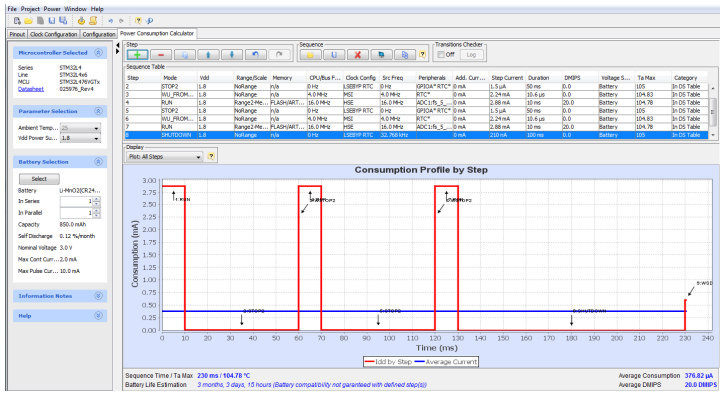
Ecosystem – STM32CubeMX

Power Consumption Calculator

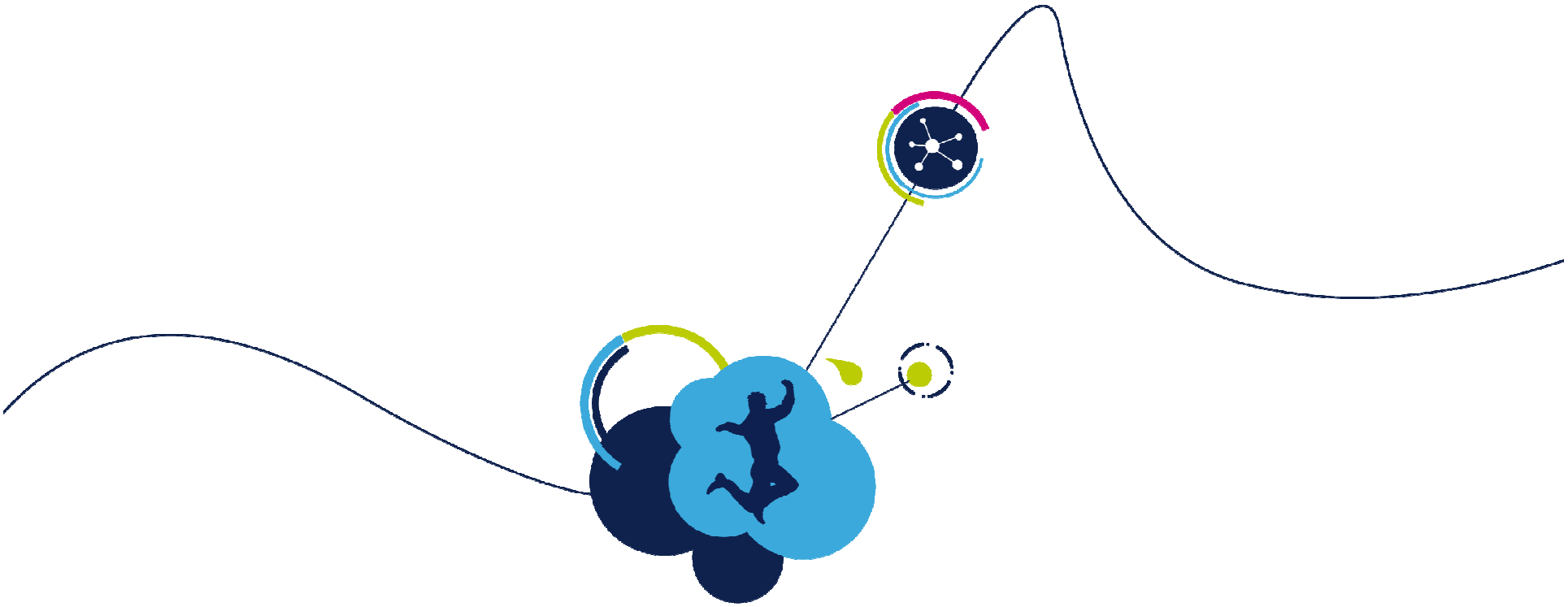
- STM32CubeMX offers the Power Consumption Calculator tab, which, given a microcontroller, a battery model and a user-defined power sequence.



- Provides the following results :
  - Average current consumption
  - Battery life
  - Average DMIPs
  - Max. ambient temperature(T_{AMAX})



www.st.com/stm32cubemx



Power Management

life.augmented

STM32 Low Power modes

- The STM32 features a variety of low power modes
  - LPRUN
  - SLEEP and LPSLEEP
  - STOP
  - STANDBY
  - SHUTDOWN

- STM32F4 vs STM32L0 vs STM32L4 : Low power modes

Low Power Mode	STM32F4	STM32L0	STM32L4
	-	LPRUN	LPRUN
	SLEEP	SLEEP	SLEEP
	-	LPSLEEP	LPSLEEP
	STOP	STOP	STOP0
			STOP1
			STOP2
	STANDBY	STANDBY	STANDBY
	-	-	SHUTDOWN
	VBAT	-	VBAT

STM32L4 Low Power modes

STM32L4 low power modes summary

Mode	Regulator	CPU	Flash	SRAM	Clocks	Peripherals In Bold : wakeup source
Run	R1 R2	Yes	ON(1)	ON	Any	All
LPRun	LPR	Yes	ON(1)	ON	Any *except PLL*	All except OTG, SDMMC, RNG
Sleep	R1 R2	No	ON(1)	ON(2)	Any	All **Any IT or event**
LPSleep	LPR	No	ON(1)	ON(2)	Any *except PLL*	All except OTG, SDMMC, RNG **Any IT or event**
Stop 0	R1 R2	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**
Stop 1	LPR	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**
Stop 2	LPR	No	OFF	ON	LSE/LSI	**Reset pin, all I/Os BOR,PVD,PVM,RTC,LCD,IWDG, COMPx,LPUART,I2C3,LPTIM1**
Standby	LPR OFF	DOWN	OFF	SRAM2 ON DOWN	LSE/LSI	**Reset pin, 5 WKUPx pins BOR, RTC, IWDG**
Shutdown	OFF	DOWN	OFF	DOWN	LSE	**Reset pin, 5 WKUPx pins RTC**

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

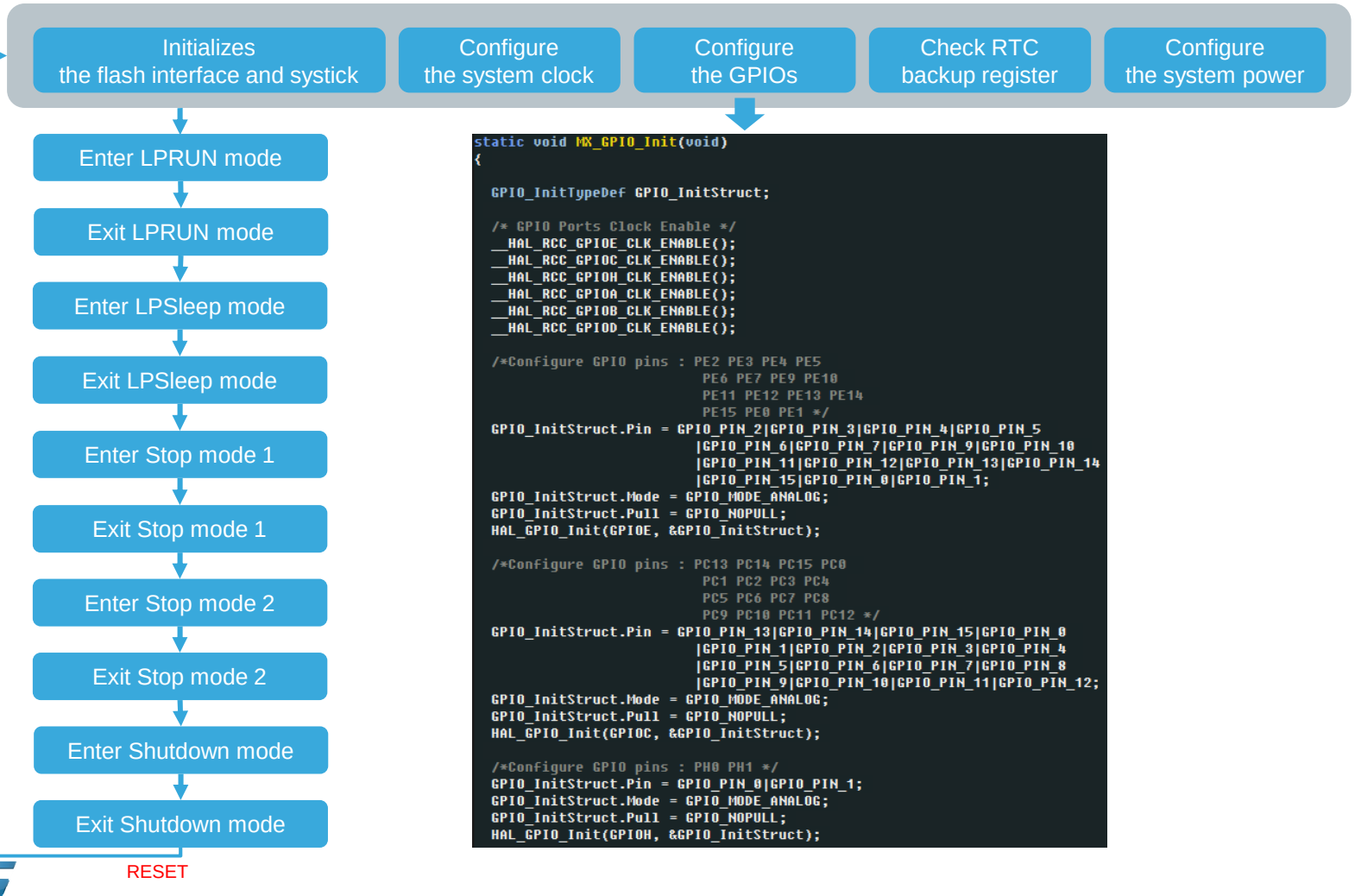
Implementation of Low Power Mode

How to initialize STM32L4


```
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_8;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV16;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV16;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the main internal regulator output voltage
    */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }
```

Implementation of Low Power Mode

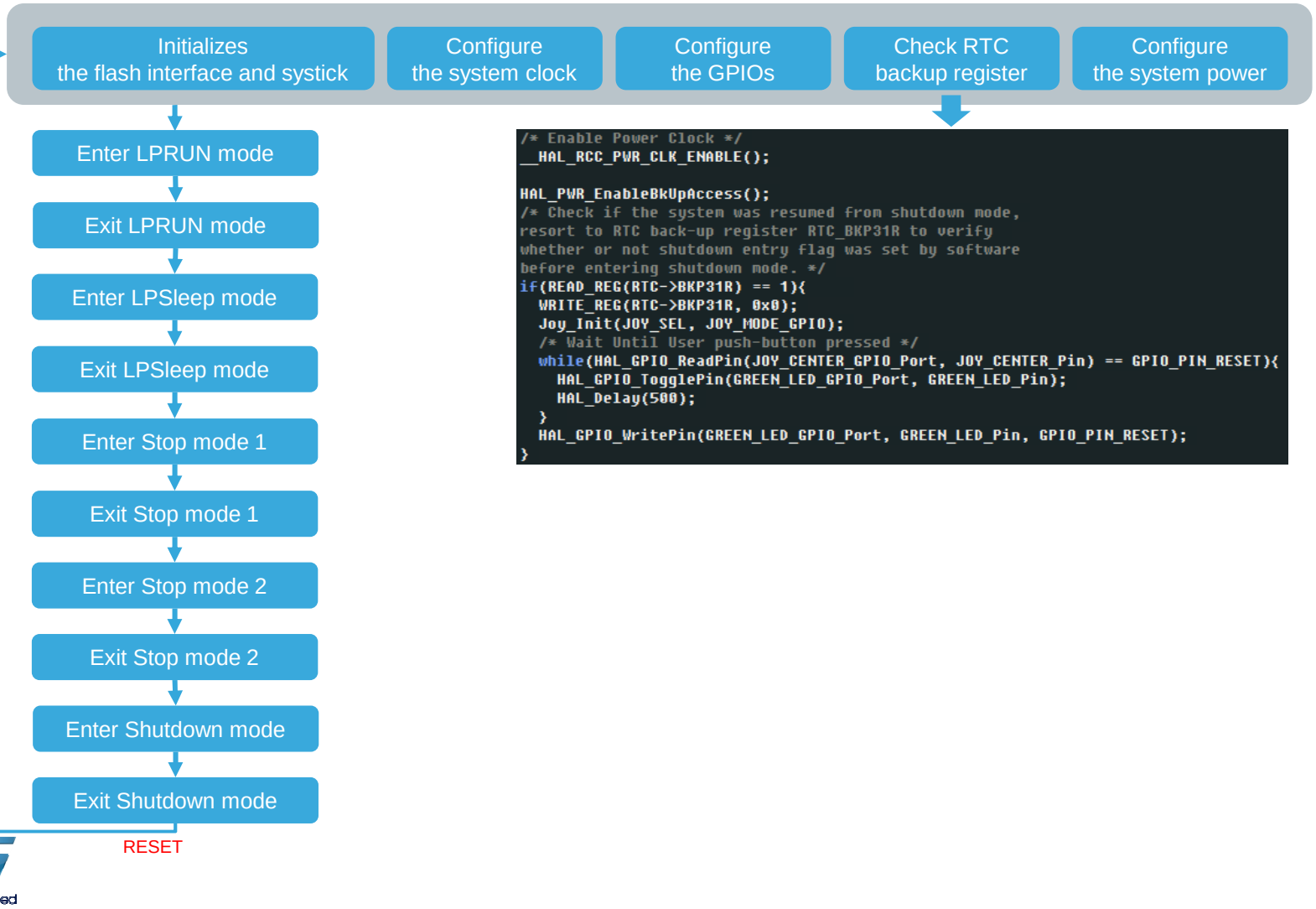
How to initialize STM32L4




```
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pins : PE2 PE3 PE4 PE5
                           PE6 PE7 PE9 PE10
                           PE11 PE12 PE13 PE14
                           PE15 PE0 PE1 */
  GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5
                        |GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_9|GPIO_PIN_10
                        |GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14
                        |GPIO_PIN_15|GPIO_PIN_0|GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  /*Configure GPIO pins : PC13 PC14 PC15 PC0
                           PC1 PC2 PC3 PC4
                           PC5 PC6 PC7 PC8
                           PC9 PC10 PC11 PC12 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15|GPIO_PIN_0
                        |GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4
                        |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8
                        |GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PH0 PH1 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);
```

Implementation of Low Power Mode

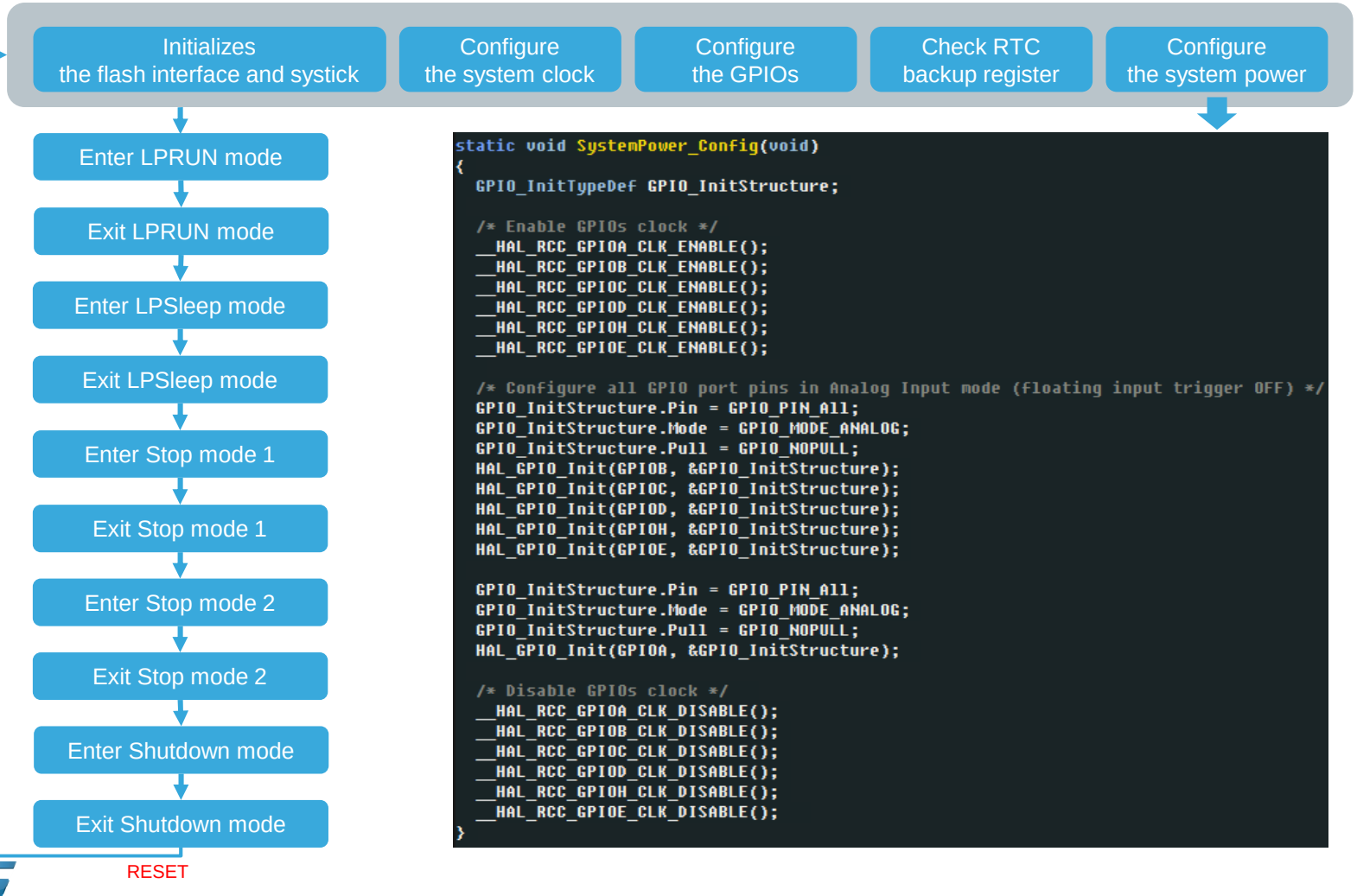
How to initialize STM32L4




```
/* Enable Power Clock */
__HAL_RCC_PWR_CLK_ENABLE();

HAL_PWR_EnableBkUpAccess();
/* Check if the system was resumed from shutdown mode,
resort to RTC back-up register RTC_BKP31R to verify
whether or not shutdown entry flag was set by software
before entering shutdown mode. */
if(READ_REG(RTC->BKP31R) == 1){
  WRITE_REG(RTC->BKP31R, 0x0);
  Joy_Init(JOY_SEL, JOY_MODE_GPIO);
  /* Wait Until User push-button pressed */
  while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET){
    HAL_GPIO_TogglePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin);
    HAL_Delay(500);
  }
  HAL_GPIO_WritePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin, GPIO_PIN_RESET);
}
```

Implementation of Low Power Mode

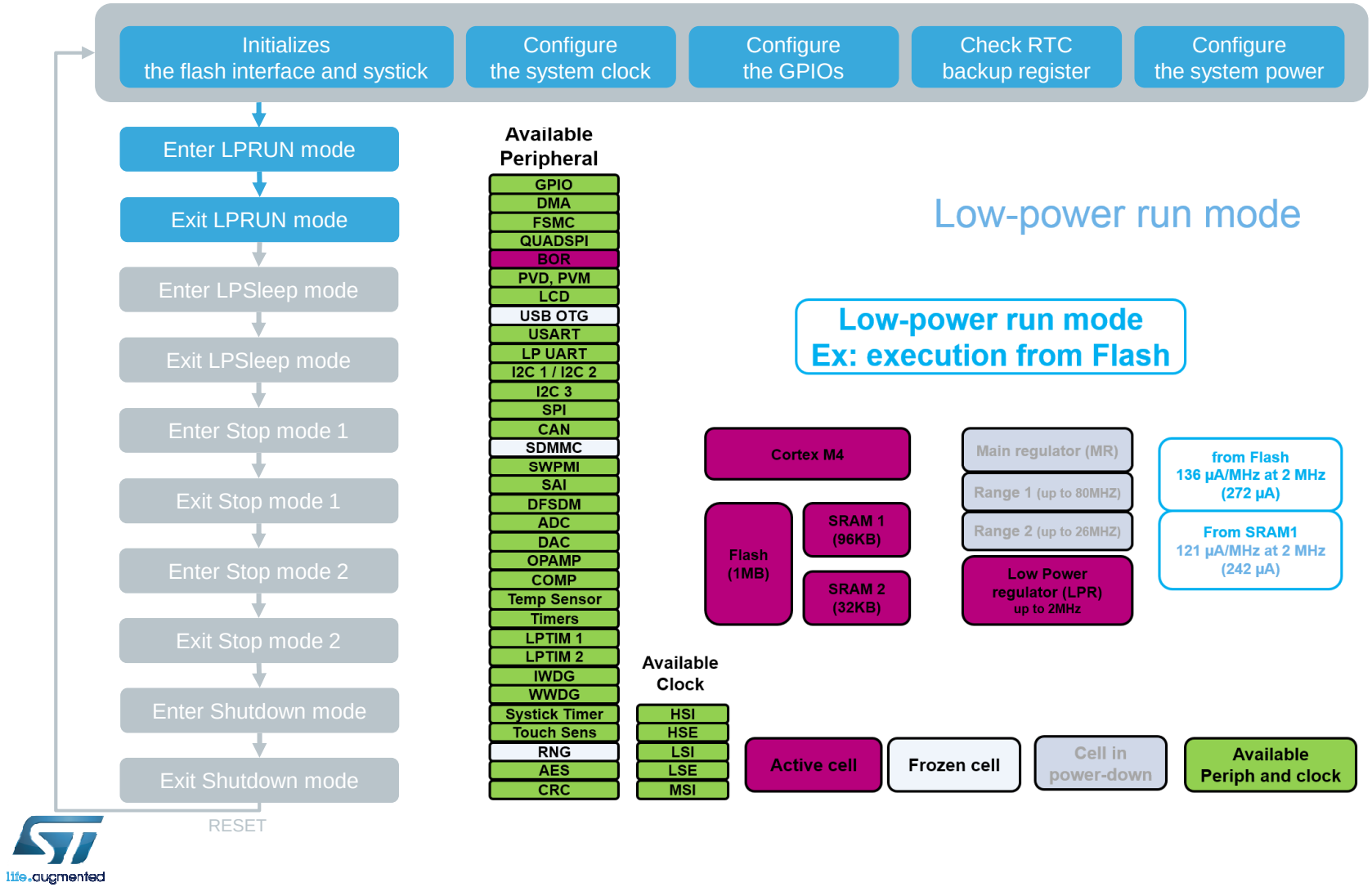
How to initialize STM32L4




```
static void SystemPower_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* Enable GPIOs clock */
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();

  /* Configure all GPIO port pins in Analog Input mode (floating input trigger OFF) */
  GPIO_InitStructure.Pin = GPIO_PIN_All;
  GPIO_InitStructure.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStructure.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStructure);
  HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);
  HAL_GPIO_Init(GPIOD, &GPIO_InitStructure);
  HAL_GPIO_Init(GPIOH, &GPIO_InitStructure);
  HAL_GPIO_Init(GPIOE, &GPIO_InitStructure);

  GPIO_InitStructure.Pin = GPIO_PIN_All;
  GPIO_InitStructure.Mode = GPIO_MODE_ANALOG;
  GPIO_InitStructure.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* Disable GPIOs clock */
  __HAL_RCC_GPIOA_CLK_DISABLE();
  __HAL_RCC_GPIOB_CLK_DISABLE();
  __HAL_RCC_GPIOC_CLK_DISABLE();
  __HAL_RCC_GPIOD_CLK_DISABLE();
  __HAL_RCC_GPIOH_CLK_DISABLE();
  __HAL_RCC_GPIOE_CLK_DISABLE();
}
```

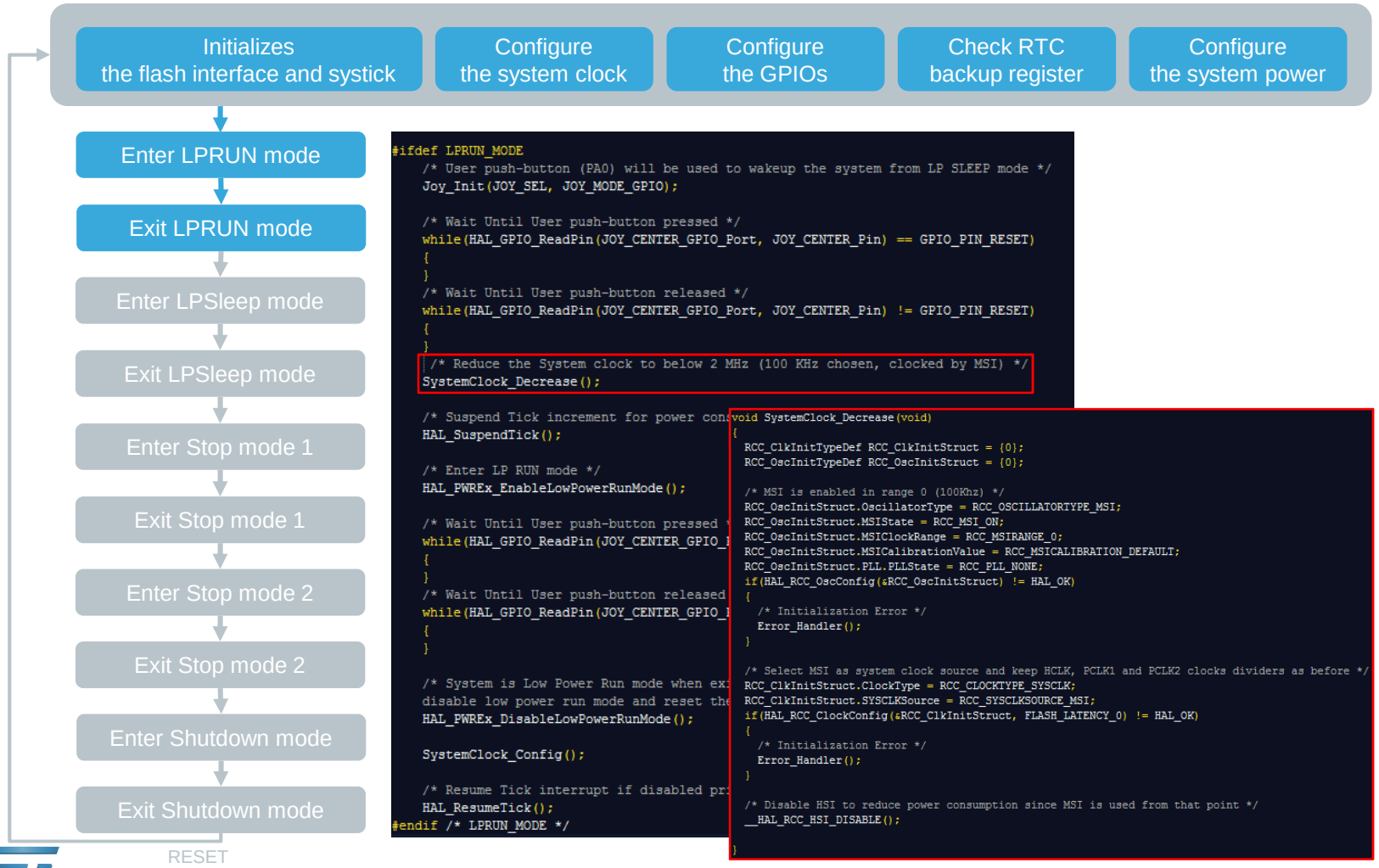
Implementation of Low Power Mode

Low Power Run Mode (LPRun mode)



Implementation of Low Power Mode

Low Power Run Mode (LPRun mode)



life.augmented

Implementation of Low Power Mode

Low Power Run Mode (LPRun mode)


```
#ifdef LPRUN_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_GPIO);

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }
    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment for power consumption purposes       */
    HAL_SuspendTick();

    /* Enter LP RUN mode */
    HAL_PWREx_EnableLowPowerRunMode();

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
    disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();

    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Run mode entry */
    HAL_ResumeTick();
#endif /* LPRUN_MODE */
```

Implementation of Low Power Mode

Low Power Run Mode (LPRun mode)


```
#ifdef LPRUN_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_GPIO);

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }
    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment for power consumption purposes        */
    HAL_SuspendTick();

    /* Enter LP RUN mode */
    HAL_PWREx_EnableLowPowerRunMode();

    /* Wait Until User push-button pressed
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_
    {
    }
    /* Wait Until U
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
    disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();

    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Run mode entry */
    HAL_ResumeTick();
#endif /* LPRUN_MODE */
```

```
void HAL_PWREx_EnableLowPowerRunMode(void)
{
  /* Set Regulator parameter */
  SET_BIT(PWR->CR1, PWR_CR1_LPR);
}
```

Bit 14 **LPR**: Low-power run

When this bit is set, the regulator is switched from main mode (MR) to low-power mode (LPR).

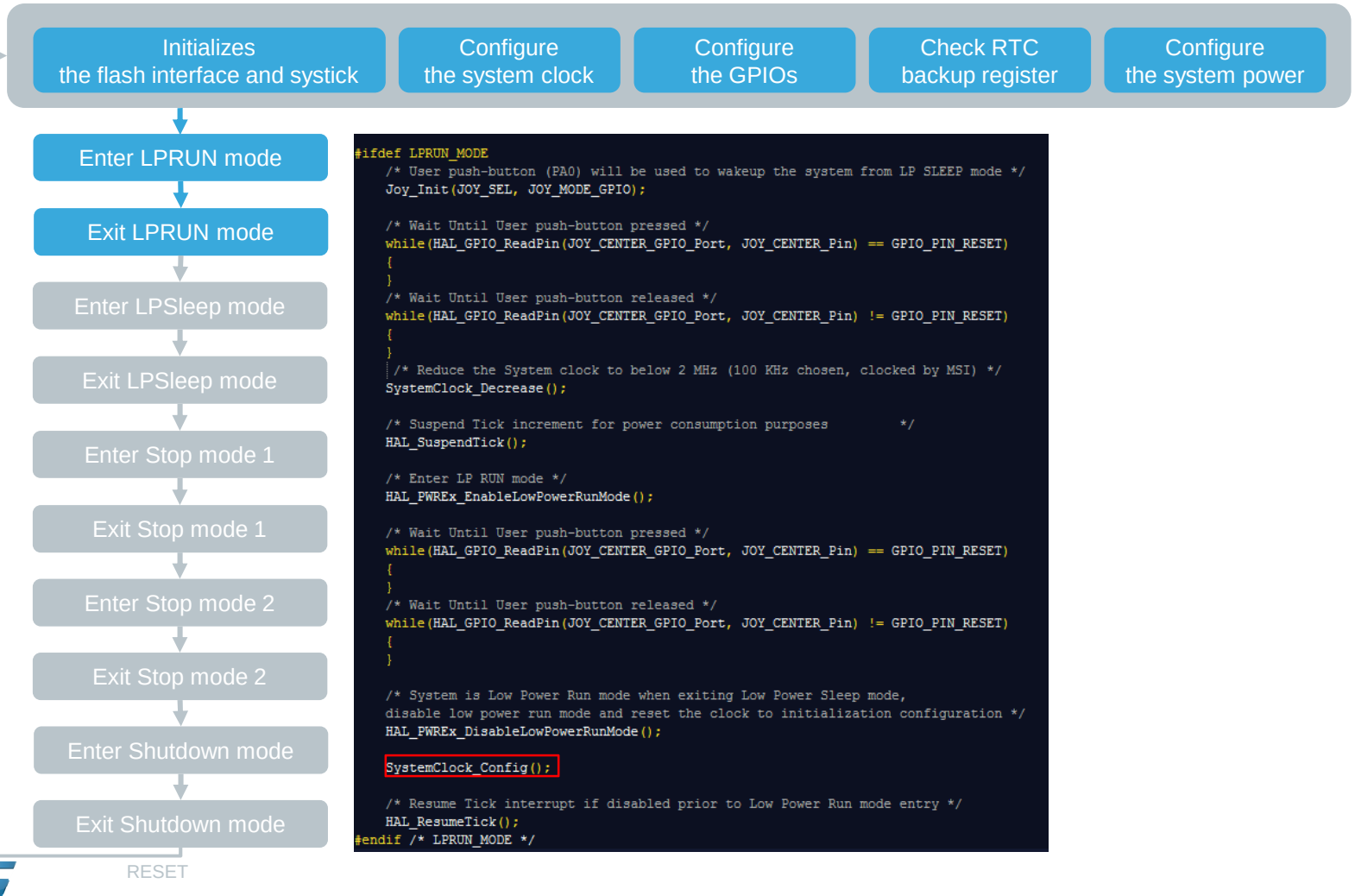
Implementation of Low Power Mode

Low Power Run Mode (LPRun mode)


```
#ifdef LPRUN_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_GPIO);

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }
    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment for power consumption purposes        */
    HAL_SuspendTick();

    /* Enter LP RUN mode */
    HAL_PWREx_EnableLowPowerRunMode();

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
    disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();

    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Run mode entry */
    HAL_ResumeTick();
#endif /* LPRUN_MODE */
```

Implementation of Low Power Mode

Low Power Run Mode (LPRun mode)




```
#ifdef LPRUN_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_GPIO);

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }
    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment for power consumption purposes        */
    HAL_SuspendTick();

    /* Enter LP RUN mode */
    HAL_PWREx_EnableLowPowerRunMode();

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
    disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();

    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Run mode entry */
    HAL_ResumeTick();
#endif /* LPRUN_MODE */
```

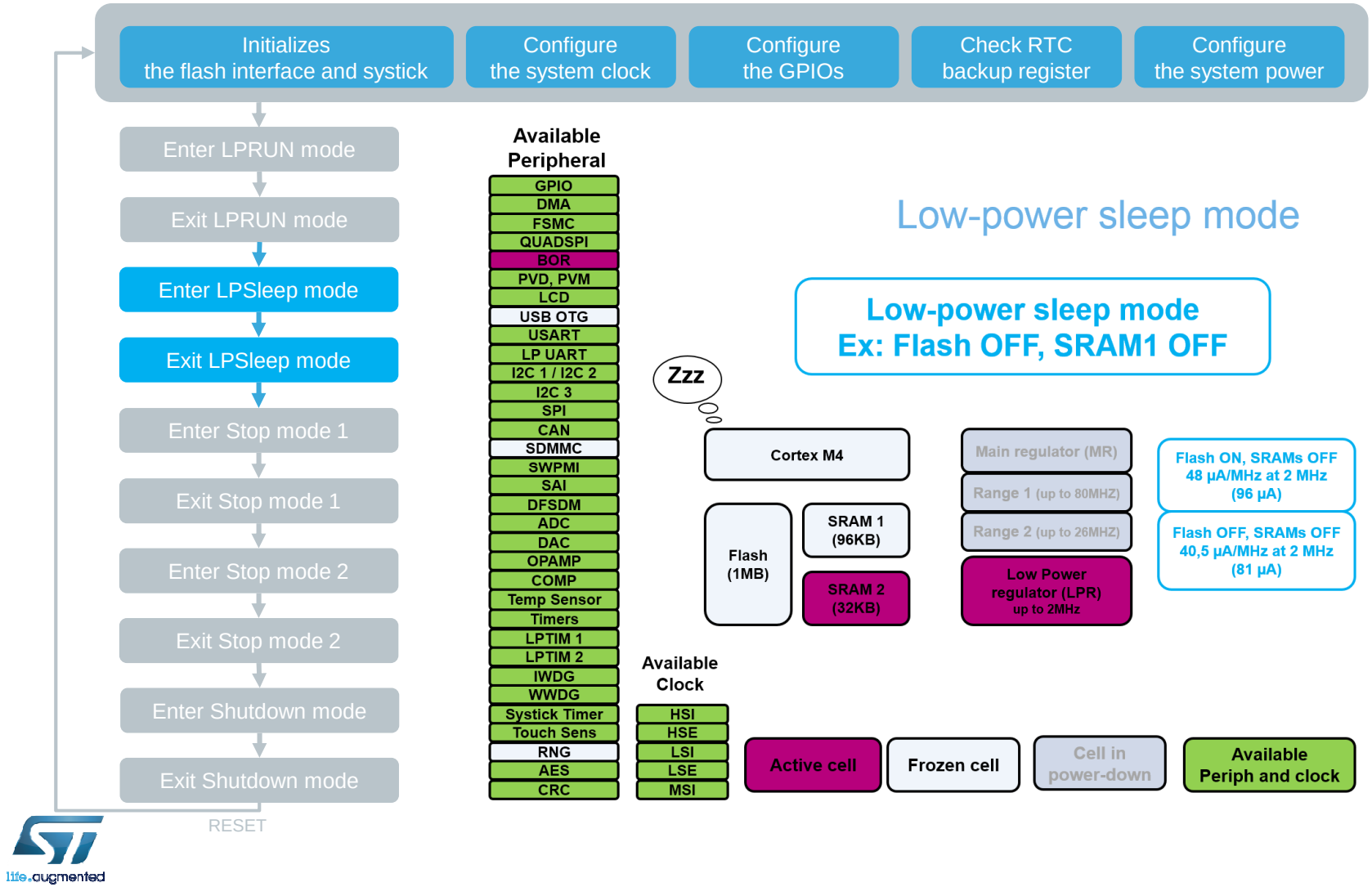
Implementation of Low Power Mode

Low Power Run Mode (LPRun mode)


```
#ifdef LPRUN_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_GPIO);

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }
    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment for power consumption purposes        */
    HAL_SuspendTick();

    /* Enter LP RUN mode */
    HAL_PWREx_EnableLowPowerRunMode();

    /* Wait Until User push-button pressed */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) == GPIO_PIN_RESET)
    {
    }
    /* Wait Until User push-button released */
    while(HAL_GPIO_ReadPin(JOY_CENTER_GPIO_Port, JOY_CENTER_Pin) != GPIO_PIN_RESET)
    {
    }

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
    disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();

    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Run mode entry */
    HAL_ResumeTick();
#endif /* LPRUN_MODE */
```

Implementation of Low Power Mode

Low Power Sleep Mode (LPSleep Mode)



Implementation of Low Power Mode

Low Power Sleep Mode (LPSleep Mode)


```
#ifdef LPSLEEP_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_EXTI);

    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment to prevent wakeup by Systick interrupt.          */
    /* Otherwise the Systick interrupt will wake up the device within 1ms      */
    /* (HAL time base).                                                        */
    HAL_SuspendTick();

    /* Enable Flash power down mode during Sleep mode     */
    __HAL_FLASH_SLEEP_POWERDOWN_ENABLE();

    /* Peripheral clock configuration in sleep and stop mode */
    PeripheralClock_Config();

    /* Enter Sleep Mode , wake up is done once Key push button is pressed */
    HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFI);

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
       disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();
    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Sleep mode entry */
    HAL_ResumeTick();
#endif /* LPSLEEP_MODE */
```

Implementation of Low Power Mode

Low Power Sleep Mode (LPSleep Mode)

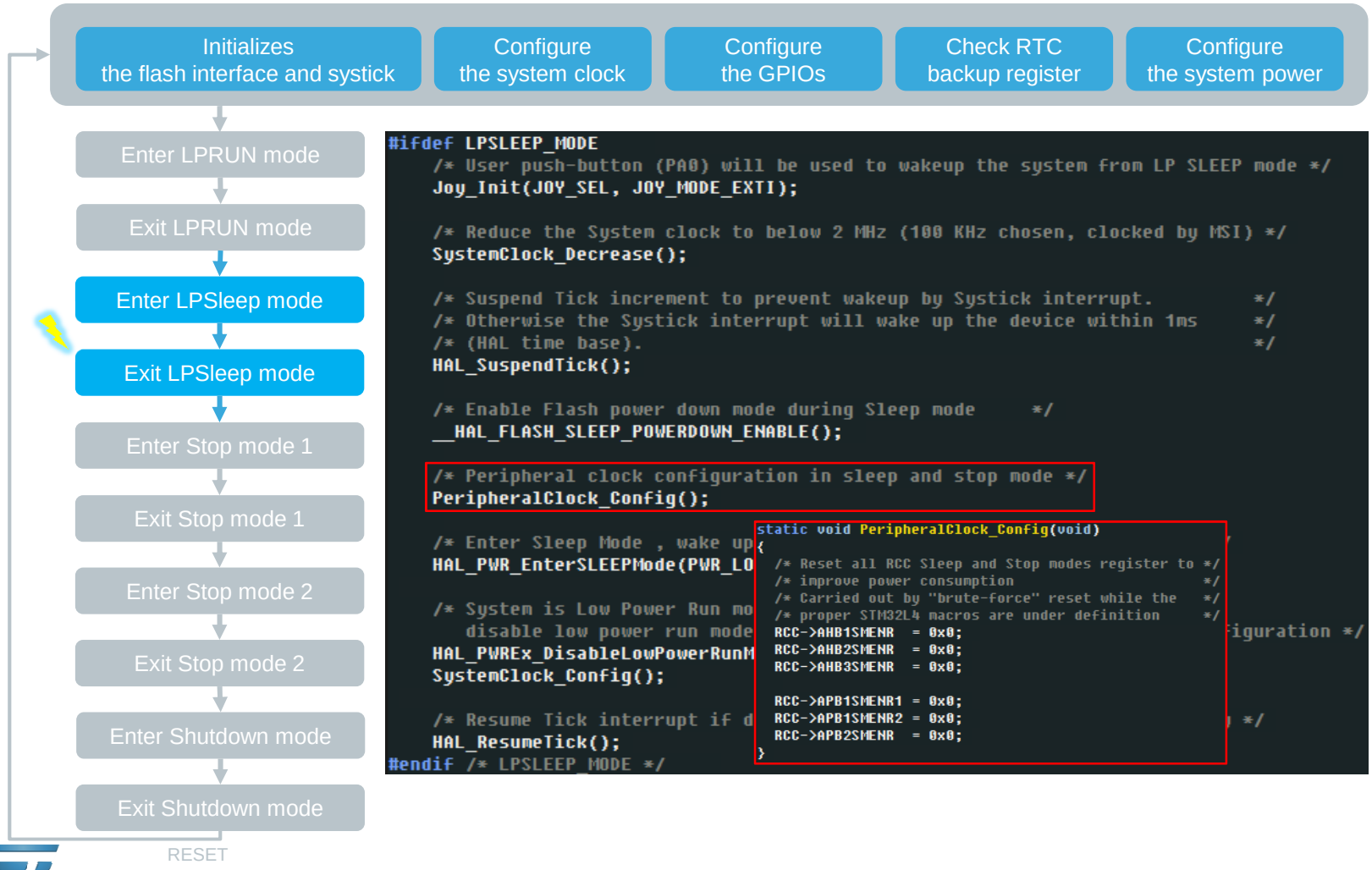

```
#ifdef LPSLEEP_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_EXTI);

    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment to prevent wakeup by Systick interrupt.          */
    /* Otherwise the Systick interrupt will wake up the device within 1ms      */
    /* (HAL time base).                                                        */
    HAL_SuspendTick();

    /* Enable Flash power down mode during Sleep mode      */
    __HAL_FLASH_SLEEP_POWERDOWN_ENABLE();

    /* Periphera
    PeripheralCl

    /* Enter Sle
    HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFI);

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
       disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();
    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Sleep mode entry */
    HAL_ResumeTick();
#endif /* LPSLEEP_MODE */
```

```
/**
  * @brief  Enable the FLASH power down during Low-Power sleep mode
  * @retval none
  */
#define __HAL_FLASH_SLEEP_POWERDOWN_ENABLE()    SET_BIT(FLASH->ACR, FLASH_ACR_SLEEP_PD)
```

life.augmented

Implementation of Low Power Mode

Low Power Sleep Mode (LPSleep Mode)




```
#ifdef LPSLEEP_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_EXTI);

    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment to prevent wakeup by Systick interrupt.          */
    /* Otherwise the Systick interrupt will wake up the device within 1ms      */
    /* (HAL time base).                                                        */
    HAL_SuspendTick();

    /* Enable Flash power down mode during Sleep mode      */
    __HAL_FLASH_SLEEP_POWERDOWN_ENABLE();

    /* Peripheral clock configuration in sleep and stop mode */
    PeripheralClock_Config();

    /* Enter Sleep Mode , wake up
    HAL_PWR_EnterSLEEPMode(PWR_LO

    /* System is Low Power Run mo
       disable low power run mode
    HAL_PWREx_DisableLowPowerRunM
    SystemClock_Config();

    /* Resume Tick interrupt if d
    HAL_ResumeTick();
#endif /* LPSLEEP_MODE */
```

```
static void PeripheralClock_Config(void)
{
  /* Reset all RCC Sleep and Stop modes register to */
  /* improve power consumption                      */
  /* Carried out by "brute-force" reset while the   */
  /* proper STM32L4 macros are under definition     */
  RCC->AHB1SMENR  = 0x0;
  RCC->AHB2SMENR  = 0x0;
  RCC->AHB3SMENR  = 0x0;

  RCC->APB1SMENR1 = 0x0;
  RCC->APB1SMENR2 = 0x0;
  RCC->APB2SMENR  = 0x0;
}
```

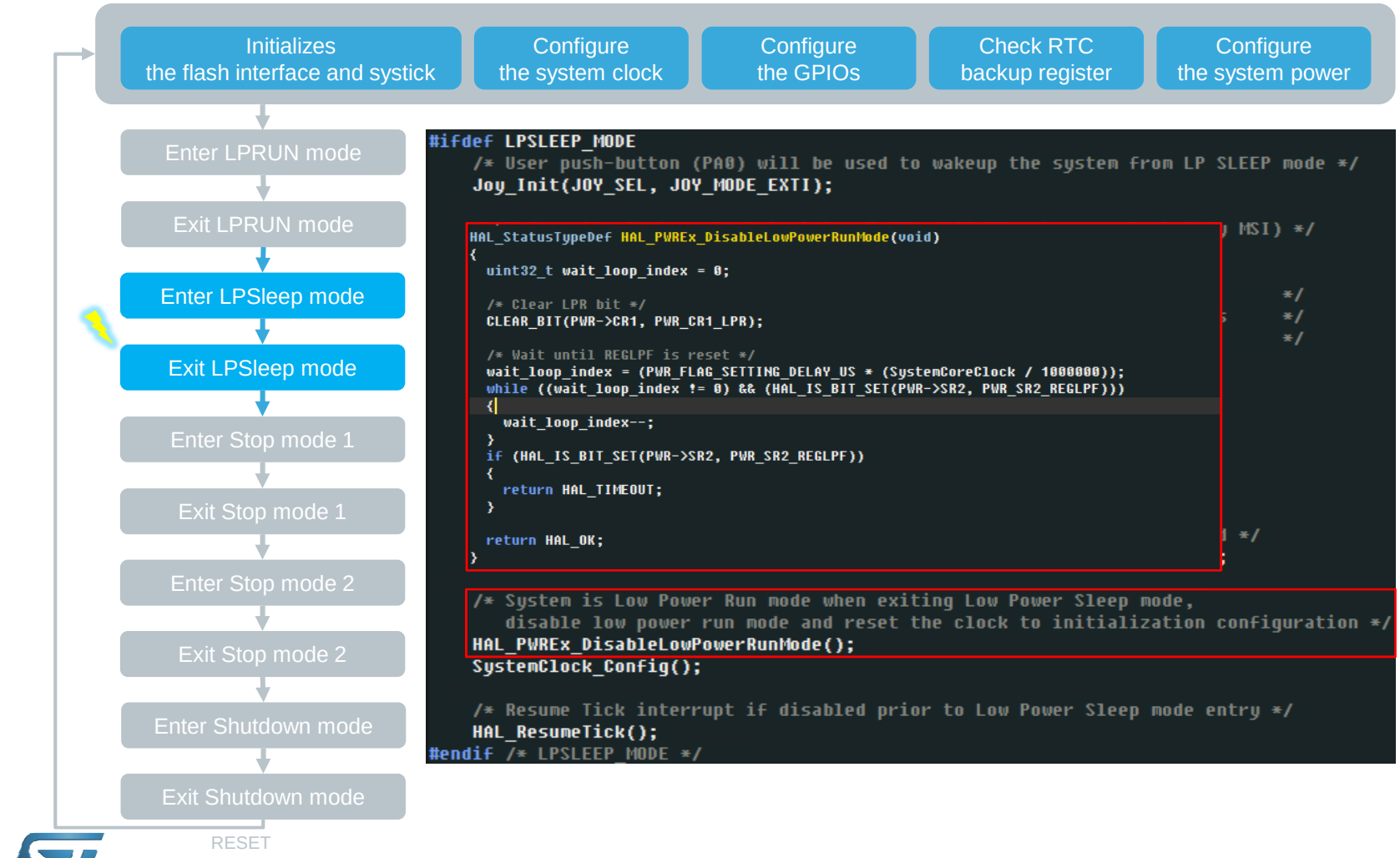
Implementation of Low Power Mode

Low Power Sleep Mode (LPSleep Mode)


```
#ifdef LPSLEEP_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_EXTI);

    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment to prevent wakeup by Systick interrupt.          */
    /* Otherwise the Systick interrupt will wake up the device within 1ms      */
    /* (HAL time base).                                                        */
    HAL_SuspendTick();

    /* Enable Flash power down mode during Sleep mode     */
    __HAL_FLASH_SLEEP_POWERDOWN_ENABLE();

    /* Peripheral clock configuration in sleep and stop mode */
    PeripheralClock_Config();

    /* Enter Sleep Mode , wake up is done once Key push button is pressed */
    HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFI);

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
       disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();
    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Sleep mode entry */
    HAL_ResumeTick();
#endif /* LPSLEEP_MODE */
```

Implementation of Low Power Mode

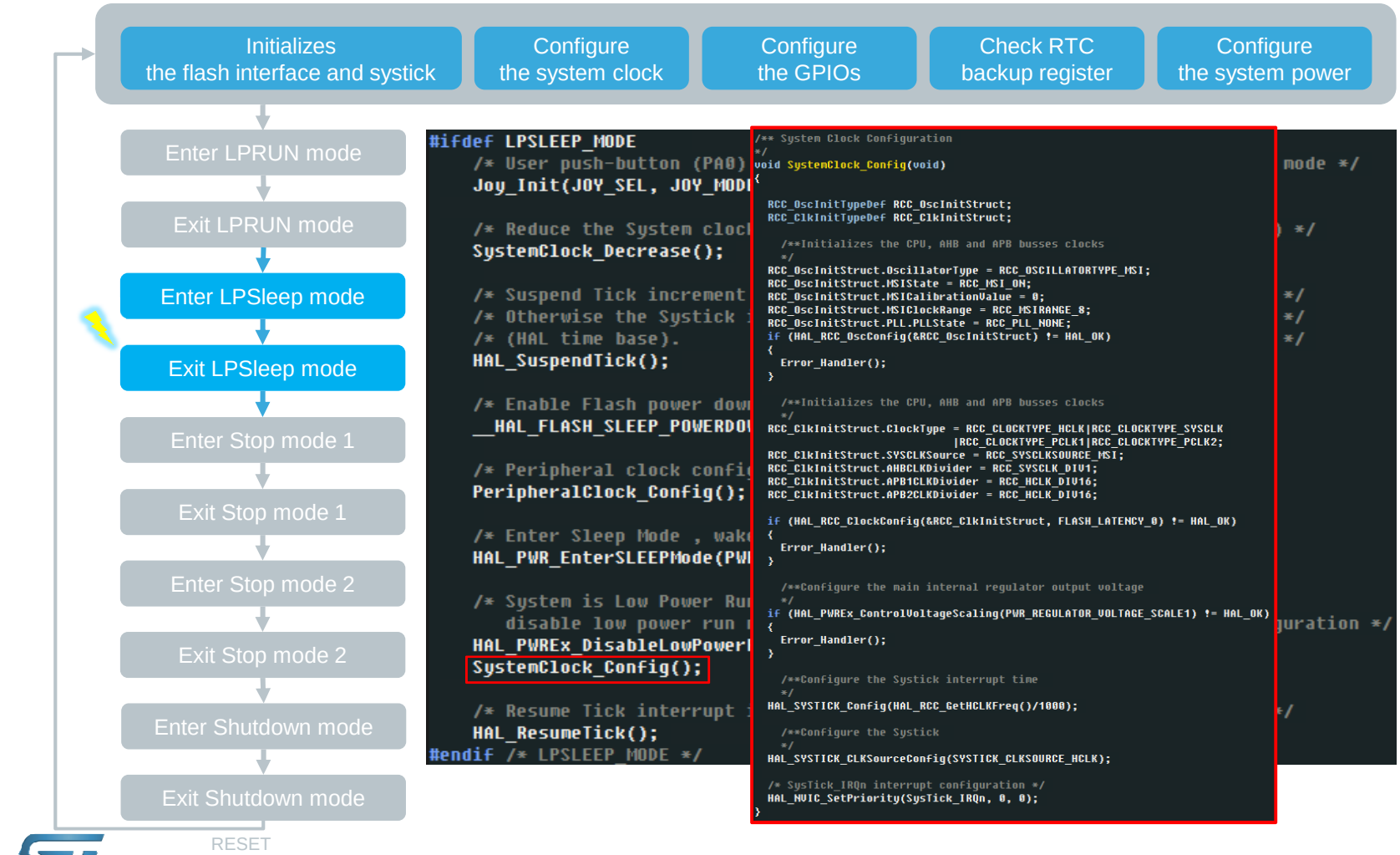
Low Power Sleep Mode (LPSleep Mode)




```
#ifdef LPSLEEP_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_EXTI);

HAL_StatusTypeDef HAL_PWREx_DisableLowPowerRunMode(void)
{
  uint32_t wait_loop_index = 0;

  /* Clear LPR bit */
  CLEAR_BIT(PWR->CR1, PWR_CR1_LPR);

  /* Wait until REGLPF is reset */
  wait_loop_index = (PWR_FLAG_SETTING_DELAY_US * (SystemCoreClock / 1000000));
  while ((wait_loop_index != 0) && (HAL_IS_BIT_SET(PWR->SR2, PWR_SR2_REGLPF)))
  {
    wait_loop_index--;
  }
  if (HAL_IS_BIT_SET(PWR->SR2, PWR_SR2_REGLPF))
  {
    return HAL_TIMEOUT;
  }

  return HAL_OK;
}

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
       disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();
    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Sleep mode entry */
    HAL_ResumeTick();
#endif /* LPSLEEP_MODE */
```

Implementation of Low Power Mode

Low Power Sleep Mode (LPSleep Mode)



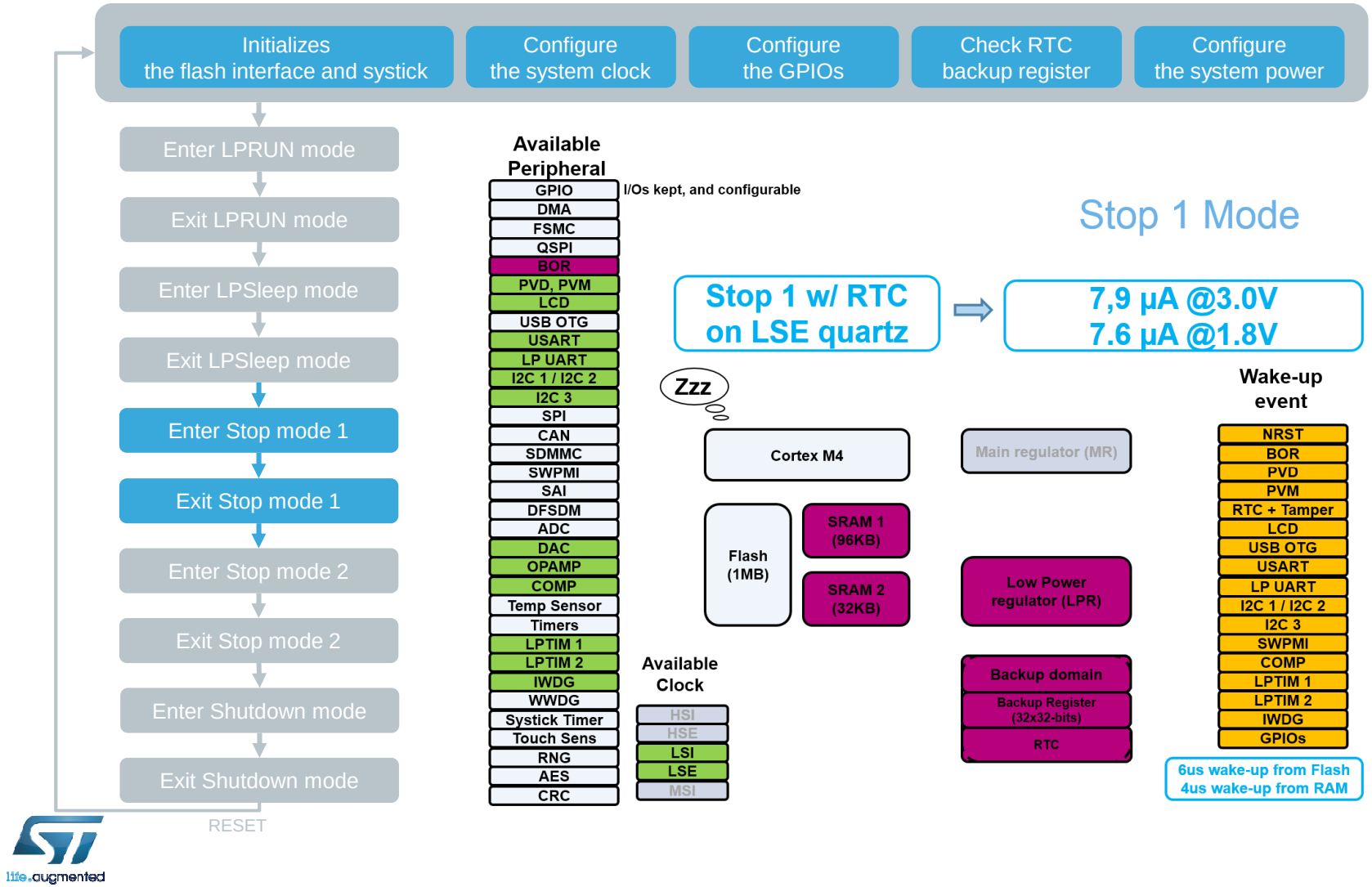
Implementation of Low Power Mode

Low Power Sleep Mode (LPSleep Mode)


```
#ifdef LPSLEEP_MODE
    /* User push-button (PA0) will be used to wakeup the system from LP SLEEP mode */
    Joy_Init(JOY_SEL, JOY_MODE_EXTI);

    /* Reduce the System clock to below 2 MHz (100 KHz chosen, clocked by MSI) */
    SystemClock_Decrease();

    /* Suspend Tick increment to prevent wakeup by Systick interrupt.          */
    /* Otherwise the Systick interrupt will wake up the device within 1ms      */
    /* (HAL time base).                                                        */
    HAL_SuspendTick();

    /* Enable Flash power down mode during Sleep mode     */
    __HAL_FLASH_SLEEP_POWERDOWN_ENABLE();

    /* Peripheral clock configuration in sleep and stop mode */
    PeripheralClock_Config();

    /* Enter Sleep Mode , wake up is done once Key push button is pressed */
    HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFI);

    /* System is Low Power Run mode when exiting Low Power Sleep mode,
       disable low power run mode and reset the clock to initialization configuration */
    HAL_PWREx_DisableLowPowerRunMode();
    SystemClock_Config();

    /* Resume Tick interrupt if disabled prior to Low Power Sleep mode entry */
    HAL_ResumeTick();
#endif /* LPSLEEP_MODE */
```

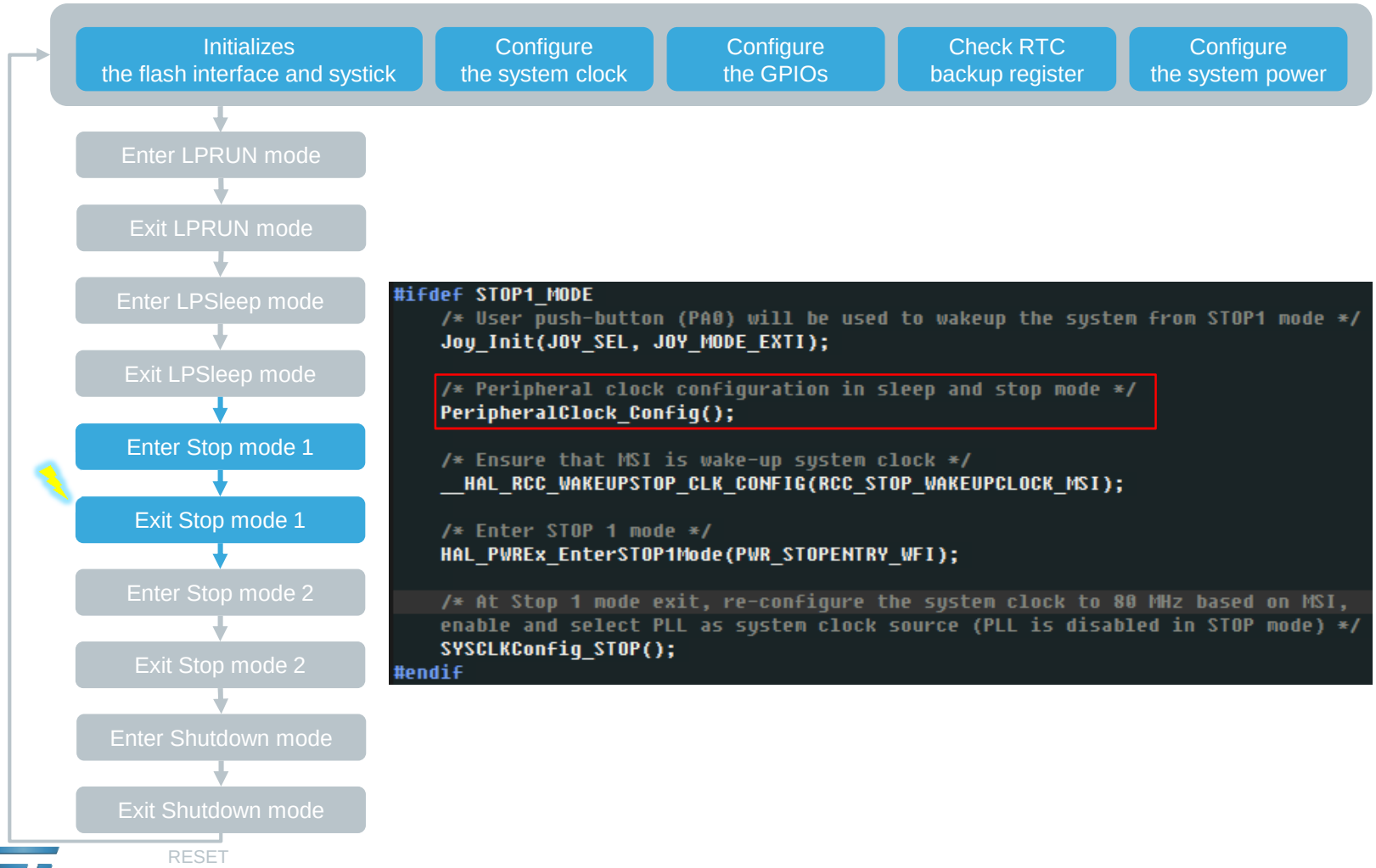
Implementation of Low Power Mode

STOP1 Mode



Implementation of Low Power Mode

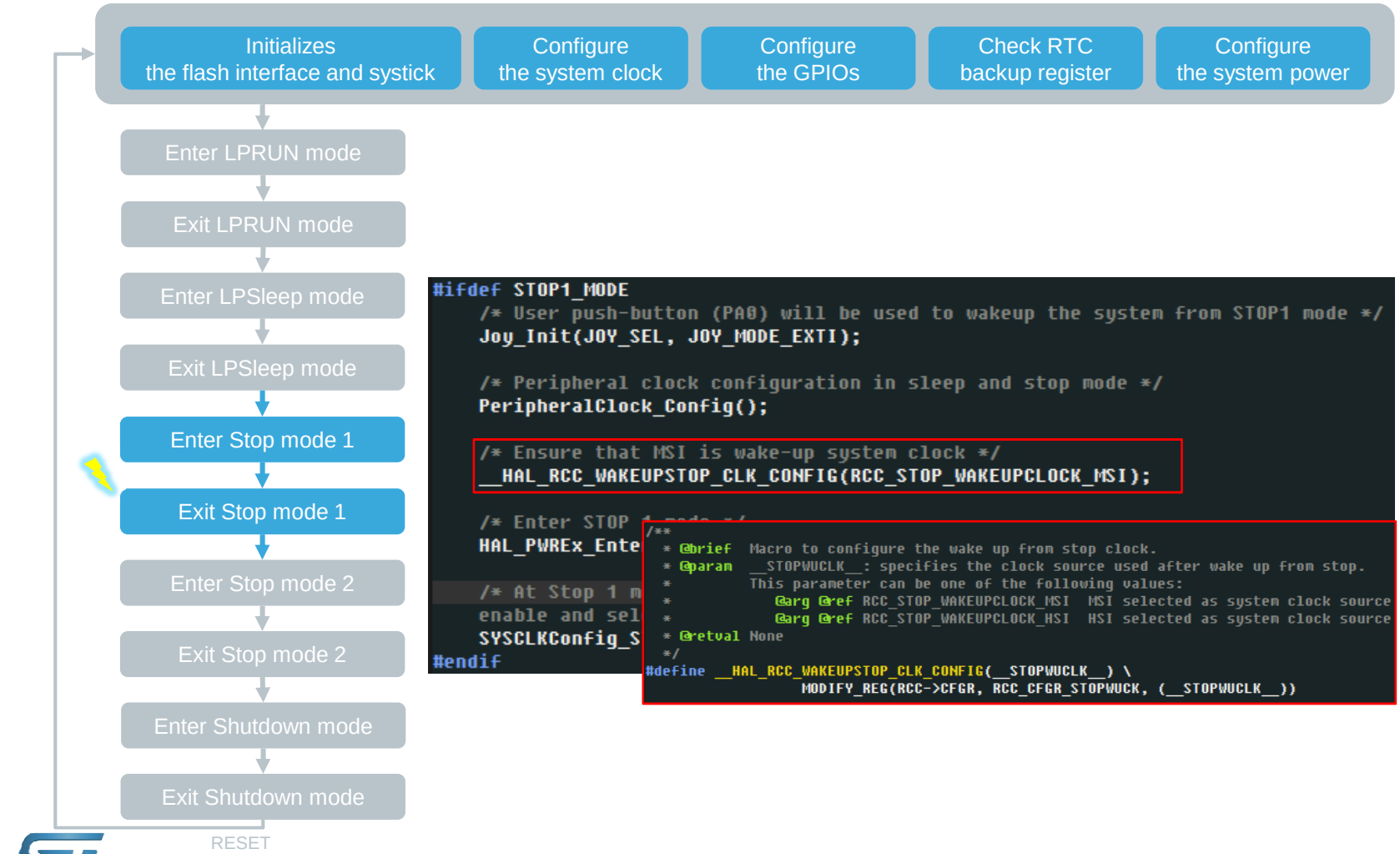
STOP1 Mode




```
#ifdef STOP1_MODE
    /* User push-button (PA0) will be used to wakeup the system from STOP1 mode */
    Joy_Init(JOY_SEL, JOY_MODE_EXTI);

    /* Peripheral clock configuration in sleep and stop mode */
    PeripheralClock_Config();

    /* Ensure that MSI is wake-up system clock */
    __HAL_RCC_WAKEUPSTOP_CLK_CONFIG(RCC_STOP_WAKEUPCLOCK_MSI);

    /* Enter STOP 1 mode */
    HAL_PWREx_EnterSTOP1Mode(PWR_STOPENTRY_WFI);

    /* At Stop 1 mode exit, re-configure the system clock to 80 MHz based on MSI,
    enable and select PLL as system clock source (PLL is disabled in STOP mode) */
    SYSCLKConfig_STOP();
#endif
```

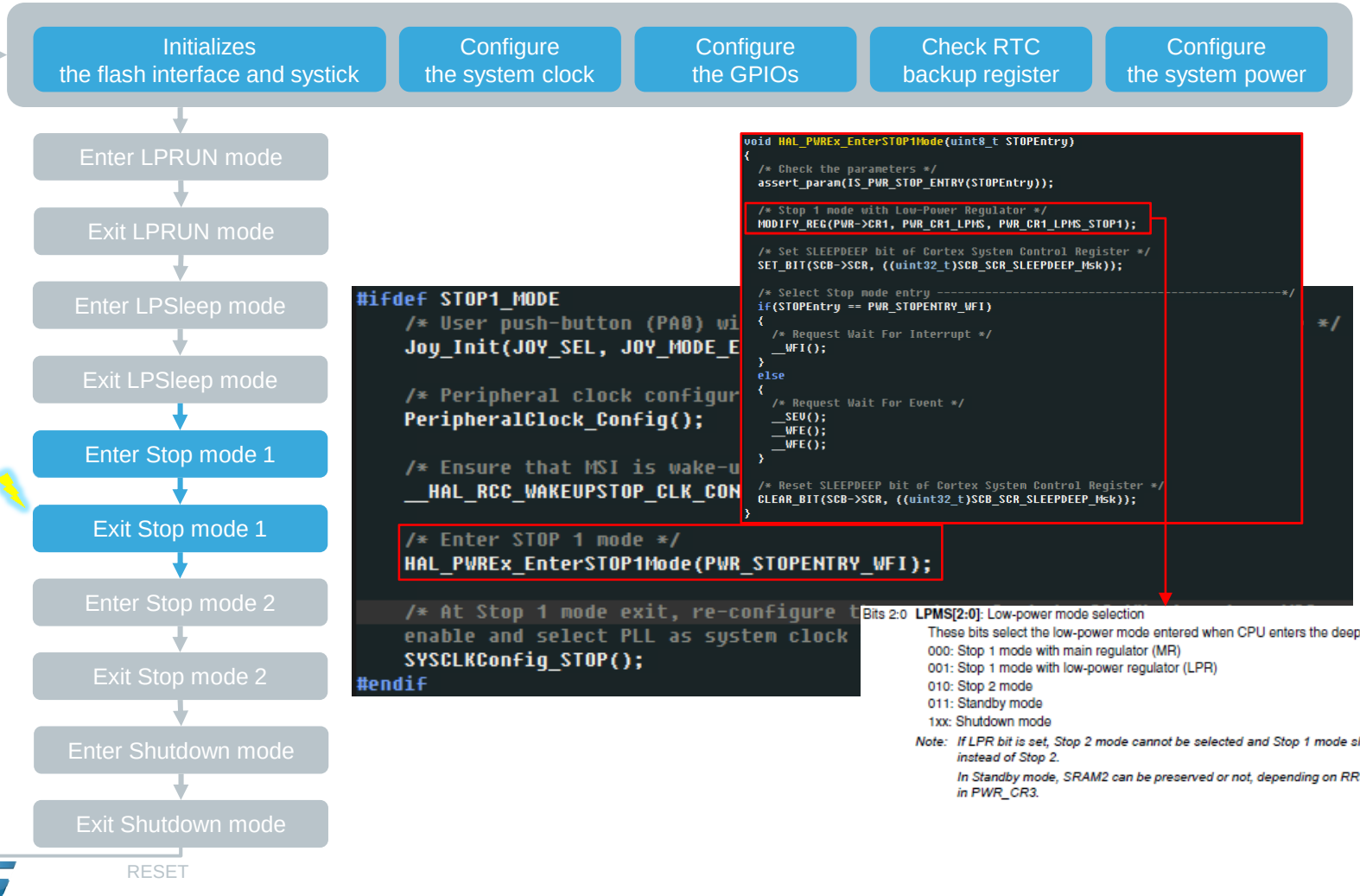
Implementation of Low Power Mode

STOP1 Mode



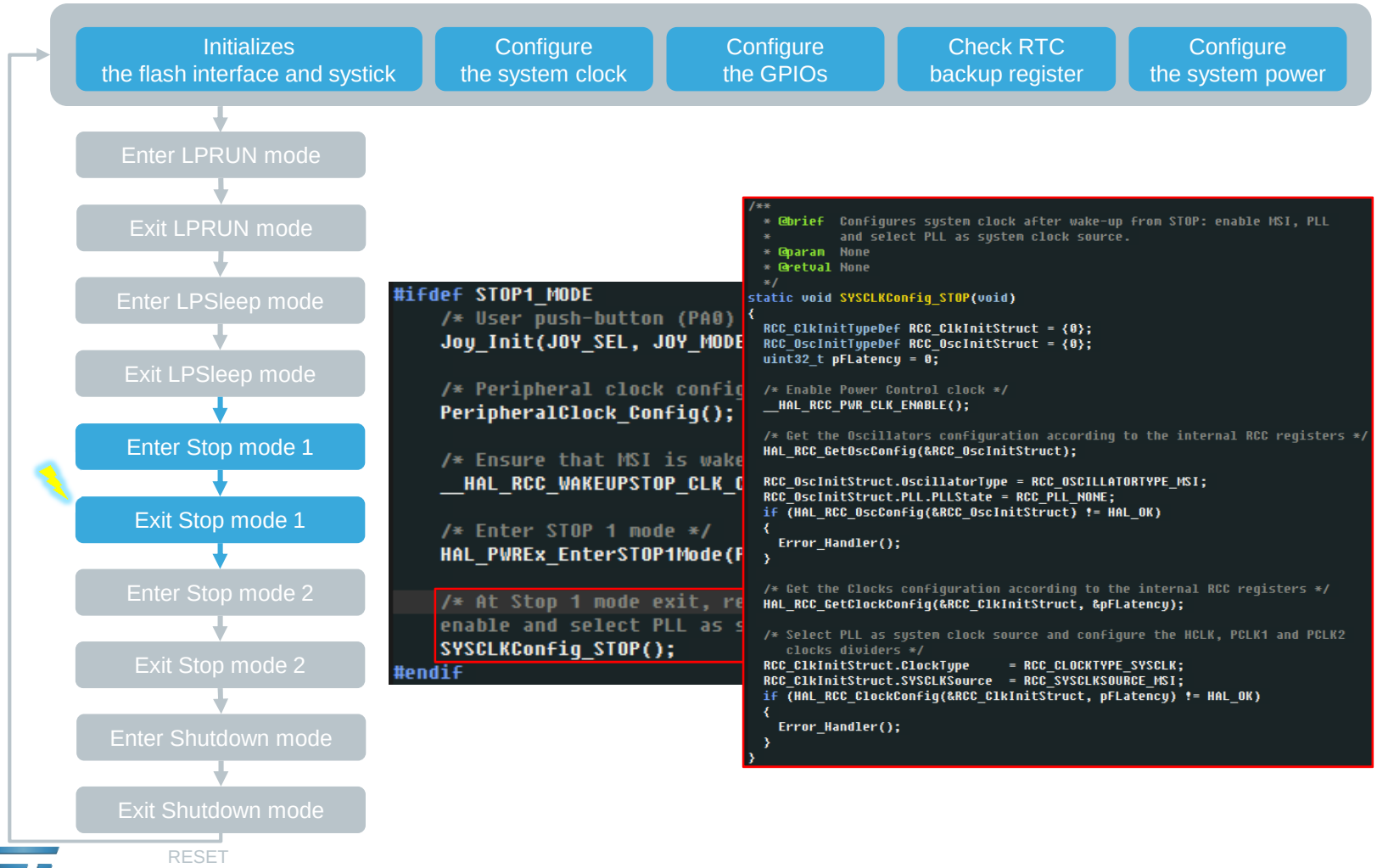
Implementation of Low Power Mode

STOP1 Mode



Implementation of Low Power Mode

STOP1 Mode

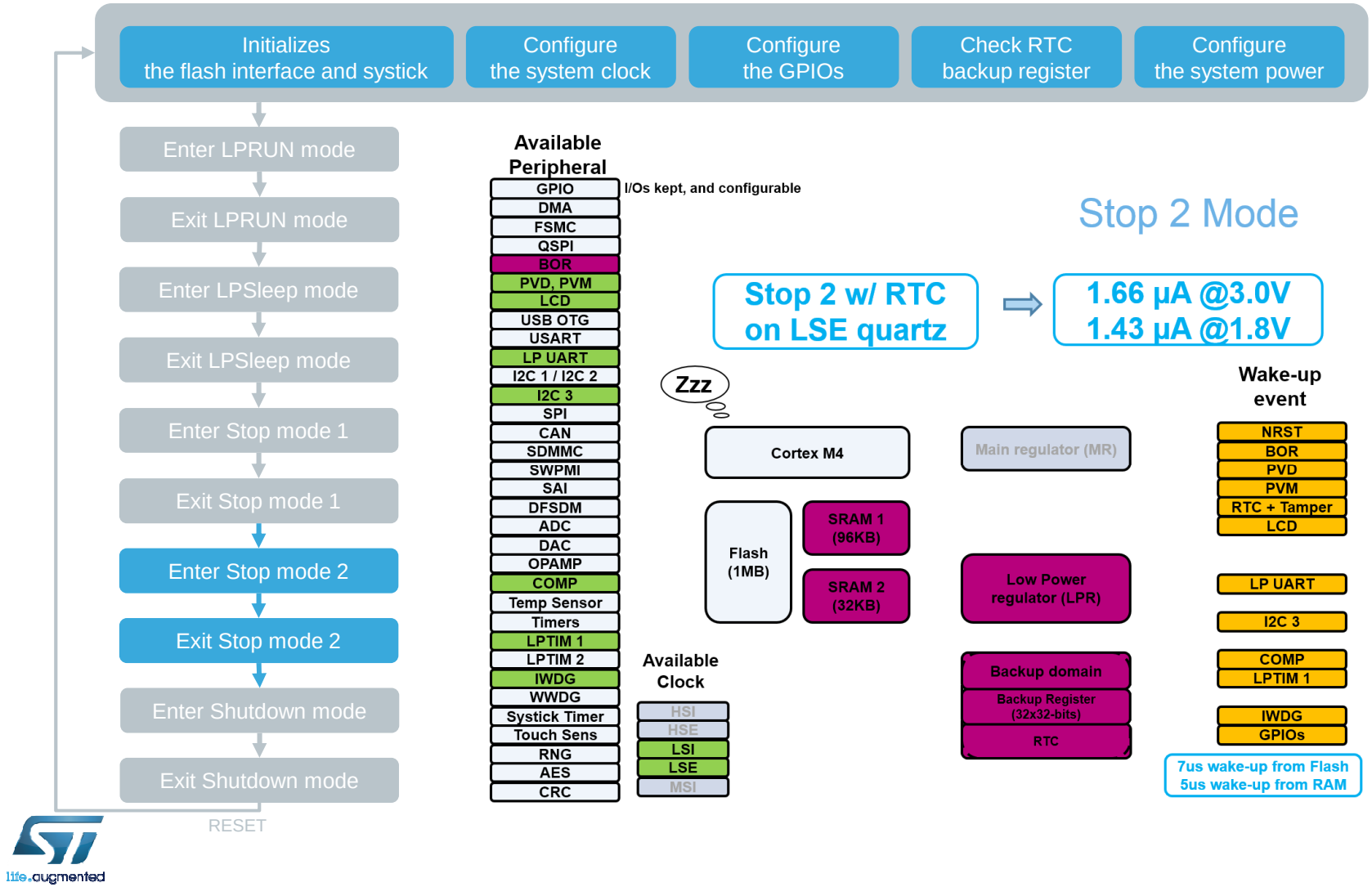



```
#ifdef STOP1_MODE
    /* User push-button (PA0)
    Joy_Init(JOY_SEL, JOY_MODE

    /* Peripheral clock config
    PeripheralClock_Config();

    /* Ensure that MSI is wake
    __HAL_RCC_WAKEUPSTOP_CLK_C

    /* Enter STOP 1 mode */
    HAL_PWREx_EnterSTOP1Mode(P

    /* At Stop 1 mode exit, re
    enable and select PLL as s
    SYSCLKConfig_STOP();
#endif
```

```
/**
  * @brief  Configures system clock after wake-up from STOP: enable MSI, PLL
  *         and select PLL as system clock source.
  * @param  None
  * @retval None
  */
static void SYSCLKConfig_STOP(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  uint32_t pFLatency = 0;

  /* Enable Power Control clock */
  __HAL_RCC_PWR_CLK_ENABLE();

  /* Get the Oscillators configuration according to the internal RCC registers */
  HAL_RCC_GetOscConfig(&RCC_OscInitStruct);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /* Get the Clocks configuration according to the internal RCC registers */
  HAL_RCC_GetClockConfig(&RCC_ClkInitStruct, &pFLatency);

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
     clocks dividers */
  RCC_ClkInitStruct.ClockType     = RCC_CLOCKTYPE_SYSCLK;
  RCC_ClkInitStruct.SYSCLKSource  = RCC_SYSCLKSOURCE_MSI;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, pFLatency) != HAL_OK)
  {
    Error_Handler();
  }
}
```

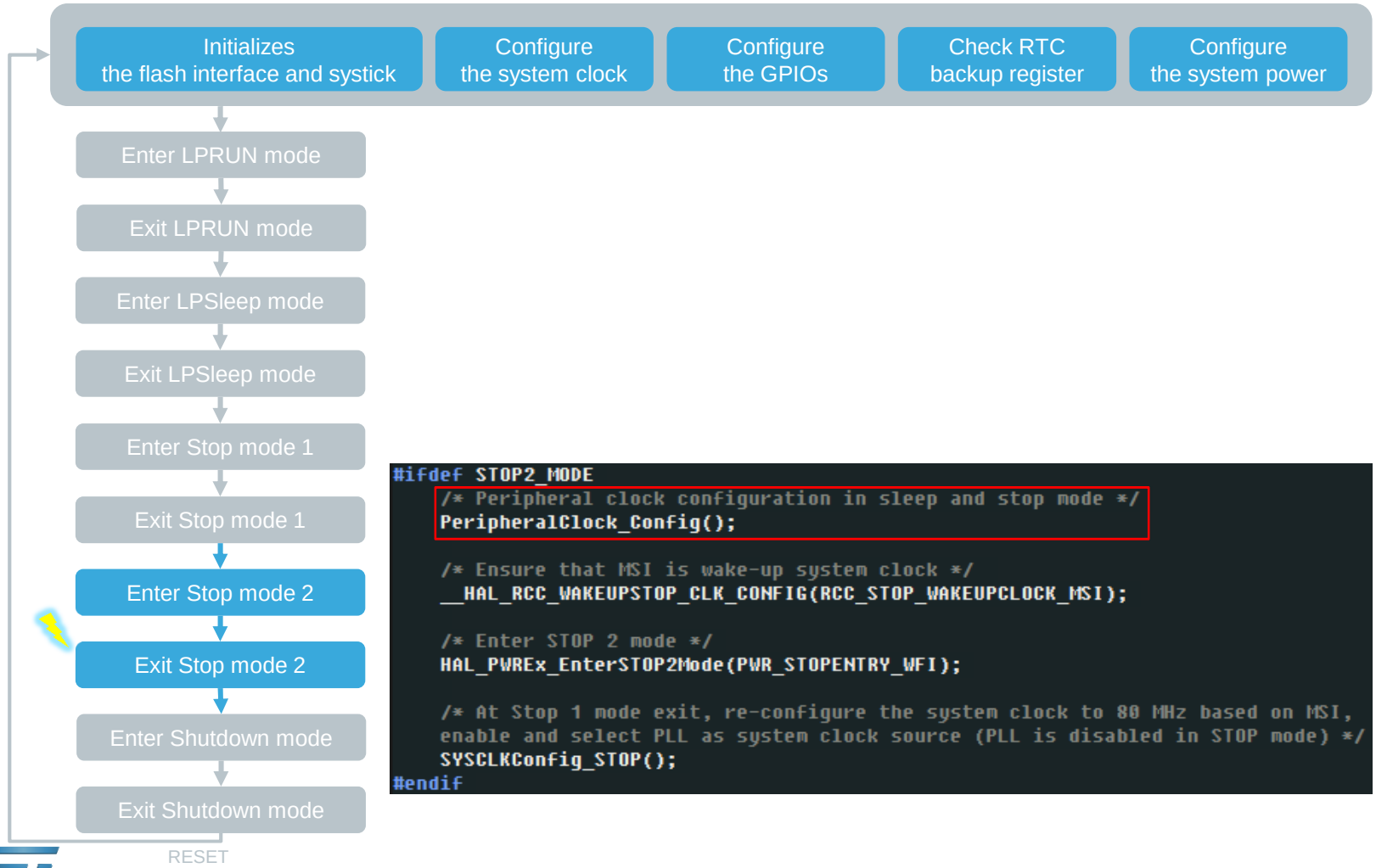
Implementation of Low Power Mode

STOP2 Mode



Implementation of Low Power Mode

STOP2 Mode




```
#ifdef STOP2_MODE
    /* Peripheral clock configuration in sleep and stop mode */
    PeripheralClock_Config();

    /* Ensure that MSI is wake-up system clock */
    __HAL_RCC_WAKEUPSTOP_CLK_CONFIG(RCC_STOP_WAKEUPCLOCK_MSI);

    /* Enter STOP 2 mode */
    HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);

    /* At Stop 1 mode exit, re-configure the system clock to 80 MHz based on MSI,
    enable and select PLL as system clock source (PLL is disabled in STOP mode) */
    SYSCLKConfig_STOP();
#endif
```

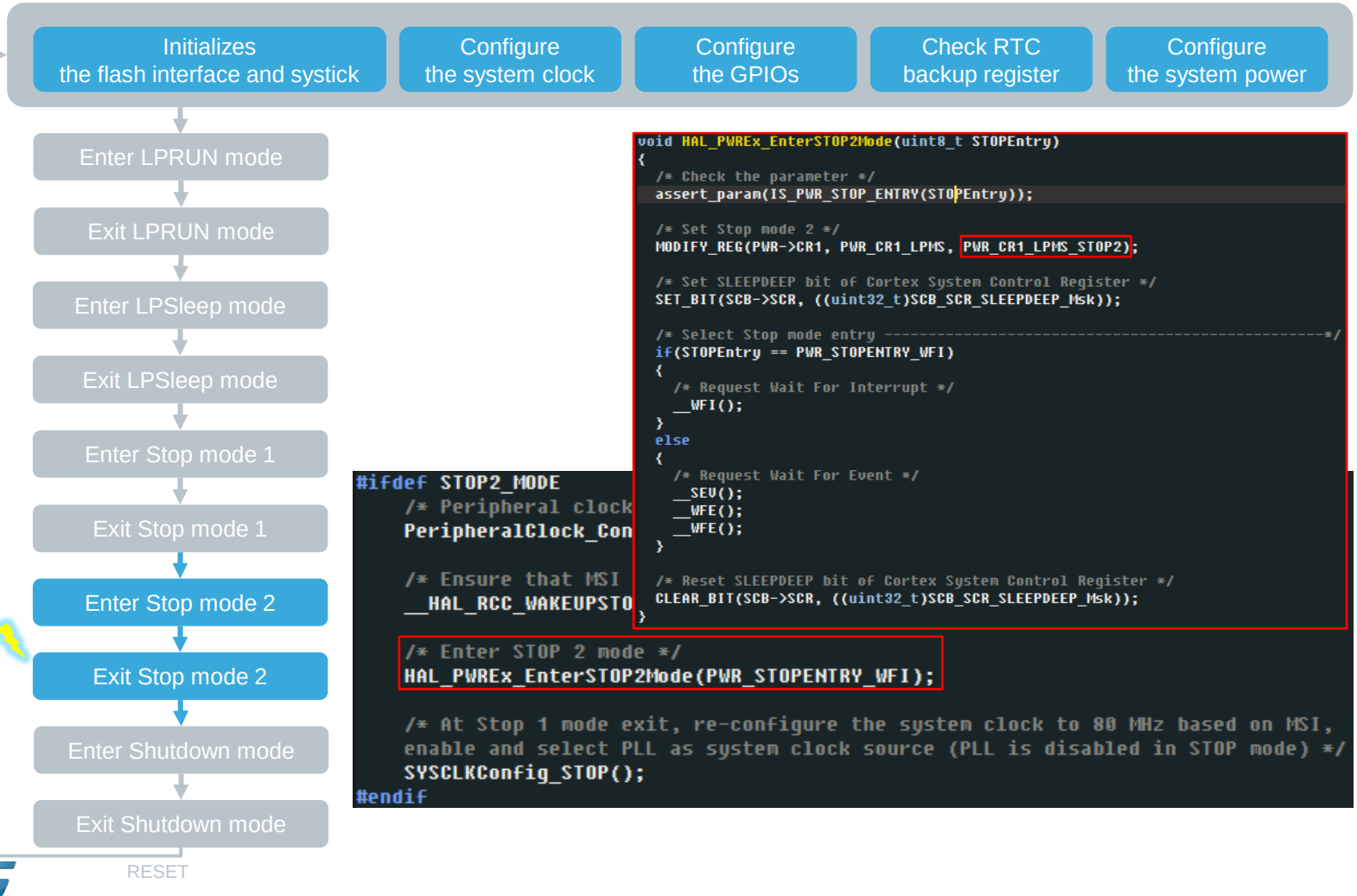
Implementation of Low Power Mode

STOP2 Mode


```
#ifdef STOP2_MODE
    /* Peripheral clock configuration in sleep and stop mode */
    PeripheralClock_Config();

    /* Ensure that MSI is wake-up system clock */
    __HAL_RCC_WAKEUPSTOP_CLK_CONFIG(RCC_STOP_WAKEUPCLOCK_MSI);

    /* Enter STOP 2 mode */
    HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);

    /* At Stop 1 mode exit, re-configure the system clock to 80 MHz based on MSI,
    enable and select PLL as system clock source (PLL is disabled in STOP mode) */
    SYSCLKConfig_STOP();
#endif
```

Implementation of Low Power Mode

STOP2 Mode




```
void HAL_PWREx_EnterSTOP2Mode(uint8_t STOPEntry)
{
  /* Check the parameter */
  assert_param(IS_PWR_STOP_ENTRY(STOPEntry));

  /* Set Stop mode 2 */
  MODIFY_REG(PWR->CR1, PWR_CR1_LPMS, PWR_CR1_LPMS_STOP2);

  /* Set SLEEPDEEP bit of Cortex System Control Register */
  SET_BIT(SCB->SCR, ((uint32_t)SCB_SCR_SLEEPDEEP_Msk));

  /* Select Stop mode entry --------------------------------------------------*/
  if(STOPEntry == PWR_STOPENTRY_WFI)
  {
    /* Request Wait For Interrupt */
    __WFI();
  }
  else
  {
    /* Request Wait For Event */
    __SEV();
    __WFE();
    __WFE();
  }

  /* Reset SLEEPDEEP bit of Cortex System Control Register */
  CLEAR_BIT(SCB->SCR, ((uint32_t)SCB_SCR_SLEEPDEEP_Msk));
}
```

```
#ifdef STOP2_MODE
    /* Peripheral clock
    PeripheralClock_Con

    /* Ensure that MSI
    __HAL_RCC_WAKEUPSTO

    /* Enter STOP 2 mode */
    HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);

    /* At Stop 1 mode exit, re-configure the system clock to 80 MHz based on MSI,
    enable and select PLL as system clock source (PLL is disabled in STOP mode) */
    SYSCLKConfig_STOP();
#endif
```

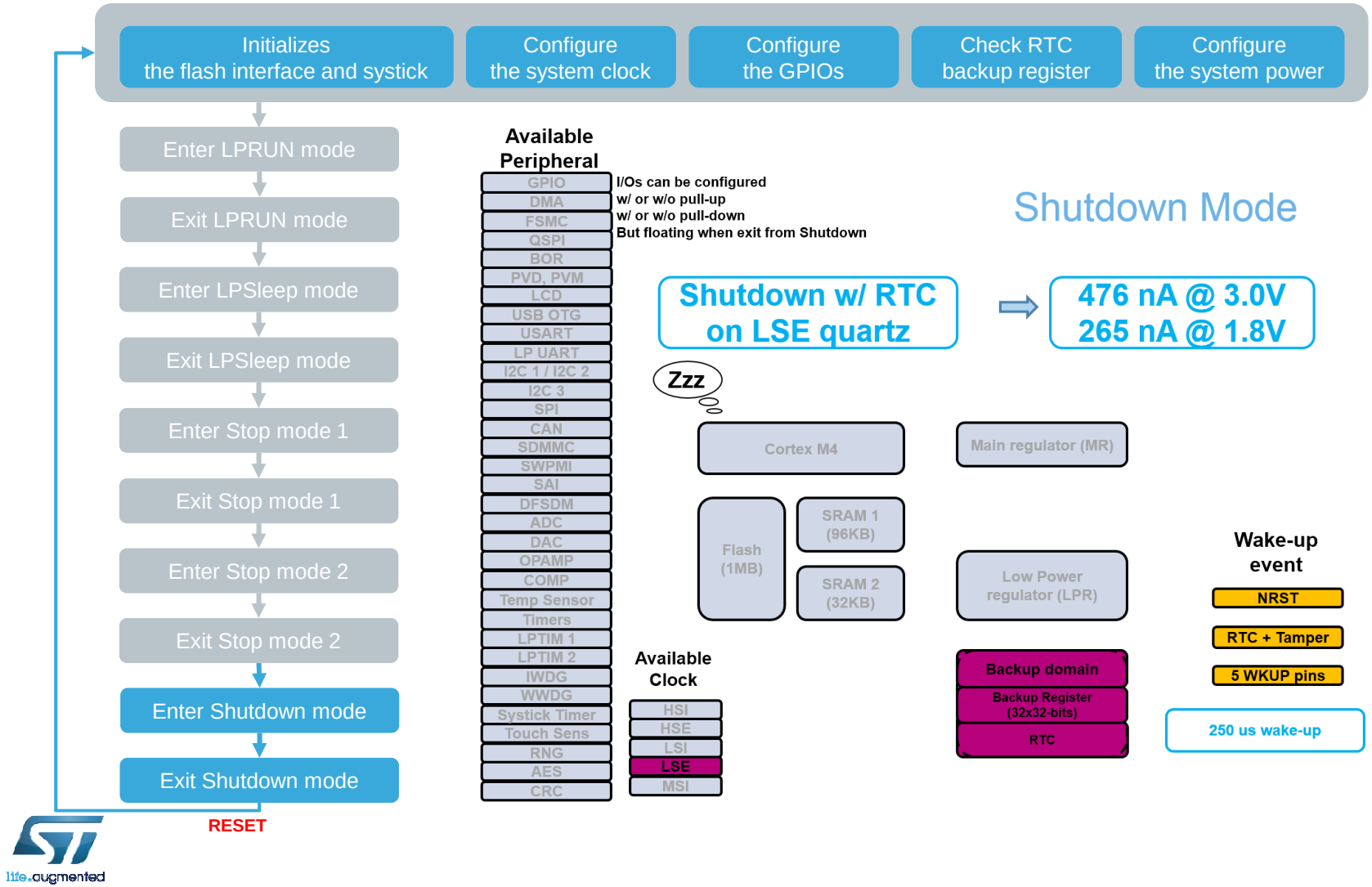
Implementation of Low Power Mode

STOP2 Mode


```
#ifdef STOP2_MODE
    /* Peripheral clock configuration in sleep and stop mode */
    PeripheralClock_Config();

    /* Ensure that MSI is wake-up system clock */
    __HAL_RCC_WAKEUPSTOP_CLK_CONFIG(RCC_STOP_WAKEUPCLOCK_MSI);

    /* Enter STOP 2 mode */
    HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);

    /* At Stop 1 mode exit, re-configure the system clock to 80 MHz based on MSI,
    enable and select PLL as system clock source (PLL is disabled in STOP mode) */
    SYSCLKConfig_STOP();
#endif
```

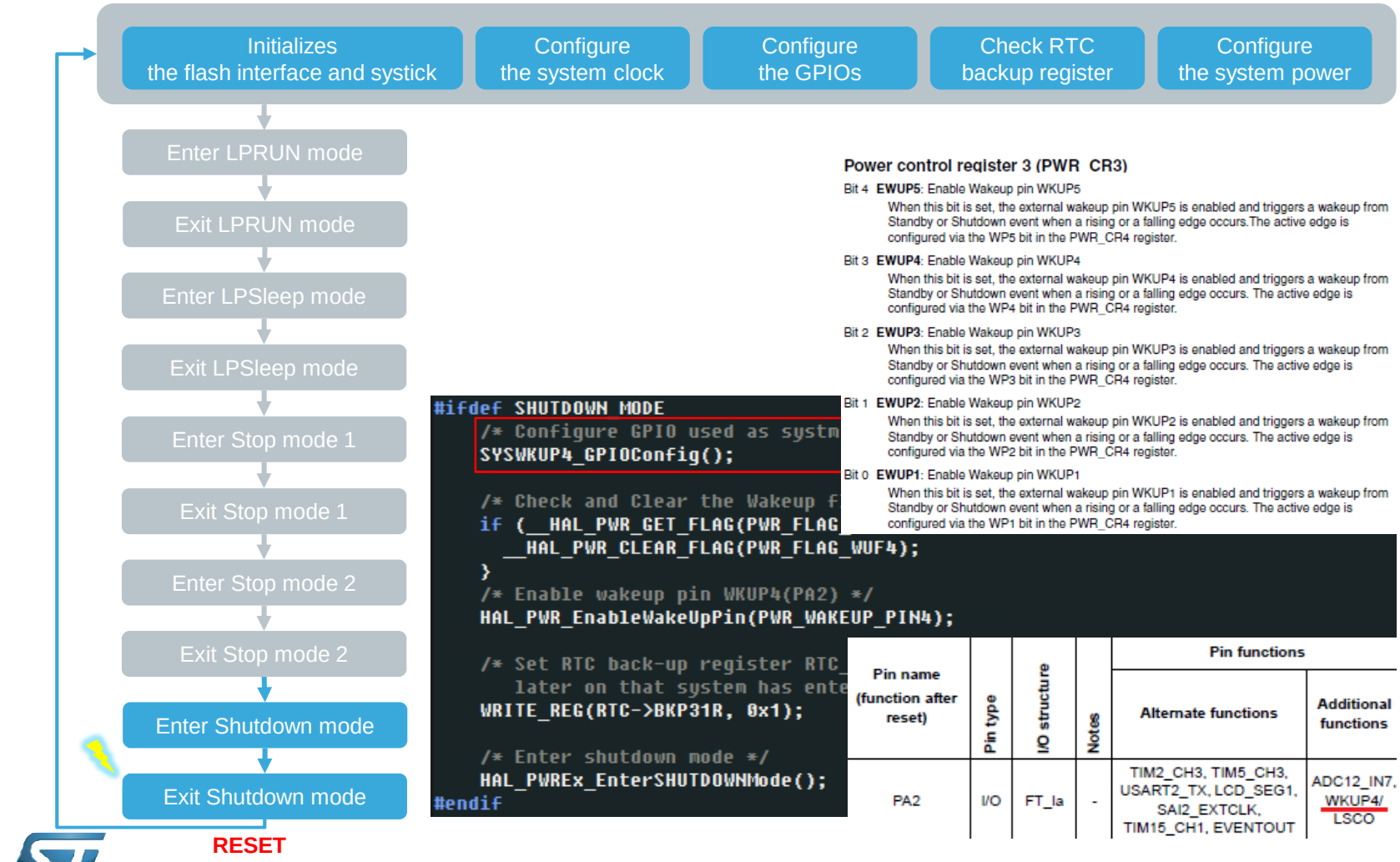
Implementation of Low Power Mode

Shutdown Mode



Implementation of Low Power Mode

Shutdown Mode




```
#ifdef SHUTDOWN_MODE
    /* Configure GPIO used as systm
    SYSWKUP4_GPIOConfig();

    /* Check and Clear the Wakeup f
    if (__HAL_PWR_GET_FLAG(PWR_FLAG
      __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WUF4);
    }
    /* Enable wakeup pin WKUP4(PA2) */
    HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN4);

    /* Set RTC back-up register RTC_
       later on that system has ente
    WRITE_REG(RTC->BKP31R, 0x1);

    /* Enter shutdown mode */
    HAL_PWREx_EnterSHUTDOWNMode();
#endif
```

Power control register 3 (PWR_CR3)

Bit 4 **EWUP5**: Enable Wakeup pin WKUP5
When this bit is set, the external wakeup pin WKUP5 is enabled and triggers a wakeup from Standby or Shutdown event when a rising or a falling edge occurs.The active edge is configured via the WP5 bit in the PWR_CR4 register.

Bit 3 **EWUP4**: Enable Wakeup pin WKUP4
When this bit is set, the external wakeup pin WKUP4 is enabled and triggers a wakeup from Standby or Shutdown event when a rising or a falling edge occurs. The active edge is configured via the WP4 bit in the PWR_CR4 register.

Bit 2 **EWUP3**: Enable Wakeup pin WKUP3
When this bit is set, the external wakeup pin WKUP3 is enabled and triggers a wakeup from Standby or Shutdown event when a rising or a falling edge occurs. The active edge is configured via the WP3 bit in the PWR_CR4 register.

Bit 1 **EWUP2**: Enable Wakeup pin WKUP2
When this bit is set, the external wakeup pin WKUP2 is enabled and triggers a wakeup from Standby or Shutdown event when a rising or a falling edge occurs. The active edge is configured via the WP2 bit in the PWR_CR4 register.

Bit 0 **EWUP1**: Enable Wakeup pin WKUP1
When this bit is set, the external wakeup pin WKUP1 is enabled and triggers a wakeup from Standby or Shutdown event when a rising or a falling edge occurs. The active edge is configured via the WP1 bit in the PWR_CR4 register.

Pin name (function after reset)	Pin type	I/O structure	Notes	Pin functions	
				Alternate functions	Additional functions
PA2	I/O	FT_la	-	TIM2_CH3, TIM5_CH3, USART2_TX, LCD_SEG1, SAI2_EXTCLK, TIM15_CH1, EVENTOUT	ADC12_IN7, WKUP4/ LSCO

Implementation of Low Power Mode

Shutdown Mode


```
#ifdef SHUTDOWN_MODE
    /* Configure GPIO used as systm wakeup pin4(PA2). */
    SYSWKUP4_GPIOConfig();

    /* Check and Clear the Wakeup flag */
    if (__HAL_PWR_GET_FLAG(PWR_FLAG_WUF4) != RESET){
      __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WUF4);
    }
    /* Enable wakeup pin WKUP4(PA2) */
    HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN4);

    /* Set RTC back-up register RTC_BKP31R to indicate
       later on that system has entered shutdown mode  */
    WRITE_REG(RTC->BKP31R, 0x1);

    /* Enter shutdown mode */
    HAL_PWREx_EnterSHUTDOWNMode();
#endif
```

Implementation of Low Power Mode

Shutdown Mode


```
#ifdef SHUTDOWN_MODE
    /* Configure GPIO used as systm wakeup pin4(PA2). */
    SYSWKUP4_GPIOConfig();

    /* Check and Clear the Wakeup flag */
    if (__HAL_PWR_GET_FLAG(PWR_FLAG_WUF4) != RESET){
      __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WUF4);
    }
    /* Enable wakeup pin WKUP4(PA2) */
    HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN4);

    /* Set RTC back-up register RTC_BKP31R to indicate
       later on that system has entered shutdown mode  */
    WRITE_REG(RTC->BKP31R, 0x1);

    /* Enter shutdown mode */
    HAL_PWREx_EnterSHUTDOWNMode();
#endif
```

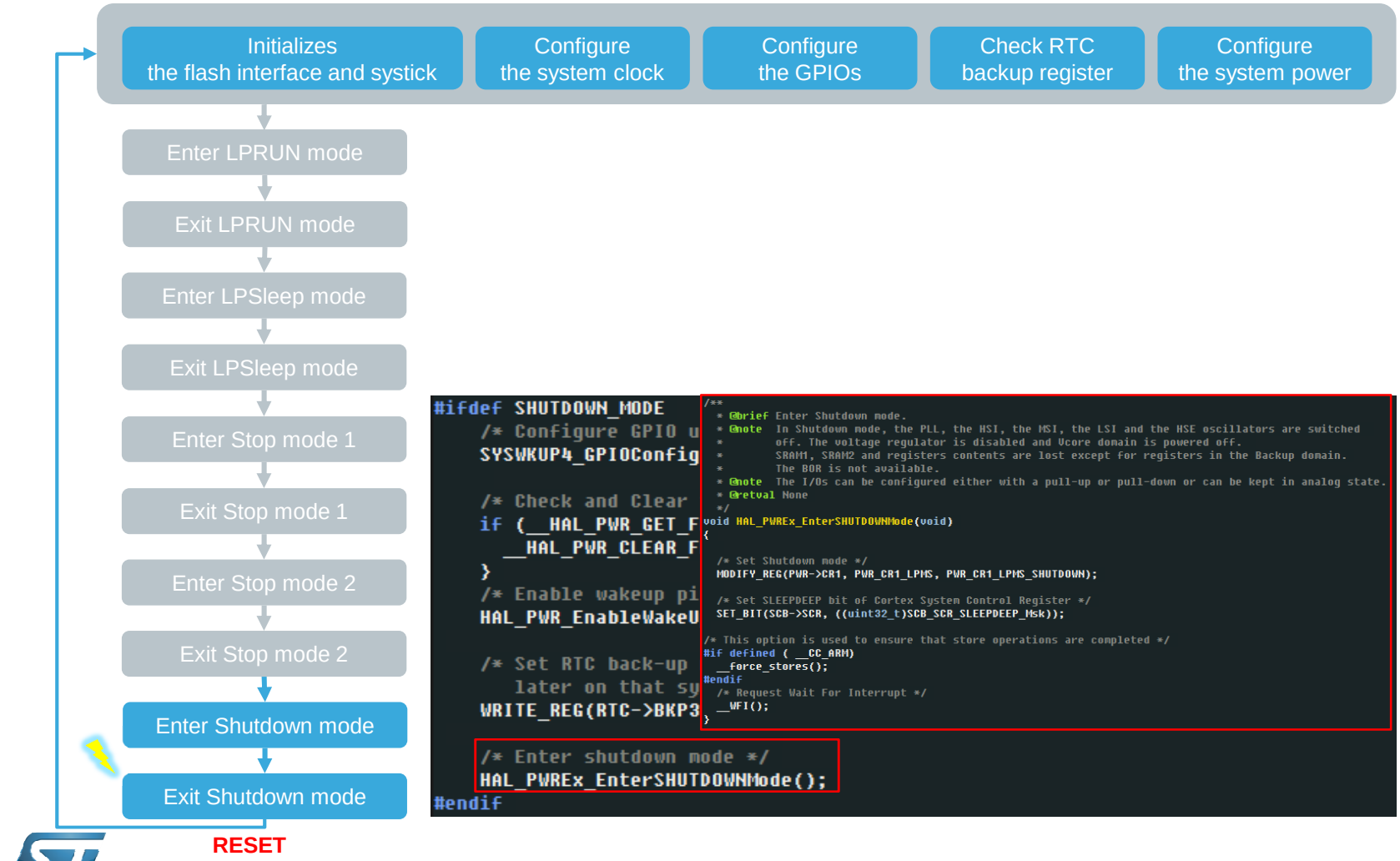
Implementation of Low Power Mode

Shutdown Mode


```
#ifdef SHUTDOWN_MODE
    /* Configure GPIO used as systm wakeup pin4(PA2). */
    SYSWKUP4_GPIOConfig();

    /* Check and Clear the Wakeup flag */
    if (__HAL_PWR_GET_FLAG(PWR_FLAG_WUF4) != RESET){
      __HAL_PWR_CLEAR_FLAG(PWR_FLAG_WUF4);
    }
    /* Enable wakeup pin WKUP4(PA2) */
    HAL_PWR_EnableWakeUpPin(PWR_WAKEUP_PIN4);

    /* Set RTC back-up register RTC_BKP31R to indicate
       later on that system has entered shutdown mode  */
    WRITE_REG(RTC->BKP31R, 0x1);

    /* Enter shutdown mode */
    HAL_PWREx_EnterSHUTDOWNMode();
#endif
```

Implementation of Low Power Mode

Shutdown Mode

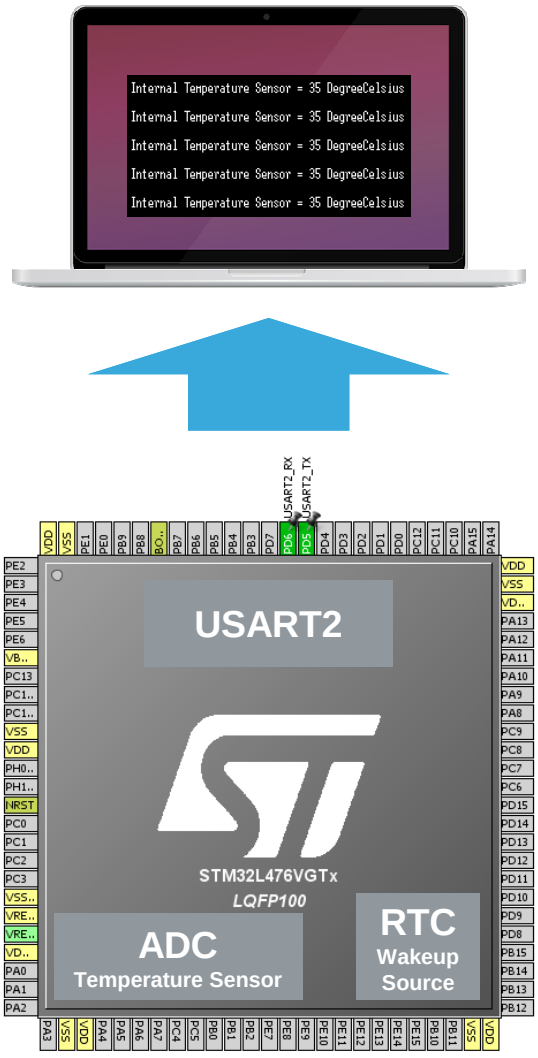



```
#ifdef SHUTDOWN_MODE
    /* Configure GPIO u
    SYSWKUP4_GPIOConfig

    /* Check and Clear
    if (__HAL_PWR_GET_F
       __HAL_PWR_CLEAR_F
    }
    /* Enable wakeup pi
    HAL_PWR_EnableWakeU

    /* Set RTC back-up
       later on that sy
    WRITE_REG(RTC->BKP3

    /* Enter shutdown mode */
    HAL_PWREx_EnterSHUTDOWNMode();
#endif
```

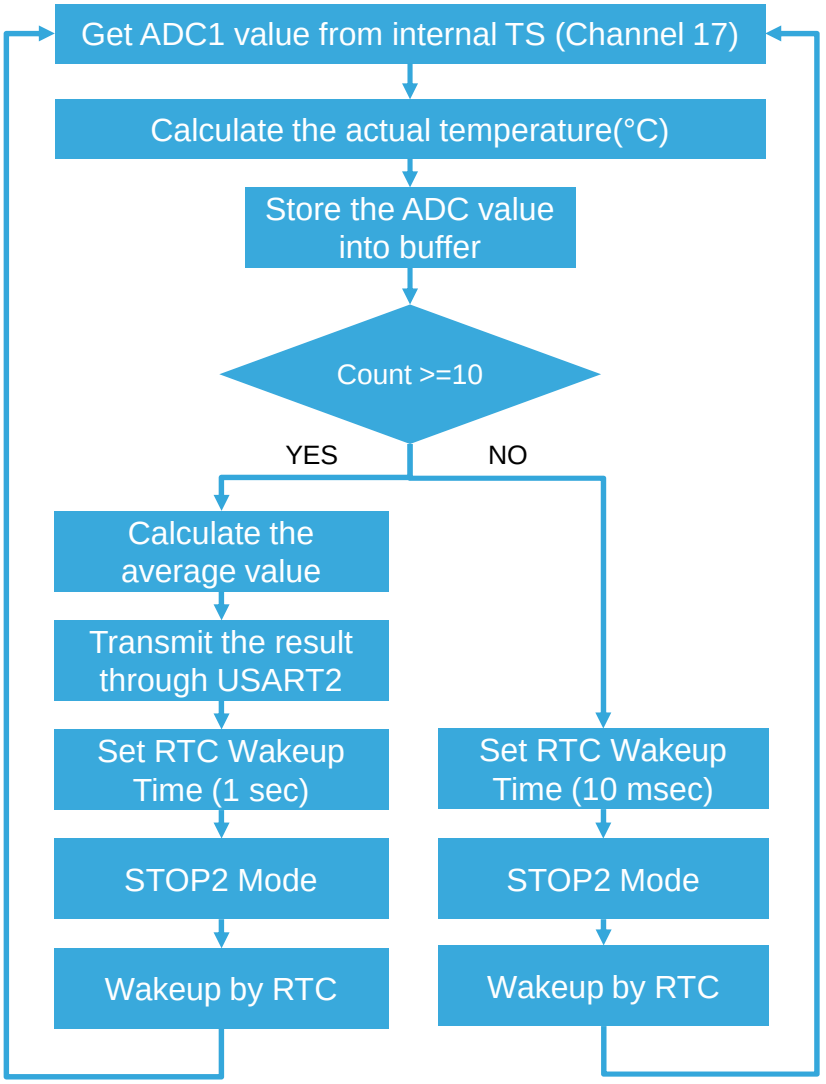
```
/**
  * @brief Enter Shutdown mode.
  * @note  In Shutdown mode, the PLL, the HSI, the MSI, the LSI and the HSE oscillators are switched
  *        off. The voltage regulator is disabled and Vcore domain is powered off.
  *        SRAM1, SRAM2 and registers contents are lost except for registers in the Backup domain.
  *        The BOR is not available.
  * @note  The I/Os can be configured either with a pull-up or pull-down or can be kept in analog state.
  * @retval None
  */
void HAL_PWREx_EnterSHUTDOWNMode(void)
{

  /* Set Shutdown mode */
  MODIFY_REG(PWR->CR1, PWR_CR1_LPMS, PWR_CR1_LPMS_SHUTDOWN);

  /* Set SLEEPDEEP bit of Cortex System Control Register */
  SET_BIT(SCB->SCR, ((uint32_t)SCB_SCR_SLEEPDEEP_Msk));

/* This option is used to ensure that store operations are completed */
#if defined ( __CC_ARM)
  __force_stores();
#endif
  /* Request Wait For Interrupt */
  __WFI();
}
```

Use Case

Block Diagram



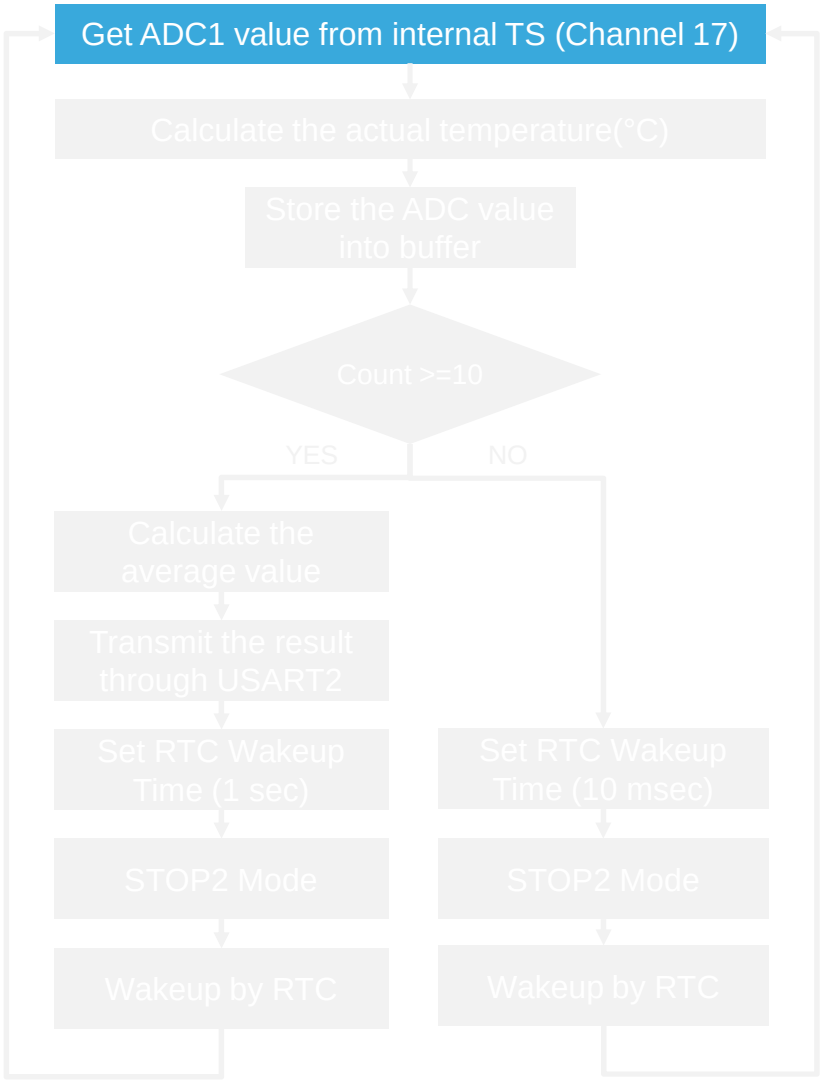
Flow Chart



Use Case

Step 1. Get ADC value

```
if (HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED) !=  HAL_OK)
{
  /* ADC Calibration Error */
  Error_Handler();
}
HAL_Delay(1);

if(HAL_ADC_Start(&hadc1) != HAL_OK){
  Error_Handler();
}

if(HAL_ADC_PollForConversion(&hadc1, 10) != HAL_OK){
  Error_Handler();
}

value_adc = HAL_ADC_GetValue(&hadc1);
```



Step 2. Calculate the actual temperature & Store the ADC value

```
/* 30 'c temperature calibration */
uint16_t ts_cal1 = *(volatile uint16_t *)(0x1FFF75A8);
/* 110 'c temperature calibration */
uint16_t ts_cal2 = *(volatile uint16_t *)(0x1FFF75CA);

float first = (float)(110 - 30) / (float)(ts_cal2 - ts_cal1);
float second = first * (float)(value_adc - ts_cal1);
float final = second + 30.0f;

adc_vals[adc_count++] = (uint16_t)final;

HAL_ADC_Stop(&hadc1);
```

Calculate the actual temperature using the following formula:

$$\text{Temperature (in °C)} = \frac{110\,°C - 30\,°C}{TS\_CAL2 - TS\_CAL1} \times (TS\_DATA - TS\_CAL1) + 30\,°C$$

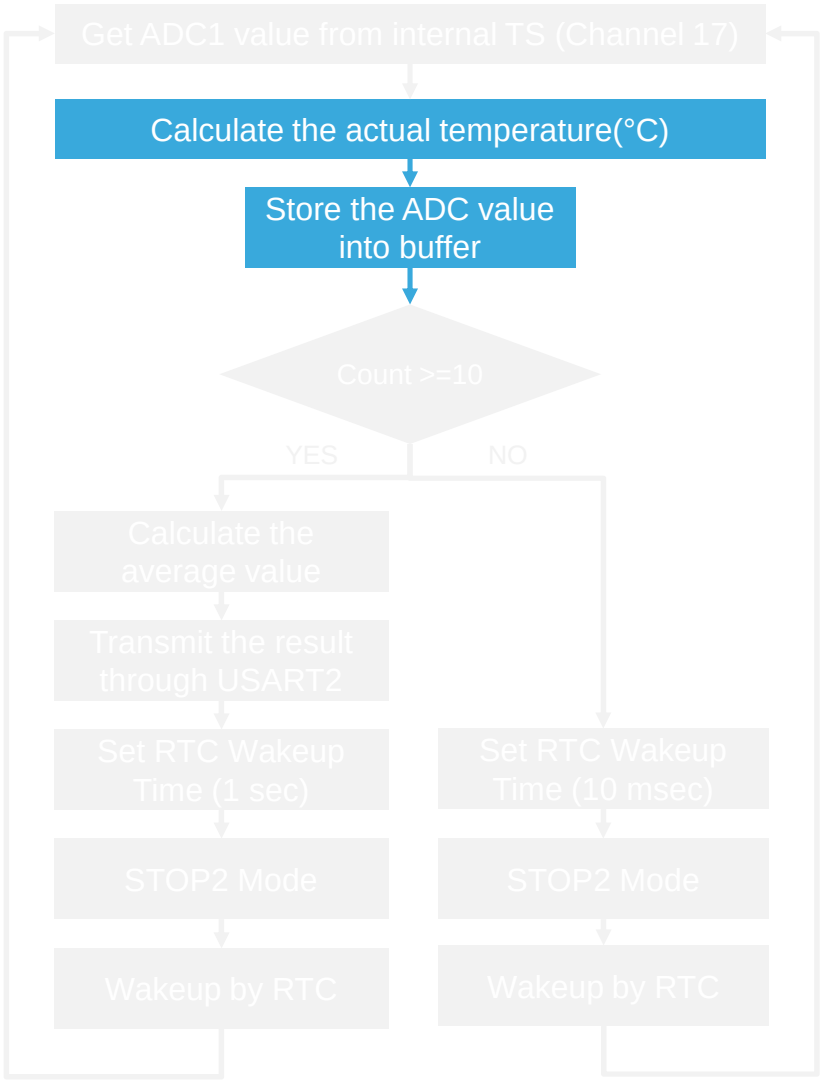
Where:

- TS_CAL2 is the temperature sensor calibration value acquired at 110°C
- TS_CAL1 is the temperature sensor calibration value acquired at 30°C
- TS_DATA is the actual temperature sensor output value converted by ADC

Refer to the device datasheet for more information about TS_CAL1 and TS_CAL2 calibration points.

Table 8. Temperature sensor calibration values

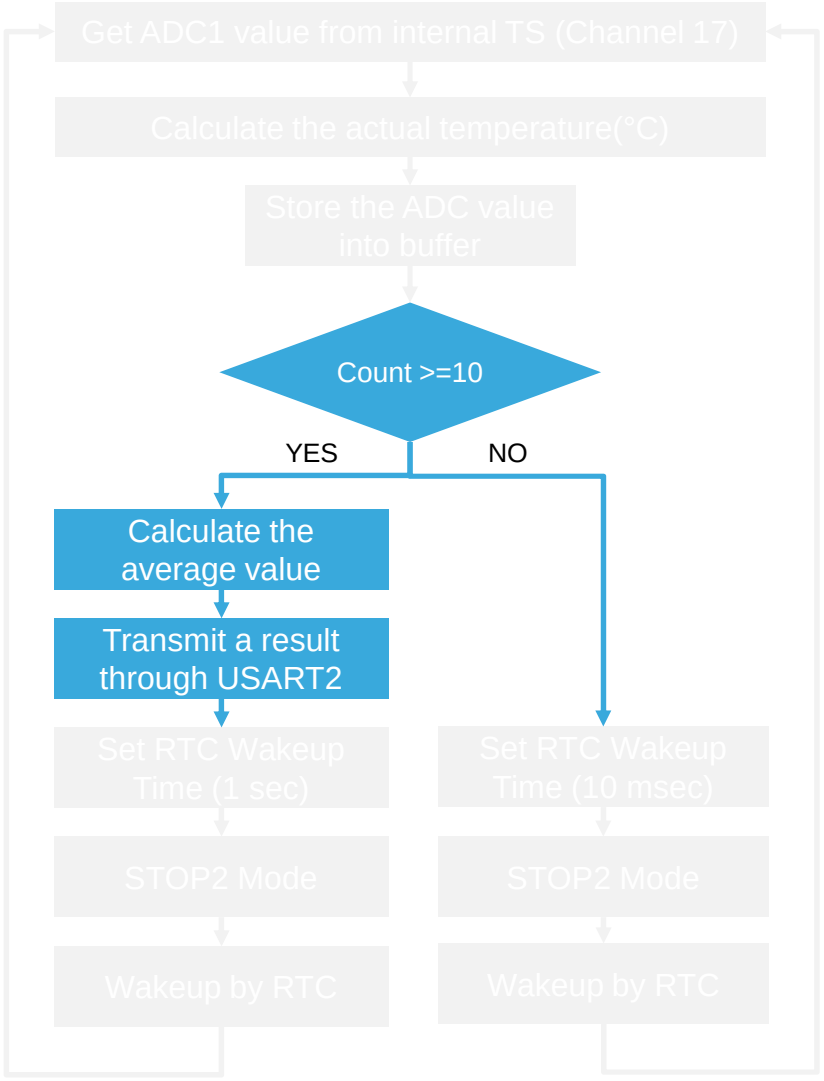
Calibration value name	Description	Memory address
TS_CAL1	TS ADC raw data acquired at a temperature of 30 °C (± 5 °C), $V_{DDA} = V_{REF+}$ = 3.0 V (± 10 mV)	0x1FFF 75A8 - 0x1FFF 75A9
TS_CAL2	TS ADC raw data acquired at a temperature of 110 °C (± 5 °C), $V_{DDA} = V_{REF+}$ = 3.0 V (± 10 mV)	0x1FFF 75CA - 0x1FFF 75CB



Use Case

Step 3. Check ADC count & Transmit a result through USART

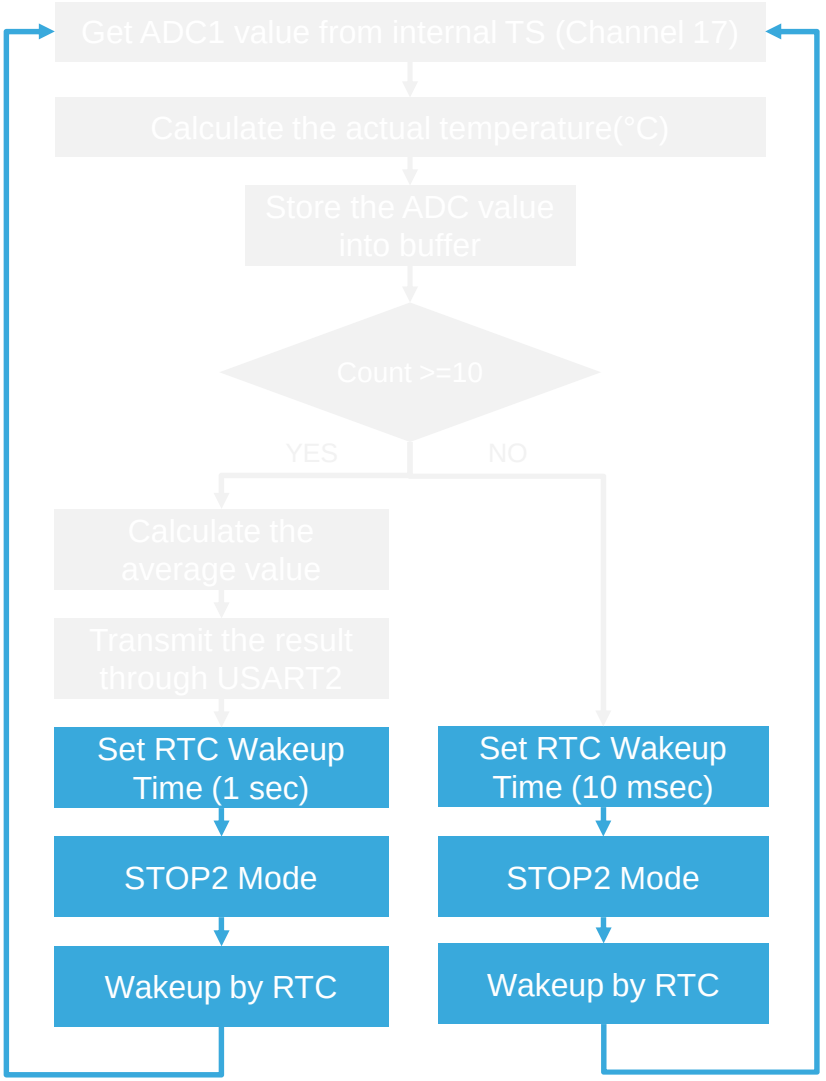
```
if (adc_count >= 10) {
  for(int i=0; i<10; i++){
    sum_temp += adc_vals[i];
  }
  avr_temp = sum_temp / 10;
  adc_count = 0;
  printf("\n\r Internal Temperature Sensor = %d DegreeCelsius\n\r", avr_temp);
  wakeup = WAKEUP_TIME_1s;
}else{
  wakeup = WAKEUP_TIME_10ms;
}
```



Use Case

Step 4. Set RTC wakeup time & Enter Stop mode

```
/* Setting RTC to wake up from stop mode */
RTC_WakeUp_Time(wakeup);

/* Enter STOP 2 mode */
HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);

/* Re-configure the system clock */
SYSCLKConfig_STOP();
```



Use Case

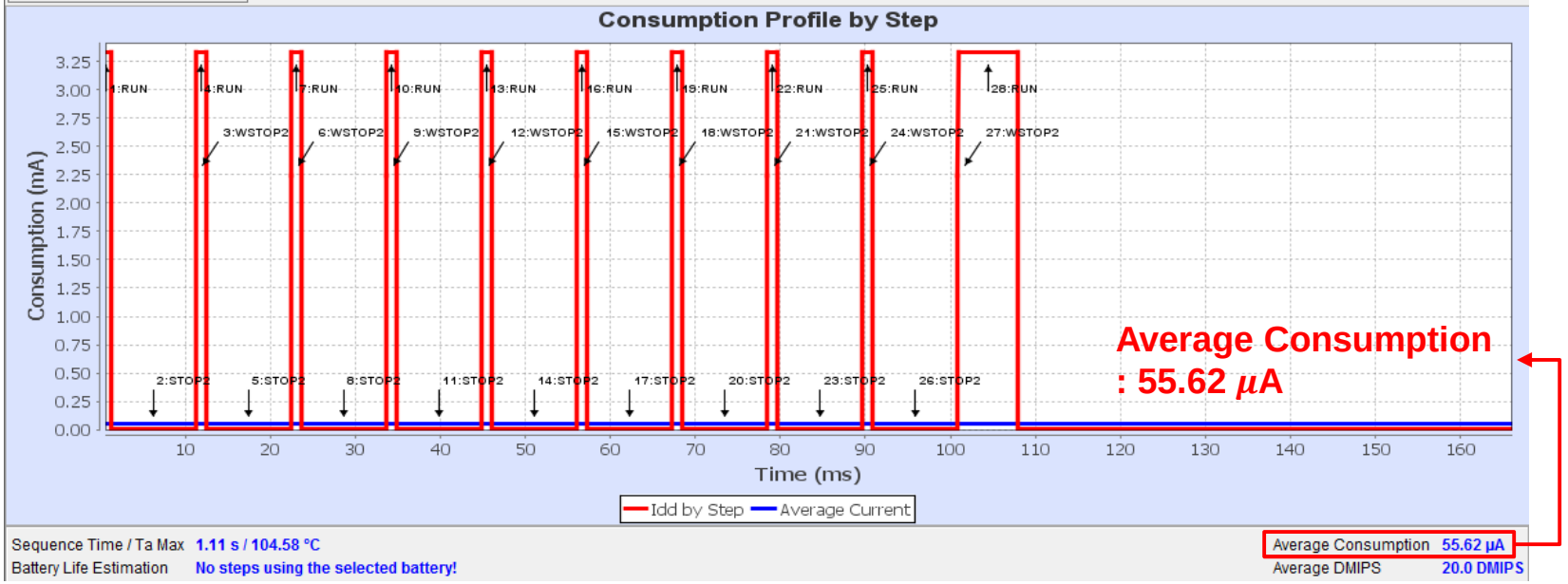
- Power Consumption Simulation using STM32CubeMX

Step | Sequence | Transitions Checker (Off, Log)

Sequence Table

Step	Mode	Vdd	Range/Scale	Memory	CPU/Bus F...	Clock Config	Src Freq	Peripherals	Add. Curr...	Step Current	Duration	DMIPS	Voltage S...	Ta Max	Category
24	WU_FROM...	3.0	NoRange	n/a	4.0 MHz	MSI	4.0 MHz	GPIOD* RTC*	0 mA	2.24 mA	10.6 µs	0.0	Vbus	104.72	In DS Table
25	RUN	3.0	Range1-High	FLASH/ART...	16.0 MHz	HSE	16.0 MHz	ADC1:fs_5_...	0 mA	3.33 mA	1.2 ms	20.0	Vbus	104.58	In DS Table
26	STOP2	3.0	NoRange	n/a	0 Hz	LSI RTC	37.0 kHz	GPIOD* RTC*	0 mA	1.64 µA	10 ms	0.0	Vbus	105	In DS Table
27	WU_FROM...	3.0	NoRange	n/a	4.0 MHz	MSI	4.0 MHz	GPIOD* RTC*	0 mA	2.24 mA	10.6 µs	0.0	Vbus	104.72	In DS Table
28	RUN	3.0	Range1-High	FLASH/ART...	16.0 MHz	HSE	16.0 MHz	ADC1:fs_5_...	0 mA	3.33 mA	7.1 ms	20.0	Vbus	104.58	In DS Table
29	STOP2	3.0	NoRange	n/a	0 Hz	LSI RTC	37.0 kHz	GPIOD* RTC*	0 mA	1.64 µA	1 s	0.0	Vbus	105	In DS Table
30	WU_FROM...	3.0	NoRange	n/a	4.0 MHz	MSI	4.0 MHz	GPIOD* RTC*	0 mA	2.24 mA	10.6 µs	0.0	Vbus	104.72	In DS Table

Display — Plot: All Steps



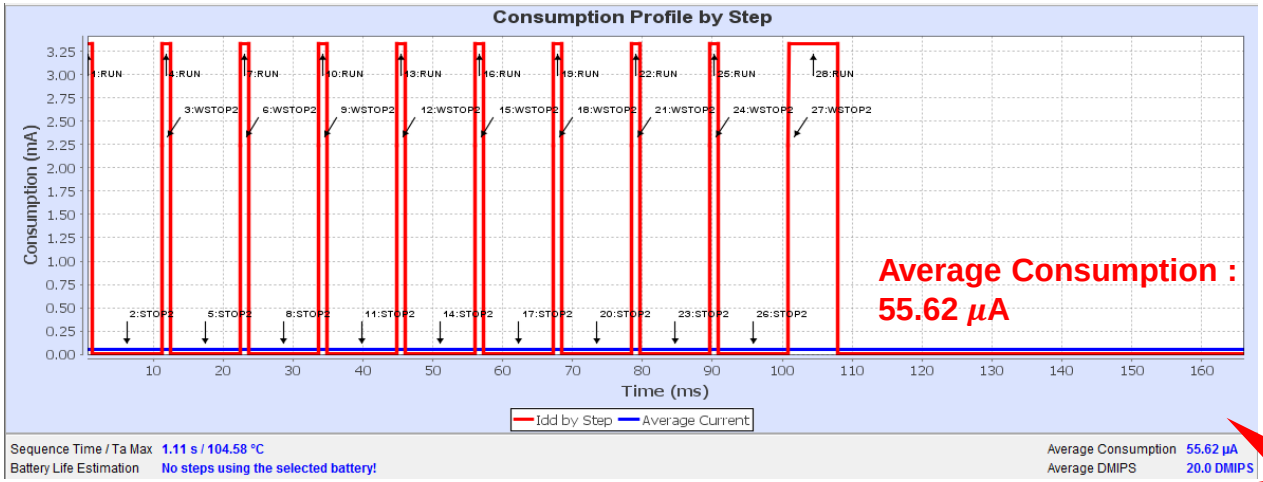
Average Consumption : 55.62 µA

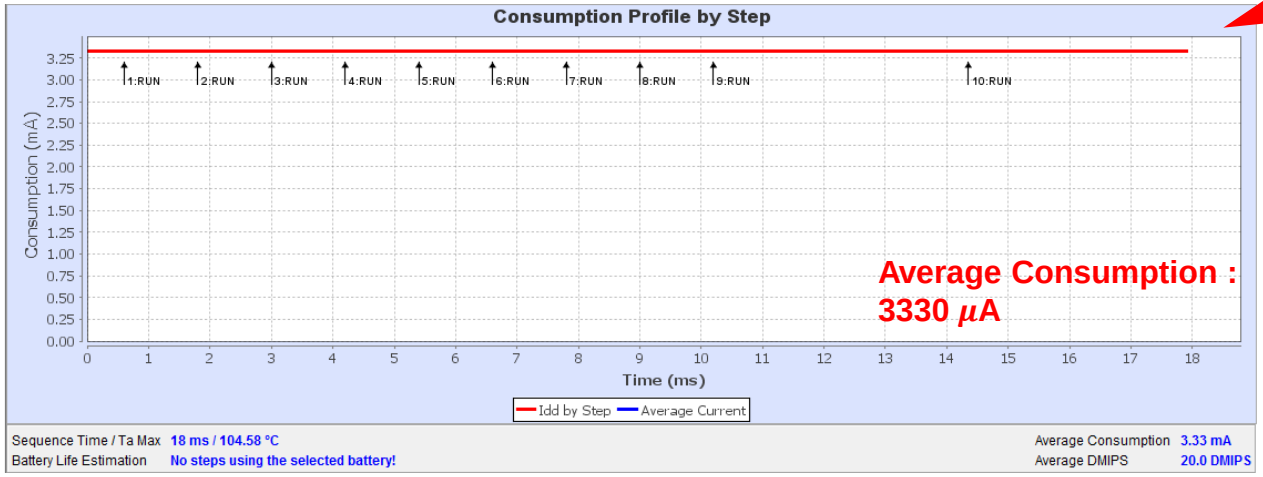
Sequence Time / Ta Max 1.11 s / 104.58 °C
Battery Life Estimation No steps using the selected battery!

Average Consumption 55.62 µA
Average DMIPS 20.0 DMIPS

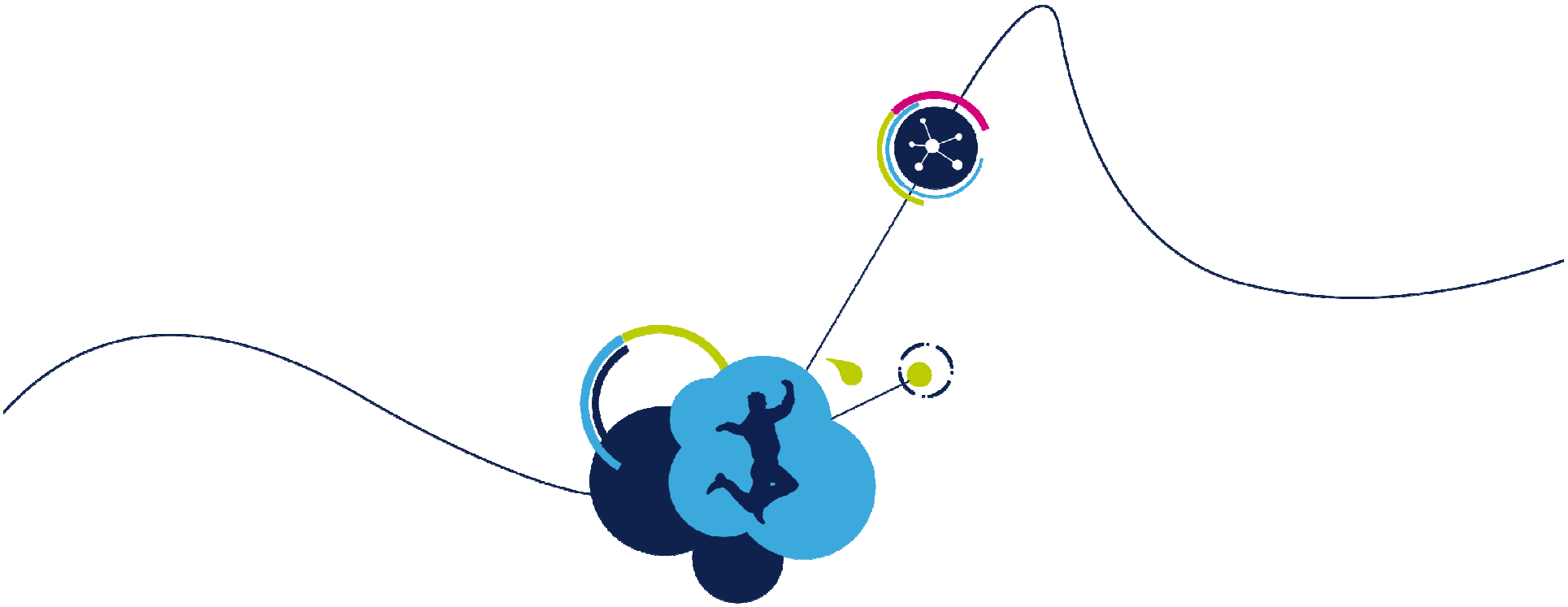
Use Case

Using Stop mode periodically VS Using Run mode only





98.32% 감소



Low Power Peripherals

life.augmented

Batch Acquisition mode (BAM)

Feature summary

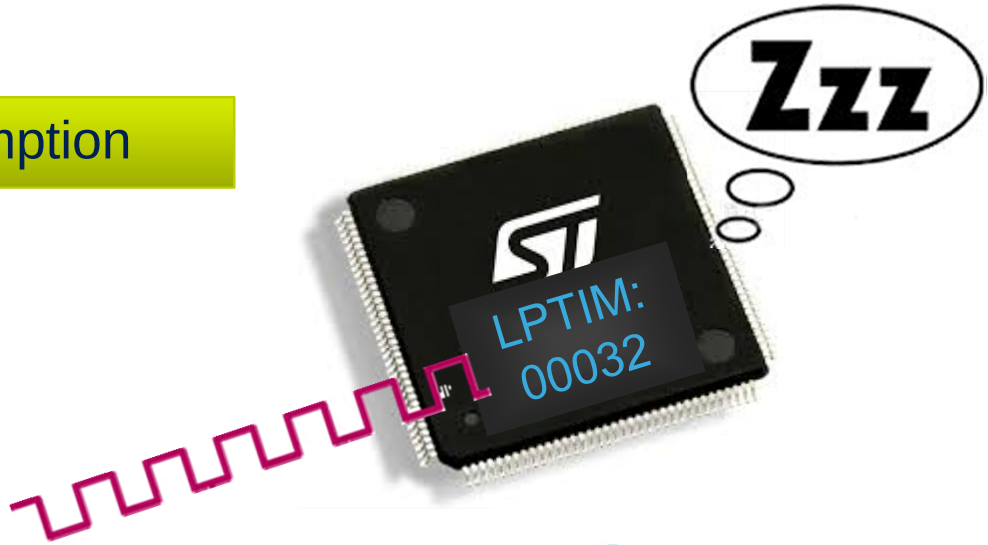
- Optimized mode for transferring data with communication peripherals, while the rest of the device is in low power.
- The BAM feature allows to keep some communication interface (I2C, SPI,…) active while in Sleep mode (CPU clock not working).

1. Only the needed communication peripheral + 1 DMA + 1 SRAM (SRAM1 or SRAM2) are configured with clock enable in Sleep mode
2. Flash is put in power-down mode and Flash clock is gated off during Sleep
3. Enter either Sleep or Low-power sleep mode
   - Note that 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.

Low Power Timer - LPTIM

Features Summary

- Asynchronous running capability
- Ultra low power-consumption



- Timeout function for wakeup from low power modes

Low Power UART - LPUART

LPUART Features

- LPUART includes all necessary hardware support to make asynchronous serial communications possible with minimum power consumption.
- Only LSE 32.768 is required to allow UART communication at up to 9600 baud → For this purpose, the baudrate generation has been changed comparing to the USART peripheral.
- Higher baudrates can be reached when the LPUART is clocked by clock sources different from the LSE clock.
- Baudrate generation and Implementation are summarized in the next slides.

USB – Link Power Management

USB-LPM

- LPM is a new power-saving state called “Sleep”, with fast entry and exit times, compared to traditional “Suspend” mode.

- Benefits : Power consumption optimization across both the host and USB devices while idle, and extend battery life of hand-held applications.

	L1 (Sleep)	L2 (Suspend)
Entry	Explicitly entered via LPM extended transaction	Implicitly entered via 3 ms of link inactivity
Exit	Device- or host-initiated via resume signaling; Remote-wake can be (optionally enabled/disabled via the LPM transaction	Device- or host-initiated via resume signaling; Remote-wake can be (optionally enabled/disabled by software
Signaling	Low- and full-speed idle	Low- and full-speed idle
Latencies	**Entry : ~ 10 µs** **Exit : > 70 µs to 1 ms (host-specific)**	**Entry : ~ 3 ms** **Exist : > 20 ms (Resume signaling)** **+ 10 ms (Resume recovery)**
Link Power Consumption	~0.6 mW (data line- pull-ups)	~0.6 mW (data line- pull-ups)
Device Power Consumption	Device power consumption level is application/implementation specific	Device consumption is limited to ≤2.5 mA
Hot Removal	Natively detected per USB 2.0 mechanisms	Natively detected per USB 2.0 mechanisms

The existing suspend/resume mechanisms have been proven to be inadequate for current and future generation mobile platforms. The bus-imposed resume latencies are so long that the mechanism doesn't support response times that are useful in many applications, especially in hand-held platforms

Thank you

www.st.com/stm32