```
#include "zb_error_handler.h"
#include "zb_mem_config_med.h"
#include "zb_zcl_reporting.h"
#include "zboss_api.h"
#include "zboss_api_core.h"
#include "zigbee_helpers.h"

#include "app_timer.h"
#include "nrf_delay.h"

#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "nrf_log_default_backends.h"

#include "zb_multi_sensor.h"
#include "zigbee.h"

#include "app_timer.h"
#include "boards.h"
#include "bsp.h"

#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "nrf_log_default_backends.h"

#include "nrf_802154.h"
/* I2c library */
//#include "I2C.h"
//#include "bh1750.h"
//#include "ADC.h"
#include "nrf_delay.h"

#define LED_BLINK ZB_MILLISECONDS_TO_BEACON_INTERVAL(200) /**< Led on-off timeout. */
#define NRF_LOG_BACKEND_UART_ENABLED 1

sensor_device_ctx_t m_dev_ctx;
int Cont;


APP_TIMER_DEF(zb_app_timer);

ZB_ZCL_DECLARE_IDENTIFY_ATTRIB_LIST(identify_attr_list, &m_dev_ctx.identify_attr.identify_time);

ZB_ZCL_DECLARE_BASIC_ATTRIB_LIST_EXT(basic_attr_list,
    &m_dev_ctx.basic_attr.zcl_version,
    &m_dev_ctx.basic_attr.app_version,
    &m_dev_ctx.basic_attr.stack_version,
    &m_dev_ctx.basic_attr.hw_version,
    m_dev_ctx.basic_attr.mf_name,
    m_dev_ctx.basic_attr.model_id,
    m_dev_ctx.basic_attr.date_code,
    &m_dev_ctx.basic_attr.power_source,
    m_dev_ctx.basic_attr.location_id,
    &m_dev_ctx.basic_attr.ph_env,
    m_dev_ctx.basic_attr.sw_ver);

ZB_ZCL_DECLARE_TEMP_MEASUREMENT_ATTRIB_LIST(temperature_attr_list,
    &m_dev_ctx.temp_attr.measure_value,
    &m_dev_ctx.temp_attr.min_measure_value,
    &m_dev_ctx.temp_attr.max_measure_value,
    &m_dev_ctx.temp_attr.tolerance);

ZB_ZCL_DECLARE_PRES_MEASUREMENT_ATTRIB_LIST(pressure_attr_list,
    &m_dev_ctx.pres_attr.measure_value,
    &m_dev_ctx.pres_attr.min_measure_value,
    &m_dev_ctx.pres_attr.max_measure_value,
    &m_dev_ctx.pres_attr.tolerance);

ZB_DECLARE_MULTI_SENSOR_CLUSTER_LIST(multi_sensor_clusters,
    basic_attr_list,
    identify_attr_list,
    temperature_attr_list,
    pressure_attr_list);

ZB_ZCL_DECLARE_MULTI_SENSOR_EP(multi_sensor_ep,
    MULTI_SENSOR_ENDPOINT,
    multi_sensor_clusters);

ZBOSS_DECLARE_DEVICE_CTX_1_EP(multi_sensor_ctx, multi_sensor_ep);

/**@brief Function for the Timer initialization.
 *
```

```
  ret_code_t err_code;

  // Initialize timer module.
  err_code = app_timer_init();

  if (err_code == NRF_SUCCESS) {

    // nrf_gpio_pin_toggle(LED2_G);
  }
  APP_ERROR_CHECK(err_code);
}

/**@brief Led blink function
 *
 * @param[in]   count of blink
 */
zb_void_t led_blink(zb_uint8_t count) {
  zb_ret_t zb_err_code;
  //NRF_LOG_INFO("led_blink. count: %d", count);
  if (count != 0) {
    nrf_gpio_pin_toggle(LED2_B);
    if (nrf_gpio_pin_out_read(LED2_B) != 0) {
      count--;
    }
    zb_err_code = ZB_SCHEDULE_APP_ALARM(led_blink, count, LED_BLINK);
  }
}

void zigbee_init(void) {

  zb_ret_t zb_err_code;
  ret_code_t err_code;
  zb_ieee_addr_t ieee_addr;

  /* Create Timer for reporting attribute */
  err_code = app_timer_create(&zb_app_timer, APP_TIMER_MODE_REPEATED, zb_app_timer_handler);
  APP_ERROR_CHECK(err_code);

  /* Set Zigbee stack logging level and traffic dump subsystem. */
  ZB_SET_TRACE_LEVEL(ZIGBEE_TRACE_LEVEL);
  ZB_SET_TRACE_MASK(ZIGBEE_TRACE_MASK);
  ZB_SET_TRAF_DUMP_OFF();

  /* Initialize Zigbee stack. */
  ZB_INIT("multi_sensor");

  /* Set device address to the value read from FICR registers. */
  zb_osif_get_ieee_eui64(ieee_addr);
  zb_set_long_address(ieee_addr);

  /* Set static long IEEE address. */
  zb_set_network_ed_role(IEEE_CHANNEL_MASK); // Set end device role, along with a channel mask // IEEE_CHANNEL_MASK is a custom channel
  // zb_set_bdb_secondary_channel_set(ZB_TRANSCEIVER_ALL_CHANNELS_MASK);
  zigbee_erase_persistent_storage(ERASE_PERSISTENT_CONFIG);

  // zb_set_ed_timeout(ED_AGING_TIMEOUT_2MIN);
  //  zb_set_ed_timeout(ED_AGING_TIMEOUT_64MIN);
  zb_set_ed_timeout(ED_AGING_TIMEOUT_32MIN);
 //zb_set_ed_timeout(ED_AGING_TIMEOUT_8MIN);


  //zb_set_keepalive_timeout(ZB_MILLISECONDS_TO_BEACON_INTERVAL(3000));
  zb_set_keepalive_timeout(ZB_MILLISECONDS_TO_BEACON_INTERVAL(60000));

  zigbee_power_down_unused_ram();
  zb_set_rx_on_when_idle(ZB_FALSE);

  zb_bdb_set_legacy_device_support(1); // Zigbee legacy mode (Zigbee PRO)

  /* Initialize application context structure. */
  UNUSED_RETURN_VALUE(ZB_MEMSET(&m_dev_ctx, 0, sizeof(m_dev_ctx)));

  /* Register temperature sensor device context (endpoints). */
  // ZB_AF_REGISTER_DEVICE_CTX(&multi_sensor_ctx);
  zb_af_register_device_ctx(&multi_sensor_ctx);

  /* Initialize sensor device attibutes */
  multi_sensor_clusters_attr_init();

  ZB_ZCL_SET_REPORT_ATTR_CB(report_attribute_cb);
```

```
  nrf_802154_tx_power_set(8);
  /** Start Zigbee Stack. */
  zb_err_code = zboss_start_no_autostart(); // start_no_autostart does not trigger the commissioning.
  //zb_err_code = zboss_start();
  // zb_err_code = zboss_start(); // start_no_autostart does not trigger the commissioning.
  ZB_ERROR_CHECK(zb_err_code);

  // bdb_start_top_level_commissioning(0); //

  // led_blink(5);
}

/**@brief Function for handling nrf app timer.
 *
 * @param[IN]   context   Void pointer to context function is called with.
 *
 * @details Function is called with pointer to sensor_device_ep_ctx_t as argument.
 */
static void zb_app_timer_handler(void *context) {
  // led_blink(1);
  zb_zcl_status_t zcl_status;
  static zb_int16_t new_temp_value, new_pres_value;

  /* Get new temperature measured value */
  new_temp_value = ms_sensorsim_measure_temperature();
  NRF_LOG_INFO("zb_app_timer_handler. temp: %d", new_temp_value);
  NRF_LOG_INFO("Is reporting on?: %d", zcl_is_attr_reported(MULTI_SENSOR_ENDPOINT, ZB_ZCL_CLUSTER_ID_TEMP_MEASUREMENT, ZB_ZCL_CLUSTER_S
  zcl_status = zb_zcl_set_attr_val(MULTI_SENSOR_ENDPOINT,
      ZB_ZCL_CLUSTER_ID_TEMP_MEASUREMENT,
      ZB_ZCL_CLUSTER_SERVER_ROLE,
      ZB_ZCL_ATTR_TEMP_MEASUREMENT_VALUE_ID,
      (zb_uint8_t *)&new_temp_value,
      ZB_FALSE);

  if (zcl_status != ZB_ZCL_STATUS_SUCCESS) {
  NRF_LOG_INFO("Set temperature value fail. zcl_status: %d", zcl_status);
  }

  /* Get new pressure measured value */

  /*  new_pres_value = ms_sensorsim_measure_pressure();
    zcl_status = zb_zcl_set_attr_val(MULTI_SENSOR_ENDPOINT,
        ZB_ZCL_CLUSTER_ID_PRESSURE_MEASUREMENT,
        ZB_ZCL_CLUSTER_SERVER_ROLE,
        ZB_ZCL_ATTR_PRES_MEASUREMENT_VALUE_ID,
        (zb_uint8_t *)&new_pres_value,
        ZB_FALSE);
    if (zcl_status != ZB_ZCL_STATUS_SUCCESS) {
      NRF_LOG_INFO("Set pressure value fail. zcl_status: %d", zcl_status);
    }
  */

 //configure_attribute_reporting();

  // nrf_gpio_pin_toggle(LED2_G);
}

/**@brief Zigbee stack event handler.
 *
 * @param[in]   bufid   Reference to the Zigbee stack buffer used to pass signal.
 */
void zboss_signal_handler(zb_bufid_t bufid) {
  zb_zdo_app_signal_hdr_t *p_sg_p = NULL;
  zb_zdo_app_signal_type_t sig = zb_get_app_signal(bufid, &p_sg_p);
  zb_ret_t status = ZB_GET_APP_SIGNAL_STATUS(bufid);
  zb_bool_t comm_status;
NRF_LOG_INFO("SIGNAL -> : %d", sig);
  /* Update network status LED */
  // zigbee_led_status_update(bufid, ZIGBEE_NETWORK_STATE_LED); // This function uses bsp (we cant use BSP and not use the gpiote drive

  // nrf_gpio_pin_toggle(USER_LED);

  // led_blink(1);

  switch (sig) {

    static uint8_t commissioning_retries = 0;
  case ZB_BDB_SIGNAL_DEVICE_REBOOT: // Signal that joinging the network was succesful
    if (status == RET_OK) {
      if (ZB_JOINED() == true) {
        NRF_LOG_INFO("JOINED NETWORK SUCCESSFULLY -> Status: %d", status);
        //nrf_gpio_pin_clear(ZIGBEE_NETWORK_STATE_LED); // LED on
```

```
      NRF_LOG_INFO("Network-Role: %d", zb_get_network_role());
      // nrf_gpio_pin_toggle(USER_LED);
      // joined_network=true;
    } else {
      NRF_LOG_INFO("JOINING NETWORK FAILED -> Status: %d, PRESS BUTTON TO TRY AGAIN", status);
      // joined_network=false;
    }
  } else {
    NRF_LOG_INFO("NO NETWORK FOUND!");
    // retry_join(ZB_NWK_LEAVE_TYPE_RESET);
    zb_bdb_set_legacy_device_support(1); // Zigbee legacy mode (Zigbee PRO)
    // nrf_gpio_pin_set(ZIGBEE_NETWORK_STATE_LED); // LED off
    led_blink(4);
  }
case ZB_BDB_SIGNAL_STEERING:
  if (status == RET_OK) {
    /* Update network status LED */
    // nrf_gpio_pin_clear(ZIGBEE_NETWORK_STATE_LED); // LED on
    zb_ext_pan_id_t extended_pan_id;
    char ieee_addr_buf[17] = {0};
    int addr_len;

    zb_get_extended_pan_id(extended_pan_id);
    addr_len = ieee_addr_to_str(ieee_addr_buf, sizeof(ieee_addr_buf), extended_pan_id);
    if (addr_len < 0) {
      strcpy(ieee_addr_buf, "unknown");
    }

    NRF_LOG_INFO("Joined network successfully (Extended PAN ID: %s, PAN ID: 0x%04hx)", NRF_LOG_PUSH(ieee_addr_buf), ZB_PIBCACHE_PAN_I
    ret_code_t err_code = app_timer_start(zb_app_timer, APP_TIMER_TICKS(30000), NULL);
    APP_ERROR_CHECK(err_code);
    zb_zdo_pim_set_long_poll_interval(300000);
    //NRF_LOG_INFO("configure_attribute_reporting()_INIZIO");
    configure_attribute_reporting();
    //NRF_LOG_INFO("configure_attribute_reporting()_FINE");
    led_blink(2);
    // nrf_gpio_pin_toggle(USER_LED);
  } else {
    /* Update network status LED */
    // nrf_gpio_pin_set(ZIGBEE_NETWORK_STATE_LED); // LED off
    NRF_LOG_ERROR("Failed to join network (status: %d)", status);

    // nrf_gpio_pin_toggle(USER_LED);

    // Retry n times
    if (commissioning_retries < 3) {
      comm_status = bdb_start_top_level_commissioning(ZB_BDB_NETWORK_STEERING);
      ZB_COMM_STATUS_CHECK(comm_status);
      commissioning_retries++;
    } else if (commissioning_retries >= 3) {
      commissioning_retries = 0;
      led_blink(5);
    }
  }
  break;

case ZB_COMMON_SIGNAL_CAN_SLEEP: {
  zb_zdo_signal_can_sleep_params_t *can_sleep_params = ZB_ZDO_SIGNAL_GET_PARAMS(p_sg_p, zb_zdo_signal_can_sleep_params_t);
  NRF_LOG_INFO("zb_sleep_now %d",Cont++);
  NRF_LOG_INFO("Can sleep for %ld ms", can_sleep_params->sleep_tmo);
  //       nrf_pwr_mgmt_run();
  NRF_LOG_INFO("ZB_TIMER %d", ZB_TIME_BEACON_INTERVAL_TO_MSEC(ZB_TIMER_GET()));

  //nrf_gpio_cfg_input(13,NRF_GPIO_PIN_NOPULL);
  //nrf_gpio_cfg_input(LED2_B,NRF_GPIO_PIN_NOPULL);
  //nrf_gpio_cfg_input(ZIGBEE_NETWORK_STATE_LED,NRF_GPIO_PIN_NOPULL);

  zb_sleep_now();
  break;
}

case ZB_ZDO_SIGNAL_LEAVE:
  /* Update network status LED */
  // bsp_board_led_off(led_idx);
  NRF_LOG_INFO("Left the network");
  // nrf_gpio_pin_set(ZIGBEE_NETWORK_STATE_LED);
  break;
case ZB_ZDO_SIGNAL_SKIP_STARTUP:
  NRF_LOG_INFO("Started ZBOSS stack without commissioning.");
  /* Update network status LED */
  // nrf_gpio_pin_clear(ZIGBEE_NETWORK_STATE_LED); // LED on
  // bdb_start_top_level_commissioning(0);
```

```
      The application is doing some other hardware initialization before the start of commissioning and prefers to use the stack multit
      The application does not start commissioning at power-up. Instead, it waits for a user action at every power-up (for example, a b
    */

  default:
    /* Call default signal handler. */
    ZB_ERROR_CHECK(zigbee_default_signal_handler(bufid));
    break;
  }

  if (bufid) {
    zb_buf_free(bufid);
  }
}

/**@brief Function for initializing all clusters attributes.
 */
void multi_sensor_clusters_attr_init(void) {
  /* Basic cluster attributes data */
  m_dev_ctx.basic_attr.zcl_version = ZB_ZCL_VERSION;
  m_dev_ctx.basic_attr.app_version = SENSOR_INIT_BASIC_APP_VERSION;
  m_dev_ctx.basic_attr.stack_version = SENSOR_INIT_BASIC_STACK_VERSION;
  m_dev_ctx.basic_attr.hw_version = SENSOR_INIT_BASIC_HW_VERSION;

  /* Use ZB_ZCL_SET_STRING_VAL to set strings, because the first byte should
   * contain string length without trailing zero.
   *
   * For example "test" string wil be encoded as:
   *   [(0x4), 't', 'e', 's', 't']
   */
  ZB_ZCL_SET_STRING_VAL(m_dev_ctx.basic_attr.mf_name,
      SENSOR_INIT_BASIC_MANUF_NAME,
      ZB_ZCL_STRING_CONST_SIZE(SENSOR_INIT_BASIC_MANUF_NAME));

  ZB_ZCL_SET_STRING_VAL(m_dev_ctx.basic_attr.model_id,
      SENSOR_INIT_BASIC_MODEL_ID,
      ZB_ZCL_STRING_CONST_SIZE(SENSOR_INIT_BASIC_MODEL_ID));

  ZB_ZCL_SET_STRING_VAL(m_dev_ctx.basic_attr.date_code,
      SENSOR_INIT_BASIC_DATE_CODE,
      ZB_ZCL_STRING_CONST_SIZE(SENSOR_INIT_BASIC_DATE_CODE));

  m_dev_ctx.basic_attr.power_source = SENSOR_INIT_BASIC_POWER_SOURCE;

  ZB_ZCL_SET_STRING_VAL(m_dev_ctx.basic_attr.location_id,
      SENSOR_INIT_BASIC_LOCATION_DESC,
      ZB_ZCL_STRING_CONST_SIZE(SENSOR_INIT_BASIC_LOCATION_DESC));

  m_dev_ctx.basic_attr.ph_env = SENSOR_INIT_BASIC_PH_ENV;

  /* Identify cluster attributes data */
  m_dev_ctx.identify_attr.identify_time = ZB_ZCL_IDENTIFY_IDENTIFY_TIME_DEFAULT_VALUE;

  /* Temperature measurement cluster attributes data */
  m_dev_ctx.temp_attr.measure_value = ZB_ZCL_ATTR_TEMP_MEASUREMENT_VALUE_UNKNOWN;
  m_dev_ctx.temp_attr.min_measure_value = ZB_ZCL_ATTR_TEMP_MEASUREMENT_MIN_VALUE_MIN_VALUE;
  m_dev_ctx.temp_attr.max_measure_value = ZB_ZCL_ATTR_TEMP_MEASUREMENT_MAX_VALUE_MAX_VALUE;
  m_dev_ctx.temp_attr.tolerance = ZB_ZCL_ATTR_TEMP_MEASUREMENT_TOLERANCE_MAX_VALUE;

  /* Pressure measurement cluster attributes data */
  m_dev_ctx.pres_attr.measure_value = ZB_ZCL_ATTR_PRES_MEASUREMENT_VALUE_UNKNOWN;
  m_dev_ctx.pres_attr.min_measure_value = ZB_ZCL_ATTR_PRES_MEASUREMENT_MIN_VALUE_MIN_VALUE;
  m_dev_ctx.pres_attr.max_measure_value = ZB_ZCL_ATTR_PRES_MEASUREMENT_MAX_VALUE_MAX_VALUE;
  m_dev_ctx.pres_attr.tolerance = ZB_ZCL_ATTR_PRES_MEASUREMENT_TOLERANCE_MAX_VALUE;
}

// From https://devzone.nordicsemi.com/f/nordic-q-a/49156/zigbee-attribute-reporting-not-reporting-as-configured
void configure_attribute_reporting(void) {
  zb_zcl_status_t zcl_status = 0;
  zb_ret_t zb_status = RET_OK;
  zb_zcl_reporting_info_t temp_rep_info;
  memset(&temp_rep_info, 0, sizeof(temp_rep_info));

  temp_rep_info.direction = ZB_ZCL_CONFIGURE_REPORTING_SEND_REPORT;
  temp_rep_info.ep = MULTI_SENSOR_ENDPOINT;
  temp_rep_info.cluster_id = ZB_ZCL_CLUSTER_ID_TEMP_MEASUREMENT;
  temp_rep_info.cluster_role = ZB_ZCL_CLUSTER_SERVER_ROLE;
  temp_rep_info.attr_id = ZB_ZCL_ATTR_TEMP_MEASUREMENT_VALUE_ID;
  temp_rep_info.dst.profile_id = ZB_AF_HA_PROFILE_ID;
  temp_rep_info.u.send_info.min_interval = 30;  // 30 seconds
  temp_rep_info.u.send_info.max_interval = 600; // 5 minutes
  temp_rep_info.u.send_info.delta.s16 = 0x0032; // 0.5 degrees
```

```
  NRF_LOG_INFO("%d = zb_zcl_put_reporting_info", zcl_status);

  // zb_status = zb_zcl_start_attr_reporting(10, 0x0402, 0x0104, 0);
  NRF_LOG_INFO("Is reporting on?: %d", zcl_is_attr_reported(MULTI_SENSOR_ENDPOINT, ZB_ZCL_CLUSTER_ID_TEMP_MEASUREMENT, ZB_ZCL_CLUSTER_S
  zb_status = zb_zcl_start_attr_reporting(MULTI_SENSOR_ENDPOINT, ZB_ZCL_CLUSTER_ID_TEMP_MEASUREMENT, ZB_ZCL_CLUSTER_SERVER_ROLE, 0);
  NRF_LOG_INFO("%d = zb_zcl_start_attr_reporting", zcl_status);

  if (zb_status = ZB_ZCL_STATUS_SUCCESS) {
    // led_blink(5);
  }
}

zb_void_t report_attribute_cb(zb_uint16_t addr, zb_uint8_t ep, zb_uint16_t cluster_id,
    zb_uint16_t attr_id, zb_uint8_t attr_type, zb_uint8_t *value) {
  NRF_LOG_INFO("addr      : 0x%04hx", addr);
  NRF_LOG_INFO("ep        : %d", ep);
  NRF_LOG_INFO("cluster_id: 0x%04hx", cluster_id);
  NRF_LOG_INFO("attr_id   : 0x%04hx", attr_id);
  NRF_LOG_INFO("attr_type : 0x%04hx", attr_type);
  NRF_LOG_INFO("attr_id   : 0x%04hx", attr_id);
  NRF_LOG_INFO("value     : %d", value);
}
```