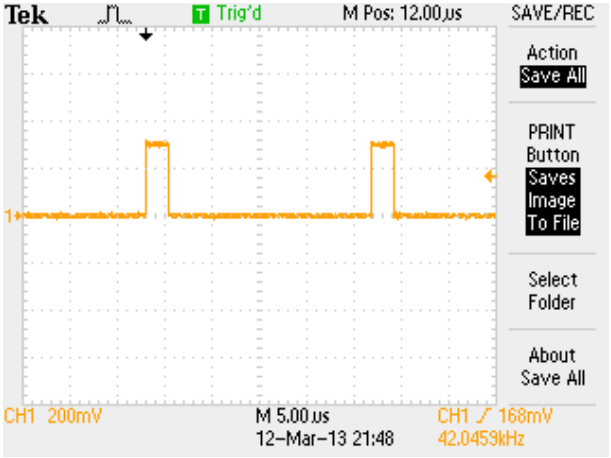
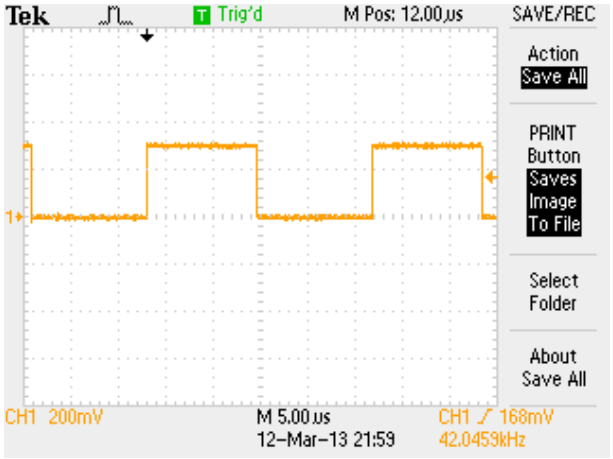
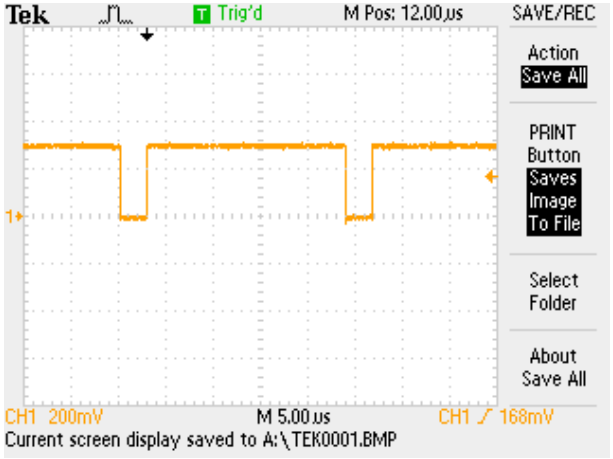
Beágyazott Rendszerek STM32F4 Discovery

PWM, EXTI

PWM
(Pulse-width modulation)





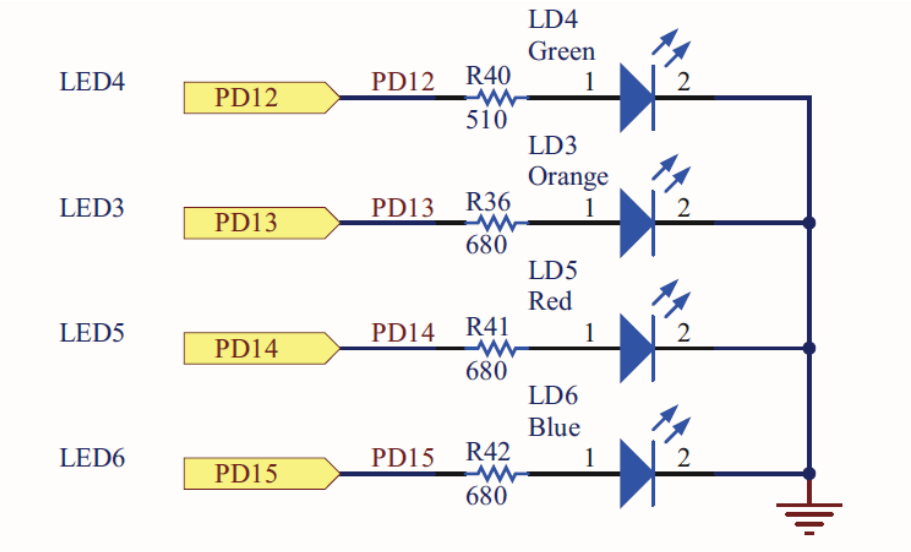


Feladat 1.:

- STM32F4Discovery panel 4 ledjének PWM fényerő szabályozása
  - TIM4 timer, mint PWM kimenet
  - Gomb kezelés, külső megszakításként (EXTI)

LED-ek

- LD3, LD4, LD5, LD6
- A processzor melyik lábára vannak kötve?



LEDs

UM1472 User Manual

Table 5. MCU pin description versus board function (page 8 of 10)

MCU pin			Board function														
Main function	Alternate functions	LQFP100	CS43L22	MP45DT02	LIS302DL	Pushbutton	LED	SWD	USB	OSC	Free I/O	Power supply	CN5	CN2	P1	P2
PD1	FSMC_D3/ CAN1_TX	82														33
PD2	TIM3_ETR/ UART5_RXSDIO_CMD / DCMI_D11	83														34
PD3	FSMC_CLK/ USART2_CTS	84														31
PD4	FSMC_NOE/ USART2_RTS	85	RESET													32
PD5	FSMC_NWE/ USART2_TX	86					RED		OverCurrent							29
PD6	FSMC_NWAIT/ USART2_RX	87														30
PD7	USART2_CK/ FSMC_NE1/ FSMC_NCE2	88														27
PD8	FSMC_D13/ USART3_TX	55													40	
PD9	FSMC_D14/ USART3_RX	56													41	
PD10	FSMC_D15/ USART3_CK	57													42	
PD11	FSMC_A16/ USART3_CTS	58													43	
PD12	FSMC_A17/ TIM4_CH1/ USART3_RTS	59					GREEN								44	
PD13	FSMC_A18/ TIM4_CH2	60					ORANGE								45	
PD14	FSMC_D0/ TIM4_CH3	61					RED								46	
PD15	FSMC_D1/ TIM4_CH4	62					BLUE								47	

File – New C (embedded) project

main.c:

```
/* Private variables ---------------------------------*/
uint32_t _TIM_Period;
/* Private function prototypes -----------------------*/
void TIM4_Config(void);
void TIM4_Init(void);
```

TIM4_Config 1/2

main() alá

```
void TIM4_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* TIM3 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

  /* GPIOD clock enable */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

  /* GPIOC Configuration: TIM4 CH1 ...CH4 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13
   | GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
```

TIM4_Config 2/2

main() alá

```
  /* Connect TIM4 pins to AF2 */
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource13, GPIO_AF_TIM4);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_TIM4);
  GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4);
}
```

```
/* ----------------------------------------------------------------------
TIM3 Configuration: generate 4 PWM signals with 4 different duty cycles.

In this example TIM3 input clock (TIM3CLK) is set to 2 * APB1 clock (PCLK1),
since APB1 prescaler is different from 1.
  TIM3CLK = 2 * PCLK1
  PCLK1 = HCLK / 4
  => TIM3CLK = HCLK / 2 = SystemCoreClock /2

To get TIM3 counter clock at 28 MHz, the prescaler is computed as follows:
   Prescaler = (TIM3CLK / TIM3 counter clock) - 1
   Prescaler = ((SystemCoreClock /2) /28 MHz) - 1

To get TIM3 output clock at 30 KHz, the period (ARR)) is computed as
    follows:
   ARR = (TIM3 counter clock / TIM3 output clock) - 1
   = 665


Note:
 SystemCoreClock variable holds HCLK frequency and is defined in
    system_stm32f4xx.c file.
 Each time the core clock (HCLK) changes, user had to call
    SystemCoreClockUpdate()
 function to update SystemCoreClock variable value. Otherwise, any
    configuration
 based on this variable will be incorrect.
  ---------------------------------------------------------------------- */
```

TIM4_Init 1/4

main() alá

```
void TIM4_Init(void)
{
   TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
   TIM_OCInitTypeDef  TIM_OCInitStructure;
   uint16_t PrescalerValue = 0;

   /* Compute the prescaler value */
   PrescalerValue = (uint16_t) ((SystemCoreClock /2) /
       28000000) - 1; //2
   _TIM_Period = 279; // (28 000 000 / 100 000)-1
   uint32_t _TIM_Pulse = _TIM_Period/5; // 20% duty cycle
```

TIM4_Init 2/4

main() alá

```
/* Time base configuration */
TIM_TimeBaseStructure.TIM_Period = _TIM_Period;
TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode =
   TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);

/* PWM1 Mode configuration parameters*/
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState =
   TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = _TIM_Pulse;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
```

TIM4_Init 3/4

main() alá

```
/* PWM1 Mode configuration: Channel1 */
TIM_OC1Init(TIM4, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel2 */
TIM_OC2Init(TIM4, &TIM_OCInitStructure);
TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel3 */
TIM_OC3Init(TIM4, &TIM_OCInitStructure);
TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);

/* PWM1 Mode configuration: Channel4 */
TIM_OC4Init(TIM4, &TIM_OCInitStructure);
TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);
```

TIM4_Init 4/4

main() alá

```
TIM_ARRPreloadConfig(TIM4, ENABLE);

/* TIM3 enable counter */
TIM_Cmd(TIM4, ENABLE);
}



   /* Ezt csak egyszer kell megírni, utána sablonként
   használható… */
```

```
int main(void)
{

  TIM4_Config();
  TIM4_Init();
  while (1);
}
```

Fordít, ellenőriz, javít ☺

PWM érték frissítése

```
void TIM_OC1Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef*
   TIM_OCInitStruct)
{
   .
   .
   .
  /* Set the Capture Compare Register value */
  TIMx->CCR1 = TIM_OCInitStruct->TIM_Pulse;
   .
   .
   .
}
```

```c
int main(void)
{

   TIM4_Config();
   TIM4_Init();

   TIM4->CCR1 = _TIM_Period / 8;  //12,5%
   TIM4->CCR2 = _TIM_Period / 4; //25%
   TIM4->CCR3 = _TIM_Period / 2; //50%
   TIM4->CCR4 = _TIM_Period; //100%

   while (1);
}
```

Fordít, ellenőriz, javít ☺

```c
int main(void)
{

   TIM4_Config();
   TIM4_Init();

   STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_EXTI);

   TIM4->CCR1=0;
   TIM4->CCR2=0;
   TIM4->CCR3=0;
   TIM4->CCR4=0;
   while (1);
}
```

STM_EVAL_PBInit

!!NEM KELL ÍRNI!!

```
void STM_EVAL_PBInit(Button_TypeDef Button, ButtonMode_TypeDef Button_Mode)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Enable the BUTTON Clock */
  RCC_AHB1PeriphClockCmd(BUTTON_CLK[Button], ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

  /* Configure Button pin as input */
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Pin = BUTTON_PIN[Button];
  GPIO_Init(BUTTON_PORT[Button], &GPIO_InitStructure);

  if (Button_Mode == BUTTON_MODE_EXTI)
  {
    /* Connect Button EXTI Line to Button GPIO Pin */
    SYSCFG_EXTILineConfig(BUTTON_PORT_SOURCE[Button], BUTTON_PIN_SOURCE[Button]);

    /* Configure Button EXTI line */
    EXTI_InitStructure.EXTI_Line = BUTTON_EXTI_LIN[Button];
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* Enable and set Button EXTI Interrupt to the lowest priority */
    NVIC_InitStructure.NVIC_IRQChannel = BUTTON_IRQn[Button];
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

    NVIC_Init(&NVIC_InitStructure);
  }
}
```

stm32f4xx_it.c

```
/* Includes alá */
extern uint32_t  _TIM_Period; //☹


/*    Cortex-M4 Processor Exceptions Handlers   */
void EXTI0_IRQHandler(void)
{
  if(EXTI_GetITStatus(EXTI_Line0) != RESET)
  {
   TIM4->CCR1 = ( TIM4->CCR1 + _TIM_Period/8 ) % _TIM_Period;
   TIM4->CCR2 = ( TIM4->CCR2 + _TIM_Period/8 ) % _TIM_Period;
   TIM4->CCR3 = ( TIM4->CCR3 + _TIM_Period/8 ) % _TIM_Period;
   TIM4->CCR4 = ( TIM4->CCR4 + _TIM_Period/8 ) % _TIM_Period;
  /* Clear the EXTI line 0 pending bit */
  EXTI_ClearITPendingBit(EXTI_Line0);
  }
}
```

stm32f4xx_it.h

```
void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler(void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);
void SysTick_Handler(void);

void EXTI0_IRQHandler(void);
```

Fordít, ellenőriz, javít ☺

Feladat 2.:

- Automatikusan világosodó, majd elsötétülő világítás (0%..100%..0%..100%..stb)
  - SYSTICK időzítő

Feladat 3.:
(opcionális)

- Szervó teszt program, 1..2ms jelet állít elő, 100hz frekvenciával
- Oda-vissza mozgatja a szervót
- TIM base config: ☺
  - `PrescalerValue = 12;`
  - `uint32_t _TIM_Period = 65535;`
  - `uint32_t _TIM_Pulse = 6500;`

```
//6500 ...13000 = 1...2ms
```