ДОДАТКИ

Додаток А

(Обов’язковий)

ПРОГРАМНИЙ КОД ПРОГРАМНО-АПАРАТНИЙ КОМУТАТОР ІНТЕРФЕЙСІВ ВБУДОВАНИХ СИСТЕМ

Лістинг 1. main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"
#include "dma.h"
#include "lwip.h"
#include "spi.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void MX_FREERTOS_Init(void);
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

	SCB_EnableICache();
	SCB_EnableDCache();
	HAL_Init();

	SystemClock_Config();

	/* Initialize all configured peripherals */
	MX_GPIO_Init();
	MX_DMA_Init();
	MX_SPI2_Init();
	MX_SPI3_Init();
	MX_UART4_Init();
	MX_SPI5_Init();
	MX_SPI4_Init();
	MX_UART5_Init();
	MX_UART7_Init();
	MX_UART8_Init();
	MX_USART1_UART_Init();
	MX_USART2_Init();
	MX_USART3_Init();
	MX_USART6_Init();
	MX_SPI1_Init();
	MX_FREERTOS_Init();

	osKernelStart();

	while (1);
}

/**
```

```
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
	LL_FLASH_SetLatency(LL_FLASH_LATENCY_7);

	if(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_7)
	{
		Error_Handler();
	}
	LL_PWR_SetRegulVoltageScaling(LL_PWR_REGU_VOLTAGE_SCALE1);
	LL_PWR_EnableOverDriveMode();
	LL_RCC_HSE_EnableBypass();
	LL_RCC_HSE_Enable();

	/* Wait till HSE is ready */
	while(LL_RCC_HSE_IsReady() != 1)
	{

	}
	LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSE,
LL_RCC_PLLM_DIV_4, 216, LL_RCC_PLLP_DIV_2);
	LL_RCC_PLL_Enable();

	/* Wait till PLL is ready */
	while(LL_RCC_PLL_IsReady() != 1)
	{

	}
	LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
	LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_4);
	LL_RCC_SetAPB2Prescaler(LL_RCC_APB2_DIV_2);
	LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);

	/* Wait till System clock is ready */
	while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL)
	{

	}
	LL_Init1msTick(216000000);
	LL_SYSTICK_SetClkSource(LL_SYSTICK_CLKSOURCE_HCLK);
	LL_SetSystemCoreClock(216000000);
	LL_RCC_SetUSARTClockSource(LL_RCC_USART1_CLKSOURCE_PCLK2);
	LL_RCC_SetUSARTClockSource(LL_RCC_USART2_CLKSOURCE_PCLK1);
	LL_RCC_SetUSARTClockSource(LL_RCC_USART3_CLKSOURCE_PCLK1);
	LL_RCC_SetUSARTClockSource(LL_RCC_USART6_CLKSOURCE_PCLK2);
	LL_RCC_SetUARTClockSource(LL_RCC_UART4_CLKSOURCE_PCLK1);
	LL_RCC_SetUARTClockSource(LL_RCC_UART5_CLKSOURCE_PCLK1);
	LL_RCC_SetUARTClockSource(LL_RCC_UART7_CLKSOURCE_PCLK1);
	LL_RCC_SetUARTClockSource(LL_RCC_UART8_CLKSOURCE_PCLK1);
}
```

Лістинг 2. httpserver-netconn.c

```
/* Includes ------------------------------------------------------------------*/
#include "lwip/opt.h"
#include "lwip/arch.h"
#include "lwip/api.h"
#include "string.h"
#include "httpserver-netconn.h"
#include "cmsis_os.h"
#include "interfaces.h"
#include "convert_json.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define WEBSERVER_THREAD_PRIO    ( tskIDLE_PRIORITY + 4 )
#define TOKENS_SIZE              20

/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
u32_t nPageHits = 0;
u16_t     var1;

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/**
  * @brief serve tcp connection
  * @param conn: pointer on connection structure
  * @retval None
  */
void http_server_serve(struct netconn *conn)
{
struct         netbuf *inbuf;
  err_t          recv_err;
char*          buf;
u16_t          buflen;
requst_after_parse_type       request;

  /* Read the data from the port, blocking if nothing yet there.
  We assume the request (the part we care about) is in one netbuf */
  recv_err = netconn_recv(conn, &inbuf);

if (recv_err == ERR_OK)
  {
if (netconn_err(conn) == ERR_OK)
    {
      netbuf_data(inbuf, (void**)&buf, &buflen);
      parse_request_from_json(buf, buflen, &request);
      process_request_to_interface(&request);
    }
  }
}
```

```
/**
  * @brief  http server thread
  * @retval None
  */
static void http_server_netconn_thread()
{
struct netconn *conn, *newconn;
err_t err, accept_err;

  /* Create a new TCP connection handle */
conn = netconn_new(NETCONN_TCP);

if (conn!= NULL)
  {
    /* Bind to port 80 (HTTP) with default IP address */
err = netconn_bind(conn, NULL, 80);

if (err == ERR_OK)
    {
      /* Put the connection into LISTEN state */
      netconn_listen(conn);

while(1)
      {
        /* accept any icoming connection */
        accept_err = netconn_accept(conn, &newconn);
if(accept_err == ERR_OK)
        {
          /* serve connection */
          http_server_serve(newconn);

          /* delete connection */
          netconn_delete(newconn);
        }
      }
    }
  }
}

/**
  * @brief  Initialize the HTTP server (start its thread)
  * @param  none
  * @retval None
  */
void http_server_netconn_init()
{
   sys_thread_new("HTTP",          (lwip_thread_fn)http_server_netconn_thread,          NULL,
DEFAULT_THREAD_STACKSIZE, WEBSERVER_THREAD_PRIO);
}
```

Листинг 3. uart_task.c

```
/* Includes ------------------------------------------------------------------*/
#include "GlobalIncludes.h"

static u8 errUartTask;

void UartTask(void *pvParameters){
u16 sizeTx;


while(1){

      OSSemPend (xUartSem, 0, &errUartTask);      /* ожидаем семафор */
if (errUartTask == OS_ERR_NONE){

if(uartMsg.readyForRead == true){
sizeTx = ProcessRX((u8*)uartMsg.bufTxRx, uartMsg.numOfRecBytes);
if(sizeTx != 0){
UARTStartTX(&(uartMsg.bufTxRx[0]), sizeTx);
}
else{
                uartMsg.readyForRead = false;

                /*Отправляем семафор еще раз что бы сконфигурироваться на прием*/
OSSemPost(xUartSem);
            }
        }
else{
UARTStartRX(&(uartMsg.bufTxRx[0]));
        }
      }
   }/* end Task loop */
}
```

Листинг 4. spi_task.c

```
/* Includes ------------------------------------------------------------------*/
#include "GlobalIncludes.h"

static u8 errSpiTask;

void SpiTask(void *pvParameters){
u16 sizeTx;


while(1){

      OSSemPend (xSpiSem, 0, &errSpiTask);
if (errSpiTask == OS_ERR_NONE){

if(spiMsg.readyForRead == true){
sizeTx = ProcessRX((u8*)spiMsg.bufTxRx, spiMsg.numOfRecBytes);
if(sizeTx != 0){
SPIStartTX(&(spiMsg.bufTxRx[0]), sizeTx);
            }
else{
                spiMsg.readyForRead = false;
OSSemPost(xSpiSem);
            }
          }
else{
SPIStartRX(&(spiMsg.bufTxRx[0]));
          }
      }
   }/* end Task loop */
}
```

Додаток Б

(Обов’язковий)

ПРОГРАМНИЙ КОД ЕМУЛЯТОРА ПЕРИФЕРІЙНОГО ПРИСТОЮ З ІНТЕРФЕЙСОВ ЗВ’ЯЗКУ UART

Листинг 1. main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "dma.h"
#include "usart.h"
#include "gpio.h"

/* Private variables ---------------------------------------------------------*/
unsigned short BlinkBlueFlag, BlinkGreenFlag;

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
   HAL_Init();

   SystemClock_Config();

   MX_GPIO_Init();
   MX_DMA_Init();
   MX_USART1_UART_Init();

while (1)
   {
if (BlinkBlueFlag && BlinkGreenFlag) {
            HAL_GPIO_WritePin(GPIOC, LD4_Pin|LD3_Pin, GPIO_PIN_SET);
            HAL_Delay(500);
            HAL_GPIO_WritePin(GPIOC, LD4_Pin|LD3_Pin, GPIO_PIN_RESET);
            HAL_Delay(500);
      } else if (BlinkBlueFlag) {
            HAL_GPIO_WritePin(GPIOC, LD4_Pin, GPIO_PIN_SET);
            HAL_Delay(500);
            HAL_GPIO_WritePin(GPIOC, LD4_Pin, GPIO_PIN_RESET);
            HAL_Delay(500);
      } else if (BlinkGreenFlag) {
            HAL_GPIO_WritePin(GPIOC, LD3_Pin, GPIO_PIN_SET);
            HAL_Delay(500);
            HAL_GPIO_WritePin(GPIOC, LD3_Pin, GPIO_PIN_RESET);
            HAL_Delay(500);
      }
   }
}

/**
  * @brief System Clock Configuration
  * @retval None
```

```
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /**Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI_DIV2;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /**Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Листинг 1. stm32f1xx_it.c

```
/**
  * @brief This function handles USART1 global interrupt.
  */
void USART1_IRQHandler(void)
{
unsigned int              size_rx, sz_response;
static char              *response;
static  char              receive_buffer[32];

   // RX - channel 5
   // TX - channel 4

   // errors (frame, noise, overrun)
        if(USART1->SR & USART_SR_FE ||
          USART1->SR & USART_SR_NE ||
      USART1->SR & USART_SR_ORE){

       // clear status
                USART1->SR &= ~USART_SR_IDLE;
                USART1->SR &= ~USART_SR_TC;

       // reconfigure DMA on recive
                DMA1_Channel5->CCR &= ~DMA_CCR_EN;
                DMA1_Channel5->CMAR = (unsigned long)(receive_buffer[0]);
                DMA1_Channel5->CNDTR = sizeof(receive_buffer);
                DMA1->IFCR |= (DMA_IFCR_CTCIF5 | DMA_IFCR_CHTIF5);
                DMA1_Channel5->CCR |= DMA_CCR_EN;

       // reciever interrupt on
       USART1->CR1 |= USART_CR1_IDLEIE;
          }

   // receive a message
        if(USART1->SR & USART_SR_IDLE){

                USART1->SR &= ~USART_SR_IDLE;
                USART1->SR &= ~USART_SR_TC;
                DMA1_Channel5->CCR &= ~DMA_CCR_EN;

       // check message and prepare response
       size_rx = sizeof(receive_buffer) - DMA1_Channel5->CNDTR;
if (strncmp(receive_buffer, "blink blue", size_rx)) {
          BlinkBlueFlag = 1;
response = "okey blue";
          sz_response = sizeof("okey blue") - 1;
       } else if (strncmp(receive_buffer, "blink green", size_rx)) {
          BlinkGreenFlag = 1;
response = "okey green";
          sz_response = sizeof("okey green") - 1;
```

```
        } else if (strncmp(receive_buffer, "off blue", size_rx)) {
            BlinkBlueFlag = 0;
response = "off";
            sz_response = sizeof("off") - 1;
        } else if (strncmp(receive_buffer, "off green", size_rx)) {
            BlinkGreenFlag = 0;
response = "off";
            sz_response = sizeof("off") - 1;
        } else if (strncmp(receive_buffer, "astalavista baby", size_rx)) {
            BlinkBlueFlag = 1;
            BlinkGreenFlag = 1;
response = "I'll be back";
            sz_response = sizeof("I'll be back") - 1;
        } else {
response = "wrong";
            sz_response = sizeof("wrong") - 1;
        }

        // reconfig transmite dma to send respone
        DMA1_Channel4->CMAR  = (unsigned long)(&response);
        DMA1_Channel4->CNDTR = sz_response;
        DMA1->IFCR          |= (DMA_IFCR_CTCIF4 | DMA_IFCR_CHTIF4);
        DMA1_Channel4->CCR  |= DMA_CCR_EN;

        // transmitter interrupt on
        USART1->CR1 |= USART_CR1_TCIE;

            }

    // transmission complete
            if(USART1->SR & USART_SR_TC){

                    USART1->CR1 &= ~USART_CR1_TCIE;
                    USART1->SR  &= ~USART_SR_TC;

        // off transmitting dma
                    DMA1_Channel4->CCR &= ~DMA_CCR_EN;

        // reconfigure DMA on recive
        DMA1_Channel5->CMAR = (unsigned long)(&receive_buffer[0]);
                    DMA1_Channel5->CNDTR = sizeof(receive_buffer);
                    DMA1->IFCR |= (DMA_IFCR_CTCIF5 | DMA_IFCR_CHTIF5);
                    DMA1_Channel5->CCR |= DMA_CCR_EN;

        // reciever interrupt on
        USART1->CR1 |= USART_CR1_IDLEIE;
}
}
```

Додаток В

(Обов’язковий)

СЛАЙДИ ПРЕЗЕНАЦІЇ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Атестаційна робота з теми «Програмно-апаратний комутатор інтерфейсів вбудованих систем»

Виконав ст.гр. ІКТм-18-1 Шевцов І.О.
Керівник: доцент Бітченко А.Н.

ПРОБЛЕМАТИКА

- Велика кількість різних сенсорів та модулів з різними інтерфейсами, що необхідно об'єднувати між собою.
- Необхідність уніфікації доступу і балансування навантаження в системах з сенсорами та модулями, що використовують різні інтерфейси.
- Необхідність підтримки непромислових протоколів зв’язку пристроями комунікації.

ПОСТАНОВКА ЗАВДАННЯ

- Спроектувати пристрій, що буде здатний взаємодіяти з різними інтерфейсами зв'язку і різними налаштуваннями інтерфейсів.
- Пристрій має відносно легко налаштовуватись для користувача, а також вміти відправляти інформацію на різні інтерфейси і вміти приймати інформацію з них по запиту ззовні.

АНАЛОГІЧНІ РІШЕННЯ

- Сенсорні панельні контролери

Широкий вибір перетворювачів в залежності від потреб, але відсутня підтримка інтерфейсів I2C, SPI і CAN

Сенсорна панель контролера СПК207

IoT Маршрутизатор з підтримкою Modbus TCP / MQTT

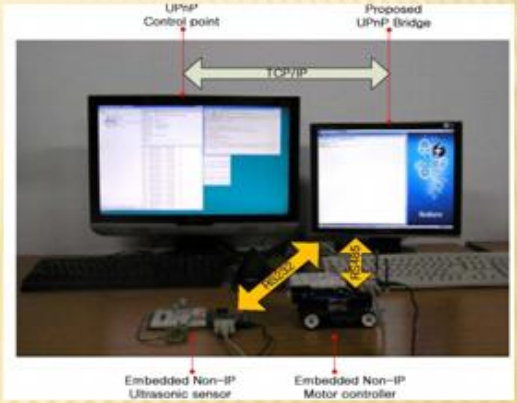
- Рішення компанії ADFWeb

Надмірний функціонал, також відсутня підтримка інтерфейсів I2C, SPI і CAN

АНАЛОГІЧНІ РІШЕННЯ (НАУКОВІ РОЗРОБКИ)

- Універсальний UPnP міст

Універсальне підключення будь-яких периферійних пристроїв, але складність налаштування для користувача.

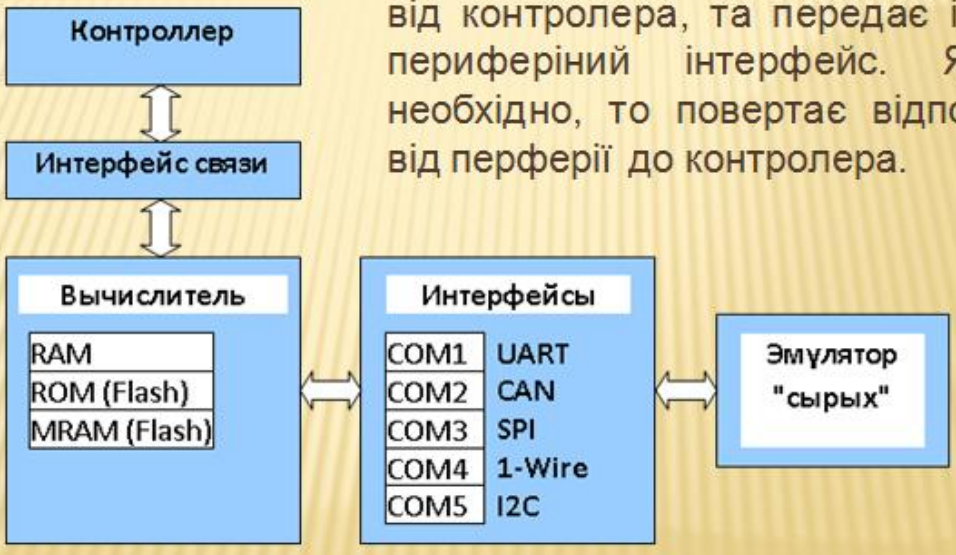


- Система взаємодії з мережами на базі архітектури ARM

Низька універсальність, відсутність налаштувань.

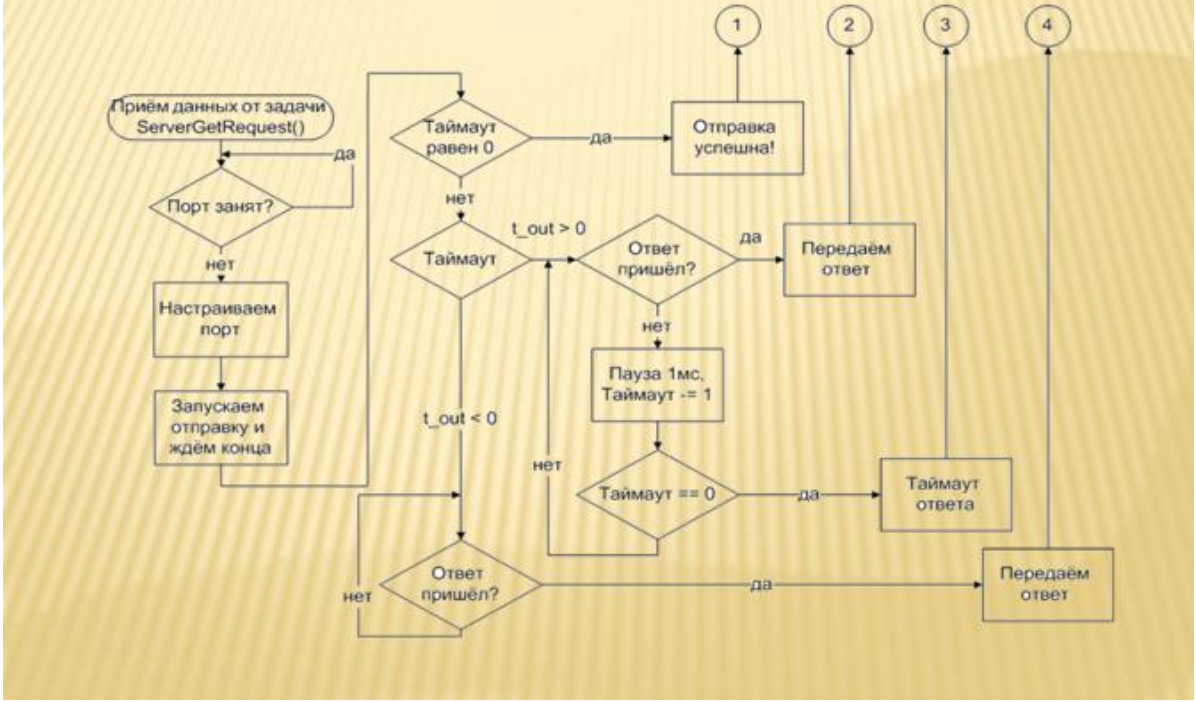
ПРИНЦИП РОБОТИ

Комутатор приймає повідомлення від контролера, та передає їх на периферіний інтерфейс. Якщо необхідно, то повертає відповідь від перферії до контролера.

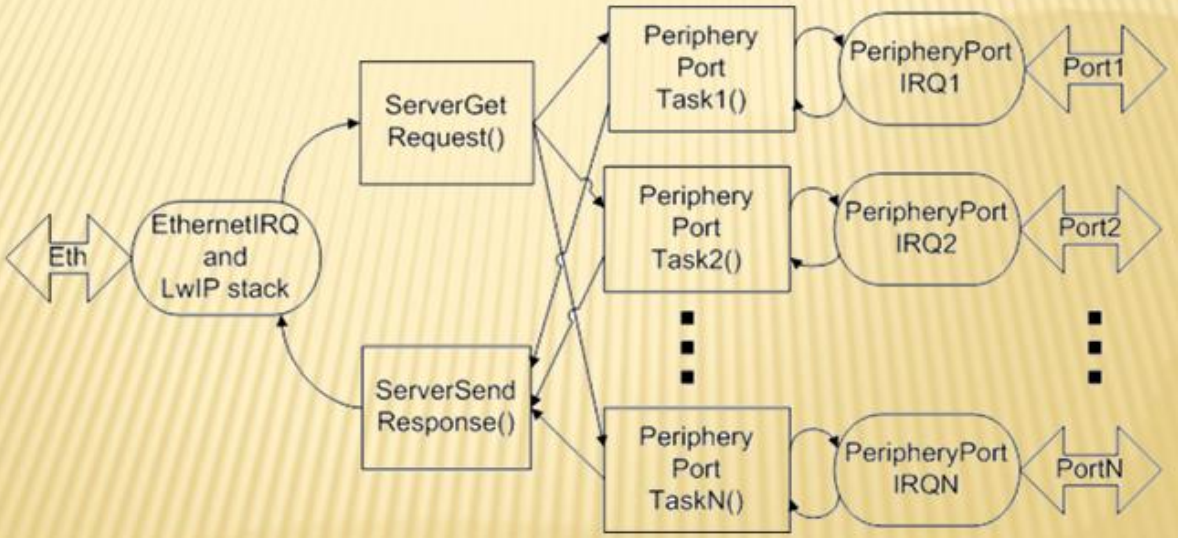


АЛГОРИТМ РОБОТИ

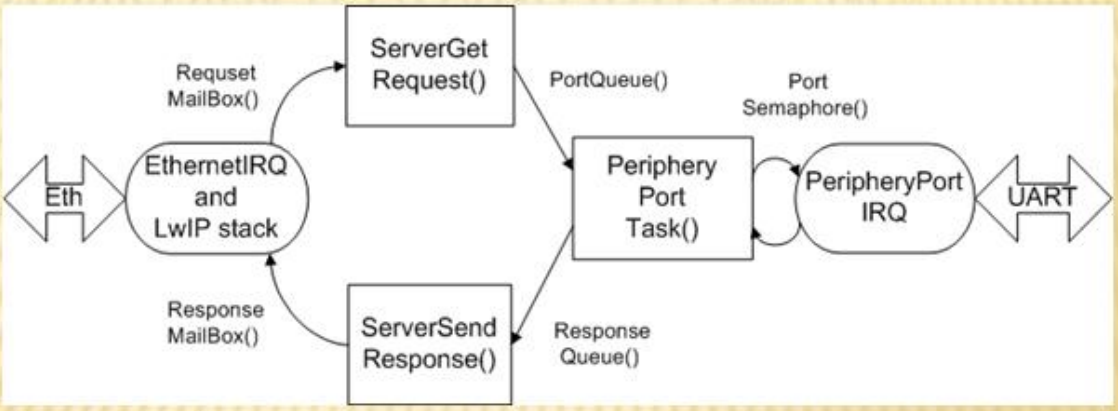
АЛГОРИТМ РОБОТИ (ПРОДОВЖЕННЯ)



СТРУКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



СТРУКТУРА ПЗ (ОДИН ІНТЕРФЕЙС)



За обробку запитів контролера відповідають задачі **ServerGetRequest()**, та **ServerGetResponse()**. Работу з периферією виконує задача **PeripheryPortTask()** та переривання **PeripheryPortIRQ**. Захист порту периферії виконує семафор **PortSemaphore()**. Для передачі повідомлень між задачами використовуються черги **ResponseQueue()** та **PortQueue()**.

ПРОБЛЕМНІ МОМЕНТИ РОЗРОБКИ

- Максимум можна підключити 8 периферійних пристроїв

Це повязано з обмеженою кількістю потоків DMA. В контролері присутні 2 контролери DMA, кожен з котрих має 8 потоків. Один інтерфейс використовує два потоки DMA.

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	SPDIFRX_DT	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	SPDIFRX_CS	SPI3_TX
Channel 1	I2C1_RX	I2C3_RX	TIM7_UP	-	TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	I2C4_RX	TIM4_CH2	-	I2C4_RX	TIM4_UP	TIM4_CH3
Channel 3	-	TIM2_UP TIM2_CH3	I2C3_RX	-	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX	UART7_TX	TIM3_CH4 TIM3_UP	UART7_RX	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	-	TIM5_UP	-
Channel 7	-	TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX
Channel 8	I2C3_TX	I2C4_RX	-	-	I2C2_TX	-	I2C4_TX	-
Channel 9	-	SPI2_RX	-	-	-	-	SPI2_TX	-

До того ж не для всіх інтерфейсів доступні всі канали DMA

ПРОБЛЕМНІ МОМЕНТИ РОЗРОБКИ-2

- Мультіплексування периферійних інтерфейсів

Деякі інтерфейси для своєї роботи використовують одні і ті ж самі піни мікроконтролера

Інтерфейс	MOSI	MISO	SCK	CS
SPI1	PB5	PA6	PA5	PG10
SPI2	PC3	PC2	PB10	PB12
SPI3	PB2	PC11	PC10	PA4
SPI4	PE6	PE5	PE4	PE2
SPI5	PF9	PF8	PF7	PF6
SPI6	PG14	PA6	PA5	PG8

На слайді показано піни мікроконтролера котрі використовуються для роботи інтерфейсів SPI1-SPI6. Можно бачити що інтерфейси SPI1, та SPI6 використовують одні і ті ж піни PA5, PA6.

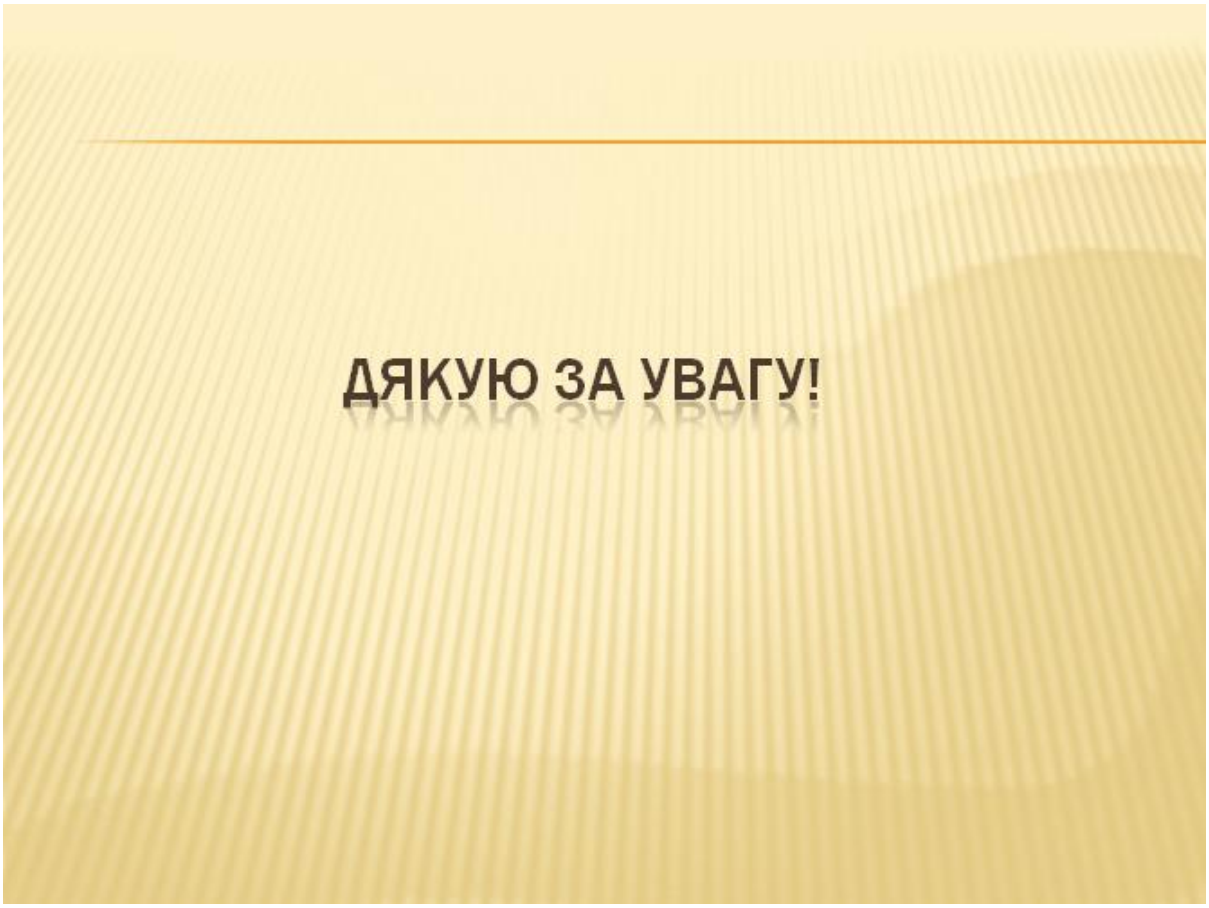
ВИСНОВКИ

Описане в пояснювальній записці технічне рішення було реалізовано на відлагоджувальній платі Nucleo-STM32F767ZI. По результатам випробувань, можна сказати, що реалізоване рішення повністю відповідає поставленій меті, а саме:

- Комутатор передає повідомлення від контролера верхнього рівня (ПК) до периферійного пристрою (емулятор UART) та повертає відповідь від периферії контролеру верхнього рівня.
- Комутатор має гнучку систему налаштувань, а також повідомлення користувача про помилки, що виникають в процесі зв'язку.

ПЕРСПЕКТИВИ РОЗВИТКУ

- Розробити апаратну частину комутатора з повною підтримку інтерфейсів RS-485, RS-232, CAN.
- Модернізувати апаратну частину з урахуванням необхідності захисту вхідних інтерфейсів від перенапруги.
- Додати підтримку перферіних інтерфейсів CAN, 1-Wire, I2S.
- Додати підтримку інтерфейсів CAN та RS-485 для контролерів верхнього рівня


ДЯКУЮ ЗА УВАГУ!

Додаток Г

(Обов’язковий)

ВІДОМІСТЬ АТЕСТАЦІЙНОГО ПРОЕКТУ

№ п/п	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮІК.509ХХХ.001 ПЗ	Пояснювальна записка	138 с.

Зм.	Арк.	№ докум.	Підп.	Дата	ГЮІК.509ХХХ.001 Д1		
Розроб.		Шевцов І.О.			Програмно-апаратний комутатор інтерфейсів вбудованих систем. Відомість атестаційної роботи	Літера	Аркуш 1 / Аркушів 1
Перевір.		БігченкоО.М.				У	
Т.контр.							ХНУРЕ Кафедра РТІКС
Н.контр.		Зарудний О.А					
Затв.		Цопа О.І.					