Relojes

Ángel Perles

r04

Contenido

- Objetivo
- Relojes “electrónicamente”
- El “clock tree”
- Las “HAL” al rescate
- Con el “cube”

Objetivo

- Comprender las implicaciones de los distintos tipos de osciladores
- Conocer el concepto de “árbol de relojes”
- Conocer la arquitectura de reloj de un STM32L4xx
- Aplicar “HAL” a la configuración de reloj en la familia STM32

Relojes “electrónicamente”

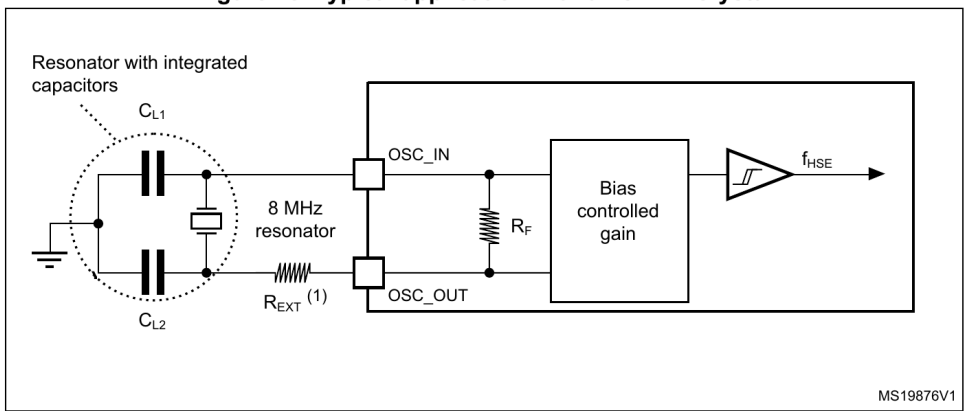
- Según el datasheet del STM32L476xx
  - Clock Sources
    - 4 to 48 MHz crystal oscillator
    - 32 kHz crystal oscillator for RTC (LSE)
    - Internal 16 MHz factory-trimmed RC (±1%)
    - Internal low-power 32 kHz RC (±5%)
    - Internal multispeed 100 kHz to 48 MHz oscillator, auto-trimmed by LSE (better than ±0.25 % accuracy)
    - 3 PLLs for system clock, USB, audio, ADC
- Si hay tantos será por algo
  - RC -> Interno (no hay que pagar más) pero mira el %
  - Crystal -> externo y caro pero muy preciso**
  - 32 kHz de mucho tipos … para medir el paso del tiempo, …
  - …

Relojes “electrónicamente”

- Analicemos las consecuencias del “%”
- Actividad:
  - Calcula cuánto dura “1 hora” con el error de los relojes anteriores
  - Busca el error de los “cristales” con el móvil
- Asumiendo que se entiende la problemática, así colocamos un reloj fuera del encapsulado

Note: *For information on selecting the crystal, refer to the application note AN2867 “Oscillator design guide for ST microcontrollers” available from the ST website www.st.com.*

Figure 25. Typical application with an 8 MHz crystal



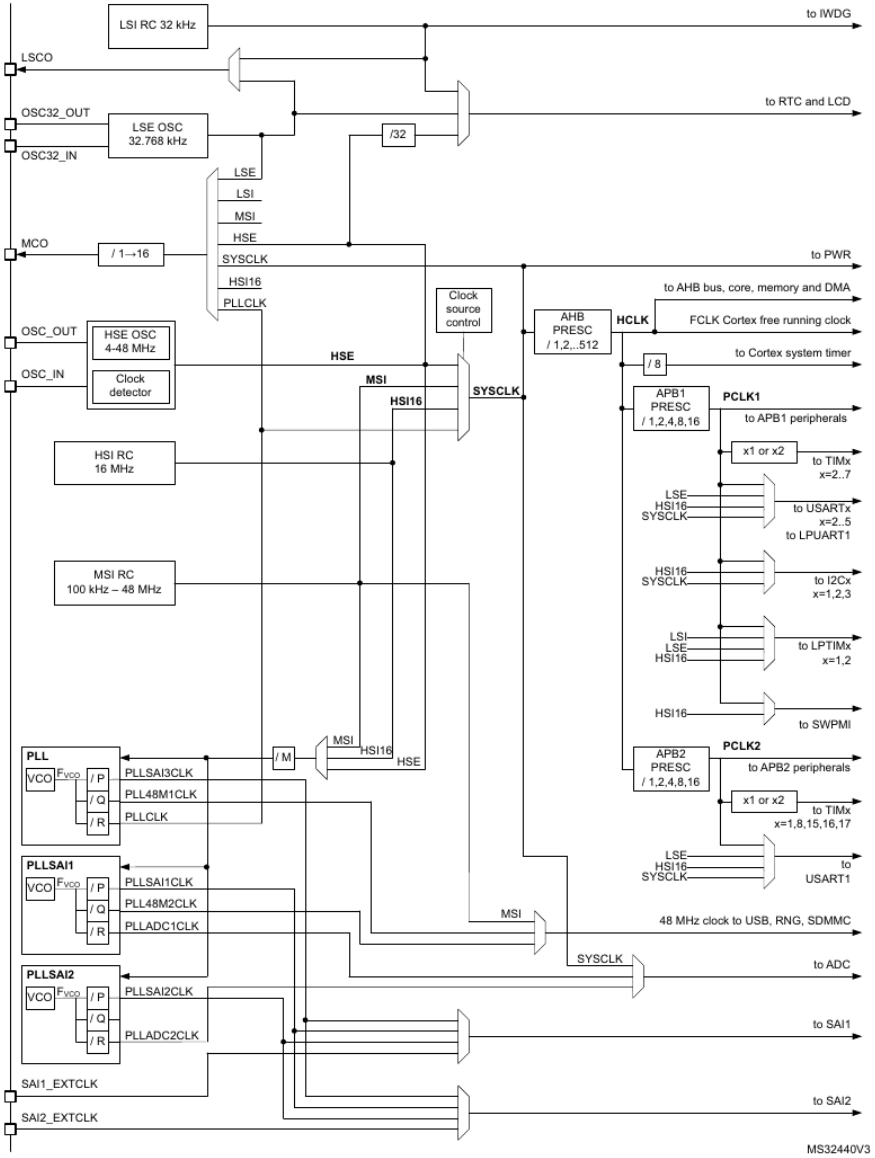
1. R_{EXT} value depends on the crystal characteristics.

El “clock tree”

- Los relojes se van ramificando para llegar a cada periférico a la frecuencia deseada
- Las abreviaturas a conocer
  - HSI: High-speed internal
  - HSE: High-speed external
  - LSI: Low-speed internal
  - LSE: Low-speed external
  - …
- Y miramos el datasheet: *“3.11 Clocks and startup”*

El “clock tree”

- “Clock tree”
  - ¡Jolines!



El “clock tree”

- Tenemos “preescalers” para dividir los relojes
- Y “PLL” para sintetizar frecuencias (pasamos al “Reference manual”)

The device embeds 3 PLLs: PLL, PLLSAI1, PLLSAI2. Each PLL provides up to three independent outputs. The internal PLLs can be used to multiply the HSI16, HSE or MSI output clock frequency. The PLLs input frequency must be between 4 and 16 MHz. The selected clock source is divided by a programmable factor PLLM from 1 to 8 to provide a clock frequency in the requested input range. Refer to *Figure 15: Clock tree (for STM32L475xx/476xx/486xx devices)* and *Figure 16: Clock tree (for STM32L496xx/4A6xx devices)* and *PLL configuration register (RCC_PLLCFGR)*.

The PLLs configuration (selection of the input clock and multiplication factor) must be done before enabling the PLL. Once the PLL is enabled, these parameters cannot be changed.

To modify the PLL configuration, proceed as follows:

1. Disable the PLL by setting PLLON to 0 in *Clock control register (RCC_CR)*.
2. Wait until PLLRDY is cleared. The PLL is now fully stopped.
3. Change the desired parameter.
4. Enable the PLL again by setting PLLON to 1.
5. Enable the desired PLL outputs by configuring PLLPEN, PLLQEN, PLLREN in *PLL configuration register (RCC_PLLCFGR)*.

El “clock tree”

- Y “PLL” para sintetizar frecuencias …

6.4.4 PLL configuration register (RCC_PLLCFGR)

Address offset: 0x0C

Reset value: 0x0000 1000

Access: no wait state, word, half-word and byte access

This register is used to configure the PLL clock outputs according to the formulas:

- f(VCO clock) = f(PLL clock input) × (PLLN / PLLM)
- f(PLL_P) = f(VCO clock) / PLLP
- f(PLL_Q) = f(VCO clock) / PLLQ
- f(PLL_R) = f(VCO clock) / PLLR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PLLPDIV[4:0]					PLLR[1:0]		PLL REN	Res.	PLLQ[1:0]		PLL QEN	Res.	Res.	PLLP	PLL PEN
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PLLN[7:0]							Res.	PLLM[2:0]			Res.	Res.	PLLSRC[1:0]	
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw	rw

Bits 31:27 **PLLPDIV[4:0]**: Main PLL division factor for PLLSAI2CLK (only for STM32L496xx/4A6xx devices)

Set and cleared by software to control the SAI1 or SAI2 clock frequency. PLLSAI3CLK output clock frequency = VCO frequency / PLLPDIV.
00000: PLLSAI3CLK is controlled by the bit PLLP
00001: Reserved.
00010: PLLSAI3CLK = VCO / 2
....
11111: PLLSAI3CLK = VCO / 31

El “clock tree”

- Y “PLL” para sintetizar frecuencias …

Bits 26:25 **PLLR[1:0]**: Main PLL division factor for PLLCLK (system clock)
Set and cleared by software to control the frequency of the main PLL output clock PLLCLK. This output can be selected as system clock. These bits can be written only if PLL is disabled.
PLLCLK output clock frequency = VCO frequency / PLLR with PLLR = 2, 4, 6, or 8
00: PLLR = 2
01: PLLR = 4
10: PLLR = 6
11: PLLR = 8

Caution: The software has to set these bits correctly not to exceed 80 MHz on this domain.

Bit 24 **PLLREN**: Main PLL PLLCLK output enable
Set and reset by software to enable the PLLCLK output of the main PLL (used as system clock).
This bit cannot be written when PLLCLK output of the PLL is used as System Clock.
In order to save power, when the PLLCLK output of the PLL is not used, the value of PLLREN should be 0.
0: PLLCLK output disable
1: PLLCLK output enable

Bit 23 Reserved, must be kept at reset value.

Bits 22:21 **PLLQ[1:0]**: Main PLL division factor for PLL48M1CLK (48 MHz clock).
Set and cleared by software to control the frequency of the main PLL output clock PLL48M1CLK. This output can be selected for USB, RNG, SDMMC (48 MHz clock). These bits can be written only if PLL is disabled.
PLL48M1CLK output clock frequency = VCO frequency / PLLQ with PLLQ = 2, 4, 6, or 8
00: PLLQ = 2
01: PLLQ = 4
10: PLLQ = 6
11: PLLQ = 8

Caution: The software has to set these bits correctly not to exceed 80 MHz on this domain.

Bit 20 **PLLQEN**: Main PLL PLL48M1CLK output enable
Set and reset by software to enable the PLL48M1CLK output of the main PLL.
In order to save power, when the PLL48M1CLK output of the PLL is not used, the value of PLLQEN should be 0.
0: PLL48M1CLK output disable
1: PLL48M1CLK output enable

Bits 19:18 Reserved, must be kept at reset value.

Bit 17 **PLLP**: Main PLL division factor for PLLSAI3CLK (SAI1 and SAI2 clock).
Set and cleared by software to control the frequency of the main PLL output clock PLLSAI3CLK. This output can be selected for SAI1 or SAI2. These bits can be written only if PLL is disabled.
(When the PLLPDIV[4:0] is set to “00000” only for STM32L496xx/4A6xx devices)PLLSAI3CLK output clock frequency = VCO frequency / PLLP with PLLP =7, or 17
0: PLLP = 7
1: PLLP = 17

Caution: The software has to set these bits correctly not to exceed 80 MHz on this domain.

El “clock tree”

- Y “PLL” para sintetizar frecuencias …

Bit 16 **PLLPEN**: Main PLL PLLSAI3CLK output enable
Set and reset by software to enable the PLLSAI3CLK output of the main PLL.
In order to save power, when the PLLSAI3CLK output of the PLL is not used, the value of PLLPEN should be 0.
0: PLLSAI3CLK output disable
1: PLLSAI3CLK output enable

Bit 15 Reserved, must be kept at reset value.

Bits 14:8 **PLLN[6:0]**: Main PLL multiplication factor for VCO
Set and cleared by software to control the multiplication factor of the VCO. These bits can be written only when the PLL is disabled.
VCO output frequency = VCO input frequency x PLLN with 8 =< PLLN =< 86
0000000: PLLN = 0 wrong configuration
0000001: PLLN = 1 wrong configuration
...
0000111: PLLN = 7 wrong configuration
0001000: PLLN = 8
0001001: PLLN = 9
...
1010101: PLLN = 85
1010110: PLLN = 86
1010111: PLLN = 87 wrong configuration
...
1111111: PLLN = 127 wrong configuration

Caution: The software has to set correctly these bits to assure that the VCO output frequency is between 64 and 344 MHz.

Bit 7 Reserved, must be kept at reset value.

Bits 6:4 **PLLM:** Division factor for the main PLL and audio PLL (PLLSAI1 and PLLSAI2) input clock
Set and cleared by software to divide the PLL, PLLSAI1 and PLLSAI2 input clock before the VCO. These bits can be written only when all PLLs are disabled.
VCO input frequency = PLL input clock frequency / PLLM with 1 <= PLLM <= 8
000: PLLM = 1
001: PLLM = 2
010: PLLM = 3
011: PLLM = 4
100: PLLM = 5
101: PLLM = 6
110: PLLM = 7
111: PLLM = 8

Caution: The software has to set these bits correctly to ensure that the VCO input frequency ranges from 4 to 16 MHz.

Bits 3:2 Reserved, must be kept at reset value.

Bits 1:0 **PLLSRC**: Main PLL, PLLSAI1 and PLLSAI2 entry clock source
Set and cleared by software to select PLL, PLLSAI1 and PLLSAI2 clock source. These bits can be written only when PLL, PLLSAI1 and PLLSAI2 are disabled.
In order to save power, when no PLL is used, the value of PLLSRC should be 00.
00: No clock sent to PLL, PLLSAI1 and PLLSAI2
01: MSI clock selected as PLL, PLLSAI1 and PLLSAI2 clock entry
10: HSI16 clock selected as PLL, PLLSAI1 and PLLSAI2 clock entry
11: HSE clock selected as PLL, PLLSAI1 and PLLSAI2 clock entry

El “clock tree”

- Y “PLL” para sintetizar frecuencias …
- Es un poco retorcido, pero permite generar distintas frecuencias
- Ejemplo:
  - Usando HSI y M = 2, N = 20, P = 7, Q = 4, R = 2
  - Actividad:
    - ¿Es una configuración válida?
    - Calcula el SYSCLK que saldría asumiendo que es válida
  - Actividad
    - Usando el HSI, pon a rular SYSCLK con PLL a 40 MHz
    - Inténtalo después con 30 MHz

Las “HAL” al rescate

- Funciones para gestionar osciladores y relojes (del manual HAL)

HAL_RCC_DeInit	
Function name	**void HAL_RCC_DeInit (void)**
Function description	Reset the RCC clock configuration to the default reset state.

HAL_RCC_OscConfig	
Function name	**HAL_StatusTypeDef HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)**
Function description	Initialize the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef.

HAL_RCC_ClockConfig	
Function name	**HAL_StatusTypeDef HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)**
Function description	Initialize the CPU, AHB and APB busses clocks according to the specified parameters in the RCC_ClkInitStruct.

Las “HAL” al rescate

- Estructuras para estas funciones

51.1.2 RCC_OscInitTypeDef

Data Fields

- *uint32_t OscillatorType*
- *uint32_t HSEState*
- *uint32_t LSEState*
- *uint32_t HSIState*
- *uint32_t HSICalibrationValue*
- *uint32_t LSIState*
- *uint32_t MSIState*
- *uint32_t MSICalibrationValue*
- *uint32_t MSIClockRange*
- *uint32_t HSI48State*
- *RCC_PLLInitTypeDef PLL*

51.1.3 RCC_ClkInitTypeDef

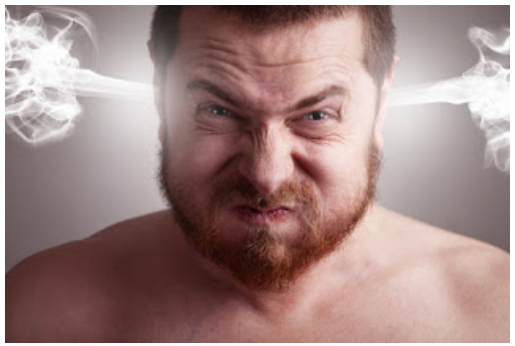
Data Fields

- *uint32_t ClockType*
- *uint32_t SYSCLKSource*
- *uint32_t AHBCLKDivider*
- *uint32_t APB1CLKDivider*
- *uint32_t APB2CLKDivider*

51.1.1 RCC_PLLInitTypeDef

Data Fields

- *uint32_t PLLState*
- *uint32_t PLLSource*
- *uint32_t PLLM*
- *uint32_t PLLN*
- *uint32_t PLLP*
- *uint32_t PLLQ*
- *uint32_t PLLR*



Las “HAL” al rescate

- Con un ejemplo se entiende mejor (sacado de los ej. oficiales)

```
/**
  * @brief  Switch the PLL source from MSI to HSI, and select the PLL as SYSCLK
  *         source.
  *         The system Clock is configured as follows :
  *            System Clock source            = PLL (HSI)
  *            SYSCLK(Hz)                     = 80000000
  *            HCLK(Hz)                       = 80000000
  *            AHB Prescaler                  = 1
  *            APB1 Prescaler                 = 1
  *            APB2 Prescaler                 = 1
  *            HSI Frequency(Hz)              = 16000000
  *            PLLM                           = 2
  *            PLLN                           = 20
  *            PLLP                           = 7
  *            PLLQ                           = 4
  *            PLLR                           = 2
  *            Flash Latency(WS)              = 4
  * @param  None
  * @retval None
  */
static void SystemClockHSI_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
```

Las “HAL” al rescate

- Con un ejemplo se entiende mejor (sacado de los ej. oficiales)

```
/* -1- Select MSI as system clock source to allow modification of the PLL configuration */
RCC_ClkInitStruct.ClockType       = RCC_CLOCKTYPE_SYSCLK;
RCC_ClkInitStruct.SYSCLKSource    = RCC_SYSCLKSOURCE_MSI;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
{
  /* Initialization Error */
  Error_Handler();
}

/* -2- Enable HSI Oscillator, select it as PLL source and finally activate the PLL */
RCC_OscInitStruct.OscillatorType    = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSIState          = RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue  = RCC_HSICALIBRATION_DEFAULT;
RCC_OscInitStruct.PLL.PLLState      = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource     = RCC_PLLSOURCE_HSI;
RCC_OscInitStruct.PLL.PLLM          = 2;
RCC_OscInitStruct.PLL.PLLN          = 20;
RCC_OscInitStruct.PLL.PLLP          = 7;
RCC_OscInitStruct.PLL.PLLQ          = 4;
RCC_OscInitStruct.PLL.PLLR          = 2;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
  /* Initialization Error */
  Error_Handler();
}
```

Las “HAL” al rescate

- Con un ejemplo se entiende mejor (sacado de los ej. oficiales)

```
  /* -3- Select the PLL as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks dividers */
  RCC_ClkInitStruct.ClockType       = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource    = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider   = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider  = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider  = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }

  /* -4- Optional: Disable MSI Oscillator */
  RCC_OscInitStruct.OscillatorType  = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState        = RCC_MSI_OFF;
  RCC_OscInitStruct.PLL.PLLState    = RCC_PLL_NONE;  /* No update on PLL */
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }
}
```

- Es un ejemplo oficial pero poco didáctico
  - Asume MSI funcionando

Las “HAL” al rescate

- ¿Y eso de la “FLASH noseque qué es?
  - La CPU puede ser más rápida que la respuesta de la Flash, se ajusta haciendo que espere en cada lectura

3.3.3 Read access latency

To correctly read data from Flash memory, the number of wait states (LATENCY) must be correctly programmed in the *Flash access control register (FLASH_ACR)* according to the frequency of the CPU clock (HCLK) and the internal voltage range of the device V_{CORE}. Refer to *Section 5.1.8: Dynamic voltage scaling management*. *Table 11* shows the correspondence between wait states and CPU clock frequency.

Table 11. Number of wait states according to CPU clock (HCLK) frequency

Wait states (WS) (LATENCY)	HCLK (MHz)	
	V_{CORE} Range 1(1)	V_{CORE} Range 2(2)
0 WS (1 CPU cycles)	≤ 16	≤ 6
1 WS (2 CPU cycles)	≤ 32	≤ 12
2 WS (3 CPU cycles)	≤ 48	≤ 18
3 WS (4 CPU cycles)	≤ 64	≤ 26
4 WS (5 CPU cycles)	≤ 80	≤ 26

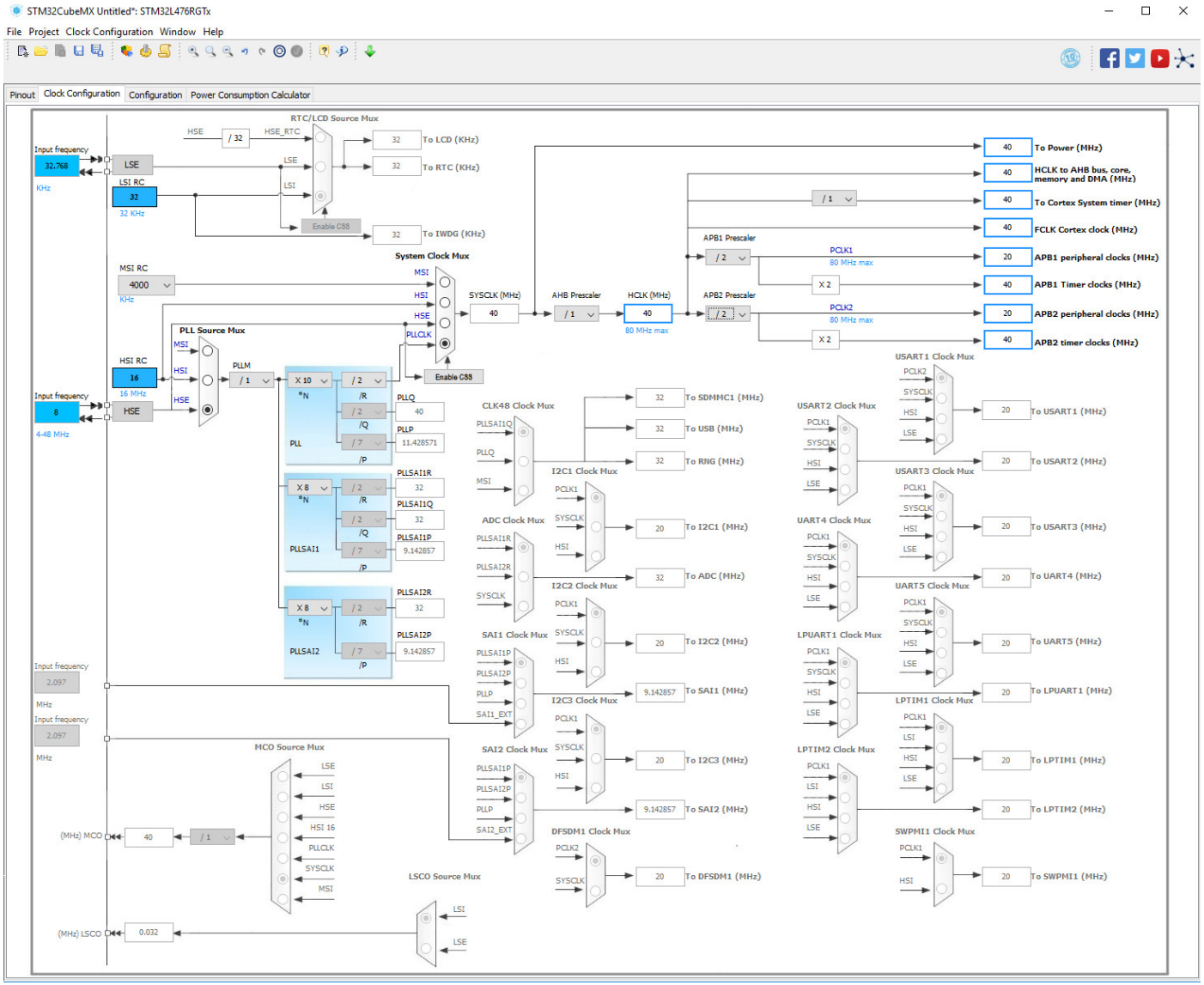
1. Also for SMPS Range1 or SMPS Range2 high.
2. Also for SMPS Range2 low.

After reset, the CPU clock frequency is 4 MHz and 0 wait state (WS) is configured in the FLASH_ACR register.

Las “HAL” al rescate

- ¿Y que pasa con *SystemCoreClock*, *HAL_Delay(), …*?
  - Si tocas directamente los registros no funcionarán bien
  - Si usas las HAL, automáticamente se reconfigurará *systick* y similares y todo seguirá funcionando bien

Con el “cube”


STM32CubeMX Untitled*: STM32L476RGTx
File Project Clock Configuration Window Help
Pinout Clock Configuration Configuration Power Consumption Calculator
RTC/LCD Source Mux
System Clock Mux
PLL Source Mux
CLK48 Clock Mux
I2C1 Clock Mux
ADC Clock Mux
I2C2 Clock Mux
SAI1 Clock Mux
I2C3 Clock Mux
SAI2 Clock Mux
DFSDM1 Clock Mux
USART1 Clock Mux
USART2 Clock Mux
USART3 Clock Mux
UART4 Clock Mux
UART5 Clock Mux
LPUART1 Clock Mux
LPTIM1 Clock Mux
LPTIM2 Clock Mux
SWPMI1 Clock Mux
MCO Source Mux
LSCO Source Mux
To Power (MHz)
HCLK to AHB bus, core, memory and DMA (MHz)
To Cortex System timer (MHz)
FCLK Cortex clock (MHz)
APB1 peripheral clocks (MHz)
APB1 Timer clocks (MHz)
APB2 peripheral clocks (MHz)
APB2 timer clocks (MHz)
To LCD (KHz)
To RTC (KHz)
To IWDG (KHz)
To SDMMC1 (MHz)
To USB (MHz)
To RNG (MHz)
To I2C1 (MHz)
To ADC (MHz)
To I2C2 (MHz)
To SAI1 (MHz)
To I2C3 (MHz)
To SAI2 (MHz)
To DFSDM1 (MHz)
To USART1 (MHz)
To USART2 (MHz)
To USART3 (MHz)
To UART4 (MHz)
To UART5 (MHz)
To LPUART1 (MHz)
To LPTIM1 (MHz)
To LPTIM2 (MHz)
To SWPMI1 (MHz)

Con el “cube”

```
/**
  * @brief System Clock Configuration
  * @retval None
  */
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_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
```

- Deberes: busca de dónde he sacado el ejemplo y míralo todo para ver cómo hacer otra configuración interesante
- Práctica con el “cube” creando proyectos.