BEÁGYAZOTT RENDSZEREK (C)

2. forduló

BOSCH
Invented for life

A kategória támogatója: Robert Bosch Kft.

Ismertető a feladathoz

Fontos információk

> Ha kifutsz az adott feladatlap kitöltésére rendelkezésre álló időből, a felület **automatikusan megpróbálja beküldeni** az addig megadott válaszokat
>
> A kérdésekre **mindig van helyes válasz**! Ha csak egy helyes válasz van az adott kérdésre, radio button-os választási lehetőségeket fogsz látni.
>
> **Olyan kérdés viszont nincs, amelyre az összes válasz helyes!**

Egyéb információkat a versenyszabályzatban találsz!

Második forduló

Ájóti János ezen a héten egy hőmérsékletszabályzót szeretne létrehozni, melyhez először egy potméter segítségével kívánja lekezelni a beállítandó értéket. A potmétert GND és 3,3V közé köti, majd a leosztott feszültséget a Nucleo-F091RC ADC perifériájának A0-s csatornájára vezeti.

Az ADC perifériát a következőképpen konfigurálta fel:

RENDELKEZÉSRE ÁLLÓ IDŐ:

60:00

```c
ADC_HandleTypeDef hadc;
static void MX_ADC_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};
  hadc.Instance = ADC1;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  if (HAL_ADC_Init(&hadc) != HAL_OK){
    Error_Handler();
  }
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Segíts Ájóti Jánosnak az analóg adatok beolvasásában!

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

Kontroller reference manual:

https://www.st.com/resource/en/reference_manual/rm0091-stm32f0x1stm32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Felhasznált idő: 00:55/60:00 **Elért pontszám:** 0/25

1. feladat 0/5 pont

Első próbálkozásként létrehoz egy taskot *adcConv* belépési függvénnyel. A konvertált adatokat UART-ra szeretné kiküldeni. Melyik kódrészlet valósítja meg helyesen az elvárt működést?

Válaszok

- [x]
```
void adcConv(void *argument)
{
  uint16_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start(&hadc);
    HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
    raw = HAL_ADC_GetValue(&hadc);

    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

    osDelay(1000);
  }
}
```

- [ ]
```
void adcConv(void *argument)
{
  uint8_t raw;
  char msg[12];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start(&hadc);
    HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
    raw = HAL_ADC_GetValue(&hadc);

    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

    osDelay(1000);
  }
}
```

- [ ]
```
void adcConv(void *argument)
{
  uint32_t raw;
  char msg[12];

  HAL_ADCEx_Calibration_Start(&hadc);
  HAL_ADC_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
    raw = HAL_ADC_GetValue(&hadc);

    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

    osDelay(1000);
  }
}
```

- [ ]

```
void adcConv(void *argument)
{
  uint16_t raw;
  char msg[12];

  HAL_ADCEx_Calibration_Start(&hadc);
  HAL_ADC_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
    raw = HAL_ADC_GetValue(&hadc);

    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

    osDelay(1000);
  }
}
```

- [x]
```
void adcConv(void *argument)
{
  uint32_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start(&hadc);
    HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
    raw = HAL_ADC_GetValue(&hadc);

    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

    osDelay(1000);
  }
}
```

- [ ]
```
void adcConv(void *argument)
{
  uint8_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  HAL_ADC_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
    raw = HAL_ADC_GetValue(&hadc);

    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);

    osDelay(1000);
  }
}
```

- [ ] Egyik sem.

Magyarázat

A konvertált ADC értéket legalább 16 bites *raw* változóban kell tárolni, hiszen 12 bitre konfiguráltuk a felbontást. Illetve a beállítások között szerepel, hogy szoftveresen triggerelve szeretnénk használni az ADC-t, ezért minden mérés előtt el kell indítani a konverziót.

2. feladat 0/5 pont

VDDA = 3,5V esetén elméletben milyen értékek között mozoghat az eredmény?

Válasz

- 0-965
- 0-1023
- 0-3861
- 0-4095
- 0-61790
- 0-65535
- Egyik sem.

Magyarázat

Mivel az ADC 3,5V-os referencia feszültségére egy 3,3V-os potmétert kötött, valamint az ADC 12 bites konverzióra lett konfigurálva, ezért (2^12-1)/3,5*3,3=3861 lesz a maximálisan mért feszültség.

3. feladat 0/5 pont

Az adatlapot böngészve rájött, hogy az SMT32 egyik DMA modulját is használhatná, mely automatikusan a memóriába másolná a konvertált értékeket. A terve, hogy nagyjából másodpercenként egyszer triggereljen egy ADC olvasást, amely DMA segítségével automatikusan a memóriába kerül, majd UART-on kiírja ellenőrzésre az eredményt. Ehhez a DMA-t az alábbi módon konfigurálta be:

```
DMA_HandleTypeDef hdma_adc;
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC1)
  {
    __HAL_RCC_ADC1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_adc.Instance = DMA1_Channel1;
    hdma_adc.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_adc.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc.Init.MemInc = DMA_MINC_DISABLE;
    hdma_adc.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc.Init.Mode = DMA_NORMAL;
    hdma_adc.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_adc) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_ADC);
    __HAL_LINKDMA(hadc,DMA_Handle,hdma_adc);
  }
}
```

Amikor a DMA elvégezte a másolási műveletet, meghívódik a *HAL_ADC_ConvCpltCallback* függvény.

Az ADC konfigurációja az intróban már ismertetett módon történik, szoftveres triggerrel 12 bites értékeket fogadunk.

Melyik kódrészlettel működik helyesen a programja?

Válaszok

- [x]
```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  hadc->Instance->CFGR1 &= ~ADC_CFGR1_DMAEN;
}
void adcConv(void *argument)
{
  uint16_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start_DMA(&hadc, (uint32_t*)&raw, 1);
    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    osDelay(1000);
  }
}
```

- [x]

```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  HAL_ADC_Stop_DMA(hadc);
}
void adcConv(void *argument)
{
  uint16_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start_DMA(&hadc, (uint32_t*)&raw, 1);
    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    osDelay(1000);
  }
}
```

```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  HAL_ADC_Stop_DMA(hadc);
}
void adcConv(void *argument)
{
  uint8_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start_DMA(&hadc, (uint32_t*)&raw, 1);
    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    osDelay(1000);
  }
}
```

```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  hadc->Instance->CFGR1 &= ~ADC_CFGR1_DMAEN;
}
void adcConv(void *argument)
{
  uint8_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start_DMA(&hadc, (uint32_t*)&raw, 1);
    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    osDelay(1000);
  }
}
```

```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  hadc->Instance->CFGR2 &= ~ADC_CFGR2_DMAEN;
}
void adcConv(void *argument)
{
  uint8_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start_DMA(&hadc, (uint32_t*)&raw, 1);
    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    osDelay(1000);
  }
}
```

- [ ]

```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  hadc->Instance->CFGR2 &= ~ADC_CFGR2_DMAEN;
}
void adcConv(void *argument)
{
  uint16_t raw;
  char msg[10];

  HAL_ADCEx_Calibration_Start(&hadc);
  /* Infinite loop */
  for(;;)
  {
    HAL_ADC_Start_DMA(&hadc, (uint32_t*)&raw, 1);
    sprintf(msg, "%hu\r\n", raw);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    osDelay(1000);
  }
}
```

- [ ] Egyik sem.

Magyarázat

A DMA transzfert vagy a HAL réteg által nyújtott *HAL_ADC_Stop_DMA* függvényhívással, vagy direkt regiszterírással is le lehet állítani a callback függvényben. Az adatlap szerint az ADC_CFGR1 regiszter DMAEN bitjét kell 0-ra állítani ehhez. Mivel 12 bites ADC adatokat fogadunk, legalább 16 bites *raw* változót kell használnunk.

4. feladat 0/5 pont

Tegyük fel, hogy Ájóti János az STM32F091 kontroller DMA1-es perifériájának 7-es csatornáján szeretné beállítani a MEM2MEM bitet 1-es értékre, hogy engedélyezze a memória-memória átviteli módot. Melyik regiszterben kell ezt beállítania? Írd le a regiszter pontos nevét az adatlap szerint.

Válaszok

A helyes válasz:

DMA_CCR7

Magyarázat

Az adatlap szerint a DMA_CCR7-es regiszterben kell beállítani ezt az értéket.

5. feladat 0/5 pont

Miután sikeresen finomított az analóg értékek beolvasási módján, eszébe jut, hogy még egyszerűbb lenne, ha a DMA nem csak a memóriába másolná az ADC által konvertált értékeket, hanem rögtön kitenné azokat UART-ra.

Melyik DMA konfiguráció segítségével tudja elérni ezt?

Válasz

```c
DMA_HandleTypeDef hdma_adc;
DMA_HandleTypeDef hdma_uart2;
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC1)
  {
    __HAL_RCC_ADC1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_adc.Instance = DMA1_Channel1;
    hdma_adc.Init.Direction = DMA_PERIPH_TO_PERIPH;
    hdma_adc.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc.Init.MemInc = DMA_MINC_DISABLE;
    hdma_adc.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc.Init.Mode = DMA_NORMAL;
    hdma_adc.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_adc) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_ADC);
    __HAL_LINKDMA(hadc,DMA_Handle,hdma_adc);
  }
}
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART2)
  {
    __HAL_RCC_USART2_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF1_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_usart2_tx.Instance = DMA1_Channel1;
    hdma_usart2_tx.Init.Direction = DMA_PERIPH_TO_PERIPH;
    hdma_usart2_tx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_usart2_tx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_usart2_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_usart2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_usart2_tx.Init.Mode = DMA_NORMAL;
    hdma_usart2_tx.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_usart2_tx) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_USART2_TX);
    __HAL_LINKDMA(huart,hdmatx,hdma_usart2_tx);
    HAL_NVIC_SetPriority(USART2_IRQn, 3, 0);
    HAL_NVIC_EnableIRQ(USART2_IRQn);
  }
}
```

```c
DMA_HandleTypeDef hdma_adc;
DMA_HandleTypeDef hdma_uart2;
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC1)
  {
    __HAL_RCC_ADC1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_adc.Instance = DMA1_Channel1;
    hdma_adc.Init.Direction = DMA_PERIPH_TO_PERIPH;
    hdma_adc.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc.Init.MemInc = DMA_MINC_DISABLE;
    hdma_adc.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc.Init.Mode = DMA_NORMAL;
    hdma_adc.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_adc) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_ADC);
    __HAL_LINKDMA(hadc,DMA_Handle,hdma_adc);
  }
}
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART2)
  {
    __HAL_RCC_USART2_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF1_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_usart2_tx.Instance = DMA1_Channel1;
    hdma_usart2_tx.Init.Direction = DMA_PERIPH_TO_PERIPH;
    hdma_usart2_tx.Init.PeriphInc = DMA_PINC_ENABLE;
    hdma_usart2_tx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_usart2_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_usart2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_usart2_tx.Init.Mode = DMA_NORMAL;
    hdma_usart2_tx.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_usart2_tx) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_USART2_TX);
    __HAL_LINKDMA(huart,hdmatx,hdma_usart2_tx);
    HAL_NVIC_SetPriority(USART2_IRQn, 3, 0);
    HAL_NVIC_EnableIRQ(USART2_IRQn);
  }
}
```

```c
DMA_HandleTypeDef hdma_adc;
DMA_HandleTypeDef hdma_uart2;
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC1)
  {
    __HAL_RCC_ADC1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_adc.Instance = DMA1_Channel1;
    hdma_adc.Init.Direction = DMA_PERIPH_TO_PERIPH;
    hdma_adc.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc.Init.MemInc = DMA_MINC_DISABLE;
    hdma_adc.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc.Init.Mode = DMA_NORMAL;
    hdma_adc.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_adc) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_ADC);
    __HAL_LINKDMA(hadc,DMA_Handle,hdma_adc);
  }
}
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART2)
  {
    __HAL_RCC_USART2_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF1_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_usart2_tx.Instance = DMA1_Channel1;
    hdma_usart2_tx.Init.Direction = DMA_PERIPH_TO_PERIPH;
    hdma_usart2_tx.Init.PeriphInc = DMA_PINC_ENABLE;
    hdma_usart2_tx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_usart2_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD
    hdma_usart2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_usart2_tx.Init.Mode = DMA_NORMAL;
    hdma_usart2_tx.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_usart2_tx) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_USART2_TX);
    __HAL_LINKDMA(huart,hdmatx,hdma_usart2_tx);
    HAL_NVIC_SetPriority(USART2_IRQn, 3, 0);
    HAL_NVIC_EnableIRQ(USART2_IRQn);
  }
}
```

```
DMA_HandleTypeDef hdma_adc;
DMA_HandleTypeDef hdma_uart2;
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC1)
  {
    __HAL_RCC_ADC1_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_adc.Instance = DMA1_Channel1;
    hdma_adc.Init.Direction = DMA_PERIPH_TO_PERIPH;
    hdma_adc.Init.PeriphInc = DMA_PINC_ENABLE;
    hdma_adc.Init.MemInc = DMA_MINC_DISABLE;
    hdma_adc.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc.Init.Mode = DMA_NORMAL;
    hdma_adc.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_adc) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_ADC);
    __HAL_LINKDMA(hadc,DMA_Handle,hdma_adc);
  }
}
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART2)
  {
    __HAL_RCC_USART2_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF1_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    hdma_usart2_tx.Instance = DMA1_Channel1;
    hdma_usart2_tx.Init.Direction = DMA_PERIPH_TO_PERIPH;
    hdma_usart2_tx.Init.PeriphInc = DMA_PINC_ENABLE;
    hdma_usart2_tx.Init.MemInc = DMA_MINC_DISABLE;
    hdma_usart2_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_usart2_tx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_usart2_tx.Init.Mode = DMA_NORMAL;
    hdma_usart2_tx.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_usart2_tx) != HAL_OK){
      Error_Handler();
    }
    __HAL_DMA1_REMAP(HAL_DMA1_CH1_USART2_TX);
    __HAL_LINKDMA(huart,hdmatx,hdma_usart2_tx);
    HAL_NVIC_SetPriority(USART2_IRQn, 3, 0);
    HAL_NVIC_EnableIRQ(USART2_IRQn);
  }
}
```

◯ Egyik sem.

Magyarázat

Az STM32 DMA modulja az adatlap szerint nem képes két periféria közötti átvitelre, csak háromféle módon lehet beállítani: memória-memória, memória-periféria, periféria-memória.

KÉSZÍTETTE

Megjelenés

Világos