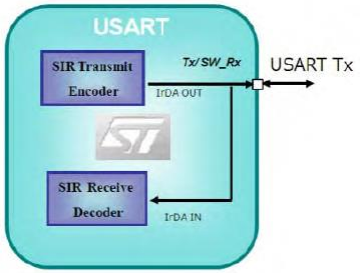
PRIMENA MIKROKONTROLERA-MS1PMK

7. deo

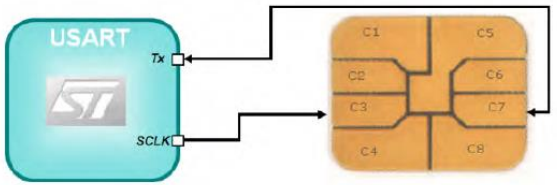
2017

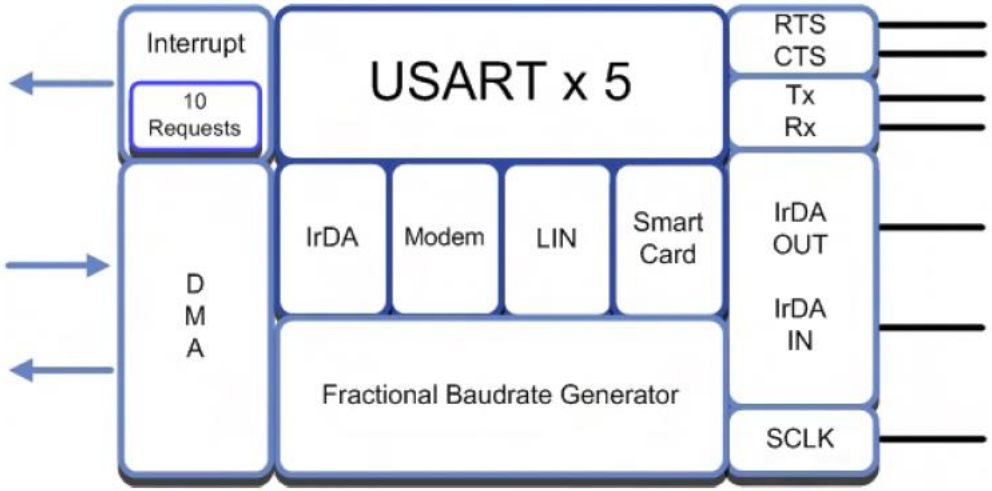
Nenad Jovičić

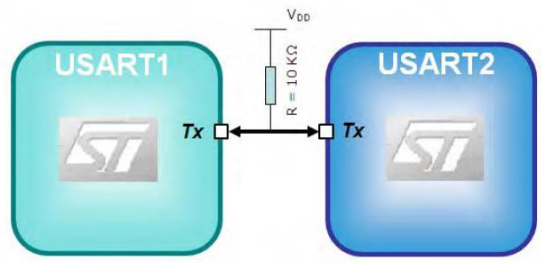
USART



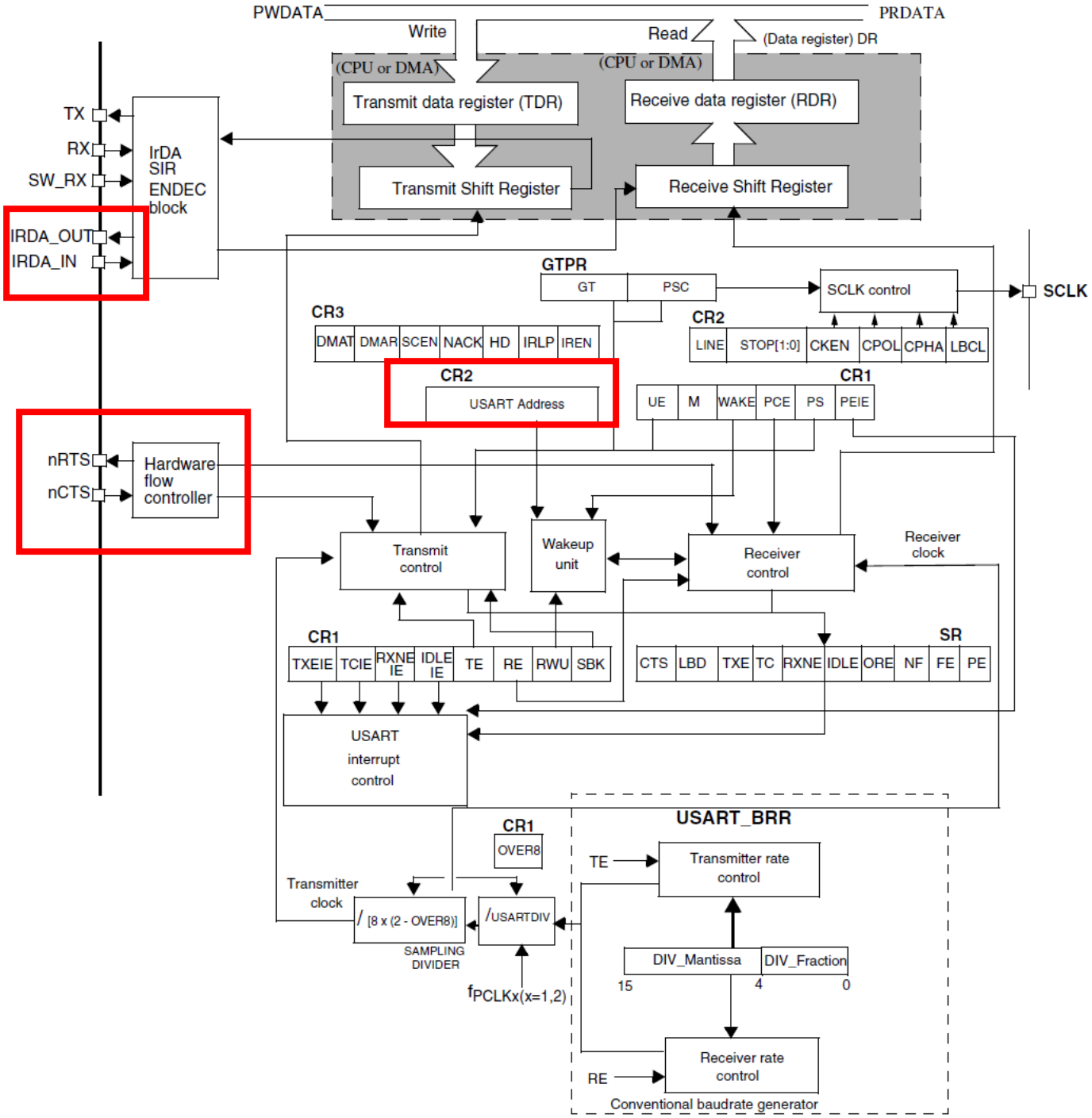
- Posebne karakteristike u odnosu na standardni USART:
  - 8 ili 16 bita oversampling
  - Frakcioni baud-rate generator (slično kao MSP)
  - 7, 8 ili 9 bita podatak
  - Encoder decoder za podršku IRDA prenosu
  - Podržan smartcard protokol ISO7816-3
  - Half-duplex komunikacija preko jedne žice
  - Dva odvojena DMA kanala za predaju i prijem
  - Četrnaest izvora prekida (regularni i oni za detekciju greške)
  - RTS/CTS hardverski handshaking





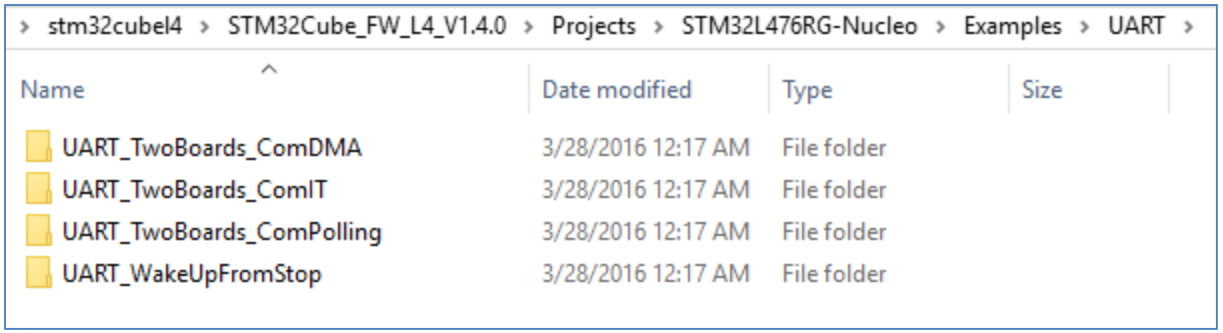


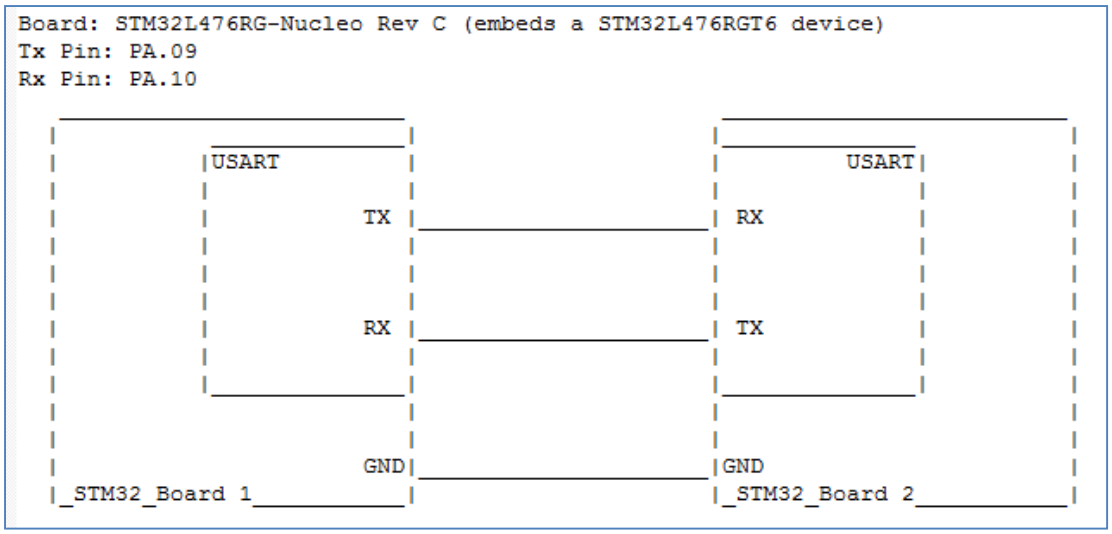
USART
blok šema



STM CUBE
UART Projekti

- Predajna ploča pošalje poruku prijemnoj, a nakon toga Prijemna ploča vrati tu istu poruku predajnoj.





Projekat UART_TwoBoards_ComPolling

main.c | startup_stm32l476xx.s

```
41  /* Includes ------------------------------------------------------------------*/
42  #include "main.h"
43
44  /** @addtogroup STM32L4xx_HAL_Examples
45    * @{
46    */
47
48  /** @addtogroup UART_TwoBoards_ComPolling
49    * @{
50    */
51
52  /* Private typedef -----------------------------------------------------------*/
53  /* Private define ------------------------------------------------------------*/
54  #define TRANSMITTER_BOARD
55
56  /* Private macro -------------------------------------------------------------*/
57  /* Private variables ---------------------------------------------------------*/
58  /* UART handler declaration */
59  UART_HandleTypeDef UartHandle;
60  __IO uint32_t UserButtonStatus = 0;  /* set to 1 after User Button interrupt  */
61
62  /* Buffer used for transmission */
63  uint8_t aTxBuffer[] = " **** UART_TwoBoards_ComPolling ****  **** UART_TwoBoards_ComPolling
64
65  /* Buffer used for reception */
66  uint8_t aRxBuffer[RXBUFFERSIZE];
67
68  /* Private function prototypes -----------------------------------------------*/
69  void SystemClock_Config(void);
70  static void Error_Handler(void);
71  static uint16_t Buffercmp(uint8_t* pBuffer1, uint8_t* pBuffer2, uint16_t BufferLength);
72
```

Predajna strana

main.c

```
126 #ifdef TRANSMITTER_BOARD
127
128   /* Configure User push-button in Interrupt mode */
129   BSP_PB_Init(BUTTON_USER, BUTTON_MODE_EXTI);
130
131   /* Wait for User push-button press before starting the Communication.
132      In the meantime, LED2 is blinking */
133   while(UserButtonStatus == 0)
134   {
135       /* Toggle LED2*/
136       BSP_LED_Toggle(LED2);
137       HAL_Delay(100);
138   }
139
140   BSP_LED_Off(LED2);
141
142
143   /* The board sends the message and expects to receive it back */
144
145   /*##-2- Start the transmission process #####################################*/
146   /* While the UART in reception process, user can transmit data through
147      "aTxBuffer" buffer */
148   if(HAL_UART_Transmit(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE, 5000)!= HAL_OK)
149   {
150     Error_Handler();
151   }
152
153
154   /*##-3- Put UART peripheral in reception process ###########################*/
155   if(HAL_UART_Receive(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE, 5000) != HAL_OK)
156   {
157     Error_Handler();
158   }
159
160
161 #else
162
163   /* The board receives the message and sends it back */
```

Prevodjenje u zavisnosti da li je ploča predajnik ili prijemnik

Sve se implementira preko takozvanih blokirajućih funkcija

Prijemna strana

main.c

```
159
160
161  #else
162
163    /* The board receives the message and sends it back */
164
165    /*##-2- Put UART peripheral in reception process ###########################*/
166    if(HAL_UART_Receive(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE, 0x1FFFFFF) != HAL_OK)
167    {
168      Error_Handler();
169    }
170
171
172    /*##-3- Start the transmission process #####################################*/
173    /* While the UART in reception process, user can transmit data through
174       "aTxBuffer" buffer */
175    if(HAL_UART_Transmit(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE, 5000)!= HAL_OK)
176    {
177      Error_Handler();
178    }
179
180
181  #endif /* TRANSMITTER_BOARD */
182
183    /*##-4- Compare the sent and received buffers ##############################*/
184    if(Buffercmp((uint8_t*)aTxBuffer,(uint8_t*)aRxBuffer,RXBUFFERSIZE))
185    {
186      Error_Handler();
187    }
188
189    /* Turn on LED2 if test passes then enter infinite loop */
190    BSP_LED_On(LED2);
191    /* Infinite loop */
192    while (1)
193    {
194    }
195  }
196
```

Prijemnik koristi iste funkcije za prijem i kasniju predaju poruke

Na kraju se vrši provera poslate i primljene poruke. Ovo ima smisla samo kod predajnika...

How to use this driver? (uvek u stm32l4xx_HAL_PPP.c)

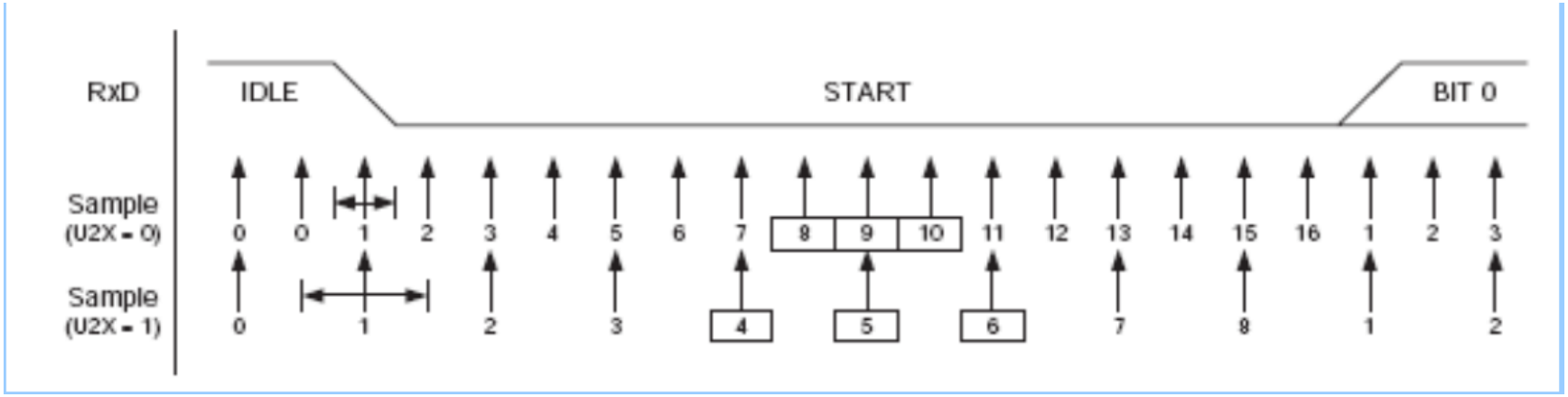
```
==============================================================================
                     ##### How to use this driver #####
==============================================================================
 [..]
   The UART HAL driver can be used as follows:

   (#) Declare a UART_HandleTypeDef handle structure (eg. UART_HandleTypeDef huart).
   (#) Initialize the UART low level resources by implementing the HAL_UART_MspInit() API:
       (++) Enable the USARTx interface clock.
       (++) UART pins configuration:
            (+++) Enable the clock for the UART GPIOs.
            (+++) Configure these UART pins as alternate function pull-up.
       (++) NVIC configuration if you need to use interrupt process (HAL_UART_Transmit_IT()
            and HAL_UART_Receive_IT() APIs):
            (+++) Configure the USARTx interrupt priority.
            (+++) Enable the NVIC USART IRQ handle.
       (++) UART interrupts handling:
             -@@-  The specific UART interrupts (Transmission complete interrupt,
               RXNE interrupt and Error Interrupts) are managed using the macros
                 HAL_UART_ENABLE_IT() and __HAL_UART_DISABLE_IT() inside the transmit and receive processes.
       (++) DMA Configuration if you need to use DMA process (HAL_UART_Transmit_DMA()
            and HAL_UART_Receive_DMA() APIs):
            (+++) Declare a DMA handle structure for the Tx/Rx channel.
            (+++) Enable the DMAx interface clock.
            (+++) Configure the declared DMA handle structure with the required Tx/Rx parameters.
            (+++) Configure the DMA Tx/Rx channel.
            (+++) Associate the initialized DMA handle to the UART DMA Tx/Rx handle.
            (+++) Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx/Rx
```

How to use this driver? (uvek u stm32l4xx_HAL_PPP.c)

```
(#) Program the Baud Rate, Word Length, Stop Bit, Parity, Hardware
    flow control and Mode (Receiver/Transmitter) in the huart handle Init structure.

(#) If required, program UART advanced features (TX/RX pins swap, auto Baud rate detection,...)
    in the huart handle AdvancedInit structure.

(#) For the UART asynchronous mode, initialize the UART registers by calling
    the HAL_UART_Init() API.

(#) For the UART Half duplex mode, initialize the UART registers by calling
    the HAL_HalfDuplex_Init() API.

(#) For the UART LIN (Local Interconnection Network) mode, initialize the UART registers
    by calling the HAL_LIN_Init() API.

(#) For the UART Multiprocessor mode, initialize the UART registers
    by calling the HAL_MultiProcessor_Init() API.

(#) For the UART RS485 Driver Enabled mode, initialize the UART registers
    by calling the HAL_RS485Ex_Init() API.

[..]
(@) These API's (HAL_UART_Init(), HAL_HalfDuplex_Init(), HAL_LIN_Init(), HAL_MultiProcessor_Init(),
    also configure the low level Hardware GPIO, CLOCK, CORTEX...etc) by
    calling the customized HAL_UART_MspInit() API.
```

Svi parametri USART-a

- Inicijalizacija USART-a
  - Baud rate (frakcioni generator, ako se radi sa 8 bita oversampling-a, 12bita celobrojni deo, 4 bita frakcioni)
  - Word length (7, 8, 9 bita)
  - Stop bits (0.5, 1, 1.5, 2)
  - Parity (none, even, odd)
  - Mode (RX, TX, RX&TX)
  - Hardware flow control (CTS i/ili RTS aktivni ili ne)
  - Oversampling (16 ili 8 bita)
  - One bit sampling (odlucuje se na osnovu jednog semplovanog bita ili na osnovu 3)



Inicijalizacija

main.c

```
102   /* Put the USART peripheral in the Asynchronous mode (UART Mode) */
103   /* UART configured as follows:
104       - Word Length = 8 Bits
105       - Stop Bit = One Stop bit
106       - Parity = None
107       - BaudRate = 9600 baud
108       - Hardware flow control disabled (RTS and CTS signals) */
109   UartHandle.Instance        = USARTx;
110
111   UartHandle.Init.BaudRate   = 9600;
112   UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
113   UartHandle.Init.StopBits   = UART_STOPBITS_1;
114   UartHandle.Init.Parity     = UART_PARITY_NONE;
115   UartHandle.Init.HwFlowCtl  = UART_HWCONTROL_NONE;
116   UartHandle.Init.Mode       = UART_MODE_TX_RX;
117   if(HAL_UART_DeInit(&UartHandle) != HAL_OK)
118   {
119     Error_Handler();
120   }
121   if(HAL_UART_Init(&UartHandle) != HAL_OK)
122   {
123     Error_Handler();
124   }
125
```

Pokretanje USART-a

- HAL_UART_DeInit(&UartHandle)
  - uart state BUSY
  - uart disable
  - clear all control registers
  - uart state RESET
- HAL_UART_Init(&UartHandle)
  - HAL_UART_MspInit
  - UART_SetConfig
  - __HAL_UART_ENABLE

stm32l4xx_hal_msp.c | main.c

```
68  void HAL_UART_MspInit(UART_HandleTypeDef *huart)
69  {
70    GPIO_InitTypeDef  GPIO_InitStruct;
71
72    /*##-1- Enable peripherals and GPIO Clocks #####################
73    /* Enable GPIO TX/RX clock */
74    USARTx_TX_GPIO_CLK_ENABLE();
75    USARTx_RX_GPIO_CLK_ENABLE();
76
77
78    /* Enable USARTx clock */
79    USARTx_CLK_ENABLE();
80
81    /*##-2- Configure peripheral GPIO ##########################
82    /* UART TX GPIO pin configuration  */
83    GPIO_InitStruct.Pin       = USARTx_TX_PIN;
84    GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
85    GPIO_InitStruct.Pull      = GPIO_PULLUP;
86    GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_VERY_HIGH;
87    GPIO_InitStruct.Alternate = USARTx_TX_AF;
88
89    HAL_GPIO_Init(USARTx_TX_GPIO_PORT, &GPIO_InitStruct);
90
91    /* UART RX GPIO pin configuration  */
92    GPIO_InitStruct.Pin = USARTx_RX_PIN;
93    GPIO_InitStruct.Alternate = USARTx_RX_AF;
94
95    HAL_GPIO_Init(USARTx_RX_GPIO_PORT, &GPIO_InitStruct);
96  }
97
```

Svaka posebna periferija, koja pripada nekoj generičkoj klasi, poseduje neke specifičnosti koje se mogu razlikovati kod svakog pojedinačnog mikrokontrolera. Na primer koji pinovi se koriste. To verovatno NIKADA neće ući u sklop generičkih drajvera...

main.c | stm32l4xx_hal_uart.c

```
616
617 /**
618   * @brief Send an amount of data in blocking mode.
619   * @param huart: UART handle.
620   * @param pData: Pointer to data buffer.
621   * @param Size: Amount of data to be sent.
622   * @param Timeout: Timeout duration.
623   * @retval HAL status
624   */
625 HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)
626 {
627    uint16_t* tmp;
628
629   if((huart->State == HAL_UART_STATE_READY) || (huart->State == HAL_UART_STATE_BUSY_RX))
630   {
631     if((pData == NULL ) || (Size == 0))
632     {
633       return  HAL_ERROR;
634     }
635
636     /* Process Locked */
637     __HAL_LOCK(huart);
638
639     huart->ErrorCode = HAL_UART_ERROR_NONE;
640     /* Check if a non-blocking receive process is
641     if(huart->State == HAL_UART_STATE_BUSY_RX)
642     {
643       huart->State = HAL_UART_STATE_BUSY_TX_RX;
644     }
645     else
646     {
647       huart->State = HAL_UART_STATE_BUSY_TX;
648     }
649
650     huart->TxXferSize = Size;
651     huart->TxXferCount = Size;
652     while(huart->TxXferCount > 0)
653     {
654       huart->TxXferCount--;
655       if(UART_WaitOnFlagUntilTimeout(huart, UART_FLAG_TXE, RESET, Timeout) != HAL_OK)
656       {
657         return HAL_TIMEOUT;
658       }
659       if ((huart->Init.WordLength == UART_WORDLENGTH_9B) && (huart->Init.Parity == UART_PARITY_NONE))
660       {
661         tmp = (uint16_t*) pData;
```

Sve metode poliranja se zasnivaju na čekanju na neki flag. U ovom slučaju flag TXE, tj. da je predajni buffer spreman za novi podatak.

Da bi funkcionalnost bila bar u nekoj meri neblokirajuća uvek se uvodi Timeout.

ain.c | stm32l4xx_hal_uart.c

```
/**
  * @brief  Handle UART Communication Timeout.
  * @param  huart: UART handle.
  * @param  Flag: specifies the UART flag to check.
  * @param  Status: the Flag status (SET or RESET).
  * @param  Timeout: Timeout duration.
  * @retval HAL status
  */
HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout(UART_HandleTypeDef *huart, uint32_t Flag, FlagStatus Status, uint32_t Timeout)
{
  uint32_t tickstart = HAL_GetTick();

  /* Wait until flag is set */
  if(Status == RESET)
  {
    while(__HAL_UART_GET_FLAG(huart, Flag) == RESET)
    {
      /* Check for the Timeout */
      if(Timeout != HAL_MAX_DELAY)
      {
        if((Timeout == 0) || ((HAL_GetTick()-tickstart) > Timeout))
        {
          /* Disable TXE, RXNE, PE and ERR (Frame error, noise error, overrun error) interrupts for the interrupt process */
          __HAL_UART_DISABLE_IT(huart, UART_IT_TXE);
          __HAL_UART_DISABLE_IT(huart, UART_IT_RXNE);
          __HAL_UART_DISABLE_IT(huart, UART_IT_PE);
          __HAL_UART_DISABLE_IT(huart, UART_IT_ERR);

          huart->State= HAL_UART_STATE_READY;

          /* Process Unlocked */
          __HAL_UNLOCK(huart);

          return HAL_TIMEOUT;
        }
      }
    }
  }
  else
  {
    while(__HAL_UART_GET_FLAG(huart, Flag) != RESET)
    {
      /* Check for the Timeout */
      if(Timeout != HAL_MAX_DELAY)
      {
```

Timeout funkcionalnost se implementira preko sistemskog tajmera

U pitanju je prosta kalkulacija korišćenjem relativnog sistemskog vremena

Ako nešto nije u redu preduzimaju se akcije...

main.c

```
90          - Set NVIC Group Priority to 4
91          - Low Level Initialization
92        */
93      HAL_Init();
94
95      /* Configure the system clock to 80 MHz */
96      SystemClock_Config();
97
98      /* Configure LED2 */
99      BSP_LED_Init(LED2);
100
```

stm32l4xx_hal.c

```
156     *
157     * @retval HAL status
158     */
159   HAL_StatusTypeDef HAL_Init(void)
160   {
161     /* Configure Flash prefetch, Instruction cache, Data cache */
162     /* Default configuration at reset is:                       */
163     /* - Prefetch disabled                                      */
164     /* - Instruction cache enabled                              */
165     /* - Data cache enabled                                     */
166   #if (INSTRUCTION_CACHE_ENABLE == 0)
167       __HAL_FLASH_INSTRUCTION_CACHE_DISABLE();
168   #endif /* INSTRUCTION_CACHE_ENABLE */
169
170   #if (DATA_CACHE_ENABLE == 0)
171       __HAL_FLASH_DATA_CACHE_DISABLE();
172   #endif /* DATA_CACHE_ENABLE */
173
174   #if (PREFETCH_ENABLE != 0)
175     __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
176   #endif /* PREFETCH_ENABLE */
177
178     /* Set Interrupt Group Priority */
179     HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
180
181     /* Use SysTick as time base source and configure 1ms tick (defau
182     HAL_InitTick(TICK_INT_PRIORITY);
183
184     /* Init the low level hardware */
185     HAL_MspInit();
```

HAL drajveri često koriste poling logiku za implementaciju raznih funkcionalnosti i sada je jasno zašto je u okviru HAL_Init() funkcije i inicijalizacija sistemskog tajmera...

STM CUBE

Projekat UART_TwoBoards_ComIT

readme.txt | main.c | stm32l4xx_hal_gpio.c | stm32l4xx.h | stm32_hal_legacy.h

```
141
142
143   /* The board sends the message and expects to receive it back */
144
145   /*##-2- Start the transmission process ###################################*/
146   /* While the UART in reception process, user can transmit data through
147      "aTxBuffer" buffer */
148   if(HAL_UART_Transmit_IT(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE)!= HAL_OK)
149   {
150     Error_Handler();
151   }
152
153   /*##-3- Wait for the end of the transfer ###################################*/
154   while (UartReady != SET)
155   {
156   }
157
158   /* Reset transmission flag */
159   UartReady = RESET;
160
161   /*##-4- Put UART peripheral in reception process ###########################*/
162   if(HAL_UART_Receive_IT(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE) != HAL_OK)
163   {
164     Error_Handler();
165   }
166
167 #else
168
169   /* The board receives the message and sends it back */
170
171   /*##-2- Put UART peripheral in reception process ###########################*/
172   if(HAL_UART_Receive_IT(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE) != HAL_OK)
173   {
174     Error_Handler();
175   }
176
```

Struktura programa je praktično identična, sa razlikom da se funkcije zovu malo drugačije....

STM CUBE

Pr…nIT

```
5   /*##-2- Start the transmission process ###################################*/
6   /* While the UART in reception process, user can transmit data through
7     "aTxBuffer" buffer */
8   if(HAL_UART_Transmit(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE, 5000)!= HAL_OK)
9   {
10    Error_Handler();
11  }
12
13
14  /*##-3- Put UART peripheral in reception process ###########################*/
15  if(HAL_UART_Receive(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE, 5000) != HAL_OK)
16  {
17    Error_Handler();
18  }
```

readme.txt | main.c

```
141
142
143   /* The boa
144
145   /*##-2- St
146   /* While the UART in reception process, user can transmit data through
147      "aTxBuffer" buffer */
148   if(HAL_UART_Transmit_IT(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE)!= HAL_OK)
149   {
150     Error_Handler();
151   }
152
153   /*##-3- Wait for the end of the transfer ###################################*/
154   while (UartReady != SET)
155   {
156   }
157
158   /* Reset transmission flag */
159   UartReady = RESET;
160
161   /*##-4- Put UART peripheral in reception process ###########################*/
162   if(HAL_UART_Receive_IT(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE) != HAL_OK)
163   {
164     Error_Handler();
165   }
166
167 #else
168
169   /* The board receives the message and sends it back */
170
171   /*##-2- Put UART peripheral in reception process ###########################*/
172   if(HAL_UART_Receive_IT(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE) != HAL_OK)
173   {
174     Error_Handler();
175   }
176
```

Ali, ne postoji TIMEOUT.

```
*/
HAL_StatusTypeDef HAL_UART_Transmit_IT(UART_HandleTy
{

  if((huart->State == HAL_UART_STATE_READY) || (huar
  {
    if((pData == NULL ) || (Size == 0))
    {
      return HAL_ERROR;
    }

    /* Process Locked */
    __HAL_LOCK(huart);

    huart->pTxBuffPtr = pData;
    huart->TxXferSize = Size;
    huart->TxXferCount = Size;

    huart->ErrorCode = HAL_UART_ERROR_NONE;
    /* Check if a receive process is ongoing or not
    if(huart->State == HAL_UART_STATE_BUSY_RX)
    {
      huart->State = HAL_UART_STATE_BUSY_TX_RX;
    }
    else
    {
      huart->State = HAL_UART_STATE_BUSY_TX;
    }

    /* Process Unlocked */
    __HAL_UNLOCK(huart);

    /* Enable the UART Transmit Data Register Empty
    __HAL_UART_ENABLE_IT(huart, UART_IT_TXE);

    return HAL_OK;
```

```
*/
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTyp
{

  if((huart->State == HAL_UART_STATE_READY) || (huar
  {
    if((pData == NULL ) || (Size == 0))
    {
      return HAL_ERROR;
    }

    /* Process Locked */
    __HAL_LOCK(huart);

    huart->pRxBuffPtr = pData;
    huart->RxXferSize = Size;
    huart->RxXferCount = Size;

    /* Computation of UART mask to apply to RDR regi
    UART_MASK_COMPUTATION(huart);

    huart->ErrorCode = HAL_UART_ERROR_NONE;
    /* Check if a transmit process is ongoing or not
    if(huart->State == HAL_UART_STATE_BUSY_TX)
    {
      huart->State = HAL_UART_STATE_BUSY_TX_RX;
    }
    else
    {
      huart->State = HAL_UART_STATE_BUSY_RX;
    }

    /* Enable the UART Parity Error Interrupt */
    __HAL_UART_ENABLE_IT(huart, UART_IT_PE);

    /* Enable the UART Error Interrupt: (Frame error
    __HAL_UART_ENABLE_IT(huart, UART_IT_ERR);

    /* Process Unlocked */
    __HAL_UNLOCK(huart);

    /* Enable the UART Data Register not empty Inter
    __HAL_UART_ENABLE_IT(huart, UART_IT_RXNE);

    return HAL_OK;
```

USARTx_IRQHandler

```
271         EXPORT      TIM4_IRQHandler                 [WEAK]
272         EXPORT      I2C1_EV_IRQHandler              [WEAK]
273         EXPORT      I2C1_ER_IRQHandler              [WEAK]
274         EXPORT      I2C2_EV_IRQHandler              [WEAK]
275         EXPORT      I2C2_ER_IRQHandler              [WEAK]
276         EXPORT      SPI1_IRQHandler                 [WEAK]
277         EXPORT      SPI2_IRQHandler                 [WEAK]
278         EXPORT      USART1_IRQHandler               [WEAK]
279         EXPORT      USART2_IRQHandler               [WEAK]
280         EXPORT      USART3_IRQHandler               [WEAK]
281         EXPORT      EXTI15_10_IRQHandler            [WEAK]
282         EXPORT      RTC_Al
```

stm32l4xx_it.c | readme.txt | main.c

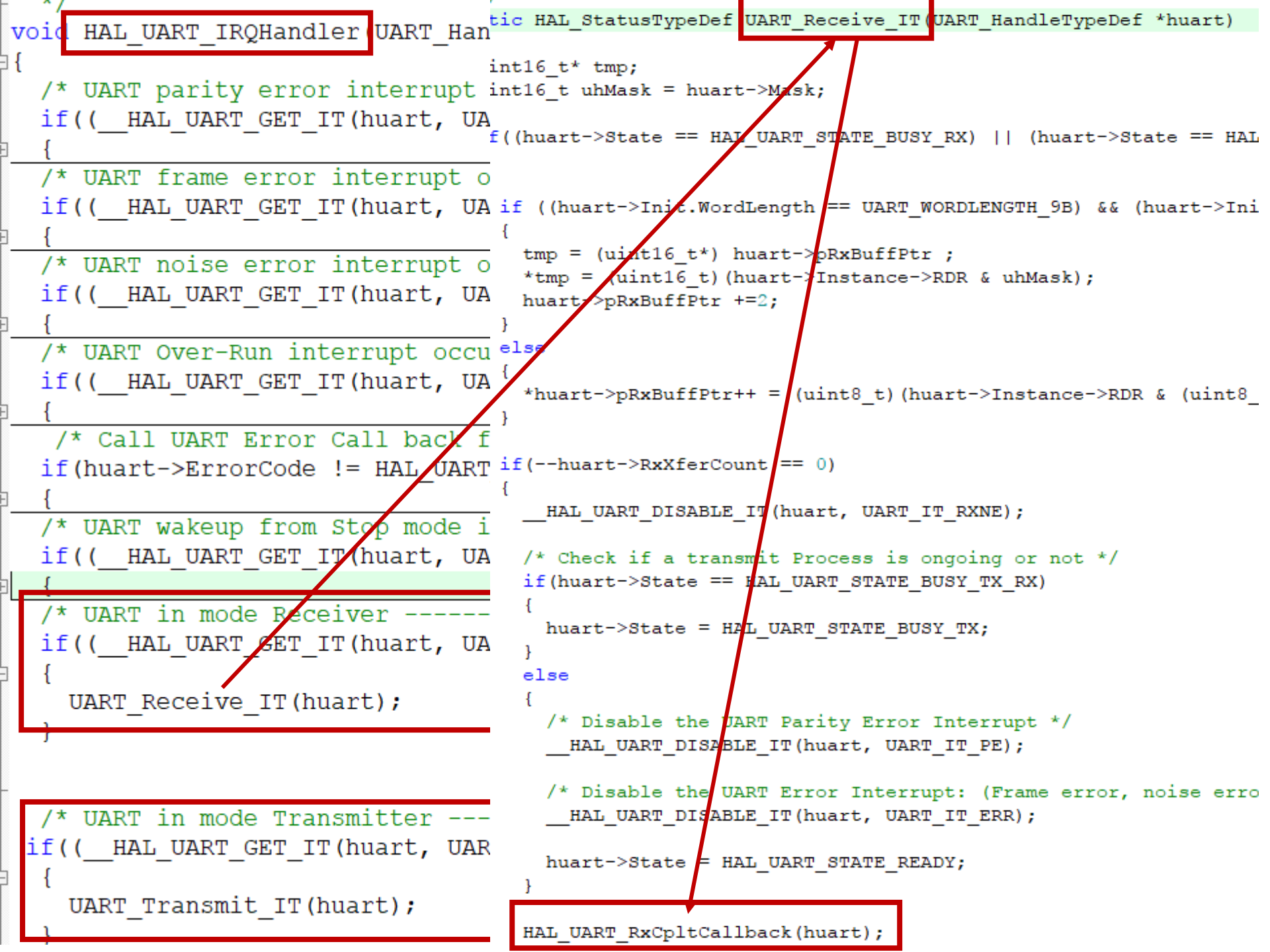
```
163  /******************************
164  /*                  STM32L4xx
165  /*  Add here the Interrupt H
166  /*  available peripheral int
167  /*  file (startup_stm32l4xx.
168  /******************************
169  /**
170    * @brief  This function handles UART interrupt request.
171    * @param  None
172    * @retval None
173    * @Note   This function is redefined in "main.h" and related to DMA
174    *         used for USART data transmission
175    */
176  void USARTx_IRQHandler(void)
177  {
178    HAL_UART_IRQHandler(&UartHandle);
179  }
```

h.c | stm32l4xx_hal_uart.c | startup_stm32l476xx.s | main.h

```
3  #define USARTx_RX_PIN                    GPIO_PIN_10
4  #define USARTx_RX_GPIO_PORT              GPIOA
5  #define USARTx_RX_AF                     GPIO_AF7_USART1
6
7  /* Definition for USARTx's NVIC */
8  #define USARTx_IRQn                      USART1_IRQn
9  #define USARTx_IRQHandler                USART1_IRQHandler
```

```
  */
void HAL_UART_IRQHandler(UART_HandleTypeDef *huart)
{
  /* UART parity error interrupt occurred -------------
  if((__HAL_UART_GET_IT(huart, UART_IT_PE) != RESET) &&
  {
  /* UART frame error interrupt occurred --------------
  if((__HAL_UART_GET_IT(huart, UART_IT_FE) != RESET) &&
  {
  /* UART noise error interrupt occurred --------------
  if((__HAL_UART_GET_IT(huart, UART_IT_NE) != RESET) &&
  {
  /* UART Over-Run interrupt occurred -----------------
  if((__HAL_UART_GET_IT(huart, UART_IT_ORE) != RESET) &
  {
   /* Call UART Error Call back function if need be ---
  if(huart->ErrorCode != HAL_UART_ERROR_NONE)
  {
  /* UART wakeup from Stop mode interrupt occurred ----
  if((__HAL_UART_GET_IT(huart, UART_IT_WUF) != RESET) &
  {
  /* UART in mode Receiver ----------------------------
  if((__HAL_UART_GET_IT(huart, UART_IT_RXNE) != RESET)
  {
    UART_Receive_IT(huart);
  }


  /* UART in mode Transmitter -------------------------
 if((__HAL_UART_GET_IT(huart, UART_IT_TXE) != RESET) &&
  {
    UART_Transmit_IT(huart);
  }
```

HAL_UART_IRQHandler


void HAL_UART_IRQHandler(UART_Han
{
/* UART parity error interrupt
if((__HAL_UART_GET_IT(huart, UA
{
/* UART frame error interrupt o
if((__HAL_UART_GET_IT(huart, UA
{
/* UART noise error interrupt o
if((__HAL_UART_GET_IT(huart, UA
{
/* UART Over-Run interrupt occu
if((__HAL_UART_GET_IT(huart, UA
{
/* Call UART Error Call back f
if(huart->ErrorCode != HAL_UART
{
/* UART wakeup from Stop mode i
if((__HAL_UART_GET_IT(huart, UA
{
/* UART in mode Receiver ------
if((__HAL_UART_GET_IT(huart, UA
{
UART_Receive_IT(huart);
}
/* UART in mode Transmitter ---
if((__HAL_UART_GET_IT(huart, UAR
{
UART_Transmit_IT(huart);
}
tic HAL_StatusTypeDef UART_Receive_IT(UART_HandleTypeDef *huart)
int16_t* tmp;
int16_t uhMask = huart->Mask;
f((huart->State == HAL_UART_STATE_BUSY_RX) || (huart->State == HAL
if ((huart->Init.WordLength == UART_WORDLENGTH_9B) && (huart->Ini
{
tmp = (uint16_t*) huart->pRxBuffPtr ;
*tmp = (uint16_t)(huart->Instance->RDR & uhMask);
huart->pRxBuffPtr +=2;
}
else
{
*huart->pRxBuffPtr++ = (uint8_t)(huart->Instance->RDR & (uint8_
}
if(--huart->RxXferCount == 0)
{
__HAL_UART_DISABLE_IT(huart, UART_IT_RXNE);
/* Check if a transmit Process is ongoing or not */
if(huart->State == HAL_UART_STATE_BUSY_TX_RX)
{
huart->State = HAL_UART_STATE_BUSY_TX;
}
else
{
/* Disable the UART Parity Error Interrupt */
__HAL_UART_DISABLE_IT(huart, UART_IT_PE);
/* Disable the UART Error Interrupt: (Frame error, noise erro
__HAL_UART_DISABLE_IT(huart, UART_IT_ERR);
huart->State = HAL_UART_STATE_READY;
}
HAL_UART_RxCpltCallback(huart);

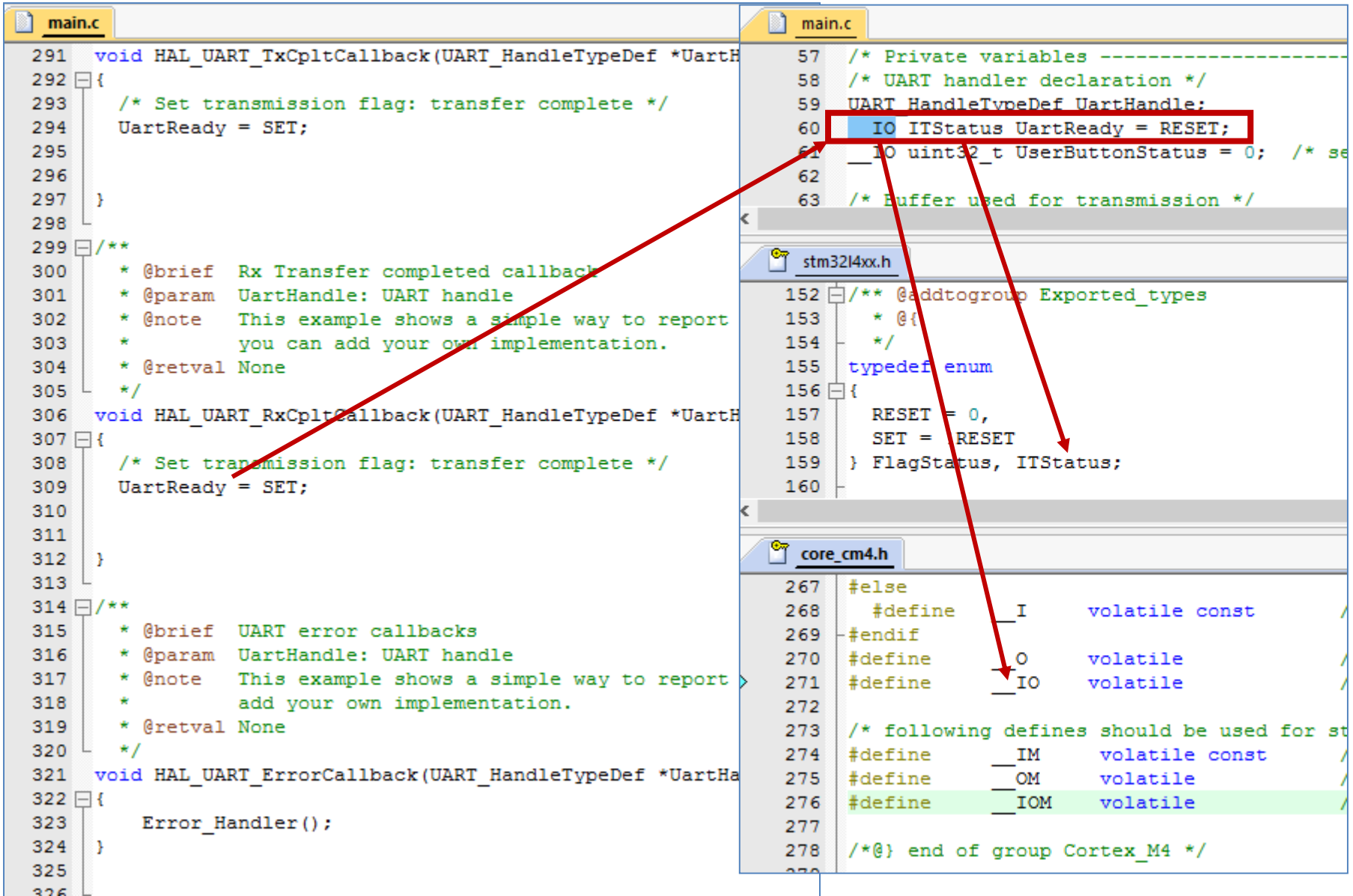
I naravno Callback funkcije

main.c

```
291  void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle)
292  {
293    /* Set transmission flag: transfer complete */
294    UartReady = SET;
295
296
297  }
298
299  /**
300    * @brief  Rx Transfer completed callback
301    * @param  UartHandle: UART handle
302    * @note   This example shows a simple way to report end of
303    *         you can add your own implementation.
304    * @retval None
305    */
306  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
307  {
308    /* Set transmission flag: transfer complete */
309    UartReady = SET;
310
311
312  }
313
314  /**
315    * @brief  UART error callbacks
316    * @param  UartHandle: UART handle
317    * @note   This example shows a simple way to report transfe
318    *         add your own implementation.
319    * @retval None
320    */
321  void HAL_UART_ErrorCallback(UART_HandleTypeDef *UartHandle)
322  {
323      Error_Handler();
324  }
325
```

Globalna promenljiva koju menjamo samo u prekidnoj rutini, a ispitujemo status u glavnom programu.

A šta kaže optimizer?



main.c

```
291  void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartH
292  {
293    /* Set transmission flag: transfer complete */
294    UartReady = SET;
295
296
297  }
298
299  /**
300    * @brief  Rx Transfer completed callback
301    * @param  UartHandle: UART handle
302    * @note   This example shows a simple way to report
303    *         you can add your own implementation.
304    * @retval None
305    */
306  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartH
307  {
308    /* Set transmission flag: transfer complete */
309    UartReady = SET;
310
311
312  }
313
314  /**
315    * @brief  UART error callbacks
316    * @param  UartHandle: UART handle
317    * @note   This example shows a simple way to report
318    *         add your own implementation.
319    * @retval None
320    */
321  void HAL_UART_ErrorCallback(UART_HandleTypeDef *UartHa
322  {
323      Error_Handler();
324  }
325
```

main.c

```
57  /* Private variables ---------------------
58  /* UART handler declaration */
59  UART_HandleTypeDef UartHandle;
60    IO ITStatus UartReady = RESET;
61  __IO uint32_t UserButtonStatus = 0;  /* se
62
63  /* Buffer used for transmission */
```

stm32l4xx.h

```
152  /** @addtogroup Exported_types
153    * @{
154    */
155  typedef enum
156  {
157    RESET = 0,
158    SET = !RESET
159  } FlagStatus, ITStatus;
160
```

core_cm4.h

```
267  #else
268    #define     __I     volatile const       /
269  #endif
270  #define       __O     volatile             /
271  #define       __IO    volatile             /
272
273  /* following defines should be used for st
274  #define       __IM     volatile const      /
275  #define       __OM     volatile            /
276  #define       __IOM    volatile            /
277
278  /*@} end of group Cortex_M4 */
```

Volatile - demistifikacija

- Volatile se koristi za:
  1. Memory-mapped peripheral registers
  2. Global variables modified by an interrupt service routine
  3. Global variables accessed by multiple tasks within a multi-threaded application

Volatile - primer

- Primer: Na adresi 0x1234 se nalazi periferijski registar, na primer ulazni osmobitni port. Potrebno će čekati logičku jedinicu na bilo kom ulazu:

```
uint8_t * pReg = (uint8_t *) 0x1234;

// Wait for register to become non-zero
while (*pReg == 0) { } // Do something else
```

- Kompajler koji štedi prostor će verovatno ovo da prevede na sledeći način:

```
  mov ptr, #0x1234 mov a, @ptr

loop:
  bz loop
```

Ako se koristi Volatile...

- Daklaracija promenljive:

```
uint8_t volatile * pReg = (uint8_t volatile *) 0x1234;
```

- Kako je to kompajler preveo:

```
  mov ptr, #0x1234

loop:
  mov a, @ptr
  bz loop
```

U našem slučaju....

- Pogrešno bi bilo pretpostaviti da je promenljiva UartReady neki periferijski registar jer nas deklaracija **__IO ITStatus UartReady** navodi na to.
- Mislim da bi bilo korektnije napisati jednostavno **volatile ITStatus UartReady**, ali efekat je isti.

Pitanje

- Ako definišemo promenljivu

  volatile uint64_t var
- Glavni čeka da varijabla postane jednaka nuli

  var=-10;

  while(var!=0);

  Function();
- A prekid tajmera radi sledeću stvar:

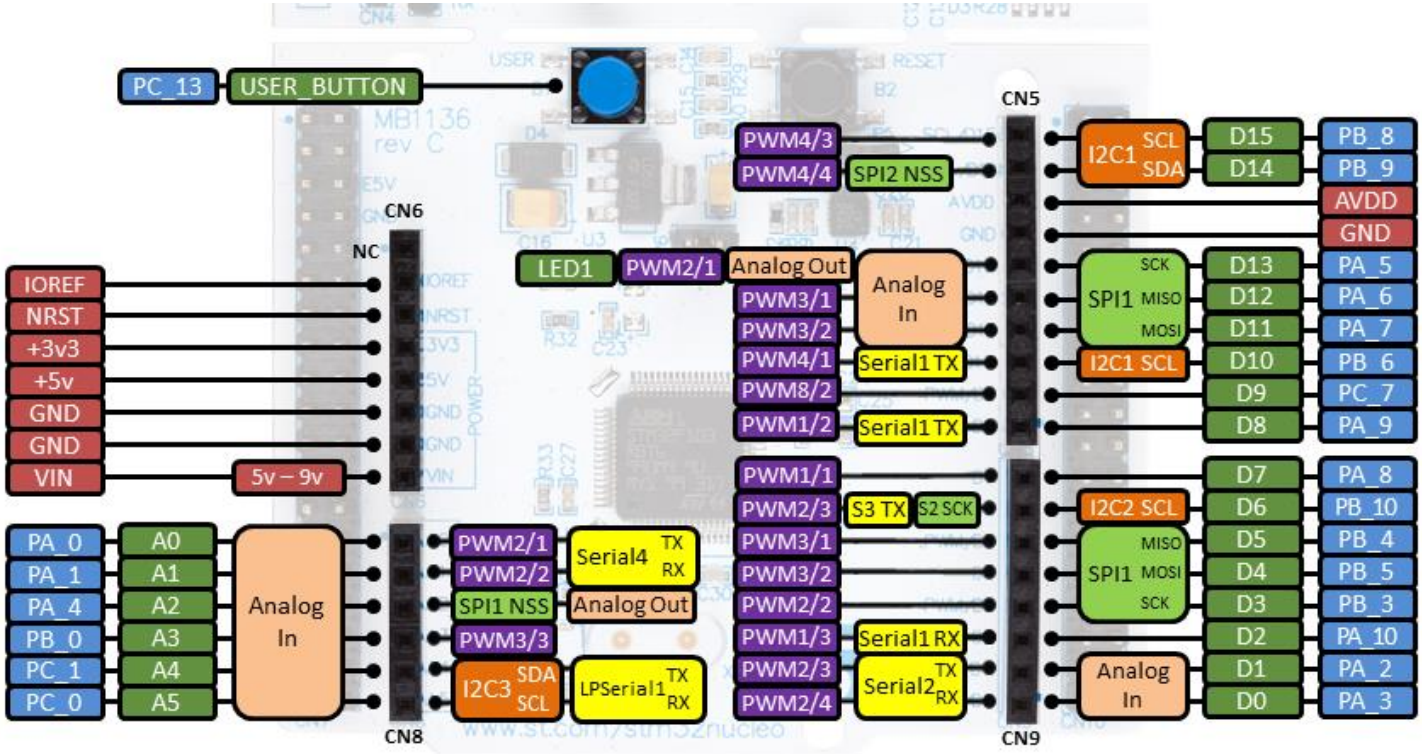
  var++;
- Da li će se funkcija Foo() garantovano izvršiti nakon 10 tajmerskih prekida? Ili možda 11...Ili...?

Zadatak

- Povezati dve ploče da komuniciraju međusobno pomoću projekta UART_TwoBoards_ComIT
- Jedna ploča je TRANSMITTER
- Druga je RECEIVER
- Na transmiter ploči se pritiskom na USER_BUTTON startuje komunikacija

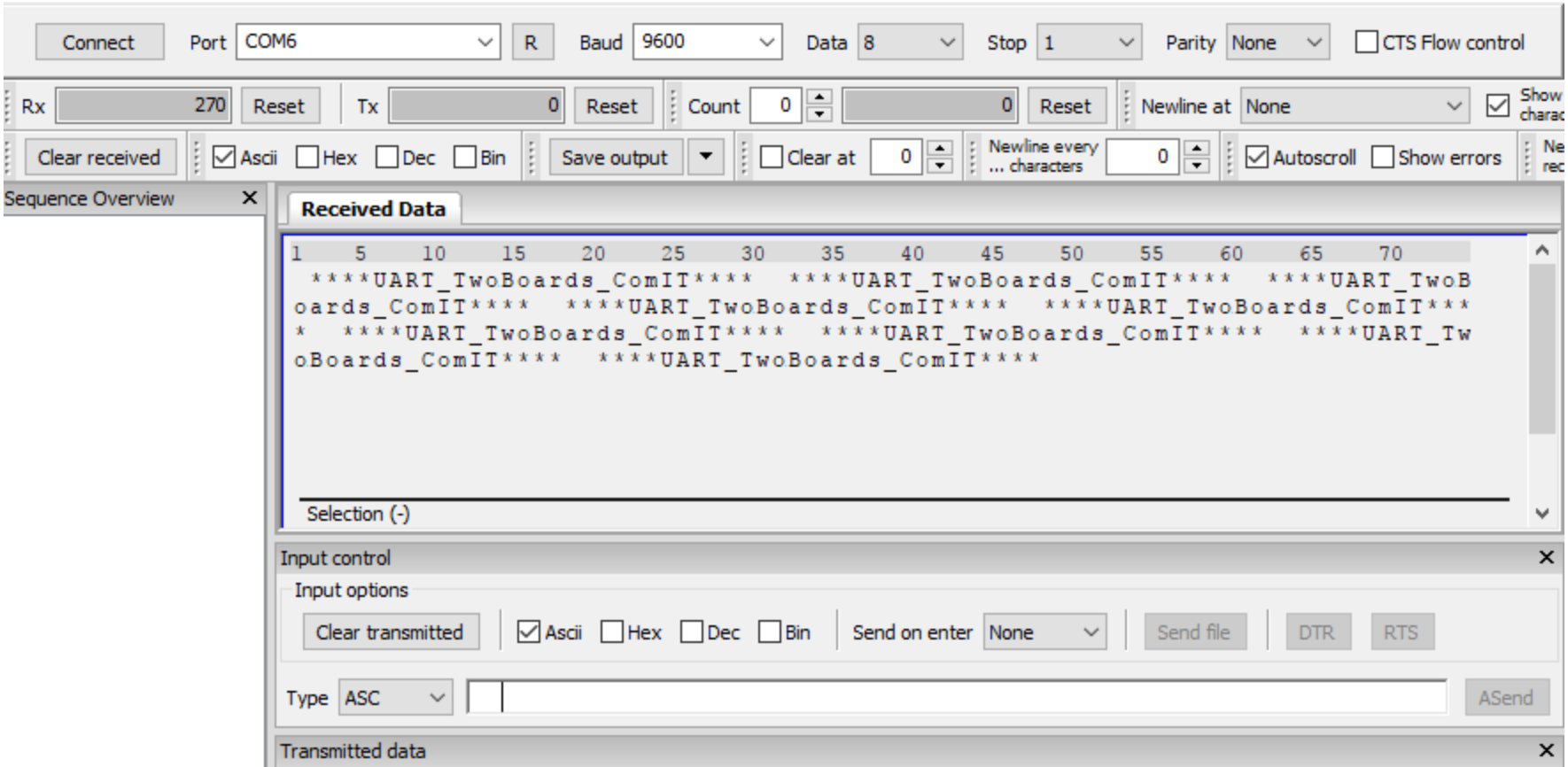
Zadatak

- Izmeniti projekat tako da se komunikacija vrši preko UART3 modula
- Datasheet za AF podešavanje pinova



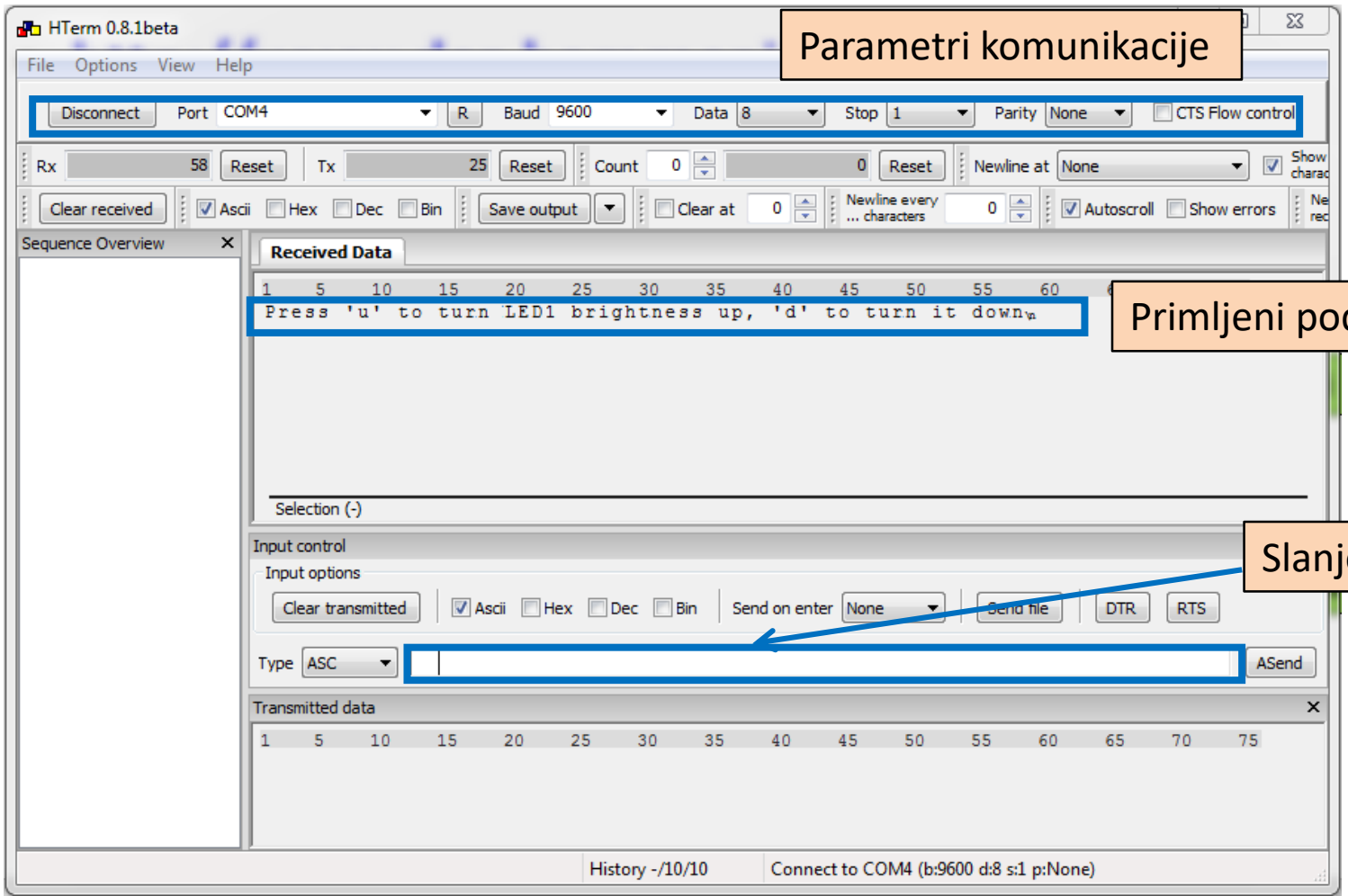
Zadatak

- Izmeniti projekat tako da se komunikacija vrši preko UART2 modula – slanje preko ST-Link-a na računar
- Datasheet za AF podešavanje pinova



Serijski protovi – testiranje korišćenjem HTerm-a

- http://www.der-hammer.info/terminal/

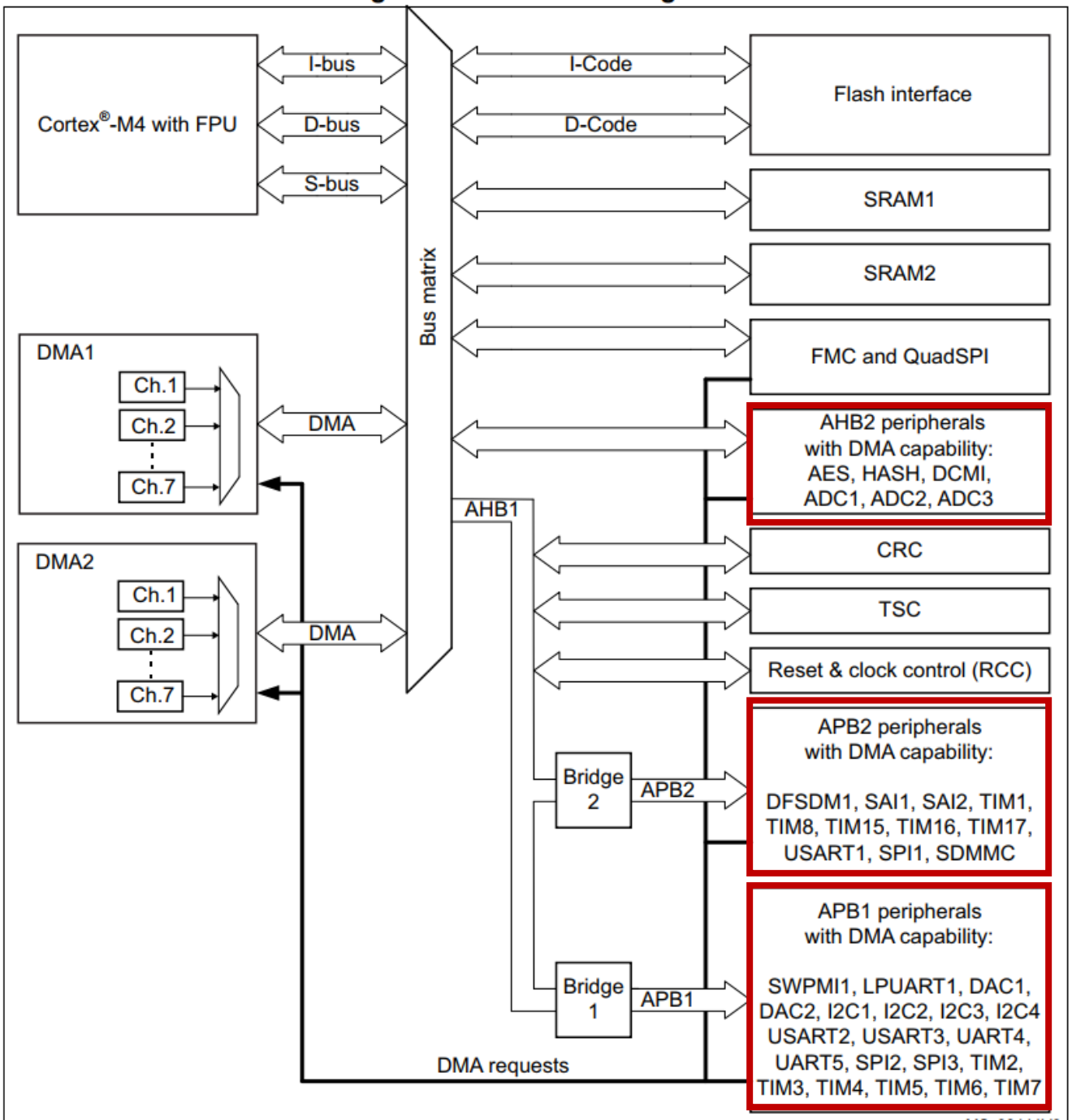


DMA na stm32l476RG

- Mikrokontroler ima 2 DMA kontrolera (DMA1, DMA2)
- DMA pruža:
  - **14 nezavisnih kanala** preko kojih se primaju DMA zahtevi
  - **Svaki kanal** proizvodi hardverski **DMA zahtev**, ali je omogućen **i softverski triger** na svakom od kanala.
  - **Prioriteti zahteva** od različitih kanala se softverski mogu podesiti na jedan od 4 dostupna nivoa. Ukoliko stignu zahtevi sa dva kanala istih prioriteta, prelazi se na hardverski prioritet gde kanal manjeg indeksa ima prednost.
  - Podržan prenos **8, 16 i 32-bitnih podataka**.
  - Podržani **cirkularni baferi**
  - Mogući su prenosi **u svim kombinacijama između memorije i periferija**.
  - Pristup Flash, SRAM memoriji i APB i AHB periferijama kao mogućim izvorima i destinacijama
  - Podržan prenos za tajmere, ADC, SPI, I2Cs, USART i DAC

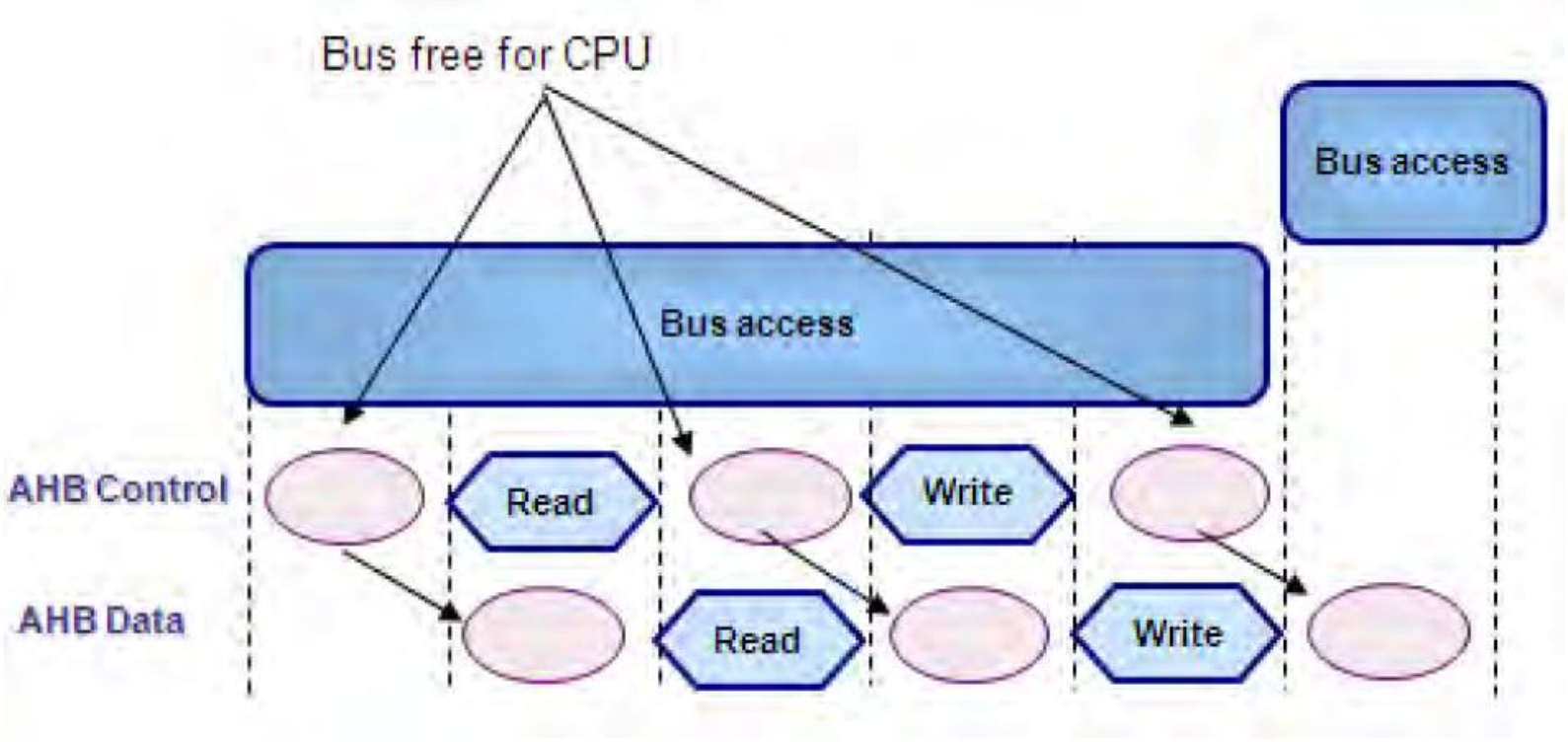
DMA arhitektura

Figure 29. DMA block diagram



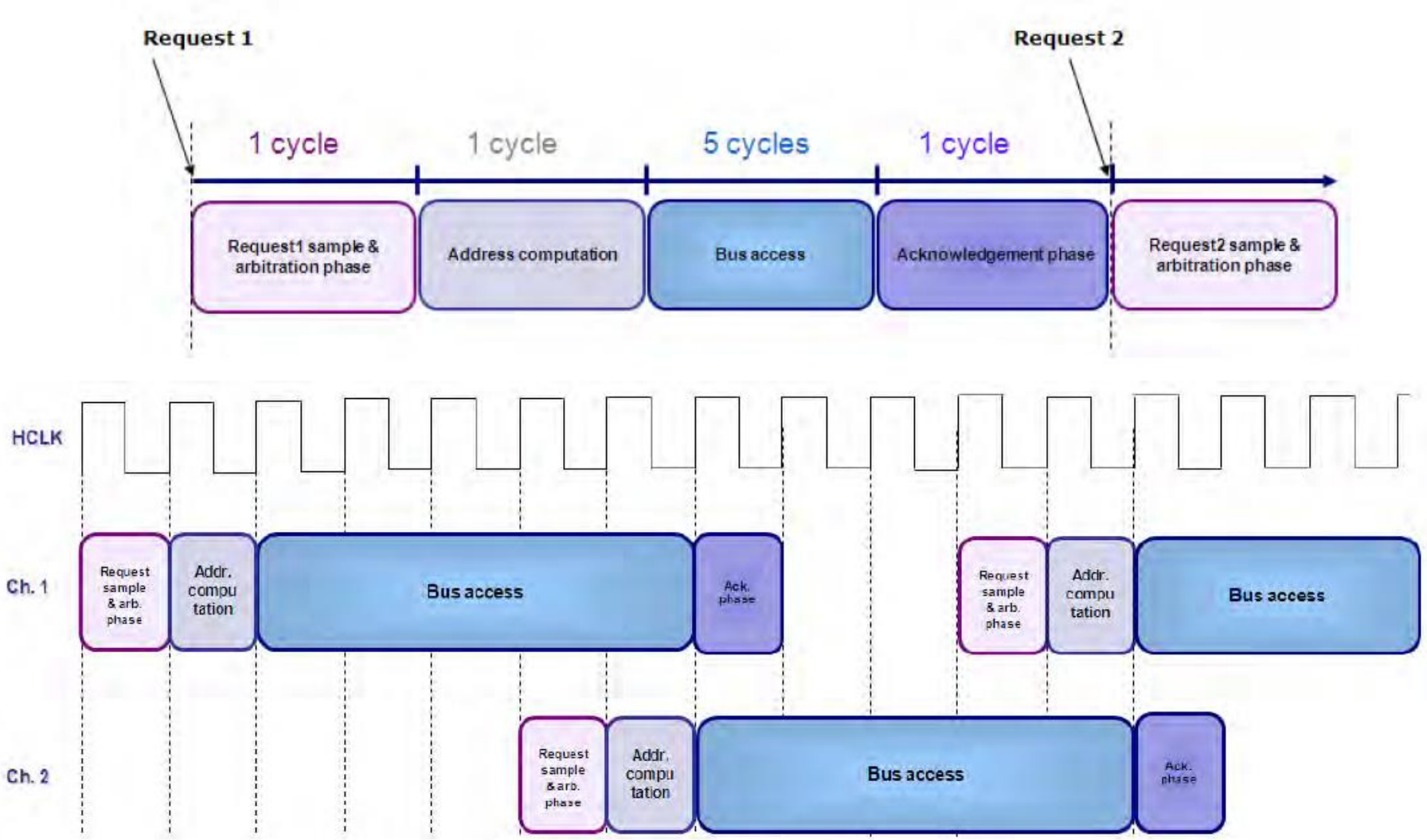
Preklapanje DMA i CPU

- DMA se preklapa sa CPU-om tako da nikada ne dolazi do potpunog blokiranja jednog ili drugog.



DMA prenos

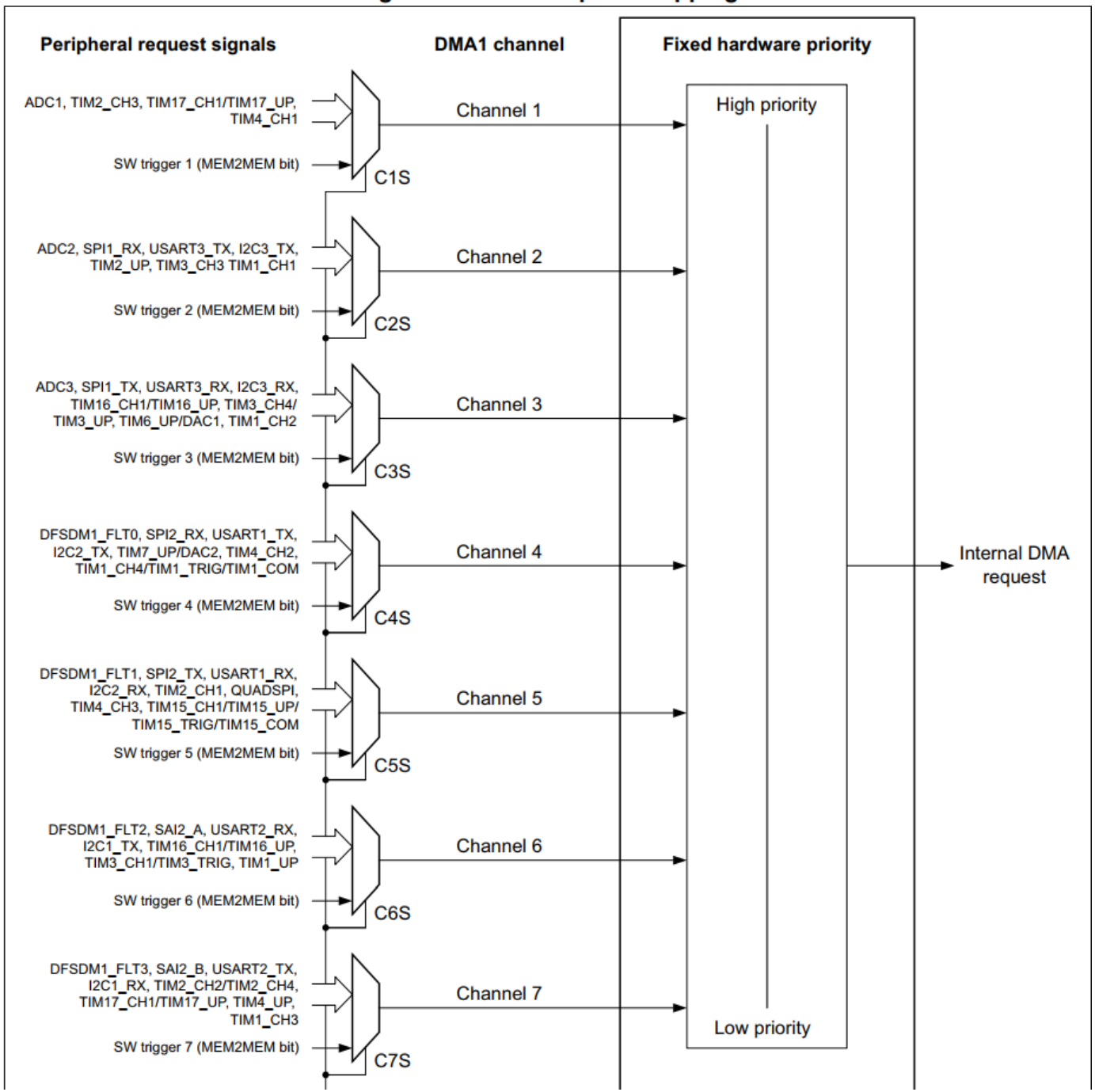
- Prenos se sastoji iz 4 faze



DMA1 kanali

- Reference manual

Figure 30. DMA1 request mapping



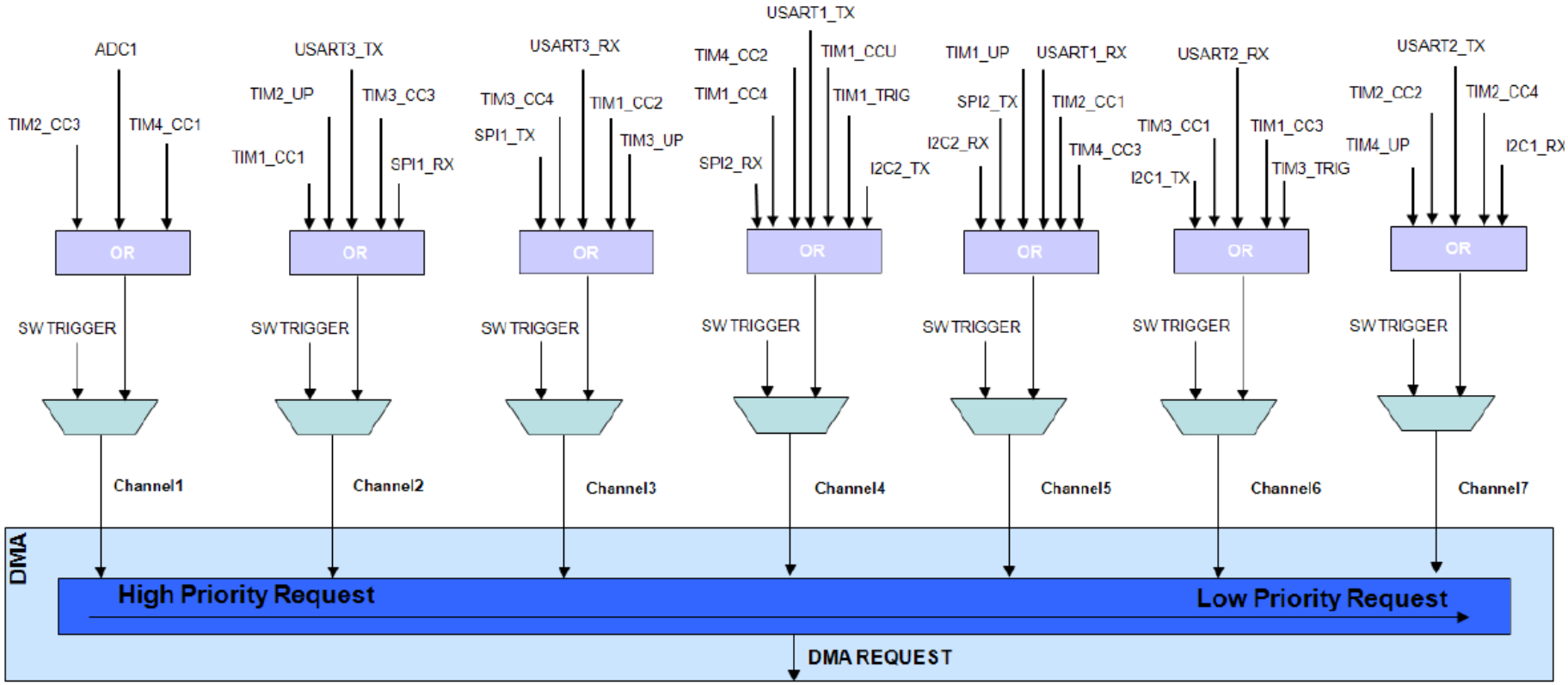
DMA2 kanali - veza sa periferijama

Table 46. Summary of the DMA2 requests for each channel

Request. number	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
0	I2C4_RX	I2C4_TX	ADC1	ADC2	ADC3	DCMI	-
1	SAI1_A	SAI1_B	SAI2_A	SAI2_B	-	SAI1_A	SAI1_B
2	UART5_TX	UART5_RX	UART4_TX	-	UART4_RX	USART1_TX	USART1_RX
3	SPI3_RX	SPI3_TX	-	TIM6_UP DAC1	TIM7_UP DAC2	-	QUADSPI
4	SWPMI1_RX	SWPMI1_TX	SPI1_RX	SPI1_TX	DCMI	LPUART1_ TX	LPUART1_ RX
5	TIM5_CH4 TIM5_TRIG	TIM5_CH3 TIM5_UP	-	TIM5_CH2	TIM5_CH1	I2C1_RX	I2C1_TX
6	AES_IN	AES_OUT	AES_OUT	-	AES_IN	-	HASH_IN
7	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	-	SDMMC1	SDMMC1	TIM8_CH1	TIM8_CH2

DMA prekidi

- Svaki DMA kanal ima tri prekida:
  - Transfer završen
  - Transfer na pola
  - Greška u transferu



Cirkularni režim

- Cirkularni režim se koristi kada postoji kontinualni dotok podataka (npr. ADC scan režim) koji se smeštaju u cirkularni bafer.
- U cirkularnom režimu se automatski obnavalja vrednost očekivanog broja podataka sa inicijalnom vrednošću podešenom pri konfiguraciji kanala, čime se obezbeđuje nesmetano kontinualno opsluživanje DMA zahteva (broj podataka nije konačan, a samim tim ni poznat).

STM CUBE

Projekat UART_TwoBoards_ComDMA

main.c

```
142 #ifdef TRANSMITTER_BOARD
143
144   /* The board sends the message and expects to receive it back */
145   /* DMA is programmed for reception before starting the transmission, in
146      be sure DMA Rx is ready when board 2 will start transmitting */
147
148   /*##-2- Program the Reception process ###################################
149   if(HAL_UART_Receive_DMA(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE)
150   {
151     Error_Handler();
152   }
153
154   /*##-3- Start the transmission process ##################################*/
155   /* While the UART in reception process, user can transmit data through
156      "aTxBuffer" buffer */
157   if(HAL_UART_Transmit_DMA(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE)!= HAL_OK)
158   {
159     Error_Handler();
160   }
161
162   /*##-4- Wait for the end of the transfer #################################*/
163   while (UartReady != SET)
164   {
165   }
166
167   /* Reset transmission flag */
168   UartReady = RESET;
169
170 #else
171
```

Struktura programa je praktično identična, sa razlikom da se funkcije zovu malo drugačije....

STM CUBE

Proj DMA

```
/*##-2- Start the transmission process ###################################*/
/* While the UART in reception process, user can transmit data through
   "aTxBuffer" buffer */
if(HAL_UART_Transmit(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE, 5000)!= HAL_OK)
{
  Error_Handler();
}


/*##-3- Put UART peripheral in reception process ##########################*/
if(HAL_UART_Receive(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE, 5000) != HAL_OK)
{
  Error_Handler();
}
```

main.c

```
#ifdef TRAN

  /* The board sends the message and expects to receive it back */
  /* DMA is programmed for reception before starting the transmission, in order to
     be sure DMA Rx is ready when board 2 will start transmitting */

  /*##-2- Program the Reception process #####################################*/
  if(HAL_UART_Receive_DMA(&UartHandle, (uint8_t *)aRxBuffer, RXBUFFERSIZE) != HAL_OK)
  {
    Error_Handler();
  }

  /*##-3- Start the transmission process ###################################*/
  /* While the UART in reception process, user can transmit data through
     "aTxBuffer" buffer */
  if(HAL_UART_Transmit_DMA(&UartHandle, (uint8_t*)aTxBuffer, TXBUFFERSIZE)!= HAL_OK)
  {
    Error_Handler();
  }

  /*##-4- Wait for the end of the transfer ##################################*/
  while (UartReady != SET)
  {
  }

  /* Reset transmission flag */
  UartReady = RESET;

#else
```

Ali, ne postoji TIMEOUT.

Inicijalizacija + Msp init

```
HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart)
{
  /* Check the UART handle allocation */
  if(huart == NULL)
  {
    return HAL_ERROR;
  }

  if(huart->Init.HwFlowCtl != UART_HWCONTROL_NONE)
  {
    /* Check the parameters */
    assert_param(IS_UART_HWFLOW_INSTANCE(huart->
  }
  else
  {
    /* Check the parameters */
    assert_param((IS_UART_INSTANCE(huart->Instan
  }

  if(huart->State == HAL_UART_STATE_RESET)
  {
    /* Allocate lock resource and initialize it
    huart->Lock = HAL_UNLOCKED;

    /* Init the low level hardware : GPIO, CLOCK
    HAL_UART_MspInit(huart);
  }
```

```
void HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
  static DMA_HandleTypeDef hdma_tx;
  static DMA_HandleTypeDef hdma_rx;

  GPIO_InitTypeDef  GPIO_InitStruct;

  /*##-1- Enable peripherals and GPIO Clocks ############
  /* Enable GPIO TX/RX clock */
  USARTx_TX_GPIO_CLK_ENABLE();
  USARTx_RX_GPIO_CLK_ENABLE();


  /* Enable USARTx clock */
  USARTx_CLK_ENABLE();

  /* Enable DMA clock */
  DMAx_CLK_ENABLE();

  /*##-2- Configure peripheral GPIO ####################
  /* UART TX GPIO pin configuration  */
  GPIO_InitStruct.Pin       = USARTx_TX_PIN;
  GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull      = GPIO_PULLUP;
  GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = USARTx_TX_AF;

  HAL_GPIO_Init(USARTx_TX_GPIO_PORT, &GPIO_InitStruct);

  /* UART RX GPIO pin configuration  */
  GPIO_InitStruct.Pin = USARTx_RX_PIN;
  GPIO_InitStruct.Alternate = USARTx_RX_AF;

  HAL_GPIO_Init(USARTx_RX_GPIO_PORT, &GPIO_InitStruct);
```

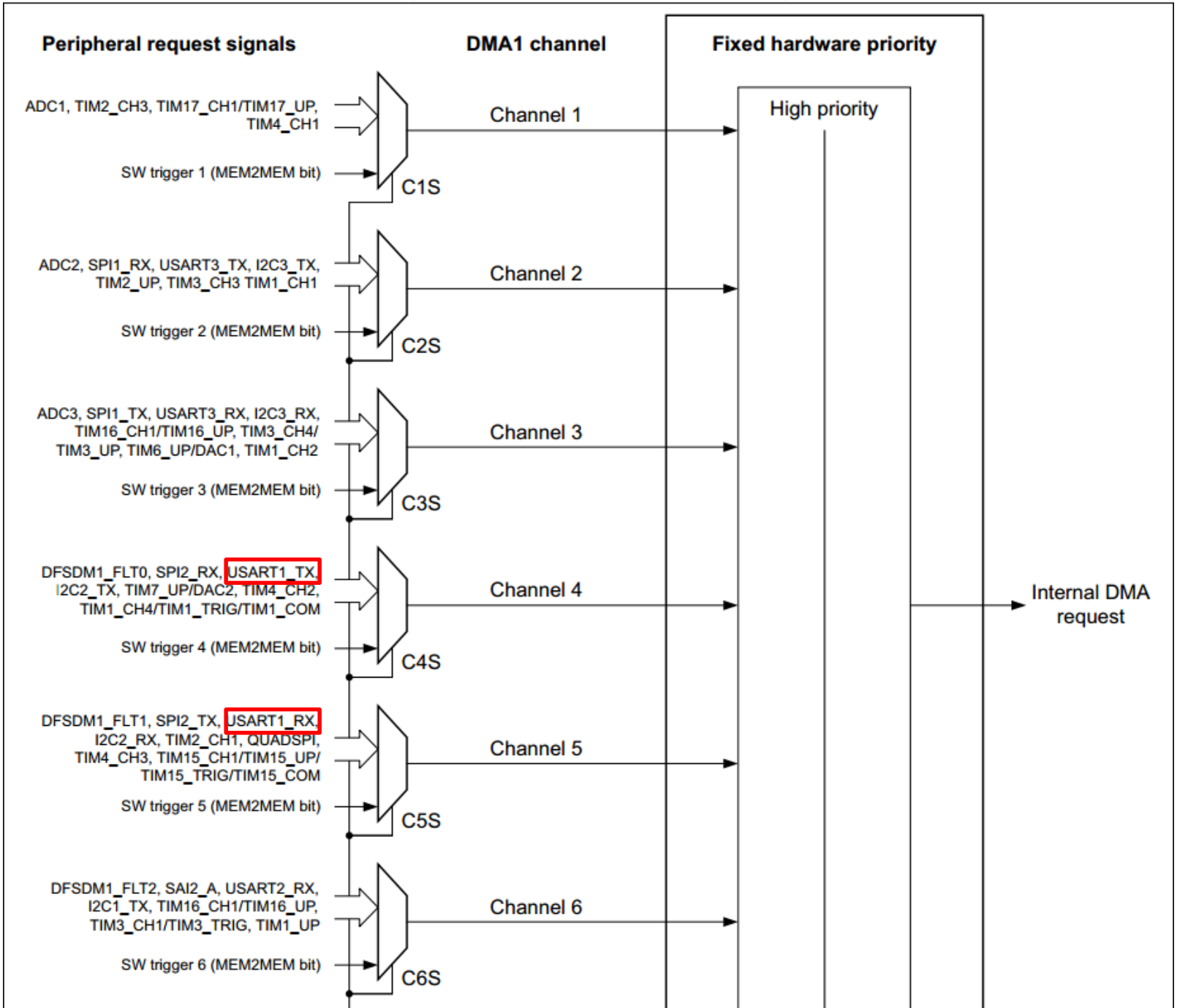
Msp init – DMA init

```
HAL_StatusTypeDef HAL_UART_Init UART_Hand
{
  /* Check the UART handle allocation */
  if(huart == NULL)
  {
    return HAL_ERROR;
  }

  if(huart->Init.HwFlowCtl != UART_HWCONTI
  {
    /* Check the parameters */
    assert_param(IS_UART_HWFLOW_INSTANCE(h
  }
  else
  {
    /* Check the ---------- */
    assert_param(
  }

  if(huart->State
  {
    /* Allocate l
    huart->Lock =

    /* Init the l
    HAL_UART_MspI
  }
```

```
/*##-3- Configure the DMA ##################################################*/
/* Configure the DMA handler for Transmission process */
hdma_tx.Instance                 = USARTx_TX_DMA_CHANNEL;
hdma_tx.Init.Direction           = DMA_MEMORY_TO_PERIPH;
hdma_tx.Init.PeriphInc           = DMA_PINC_DISABLE;
hdma_tx.Init.MemInc              = DMA_MINC_ENABLE;
hdma_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
hdma_tx.Init.MemDataAlignment    = DMA_MDATAALIGN_BYTE;
hdma_tx.Init.Mode                = DMA_NORMAL;
hdma_tx.Init.Priority            = DMA_PRIORITY_LOW;
hdma_tx.Init.Request             = USARTx_TX_DMA_REQUEST;

HAL_DMA_Init(&hdma_tx);

/* Associate the initialized DMA handle to the UART handle */
__HAL_LINKDMA(huart, hdmatx, hdma_tx);

/* Configure the DMA handler for reception process */
hdma_rx.Instance                 = USARTx_RX_DMA_CHANNEL;
hdma_rx.Init.Direction           = DMA_PERIPH_TO_MEMORY;
hdma_rx.Init.PeriphInc           = DMA_PINC_DISABLE;
hdma_rx.Init.MemInc              = DMA_MINC_ENABLE;
hdma_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
hdma_rx.Init.MemDataAlignment    = DMA_MDATAALIGN_BYTE;
hdma_rx.Init.Mode                = DMA_NORMAL;
hdma_rx.Init.Priority            = DMA_PRIORITY_HIGH;
hdma_rx.Init.Request             = USARTx_RX_DMA_REQUEST;
```

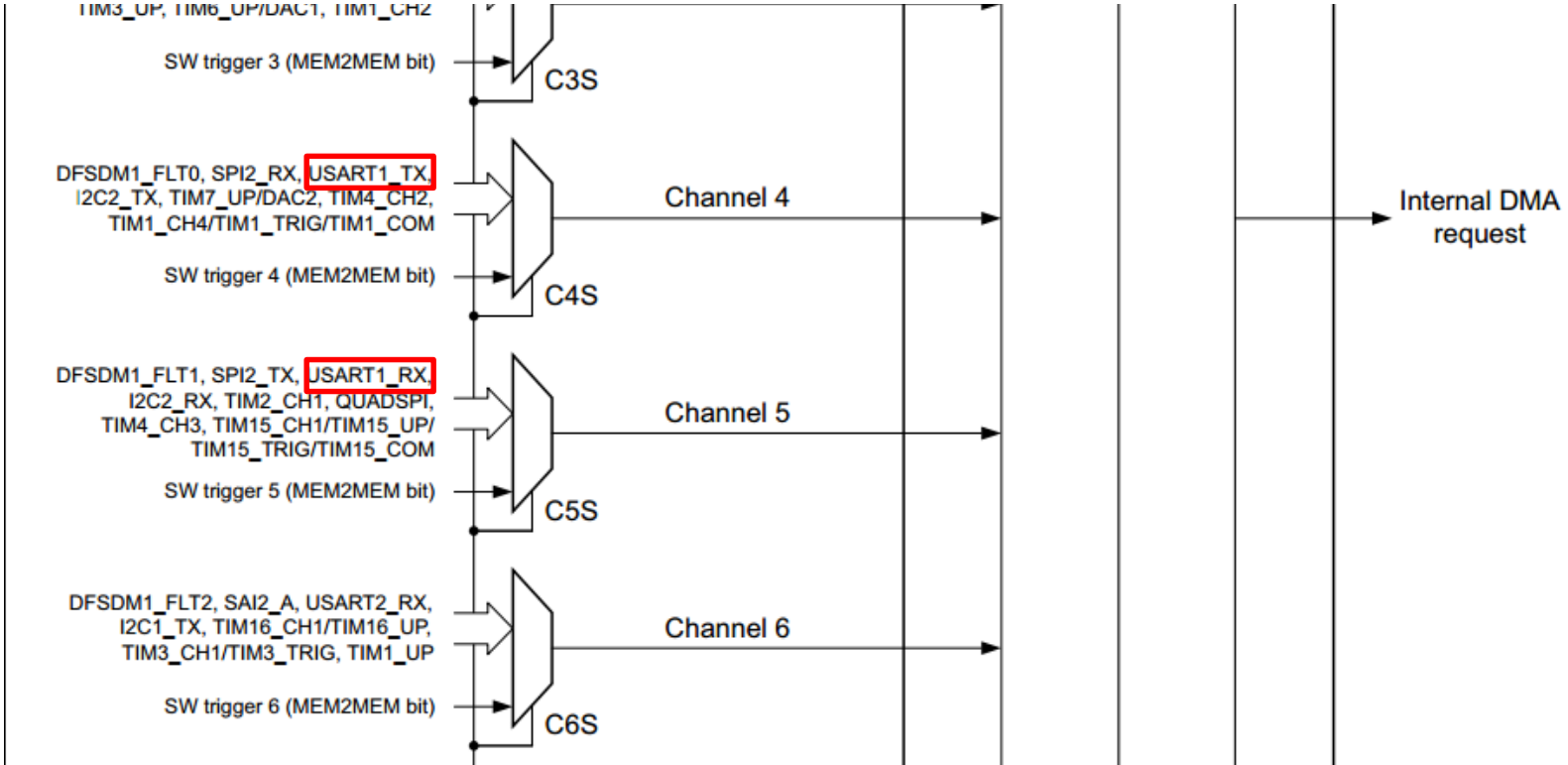
```
/**
  * @brief  DMA handle Structure definition
  */
typedef struct __DMA_HandleTypeDef
{
  DMA_Channel_TypeDef    *Instance;                                    /*!< Register base address                  */

  DMA_InitTypeDef        Init;                                        /*!< DMA communication parameters           */

  HAL_LockTypeDef        Lock;                                        /*!< DMA locking object                     */

  __IO HAL_DMA_StateTypeDef  State;                                   /*!< DMA transfer state                     */

  void                   *Parent;                                     /*!< Parent object state                    */

  void                   (* XferCpltCallback)(struct __DMA_HandleTypeDef * hdma);     /*!< DMA transfer complete callback  */

  void                   (* XferHalfCpltCallback)(struct __DMA_HandleTypeDef * hdma); /*!< DMA Half transfer complete callback */

  void                   (* XferErrorCallback)(struct __DMA_HandleTypeDef * hdma);    /*!< DMA transfer error callback     */

  __IO uint32_t          ErrorCode;                                   /*!< DMA Error code                         */
}DMA_HandleTypeDef;
```

Figure 30. DMA1 request mapping



n.c | stm32l4xx_hal_uart.c | stm32l4xx_hal_dma.c | stm32l4xx_it.c | main.h | stm32l4xx_hal_msp.c | stm32l4xx_hal_dma.h

```
7
8  /* Definition for USARTx's DMA */
9  #define USARTx_TX_DMA_CHANNEL             DMA1_Channel4
0  #define USARTx_RX_DMA_CHANNEL             DMA1_Channel5
1
2  /* Definition for USARTx's DMA Request */
3  #define USARTx_TX_DMA_REQUEST             DMA_REQUEST_2
4  #define USARTx_RX_DMA_REQUEST             DMA_REQUEST_2
5
6  /* Definition for USARTx's NVIC */
7  #define USARTx_DMA_TX_IRQn                DMA1_Channel4_IRQn
8  #define USARTx_DMA_RX_IRQn                DMA1_Channel5_IRQn
9  #define USARTx_DMA_TX_IRQHandler          DMA1_Channel4_IRQHandler
0  #define USARTx_DMA_RX_IRQHandler          DMA1_Channel5_IRQHandler
1
```



Msp init – DMA init

```
HAL_StatusTypeDef HAL_UART_Init UART_Handl
{
  /* Check the UART handle allocation */
  if(huart == NULL)
  {
    return HAL_ERROR;
  }

  if(huart->Init.HwFlowCtl != UART_HWCONTI
  {
    /* Check the parameters */
    assert_param(IS_UART_HWFLOW_INSTANCE(
  }
  else
  {
    /* Check the parameters */
    assert_param((IS_UART_INSTANCE(huart-
  }

  if(huart->State == HAL_UART_STATE_RESET
  {
    /* Allocate lock resource and initial
    huart->Lock = HAL_UNLOCKED;

    /* Init the low level hardware : GPIO,
    HAL_UART_MspInit(huart);
  }
```

```
/*##-3- Configure the DMA ##################################################
/* Configure the DMA handler for Transmission process */
hdma_tx.Instance                 = USARTx_TX_DMA_CHANNEL;
hdma_tx.Init.Direction           = DMA_MEMORY_TO_PERIPH;
hdma_tx.Init.PeriphInc           = DMA_PINC_DISABLE;
hdma_tx.Init.MemInc              = DMA_MINC_ENABLE;
hdma_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
hdma_tx.Init.MemDataAlignment    = DMA_MDATAALIGN_BYTE;
hdma_tx.Init.Mode                = DMA_NORMAL;
hdma_tx.Init.Priority            = DMA_PRIORITY_LOW;
hdma_tx.Init.Request             = USARTx_TX_DMA_REQUEST;

HAL_DMA_Init(&hdma_tx);

/* Associate the initialized DMA handle to the UART handle */
__HAL_LINKDMA(huart, hdmatx, hdma_tx);

/* Configure the DMA handler for reception process */
hdma_rx.Instance                 = USARTx_RX_DMA_CHANNEL;
hdma_rx.Init.Direction           = DMA_PERIPH_TO_MEMORY;
hdma_rx.Init.PeriphInc           = DMA_PINC_DISABLE;
hdma_rx.Init.MemInc              = DMA_MINC_ENABLE;
hdma_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
hdma_rx.Init.MemDataAlignment    = DMA_MDATAALIGN_BYTE;
hdma_rx.Init.Mode                = DMA_NORMAL;
hdma_rx.Init.Priority            = DMA_PRIORITY_HIGH;
hdma_rx.Init.Request             = USARTx_RX_DMA_REQUEST;

HAL_DMA_Init(&hdma_rx);

/* Associate the initialized DMA handle to the the UART handle */
__HAL_LINKDMA(huart, hdmarx, hdma_rx);

/*##-4- Configure the NVIC for DMA #########################################
/* NVIC configuration for DMA transfer complete interrupt (USART1_TX) */
HAL_NVIC_SetPriority(USARTx_DMA_TX_IRQn, 0, 1);
HAL_NVIC_EnableIRQ(USARTx_DMA_TX_IRQn);

/* NVIC configuration for DMA transfer complete interrupt (USART1_RX) */
HAL_NVIC_SetPriority(USARTx_DMA_RX_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(USARTx_DMA_RX_IRQn);

/* NVIC for USART, to catch the TX complete */
HAL_NVIC_SetPriority(USARTx_IRQn, 0, 1);
HAL_NVIC_EnableIRQ(USARTx_IRQn);
```

Šta se posle dešava?

- DMA čeka da se pojavi podatak u RX data registru
- Kada se pojavi prebacuje ga u memoriju
- Do kada se ovo ponavlja?

Kada je kraj transfera?

- Pri pozivu funkcije za transmit i receive korisnik definiše koje je veličine poruka.
- DMA na onovu te informacije zna kada će se pojaviti prekidi HalfTransfer complete, Transfer complete.
- Šta se dešava kada se pojavi prekid za kraj transfera?

DMA IRQ handler

main.c | stm32l4xx_hal_uart.c | stm32l4xx_hal_dma.c | stm32l4xx_it.c | main.h

```
73  #define USARTx_TX_DMA_REQUEST                DMA_REQUEST_2
74  #define USARTx_RX_DMA_REQUEST                DMA_REQUEST_2
75
76  /* Definition for USARTx's NVIC */
77  #define USARTx_DMA_TX_IRQn                   DMA1_Channel4_IRQn
78  #define USARTx_DMA_RX_IRQn                   DMA1_Channel5_IRQn
79  #define USARTx_DMA_TX_IRQHandler             DMA1_Channel4_IRQHandler
80  #define USARTx_DMA_RX_IRQHandler             DMA1_Channel5_IRQHandler
```

main.c | stm32l4xx_hal_uart.c | stm32l4xx_hal_dma.c | stm32l4xx_it.c | main.h

```
173   * @Note   This function is redefined in "ma
174   *         used for USART data transmission
175   */
176 void USARTx_DMA_RX_IRQHandler(void)
177 {
178   HAL_DMA_IRQHandler(UartHandle.hdmarx);
179 }
180
```

DMA IRQ handler

main.c | stm32l4xx_hal_uart.c | stm32l4xx_hal_dma.c | stm32l4xx_it.c | main.h

```
73  #define USARTx_TX_DMA_REQUEST                DMA_REQUEST_2
74  #define USARTx_RX_DMA_REQUEST                DMA_REQUEST_2
75
76  /* Definition for USARTx
77  #define USARTx_DMA_TX_IR
78  #define USARTx_DMA_RX_IR
79  #define USARTx_DMA_TX_IR
80  #define USARTx_DMA_RX_IR
```

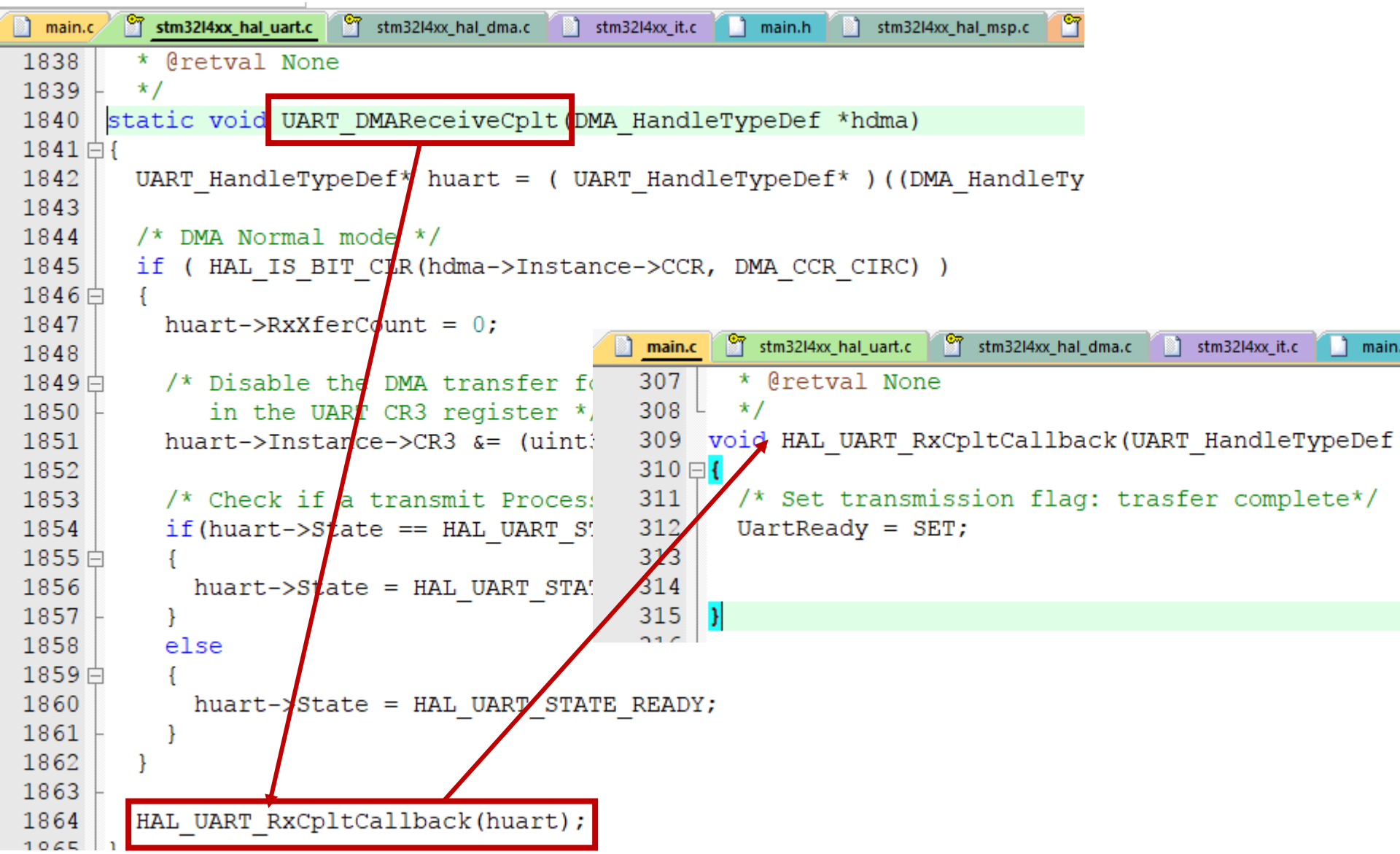
main.c | stm32l4xx_hal_uart.c | stm32l4xx_hal_dma.c

```
173   * @Note   This function is
174   *         used for USART da
175   */
176 void USARTx_DMA_RX_IRQHandler
177 {
178   HAL_DMA_IRQHandler(UartHand
179 }
180
```

stm32l4xx_hal_uart.c | stm32l4xx_hal_dma.c | stm32l4xx_it.c | main.h | stm32l4

```
if(__HAL_DMA_GET_FLAG(hdma, __HAL_DMA_GET_TC_FLAG_INDEX
{
  if(__HAL_DMA_GET_IT_SOURCE(hdma, DMA_IT_TC) != RESET)
  {
    if((hdma->Instance->CCR & DMA_CCR_CIRC) == 0)
    {
      /* Disable the transfer complete interrupt */
      __HAL_DMA_DISABLE_IT(hdma, DMA_IT_TC);
    }
    /* Clear the transfer complete flag */
    __HAL_DMA_CLEAR_FLAG(hdma, __HAL_DMA_GET_TC_FLAG_IN

    /* Update error code */
    hdma->ErrorCode |= HAL_DMA_ERROR_NONE;

    /* Change the DMA state */
    hdma->State = HAL_DMA_STATE_READY;

    /* Process Unlocked */
    __HAL_UNLOCK(hdma);

    if(hdma->XferCpltCallback != NULL)
    {
      /* Transfer complete callback */
      hdma->XferCpltCallback(hdma);
```

DMA završio, UART stupa na scenu

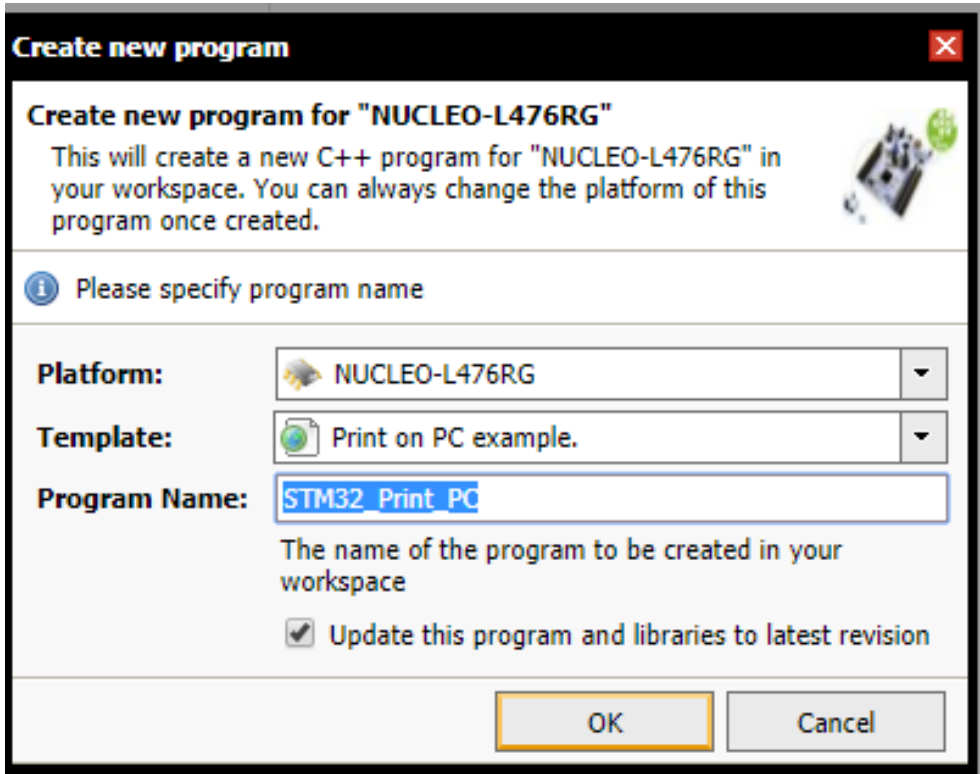


Serijski portovi – klasa Serial

http://mbed.org/handbook/SerialPC

- Klasa Serial služi za korišćenje bilo kog serijskog porta uključujući i onog koji se koristi za vezu preko USB prema PC-ju. U tom slučaju jedan od UART-a je zauzet.

```
#include "mbed.h"
Serial pc(USBTX, USBRX); // tx, rx
int main() {
        pc.printf("Hello World!\n");
}
```



Klasa Serial

Serial Class Reference

`#include <Serial.h>`

Inherits mbed::SerialBase, and mbed::Stream.

Public Member Functions

Serial (PinName tx, PinName rx, const char *name=NULL)
Create a **Serial** port, connected to the specified transmit and receive pins.

void baud (int baudrate)
Set the baud rate of the serial port.

void format (int bits=8, Parity parity=SerialBase::None, int stop_bits=1)
Set the transmission format used by the serial port.

int readable ()
Determine if there is a character available to read.

int writeable ()
Determine if there is space available to write a character.

void attach (void(*fptr)(void), IrqType type=RxIrq)
Attach a function to call whenever a serial interrupt is generated.

template<typename T >
void attach (T *tptr, void(T::*mptr)(void), IrqType type=RxIrq)
Attach a member function to call whenever a serial interrupt is generated.

void send_break ()
Generate a break condition on the serial line.

void set_flow_control (Flow type, PinName flow1=NC, PinName flow2=NC)
Set the flow control type on the serial port.

Nedavno dodate funkcije

int write (const uint8_t *buffer, int length, const event_callback_t &callback, int event=SERIAL_EVENT_TX_COMPL

Begin asynchronous write using 8bit buffer.

int write (const uint16_t *buffer, int length, const event_callback_t &callback, int event=SERIAL_EVENT_TX_COMP

Begin asynchronous write using 16bit buffer.

void abort_write ()

Abort the on-going write transfer.

int read (uint8_t *buffer, int length, const event_callback_t &callback, int event=SERIAL_EVENT_RX_COMPLETE, u

Begin asynchronous reading using 8bit buffer.

int read (uint16_t *buffer, int length, const event_callback_t &callback, int event=SERIAL_EVENT_RX_COMPLETE,

Begin asynchronous reading using 16bit buffer.

void abort_read ()

Abort the on-going read transfer.

int set_dma_usage_tx (DMAUsage usage)

Configure DMA usage suggestion for non-blocking TX transfers.

int set_dma_usage_rx (DMAUsage usage)

Configure DMA usage suggestion for non-blocking RX transfers.

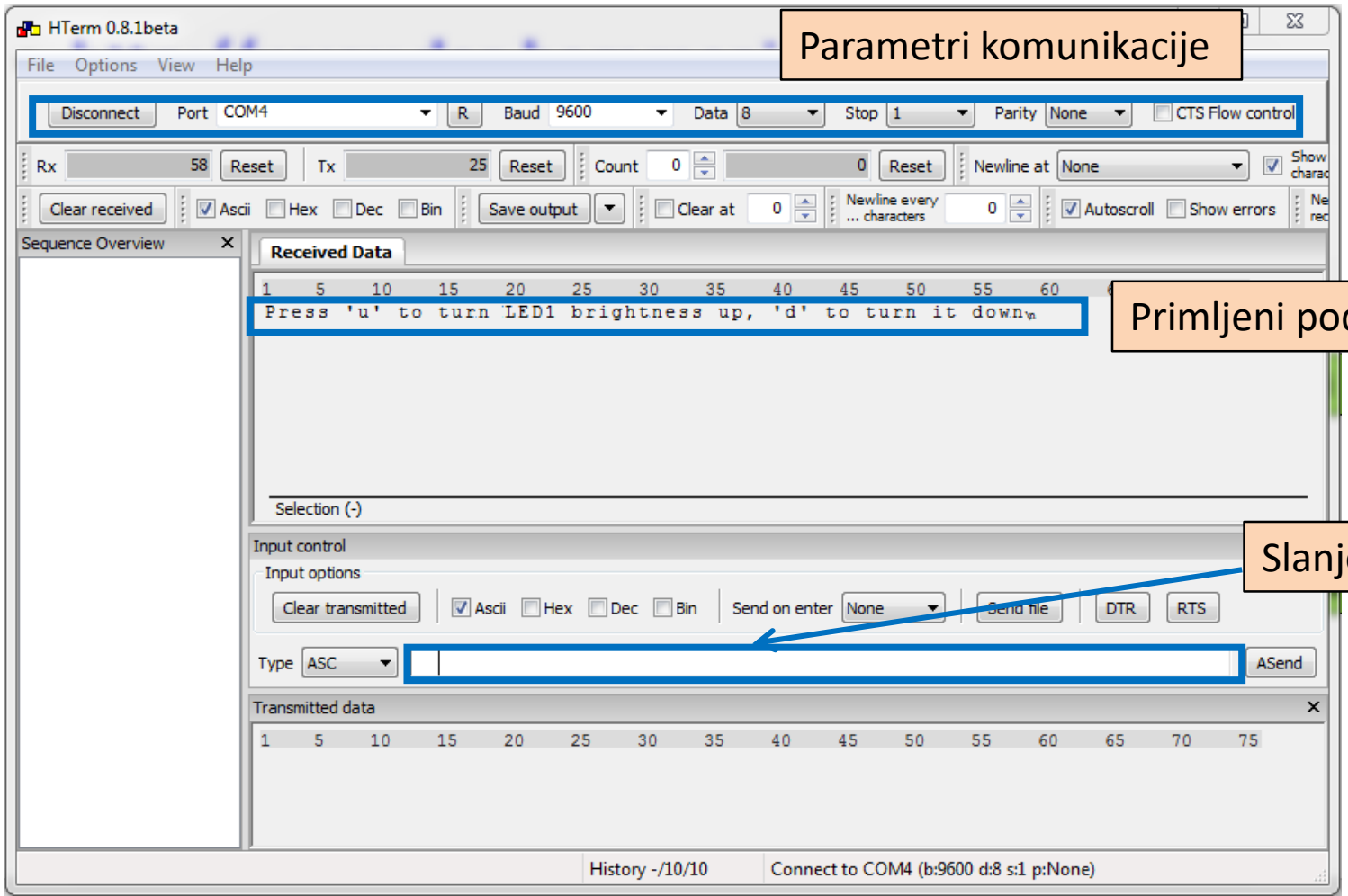
Serijski portovi

- Osim funkcije printf() koja se standardno koristi za prenos preko serijskog porta moguće je koristiti i funkcije koje su orjentisane ka prenosu karaktera.

```
#include "mbed.h"
Serial pc(SERIAL_TX, SERIAL_RX); // tx, rx
PwmOut led(LED1);
float brightness = 0.0f;
int main() {
        pc.printf("Press 'u' to turn LED1 brightness up, 'd' to turn it down\n");
        while(1) {
                char c = pc.getc();
                if((c == 'u') && (brightness < 0.7f)) {
                        brightness += 0.05f; led = brightness;
                }
                if((c == 'd') && (brightness > 0.0f)) {
                        brightness -= 0.05f; led = brightness;
                }
        }
}
```

Serijski protovi – testiranje korišćenjem HTerm-a

- http://www.der-hammer.info/terminal/



Korišćenje prekida

http://mbed.org/cookbook/Serial-Interrupts

- Osnovna mana funkcija kao što su printf() i scanf() je to što se program praktično zaglavljuje u tim funkcijama sve dok traje prenos podataka.
- Jedan od načina obezbeđivanja “pozadinske obrade” je korišćenje prekida.
- Klasa Serial dozvoljava povezivanje korisničkih funkcija za dva tipa prekida: Rx i Tx.

- Primer:

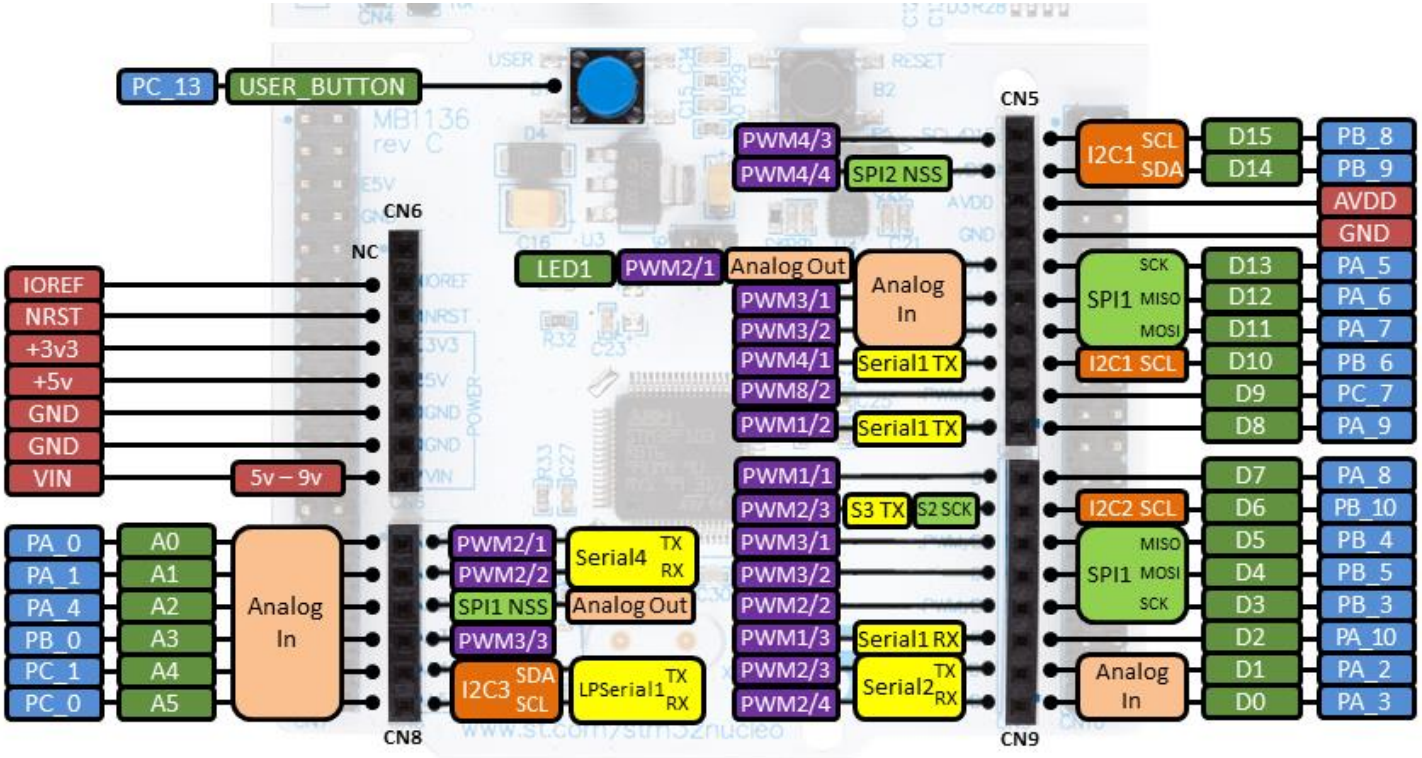
```
pc.attach(&UserInterrupt,Serial::RxIrq);
```

Zadatak Serial

- Napisati program koji obezbeđuje kontrolu osvetljaja diode sa tastature.
  - ‘u’- povećati osvetljaj za 10%
  - ‘d’- smanjiti osvetljaj za 10%
  - Kada se dođe do osvetljaja 0% ne ide se niže, kada se dođe do osvetljaja 100% ne inkrementira se više.
  - Primljena poruka preko serijske veze treba da generiše prekid u kome se analizira poslati karakter i menja osvetljaj diode
  - Kao kontrolu u prekidnoj rutini obezbediti slanje ack signala nazad računaru u vidu karaktera ‘k’

Zadatak PWM

- Implementirati PWMOutput i PWMInput funkcionalnosti na MBED platformi.
- PWMOutput je izuzetno lako implementirati.
- PWMInput nema direktnu podršku...



Zadatak PWM

- Ideja za PWMInput:
  1. Uvede se jedan Ticker tajmer koji radi nekom velikom brzinom i inkrementira neku promenljivu *time*.
  2. Ulazni signal se poveže na neka dva InterruptIn ulaza koji generišu prekide na rastuću i opadajuću ivicu.
  3. U prekidima se očitava promenljiva *time* i na osnovu sukcesivnih vrednosti se računaju parametri.
- Vratiti preko UART-a informaciju o učestanosti i ispunjenosti (duty ratio)