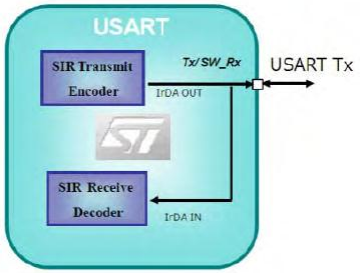
PRIMENA MIKROKONTROLERA-MS1PMK

7. deo

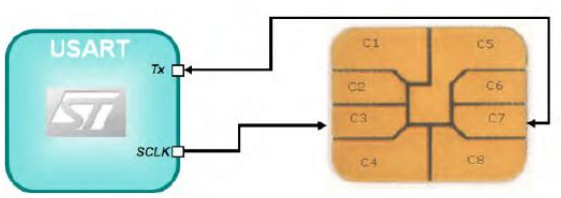
2017

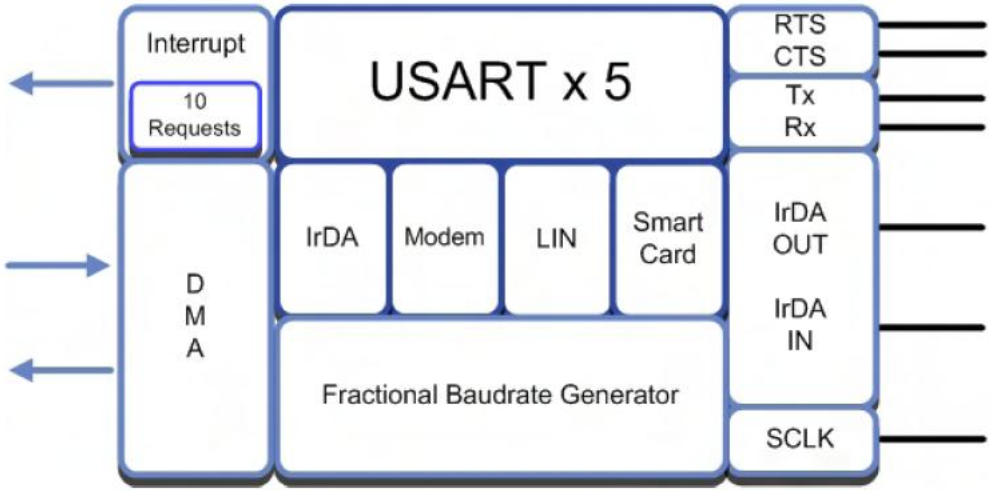
Nenad Jovičić

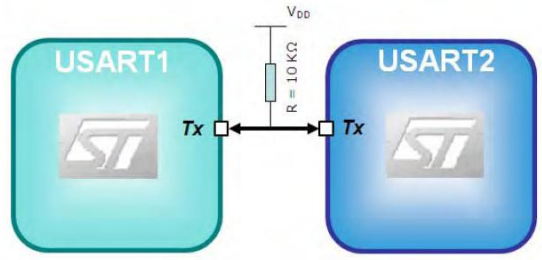
USART



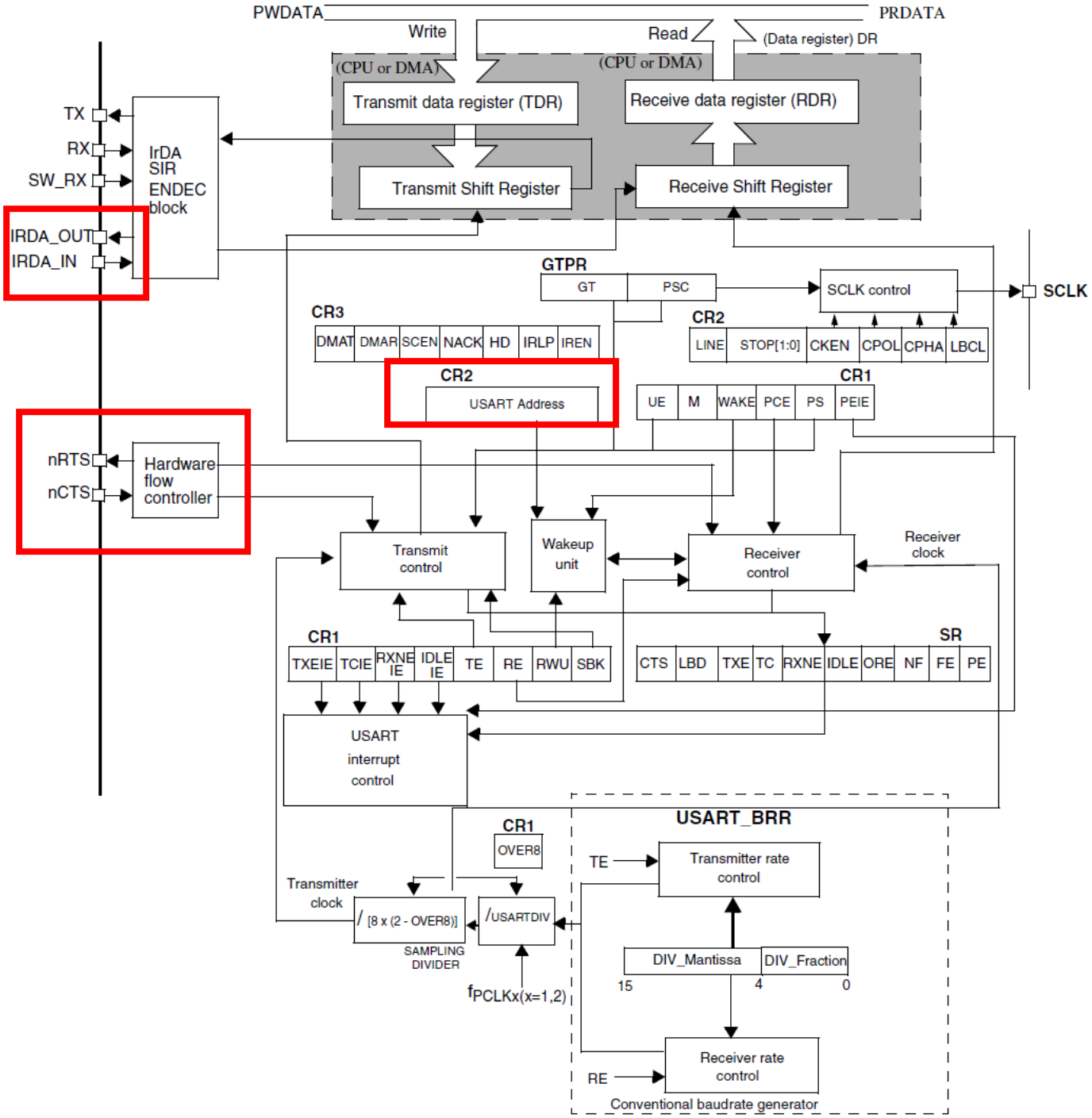
- Posebne karakteristike u odnosu na standardni USART:
  - 8 ili 16 bita oversampling
  - Frakcioni baud-rate generator (slično kao MSP)
  - 7, 8 ili 9 bita podatak
  - Encoder decoder za podršku IRDA prenosu
  - Podržan smartcard protocol ISO7816-3
  - Half-duplex komunikacija preko jedne žice
  - Dva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





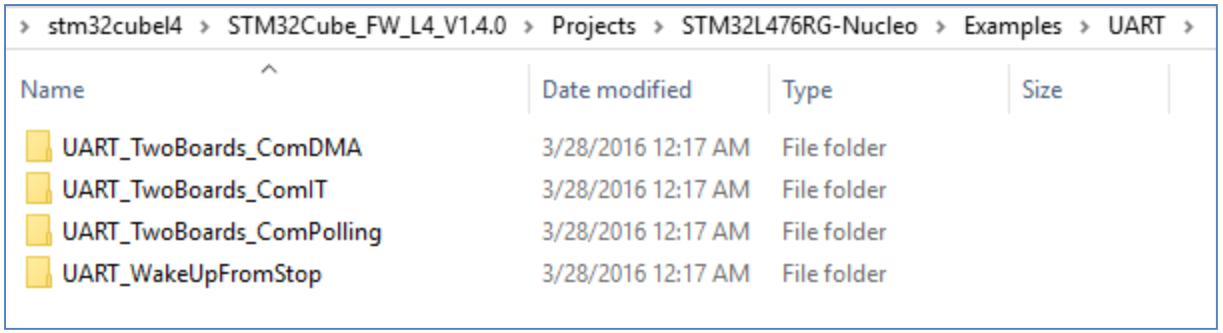


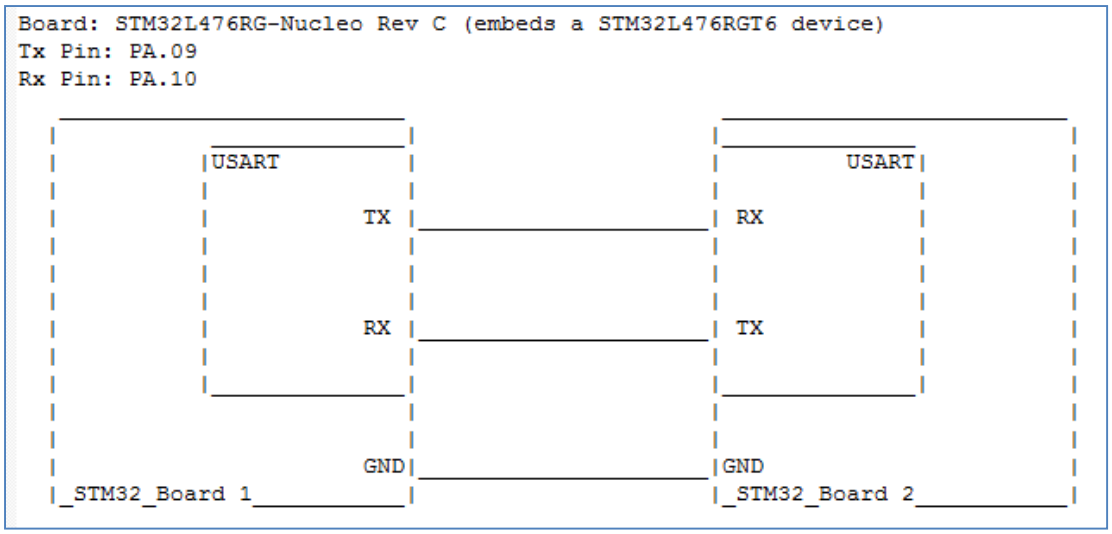
USART
blok šema



STM CUBE
UART Projekti

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





STM CUBE

Projekat

UART_TwoBoards_ComPolling

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
```

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...

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 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

```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...

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
98  /**
99    * @brief UART MSP De-Initialization
100   *        This function frees the hardware resources used in this example:
101   *          - Disable the Peripheral's clock
102   *          - Revert GPIO configuration to their default state
103   * @param huart: UART handle pointer
104   * @retval None
105   */
106 void HAL_UART_MspDeInit(UART_HandleTypeDef *huart)
107 {
108   /*##-1- Reset peripherals ##################################################*/
109   USARTx_FORCE_RESET();
110   USARTx_RELEASE_RESET();
```

Ovo nikada ne treba izgubiti iz vid-a.

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...

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

```
/*##-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();
}
```

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.

Tu su negde i prekidi

stm32l4xx_it.c | readme.txt | main.c | stm32l4xx_hal_gpio.c | stm32l4xx.h | stm32_hal_legacy.h

```
163  /******************************************************************************/
164  /*                 STM32L4xx Peripherals Interrupt Handlers                   */
165  /*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
166  /*  available peripheral interrupt handler's name please refer to the startup */
167  /*  file (startup_stm32l4xx.s).                                               */
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  }
```

Filozofija HAL drajvera i njihovih kreatora: Prekidi su toliko komplikovani da je bolje da pustite “NAS” da se brinemo oko flag-ova a vaše je da implementirate samo određene Callback funkcije

I naravno Callback funkcije

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
```

A gde smo to mi dozvolili UART prekide??

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
    /*@} end of group Cortex_M4 */
```

I naravno Callback funkcije

main.c | main.h | stm32l4xx_hal_msp.c

```
66   * @param huart: UART handle pointer
67   * @retval None
68   */
69  void HAL_UART_MspInit(UART_HandleTypeDef *huart)
70  {
71    GPIO_InitTypeDef  GPIO_InitStruct;
72
73    /*##-1- Enable peripherals and GPIO Clocks #########################
74    /* Enable GPIO TX/RX clock */
75    USARTx_TX_GPIO_CLK_ENABLE();
76    USARTx_RX_GPIO_CLK_ENABLE();
77
78
79    /* Enable USARTx clock */
80    USARTx_CLK_ENABLE();
81
82    /*##-2- Configure peripheral GPIO ##################################
83    /* UART TX GPIO pin configuration  */
84    GPIO_InitStruct.Pin       = USARTx_TX_PIN;
85    GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
86    GPIO_InitStruct.Pull      = GPIO_PULLUP;
87    GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_VERY_HIGH;
88    GPIO_InitStruct.Alternate = USARTx_TX_AF;
89
90    HAL_GPIO_Init(USARTx_TX_GPIO_PORT, &GPIO_InitStruct);
91
92    /* UART RX GPIO pin configuration  */
93    GPIO_InitStruct.Pin = USARTx_RX_PIN;
94    GPIO_InitStruct.Alternate = USARTx_RX_AF;
95
96    HAL_GPIO_Init(USARTx_RX_GPIO_PORT, &GPIO_InitStruct);
97
98    /*##-3- Configure the NVIC for UART ################################
99    /* NVIC for USART */
100   HAL_NVIC_SetPriority(USARTx_IRQn, 0, 1);
101   HAL_NVIC_EnableIRQ(USARTx_IRQn);
102 }
103
104 /**
324 }
325
326
```

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
/*@} end of group Cortex_M4 */
```

A gde smo to mi dozvolili UART prekide??

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...?

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.

STM CUBE

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

```
955   * @retval HAL status
956   */
957  HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)
958  {
959    uint32_t *tmp;
960
961    if((huart->State == HAL_UART_STATE_READY) || (huart->State == HAL_UART_STATE_BUSY_TX))
962    {
963      if((pData == NULL ) || (Size == 0))
964      {
965        return HAL_ERROR;
966      }
967
968      /* Process Locked */
969      __HAL_LOCK(huart);
970
971      huart->pRxBuffPtr = pData;
972      huart->RxXferSize = Size;
973
974      huart->ErrorCode = HAL_UART_ERROR_NONE;
975      /* Check if a transmit process is ongoing or not */
976      if(huart->State == HAL_UART_STATE_BUSY_TX)
977      {
978        huart->State = HAL_UART_STATE_BUSY_TX_RX;
979      }
980      else
981      {
982        huart->State = HAL_UART_STATE_BUSY_RX;
983      }
984
985      /* Set the UART DMA transfer complete callback */
986      huart->hdmarx->XferCpltCallback = UART_DMAReceiveCplt;
987
988      /* Set the UART DMA Half transfer complete callback */
989      huart->hdmarx->XferHalfCpltCallback = UART_DMARxHalfCplt;
990
991      /* Set the DMA error callback */
992      huart->hdmarx->XferErrorCallback = UART_DMAError;
993
994      /* Enable the DMA channel */
995      tmp = (uint32_t*)&pData;
996      HAL_DMA_Start_IT(huart->hdmarx, (uint32_t)&huart->Instance->RDR, *(uint32_t*)tmp, Size);
997
```

DMA

UART ima pridruzenu DMA instancu, gde se to obavlja?

stm32l4xx_hal_msp.c | main.c

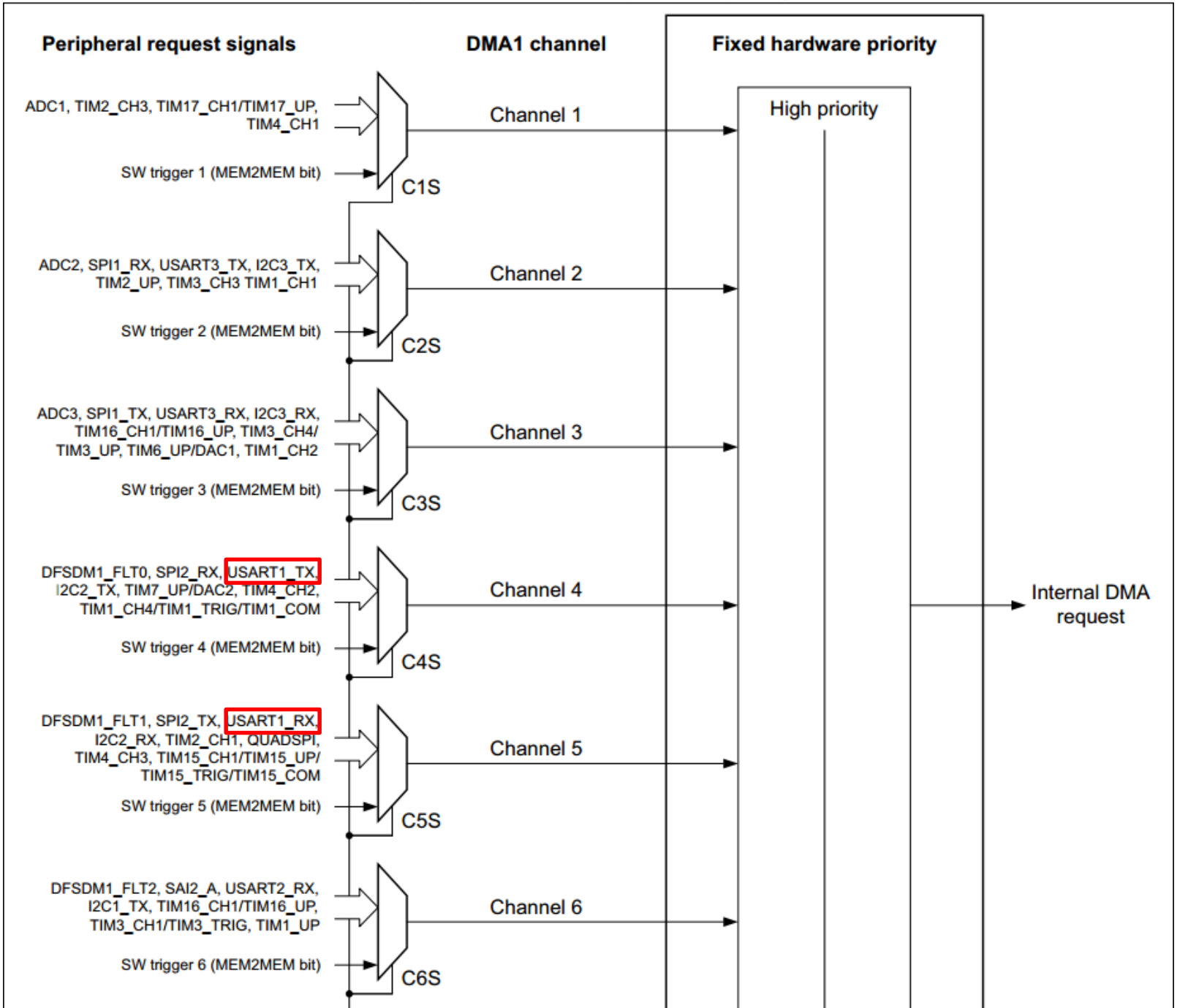
```
70  void HAL_UART_MspInit(UART_HandleTypeDef *huart)
71  {
72    static DMA_HandleTypeDef hdma_tx;
73    static DMA_HandleTypeDef hdma_rx;
74
75    GPIO_InitTypeDef  GPIO_InitStruct;
76
77    /*##-1- Enable peripherals and GPIO Clocks #####
78    /* Enable GPIO TX/RX clock */
79    USARTx_TX_GPIO_CLK_ENABLE();
80    USARTx_RX_GPIO_CLK_ENABLE();
81
82
83    /* Enable USARTx clock */
84    USARTx_CLK_ENABLE();
85
86    /* Enable DMA clock */
87    DMAx_CLK_ENABLE();
88
89    /*##-2- Configure peripheral GPIO ##############
90    /* UART TX GPIO pin configuration  */
91    GPIO_InitStruct.Pin       = USARTx_TX_PIN;
92    GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
93    GPIO_InitStruct.Pull      = GPIO_PULLUP;
94    GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_VERY_HIGH;
95    GPIO_InitStruct.Alternate = USARTx_TX_AF;
96
97    HAL_GPIO_Init(USARTx_TX_GPIO_PORT, &GPIO_InitStruct);
98
99    /* UART RX GPIO pin configuration  */
00    GPIO_InitStruct.Pin = USARTx_RX_PIN;
01    GPIO_InitStruct.Alternate = USARTx_RX_AF;
02
03    HAL_GPIO_Init(USARTx_RX_GPIO_PORT, &GPIO_InitStruct);
04
05    /*##-3- Configure the DMA ##########################################################*/
06    /* Configure the DMA handler for Transmission process */
07    hdma_tx.Instance                 = USARTx_TX_DMA_CHANNEL
08    hdma_tx.Init.Direction           = DMA_MEMORY_TO_PERIPH
09    hdma_tx.Init.PeriphInc           = DMA_PINC_DISABLE;
10    hdma_tx.Init.MemInc              = DMA_MINC_ENABLE;
11    hdma_tx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
12    hdma_tx.Init.MemDataAlignment    = DMA_MDATAALIGN_BYTE;
13    hdma_tx.Init.Mode                = DMA_NORMAL;
14    hdma_tx.Init.Priority            = DMA_PRIORITY_LOW;
```

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

```
65  #define USARTx_RX_GPIO_PORT                GPIOA
66  #define USARTx_RX_AF                       GPIO_AF7_USART1
67
68  /* Definition for USARTx's DMA */
69  #define USARTx_TX_DMA_CHANNEL               DMA1_Channel4
70  #define USARTx_RX_DMA_CHANNEL               DMA1_Channel5
71
72  /* Definition for USARTx's DMA Request */
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
```

DMA kontroler se podešava u MspInit funkciji

Figure 30. DMA1 request mapping



stm32l4xx_it.c | stm32l4xx_hal_msp.c | main.c

```
121
122   /* Configure the DMA handler for reception process */
123   hdma_rx.Instance                 = USARTx_RX_DMA_CHANNEL;
124   hdma_rx.Init.Direction           = DMA_PERIPH_TO_MEMORY;
125   hdma_rx.Init.PeriphInc           = DMA_PINC_DISABLE;
126   hdma_rx.Init.MemInc              = DMA_MINC_ENABLE;
127   hdma_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
128   hdma_rx.Init.MemDataAlignment    = DMA_MDATAALIGN_BYTE;
129   hdma_rx.Init.Mode                = DMA_NORMAL;
130   hdma_rx.Init.Priority            = DMA_PRIORITY_HIGH;
131   hdma_rx.Init.Request             = USARTx_RX_DMA_REQUEST;
132
133   HAL_DMA_Init(&hdma_rx);
134
135   /* Associate the initialized DMA handle to the the UART handle */
136   __HAL_LINKDMA(huart, hdmarx, hdma_rx);
137
138   /*##-4- Configure the NVIC for DMA ###########################################*
139   /* NVIC configuration for DMA transfer complete interrupt (USART1_TX) */
140   HAL_NVIC_SetPriority(USARTx_DMA_TX_IRQn, 0, 1);
141   HAL_NVIC_EnableIRQ(USARTx_DMA_TX_IRQn);
142
143   /* NVIC configuration for DMA transfer complete interrupt (
144   HAL_NVIC_SetPriority(USARTx_DMA_RX_IRQn, 0, 0);
145   HAL_NVIC_EnableIRQ(USARTx_DMA_RX_IRQn);
146
147   /* NVIC for USART, to catch the TX complete */
148   HAL_NVIC_SetPriority(USARTx_IRQn, 0, 1);
149   HAL_NVIC_EnableIRQ(USARTx_IRQn);
150 }
```

stm32l4xx_it.c | stm32l4xx_hal_msp.c | main.c

```
169 /**
170   * @brief  This function handles DMA inte
171   * @param  None
172   * @retval None
173   * @Note   This function is redefined in
174   *         used for USART data transmissi
175   */
176 void USARTx_DMA_RX_IRQHandler(void)
177 {
178   HAL_DMA_IRQHandler(UartHandle.hdmarx);
179 }
180
181 /**
182   * @brief  This function handles DMA inte
183   * @param  None
184   * @retval None
185   * @Note   This function is redefined in
186   *         used for USART data reception
187   */
188 void USARTx_DMA_TX_IRQHandler(void)
189 {
190   HAL_DMA_IRQHandler(UartHandle.hdmatx);
```

stm32l4xx_it.c | stm32l4xx_hal_msp.c | main.c

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

Dozvola DMA prekida

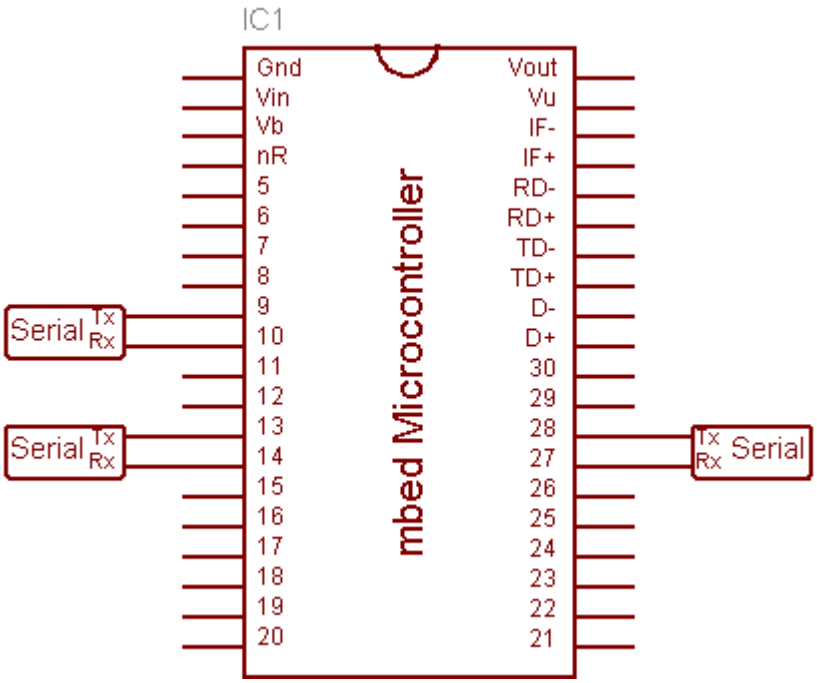
Callback funkcije se isto zovu kao i u projektu sa prekidima???

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_COMPLETE)
Begin asynchronous write using 8bit buffer.

int write (const uint16_t *buffer, int length, const event_callback_t &callback, int event=SERIAL_EVENT_TX_COMPLETE)
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, unsigned char char_match=SERIAL_RESERVED_CHAR_MATCH)
Begin asynchronous reading using 8bit buffer.

int read (uint16_t *buffer, int length, const event_callback_t &callback, int event=SERIAL_EVENT_RX_COMPLETE, unsigned char char_match=SERIAL_RESERVED_CHAR_MATCH)
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.

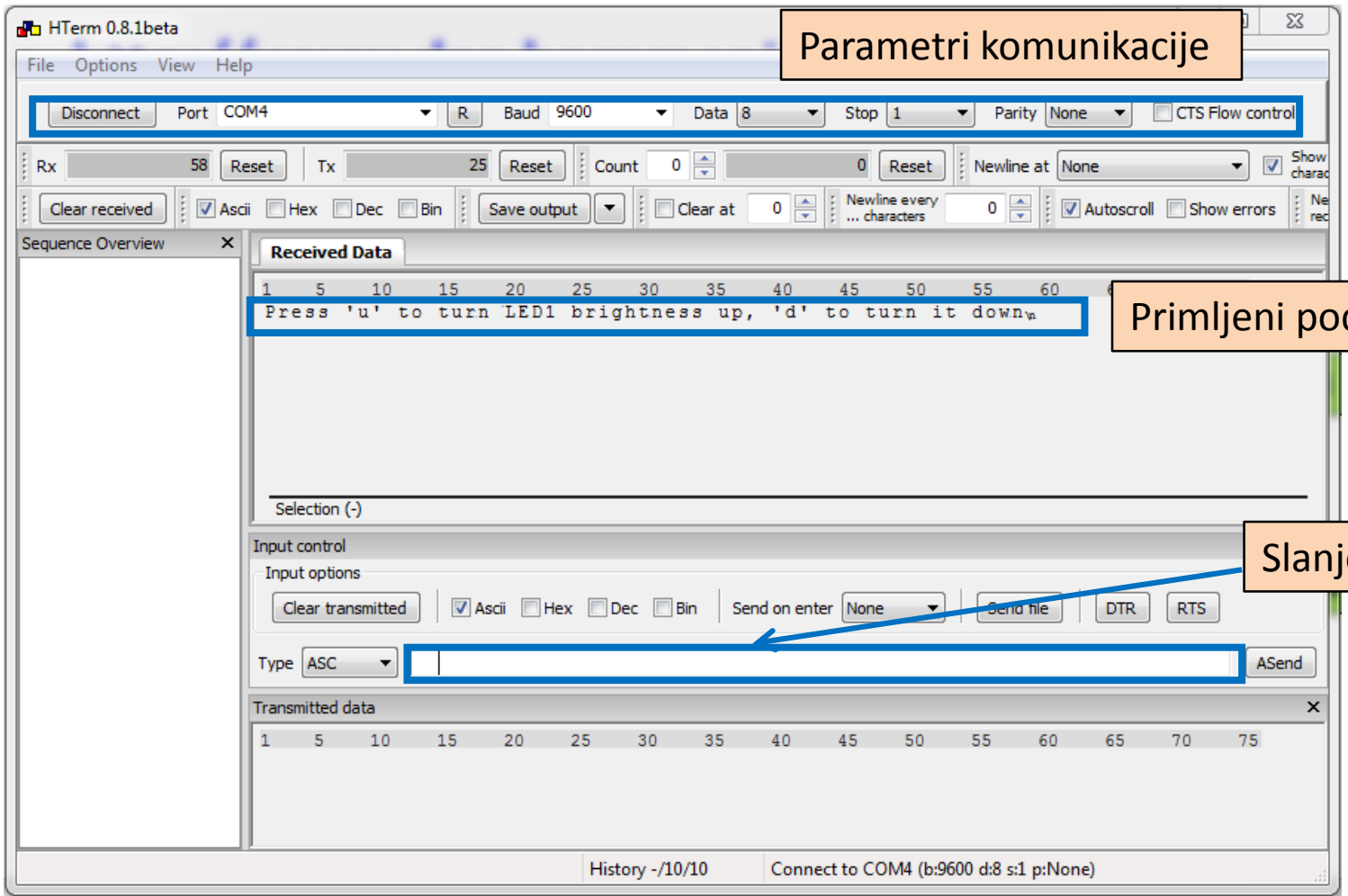
Serijski portovi

- 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.

```
#include "mbed.h"
Serial pc(SERIAL_TX, SERIAL_TX);
PwmOut led(LED1);
void UserInterrupt (void){
    led=(float)pc.getc()/255;
};
int main() {
    pc.attach(&UserInterrupt, Serial::RxIrq);
    while (1);//glavni program ne radi nista, tj. slobodan je za NESTO ☺
}
```