Paulina Matyaszek

Paweł Głogowski

Informatyka

II rok

Komunikacja skryptu Python z stm32f3discovery

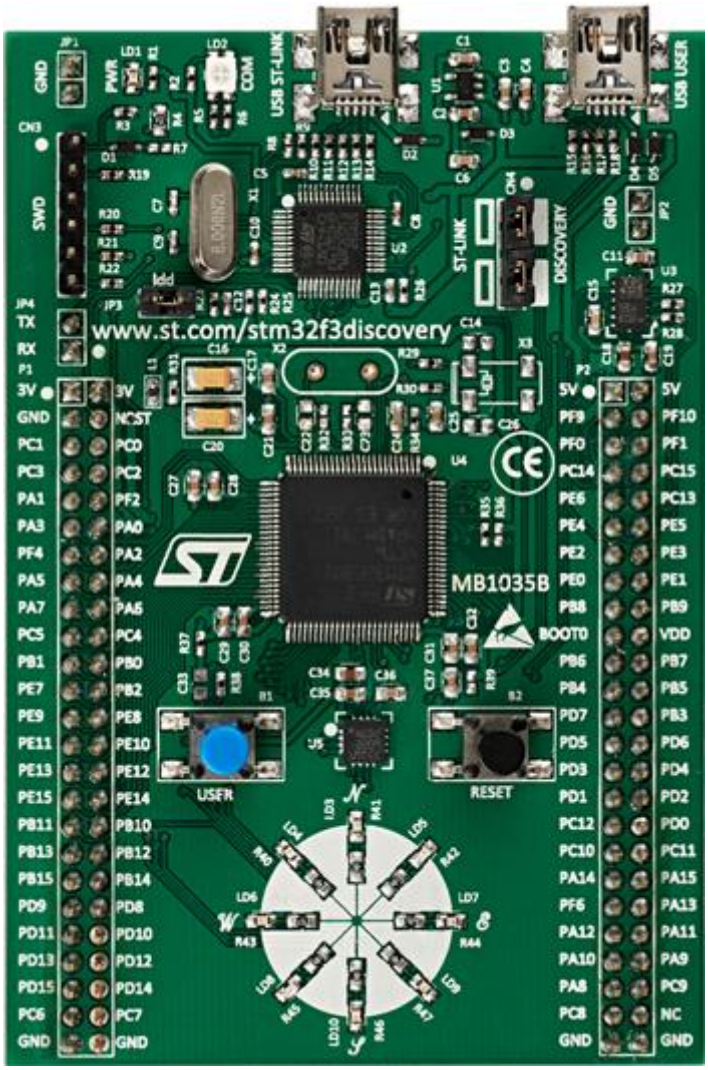
Zawartość:

1. Cel projektu

Celem projektu jest wykonanie i zaimplementowanie komunikacji między urządzeniem stm32f3discovery a skryptem Python. Komunikacja odbywa się przy pomocy portu COM. Efektem komunikacji powinno być zapalenie poszczególnych diod umieszczonych na płytce stm32f3, odczytanie temperatury z czujnika oraz wysłanie stringa do mikrokontrolera i odebranie go.

2. Środowisko

Mikrokontroler zastosowany w projekcie to stm32f3discovery.

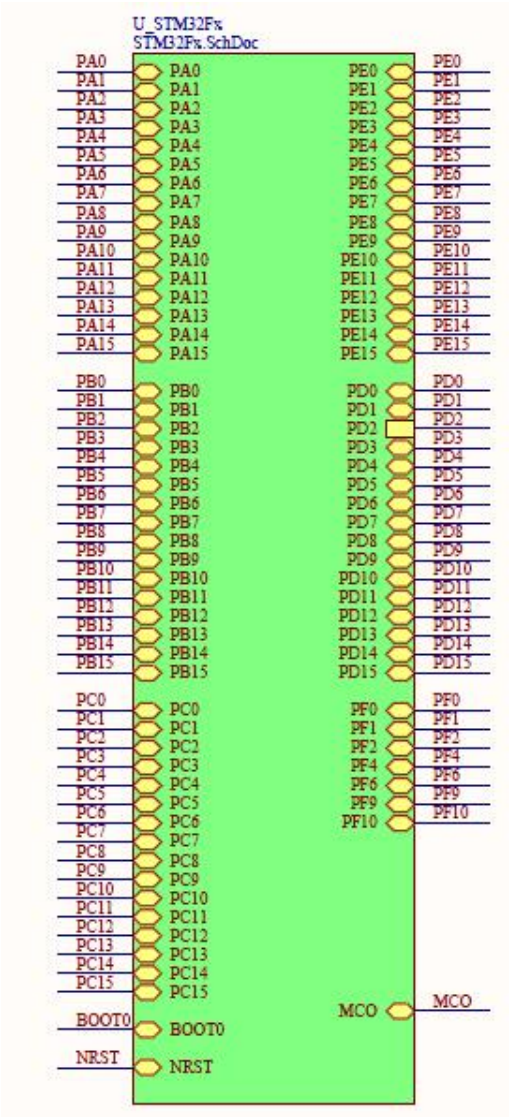


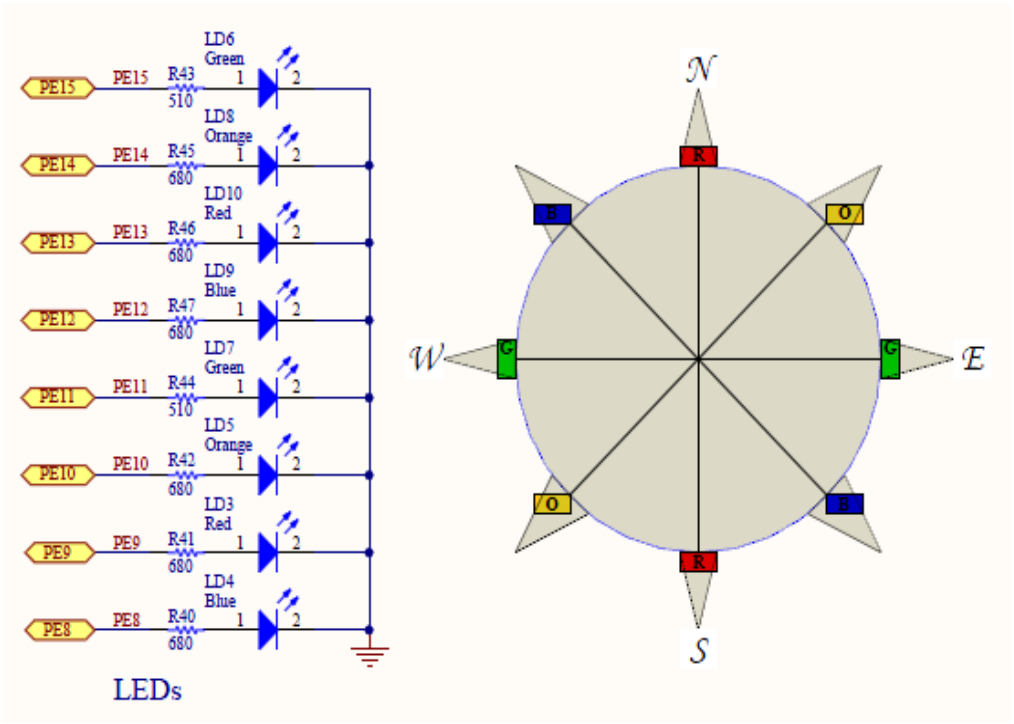
Cechują go następujące ogólne parametry:

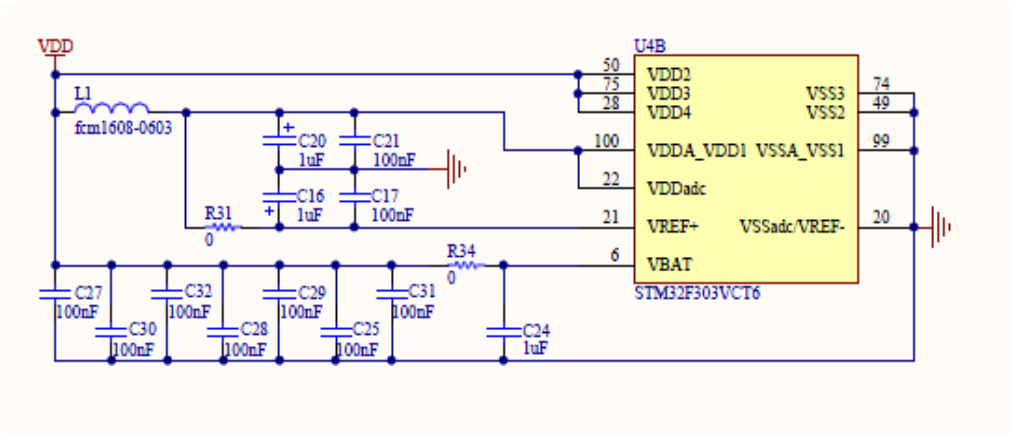
- Rdzeń: Cortex M4F
- Taktowanie 72 MHz
- 256 kB Flash

- 48kB RAM
- Obudowa LQFP100
- Debugger ST-Link/V2 umieszczony na płytce z możliwością pracy jako oddzielne urządzenie z wyjściem SWD
- Układ zasilany z USB lub z zewnętrznego źródła: 5V/3,3V
  Na płytce znajdują się także:
- 3- osiowy, cyfrowy żyroskop L3GD20
- 3- osiowy, cyfrowy akcelerometr z magnetometrem LSM303DLHC
- 10 diod LED
  (osiem do dyspozycji użytkownika)
- Dwa przyciski (użytkownika oraz reset)
- Złącze miniUSB
- Wyprowadzenia goldpin dla portów I/O

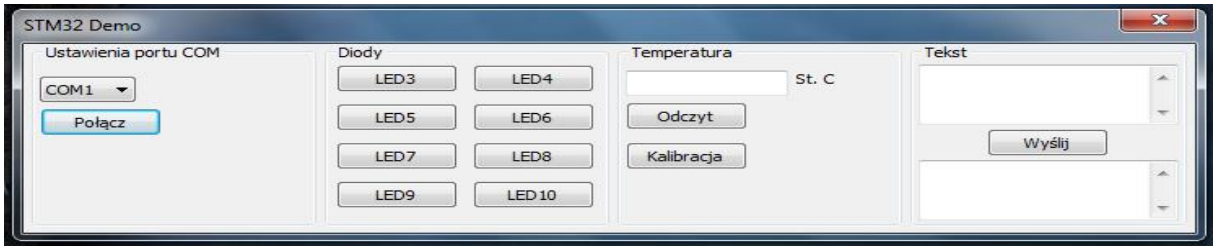
3. Realizacja

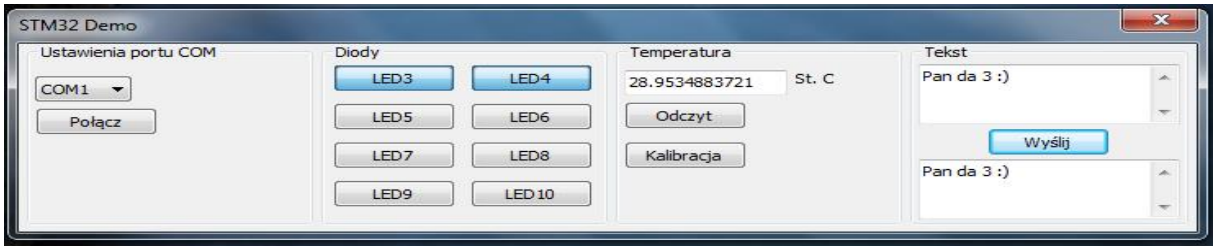






Skrypt Python





Aplikacja pozwala na zapalenie/gaszenie diod umieszczonych na płytce stm32, odczytanie temperatury z czujnika oraz wysyłanie i odbieranie stringa z mikrokontrolera . Skrypt umożliwia nam wybranie portu COM urządzenia oraz połączenie się z nim. Do dyspozycji mamy 8 przycisków obsługujących włączanie/wyłączanie diod, 2 przyciski umożliwiające odczytanie temperatury oraz kalibrację czujnika, 2 pola tekstowe.

4. Budowa ablikacji stm32

```
int main( void )
{
  // Zmienne uchwytów do kolejek
  static xQueueHandle xLedQueue = NULL;
  static xQueueHandle xTempQueue = NULL;
  static xQueueHandle xUSBOUTQueue = NULL;
  static xQueueHandle xStringQueue = NULL;
  static struct xQueueHandles xHandles;

  /* Konfiguracja sprzetu */
  prvSetupHardware();
  LED_Init();
  USB_Config();
  ADC_Config();

  /* Tworzenie kolejek */
  xUSBINQueue = xQueueCreate( 255, ( unsigned portBASE_TYPE ) sizeof( uint8_t ) );
  xUSBOUTQueue = xQueueCreate( 255, ( unsigned portBASE_TYPE ) sizeof( uint8_t ) );
  xLedQueue = xQueueCreate( 10, ( unsigned portBASE_TYPE ) sizeof( uint8_t ) );
  xTempQueue = xQueueCreate( 10, ( unsigned portBASE_TYPE ) sizeof( uint8_t ) );
  xStringQueue = xQueueCreate( 255, ( unsigned portBASE_TYPE ) sizeof( uint8_t ) );

  /* Struktura zawierajaca adresy kolejek */
  xHandles.LedHandle = xLedQueue;
  xHandles.StringHandle = xStringQueue;
  xHandles.TempHandle = xTempQueue;
  xHandles.USBINHandle = xUSBINQueue;
  xHandles.USBOUTHandle = xUSBOUTQueue;

  /*
  * Rejestr kolejek ulatwia debugowanie pogramu.
  * Dzieki temu debugger wyposazony w obsluge freeRTOS moze wyswietlic aktywne kolejki i
ich zawartosc.
  * Jezeli configQUEUE_REGISTRY_SIZE jest mniejsze od 1 to rejestr nie powstanie.
  */
  vQueueAddToRegistry( xUSBINQueue, ( signed char * ) "USBINQueue" );
  vQueueAddToRegistry( xUSBOUTQueue, ( signed char * ) "USBOUTQueue" );
  vQueueAddToRegistry( xLedQueue, ( signed char * ) "LedQueue" );
  vQueueAddToRegistry( xTempQueue, ( signed char * ) "TempQueue" );
  vQueueAddToRegistry( xStringQueue, ( signed char * ) "StringQueue" );

  /*
   * Tworzenie watkow
   */
  xTaskCreate( vUSBTask, ( signed char * ) "usb", configMINIMAL_STACK_SIZE * 2, ( void * )
&xHandles, 3, NULL );
```

```
  xTaskCreate( vLedTask, ( signed char * ) "led", configMINIMAL_STACK_SIZE * 2, ( void * )
&xLedQueue, 3, NULL );
  xTaskCreate( vTempTask, ( signed char * ) "temp", configMINIMAL_STACK_SIZE * 2, ( void *
) &xHandles, 3, NULL );
  xTaskCreate( vStringTask, ( signed char * ) "string", configMINIMAL_STACK_SIZE * 2, (
void * ) &xHandles, 3, NULL );

  /* Uruchomienie wątków. */
  vTaskStartScheduler();

  /*
   * Program nie powinien dotrzec do tego miejca. Moze sie to zdarzyc tylko w wypadku,
   * gdy brakuje RAMu dla utworzenia watku jałowego
   */
  for( ;; );
}
```

5. Podsumowanie

Tematyka i cel pracy, jakim było wykonanie i zaimplementowanie komunikacji pomiędzy mikrokontrolerem a skryptem Python za pomocą portu COM został pomyślnie zrealizowany. Udało się oprogramować układ elektryczny tak aby za pomocą skryptu Python można było zaświecać/gasić poszczególne diody, odczytywać temperaturę oraz wysyłać i odbierać stringa z mikrokontrolera.

6. Źródła

http://www.pezzino.ch/freertos-hello-world/ FreeRTOS HelloWorld

http://www.freertos.org/a00106.html FreeRTOS API Reference

http://www.freertos.org/a00102.html FreeRTOS Demo Projects

http://pyserial.sourceforge.net/pyserial_api.html PySerial API

http://wiki.wxpython.org/Getting%20Started wxPython Getting Started

90% Interfejsu graficznego zostało stworzone za pomocą programu wxGlade

http://wxglade.sourceforge.net/ (w folderze Python znajduje się plik gui.wxg będący plikiem projektu wxGlade)

http://www.beyondlogic.org/usbnutshell/usb1.shtml USB In a NutShell

http://docs.python.org/2/tutorial/ The Python Tutorial

http://www.icbase.com/File/HTML/hotic/html/docs/13465.pdf STM Virtual Com Port Demo (rozdział 6)

http://www.st.com/st-web-ui/static/active/en/st_prod_software_internet/resource/technical/software/driver/stsw-stm32102.zip STM32_USB_FS_Device_Lib_V4.0.0 i zawarte demo Virtual Com Port

Praktycznie cała obsługa wirtualnego portu COM pochodzi z tego dema. Zmieniliśmy tylko źródło i cel przesyłanych danych. W demie był to port USART1, w projekcie kolejki.

http://www.st.com/web/en/catalog/tools/PF258154 STM32F3 Discovery kit firmware

http://www.st.com/web/en/resource/technical/document/reference_manual/DM00043574.pdf STM32F3 Reference Manual

http://www.st.com/web/en/resource/technical/document/datasheet/DM00058181.pdf STM32F3x3 Datasheet