POROČILO ZA PROJEKT VIN

2023/24 Tema:

# LIGHTRANGER 8 CLICK

**Avtor:**

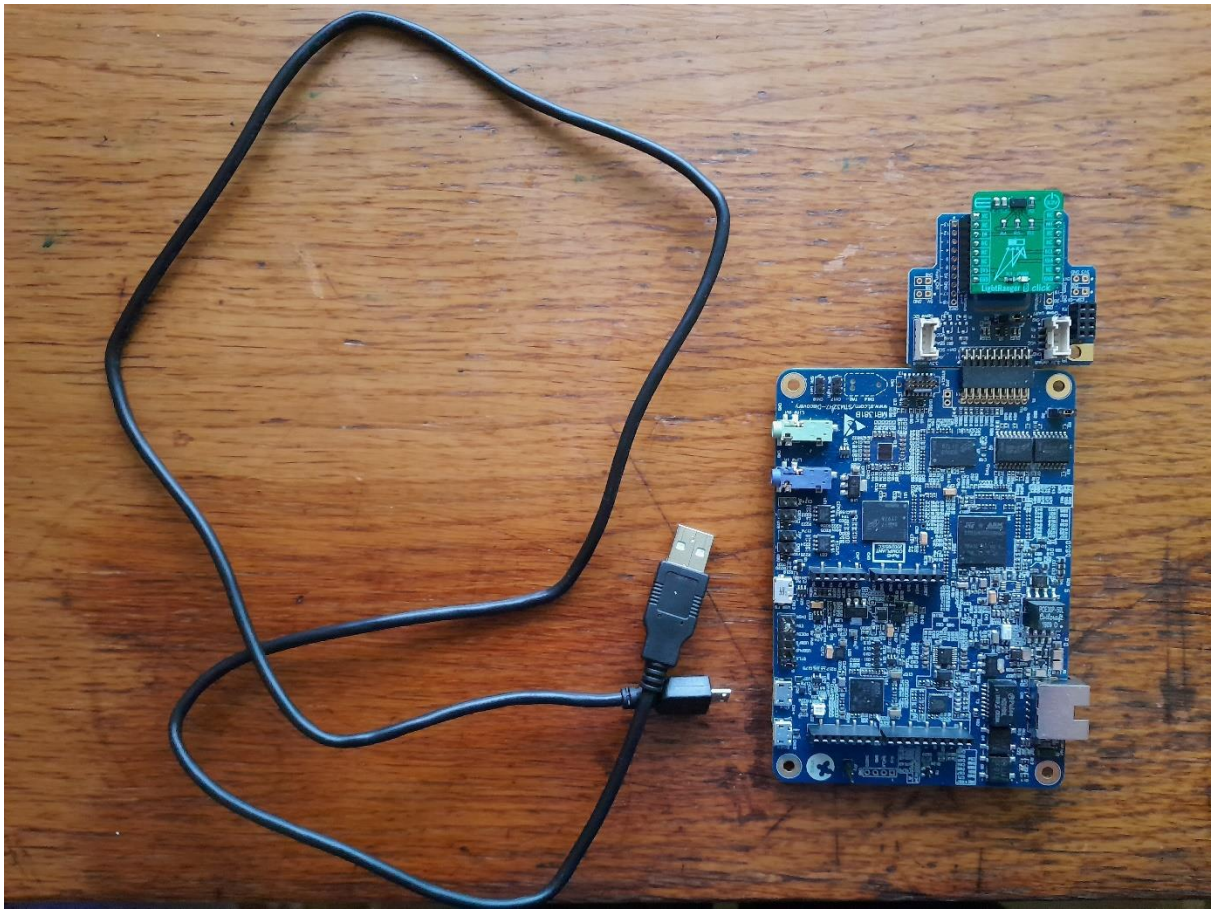
Klemen Braniselj

# KAZALO

# UVOD

Za VIN projekt sem se odločil delati z LIGHTRANGER 8. ToF senzorji so mi na sploh zanimiva tema. Za 1. domačo nalogo sem jih predstavil bolj na splošno, kako delujejo in kje jih uporabljamo.  Za projekt pa sem bom osredotočil na en specifičen senzor, in ga s STM32H750 sistem sprogramiral

Senzor, ki ga bom sprogramiral je VL53L3CX. Pri tem si bom pomagal s uradnimi gonilniki, ki jih bom dobil na spletni strani proizvajalca. Program bom napisal v STM32 CUBE IDE. Potreboval bom tudi STM32H750 in USB-C kabel

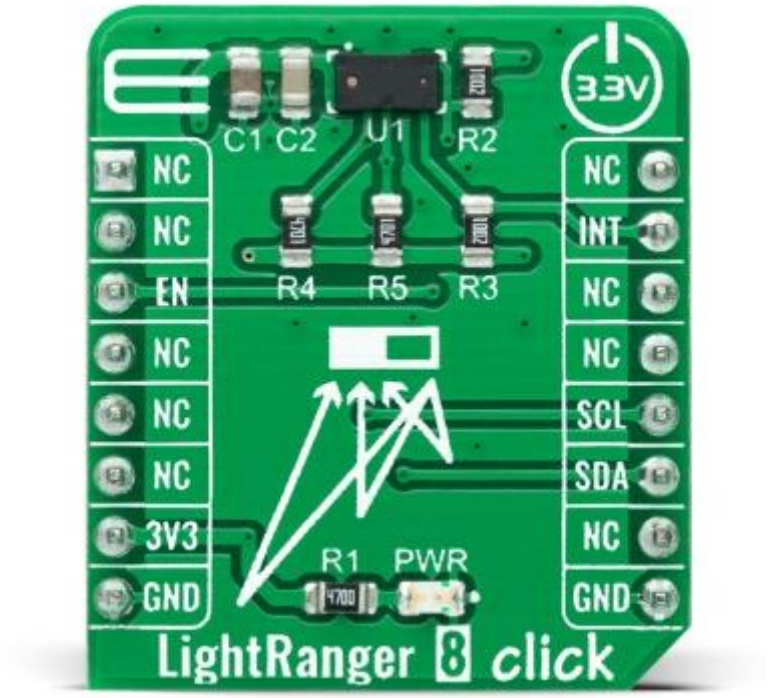


# PREDSTAVITEV NAPRAVE

## OPIS

Lightranger 8 Click ploščica vsebuje VL53L3CX senzor. Senzor VL53L3CX deluje na osnovi tehnologije Time-of-Flight (ToF), ki meri čas, ki ga svetlobni pulz potrebuje od oddaje do odboja od predmeta in nazaj do senzorja. Glavne komponente senzorja vključujejo laser za oddajanje svetlobnih impulzov, časovno letalski senzor za zaznavanje odbojnih impulzov in integriran signalni procesor za obdelavo podatkov.

## SLIKA



# INICIALIZACIJA IN KOMUNIKACIJA

Pri programiranju sem uporabil objektni pristop. Na začetku bom ustvaril 2 objektna tipa, ki bosta predstavljala senzor. lightranger8_t, in lightranger8_cfg_t

```
// Define the context object
typedef struct {

     GPIO_TypeDef *en_port;      /**< GPIO port for EN (Xshut) pin */
     uint16_t en_pin;            /**< GPIO pin number for EN (Xshut) */

     // Input pins
     GPIO_TypeDef *int_pin_port; /**< GPIO port for interrupt pin */
     uint16_t int_pin;           /**< GPIO pin number for interrupt pin */


     i2c_master_t i2c;

    I2C_HandleTypeDef *i2c_handle;
    uint16_t i2c_address;
    uint8_t slave_address;

    uint16_t fast_osc_frequency;
    uint16_t osc_calibrate_val;




} lightranger8_t;
```

To je glavni objekt za senzor. V njem hranim en pin in interrupt pin, i2c nastavitve, ipd

```
typedef struct
{
    GPIO_TypeDef* scl_port;   /**< Port for the I2C clock pin (SCL). */
    uint16_t scl_pin;         /**< Pin number for the I2C clock pin (SCL). */

    GPIO_TypeDef* sda_port;   /**< Port for the I2C data pin (SDA). */
    uint16_t sda_pin;         /**< Pin number for the I2C data pin (SDA). */

    GPIO_TypeDef* en_port;    /**< Port for the Xshut (enable) pin. */
    uint16_t en_pin;          /**< Pin number for the Xshut (enable) pin. */

    GPIO_TypeDef* int_port;   /**< Port for the interrupt pin (GPIO1). */
    uint16_t int_pin;         /**< Pin number for the interrupt pin (GPIO1). */

    uint32_t  i2c_speed;      /**< I2C serial speed. */
    uint8_t   i2c_address;    /**< I2C slave address. */

} lightranger8_cfg_t;
```

To je pa pomožen objekt za senzor. Vsebuje iste večinoma iste atribute kot glavni objkekt. Namesto da bi nastavljal vrednosti glavnega objekta, nastavljam preko pomožnega objekta. Tak imam lahko več senzorjev hkrati na vezju in večjo fleksibilnost.

```
void lightranger8_cfg_setup(lightranger8_cfg_t *cfg) {

   cfg->scl_port = GPIOD;
   cfg->sda_port = GPIOD;
   cfg->scl_pin = GPIO_PIN_12;
   cfg->sda_pin = GPIO_PIN_13;

     // Additional GPIO pins
     cfg->en_port = GPIOA;
     cfg->en_pin = GPIO_PIN_15;
     cfg->int_port = GPIOH;
     cfg->int_pin = GPIO_PIN_12;
*
     // I2C configuration
    //cfg->i2c_speed = 1000;
     */
     cfg->i2c_address = 0x52;  // Device I2C

 }
```

Ker imamo lahko za vsak senzor svojo konfiguracijsko datoteko, lahko nastavljamo pine po želji

Objekt ustvarimo v metodi main na naslednji način

```
lightranger8_t lightranger8;
lightranger8_cfg_t lightranger8_cfg;

lightranger8_cfg_setup( &lightranger8_cfg );
lightranger8_init( &lightranger8, &lightranger8_cfg );
```

Naslednji korak je Preverjanje komunikacije. V dokumentaciji piše da ima register 0X010F

Vrednot 0XEA. Is skeniranja I2C razberem da je naslov naprave 0x29. Nato iz tega naslova preberem ta register in potrdim da komunikacija deluje

```
HAL_StatusTypeDef retval = HAL_I2C_Mem_Read(&hi2c4, (0x29 << 1), 0x010F, I2C_MEMADD_SIZE_16BIT, dataBuffer, 1, HAL_MAX_DELAY);

snprintf(SendBuffer, BUFSIZE, "LIGHTRANGER ID: 0x%02X \n\r ",
dataBuffer[0]);

HAL_UART_Transmit(&huart3, (uint8_t*)SendBuffer, strlen(SendBuffer), 100);
```

```
I2C Scanning started !
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 0x1A[0x34] -  -  -  -  -
 -  -  -  -  -  -  -  -  -  - 0x29[0x52] -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 0x38[0x70] -  -  -
  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
- I2C Scanning stopped !
LIGHTRANGER ID: 0xEA
```

# BRANJE IN PISANJE

Branje in spreminjanje registrov sem implementiral z uporabov večih funkcij. Določene funkcije uporabljajo 8 bitne druge 16 bitne in nekateri celo 32 bitne.  Za to sem imel za vsako velikost podatkov svoje funkcije.

```
uint8_t read_data_u8 (lightranger8_t *ctx,uint16_t MEMORY_ADDRESS )
  {
      uint8_t bytes[1];

      retval = HAL_I2C_Mem_Read(&hi2c4,ctx->i2c_address,MEMORY_ADDRESS,I2C_MEMADD_SIZE_16BIT,bytes,1,HAL_MAX_DELAY);

      return bytes[0];
  }
```

Večinoma funkcije delujejo po istem principu. Uporabim ihi2c4 ročico iz objekta preberem i2c naslov. Potem pa glede na to kakšne podatke želim kodo ustrezno modificiram več bytov združim skupaj) ipd.

V originalnem gonilniku je bilo to izvedeno drugače. Vedno se kliče generic read ali write funkcije in potem se rezultat samo malo ustrezno oblikuje podatke glede na želen rezultat. Dosti bolj enostavno je bilo spustiti to funkcijo in kar v sami ločenih funkciji to sprogramirati.

```
void lightranger8_generic_write(I2C_HandleTypeDef *hi2c, uint16_t i2c_address, uint16_t reg, uint8_t *tx_buf, uint8_t tx_len) {
    uint8_t data_buf[257];  // Buffer to hold the register address and data to be sent

    // Set the register address in the first two bytes of data_buf
    data_buf[0] = reg >> 8;  // MSB of the register address
    data_buf[1] = reg & 0xFF; // LSB of the register address

    // Copy the data to be sent into the data buffer starting at position 2
    for (uint8_t cnt = 0; cnt < tx_len; cnt++) {
        data_buf[cnt + 2] = tx_buf[cnt];
    }


    HAL_I2C_Master_Transmit(hi2c, i2c_address << 1, data_buf, tx_len + 2, HAL_MAX_DELAY);


    }


void lightranger8_generic_read(I2C_HandleTypeDef *hi2c, uint16_t i2c_address, uint16_t reg, uint8_t *rx_buf, uint8_t rx_len) {
    uint8_t tx_buf[2];

    // Prepare the register address to be sent
    tx_buf[0] = reg >> 8;          // MSB of the register address
    tx_buf[1] = reg & 0xFF;        // LSB of the register address

    // Write the register address to the sensor
    if (HAL_I2C_Master_Transmit(hi2c, i2c_address << 1, tx_buf, 2, HAL_MAX_DELAY) != HAL_OK) {


    }

    // Read the data from the sensor
    if (HAL_I2C_Master_Receive(hi2c, i2c_address << 1, rx_buf, rx_len, HAL_MAX_DELAY) != HAL_OK) {


    }
}
```

# MERJENJE RAZDALJE IN UGOTOVITVE

Merjenje razdalje se izvaja z klicanjem funkcije lightranger8_start_measurement. Najprej se prebere podatki za kalibracijo, nato se PREKINITVEN nastavi na 0, ker pomeni da se lahko začne merjenje.

```
void lightranger8_start_measurement (lightranger8_t *ctx, uint32_t period_ms )
{
      uint16_t oscCalibrateVal = read_data_u16 ( ctx, LIGHTRANGER8_RESULT_OSC_CALIBRATE_VAL );
      write_data_u32(ctx, LIGHTRANGER8_SYSTEM_INTERMEASUREMENT_PERIOD, oscCalibrateVal * period_ms);
      lightranger8_system_interrupt_clear(ctx);

  write_data_u8(ctx, LIGHTRANGER8_SYSTEM_MODE_START, DEV_CMD_RANGING_ENABLE);
}
```

Senzor ima tudi možnost izbiranja razdalje. Na voljo so 3 možnosti SHORT, MEDIUM, LONG. Ta izbira spremeni določene registre ki določajo način merjenje razdalje.

```
uint8_t lightranger8_set_distance_mode ( lightranger8_t *ctx, uint8_t distance_mode ) {
    switch ( distance_mode ) {
        case LIGHTRANGER8_DISTANCE_MODE_SHORT: {
          write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VCSEL_PERIOD_A, DEV_SHORT_MODE_RAN_CONF_PERIOD_A );
          write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VCSEL_PERIOD_B, DEV_SHORT_MODE_RAN_CONF_PERIOD_B );
          write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VALID_PHASE_HIGH, DEV_SHORT_MODE_RAN_CONF_VAL_PH_H );
          write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_WOI_SD0, DEV_SHORT_MODE_SD_CONFIG_WOI_SD0 );
          write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_WOI_SD1, DEV_SHORT_MODE_SD_CONFIG_WOI_SD1 );
          write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_INITIAL_PHASE_SD0, DEV_SHORT_MODE_SD_CONF_IN_PH_SD0 );
          write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_INITIAL_PHASE_SD1, DEV_SHORT_MODE_SD_CONF_IN_PH_SD1 );
            break;
        }
        case LIGHTRANGER8_DISTANCE_MODE_MEDIUM: {
          write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VCSEL_PERIOD_A, DEV_MEDIUM_MODE_RAN_CONF_PERIOD_A );
          write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VCSEL_PERIOD_B, DEV_MEDIUM_MODE_RAN_CONF_PERIOD_B );
          write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VALID_PHASE_HIGH, DEV_MEDIUM_MODE_RAN_CONF_VAL_PH_H );
          write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_WOI_SD0, DEV_MEDIUM_MODE_SD_CONFIG_WOI_SD0 );
          write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_WOI_SD1, DEV_MEDIUM_MODE_SD_CONFIG_WOI_SD1 );
          write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_INITIAL_PHASE_SD0, DEV_MEDIUM_MODE_SD_CONF_IN_PH_SD0 );
          write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_INITIAL_PHASE_SD1, DEV_MEDIUM_MODE_SD_CONF_IN_PH_SD1 );
            break;
        }
        case LIGHTRANGER8_DISTANCE_MODE_LONG: {
           write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VCSEL_PERIOD_A, DEV_LONG_MODE_RAN_CONF_PERIOD_A );
           write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VCSEL_PERIOD_B, DEV_LONG_MODE_RAN_CONF_PERIOD_B );
           write_data_u8( ctx, LIGHTRANGER8_RANGE_CONFIG_VALID_PHASE_HIGH, DEV_LONG_MODE_RAN_CONF_VAL_PH_H );
           write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_WOI_SD0, DEV_LONG_MODE_SD_CONFIG_WOI_SD0 );
           write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_WOI_SD1, DEV_LONG_MODE_SD_CONFIG_WOI_SD1 );
           write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_INITIAL_PHASE_SD0, DEV_LONG_MODE_SD_CONF_IN_PH_SD0 );
           write_data_u8( ctx, LIGHTRANGER8_SD_CONFIG_INITIAL_PHASE_SD1, DEV_LONG_MODE_SD_CONF_IN_PH_SD1 );
                  break;
              }
        default:
        {
            return LIGHTRANGER8_RESP_WRONG_MODE;
        }
    }
}
```

To sem testiral tako da sem meril višino stropa (od mize na kateri je bil senzor) ki je 178cm.Po večih poskusih sem prišel do naslednjih ugotovitev:

SHORT: cca 150 cm

MEDIUM: cca 175 cm

LONG : cca 190 cm.

MEDIUM nastavitev je bila še najbolj točna. Pri razdaljah sem tudi opazil različne rezultate, če se vmes postavil predmet in ga višal ali nižal. SHORT in MEDIUM sta sorazmerno višali in nižali rezultat(recimo če sem predmet dvignil za 2 cm je bila razdalja spremenjena tudi za približno 1-4 cm). Pri LONG načinu pa ko se predmet premaknil za 2 cm(na višini 70 cm) se je rezultat spremenil za 10 do 15 cm.

Svetloba ima tudi zelo velik vpliv. Že če samo malo postavim senzor v senco se takoj pozna pri meritvi 20 – 30 cm.

# ZAKLJUČEK

Pri tem projektu sem se podrobno seznanil z ToF senzorjem. Na začetku se je naloga zdela nemogoča, vendar sem čez čas z nekaj truda prišel do delujoče rešitve. Zelo so bi bili pomoč gonilniki od MIKROE. Močno pa sem pogrešal bolj natančno dokumentacijo(kaj naredijo posamezni registri ipd.) V projektu nisem implementiral čisto vseh funkcionalnosti prvotne kode, ker bi to presegalo obseg projekta. Ugotovil sem tudi, da so nastavitve razdalje pri tem senzorju nadvse pomembne, čeprav se na prvi pogled ne zdijo. Svetloba je tudi zelo pomembna pri delovanju senzorja.

# VIRI

https://www.mikroe.com/lightranger-8-click?srsltid=AfmBOorBRPTVayk4FNiAbVnU71uZbWeQ3h8K2NJbN7bNiLIO40ig7FEg

https://libstock.mikroe.com/projects/view/4320/lightranger-8-click

https://download.mikroe.com/documents/datasheets/VL53L3CX%20Datasheet.pdf