ARM_Cortex
Magasszintű fejlesztés támogatás

Scherer Balázs

Méréstechnika és
Információs Rendszerek
Tanszék

Fejlesztő környezetek változása az utóbbi években

- Az elmúlt években rengeteget változtak fejlesztő környezetek
- A traditcionális firmware driver támogatás egyre kevésbé elég
- A bonyolult és gépies hardware függő konfigurációkat egyre több helyen grafikus támogatással látják el.

Kiváltható részek

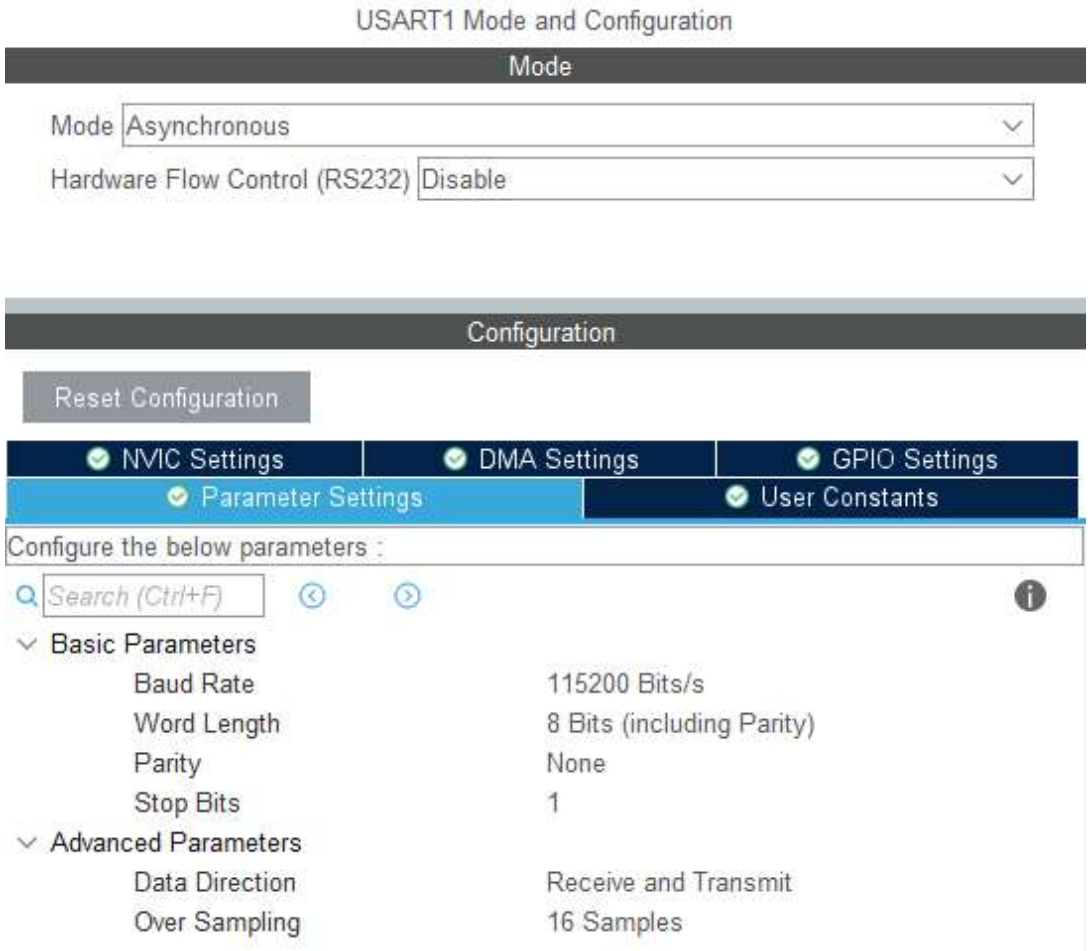
- A gépies konfigurációk egyszerűbben és átláthatóbban megoldhatók grafikus módon

```
LL_USART_InitTypeDef  USART_InitStruct;

USART_InitStruct.BaudRate = 9600;
USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
USART_InitStruct.Parity = LL_USART_PARITY_NONE;
USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
LL_USART_Init  (USART1, &USART_InitStruct);
```

Kiváltható részek

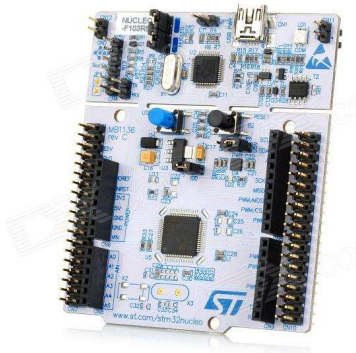
- A gépies konfigurációk egyszerűbben és átláthatóbban megoldhatók grafikus módon



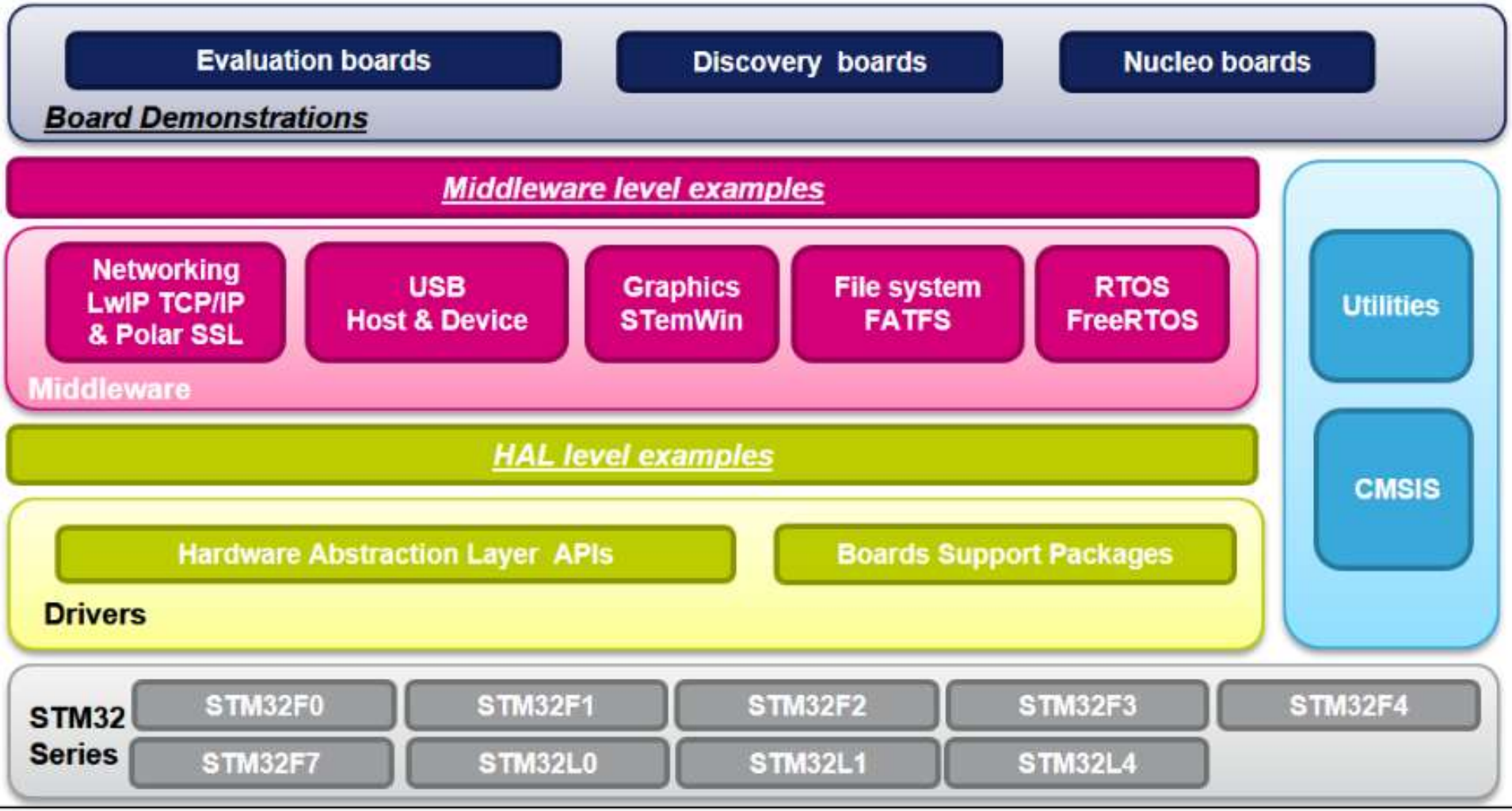
STM Cube HAL, egy új generációs firmware library

STM32 Cube HAL

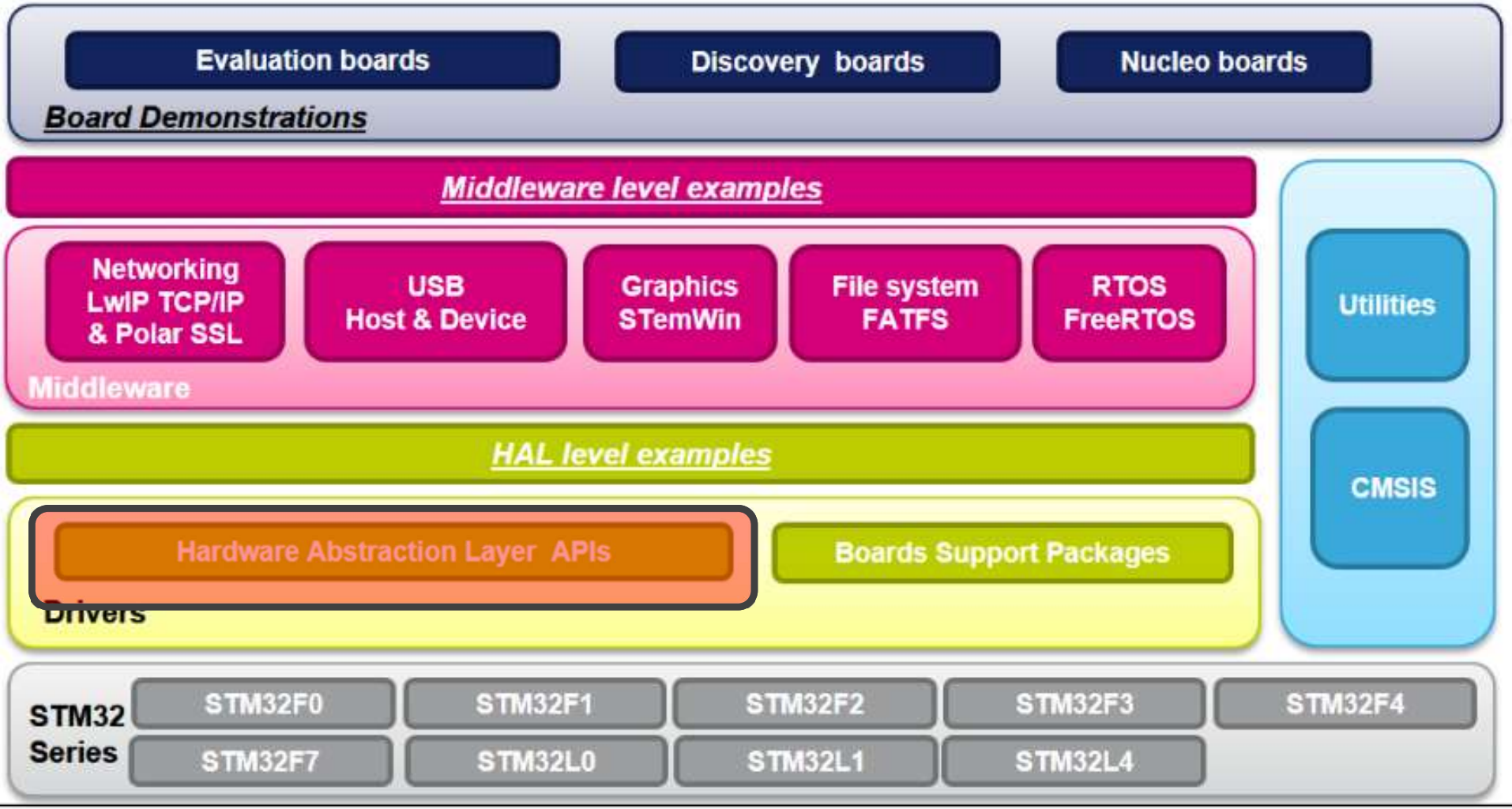
- 2014-ben jelent meg a Firmware Library leváltására
- Két támogatott demóboard készlet
  - Nucleo sorozat
  - Discovery sorozat
- Minden sorozathoz egyedi letöltendő Cube platform különbségekkel, de az alapok ugyanazok



STM32 Cube architektúra



STM32 Cube architektúra



STM32 Cube HAL (Hardware Abstraction Layer)

- Több száz oldalas leírás pdf-ben (pl. Description of STM32F4xx HAL drivers ~900 oldal)

- **Generic drivers:** Közös az összes adott cube sorozatban lévő STM32 mikrovezérlőre

- **Extension drivers:** Egyedi mikrovezérlő, vagy sorozat függő képességekhez tartozó tulajdonság software-es kezelése

STM32 Cube HAL általános tulajdonságok

- Három programozói modell
  - Polling
  - Interrupt
  - DMA
- RTOS kompatibilis működés
  - Reentráns (thread safe) függvények
  - Szisztematikus timer kezelés
  - Közös erőforrás védelem
- Callback alapú működés
  - Beépített Callback felprogramozási lehetőségek
- Több periféria egyed kezelése ugyanazokkal az általános függvényekkel
  - Handler struktúra alapú API szerkezet

STM32 Cube HAL elnevezési szabályok

- HAL driverekre (belső API file-ok konvenciói)

Név	Leírás
stm32f4xx_hal_ppp.c	Alap (**Generic**) driver API-k, amik az összes STM32 vezérlőre igazak például: stm32f4xx_hal_adc.c
stm32f4xx_hal_ppp.h	Alap (**Generic**) driver API-k, hoz tartozó deffiniciók, struktúrák amik az összes STM32 vezérlőre igazak például: stm32f4xx_hal_adc.h
stm32f4xx_hal_ppp_ex.c	Kiegészítő (**Extension**) driver API-k, amik egyedi STM32 vezérlőkre, sorozatokra igazak például: stm32f4xx_hal_adc_ex.c
stm32f4xx_hal_ppp_ex.h	Kiegészítő (Extension) driver API-khoz tartozó deffiniciók, struktúrák, amik egyedi STM32 vezérlőkre, sorozatokra igazak például: stm32f4xx_hal_adc_ex.c

STM32 Cube HAL elnevezési szabályok

- HAL driverekre (belső API file-ok konvenciói)

Név	Leírás
stm32f4xx_ll_ppp.c	Alacsonyszintű API driverek amiket a magasabb szintű Generic, Extension API-k használnak. **Nem direktben user használatra szántak!**
stm32f4xx_ll_ppp.h	Alacsonyszintű API driverek amiket a magasabb szintű Generic, Extension API-k használnak. **Nem direktben user használatra szántak!**
stm32f4xx_hal.c **stm32f4xx_hal.h**	A HAL inicializációs részeit tartalmazza.
stm32f4xx_hal_msp_template.c	MSP: Microcontroller Specific Package. Felhasználói templare, ami a mikrovezérlő package (F1, F2, F3, F4 ...) függő inicializációkat tartalmazza: main függvény, alap callback-ek
stm32f4xx_hal_conf_template.h	Template file a felhasznált API-k konfigurálására
stm32f4xx_hal_def.h	Általános HAL definiciók

STM32 Cube HAL minimális user alkalamzás file-ai

Név	Leírás
system_stm32f4xx.c	CMSIS: device.c. Tartalmazza a **SystemInit()** hívást. **Nem tartalmazza az órajel beállításokat**. Külön HAL apikkal kell megcsinálni, ha szükséges.
startup_stm32f4xx.s	CMSIS: startup_device. Taralmazza a reset vektort és IT vektorok definicióját. Illetve sok esetben a stack, heap alapmegadásokat.
stm32f4xx_hal_msp.c	Tartalmazza a main rutint, az MSP inicializációkat és az alap Callback-eket
stm32f4xx_hal_conf.h	Tartalmazza a felhasznált API-k konfigurálását
stm32f4xx_it.c/.h	Ez a file tartalmazza az egyes interrupt kezelő függvényeket. Ebben a file-ban hívódik meg a HAL_IncTick() függvény, ami a HAL alap 1ms-os órajelét adja a Systick IT segítségével. A **PPP_IRQHandler** rutinoknak meg kell hívniuk a **HAL_PPP_IRQHandler()** függvényeket ha IT alapú API használatot szeretnénk.
main.c/.h	Main függvény. Tartalmaznia kell a **HAL_Init()** függvény meghívását, system clock beállítását. A szükséges periféria inicializációkat és user kódot.

STM32 Cube HAL API elnevezési szabályok

	Generic	Family specific	Device specific
File names	*stm32f4xx_hal_ppp (c/h)*	*stm32f4xx_hal_ppp_ex (c/h)*	*stm32f4xx_ hal_ppp_ex (c/h)*
Module name	*HAL_PPP_ MODULE*		
Function name	*HAL_PPP_Function HAL_PPP_FeatureFunction_ MODE*	*HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_ MODE*	*HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_ MODE*
Handle name	*PPP_HandleTypedef*	*NA*	*NA*
Init structure name	*PPP_InitTypeDef*	*NA*	*PPP_InitTypeDef*
Enum name	*HAL_PPP_StructnameTypeDef*	*NA*	*NA*

STM32 Cube HAL IT engedélyezés, speciális makrók

- Az adott perifériára, modulra külön definiáltak
  - Példák

Macros	Description
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)	Enables a specific peripheral interrupt
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)	Disables a specific peripheral interrupt
__HAL_PPP_GET_IT (__HANDLE__, __ INTERRUPT __)	Gets a specific peripheral interrupt status
__HAL_PPP_CLEAR_IT (__HANDLE__, __ INTERRUPT __)	Clears a specific peripheral interrupt status
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)	Gets a specific peripheral flag status
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)	Clears a specific peripheral flag status

STM32 Cube HAL IT handler és User Callbackek

- A ***HAL_PPP_IRQHandler()*** függvényt meg kell hívni az ***stm32f4xx_it.c***-ből
- User Call back függvények mindig **weak**-ként vannak létrehozva, tehát felüldefiniálhatóak a felhasználói kódban
- User Callback-ek

Callback functions	Example
HAL_PPP_MspInit() / _DeInit()	Ex: HAL_USART_MspInit() Called from HAL_PPP_Init() API function to perform peripheral system level initialization (GPIOs, clock, DMA, interrupt)
HAL_PPP_ProcessCpltCallback	Ex: HAL_USART_TxCpltCallback Called by peripheral or DMA interrupt handler when the process completes
HAL_PPP_ErrorCallback	Ex: HAL_USART_ErrorCallback Called by peripheral or DMA interrupt handler when an error occurs

STM32 Cube HAL API-k általános szerkezete

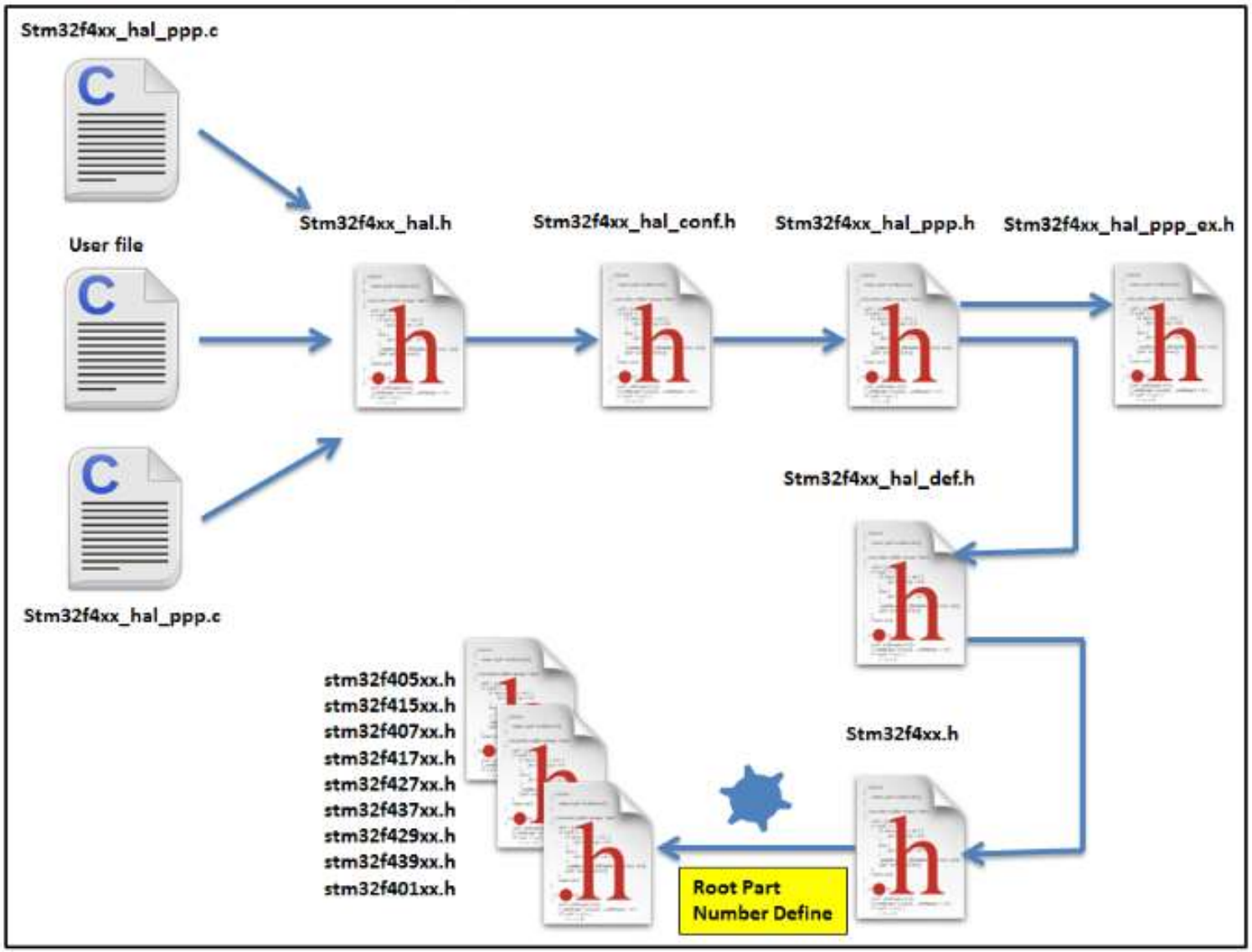
Csoport	Tipikus függvények	Leírás
Inicializációs függvények	**HAL_PPP_Init()** **HAL_PPP_DeInit()**	Alap inicializációs és de-inicializációs függvényei a perifériának
Periféria kimenet, bemenet kezelő függvények	**HAL_PPP_Read()** **HAL_PPP_Write()** **HAL_PPP_Transmit()** **HAL_PPP_Receive()** **HAL_PPP_...**	A periféria adatátviteléhez kapcsolódó függvények
Vezérlő függvények	**HAL_PPP_Set()** **HAL_PPP_Get()** **HAL_PPP_...**	A periféria konfigurációját vezérlő függvények
Állapot lekérdező és hibakezelő függvények	**HAL_PPP_GetState()** **HAL_PPP_GetError()** **HAL_PPP_...**	Állapot és hibainformáció lekérdezése

STM32 Cube HAL API-k szerkezete példa

Function Group	Common API Name	Description
Initialization group	*HAL_ADC_Init()*	This function initializes the peripheral and configures the low -level resources (clocks, GPIO, AF..)
	HAL_ADC_DeInit()	This function restores the peripheral default state, frees the low-level resources and removes any direct dependency with the hardware.
IO operation group	*HAL_ADC_Start ()*	This function starts ADC conversions when the polling method is used
	HAL_ADC_Stop ()	This function stops ADC conversions when the polling method is used
	HAL_ADC_PollForConversion()	This function allows waiting for the end of conversions when the polling method is used. In this case, a timout value is specified by the user according to the application.
	HAL_ADC_Start_IT()	This function starts ADC conversions when the interrupt method is used
	HAL_ADC_Stop_IT()	This function stops ADC conversions when the interrupt method is used
	HAL_ADC_IRQHandler()	This function handles ADC interrupt requests
	HAL_ADC_ConvCpltCallback()	Callback function called in the IT subroutine to indicate the end of the current process or when a DMA transfer has completed
	HAL_ADC_ErrorCallback()	Callback function called in the IT subroutine if a peripheral error or a DMA transfer error occurred
Control group	*HAL_ADC_ConfigChannel()*	This function configures the selected ADC regular channel, the corresponding rank in the sequencer and the sample time
	HAL_ADC_AnalogWDGConfig	This function configures the analog watchdog for the selected ADC
State and Errors group	*HAL_ADC_GetState()*	This function allows getting in runtime the peripheral and the data flow states.
	HAL_ADC_GetError()	This fuction allows getting in runtime the error that occurred during IT routine

STM32 Cube HAL Include file-ok szerkezete

- Elméletileg csak a Stm32f4xx_hal.h-ra van szükségünk (Cube F4 sorozat)



STM32 Cube HAL Közös erőforrások, részek

- A **stm32f4xx_hal_def.h** tartalmazza a közös típusokat definiciókat
  - **HAL Status:** visszajelző felsorolásos típus szinte az összes API használja

```
Typedef enum
{ HAL_OK = 0x00, HAL_ERROR = 0x01, HAL_BUSY = 0x02, HAL_TIMEOUT = 0x03
} HAL_StatusTypeDef;
```

  - **HAL Locked:** Az API által használt állapotjelző, ami segítségével a kölcsönös kizárás megvalósított az API-ban

```
typedef enum
{ HAL_UNLOCKED = 0x00, /*!<Resources unlocked */ HAL_LOCKED = 0x01 /*!< Resources
locked */
} HAL_LockTypeDef;
```

  - **CMSIS Device.h**-t is ez a file include-olja tovább
  - **Közös makrók:** HAL_MAX_DELAY konstans

STM32 Cube HAL konfigurálás

- A **stm32f4xx_hal_conf.h** tartalmazza az alap HAL beállítási paramétereket

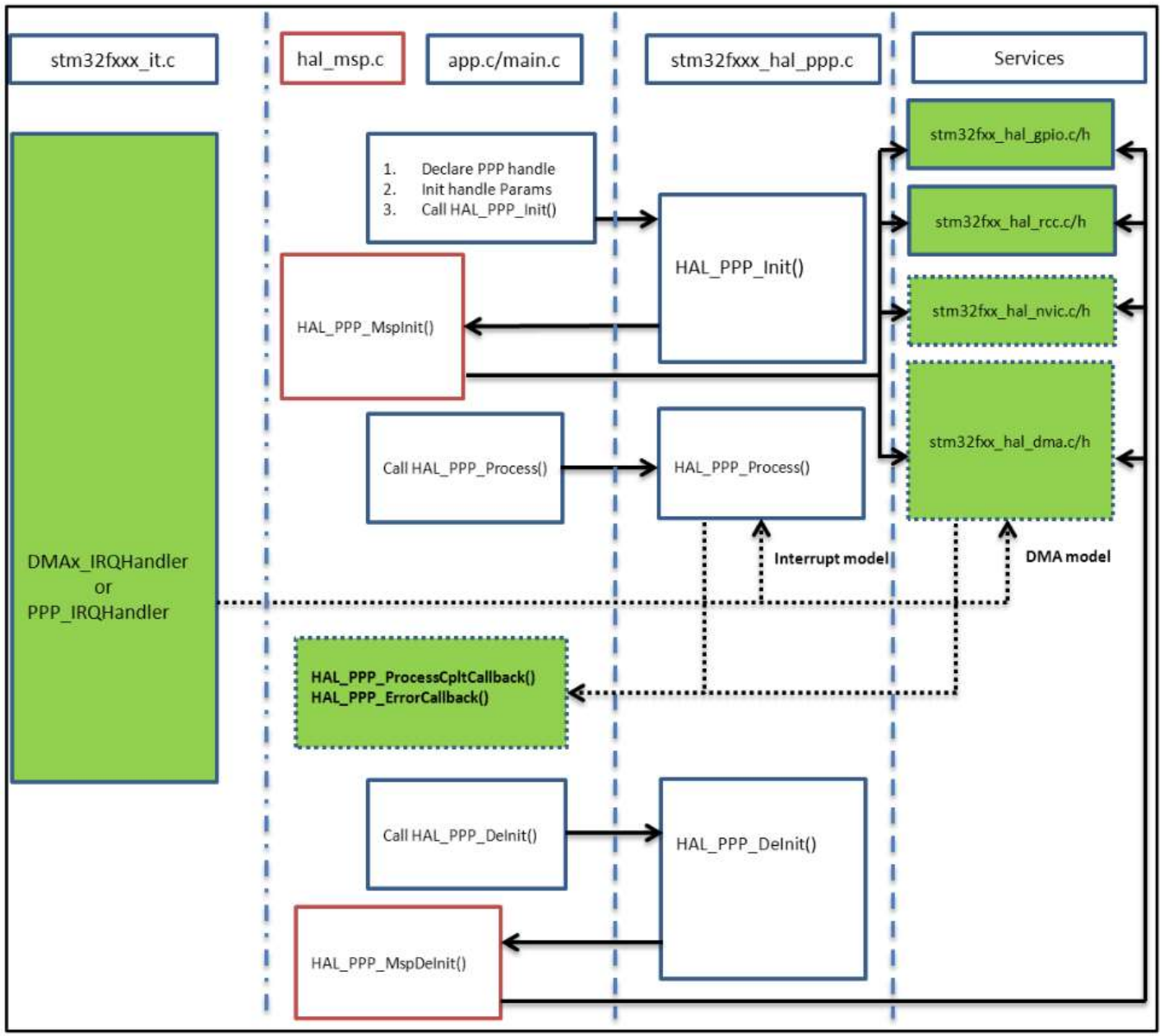
Configuration item	Description	Default Value
HSE_VALUE	Defines the value of the external oscillator (HSE) expressed in Hz. The user must adjust this define statement when using a different crystal value.	25 000 000 (Hz)
HSE_STARTUP_TIMEOUT	Timeout for HSE start up, expressed in ms	5000
HSI_VALUE	Defines the value of the internal oscillator (HSI) expressed in Hz.	16 000 000 (Hz)
EXTERNAL_CLOCK_VALUE	This value is used by the I2S/SAI HAL module to compute the I2S/SAI clock source frequency, this source is inserted directly through I2S_CKIN pad.	12288000 (Hz)
VDD_VALUE	VDD value	3300 (mV)
USE_RTOS	Enables the use of RTOS	FALSE (for future use)
PREFETCH_ENABLE	Enables prefetch feature	TRUE

STM32 Cube HAL konfigurálás folytatás

- A **stm32f4xx_hal_conf.h** tartalmazza az alap HAL beállítási paramétereket

Configuration item	Description	Default Value
INSTRUCTION_CACHE_ENABLE	Enables instruction cache	TRUE
DATA_CACHE_ENABLE	Enables data cache	TRUE
USE HAL_PPP_MODULE	Enables module to be used in the HAL driver	
MAC_ADDRx	Ethernet peripheral configuration : MAC address	
ETH_RX_BUF_SIZE	Ethernet buffer size for receive	ETH_MAX_PACKET_SIZE
ETH_TX_BUF_SIZE	Ethernet buffer size for trasmit	ETH_MAX_PACKET_SIZE
ETH_RXBUFNB	The number of Rx buffers of size ETH_RX_BUF_SIZE	4
ETH_TXBUFNB	The number of Tx buffers of size ETH_RX_BUF_SIZE	4
DP83848_PHY_ADDRESS	DB83848 Ethernet PHY Address	0x01
PHY_RESET_DELAY	PHY Reset delay these values are based on a 1 ms Systick interrupt	0x000000FF
PHY_CONFIG_DELAY	PHY Configuration delay	0x00000FFF
PHY_BCR PHY_BSR	Common PHY Registers	
PHY_SR PHY_MICR PHY_MISR	Extended PHY registers	

STM32 Cube HAL felhasználási modell



STM32 Cube HAL inicializáció

- **HAL_Init()**
  - Inicializálja data/instruction cache és pre-fetch queue
  - Systick timer beállítása a 1ms-os órajelre (HSI óra)
  - IT prioritás csoportok és preempció beállítása
  - Meghívja a HAL_MspInit() callback-et a szükséges egyéb inicializációkhoz
- HAL_MspInit() “weak” ként van létrehozva

STM32 Cube HAL órajel konfiguráció

- Tipikus órajelkonfiguráció: kell, mert a SysInit nem csinálja meg

```
static void SystemClock_Config(void)
{
RCC_ClkInitTypeDef RCC_ClkInitStruct;
RCC_OscInitTypeDef RCC_OscInitStruct;
/* Enable HSE Oscillator and activate PLL with HSE as source */
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 = 25;
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 7;
HAL_RCC_OscConfig(&RCC_OscInitStruct);
/* Select 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_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5); }
```

STM32 Cube HAL MSP Inicializáció

- **HAL_PPP_Init()** elvégzi a periféria inicializációt az egyes erőforrásokat amiket a periféria használ (GPIO láb stb.) pedig a **HAL_PPP_MspInit()**-ben kell elvégezni.

Routine	Description
void HAL_MspInit()	Global MSP initialization routine
void HAL_MspDeInit()	Global MSP de-initialization routine
void HAL_PPP_MspInit()	PPP MSP initialization routine
void HAL_PPP_MspDeInit()	PPP MSP de-initialization routine

STM32 Cube HAL Működési módok: Polling

- **Blokkolás alapú végrehajtás.** A funkció végrehajtódott, ha a HAL_OK –értékkel visszatér a függvény. Timeout megadható.

- Blokkoló függvények tipikus belső felépítése

```
      HAL_StatusTypeDef HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle, uint8_t
pData,
int16_tSize,uint32_tTimeout)
{
if((pData == NULL ) || (Size == 0))
{
return HAL_ERROR;
}
(…) while (data processing is running)
{
if( timeout reached )
{
return HAL_TIMEOUT;
}
```

STM32 Cube HAL Timeout kezelés

- A legtöbb esetben olyan API-k használják, amik polling módban működnek.

Timeout value	Description
0	No poll : Immediate process check and exit
1 ... (HAL_MAX_DELAY -1)(1)	Timeout in ms
HAL_MAX_DELAY	Infinite poll till process is successful

STM32 Cube HAL Működési módok: IT alapú kiszolgálás

- A HAL függvény visszatér miután elindította az adott folyamatot és engedélyezte az ahhoz tartozó IT-t.
- **Az átvitel végéről egy Callback függvény segítségével értesít minket a rendszer** (Alapból weak tehát felüldefiniálható).
- Az IT alapú működéshez 4 függvény minta tartozik
  - *HAL_PPP_Process_IT():* elindítja az adatátvitelt
  - *HAL_PPP_IRQHandler():* A periféria IT handlerje (alacsony szint): stm32f4xx_it.c-ben kell meghívni.
  - *__weak HAL_PPP_ProcessCpltCallback ():* Az átvitel végét jelző Callback
  - *__weak HAL_PPP_ProcessErrorCallback():* Átviteli hibát jelző Callback

STM32 Cube HAL Működési módok: IT alapú kiszolgálás

- Main.c:

```
      UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART3;
HAL_UART_Init(&UartHandle);
HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
}
```

- *stm32f4xx_it.c:*

```
      extern UART_HandleTypeDef UartHandle;
void USART3_IRQHandler(void)
{
HAL_UART_IRQHandler(&UartHandle);
 }
```

STM32 Cube HAL Működési módok: DMA alapú

- A HAL függvény visszatér miután elindította az adatátviteli folyamatot DMA segítségével.
- Az adatátviteli folyamat végéről egy DMA IT segítségével kapunk értesítést
- A DMA inicializáció a *HAL_PPP_MspInit()* függvényben kell, hogy megtörténjen: Össze kell rendelni a DMA-t a PPP-vel
- A DMA alapú működéshez 4 függvény minta tartozik
  - *HAL_PPP_Process_DMA():* elindítja az adatátvitelt
  - *HAL_PPP_DMA_IRQHandler():* A periféria IT handlerje (alacsony szint): stm32f4xx_it.c-ben kell meghívni.
  - *__weak HAL_PPP_ProcessCpltCallback ():* Az átvitel végét jelző Callback
  - *__weak HAL_PPP_ProcessErrorCallback():* Átviteli hibát jelző Callback

STM32 Cube HAL Működési módok: DMA alapú

- Main.c:

```
typedef struct
{
PPP_TypeDef *Instance; /* Register base address */
PPP_InitTypeDef Init; /* PPP communication parameters */
HAL_StateTypeDef State; /* PPP communication state */
(…)
DMA_HandleTypeDef *hdma; /* associated DMA handle */
} PPP_HandleTypeDef;
```

```
int main(void)
{
/* Set User Parameters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = UART3;
HAL_UART_Init(&UartHandle);
(..)
}
void HAL_USART_MspInit (UART_HandleTypeDef * huart)
{
static DMA_HandleTypeDef hdma_tx;
static DMA_HandleTypeDef hdma_rx;
(…)
 __HAL_LINKDMA(UartHandle, DMA_Handle_tx, hdma_tx);
     __HAL_LINKDMA(UartHandle, DMA_Handle_rx, hdma_rx);
(…)
}
```

STM32 Cube HAL Működési módok: DMA alapú

- Main.c:

```
UART_HandleTypeDef UartHandle;
int main(void)
{
/* Set User Paramaters */
UartHandle.Init.BaudRate = 9600;
UartHandle.Init.WordLength = UART_DATABITS_8;
UartHandle.Init.StopBits = UART_STOPBITS_1;
UartHandle.Init.Parity = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
UartHandle.Init.Mode = UART_MODE_TX_RX;
UartHandle.Init.Instance = USART3;
HAL_UART_Init(&UartHandle);
HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
}
```

- *stm32f4xx_it.c:*

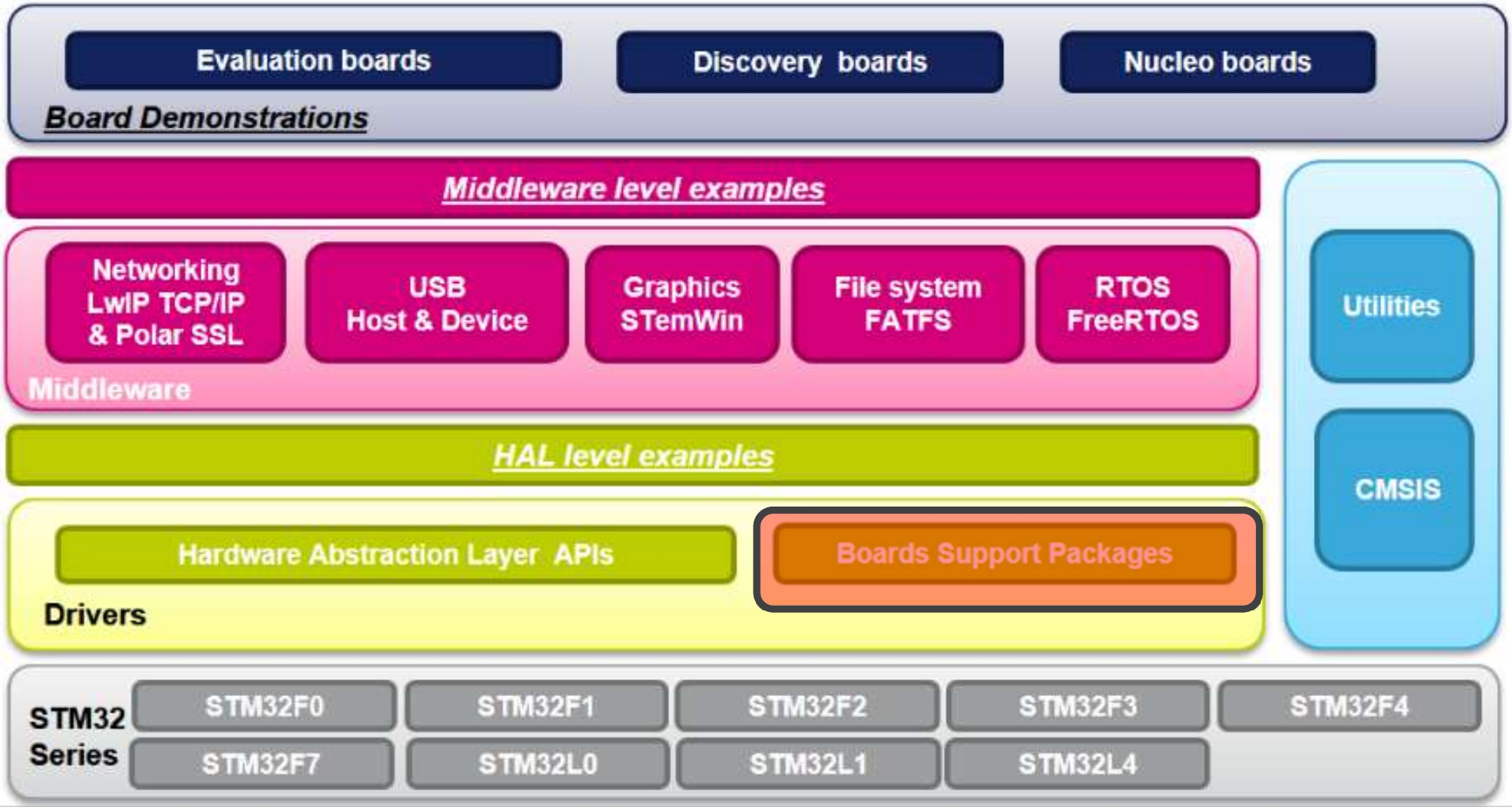
```
extern UART_HandleTypeDef UartHandle;
void DMAx_IRQHandler(void)
{
HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}
```

STM32 Cube HAL Működési módok: DMA alapú

- A DMA transfer complete IT-t még össze kell rendelnünk a Callback-ekkel

```
HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params….)
{
(…)
hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
(…)
}
```

STM32 Cube architektúra



STM32 Cube: BSP Board Support Package

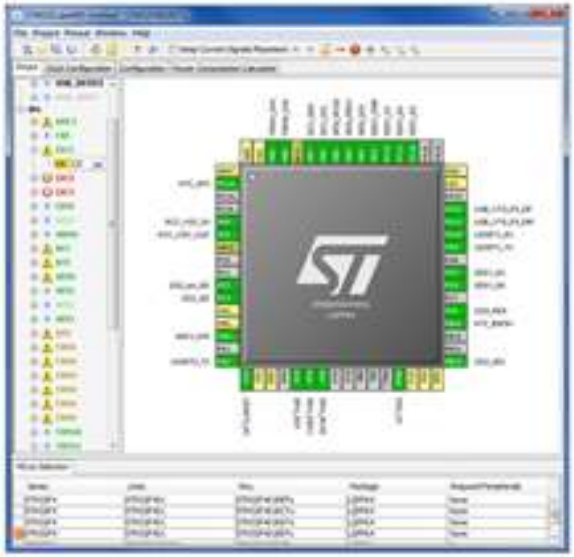
- Az adott fejlesztőkártya függő API-kat tartalmazza
- A fejlesztőkártyán lévő külső perifériák kezelésének támogatása
  - Audio Codec
  - MEMS gyorsulásmérő

- Elnevezési konvenció
  - BSP_FUNCT_Action()
    - BSP_LED_Init()
    - BSP_LED_On()

STMCube32 Mx

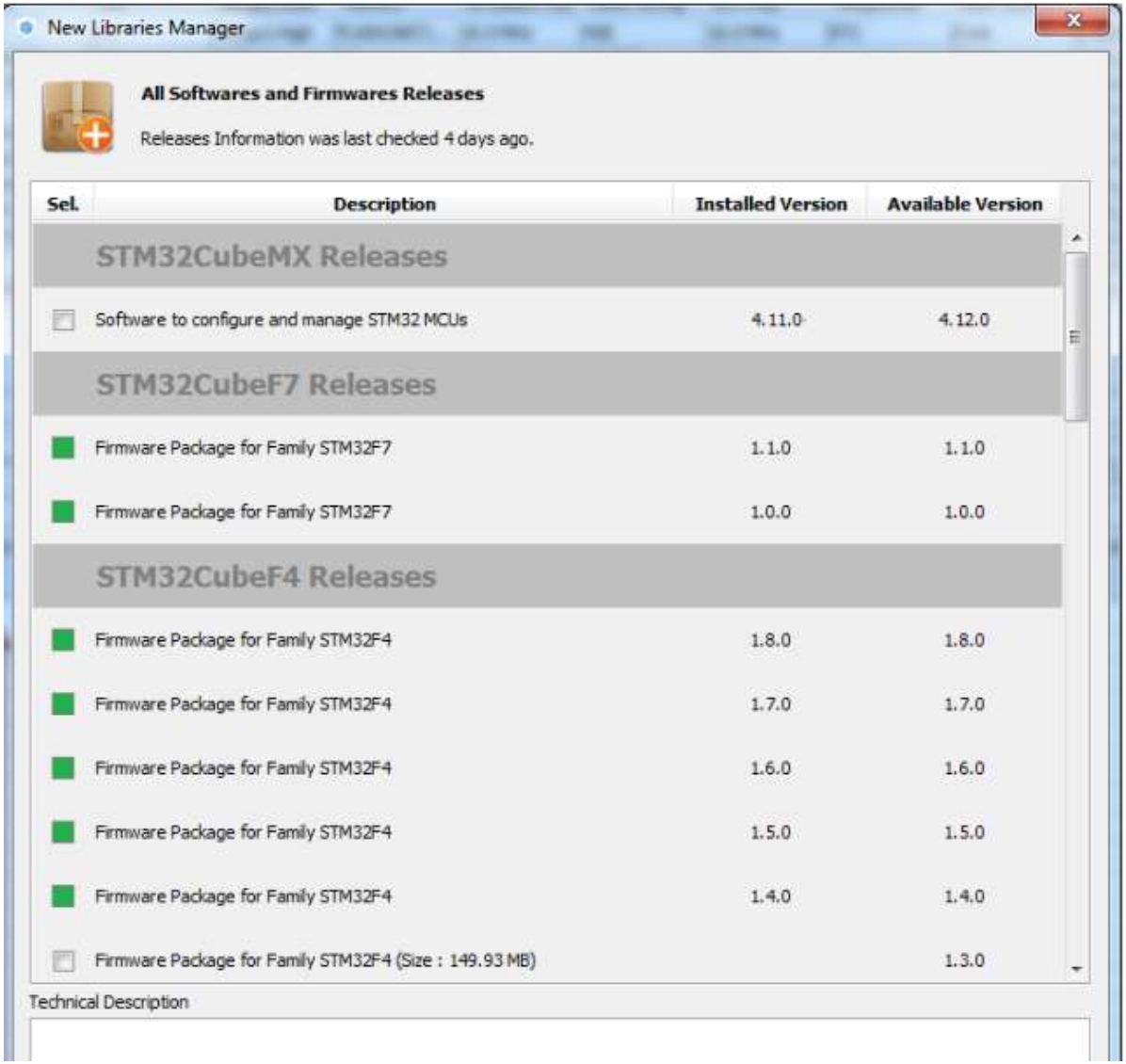
STMCube32 Mx

- Grafikus periféria, és órajel inicializáló rendszer, fogyasztás kalkulációval



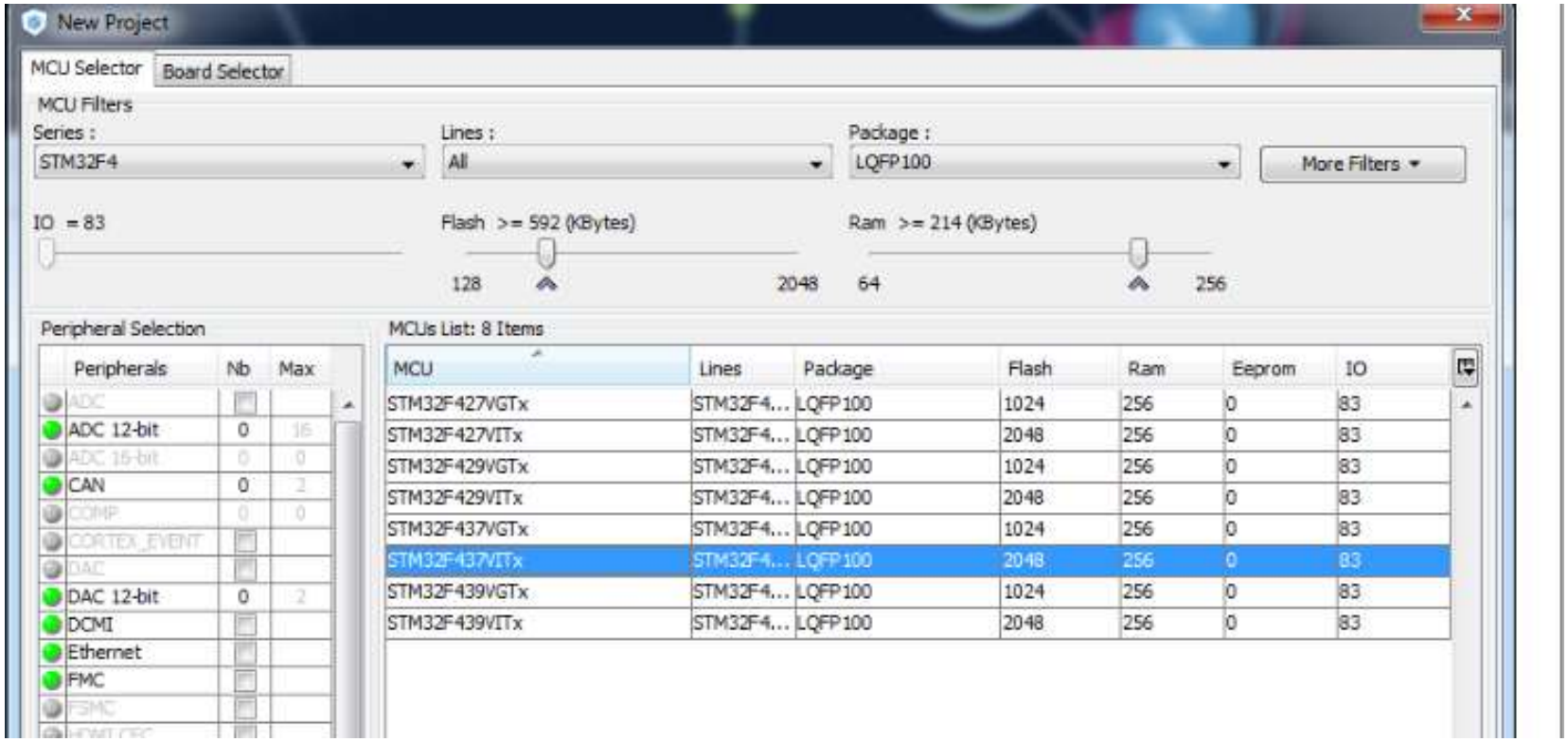
STMCube32 Mx Library menedzser nézet

- Library-k állapotának áttekintése

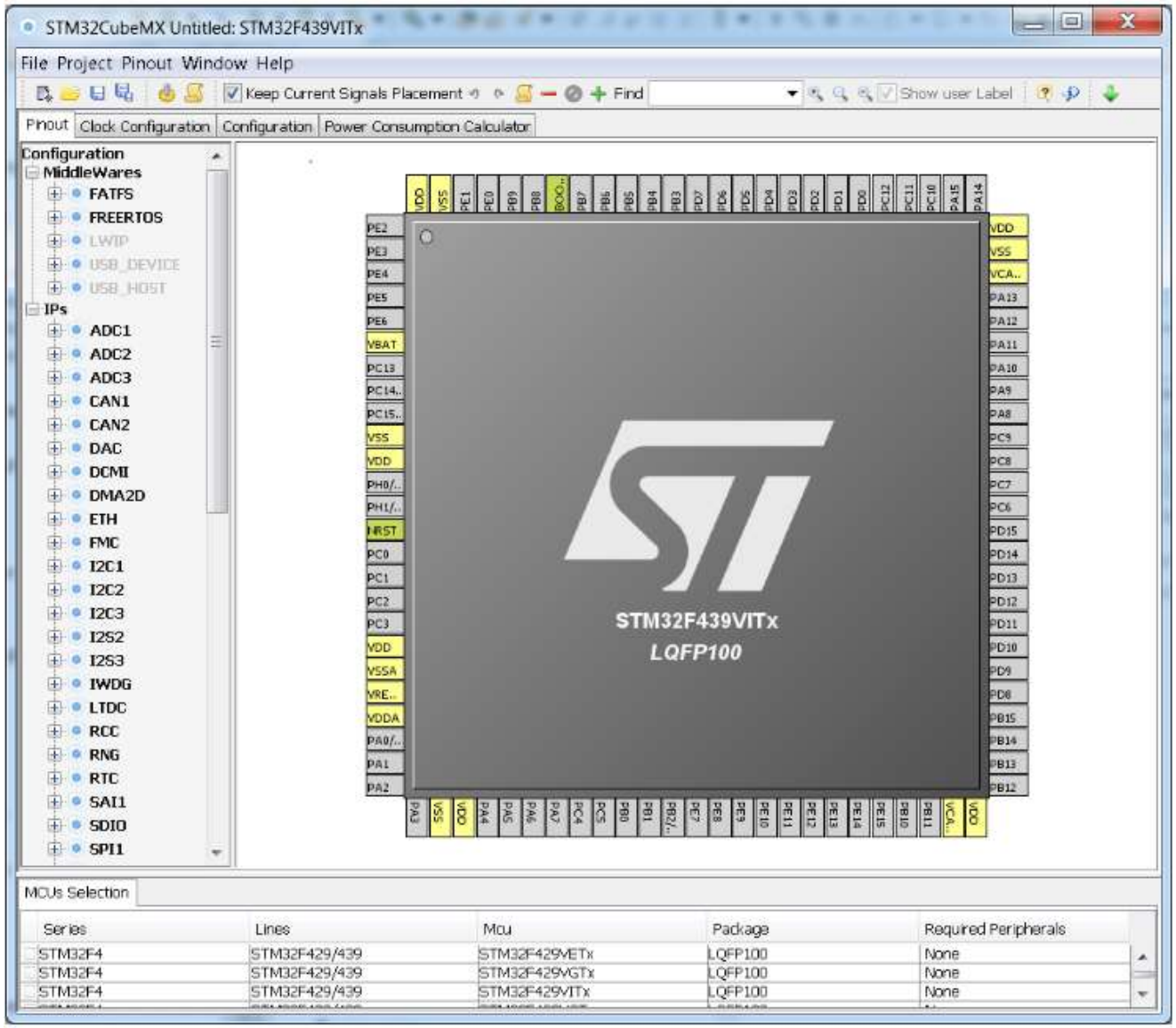


STMCube32 Mx Új konfiguráció

- MCU kiválasztása

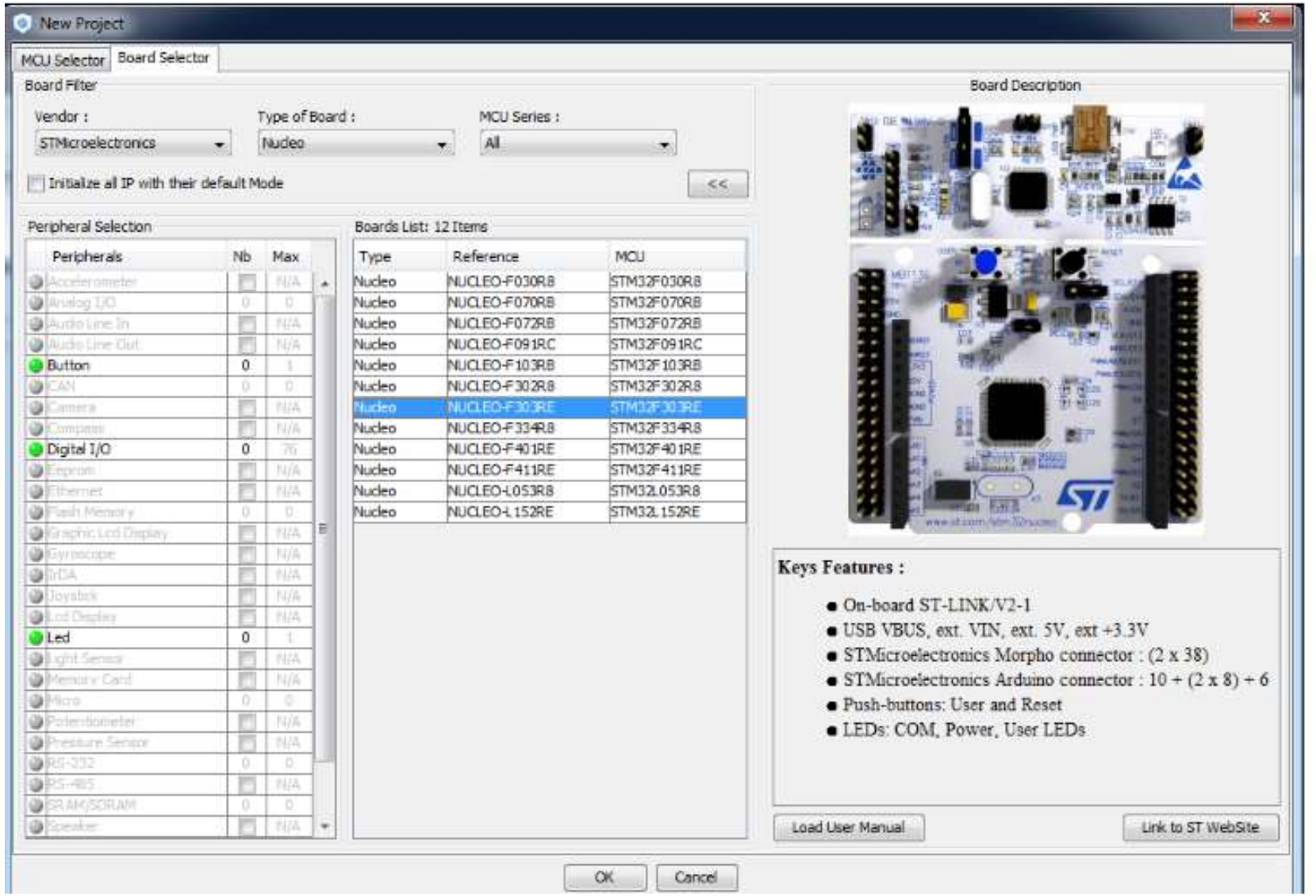


STMCube32 Mx alap konfigurációs ablak MCU módban



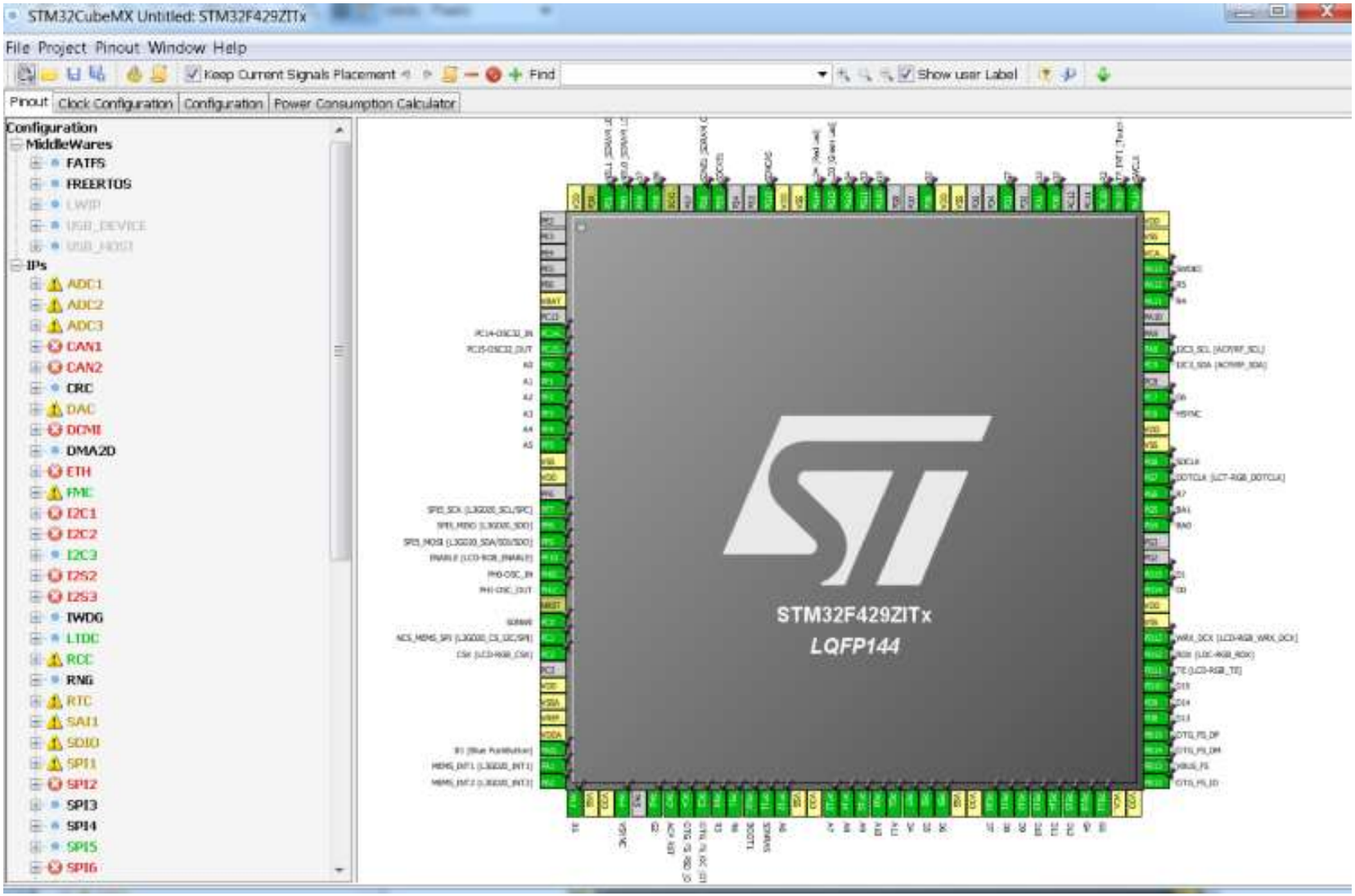
STMCube32 Mx Új konfiguráció

- Board kiválasztása



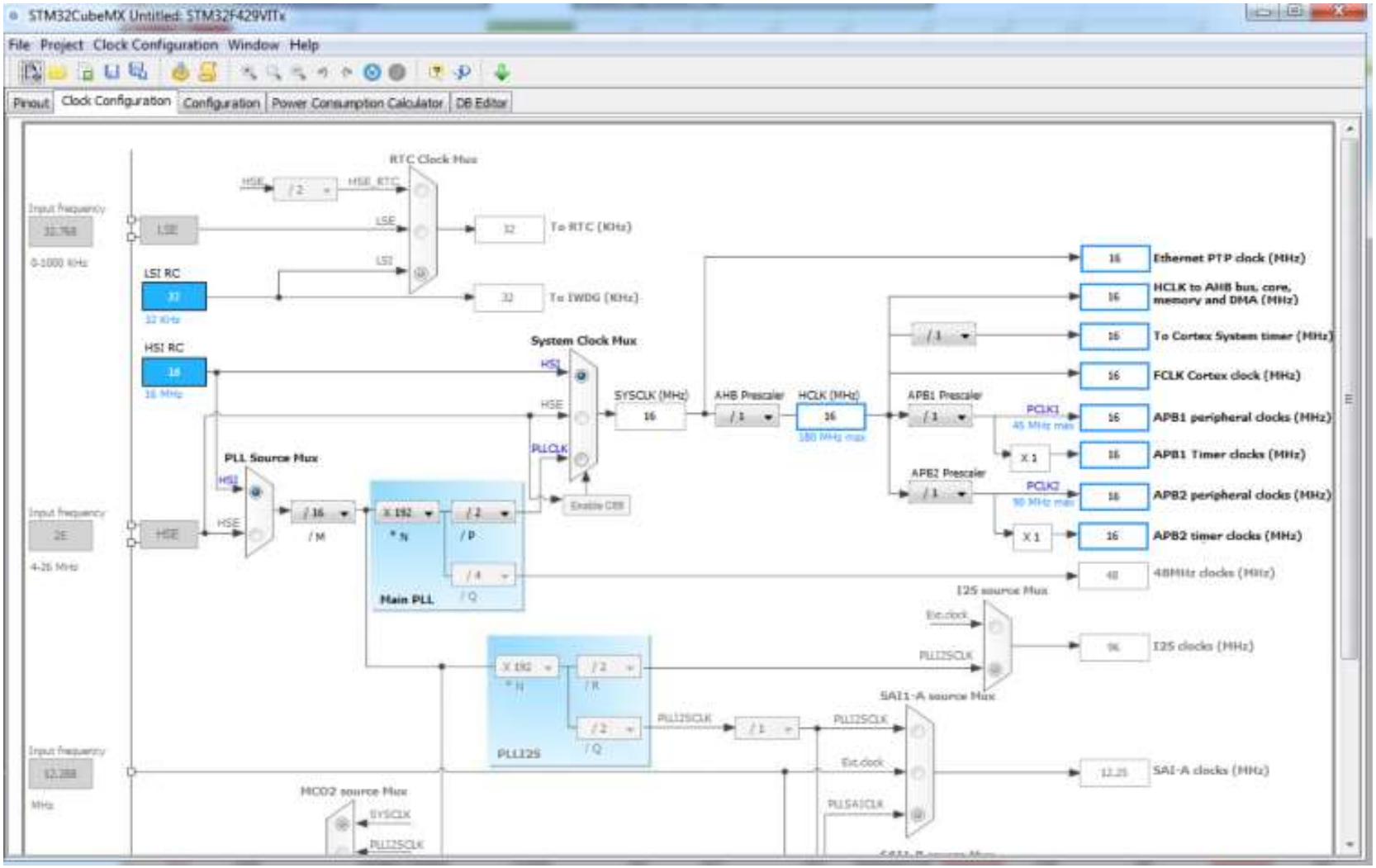
STMCube32 Mx alap konfigurációs ablak Board módban

- Lehet egy alap periféria konfigurációt használni



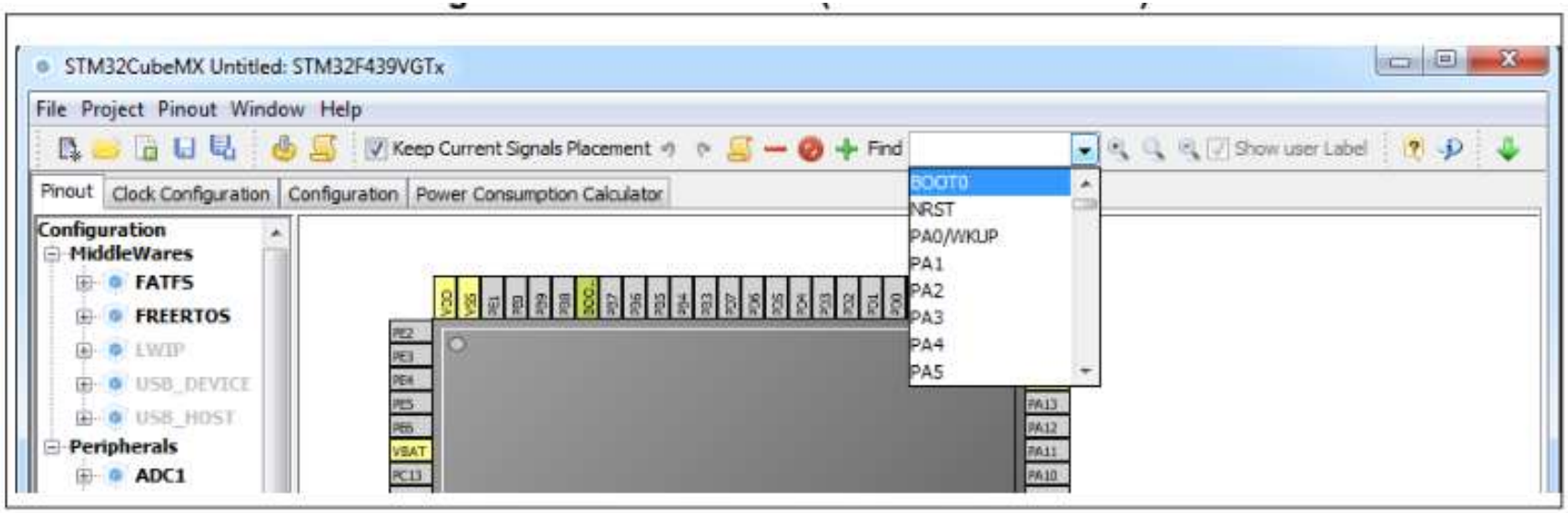
STMCube32 Mx órajel konfiguráció

- Az egész órajel Tree beállítása



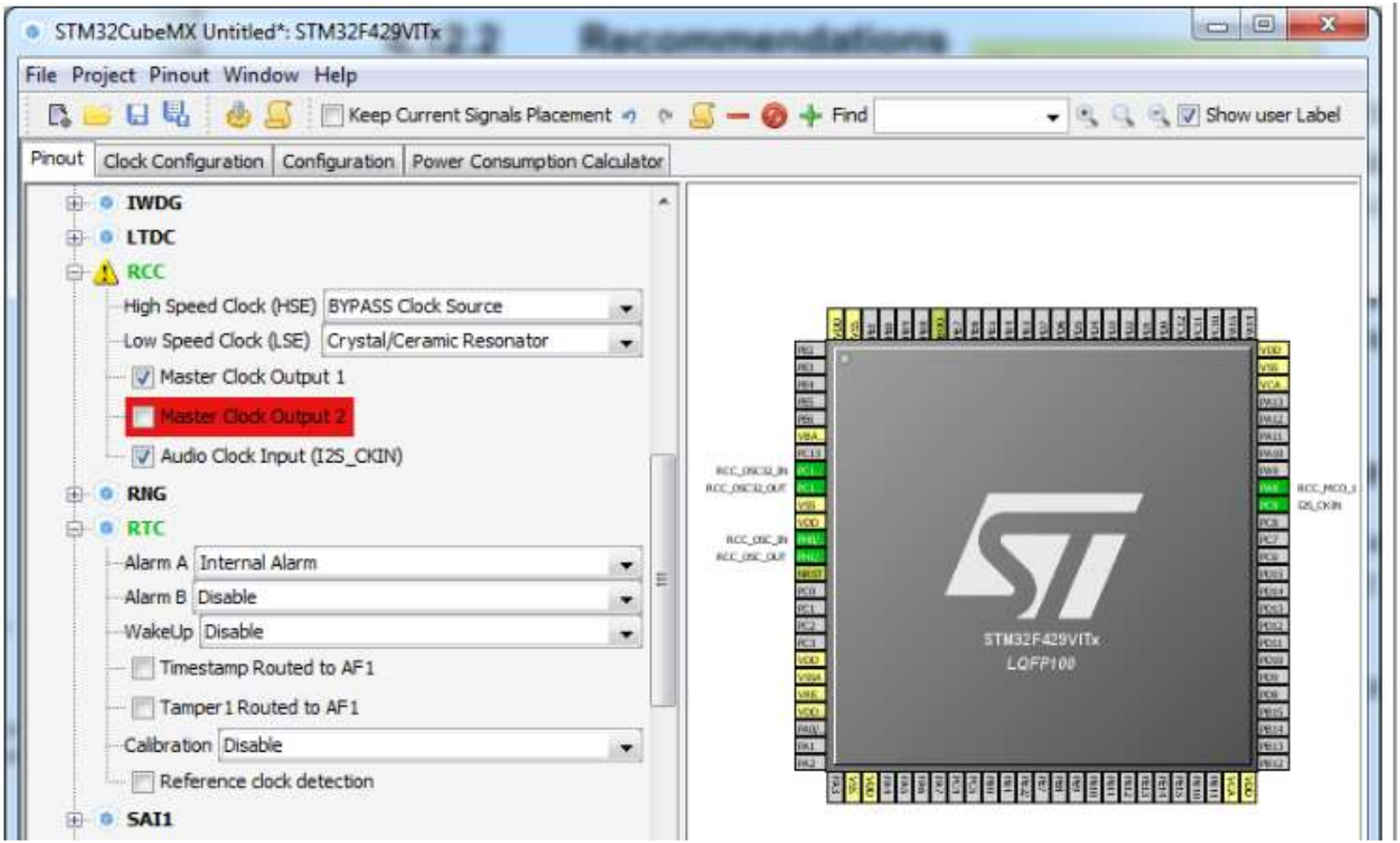
STMCube32 Mx konfiguráció

- Lábak állapotának konfigurálása



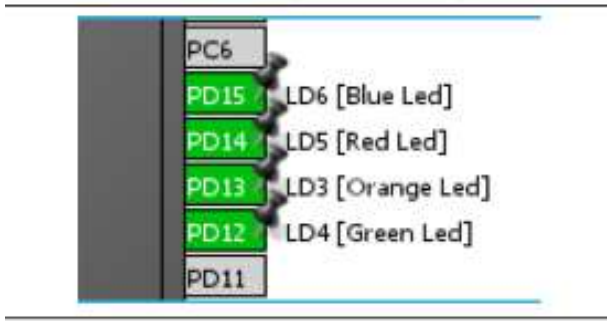
STMCube32 Mx konfiguráció

- Periféria beállítások



STMCube32 Mx kódgenerálás

- **stm32f4xx_hal_conf.h:** HAL modulok engedélyezése
- **stm32f4xx_hal_msp.c:** Inicializációs függvények
- **main.c:** System Clock, GPIO konfiguráció
- **mxconstants.h:** Cubemx konstans definiciók




```
#define MyTimeOut       10
#define LD4_Pin         GPIO_PIN_12
#define LD4_GPIO_Port   GPIOD
#define LD3_Pin         GPIO_PIN_13
#define LD3_GPIO_Port   GPIOD
#define LD5_Pin         GPIO_PIN_14
#define LD5_GPIO_Port   GPIOD
#define LD6_Pin         GPIO_PIN_15
```