BEÁGYAZOTT RENDSZEREK (C)

1. forduló

BOSCH
Invented for life

A kategória támogatója: Robert Bosch Kft.

Ismertető a feladathoz

Fontos információk:

A kérdésekre **mindig van helyes válasz**! Ha csak egy helyes válasz van az adott kérdésre, radio button-os választási lehetőségeket fogsz látni.

Kérjük, hogy a feladatok szövegeit **ne másold** és a böngésződ fejlesztő eszközét/ konzolját se nyisd meg feladatmegoldás közben! Mindkettő kizárást vonhat maga után.

Minden forduló után a **megoldások csütörtök reggel 8 órakor** lesznek elérhetőek.

A megoldásokkal kapcsolatos esetleges **észrevételeket a megoldások megjelenését követő kedd éjfélig** várjuk.

A több válaszlehetőségű feleletválasztós kérdéseknél járnak **részpontszámok, ha egyik rossz választ sem jelölöd be**.

Ha kifutsz az adott feladatlap kitöltésére rendelkezésre álló időből, a felület **automatikusan megpróbálja beküldeni** az addig megadott válaszokat.

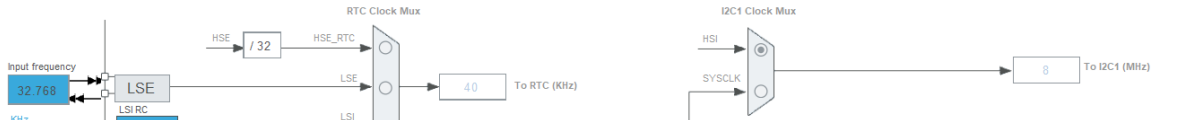
Minden feladatsornak van egy **becsült minimum megoldási ideje** (legalább a feladat elolvasási ideje). Aki ennél rövidebb idő alatt oldja meg, kizárható a versenyből.

Az első kategória után, amelynek a feladatlapját megoldod a fordulóban, kapni fogsz egy 2-3 perc alatt kitölthető **kérdőívet**. Az ezekből összeállított piackutatás legfontosabb eredményeit a díjátadót követően Veled is megosztjuk majd. Formáljuk közösen a piacot!

Első forduló

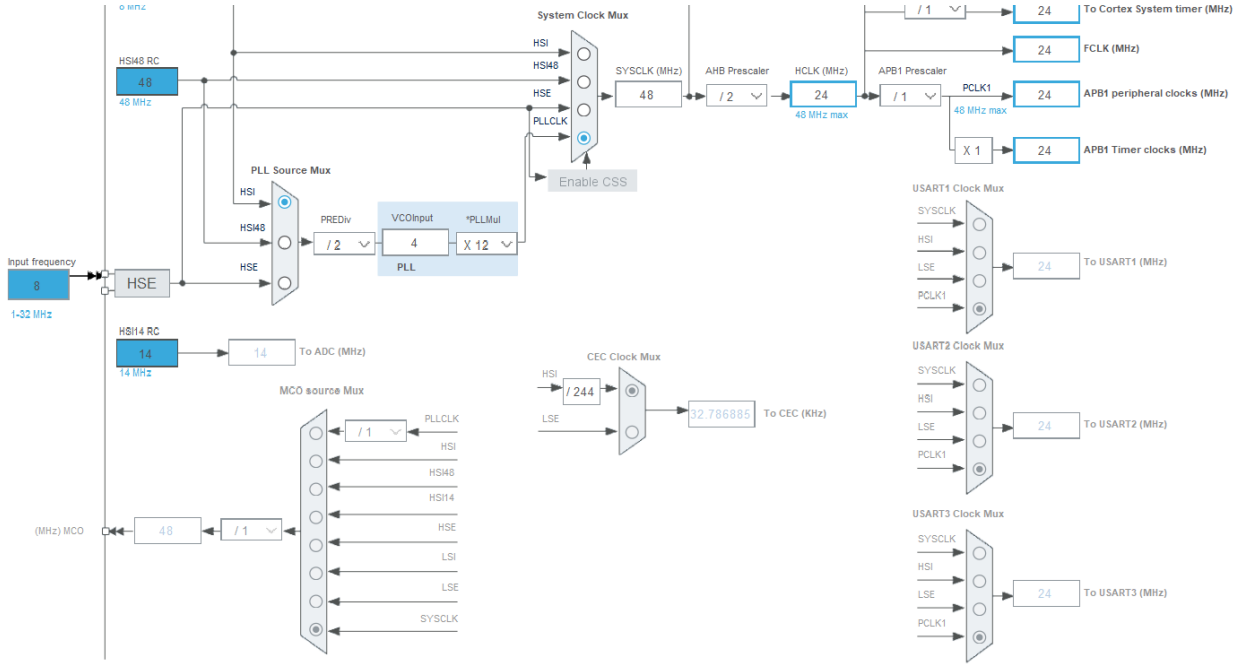
Ájóti János folytatja tavalyi kalandjait az okosotthonok világában. A múlt évi C-s tudását kamatoztatja idén, STM32 fejlesztőkészletekkel. Első otthoni projektje során egy STM32F091 Nucleo-64 Boardot használ. Első feladatként szeretne egy folytonosan villogó visszajelző fényt készíteni. A LED-et 1 másodpercenként szeretné villogtatni (T=2s, PWM=50%).

A beállított clock tree-t a következő ábra szemlélteti.



RENDELKEZÉSRE ÁLLÓ IDŐ:

40:00



A megoldásban a következő adatlapok lesznek segítségetekre:

STM32F091RC adatlap:

https://www.st.com/resource/en/datasheet/stm32f091rc.pdf

NUCLEO-F091RC adatlap:

https://www.st.com/resource/en/data_brief/nucleo-f091rc.pdf

HAL API dokumentáció:

https://www.st.com/resource/en/user_manual/dm00122015-description-of-stm32f0-hal-and-lowlayer-drivers-stmicroelectronics.pdf

Felhasznált idő: 00:56/40:00 **Elért pontszám:** 0/45

1. feladat 0/5 pont

Első ötlete, hogy rátölt a boardra egy CMSIS_V2 API-val rendelkező FreeRTOS-t. Ezekután létrehoz egy taskot *toggleTask* néven, mely bizonyos időközönként lefutva villogtatja a LED-et. A SysTick-et 1ms-ra konfigurálja. A task belépési pontja a *toggle* függvény. A STM32F091 Nucleo-64 boardot tekintve melyik kódrészlet valósítja meg helyesen a leírt viselkedést? Fontos Ájóti János számára, hogy a feladat megoldása ne akadályozza a jövőbeni feature-ök megvalósítását, és azok minél kevésbé befolyásolják a LED viselkedését.

Válasz

1.válasz:

```
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Definitions for toggleTask */
osThreadId_t toggleTaskHandle;
const osThreadAttr_t toggleTask_attributes = {
  .name = "toggleTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityRealtime,
};

/* Private function prototypes */
void toggle(void*);

int main(void)
{
  /* ... */

  /* creation of toggleTask */
  toggleTaskHandle = osThreadNew(toggle, NULL, &toggleTask_attributes);

  /* ... */
}

void toggle(void *argument)
{
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    osDelay(2400);
  }
}
```

2.válasz:

```
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Definitions for toggleTask */
osThreadId_t toggleTaskHandle;
const osThreadAttr_t toggleTask_attributes = {
  .name = "toggleTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityRealtime,
};

/* Private function prototypes */
void toggle(void*);

int main(void)
{
  /* ... */

  /* creation of toggleTask */
  toggleTaskHandle = osThreadNew(toggle, NULL, &toggleTask_attributes);

  /* ... */
}

void toggle(void *argument)
{
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    osDelay(1000);
  }
}
```

3.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Definitions for toggleTask */
osThreadId_t toggleTaskHandle;
const osThreadAttr_t toggleTask_attributes = {
  .name = "toggleTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityLow,
};

/* Private function prototypes */
void toggle(void*);

int main(void)
{
  /* ... */

  /* creation of toggleTask */
  toggleTaskHandle = osThreadNew(toggle, NULL, &toggleTask_attributes);

  /* ... */
}

void toggle(void *argument)
{
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    osDelay(1000);
  }
}
```

4.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Definitions for toggleTask */
osThreadId_t toggleTaskHandle;
const osThreadAttr_t toggleTask_attributes = {
  .name = "toggleTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityRealtime,
};

/* Private function prototypes */
void toggle(void*);

int main(void)
{
  /* ... */

  /* creation of toggleTask */
  toggleTaskHandle = osThreadNew(toggle, NULL, &toggleTask_attributes);

  /* ... */
}

void toggle(void *argument)
{
  for(;;)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    HAL_Delay(1000);
  }
}
```

Magyarázat

Az osDelay paramétere határozza meg, hogy hány SysTick után futhat tovább a task. Mivel Ájóti János 1ms-ra konfigurálta a SysTick-et, így 1000-et kell megadni, mint paraméter. (A LED kapcsolási idejétől eltekintünk.)

A *toggleTask* prioritása fontos, hogy ne legyen túl alacsony, hisz akkor a jövőbeni feature-ök kiéheztethetik a folyamatot, befolyásolva a LED égésének idejét.

Az osDelay függvényt használva visszakerül a vezérlés az OS-hez, ezáltal tud más folyamatokat ütemezni, amíg a *toggleTask* vár. A *HAL_Delay* hívás ezzel ellentétben blokkolja a futást, mivel az OS nem értesül a várakozásról. Ezáltal minden task, ami kisebb prioritású, mint a *toggleTask*, kiéhezik.

2. feladat 0/10 pont

A sikeres első próbálkozás után szeretné még pontosabbá tenni a villogás frekvenciáját. Eszébe jut, hogy használhatna hardveres megszakítást is a LED állapotának változtatására. Az adatlap alapján kinézi, hogy a 7-es timer egyszerűsége elég a feladat megoldásához. Melyik konfiguráció oldja meg a helyesen az előbbi feladatot?

Válasz

1.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htim7;

/* Private function prototypes */
static void MX_TIM7_Init(void);

int main(void)
{
  /* ... */

  MX_TIM7_Init();
  HAL_TIM_Base_Start(&htim7);

  /* ... */
}

static void MX_TIM7_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim7.Instance               = TIM7;
  htim7.Init.Prescaler         = 2400 - 1;
  htim7.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htim7.Init.Period            = 10000 - 1;
  htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM7){
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
  }
}
```

2.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htim7;

/* Private function prototypes */
static void MX_TIM7_Init(void);

int main(void)
{
  /* ... */

  MX_TIM7_Init();
  HAL_TIM_Base_Start_IT(&htim7);

  /* ... */
}

static void MX_TIM7_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim7.Instance               = TIM7;
  htim7.Init.Prescaler         = 2400;
  htim7.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htim7.Init.Period            = 10000;
  htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
}
```

3.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htim7;

/* Private function prototypes */
static void MX_TIM7_Init(void);

int main(void)
{
  /* ... */

  MX_TIM7_Init();
  HAL_TIM_Base_Start_IT(&htim7);

  /* ... */
}

static void MX_TIM7_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim7.Instance               = TIM7;
  htim7.Init.Prescaler         = 2400 - 1;
  htim7.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htim7.Init.Period            = 10000 - 1;
  htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM7){
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
  }
}
```

4.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htim7;

/* Private function prototypes */
static void MX_TIM7_Init(void);

int main(void)
{
  /* ... */

  MX_TIM7_Init();
  HAL_TIM_Base_Start_IT(&htim7);

  /* ... */
}

static void MX_TIM7_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim7.Instance               = TIM7;
  htim7.Init.Prescaler         = 2400;
  htim7.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htim7.Init.Period            = 10000;
  htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM7){
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
  }
}
```

Magyarázat

Ahhoz, hogy a timer generáljon egy megszakítást, a *HAL_TIM_Base_Start_IT* függvénnyel kell elindítani a TIM7-et.

Mivel a számlálás 0-tól kezdődik, így 1-gyel kisebb értéket kell megadni mind a Prescaler, mind a Period értékénél.

Mivel minden timer a *HAL_TIM_PeriodElapsedCallback* függvényt hívja meg, így fontos, hogy ellenőrizzük, hogy az általunk megadott timer triggerelt-e.

3. feladat 0/1 pont

A sikeres második próbálkozás után szeretné még egyszerűbbé tenni a program logikáját. Eszébe jut, hogy használhatna PWM-et is a LED állapotának változtatására. Az adatlap alapján melyik timert tudná erre a célra használni? Add meg a timer sorszámát! (pl.: TIM17 esetén 17)

Válaszok

A helyes válasz:
2

Magyarázat

Az adatlap alapján a PA5-ös pinre a TIM2 első csatornája (TIM2_CH1) van kivezetve.

4. feladat 0/14 pont

Miután meghatározta a helyes timert, nekikezd a megvalósításnak. A megadott válaszokban szereplő timx/TIMx a 3. feladatban megtalált timert jelöli. Az alábbi kódrészletek közül melyikkel tudja megvalósítani a kívánt működést?

Válaszok

☑ **1.válasz:**

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htimx;

/* Private function prototypes */
static void MX_TIMx_Init(void);

int main(void)
{
  /* ... */
  MX_TIMx_Init();
  HAL_TIM_PWM_Start(&htimx);
  /* ... */
}

static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 24 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 2000000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV1;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_PWM1;
  sConfigOC.Pulse      = 1000000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

2.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htimx;

/* Private function prototypes */
static void MX_TIMx_Init(void);

int main(void)
{
  /* ... */
  MX_TIMx_Init();
  HAL_TIM_PWM_Start(&htimx);
  /* ... */
}

static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 24000 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 2000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV1;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_PWM1;
  sConfigOC.Pulse      = 1000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

3.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htimx;

/* Private function prototypes */
static void MX_TIMx_Init(void);

int main(void)
{
  /* ... */
  MX_TIMx_Init();
  HAL_TIM_PWM_Start(&htimx);
  /* ... */
}

static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 24000 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 2000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV1;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_PWM1;
  sConfigOC.Pulse      = 1000 - 1;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

4.válasz:

```c
static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 24 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 2000000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV1;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_PWM1;
  sConfigOC.Pulse      = 1000000 - 1;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

5.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htimx;

/* Private function prototypes */
static void MX_TIMx_Init(void);

int main(void)
{
  /* ... */
  MX_TIMx_Init();
  HAL_TIM_PWM_Start(&htimx);
  /* ... */
}

static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 6000 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 8000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV4;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_PWM1;
  sConfigOC.Pulse      = 4000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

Magyarázat

Mivel a Pulse elérésekor és a Period átlépésekor változik az output értéke, így a Period 1-gyel kisebb, mint a szándékolt érték, viszont az Pulse meg kell hogy egyezzen a kívánt értékkel.

24000-es Prescale mellett a számláló 24MHz/24000=1kHz-es frekvenciával fog nőni. Ahhoz, hogy 2 másodperc legyen a periódusidő, 2000 - 1-et kell beállítani. Hisz 0-tól 1999-ig pontosan 2000-szer nő a counter értéke. Viszont, a Pulse értékét 1000-re kell állítani, hisz 0-tól 999 -ig történik 1000 számlálás, majd 1000-től 1999-ig szintúgy.

5. feladat 0/15 pont

A sikeres harmadik próbálkozás után szeretné még egyféleképpen megoldani a problémát. Eszébe jut, hogy használhatná a timerek output compare funkcionalitását is a LED állapotának változtatására. A megadott válaszokban szereplő timx/TIMx a 3. feladatban megtalált timert jelöli. Melyik konfiguráció oldja meg helyesen az előbbi feladatot?

Válaszok

- [ ] **1.válasz:**

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htimx;

/* Private function prototypes */
static void MX_TIMx_Init(void);

int main(void)
{
  /* ... */
  MX_TIMx_Init();
  HAL_TIM_OC_Start(&htimx);
  /* ... */
}

static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 2400 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 10000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV1;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_OC_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_ACTIVE;
  sConfigOC.Pulse      = 5000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_OC_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

- [x] **2.válasz:**

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htimx;

/* Private function prototypes */
static void MX_TIMx_Init(void);

int main(void)
{
  /* ... */
  MX_TIMx_Init();
  HAL_TIM_OC_Start(&htimx);
  /* ... */
}

static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 2400 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 10000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV1;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_OC_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_TOGGLE;
  sConfigOC.Pulse      = 5000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_OC_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

3.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htimx;

/* Private function prototypes */
static void MX_TIMx_Init(void);

int main(void)
{
  /* ... */
  MX_TIMx_Init();
  HAL_TIM_OC_Start(&htimx);
  /* ... */
}

static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 2400 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 10000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV1;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_OC_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_ACTIVE;
  sConfigOC.Pulse      = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_OC_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

4.válasz:

```c
#include "stm32f0xx_hal.h"
#include "cmsis_os.h"

/* Private variables */
TIM_HandleTypeDef htimx;

/* Private function prototypes */
static void MX_TIMx_Init(void);

int main(void)
{
  /* ... */
  MX_TIMx_Init();
  HAL_TIM_OC_Start(&htimx);
  /* ... */
}

static void MX_TIMx_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig     = {0};
  TIM_OC_InitTypeDef sConfigOC              = {0};
  htimx.Instance               = TIMx;
  htimx.Init.Prescaler         = 2400 - 1;
  htimx.Init.CounterMode       = TIM_COUNTERMODE_UP;
  htimx.Init.Period            = 10000 - 1;
  htimx.Init.ClockDivision     = TIM_CLOCKDIVISION_DIV1;
  htimx.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htimx, &sClockSourceConfig) != HAL_OK){
    Error_Handler();
  }
  if (HAL_TIM_OC_Init(&htimx) != HAL_OK){
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode     = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htimx, &sMasterConfig) != HAL_OK){
    Error_Handler();
  }
  sConfigOC.OCMode     = TIM_OCMODE_TOGGLE;
  sConfigOC.Pulse      = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_OC_ConfigChannel(&htimx, &sConfigOC, TIM_CHANNEL_1) != HAL_OK){
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htimx);
}
```

Magyarázat

Az output compare funkcionalitás jelenlegi felhasználása esetén nem számít a Pulse értéke. Ami fontos, hogy az OCMode Toggle-re legyen konfigurálva.

Megjelenés

Világos