Zamek szyfrowy

AUTHOR Marcin Jonik, Michal Watroba
Wersja 2.0
N, 12 lut 2023

Spis treści

Table of contents

Indeks grup

Moduły

Tutaj znajduje się lista wszystkich grup:

Indeks plików

Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Dokumentacja grup

CMSIS

Moduły

Stm32f1xx_system

Opis szczegółowy

Stm32f1xx_system

Moduły

- STM32F1xx_System_Private_IncludesSTM32F1xx_System_Private_TypesDefinitions
- STM32F1xx_System_Private_Defines
- STM32F1xx_System_Private_Macros
- STM32F1xx_System_Private_Variables
- STM32F1xx_System_Private_FunctionPrototypes
- STM32F1xx_System_Private_Functions

Opis szczegółowy

STM32F1xx_System_Private_Includes

STM32F1xx_System_Private_TypesDefinitions

STM32F1xx_System_Private_Defines

Definicje

- #define HSE_VALUE 8000000U
- #define HSI_VALUE 8000000U

Opis szczegółowy

Dokumentacja definicji

#define HSE_VALUE 8000000U

Default value of the External oscillator in Hz. This value can be provided and adapted by the user application.

Definicja w linii 79 pliku system_stm32f1xx.c.

#define HSI_VALUE 8000000U

Default value of the Internal oscillator in Hz. This value can be provided and adapted by the user application.

Definicja w linii 84 pliku system_stm32f1xx.c.

STM32F1xx_System_Private_Macros

STM32F1xx_System_Private_Variables

Zmienne

- uint32_t SystemCoreClock = 16000000
- const uint8_t AHBPrescTable [16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t APBPrescTable [8U] = {0, 0, 0, 0, 1, 2, 3, 4}

Opis szczegółowy

Dokumentacja zmiennych

const uint8_t AHBPrescTable[16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}

Definicja w linii 143 pliku system_stm32f1xx.c.

const uint8_t APBPrescTable[8U] = {0, 0, 0, 0, 1, 2, 3, 4}

Definicja w linii 144 pliku system_stm32f1xx.c.

uint32_t SystemCoreClock = 16000000

Definicja w linii 142 pliku system_stm32f1xx.c.

STM32F1xx_System_Private_FunctionPrototypes

STM32F1xx_System_Private_Functions

Funkcje

- void SystemInit (void)
  Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.

- void SystemCoreClockUpdate (void)
  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Opis szczegółowy

Dokumentacja funkcji

void SystemCoreClockUpdate (void)

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Nota

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:

- If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(*)
- If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(**)
- If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(**) or HSI_VALUE(*) multiplied by the PLL factors.

(*) HSI_VALUE is a constant defined in stm32f1xx.h file (default value 8 MHz) but the real value may vary depending on the variations in voltage and temperature.

(**) HSE_VALUE is a constant defined in stm32f1xx.h file (default value 8 MHz or 25 MHz, depending on the product used), user has to ensure that HSE_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.

- The result of this function could be not correct when using fractional value for HSE crystal.
- **Parametry**

None	

- **Zwracane wartości**

None	

Definicja w linii 225 pliku system_stm32f1xx.c.

void SystemInit (void)

Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.

Nota

This function should be used only after reset.

Parametry

None	

Zwracane wartości

None	

Definicja w linii 176 pliku system_stm32f1xx.c.

Dokumentacja plików

Dokumentacja pliku STM3210C-Enkoder-Lock/Core/Src/freertos.c

```
#include "FreeRTOS.h"
#include "task.h"
#include "main.h"
#include "cmsis_os.h"
```

Funkcje

- void Wyswietlacz (void const *argument)
  Function implementing the Task1 thread.

- void MX_FREERTOS_Init (void)
  FreeRTOS initialization.

- void vApplicationGetIdleTaskMemory (StaticTask_t **ppxIdleTaskTCBBuffer, StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize)
- void vApplicationGetTimerTaskMemory (StaticTask_t **ppxTimerTaskTCBBuffer, StackType_t **ppxTimerTaskStackBuffer, uint32_t *pulTimerTaskStackSize)
- void vApplicationTickHook (void)

Zmienne

- osThreadId Task1Handle

Dokumentacja funkcji

void MX_FREERTOS_Init (void)

FreeRTOS initialization.

Parametry

None	

Zwracane wartości

None	

Definicja w linii 112 pliku freertos.c.

void vApplicationGetIdleTaskMemory (StaticTask_t ** *ppxIdleTaskTCBBuffer*, StackType_t ** *ppxIdleTaskStackBuffer*, uint32_t * *pulIdleTaskStackSize*)

Definicja w linii 85 pliku freertos.c.

void vApplicationGetTimerTaskMemory (StaticTask_t ** *ppxTimerTaskTCBBuffer*, StackType_t ** *ppxTimerTaskStackBuffer*, uint32_t * *pulTimerTaskStackSize*)

Definicja w linii 98 pliku freertos.c.

__weak void vApplicationTickHook (void)

Definicja w linii 71 pliku freertos.c.

void Wyswietlacz (void const * *argument*)

Function implementing the Task1 thread.

Parametry

argument	Not used

Zwracane wartości

None	

Definicja w linii 151 pliku freertos.c.

Dokumentacja zmiennych

osThreadId Task1Handle

File Name : freertos.c Description : Code for freertos applications

Uwaga

Definicja w linii 50 pliku freertos.c.

freertos.c

Idź do dokumentacji tego pliku.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Includes ------------------------------------------------------------------*/
00021 #include "FreeRTOS.h"
00022 #include "task.h"
00023 #include "main.h"
00024 #include "cmsis_os.h"
00025
00026 /* Private includes ----------------------------------------------------------*/
00027 /* USER CODE BEGIN Includes */
00028
00029 /* USER CODE END Includes */
00030
00031 /* Private typedef -----------------------------------------------------------*/
00032 /* USER CODE BEGIN PTD */
00033
00034 /* USER CODE END PTD */
00035
00036 /* Private define ------------------------------------------------------------*/
00037 /* USER CODE BEGIN PD */
00038
00039 /* USER CODE END PD */
00040
00041 /* Private macro -------------------------------------------------------------*/
00042 /* USER CODE BEGIN PM */
00043
00044 /* USER CODE END PM */
00045
00046 /* Private variables ---------------------------------------------------------*/
00047 /* USER CODE BEGIN Variables */
00048
00049 /* USER CODE END Variables */
00050 osThreadId Task1Handle;
00051
00052 /* Private function prototypes -----------------------------------------------*/
00053 /* USER CODE BEGIN FunctionPrototypes */
00054
00055 /* USER CODE END FunctionPrototypes */
00056
00057 void Wyswietlacz(void const * argument);
00058
00059 void MX_FREERTOS_Init(void); /* (MISRA C 2004 rule 8.1) */
00060
00061 /* GetIdleTaskMemory prototype (linked to static allocation support) */
00062 void vApplicationGetIdleTaskMemory( StaticTask_t **ppxIdleTaskTCBBuffer,
StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize );
00063
00064 /* GetTimerTaskMemory prototype (linked to static allocation support) */
00065 void vApplicationGetTimerTaskMemory( StaticTask t **ppxTimerTaskTCBBuffer,
StackType_t **ppxTimerTaskStackBuffer, uint32_t *pulTimerTaskStackSize );
00066
00067 /* Hook prototypes */
00068 void vApplicationTickHook(void);
00069
00070 /* USER CODE BEGIN 3 */
00071 __weak void vApplicationTickHook( void )
00072 {
00073    /* This function will be called by each tick interrupt if
00074    configUSE_TICK_HOOK is set to 1 in FreeRTOSConfig.h. User code can be
00075    added here, but the tick hook is called from an interrupt context, so
00076    code must not attempt to block, and only the interrupt safe FreeRTOS API
00077    functions can be used (those that end in FromISR()). */
00078 }
00079 /* USER CODE END 3 */
00080
00081 /* USER CODE BEGIN GET IDLE TASK MEMORY */
00082 static StaticTask_t xIdleTaskTCBBuffer;
00083 static StackType_t xIdleStack[configMINIMAL_STACK_SIZE];
00084
00085 void vApplicationGetIdleTaskMemory( StaticTask_t **ppxIdleTaskTCBBuffer,
StackType_t **ppxIdleTaskStackBuffer, uint32_t *pulIdleTaskStackSize )
00086 {
```

```
00087   *ppxIdleTaskTCBBuffer = &xIdleTaskTCBBuffer;
00088   *ppxIdleTaskStackBuffer = &xIdleStack[0];
00089   *pulIdleTaskStackSize = configMINIMAL_STACK_SIZE;
00090   /* place for user code */
00091 }
00092 /* USER CODE END GET_IDLE_TASK_MEMORY */
00093
00094 /* USER CODE BEGIN GET_TIMER_TASK_MEMORY */
00095 static StaticTask_t xTimerTaskTCBBuffer;
00096 static StackType_t xTimerStack[configTIMER_TASK_STACK_DEPTH];
00097
00098 void vApplicationGetTimerTaskMemory( StaticTask_t **ppxTimerTaskTCBBuffer,
StackType_t **ppxTimerTaskStackBuffer, uint32_t *pulTimerTaskStackSize )
00099 {
00100   *ppxTimerTaskTCBBuffer = &xTimerTaskTCBBuffer;
00101   *ppxTimerTaskStackBuffer = &xTimerStack[0];
00102   *pulTimerTaskStackSize = configTIMER_TASK_STACK_DEPTH;
00103   /* place for user code */
00104 }
00105 /* USER CODE END GET_TIMER_TASK_MEMORY */
00106
00112 void MX_FREERTOS_Init(void) {
00113   /* USER CODE BEGIN Init */
00114
00115   /* USER CODE END Init */
00116
00117   /* USER CODE BEGIN RTOS_MUTEX */
00118   /* add mutexes, ... */
00119   /* USER CODE END RTOS_MUTEX */
00120
00121   /* USER CODE BEGIN RTOS_SEMAPHORES */
00122   /* add semaphores, ... */
00123   /* USER CODE END RTOS_SEMAPHORES */
00124
00125   /* USER CODE BEGIN RTOS_TIMERS */
00126   /* start timers, add new ones, ... */
00127   /* USER CODE END RTOS_TIMERS */
00128
00129   /* USER CODE BEGIN RTOS_QUEUES */
00130   /* add queues, ... */
00131   /* USER CODE END RTOS_QUEUES */
00132
00133   /* Create the thread(s) */
00134   /* definition and creation of Task1 */
00135   osThreadDef(Task1, Wyswietlacz, osPriorityIdle, 0, 128);
00136   Task1Handle = osThreadCreate(osThread(Task1), NULL);
00137
00138   /* USER CODE BEGIN RTOS_THREADS */
00139   /* add threads, ... */
00140   /* USER CODE END RTOS_THREADS */
00141
00142 }
00143
00144 /* USER CODE BEGIN Header_Wyswietlacz */
00150 /* USER CODE END Header_Wyswietlacz */
00151 void Wyswietlacz(void const * argument)
00152 {
00153   /* USER CODE BEGIN Wyswietlacz */
00154   /* Infinite loop */
00155   for(;;)
00156   {
00157     osDelay(1);
00158   }
00159   /* USER CODE END Wyswietlacz */
00160 }
00161
00162 /* Private application code --------------------------------------------------*/
00163 /* USER CODE BEGIN Application */
00164
00165 /* USER CODE END Application */
00166
```

Dokumentacja pliku STM3210C-Enkoder-Lock/Core/Src/gpio.c

This file provides code for the configuration of all used GPIO pins.

```
#include "gpio.h"
```

Funkcje

- void MX_GPIO_Init (void)

Opis szczegółowy

This file provides code for the configuration of all used GPIO pins.

Uwaga

Definicja w pliku gpio.c.

Dokumentacja funkcji

void MX_GPIO_Init (void)

Configure pins as Analog Input Output EVENT_OUT EXTI

Definicja w linii 42 pliku gpio.c.

gpio.c

Idź do dokumentacji tego pliku.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ------------------------------------------------------------------*/
00022 #include "gpio.h"
00023
00024 /* USER CODE BEGIN 0 */
00025
00026 /* USER CODE END 0 */
00027
00028 /*----------------------------------------------------------------------------*/
00029 /* Configure GPIO                                                             */
00030 /*----------------------------------------------------------------------------*/
00031 /* USER CODE BEGIN 1 */
00032
00033 /* USER CODE END 1 */
00034
00042 void MX_GPIO_Init(void)
00043 {
00044
00045   GPIO_InitTypeDef GPIO_InitStruct = {0};
00046
00047   /* GPIO Ports Clock Enable */
00048   __HAL_RCC_GPIOC_CLK_ENABLE();
00049   __HAL_RCC_GPIOB_CLK_ENABLE();
00050   __HAL_RCC_GPIOE_CLK_ENABLE();
00051   __HAL_RCC_GPIOD_CLK_ENABLE();
00052
00053   /*Configure GPIO pin Output Level */
00054   HAL_GPIO_WritePin(GPIOB, Anoda_1_Pin|Anoda_4_Pin|Anoda_2_Pin|Anoda_3_Pin,
GPIO_PIN_RESET);
00055
00056   /*Configure GPIO pin Output Level */
00057   HAL_GPIO_WritePin(GPIOE, LED_1_Pin|LED_2_Pin, GPIO_PIN_RESET);
00058
00059   /*Configure GPIO pin Output Level */
00060   HAL_GPIO_WritePin(GPIOD, Katoda_A_Pin|Katoda_B_Pin|Katoda_C_Pin|Katoda_D_Pin
00061                           |Katoda_E_Pin|Katoda_F_Pin|Katoda_G_Pin|Katoda_H_Pin,
GPIO_PIN_RESET);
00062
00063   /*Configure GPIO pins : PCPin PCPin */
00064   GPIO_InitStruct.Pin = Enkoder_1_Pin|Klawisz_Pin;
00065   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00066   GPIO_InitStruct.Pull = GPIO_PULLUP;
00067   HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
00068
00069   /*Configure GPIO pins : PBPin PBPin PBPin PBPin */
00070   GPIO_InitStruct.Pin = Anoda_1_Pin|Anoda_4_Pin|Anoda_2_Pin|Anoda_3_Pin;
00071   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00072   GPIO_InitStruct.Pull = GPIO_NOPULL;
00073   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00074   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00075
00076   /*Configure GPIO pins : PEPin PEPin */
00077   GPIO_InitStruct.Pin = LED_1_Pin|LED_2_Pin;
00078   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00079   GPIO_InitStruct.Pull = GPIO_NOPULL;
00080   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00081   HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
00082
00083   /*Configure GPIO pin : PtPin */
00084   GPIO_InitStruct.Pin = Enkoder_2_Pin;
00085   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00086   GPIO_InitStruct.Pull = GPIO_PULLUP;
00087   HAL_GPIO_Init(Enkoder_2_GPIO_Port, &GPIO_InitStruct);
00088
00089   /*Configure GPIO pins : PDPin PDPin PDPin PDPin
00090                           PDPin PDPin PDPin PDPin */
00091   GPIO_InitStruct.Pin = Katoda_A_Pin|Katoda_B_Pin|Katoda_C_Pin|Katoda_D_Pin
00092                           |Katoda_E_Pin|Katoda_F_Pin|Katoda_G_Pin|Katoda_H_Pin;
00093   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00094   GPIO_InitStruct.Pull = GPIO_NOPULL;
00095   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
00096   HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);
00097
00098 }
00099
00100 /* USER CODE BEGIN 2 */
00101
00102 /* USER CODE END 2 */
```

Dokumentacja pliku STM3210C-Enkoder-Lock/Core/Src/main.c

Main program body.

```
#include "main.h"
#include "cmsis_os.h"
#include "gpio.h"
#include "Zamek.h"
```

Funkcje

- void SystemClock_Config (void)
  System Clock Configuration.

- void MX_FREERTOS_Init (void)
  FreeRTOS initialization.

- void vTask1 (void *pvParameters)
  Funkcja obslugi wyswietlacza siedmio-segmentowego.

- void vTask2 (void *pvParameters)
  Sprawdzanie poprawnosci wprowadzonego hasla.

- void vTask3 (void *pvParameters)
  Funkcja obslugujaca enkoder obrotowy.

- void vApplicationTickHook (void)
- int main (void)
  The application entry point.

- void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim)
  Period elapsed callback in non blocking mode.

- void Error_Handler (void)
  This function is executed in case of error occurrence.

Zmienne

- xSemaphoreHandle Tim50ms
- xSemaphoreHandle Tim20ms
- xSemaphoreHandle Tim2ms
- xQueueHandle Wys_A
- xQueueHandle Wys_B
- xQueueHandle Wys_C
- xQueueHandle Wys_D
- xQueueHandle HasloJendenZnak
- xQueueHandle ZmianaCyfry
- xQueueHandle ZmianaCyfry2

Opis szczegółowy

Main program body.

Autor

Marcin Jonik Michal Watroba

Wersja

2.0

Data

12.02.2023

Uwaga

Definicja w pliku main.c.

Dokumentacja funkcji

void Error_Handler (void)

This function is executed in case of error occurrence.

Zwracane wartości

None	

Definicja w linii 263 pliku main.c.

void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * *htim*)

Period elapsed callback in non blocking mode.

Nota

This function is called when TIM6 interrupt took place, inside HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment a global variable "uwTick" used as application time base.

Parametry

htim	: TIM handle

Zwracane wartości

None	

Definicja w linii 246 pliku main.c.

int main (void)

The application entry point.

Zwracane wartości

int	

Definicja w linii 115 pliku main.c.

void MX_FREERTOS_Init (void)

FreeRTOS initialization.

Parametry

None	

Zwracane wartości

None	

Definicja w linii 112 pliku freertos.c.

void SystemClock_Config (void)

System Clock Configuration.

Zwracane wartości

None	

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.

Initializes the CPU, AHB and APB buses clocks

Configure the Systick interrupt time

Definicja w linii 193 pliku main.c.

void vApplicationTickHook (void)

Definicja w linii 77 pliku main.c.

void vTask1 (void * *pvParameters*)

Funkcja obslugi wyswietlacza siedmio-segmentowego.

Parametry

argument	Not used

Zwracane wartości

None	*void vTask1();*

Obsluga wyswietlacza

Odpowiedzialne za: odczytanie danych z kolejki, zapisanie danych do tablicy znakow (numer komorek tablicy odpowiada numerowi wyswietlacza, wysylanie odpowiedniej zmiennej na dany wysiwetlacz, uaktywnianie tylko jednego wyswietlacza

Definicja w linii 65 pliku Zamek.c.

void vTask2 (void * *pvParameters*)

Sprawdzanie poprawnosci wprowadzonego hasla.

Parametry

argument	Not used

Zwracane wartości

None	`*void vTask2();*`

Sprawdzanie hasla

Odpowiedzialne za: odczytanie danych z kolejki, zapisanie danych do tablicy znakow (numer komorek tablicy odpowiada numerowi wyswietlacza, porównanie wprowadzonego hasla z haslem wzorcowym uaktywnienie diod led imitujacych otwarcie zamka

Definicja w linii 132 pliku Zamek.c.

void vTask3 (void * *pvParameters*)

Funkcja obslugujaca enkoder obrotowy.

Parametry

argument	Not used

Zwracane wartości

None	`*void vTask3();*`

Enkoder obrotowy

Odpowiedzialne za: pobieranie danych z enkodera, interpretowanie kierunku obracania enkodera, zmiana wartosci zmiennej przechowujacej aktualna wartosc wysylanie kolejka danych

Definicja w linii 191 pliku Zamek.c.

Dokumentacja zmiennych

xQueueHandle HasloJendenZnak

Definicja w linii 60 pliku main.c.

xSemaphoreHandle Tim20ms

Definicja w linii 52 pliku main.c.

xSemaphoreHandle Tim2ms

Definicja w linii 53 pliku main.c.

xSemaphoreHandle Tim50ms

Definicja w linii 51 pliku main.c.

xQueueHandle Wys_A

Definicja w linii 55 pliku main.c.

xQueueHandle Wys_B

Definicja w linii 56 pliku main.c.

xQueueHandle Wys_C

Definicja w linii 57 pliku main.c.

xQueueHandle Wys_D

Definicja w linii 58 pliku main.c.

xQueueHandle ZmianaCyfry

Definicja w linii 61 pliku main.c.

xQueueHandle ZmianaCyfry2

Definicja w linii 62 pliku main.c.

main.c

Idź do dokumentacji tego pliku.

```
00001 /* USER CODE BEGIN Header */
00023 /* USER CODE END Header */
00024 /* Includes ------------------------------------------------------------------*/
00025 #include "main.h"
00026 #include "cmsis_os.h"
00027 #include "gpio.h"
00028
00029 /* Private includes ----------------------------------------------------------*/
00030 /* USER CODE BEGIN Includes */
00031 #include "Zamek.h"
00032 /* USER CODE END Includes */
00033
00034 /* Private typedef -----------------------------------------------------------*/
00035 /* USER CODE BEGIN PTD */
00036
00037 /* USER CODE END PTD */
00038
00039 /* Private define ------------------------------------------------------------*/
00040 /* USER CODE BEGIN PD */
00041
00042 /* USER CODE END PD */
00043
00044 /* Private macro -------------------------------------------------------------*/
00045 /* USER CODE BEGIN PM */
00046
00047 /* USER CODE END PM */
00048
00049 /* Private variables ---------------------------------------------------------*/
00050 /* USER CODE BEGIN PV */
00051 xSemaphoreHandle Tim50ms;
00052 xSemaphoreHandle Tim20ms;
00053 xSemaphoreHandle Tim2ms;
00054
00055 xQueueHandle Wys_A;
00056 xQueueHandle Wys_B;
00057 xQueueHandle Wys_C;
00058 xQueueHandle Wys_D;
00059
00060 xQueueHandle HasloJendenZnak;
00061 xQueueHandle ZmianaCyfry;
00062 xQueueHandle ZmianaCyfry2;
00063 /* USER CODE END PV */
00064
00065 /* Private function prototypes -----------------------------------------------*/
00066 void SystemClock_Config(void);
00067 void MX_FREERTOS_Init(void);
00068 /* USER CODE BEGIN PFP */
00069 void vTask1(void *pvParameters);
00070 void vTask2(void *pvParameters);
00071 void vTask3(void *pvParameters);
00072 /* USER CODE END PFP */
00073
00074 /* Private user code ---------------------------------------------------------*/
00075 /* USER CODE BEGIN 0 */
00076
00077 void vApplicationTickHook(void)
00078 {
00079     /* definiowanie zmiennych lokalnych funckji */
00080   static uint8_t time50ms = 0;
00081   static uint8_t time20ms = 0;
00082   static uint8_t time2ms = 0;
00083   signed portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
00084
00085     /* Tworzenie impulsu z czestoscia odswiezania 2ms */
00086         if (time2ms++ > 1)
00087         {
00088             time2ms = 0;
00089             /* Wysylanie zmiennej time2ms przez Semaphor */
00090             xSemaphoreGiveFromISR(Tim2ms, &xHigherPriorityTaskWoken);
00091         }
00092     /* Tworzenie impulsu z czestoscia odswiezania 2ms */
00093         if (time20ms++ > 10)
00094         {
```

```
00095             time20ms = 0;
00096             /* Wysylanie zmiennej time20ms przez Semaphor */
00097             xSemaphoreGiveFromISR(Tim20ms, &xHigherPriorityTaskWoken);
00098         }
00099     /* Tworzenie impulsu z czestoscia odswiezania 2ms */
00100         if (time50ms++ >25)
00101         {
00102             time50ms = 0;
00103             /* Wysylanie zmiennej time50ms przez Semaphor */
00104             xSemaphoreGiveFromISR(Tim50ms, &xHigherPriorityTaskWoken);
00105         }
00106
00107   if (xHigherPriorityTaskWoken == pdTRUE) taskYIELD();
00108 }
00109 /* USER CODE END 0 */
00110
00115 int main(void)
00116 {
00117   /* USER CODE BEGIN 1 */
00118
00119   /* USER CODE END 1 */
00120
00121   /* MCU Configuration--------------------------------------------------------*/
00122
00123   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
00124   HAL_Init();
00125
00126   /* USER CODE BEGIN Init */
00127
00128   /* USER CODE END Init */
00129
00130   /* Configure the system clock */
00131   SystemClock_Config();
00132
00133   /* USER CODE BEGIN SysInit */
00134
00135   /* USER CODE END SysInit */
00136
00137   /* Initialize all configured peripherals */
00138   MX_GPIO_Init();
00139   /* USER CODE BEGIN 2 */
00140     /* Wstepne ustawianie diod LED */
00141   HAL_GPIO_WritePin(LED_1_GPIO_Port, LED_1_Pin, GPIO_PIN_SET);
00142   HAL_GPIO_WritePin(LED_2_GPIO_Port, LED_2_Pin, GPIO_PIN_SET);
00143
00144 /* definiowanie Semaphora imitujacego czas 50ms */
00145     vSemaphoreCreateBinary(Tim50ms);
00146     /* definiowanie Semaphora imitujacego czas 20ms */
00147   vSemaphoreCreateBinary(Tim20ms);
00148     /* definiowanie Semaphora imitujacego czas 2ms */
00149   vSemaphoreCreateBinary(Tim2ms);
00150
00151 /* definiowanie kolejek przesylajacych wyswietlane dane */
00152   Wys_A = xQueueCreate( 16, sizeof(uint8_t) );
00153     Wys_B = xQueueCreate( 16, sizeof(uint8_t) );
00154     Wys_C = xQueueCreate( 16, sizeof(uint8_t) );
00155     Wys_D = xQueueCreate( 16, sizeof(uint8_t) );
00156
00157 /* definiowanie kolejki przesylajacej jeden znak hasla */
00158   HasloJendenZnak = xQueueCreate( 16, sizeof(uint8_t) );
00159 /* definiowanie kolejki przesylajacej numer wyswietlanej cyfry hasla */
00160   ZmianaCyfry = xQueueCreate( 16, sizeof(uint8_t) );
00161 /* definiowanie kolejki przesylajacej numer wyswietlanej cyfry hasla */
00162   ZmianaCyfry2 = xQueueCreate( 16, sizeof(uint8_t) );
00163
00164 /* definiowanie Taska1 odpowiadajacego za obsluge wyswietlaczy siedmiosegmentowych
*/
00165   xTaskCreate(vTask1, "WyswietlaczSiedmioSegmentowy", configMINIMAL_STACK_SIZE,
NULL,
00166 /* definiowanie Taska1 odpowiadajacego za sprawdzanie poprawnosci wprowadzonego
hasla */ main_TASK_PRIORITY, NULL);
00167     xTaskCreate(vTask2, "PorownywanieHasla", configMINIMAL STACK SIZE, NULL,
main_TASK_PRIORITY, NULL);
00168 /* definiowanie Taska1 odpowiadajacego za obsluge enkodera obrotowego */
00169     xTaskCreate(vTask3, "ObslugaEnkodera", configMINIMAL_STACK_SIZE, NULL,
main_TASK_PRIORITY, NULL);
00170   /* USER CODE END 2 */
```

```
00171
00172   /* Call init function for freertos objects (in freertos.c) */
00173   MX_FREERTOS_Init();
00174   /* Start scheduler */
00175   osKernelStart();
00176
00177   /* We should never get here as control is now taken by the scheduler */
00178   /* Infinite loop */
00179   /* USER CODE BEGIN WHILE */
00180   while (1)
00181   {
00182     /* USER CODE END WHILE */
00183
00184     /* USER CODE BEGIN 3 */
00185   }
00186   /* USER CODE END 3 */
00187 }
00188
00193 void SystemClock_Config(void)
00194 {
00195   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
00196   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
00197
00201   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00202   RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00203   RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV5;
00204   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
00205   RCC_OscInitStruct.Prediv1Source = RCC_PREDIV1_SOURCE_PLL2;
00206   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00207   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00208   RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
00209   RCC_OscInitStruct.PLL2.PLL2State = RCC_PLL2_ON;
00210   RCC_OscInitStruct.PLL2.PLL2MUL = RCC_PLL2_MUL10;
00211   RCC_OscInitStruct.PLL2.HSEPrediv2Value = RCC_HSE_PREDIV2_DIV2;
00212   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
00213   {
00214     Error_Handler();
00215   }
00218   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
00219                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
00220   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00221   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00222   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
00223   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
00224
00225   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
00226   {
00227     Error_Handler();
00228   }
00231   __HAL_RCC_PLLI2S_ENABLE();
00232 }
00233
00234 /* USER CODE BEGIN 4 */
00235
00236 /* USER CODE END 4 */
00237
00246 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
00247 {
00248   /* USER CODE BEGIN Callback 0 */
00249
00250   /* USER CODE END Callback 0 */
00251   if (htim->Instance == TIM6) {
00252     HAL_IncTick();
00253   }
00254   /* USER CODE BEGIN Callback 1 */
00255
00256   /* USER CODE END Callback 1 */
00257 }
00258
00263 void Error_Handler(void)
00264 {
00265   /* USER CODE BEGIN Error_Handler_Debug */
00266   /* User can add his own implementation to report the HAL error return state */
00267   __disable_irq();
00268   while (1)
00269   {
00270   }
```

```
00271   /* USER CODE END Error_Handler_Debug */
00272 }
00273
00274 #ifdef  USE_FULL_ASSERT
00282 void assert_failed(uint8_t *file, uint32_t line)
00283 {
00284   /* USER CODE BEGIN 6 */
00285   /* User can add his own implementation to report the file name and line number,
00286      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
00287   /* USER CODE END 6 */
00288 }
00289 #endif /* USE_FULL_ASSERT */
00290
```

Dokumentacja pliku STM3210C-Enkoder-Lock/Core/Src/stm32f1xx_hal_msp.c

This file provides code for the MSP Initialization and de-Initialization codes.

```
#include "main.h"
```

Funkcje

- void HAL_MspInit (void)

Opis szczegółowy

This file provides code for the MSP Initialization and de-Initialization codes.

Uwaga

Definicja w pliku stm32f1xx_hal_msp.c.

Dokumentacja funkcji

void HAL_MspInit (void)

Initializes the Global MSP.

DISABLE: JTAG-DP Disabled and SW-DP Disabled

Definicja w linii 63 pliku stm32f1xx_hal_msp.c.

stm32f1xx_hal_msp.c

Idź do dokumentacji tego pliku.

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Includes ------------------------------------------------------------------*/
00022 #include "main.h"
00023 /* USER CODE BEGIN Includes */
00024
00025 /* USER CODE END Includes */
00026
00027 /* Private typedef -----------------------------------------------------------*/
00028 /* USER CODE BEGIN TD */
00029
00030 /* USER CODE END TD */
00031
00032 /* Private define ------------------------------------------------------------*/
00033 /* USER CODE BEGIN Define */
00034
00035 /* USER CODE END Define */
00036
00037 /* Private macro -------------------------------------------------------------*/
00038 /* USER CODE BEGIN Macro */
00039
00040 /* USER CODE END Macro */
00041
00042 /* Private variables ---------------------------------------------------------*/
00043 /* USER CODE BEGIN PV */
00044
00045 /* USER CODE END PV */
00046
00047 /* Private function prototypes -----------------------------------------------*/
00048 /* USER CODE BEGIN PFP */
00049
00050 /* USER CODE END PFP */
00051
00052 /* External functions --------------------------------------------------------*/
00053 /* USER CODE BEGIN ExternalFunctions */
00054
00055 /* USER CODE END ExternalFunctions */
00056
00057 /* USER CODE BEGIN 0 */
00058
00059 /* USER CODE END 0 */
00063 void HAL_MspInit(void)
00064 {
00065   /* USER CODE BEGIN MspInit 0 */
00066
00067   /* USER CODE END MspInit 0 */
00068
00069   __HAL_RCC_AFIO_CLK_ENABLE();
00070     HAL_RCC_PWR_CLK_ENABLE();
00071
00072   /* System interrupt init*/
00073   /* PendSV_IRQn interrupt configuration */
00074   HAL_NVIC_SetPriority(PendSV_IRQn, 15, 0);
00075
00078   __HAL_AFIO_REMAP_SWJ_DISABLE();
00079
00080   /* USER CODE BEGIN MspInit 1 */
00081
00082   /* USER CODE END MspInit 1 */
00083 }
00084
00085 /* USER CODE BEGIN 1 */
00086
00087 /* USER CODE END 1 */
00088
```

Dokumentacja pliku STM3210C-Enkoder-Lock/Core/Src/stm32f1xx_hal_timebase_tim.c

HAL time base based on the hardware TIM.

```
#include "stm32f1xx_hal.h"
#include "stm32f1xx_hal_tim.h"
```

Funkcje

- HAL_StatusTypeDef HAL_InitTick (uint32_t TickPriority)
  This function configures the TIM6 as a time base source. The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.

- void HAL_SuspendTick (void)
  Suspend Tick increment.

- void HAL_ResumeTick (void)
  Resume Tick increment.

Zmienne

- TIM_HandleTypeDef htim6

Opis szczegółowy

HAL time base based on the hardware TIM.

Uwaga

Definicja w pliku stm32f1xx_hal_timebase_tim.c.

Dokumentacja funkcji

HAL_StatusTypeDef HAL_InitTick (uint32_t *TickPriority*)

This function configures the TIM6 as a time base source. The time source is configured to have 1ms time base with a dedicated Tick interrupt priority.

Nota

This function is called automatically at the beginning of program after reset by HAL_Init() or at any time when clock is configured, by HAL_RCC_ClockConfig().

Parametry

TickPriority	Tick interrupt priority.

Zwracane wartości

HAL	status

Definicja w linii 41 pliku stm32f1xx_hal_timebase_tim.c.

void HAL_ResumeTick (void)

Resume Tick increment.

Nota

Enable the tick increment by Enabling TIM6 update interrupt.

Parametry

None	

Zwracane wartości

None	

Definicja w linii 106 pliku stm32f1xx_hal_timebase_tim.c.

void HAL_SuspendTick (void)

Suspend Tick increment.

Nota

Disable the tick increment by disabling TIM6 update interrupt.

Parametry

None	

Zwracane wartości

None	

Definicja w linii 94 pliku stm32f1xx_hal_timebase_tim.c.

Dokumentacja zmiennych

TIM_HandleTypeDef htim6

Definicja w linii 28 pliku stm32f1xx_hal_timebase_tim.c.

stm32f1xx_hal_timebase_tim.c

Idź do dokumentacji tego pliku.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Includes ------------------------------------------------------------------*/
00021 #include "stm32f1xx_hal.h"
00022 #include "stm32f1xx_hal_tim.h"
00023
00024 /* Private typedef -----------------------------------------------------------*/
00025 /* Private define ------------------------------------------------------------*/
00026 /* Private macro -------------------------------------------------------------*/
00027 /* Private variables ---------------------------------------------------------*/
00028 TIM_HandleTypeDef        htim6;
00029 /* Private function prototypes -----------------------------------------------*/
00030 /* Private functions ---------------------------------------------------------*/
00031
00041 HAL_StatusTypeDef HAL_InitTick(uint32_t TickPriority)
00042 {
00043   RCC_ClkInitTypeDef    clkconfig;
00044   uint32_t              uwTimclock = 0;
00045   uint32_t              uwPrescalerValue = 0;
00046   uint32_t              pFLatency;
00047   /*Configure the TIM6 IRQ priority */
00048   HAL_NVIC_SetPriority(TIM6_IRQn, TickPriority ,0);
00049
00050   /* Enable the TIM6 global Interrupt */
00051   HAL_NVIC_EnableIRQ(TIM6_IRQn);
00052
00053   /* Enable TIM6 clock */
00054   __HAL_RCC_TIM6_CLK_ENABLE();
00055
00056   /* Get clock configuration */
00057   HAL_RCC_GetClockConfig(&clkconfig, &pFLatency);
00058
00059   /* Compute TIM6 clock */
00060   uwTimclock = 2*HAL_RCC_GetPCLK1Freq();
00061   /* Compute the prescaler value to have TIM6 counter clock equal to 1MHz */
00062   uwPrescalerValue = (uint32_t) ((uwTimclock / 1000000U) - 1U);
00063
00064   /* Initialize TIM6 */
00065   htim6.Instance = TIM6;
00066
00067   /* Initialize TIMx peripheral as follow:
00068   + Period = [(TIM6CLK/1000) - 1]. to have a (1/1000) s time base.
00069   + Prescaler = (uwTimclock/1000000 - 1) to have a 1MHz counter clock.
00070   + ClockDivision = 0
00071   + Counter direction = Up
00072   */
00073   htim6.Init.Period = (1000000U / 1000U) - 1U;
00074   htim6.Init.Prescaler = uwPrescalerValue;
00075   htim6.Init.ClockDivision = 0;
00076   htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
00077
00078   if(HAL_TIM_Base_Init(&htim6) == HAL_OK)
00079   {
00080     /* Start the TIM time Base generation in interrupt mode */
00081     return HAL_TIM_Base_Start_IT(&htim6);
00082   }
00083
00084   /* Return function status */
00085   return HAL_ERROR;
00086 }
00087
00094 void HAL_SuspendTick(void)
00095 {
00096   /* Disable TIM6 update Interrupt */
00097   __HAL_TIM_DISABLE_IT(&htim6, TIM_IT_UPDATE);
00098 }
00099
00106 void HAL_ResumeTick(void)
00107 {
00108   /* Enable TIM6 Update interrupt */
00109   __HAL_TIM_ENABLE_IT(&htim6, TIM_IT_UPDATE);
00110 }
```

```
00111
```

Dokumentacja pliku STM3210C-Enkoder-Lock/Core/Src/stm32f1xx_it.c

Interrupt Service Routines.

```
#include "main.h"
#include "stm32f1xx_it.h"
```

Funkcje

- void NMI_Handler (void)
  This function handles Non maskable interrupt.

- void HardFault_Handler (void)
  This function handles Hard fault interrupt.

- void MemManage_Handler (void)
  This function handles Memory management fault.

- void BusFault_Handler (void)
  This function handles Prefetch fault, memory access fault.

- void UsageFault_Handler (void)
  This function handles Undefined instruction or illegal state.

- void DebugMon_Handler (void)
  This function handles Debug monitor.

- void TIM6_IRQHandler (void)
  This function handles TIM6 global interrupt.

Zmienne

- TIM_HandleTypeDef htim6

Opis szczegółowy

Interrupt Service Routines.

Uwaga

Definicja w pliku stm32f1xx_it.c.

Dokumentacja funkcji

void BusFault_Handler (void)

This function handles Prefetch fault, memory access fault.
Definicja w linii 115 pliku stm32f1xx_it.c.

void DebugMon_Handler (void)

This function handles Debug monitor.
Definicja w linii 145 pliku stm32f1xx_it.c.

void HardFault_Handler (void)

This function handles Hard fault interrupt.
Definicja w linii 85 pliku stm32f1xx_it.c.

void MemManage_Handler (void)

This function handles Memory management fault.
Definicja w linii 100 pliku stm32f1xx_it.c.

void NMI_Handler (void)

This function handles Non maskable interrupt.
Definicja w linii 70 pliku stm32f1xx_it.c.

void TIM6_IRQHandler (void)

This function handles TIM6 global interrupt.
Definicja w linii 165 pliku stm32f1xx_it.c.

void UsageFault_Handler (void)

This function handles Undefined instruction or illegal state.
Definicja w linii 130 pliku stm32f1xx_it.c.

Dokumentacja zmiennych

TIM_HandleTypeDef htim6 `[extern]`

Definicja w linii 28 pliku stm32f1xx_hal_timebase_tim.c.

stm32f1xx_it.c

Idź do dokumentacji tego pliku.

```
00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019
00020 /* Includes ------------------------------------------------------------------*/
00021 #include "main.h"
00022 #include "stm32f1xx_it.h"
00023 /* Private includes ----------------------------------------------------------*/
00024 /* USER CODE BEGIN Includes */
00025 /* USER CODE END Includes */
00026
00027 /* Private typedef -----------------------------------------------------------*/
00028 /* USER CODE BEGIN TD */
00029
00030 /* USER CODE END TD */
00031
00032 /* Private define ------------------------------------------------------------*/
00033 /* USER CODE BEGIN PD */
00034
00035 /* USER CODE END PD */
00036
00037 /* Private macro -------------------------------------------------------------*/
00038 /* USER CODE BEGIN PM */
00039
00040 /* USER CODE END PM */
00041
00042 /* Private variables ---------------------------------------------------------*/
00043 /* USER CODE BEGIN PV */
00044
00045 /* USER CODE END PV */
00046
00047 /* Private function prototypes -----------------------------------------------*/
00048 /* USER CODE BEGIN PFP */
00049
00050 /* USER CODE END PFP */
00051
00052 /* Private user code ---------------------------------------------------------*/
00053 /* USER CODE BEGIN 0 */
00054
00055 /* USER CODE END 0 */
00056
00057 /* External variables --------------------------------------------------------*/
00058 extern TIM_HandleTypeDef htim6;
00059
00060 /* USER CODE BEGIN EV */
00061
00062 /* USER CODE END EV */
00063
00064 /******************************************************************************/
00065 /*           Cortex-M3 Processor Interruption and Exception Handlers          */
00066 /******************************************************************************/
00070 void NMI_Handler(void)
00071 {
00072   /* USER CODE BEGIN NonMaskableInt_IRQn 0 */
00073
00074   /* USER CODE END NonMaskableInt IRQn 0 */
00075   /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
00076   while (1)
00077   {
00078   }
00079   /* USER CODE END NonMaskableInt_IRQn 1 */
00080 }
00081
00085 void HardFault Handler(void)
00086 {
00087   /* USER CODE BEGIN HardFault_IRQn 0 */
00088
00089   /* USER CODE END HardFault IRQn 0 */
00090   while (1)
00091   {
00092     /* USER CODE BEGIN W1_HardFault_IRQn 0 */
00093     /* USER CODE END W1_HardFault_IRQn 0 */
00094   }
00095 }
```

```
00096
00100 void MemManage_Handler(void)
00101 {
00102   /* USER CODE BEGIN MemoryManagement_IRQn 0 */
00103
00104   /* USER CODE END MemoryManagement_IRQn 0 */
00105   while (1)
00106   {
00107     /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
00108     /* USER CODE END W1_MemoryManagement_IRQn 0 */
00109   }
00110 }
00111
00115 void BusFault_Handler(void)
00116 {
00117   /* USER CODE BEGIN BusFault_IRQn 0 */
00118
00119   /* USER CODE END BusFault_IRQn 0 */
00120   while (1)
00121   {
00122     /* USER CODE BEGIN W1_BusFault_IRQn 0 */
00123     /* USER CODE END W1_BusFault_IRQn 0 */
00124   }
00125 }
00126
00130 void UsageFault_Handler(void)
00131 {
00132   /* USER CODE BEGIN UsageFault_IRQn 0 */
00133
00134   /* USER CODE END UsageFault_IRQn 0 */
00135   while (1)
00136   {
00137     /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
00138     /* USER CODE END W1_UsageFault_IRQn 0 */
00139   }
00140 }
00141
00145 void DebugMon_Handler(void)
00146 {
00147   /* USER CODE BEGIN DebugMonitor_IRQn 0 */
00148
00149   /* USER CODE END DebugMonitor_IRQn 0 */
00150   /* USER CODE BEGIN DebugMonitor_IRQn 1 */
00151
00152   /* USER CODE END DebugMonitor_IRQn 1 */
00153 }
00154
00155 /******************************************************************************/
00156 /* STM32F1xx Peripheral Interrupt Handlers                                    */
00157 /* Add here the Interrupt Handlers for the used peripherals.                  */
00158 /* For the available peripheral interrupt handler names,                      */
00159 /* please refer to the startup file (startup_stm32f1xx.s).                    */
00160 /******************************************************************************/
00161
00165 void TIM6_IRQHandler(void)
00166 {
00167   /* USER CODE BEGIN TIM6_IRQn 0 */
00168
00169   /* USER CODE END TIM6_IRQn 0 */
00170   HAL_TIM_IRQHandler(&htim6);
00171   /* USER CODE BEGIN TIM6_IRQn 1 */
00172
00173   /* USER CODE END TIM6_IRQn 1 */
00174 }
00175
00176 /* USER CODE BEGIN 1 */
00177
00178 /* USER CODE END 1 */
00179
```

Dokumentacja pliku STM3210C-Enkoder-Lock/Core/Src/system_stm32f1xx.c

CMSIS Cortex-M3 Device Peripheral Access Layer System Source File.
`#include "stm32f1xx.h"`

Definicje

- #define HSE_VALUE 8000000U
- #define HSI_VALUE 8000000U

Funkcje

- void SystemInit (void)
  Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and update the SystemCoreClock variable.

- void SystemCoreClockUpdate (void)
  Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

Zmienne

- uint32_t SystemCoreClock = 16000000
- const uint8_t AHBPrescTable [16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9}
- const uint8_t APBPrescTable [8U] = {0, 0, 0, 0, 1, 2, 3, 4}

Opis szczegółowy

CMSIS Cortex-M3 Device Peripheral Access Layer System Source File.

Autor

MCD Application Team

1. This file provides two functions and one global variable to be called from user application:
   - SystemInit(): Setups the system clock (System clock source, PLL Multiplier factors, AHB/APBx prescalers and Flash settings). This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f1xx_xx.s" file.
   - SystemCoreClock variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
   - SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.
2. After each device reset the HSI (8 MHz) is used as system clock source. Then SystemInit() function is called, in "startup_stm32f1xx_xx.s" file, to configure the system clock before to branch to main program.
3. The default value of HSE crystal is set to 8 MHz (or 25 MHz, depending on the product used), refer to "HSE_VALUE". When HSE is used as system clock source, directly or through PLL, and you are using different crystal you have to adapt the HSE value to your own configuration.

Uwaga

Definicja w pliku system_stm32f1xx.c.

system_stm32f1xx.c

Idź do dokumentacji tego pliku.

```
00001
00059 #include "stm32f1xx.h"
00060
00077 #if !defined  (HSE_VALUE)
00078   #define HSE_VALUE               8000000U
00080 #endif /* HSE_VALUE */
00081
00082 #if !defined  (HSI_VALUE)
00083   #define HSI_VALUE               8000000U
00085 #endif /* HSI_VALUE */
00086
00088 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) ||
defined(STM32F103xE) || defined(STM32F103xG)
00089 /* #define DATA_IN_ExtSRAM */
00090 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG
*/
00091
00092 /* Note: Following vector table addresses must be defined in line with linker
00093          configuration. */
00097 /* #define USER_VECT_TAB_ADDRESS */
00098
00099 #if defined(USER_VECT_TAB_ADDRESS)
00102 /* #define VECT_TAB_SRAM */
00103 #if defined(VECT_TAB_SRAM)
00104 #define VECT_TAB_BASE_ADDRESS   SRAM_BASE
00106 #define VECT_TAB_OFFSET         0x00000000U
00108 #else
00109 #define VECT_TAB_BASE_ADDRESS   FLASH_BASE
00111 #define VECT_TAB_OFFSET         0x00000000U
00113 #endif /* VECT_TAB_SRAM */
00114 #endif /* USER_VECT_TAB_ADDRESS */
00115
00116 /******************************************************************************/
00117
00134   /* This variable is updated in three ways:
00135       1) by calling CMSIS function SystemCoreClockUpdate()
00136       2) by calling HAL API function HAL_RCC_GetHCLKFreq()
00137       3) each time HAL RCC ClockConfig() is called to configure the system clock
frequency
00138          Note: If you use this function to configure the system clock; then there
00139                is no need to call the 2 first functions listed above, since
SystemCoreClock
00140                variable is updated automatically.
00141   */
00142 uint32_t SystemCoreClock = 16000000;
00143 const uint8_t AHBPrescTable[16U] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9};
00144 const uint8_t APBPrescTable[8U] =  {0, 0, 0, 0, 1, 2, 3, 4};
00145
00154 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) ||
defined(STM32F103xE) || defined(STM32F103xG)
00155 #ifdef DATA_IN_ExtSRAM
00156   static void SystemInit_ExtMemCtl(void);
00157 #endif /* DATA_IN_ExtSRAM */
00158 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG
*/
00159
00176 void SystemInit (void)
00177 {
00178 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) ||
defined(STM32F103xE) || defined(STM32F103xG)
00179   #ifdef DATA IN ExtSRAM
00180     SystemInit_ExtMemCtl();
00181   #endif /* DATA_IN_ExtSRAM */
00182 #endif
00183
00184   /* Configure the Vector Table location ----------------------------------*/
00185 #if defined(USER VECT TAB ADDRESS)
00186   SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table Relocation
in Internal SRAM. */
00187 #endif /* USER_VECT_TAB_ADDRESS */
00188 }
00189
00225 void SystemCoreClockUpdate (void)
```

```
00226 {
00227   uint32_t tmp = 0U, pllmull = 0U, pllsource = 0U;
00228
00229 #if defined(STM32F105xC) || defined(STM32F107xC)
00230   uint32_t prediv1source = 0U, prediv1factor = 0U, prediv2factor = 0U, pll2mull =
0U;
00231 #endif /* STM32F105xC */
00232
00233 #if defined(STM32F100xB) || defined(STM32F100xE)
00234   uint32_t prediv1factor = 0U;
00235 #endif /* STM32F100xB or STM32F100xE */
00236
00237   /* Get SYSCLK source -------------------------------------------------------*/
00238   tmp = RCC->CFGR & RCC_CFGR_SWS;
00239
00240   switch (tmp)
00241   {
00242     case 0x00U:  /* HSI used as system clock */
00243       SystemCoreClock = HSI_VALUE;
00244       break;
00245     case 0x04U:  /* HSE used as system clock */
00246       SystemCoreClock = HSE_VALUE;
00247       break;
00248     case 0x08U:  /* PLL used as system clock */
00249
00250       /* Get PLL clock source and multiplication factor ----------------------*/
00251       pllmull = RCC->CFGR & RCC_CFGR_PLLMULL;
00252       pllsource = RCC->CFGR & RCC_CFGR_PLLSRC;
00253
00254 #if !defined(STM32F105xC) && !defined(STM32F107xC)
00255       pllmull = ( pllmull >> 18U) + 2U;
00256
00257       if (pllsource == 0x00U)
00258       {
00259         /* HSI oscillator clock divided by 2 selected as PLL clock entry */
00260         SystemCoreClock = (HSI_VALUE >> 1U) * pllmull;
00261       }
00262       else
00263       {
00264  #if defined(STM32F100xB) || defined(STM32F100xE)
00265        prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
00266        /* HSE oscillator clock selected as PREDIV1 clock entry */
00267        SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
00268  #else
00269        /* HSE selected as PLL clock entry */
00270        if ((RCC->CFGR & RCC_CFGR_PLLXTPRE) != (uint32_t)RESET)
00271        {/* HSE oscillator clock divided by 2 */
00272          SystemCoreClock = (HSE_VALUE >> 1U) * pllmull;
00273        }
00274        else
00275        {
00276          SystemCoreClock = HSE_VALUE * pllmull;
00277        }
00278  #endif
00279       }
00280 #else
00281       pllmull = pllmull >> 18U;
00282
00283       if (pllmull != 0x0DU)
00284       {
00285          pllmull += 2U;
00286       }
00287       else
00288       { /* PLL multiplication factor = PLL input clock * 6.5 */
00289         pllmull = 13U / 2U;
00290       }
00291
00292       if (pllsource == 0x00U)
00293       {
00294         /* HSI oscillator clock divided by 2 selected as PLL clock entry */
00295         SystemCoreClock = (HSI_VALUE >> 1U) * pllmull;
00296       }
00297       else
00298       {/* PREDIV1 selected as PLL clock entry */
00299
00300         /* Get PREDIV1 clock source and division factor */
00301         prediv1source = RCC->CFGR2 & RCC_CFGR2_PREDIV1SRC;
```

```
00302         prediv1factor = (RCC->CFGR2 & RCC_CFGR2_PREDIV1) + 1U;
00303
00304         if (prediv1source == 0U)
00305         {
00306           /* HSE oscillator clock selected as PREDIV1 clock entry */
00307           SystemCoreClock = (HSE_VALUE / prediv1factor) * pllmull;
00308         }
00309         else
00310         {/* PLL2 clock selected as PREDIV1 clock entry */
00311
00312           /* Get PREDIV2 division factor and PLL2 multiplication factor */
00313           prediv2factor = ((RCC->CFGR2 & RCC_CFGR2_PREDIV2) >> 4U) + 1U;
00314           pll2mull = ((RCC->CFGR2 & RCC_CFGR2_PLL2MUL) >> 8U) + 2U;
00315           SystemCoreClock = (((HSE_VALUE / prediv2factor) * pll2mull) /
prediv1factor) * pllmull;
00316         }
00317       }
00318 #endif /* STM32F105xC */
00319       break;
00320
00321     default:
00322       SystemCoreClock = HSI_VALUE;
00323       break;
00324   }
00325
00326   /* Compute HCLK clock frequency ----------------*/
00327   /* Get HCLK prescaler */
00328   tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) >> 4U)];
00329   /* HCLK clock frequency */
00330   SystemCoreClock >>= tmp;
00331 }
00332
00333 #if defined(STM32F100xE) || defined(STM32F101xE) || defined(STM32F101xG) ||
defined(STM32F103xE) || defined(STM32F103xG)
00340 #ifdef DATA_IN_ExtSRAM
00350 void SystemInit_ExtMemCtl(void)
00351 {
00352   __IO uint32_t tmpreg;
00356   /* Enable FSMC clock */
00357   RCC->AHBENR = 0x00000114U;
00358
00359   /* Delay after an RCC peripheral clock enabling */
00360   tmpreg = READ_BIT(RCC->AHBENR, RCC_AHBENR_FSMCEN);
00361
00362   /* Enable GPIOD, GPIOE, GPIOF and GPIOG clocks */
00363   RCC->APB2ENR = 0x000001E0U;
00364
00365   /* Delay after an RCC peripheral clock enabling */
00366   tmpreg = READ_BIT(RCC->APB2ENR, RCC_APB2ENR_IOPDEN);
00367
00368   (void)(tmpreg);
00369
00370 /* ---------------  SRAM Data lines, NOE and NWE configuration ---------------*/
00371 /*----------------  SRAM Address lines configuration -------------------------*/
00372 /*----------------  NOE and NWE configuration ---------------------------------*/
00373 /*----------------  NE3 configuration -----------------------------------------*/
00374 /*----------------  NBL0, NBL1 configuration ----------------------------------*/
00375
00376   GPIOD->CRL = 0x44BB44BBU;
00377   GPIOD->CRH = 0xBBBBBBBBU;
00378
00379   GPIOE->CRL = 0xB44444BBU;
00380   GPIOE->CRH = 0xBBBBBBBBU;
00381
00382   GPIOF->CRL = 0x44BBBBBBU;
00383   GPIOF->CRH = 0xBBBB4444U;
00384
00385   GPIOG->CRL = 0x44BBBBBBU;
00386   GPIOG->CRH = 0x444B4B44U;
00387
00388 /*----------------  FSMC Configuration ----------------------------------------*/
00389 /*----------------  Enable FSMC Bank1 SRAM Bank -------------------------------*/
00390
00391   FSMC_Bank1->BTCR[4U] = 0x00001091U;
00392   FSMC_Bank1->BTCR[5U] = 0x00110212U;
00393 }
00394 #endif /* DATA_IN_ExtSRAM */
```

```
00395 #endif /* STM32F100xE || STM32F101xE || STM32F101xG || STM32F103xE || STM32F103xG
*/
00396
00408 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Dokumentacja pliku STM3210C-Enkoder-Lock/Core/Src/Zamek.c

Definicje funkcji Task.
`#include "Zamek.h"`

Funkcje

- void vTask1 (void *pvParameters)
  Funkcja obslugi wyswietlacza siedmio-segmentowego.

- void vTask2 (void *pvParameters)
  Sprawdzanie poprawnosci wprowadzonego hasla.

- void vTask3 (void *pvParameters)
  Funkcja obslugujaca enkoder obrotowy.

Zmienne

- const unsigned char seg7 []
- uint8_t LED_buf [4]
- uint8_t LED_ptr
- xSemaphoreHandle Tim50ms
- xSemaphoreHandle Tim20ms
- xSemaphoreHandle Tim2ms
- xQueueHandle Wys_A
- xQueueHandle Wys_B
- xQueueHandle Wys_C
- xQueueHandle Wys_D
- xQueueHandle HasloJendenZnak
- xQueueHandle ZmianaCyfry
- xQueueHandle ZmianaCyfry2

Opis szczegółowy

Definicje funkcji Task.

Autor

Marcin Jonik Michal Watroba

Wersja

2.0

Data

12.02.2023

Uwaga

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definicja w pliku Zamek.c.

Dokumentacja funkcji

void vTask1 (void * *pvParameters*)

Funkcja obslugi wyswietlacza siedmio-segmentowego.

Parametry

argument	Not used

Zwracane wartości

None	

```
*void vTask1();*
```

Obsluga wyswietlacza

Odpowiedzialne za: odczytanie danych z kolejki, zapisanie danych do tablicy znakow (numer komorek tablicy odpowiada numerowi wyswietlacza, wysylanie odpowiedniej zmiennej na dany wysiwetlacz, uaktywnianie tylko jednego wyswietlacza

Definicja w linii 65 pliku Zamek.c.

void vTask2 (void * *pvParameters*)

Sprawdzanie poprawnosci wprowadzonego hasla.

Parametry

argument	Not used

Zwracane wartości

None	

```
*void vTask2();*
```

Sprawdzanie hasla

Odpowiedzialne za: odczytanie danych z kolejki, zapisanie danych do tablicy znakow (numer komorek tablicy odpowiada numerowi wyswietlacza, porównanie wprowadzonego hasla z haslem wzorcowym uaktywnienie diod led imitujacych otwarcie zamka

Definicja w linii 132 pliku Zamek.c.

void vTask3 (void * *pvParameters*)

Funkcja obslugujaca enkoder obrotowy.

Parametry

argument	Not used

Zwracane wartości

None	*void vTask3();*

Enkoder obrotowy

Odpowiedzialne za: pobieranie danych z enkodera, interpretowanie kierunku obracania enkodera, zmiana wartosci zmiennej przechowujacej aktualna wartosc wysylanie kolejka danych

Definicja w linii 191 pliku Zamek.c.

Dokumentacja zmiennych

xQueueHandle HasloJendenZnak [extern]

Definicja w linii 60 pliku main.c.

uint8_t LED_buf[4]

Definicja w linii 28 pliku Zamek.c.

uint8_t LED_ptr

Definicja w linii 29 pliku Zamek.c.

const unsigned char seg7[]

```
Wartość początkowa:= {0xC0, 0xF9, 0xA4, 0xB0,
                      0x99, 0x92, 0x82, 0xF8,
                      0x80, 0x90, 0x5F}
```

Definicja w linii 24 pliku Zamek.c.

xSemaphoreHandle Tim20ms [extern]

Definicja w linii 52 pliku main.c.

xSemaphoreHandle Tim2ms [extern]

Definicja w linii 53 pliku main.c.

xSemaphoreHandle Tim50ms [extern]

Definicja w linii 51 pliku main.c.

xQueueHandle Wys_A [extern]

Definicja w linii 55 pliku main.c.

xQueueHandle Wys_B [extern]

Definicja w linii 56 pliku main.c.

xQueueHandle Wys_C `[extern]`

Definicja w linii 57 pliku main.c.

xQueueHandle Wys_D `[extern]`

Definicja w linii 58 pliku main.c.

xQueueHandle ZmianaCyfry `[extern]`

Definicja w linii 61 pliku main.c.

xQueueHandle ZmianaCyfry2 `[extern]`

Definicja w linii 62 pliku main.c.

Zamek.c

```
Idź do dokumentacji tego pliku.00001
00022 #include "Zamek.h"
00023
00024 const unsigned char seg7[] = {0xC0, 0xF9, 0xA4, 0xB0,
00025                               0x99, 0x92, 0x82, 0xF8,
00026                               0x80, 0x90, 0x5F};
00027
00028 uint8_t LED_buf[4];
00029 uint8_t LED_ptr;
00030
00031 extern xSemaphoreHandle Tim50ms;
00032 extern xSemaphoreHandle Tim20ms;
00033 extern xSemaphoreHandle Tim2ms;
00034
00035 extern xQueueHandle Wys_A;
00036 extern xQueueHandle Wys_B;
00037 extern xQueueHandle Wys_C;
00038 extern xQueueHandle Wys_D;
00039
00040 extern xQueueHandle HasloJendenZnak;
00041 extern xQueueHandle ZmianaCyfry;
00042 extern xQueueHandle ZmianaCyfry2;
00043
00044 /* USER CODE BEGIN Header_vTask1 */
00050 /* USER CODE END Header_vTask1 */
00051
00065 void vTask1(void *pvParameters)
00066 {
00067   static uint8_t znak[4] = {0,0,0,0};
00068     uint8_t wyswietl = 0;
00069
00070     for( ;; )
00071     {
00072         /* Odswierzanie wyswietlaczy z czestoscia 2ms */
00073         if (xSemaphoreTake(Tim2ms, portMAX_DELAY))
00074         {
00075             /* Pobieranie danych z kolejek */
00076             xQueueReceive(Wys_A, &znak[0], ( TickType t ) 0 );
00077             xQueueReceive(Wys_B, &znak[1], ( TickType_t ) 0 );
00078             xQueueReceive(Wys_C, &znak[2], ( TickType_t ) 0 );
00079             xQueueReceive(Wys_D, &znak[3], ( TickType_t ) 0 );
00080             xQueueReceive(ZmianaCyfry2, &wyswietl, (TickType_t)0);
00081             LED_buf[0] = seg7[znak[0]];
00082             LED_buf[1] = seg7[znak[1]];
00083             LED_buf[2] = seg7[znak[2]];
00084             LED_buf[3] = seg7[znak[3]];
00085           /* obsluga wyswietlacza siedmiosegmentowego LED */
00086             if ((++LED_ptr) > 3) LED_ptr = 0;
00087
00088             /* Resetowanie wyswietlaczy */
00089       HAL_GPIO_WritePin(Anoda_1_GPIO_Port, Anoda_4_Pin|Anoda_3_Pin|Anoda_2_Pin
00090                                           |Anoda_1_Pin, GPIO_PIN_RESET);
00091         /* Ustawienie segmentow */
00092       Katoda_A_GPIO_Port->BSRR = (uint32_t)LED_buf[wyswietl];
00093         Katoda A GPIO Port->BSRR = (uint32 t)(~LED_buf[wyswietl]) << 16;
00094
00095         /* Wybor wyswietlacza */
00096       switch (wyswietl)
00097       {
00098         case 0: HAL_GPIO_WritePin(Anoda_1_GPIO_Port, Anoda_1_Pin, GPIO_PIN_SET);
00099                 break;
00100         case 1: HAL_GPIO_WritePin(Anoda_2_GPIO_Port, Anoda_2_Pin, GPIO_PIN_SET);
00101                 break;
00102         case 2: HAL_GPIO_WritePin(Anoda_3_GPIO_Port, Anoda_3_Pin, GPIO_PIN_SET);
00103                 break;
00104         case 3: HAL_GPIO_WritePin(Anoda_4_GPIO_Port, Anoda_4_Pin, GPIO_PIN_SET);
00105                 break;
00106       }
00107         }
00108     }
00109 }
00110
00111     /* USER CODE BEGIN Header_vTask2 */
```

```
00117 /* USER CODE END Header_vTask2 */
00118
00132 void vTask2(void *pvParameters)
00133 {
00134   static uint8_t dane_enkoder = 0;
00135   static uint8_t klawisz_enkoder = 0;
00136   static uint8_t dane_wyswietlacz[4] = {0, 0, 0, 0};
00137
00138   for(;;)
00139   {
00140         /* Odswierzanie wyswietlaczy z czestoscia 50ms */
00141     if(xSemaphoreTake(Tim50ms, portMAX_DELAY))
00142     {
00143             /* Pobieranie danych z kolejek */
00144       xQueueReceive(HasloJendenZnak, &dane_enkoder, (TickType_t)0);
00145       xQueueReceive(ZmianaCyfry, &klawisz_enkoder, (TickType_t)0);
00146       dane_wyswietlacz[klawisz_enkoder] = dane_enkoder;
00147       if(klawisz_enkoder == 4)
00148       {
00149                 /* Sprawdzanie poprawnosci wprowadzonego hasla */
00150         if(dane_wyswietlacz[0] == 1 && dane_wyswietlacz[1] == 2 &&
dane_wyswietlacz[2] == 3 && dane_wyswietlacz[3]== 4)
00151         {
00152                     /* uruchamianie diod LED imitujacych otwarcie zamka */
00153           HAL_GPIO_WritePin(LED_1_GPIO_Port, LED_1_Pin, GPIO_PIN_RESET);
00154                     HAL_GPIO_WritePin(LED_2_GPIO_Port, LED_2_Pin,
GPIO_PIN_RESET);
00155         }
00156       }
00157       else
00158       {
00159         HAL_GPIO_WritePin(LED_1_GPIO_Port, LED_1_Pin, GPIO_PIN_SET);
00160         HAL_GPIO_WritePin(LED_2_GPIO_Port, LED_2_Pin, GPIO_PIN_SET);
00161       }
00162             xQueueSend(Wys_A , &dane_wyswietlacz[0], portMAX_DELAY);
00163             xQueueSend(Wys_B , &dane_wyswietlacz[1], portMAX_DELAY);
00164             xQueueSend(Wys_C , &dane_wyswietlacz[2], portMAX_DELAY);
00165             xQueueSend(Wys_D , &dane_wyswietlacz[3], portMAX_DELAY);
00166     }
00167   }
00168 }
00169
00170     /* USER CODE BEGIN Header_vTask3 */
00176 /* USER CODE END Header_vTask3 */
00177
00191 void vTask3(void *pvParameters)
00192 {
00193
00194   static uint8_t EnkoderWartosc = 0;
00195     static uint8_t flaga_zmiany = 0;
00196   static uint8_t klawiszWyswietlacza = 0;
00197     static uint8_t debouncer = 0, klawisz = 0;
00198     static uint8_t left = 0, right = 0;
00199
00200   for(;;)
00201   {
00202         /* Odswierzanie wyswietlaczy z czestoscia 2ms */
00203     if (xSemaphoreTake(Tim2ms, portMAX_DELAY))
00204         {
00205             /* Pobieranie danych z Enkodera */
00206             left = !HAL GPIO ReadPin(Enkoder 1 GPIO Port, Enkoder 1 Pin);
00207             right = !HAL_GPIO_ReadPin(Enkoder_2_GPIO_Port, Enkoder_2_Pin);
00208             switch(debouncer)
00209             {
00210                 case 0: //00
00211                     if(left&&!right) debouncer = 1;
00212                     if(!left&&right) debouncer = 4;
00213                     break;
00214                 case 1: //01
00215                     if(left&&!right) debouncer = 1;
00216                     else{ if(left&&right) debouncer = 2;
00217                     else debouncer = 0;}
00218                     break;
00219                 case 2: //11
00220                     if(left&&right) debouncer = 2;
00221                     else{ if(!left&&right) debouncer = 3;
00222                     else debouncer = 1;}
```

```
00223                     break;
00224                 case 3: //10
00225                     if(!left&&right) debouncer = 3;
00226                     else{ if(!left&&!right)
00227                     {
00228                         if(flaga_zmiany == 0)
00229                         {
00230                         if(EnkoderWartosc == 0 ) EnkoderWartosc = 9;
00231                         else EnkoderWartosc--;
00232                         flaga_zmiany = 1;
00233                         }
00234                         debouncer = 0;
00235                     }
00236                     else debouncer = 2;}
00237                     break;
00238                 case 4: //10
00239                     if(!left&&right) debouncer = 4;
00240                     else{ if(left&&right) debouncer = 5;
00241                     else debouncer = 0;}
00242                     break;
00243                 case 5: //11
00244                     if(left&&right) debouncer = 5;
00245                     else{ if(left&&!right) debouncer = 6;
00246                     else debouncer = 3;}
00247                     break;
00248                 case 6: //01
00249                     if(left&&!right) debouncer = 6;
00250                     else{ if(!left&&!right)
00251                     {
00252                         if(flaga_zmiany == 0)
00253                         {
00254                         if(EnkoderWartosc == 9) EnkoderWartosc = 0;
00255                         else EnkoderWartosc++;
00256                         flaga_zmiany = 1;
00257                         }
00258                         debouncer = 0;
00259                     }
00260                     else debouncer = 5;}
00261                     break;
00262 
00263             }
00264         }
00265         flaga_zmiany = 0;
00266         /* Odswierzanie wyswietlaczy z czestoscia 20ms */
00267         if (xSemaphoreTake(Tim20ms, portMAX_DELAY))
00268         {
00269             /* Debouncer przycisku potwierdzajacego zapisanie hasla */
00270       switch(klawisz)
00271             {
00272                 case 0:
00273                     if(HAL_GPIO_ReadPin(Klawisz_GPIO_Port,
Klawisz_Pin))klawisz=0;
00274                     else klawisz=1;
00275                     break;
00276                 case 1:
00277                     if(HAL_GPIO_ReadPin(Klawisz_GPIO_Port,
Klawisz_Pin))klawisz=1;
00278                     else
00279                         {
00280                             if(klawiszWyswietlacza == 4) klawiszWyswietlacza = 0;
00281                             else klawiszWyswietlacza++;
00282                             EnkoderWartosc = 0;
00283                             klawisz=2;
00284                         }
00285                     break;
00286                 case 2:
00287                     if(!HAL GPIO ReadPin(Klawisz GPIO Port,
Klawisz_Pin))klawisz=2;
00288                     else klawisz=3;
00289                     break;
00290                 case 3:
00291                     if(!HAL GPIO ReadPin(Klawisz GPIO Port,
Klawisz_Pin))klawisz=3;
00292                     else klawisz=0;
00293                     break;
00294             }
00295         }
```

```
00296         /* Wysylanie danych stosowna kolejka */
00297             xQueueSend(HasloJendenZnak , &EnkoderWartosc, portMAX_DELAY);
00298             xQueueSend(ZmianaCyfry , &klawiszWyswietlacza, portMAX_DELAY);
00299             xQueueSend(ZmianaCyfry2 , &klawiszWyswietlacza, portMAX_DELAY);
00300   }
00301 }
00302
```

Indeks

INDEX