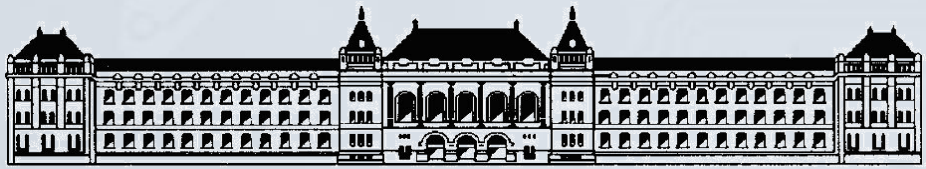
Szoftverfejlesztés STM32 platformon

szeminárium

2024/2025 őszi félév

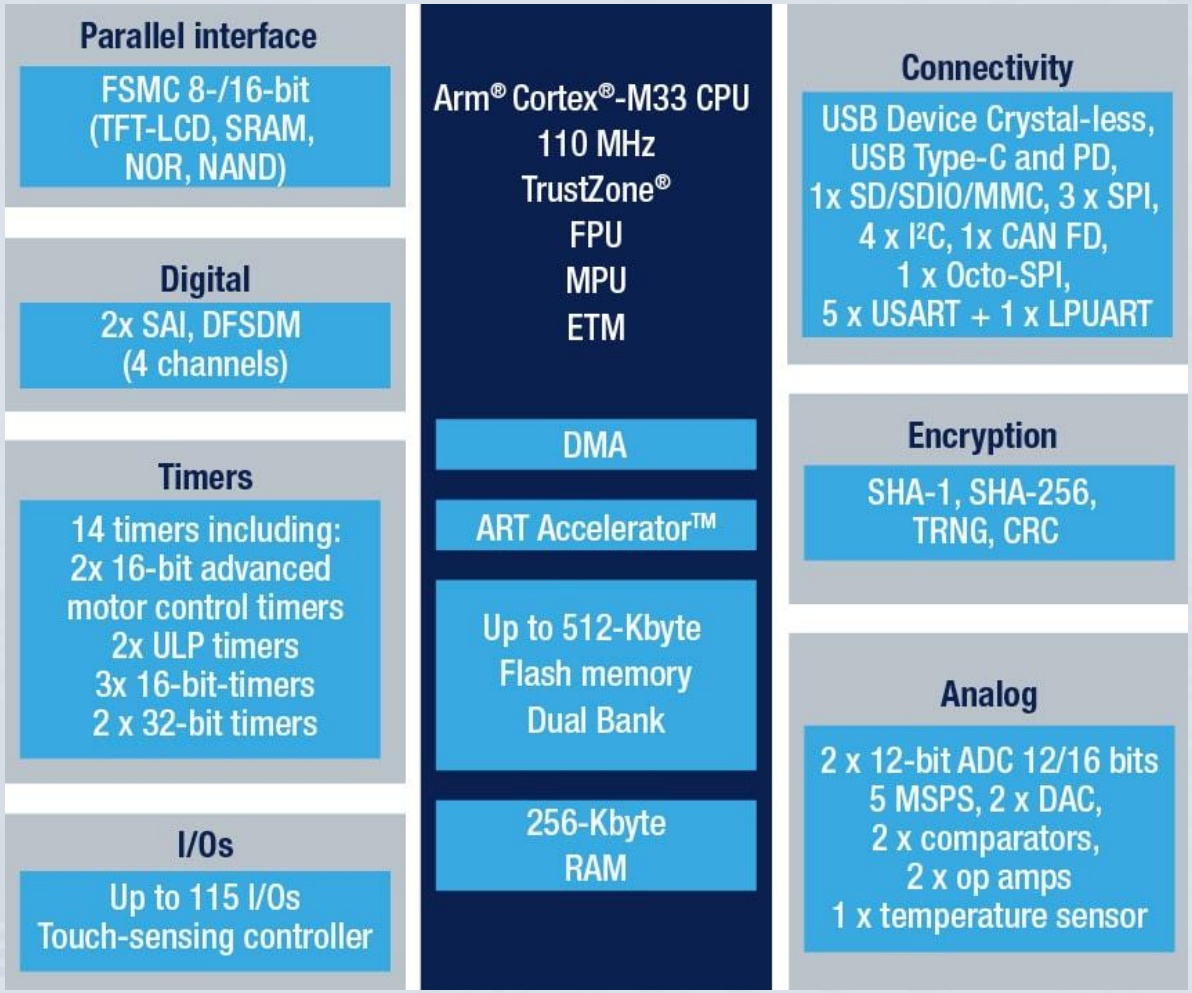
Szabó Zoltán



BME Automatizálási és Alkalmazott Informatikai Tanszék

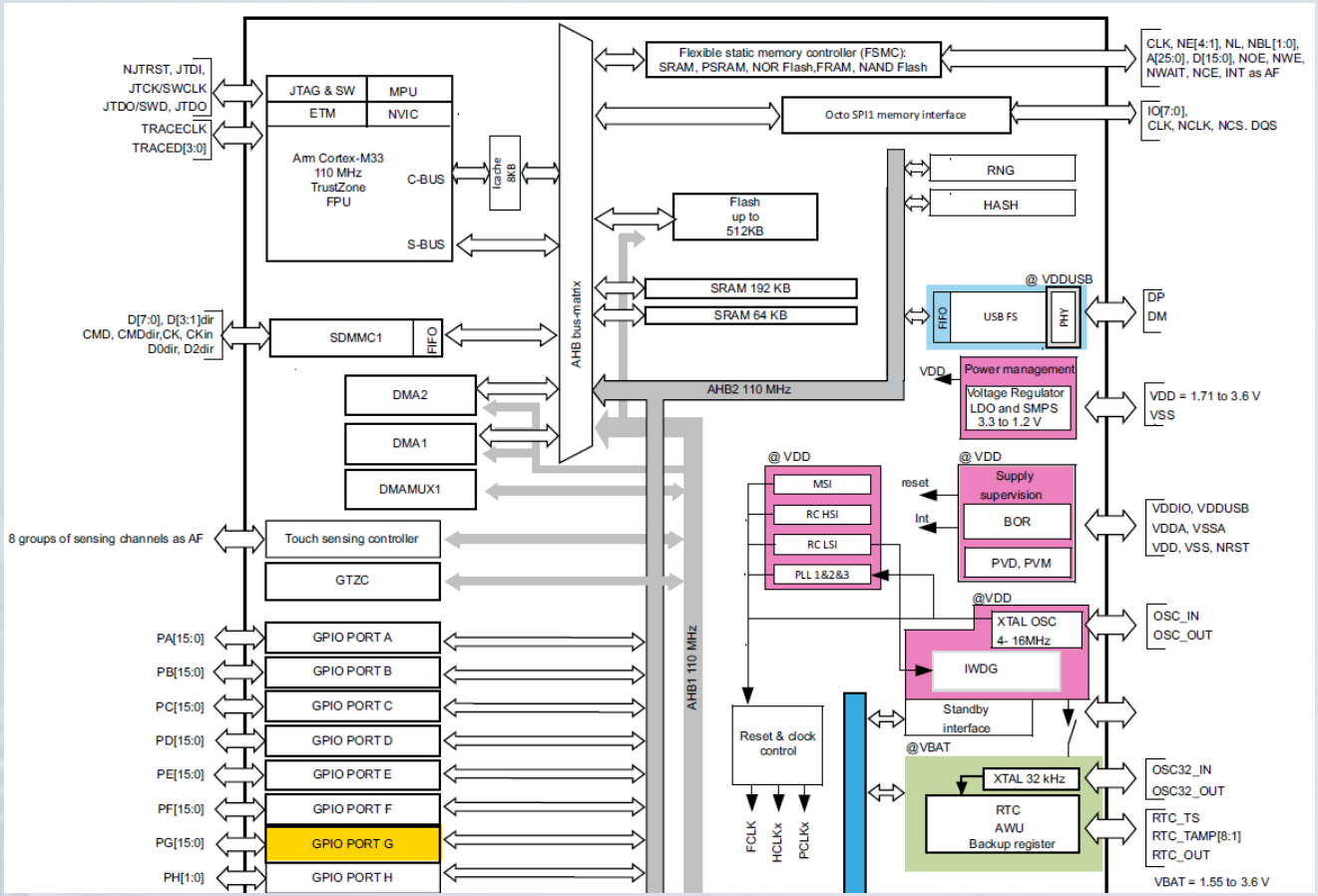
Szoftverfejlesztés ARM rendszerre

STM32L5 sorozatú MCU



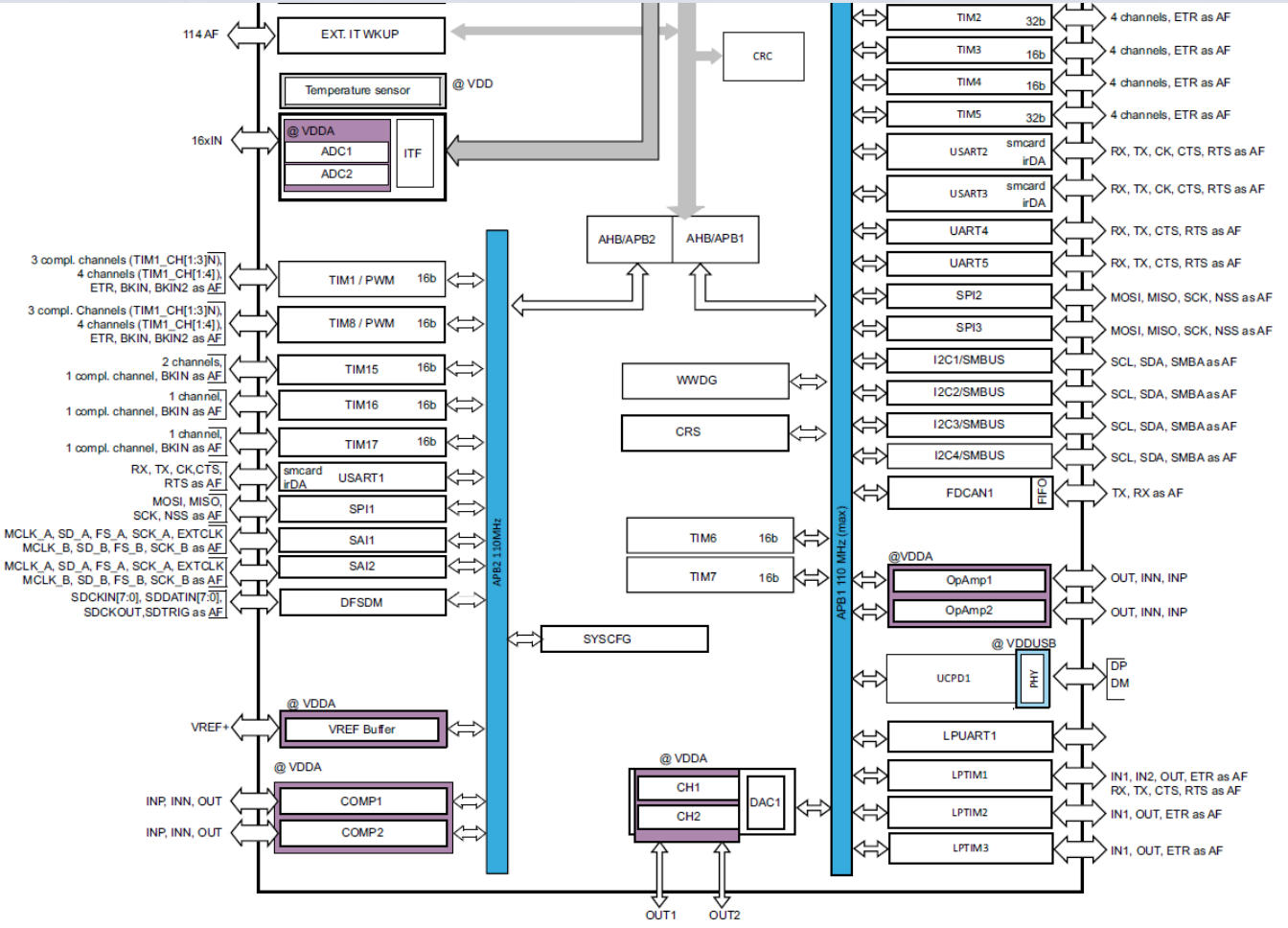
Szoftverfejlesztés ARM rendszerre (2)

STM32L552ZE MCU



Szoftverfejlesztés ARM rendszerre (3)

STM32L552ZE MCU



Szoftverfejlesztés ARM rendszerre (4)

STM32L552ZE Nucleo

- STM32L5 mikrokontroller
- 3 LED
- 1 gomb
- SWD Debugger
- Stb.

Szoftverfejlesztés ARM rendszerre (5)

Fejlesztőeszközök:

- Eclipse + GCC
- System Workbench for STM32 (OpenSTM32)
- IAR Embedded Workbench
- Keil µVision
- **STM32CubeIDE**
- …

Szoftverfejlesztés ARM rendszerre (6)

Rendelkezésünkre áll:

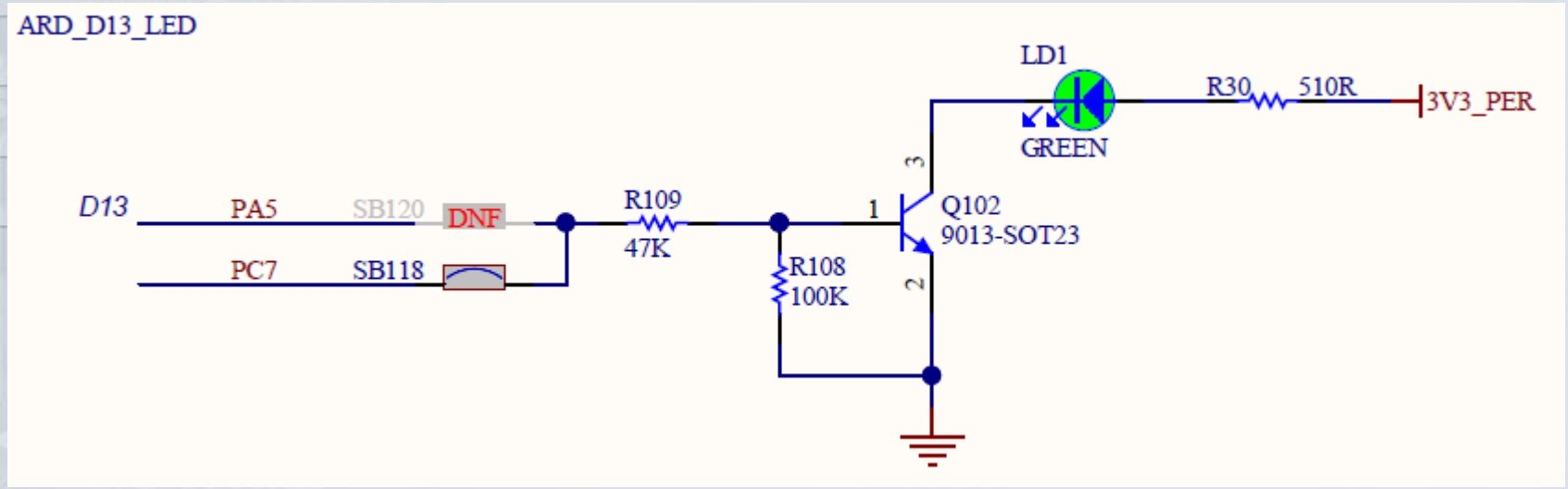
- Cortex M33 dokumentáció (ARM)
- Processzor és a perifériák dokumentációja (ST)
- Periféria függvénykönyvtár – ST
- Rengeteg mintapélda – ST
- Fejlesztőkörnyezet – STM32CubeIDE

Mintapélda - GPIO

Legegyszerűbb feladat:

Villogó LED

A LED a PC7 vonalon van



Mintapélda – GPIO (2)

Mit kell tudnunk:

1. Fogyasztáscsökkentési okokból a perifériák nem kapnak órajelet – be kell kapcsolni
2. A GPIO lábak alapértelmezés szerint
   - 16 bites portokba vannak szervezve (GPIOA,GPIOB, … GPIOH)
   - Bemenetek (legalacsonyabb fogyasztás, összehajtások elkerülése)

Mintapélda – GPIO (3)

Mit kell tennünk:

0. Mikrokontroller órajelének beállítása
1. Periféria órajelének engedélyezése
2. Port inicializálás
    1. GPIO port sebesség beállítása
    2. Kimenet legyen
    3. Push-pull mód
    4. Belső felhúzó ellenállások letiltva
3. LED-ek ki/be kapcsolása, közben várakozás

Mintapélda – GPIO (4)

Milyen lehetőségeink vannak?

1.Dokumentáció alapos tanulmányozása, a megfelelő konfigurációs regiszterek közvetlen beállítása

- ARM Cortex M33 Technical Reference Manual (~130 oldal)
- ARM Cortex M33 Generic User Guide (~315 oldal)
- STM32L5 Datasheet (~340 oldal)
- STM32L5 Reference manual (~2200 oldal)

2. Adatlap tanulmányozása, periféria függvénykönyvtár használata a regiszterek beállításához☺

- Perifériák Inicializálásához szükséges kódot az STM32CubeIDE fejlesztőeszköz képes generálni

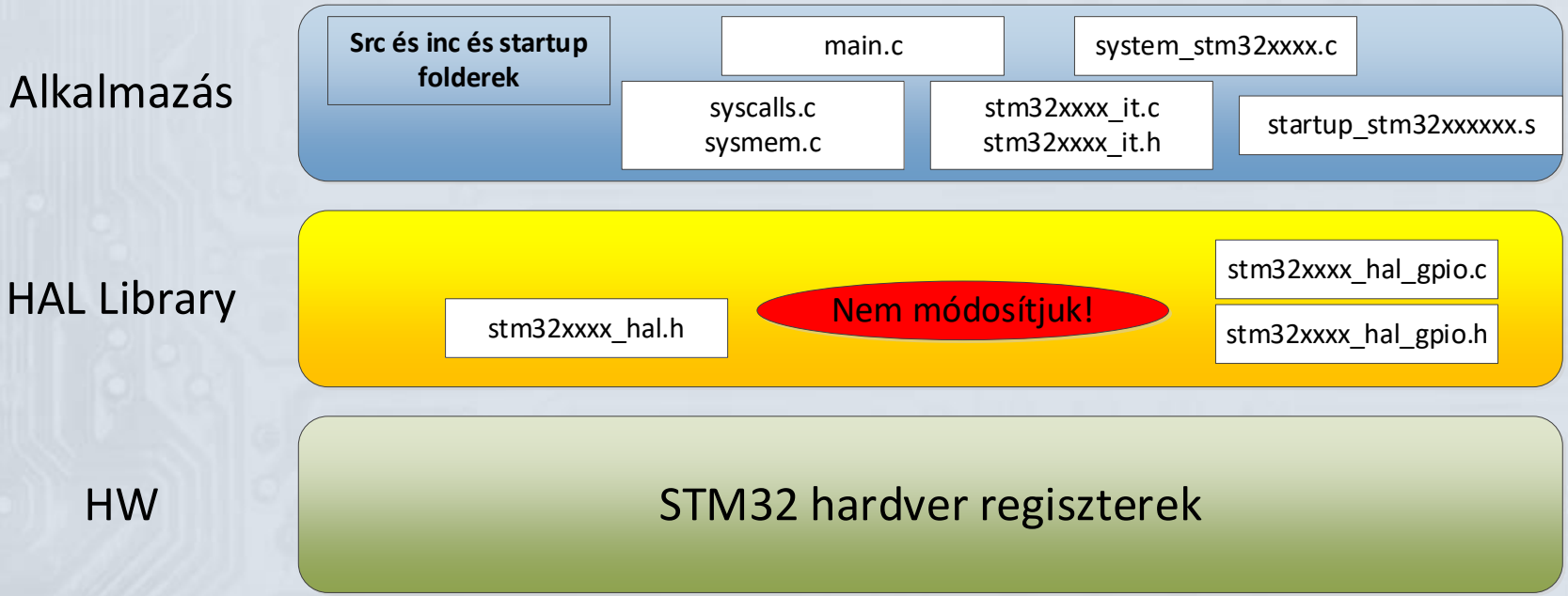
Mintapélda – GPIO (5)

ST Periféria függvénykönyvtár (STM32 HAL):

- Minden periféria csoporthoz külön c és h fájl, a fájlok elnevezése utal a perifériára pl: stm32l5xx_hal_gpio.c
- Nem kell a regisztereket fejből tudni, beszédes nevű függvényeket kell meghívni pl: HAL_GPIO_Init
- Természetesen a perifériák működésével tisztában kell lenni
- Forráskód szinten kapjuk – ötleteket lehet belőle nyerni

Mintapélda – GPIO (6)

Az alkalmazásunk felépítése:



Mintapélda – GPIO (7)

Az alkalmazásunk felépítése:

- main.c
  - a main függvényünket tartalmazó fájl
- stm32l5xx_it.h, stm32l5xx_it.c
  - Megszakításkezelő függvények
- startup_stm32fl552xx.s, system_stm32l5xx.c
  - alapvető hardver inicializálás
- syscalls.c, sysmem.c
  - alapvető C rendszerhívások (I/O, heap)

Mintapélda – GPIO (8)

Milyen függvényekre lesz szükségünk:

Órajel engedélyezés – RCC (Reset and Clock Configuration) modul (stm32l5xx_hal_rcc)

- Meg kell néznünk, hogy melyik porton vannak a LED-ek (GPIOC)
- Keresünk hozzá függvényt v. makrót, ami engedélyezi:

```
__GPIOD_CLK_ENABLE();
```

- Milyen paraméterekkel hívjuk meg?
  - Mint láthatjuk nincs paramétere☺

Mintapélda – GPIO (9)

Milyen függvényekre lesz szükségünk:

GPIO inicializálás – GPIO modul (stm32l5xx_hal_gpio)

```
void HAL_GPIO_Init(GPIO_TypeDef* GPIOx,
      GPIO_InitTypeDef* GPIO_Init)
```

- Milyen paraméterekkel hívjuk meg?
  1. GPIOC – a könyvtárban egy define (egy memóriacím)
  2. Ez egy struktúra, amit nekünk kell kitöltenünk

Mintapélda – GPIO (10)

Milyen függvényekre lesz szükségünk:

```
typedef struct
{
  uint32_t Pin;                    /* GPIO Pin */
  uint32_t Mode;                   /* Mode */
  uint32_t Speed;                  /* Speed */
  uint32_t Alternate;              /* Alternate function */
  uint32_t Pull;                   /* Pull-up/Pull down */
}GPIO_InitTypeDef;
```

- A legtöbb tag beszédes nevű define használatával tölthető ki pl.: *GPIO_MODE_OUTPUT_PP, GPIO_NOPULL*

Mintapélda – GPIO (11)

Milyen függvényekre lesz szükségünk:

GPIO értékbeállítás – GPIO modul (stm32l5xx_hal_gpio)

- Keressünk függvényt hozzá:

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);

void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

- Paraméterek: Port, Pin, Érték (értelem szerint)

Mintapélda – GPIO

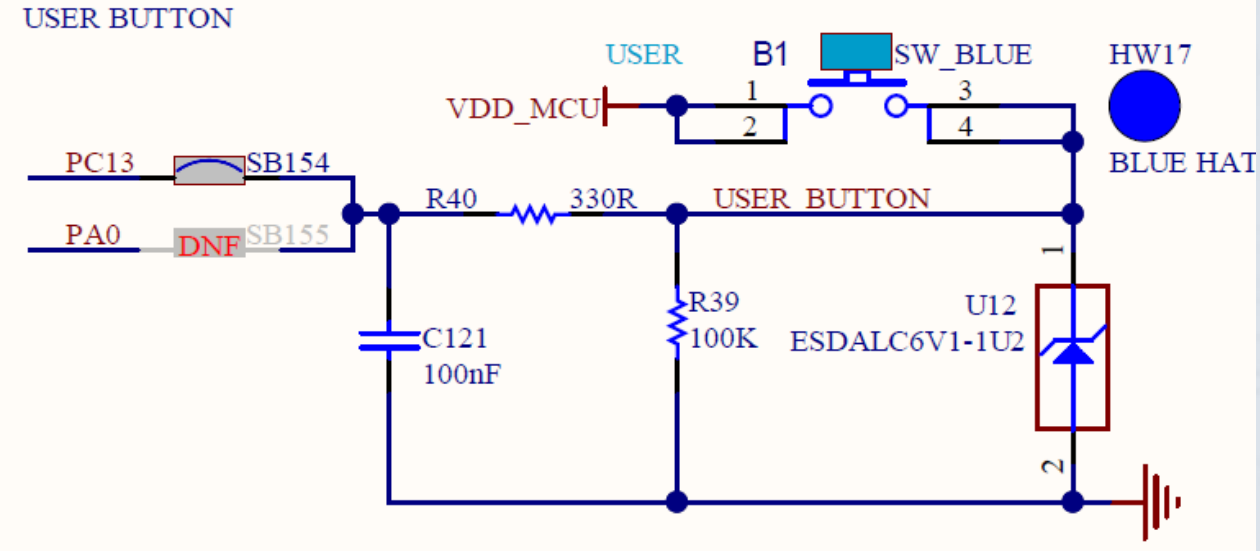
Írjuk meg az alkalmazást!

Mintapélda – GPIO, EXTI

Fejlesszük tovább a projektünket:

Villogó LED, gombnyomásra megáll, következőre újraindul

A nyomógomb a PC13 vonalon van!



Mintapélda – GPIO, EXTI (2)

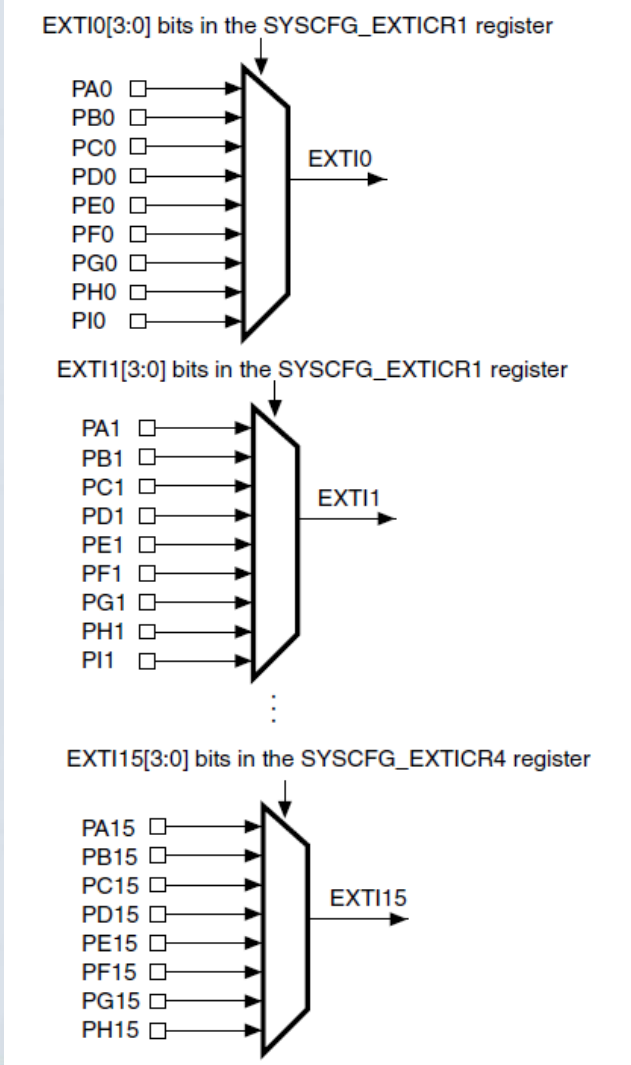
Mit kell tennünk:

1. GPIO beállítás (külső megszakításra)
2. Megszakítás prioritásának beállítása
3. Megszakítás engedélyezése
4. Külső megszakítás kezelése

Mintapélda – GPIO, EXTI (3)

Külső megszakítások (EXTI):

- Minden láb lehet megszakítás forrása
- A megszakítás vezérlőre 16 általános célú külső megszakításvonal kapcsolódik
- Konfigurálható, hogy mely bemenetek generáljanak megszakítást



Mintapélda – GPIO, EXTI (4)

Megszakítás vezérlő (NVIC):

- 87 (perifériák) + 16 (mag) megszakítás vonal
- 4 biten beállítható prioritási szintek

PRIGROUP (3 bits)	Binary Point (group.sub)		Preemting Priority (Group Priority)		Sub-Priority	
			Bits	Levels	Bits	Levels
011 (NVIC_PriorityGroup_4)	4.0	gggg	4	16	0	0
100 (NVIC_PriorityGroup_3)	3.1	gggs	3	8	1	2
101 (NVIC_PriorityGroup_2)	2.2	ggss	2	4	2	4
110 (NVIC_PriorityGroup_1)	1.3	gsss	1	2	3	8
111 (NVIC_PriorityGroup_0)	0.4	ssss	0	0	4	16

Magasabb prioritási csoportba tartozó megszakítás megszakíthatja a kisebbe tartozót, az al-prioritás az egyszerre bekövetkező azonos csoportú megszakítások végrehajtási sorrendjét szabja meg. (Mindig a kisebb szám jelenti a nagyobb prioritást)

Mintapélda – GPIO, EXTI (5)

Külső megszakítás kezelése:

- A GPIO mód beállításnál ki kell választani, hogy bemenet legyen és megszakítás generálódjon
  - A nyomógomb az C port 13-s lábán van
  - Lefutó élre generáljon megszakítást
- Be kell állítani a megszakítás prioritását
- Engedélyezzük a megszakítást
- Lekezeljük a megszakítást

Mintapélda – GPIO, EXTI (6)

GPIO beállítások:

```
__GPIOA_CLK_ENABLE();

portInit.Mode = GPIO_MODE_IT_FALLING;
portInit.Speed = GPIO_SPEED_LOW;
portInit.Pull = GPIO_NOPULL;
portInit.Pin = GPIO_PIN_0;

HAL_GPIO_Init(BUTTON_PORT, &portInit);
```

Megszakítás prioritás beállítás, engedélyezés:

```
HAL_NVIC_SetPriority(EXTI0_IRQn,0,0);

HAL_NVIC_EnableIRQ(EXTI0_IRQn);
```

Megszakítás kezelése:

```
void EXTI0_IRQHandler(void)
{
   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
}
```

Mintapélda – GPIO, EXTI (7)

Írjuk meg az alkalmazást!

Mintapélda – GPIO, Timer, EXTI

Fejlesszük tovább a projektünket:

Villogó LED hardveres időzítéssel (GP Timer)
gombnyomásra megáll, következőre újraindul

Mintapélda – GPIO, Timer, EXTI (2)

Mit kell tennünk:

1. Timer beállítások
2. Timer megszakítás kezelése

Mintapélda – GPIO, Timer, EXTI (3)

Timer beállítások (Tim4, felfelé számláló, 500ms IT):

```
/* Timer (TIM4) periféria órajel engedélyezés */
__TIM4_CLK_ENABLE();
/* Timer konfigurálása */
Tim4Handle.Instance = TIM4;
Tim4Handle.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
Tim4Handle.Init.CounterMode = TIM_COUNTERMODE_UP;
Tim4Handle.Init.Prescaler =54999;
Tim4Handle.Init.Period = 999;

HAL_TIM_Base_Init(&Tim4Handle);
```

Miért pont így kell kitölteni a struktúrát?

Mintapélda – GPIO, Timer, EXTI (4)

Prescaler (16 bit):

Beállítja, hogy a buszfrekvenciához képest milyen frekvenciával számoljon a számlálónk

Prescaler = (APBx_clk / TIM_counter_clk) -1

Period (16 bit):

A számláló végértékét állítja be

Period = (TIM_counter_clk / f) -1

Azt szeretnénk, hogy 500msec gyakorisággal váltsanak állapotot a LED-ek!

Mintapélda – GPIO, Timer, EXTI (5)

Prescaler = (APBx_clk / TIM_counter_clk) - 1

Period = (TIM_counter_clk / f) - 1

APBx_clk = 110 MHz

f = 2Hz

Olyan értékeket kell választanunk, amik beleférnek a 16 bites regiszterekbe (0..65535)

A számlálónk számoljon mondjuk a buszfrekvencia ezredével, tehát 110 kHz-el

Prescaler = (110000000/110000) – 1 = 1000 -1 = 999

Az előosztót kiszámoltuk, számoljuk ki a periódust

Period = (110000 / 2) – 1 = 55000 – 1 = 54999

Mintapélda – GPIO, Timer, EXTI (6)

Timer megszakítás engedélyezése:

```
HAL_NVIC_SetPriority(TIM4_IRQn, 0, 1);

HAL_NVIC_EnableIRQ(TIM4_IRQn);

HAL_TIM_Base_Start_IT(&Tim4Handle);
```

Timer megszakítás kezelése:

```
void TIM4_IRQHandler(void)
{
    HAL_TIM_IRQHandler(&Tim4Handle);
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
        if ( htim->Instance == TIM4 )
                //…
}
```

Mintapélda – GPIO, Timer, EXTI (7)

Írjuk meg az alkalmazást!

A FreeRTOS operációs rendszer

Tulajdonságok:

- 32 bites architektúrákra optimalizált
- Mikrokontroller kategóriájú (pl. Cortex M család) ARM processzorok egyszerű operációs rendszere
- Alapértelmezés szerint teljesen ingyenes és nyílt forrású (GPL)

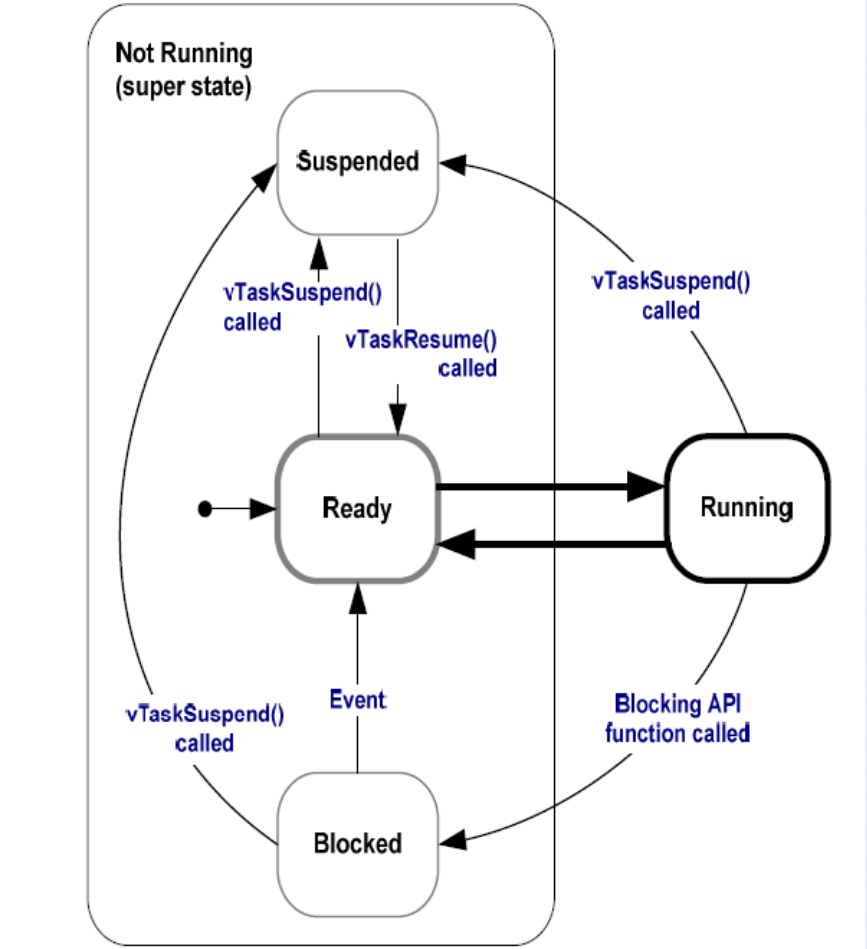
A FreeRTOS operációs rendszer (2)

- **Ütemezője lehet**
  - Kooperatív (nem-preemptív)
  - Preemptív
- **Rugalmas taszk prioritás kezelés**
  - Azonos prioritású taszkok között Round-Robin ütemezés
- **Várakozási sorok támogatása**
  - Minden egyéb taszk szinkronizációs eszköz erre épül
- **Szemaforok támogatása**
  - Bináris
  - Számláló jellegű
  - Rekurzív
  - Mutex
- **Esemény csoportok támogatása**

A FreeRTOS operációs rendszer (3)

- **Stack túlcsordulás figyelés**
- **Trace támogatás**
- **MPU támogatás**
- **Hook függvények**
  - Idle
  - Tick
- **CoRoutine támogatás – lightweight task**
- **Timer szolgáltatások – szoftveres timer**
- **Dinamikus memória kezelés**
  - Több verzióban elérhető

A FreeRTOS taszk állapotai



Taszkok kezelése

Taszk létrehozása:

```
BaseType_t xTaskCreate( TaskFunction_t pxTaskCode,  const char * const pcName,
        const uint16_t usStackDepth, void * const pvParameters,
        UBaseType_t uxPriority, TaskHandle_t * const pxCreatedTask);
```

Taszk függvény prototípusa:

```
void TaskFunction(void const * argument);
```

Taszk törlése:

```
void vTaskDelete( TaskHandle_t xTaskToDelete );
```

Taszk prioritásának módosítása:

```
void vTaskPrioritySet( TaskHandle_t xTask, UBaseType_t uxNewPriority );
```

Taszk felfüggesztése és folytatása:

```
void vTaskSuspend( TaskHandle_t xTaskToSuspend );
void vTaskResume( TaskHandle_t xTaskToResume );
```

Ütemezés

Az ütemezési algoritmus futási ideje kritikus!

- Gyakran fut, ezért nagy overheadet jelentene, ha hosszú ideig futna
- Nem célszerű benne hosszú ciklusokat csinálni!
- Legjobb lenne, ha csak pár utasítás lenne!
- Kulcsfontosságú a nyilvántartás módja, hiszen okos nyilvántartással csökkenthetjük az utasítások számát

- Nincs fifikás nyilvántartás, egyszerű rendezett láncolt listában tartja nyilván a futásra kész taszkokat

Megszakítások

Az operációs rendszernek mindig tudnia kell róla, ha egy IT rutinba belépünk, vagy kilépünk onnan!

- **Speciális függvényeket használunk az értesítéshez**
- **Megszakítás kezelő függvényből csak olyan OS függvényeket szabad meghívni, amelyek neve FromISR-re végződik pl**. xQueueReceiveFromISR

Operációs rendszer indítása

Operációs rendszer elindítása:

```
void vTaskStartScheduler( void );
```

Indítási folyamat:

```
int  main (void)
{
   xTaskCreate( … ); // Legalább egy taszk létrehozása
   vTaskStartScheduler(); // operációs rendszer indítása
}
```

Várakozási sorok

Létrehozás:

```
QueueHandle_t xQueueCreate(UBaseType_t uxQueueLength, UBaseType_t uxItemSize);
```

Törlés:

```
vQueueDelete( QueueHandle_t xQueue );
```

Elem kivétele a sorból:

```
BaseType_t xQueueReceive(QueueHandle_t xQueue, void *pvBuffer, TickType_t
   xTicksToWait);
```

Olvasás a sorból:

```
BaseType_t xQueuePeek(QueueHandle_t xQueue, void *pvBuffer, TickType_t
   xTicksToWait );
```

Elem betétele a sorba:

```
BaseType_t xQueueSendToToFront(QueueHandle_t xQueue, const void
   *pvItemToQueue, TickType_t xTicksToWait );
BaseType_t xQueueSendToToBack(QueueHandle_t xQueue, const void *pvItemToQueue,
   TickType_t xTicksToWait );
BaseType_t xQueueSendToFrontFromISR(QueueHandle_t xQueue,
const void *pvItemToQueue, BaseType_t *pxHigherPriorityTaskWoken );
```

Egyebek

Taszkváltás kényszerítése:

```
void taskYield( void );
```

Kritikus szakasz:

```
taskENTER_CRITICAL();
taskEXIT_CRITICAL();
```

Megszakítások engedélyezése és tiltása:

```
taskENABLE_INTERRUPTS();
taskDISABLE_INTERRUPTS();
```

Mintapélda – FreeRTOS

Fejlesszük tovább a projektünket:

- Használjunk operációs rendszert!
- LED villogtatás – Taszk
- Gombnyomás jelzése – Jelzőflag

Mintapélda – FreeRTOS

Írjuk meg az alkalmazást!