Documentation to the ZigBee Systems

Presentation

Для початку, варто згадати, у чому полягає сутьпроекту, та які інструменти використовуються для вирішення цієї проблеми.

У чому полягає задача та основна ідея zigbee:

- Система комунікації, між приладами, які ії підтримують.
- Минулого разу ми розповіли, чому саме обрати ZigBee:
    - Енергоспоживання
    - Радіус дії і безпека
    - Відмовостійкість
    - Взаємна сумісність
    - Ціна

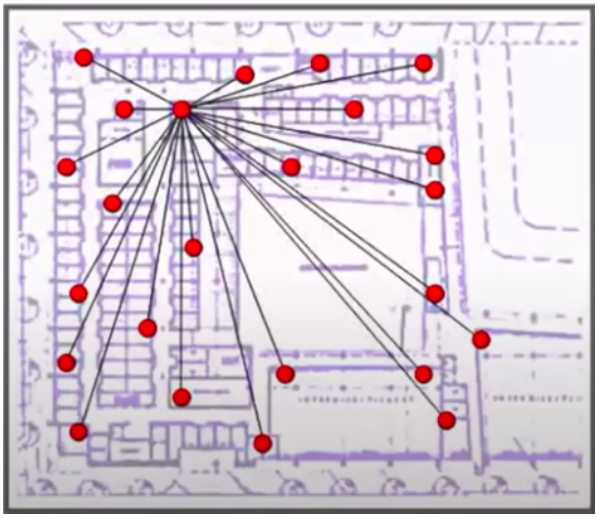
Ці критерії стає важливими у виборі системи комунікації. Зараз ZigBee набуває дуже великої популярності серед виробників. Більшість компаній, які випускають смарт девайси або ж датчики мають підтримку ZigBee. Ця мережа є наджзичайно доступною та портабельною.

Типи з’єднань

Типи з’єднань, які можуть утворювати певні системи комунікації.

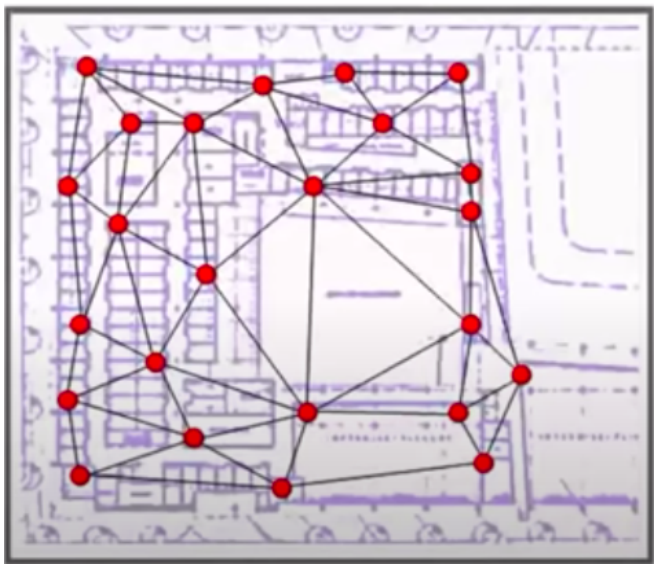
- **Star network**

  Утворена мережа ZigBee у вигляді star network, подібна до того як будуться такі системи передачі даних, як wifi чи bluetooth, передає усі дані через центральний девайс який є так званий bottleneck у системі. Це робить центральний вузол вузьким місцем для трафіку, суттєвим пунктом відмови та надійності, а обмін повідомленнями в мережевих пристроях обмежується можливістю спілкування, оскільки вони повинні спілкуватися через центральний пристрій, навіть коли джерело та пристрої призначення можуть знаходитись поруч із кожним іншим



- **Mesh network**

  Зараз напротивагу star network розглянемо кращий варіант побудови мережі - mesh network. Кожна точка мережі стає точкою ретрансляції для з'єднання до решти, а один пристрій потенційно має кілька шляхів до своїх сусідів. У такому ідеальному сценарії побудови мережі, ніхто має бути центральною точкою відмови, оскільки кожен вузол має більший вибір посилань, на які вони можуть розраховувати, щоб доставити повідомлення через мережу до місця призначення



Зауважимо, що на відміну від типу ретрансляторів або точок доступу розширювача діапазону (extenders), знайдених у мережах Wi-Fi, ці проміжні вузли в mesh network ZigBee можуть бути самими джерелами або місцями призначення віддачі чи прийому повдомлень - тобто даних.
Коли мова йде про зв'язок на рівні програми, максимізується їх ефективність у зменшенні потреби у непотрібних пристроях, використовуючи топологію mesh network. Самі мережі ZigBee максимізують ефективний діапазон зв'язку (в залежності від сили з’єднання будується мережа). Вони цим знижують вартість загального рішення, не вимагаючи спеціалізованих повторювачів або центральних точок доступу, а також покращують надійність та швидкість зв'язку

Архітектура системи ZigBee

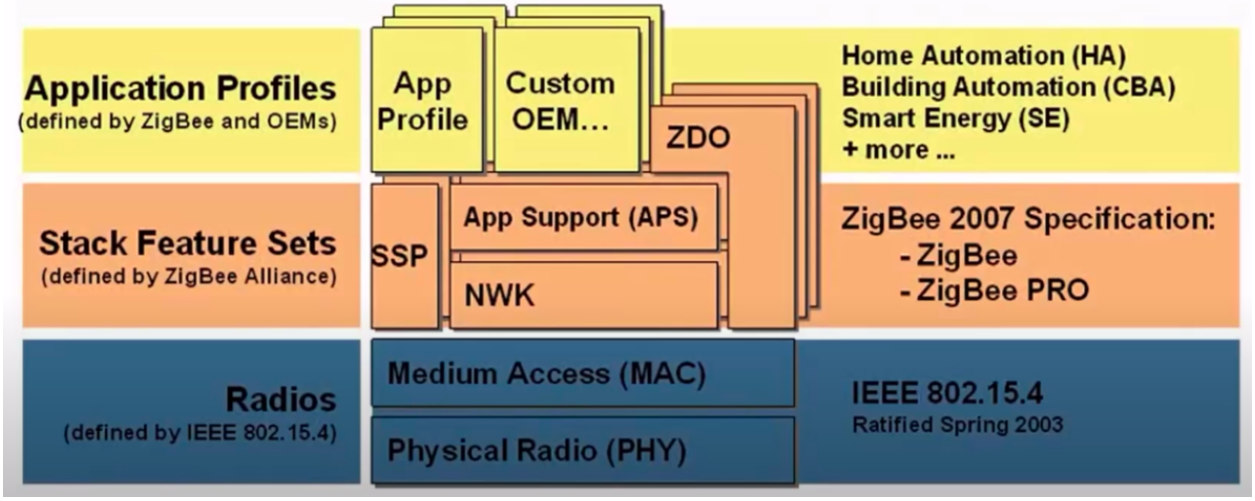
На нижньому рівні ми маємо **mac** - контроль доступу до середовища (**medium access control**).

- Відповідає за забезпечення надійного зв'язку між вузлом та його прямими сусідами. Він відповідає за генерацію маяків (**beacons**) і контроль синхронізації пристрою через кадри маяків

І нарешті, фізичний рівень (**physical layer**), який стосується радіо та способу його доступу до ефіру та передачі даних

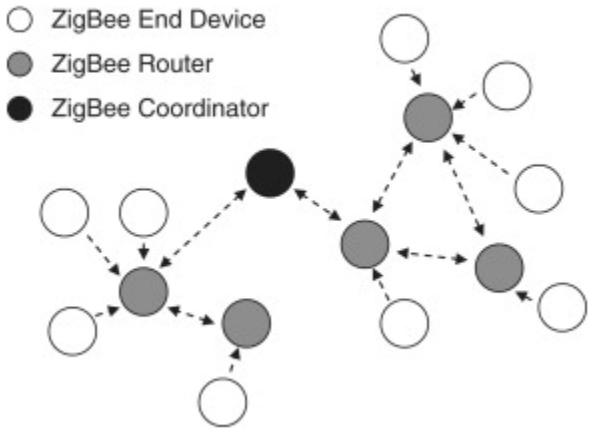
На найнижчому рівні є теж саме раді, а вище цього - контроль доступу до середовища або MAC шар. Шар MAC у ZigBee базується на протоколі IEEE 802 15.4

MAC розшифровується як Media Access Control - це другий рівень. В IoT у є певні проблеми з MAC. Потрібно розробити протоколи, які дозволять економити енергію, яка може передавати дані в спільному каналі, і виникає купа проблем
Одна дуже ключова специфікація - це та, якою керує IEEE. І специфікаційний номер, який вони використовують, - 802.15.4. І що це таке, це специфікація класу бездротових бездротових протоколів MAC



Типи вузлів

ZigBee має три типи базових вузлів - це координатор ZigBee або ZC. Роутер ZigBee, який ми скорочуємо як ZR, та кінцевий пристрій ZigBee - ZED.
Різниця між цими типами вузлів або пристроїв зводиться головним чином до того, як вони взаємодіють з іншими вузлами в мережі



ZC (coordinator)
По-перше, координатор ZigBee. Координатор, по суті, є маршрутизатором з деякою додатковою функціональністю, він відповідає за формування PAN - персональної мережі, і тому він також є першим хто приєднується та формує структуру. У даній мережі може бути лише один координатор. У будь-який момент часу після того, як координатор формує мережу, ніхто інший не може приєднатися до мережі як координатор

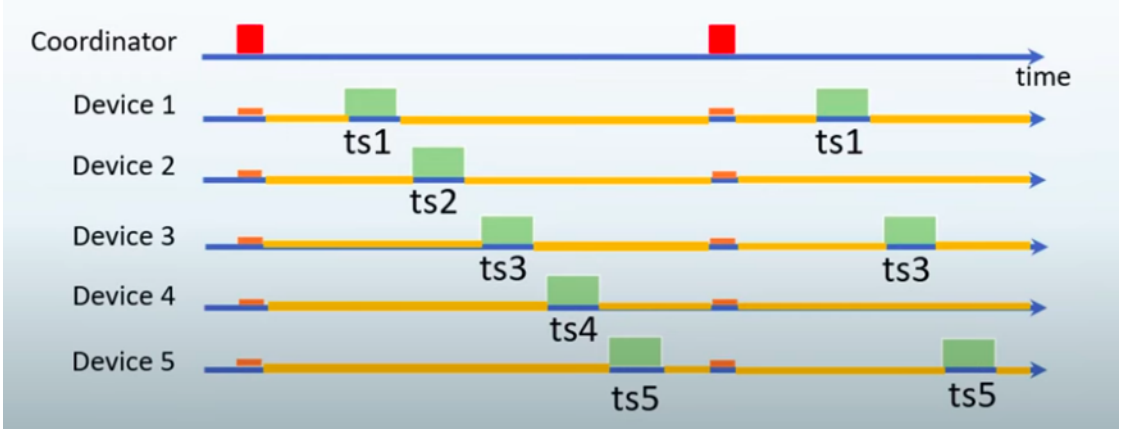
ZR (router)
Далі, давайте розглянемо тип пристрою роутера. Роутера має можливість не тільки надсилати та отримувати власні пакети, але й ретранслявати інші вузли в мережі

ZED (end device)
Кінцеві пристрої - це пристрої, які не беруть участі в жодній маршрутизації. Єдина концепція маршрутизації полягає в тому, що вони повинні надсилати дані батькам

Channel access

GTS

Уся мережа працює за рахунок того, що координотор визначає певний проміжок часу для кожного приєднаного девайсу аби той міг переслати дані. Це називається guaranteed time slot (**gts**)



Синхронність

Щоб правильно призначити GTS, координатору потрібно забезпечити синхронізацію всіх пристроїв у мережі. Для досягнення цього координатор періодично передає повідомлення, що називається маяком (або **beacon**), як тільки пристрої отримають цей маяк, вони налаштують або синхронізують свої годинники з годинником координатора. Основною причиною цього є те, що маяк містить інформацію про те, коли кожен пристрій може передавати дані. Отже, зараз усі пристрої заздалегідь знають, коли вони можуть це робити. Потім кожен пристрій чекає своєї черги і передає свої дані в певний часовий інтервал, визначений координатором. Ця ж операція з часом повторюється

- Ця методологія забезпеучє low-power mode.

Також девайси можуть бути не синхронізовані. Вони спочатку сканують чи є кому передавати дані, і надсилають request. Якщо є відповідь на нього (MAC ACKnowledgment) - дані пересилаються, як ні, тоді девайс відступає та пробує знову за певний проміжок часу

-

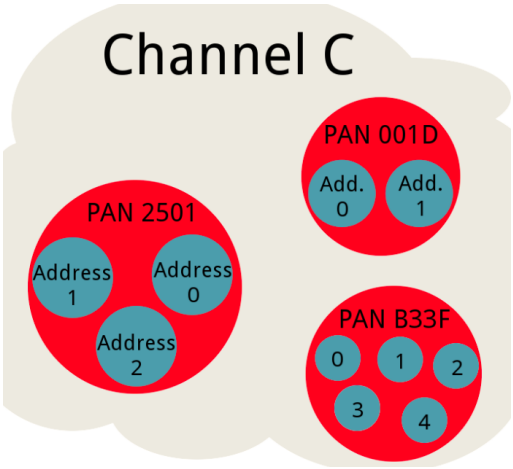
Мережа PAN

Тепер поговоримо про ідентифікатор **PAN** - мережа PAN або personal area network відділена від інших мереж за допомогою свого ідентифікатора pan, це 16-розрядний ідентифікатор, яким будуть спільно користуватися всі вузли в одному PAN

- Ідентифікатор розміщується в заголовку низькорівневого рівня MAC у кожному вихідному пакеті і дозволяє пристроям, які отримують пакет, фільтрувати повідомлення, які не відносяться до їх мережі

Якщо конфлікт PAN ID коли-небудь трапляється (інша мережа із таким же PAN ID надсилає дані), стек насправді може виявити такий конфлікт і може автоматично оновити свій PAN ID

- Звичайний короткий 16-розрядний ідентифікатор передається по повітрю у всіх пакетах, оскільки він короткий і простий. 64-розрядний розширений PAN ID (EPID) рідко передається по повітрю. Розширений ідентифікатор PAN також унікальний для кожного PAN. В основному він використовується як критерій резервного копіювання, коли 16-бітового ідентифікатора PAN недостатньо, щоб завжди відрізняти одну мережу від іншої



Eui-64 / Network Address

Самі вузли відрізняються від інших за індивідуальною адресою вузла. Вузол має коротку адресу та довгу адресу. Довга адреса - це MAC-адреса, призначена протоколом IEEE, або її ще називають **eui-64**, це 64-бітна адреса

При передачі даних використовується набагато коротша 16-бітна адреса - **Network Address**, яка називається ідентифікатором вузла і є унікальною в мережі. Вона визначається, коли вузол потрапляє в мережу

Endpoint

Також існує поняття endpoint. Ця 8-розрядна **endpoint** точка визначає кожну програму, що працює на вузлі ZigBee

- (у подальному, якщо надавати пристрою змогу чи тати температуру, вологу, включати світло, ми маємо використовувати ndpoint для зазначення програми)

Вузол ZigBee може містити одну або кілька кінцевих точок. З одним фізичним пристроєм можуть існувати різні профілі програм та декілька логічних пристроїв, крім того кожна кінцева точка може підтримувати функціональність одного або декількох кластерів. **cluster ID** - це 16-бітове значення, що використовується як тип повідомлення для конкретного додатку

Additional Applicationl Layer Information

Application layer

The ZigBee application layer is divided into 4 parts:

- the Application Support Layer (APS)
- the application framework layer
- the ZigBee Device Objects layer (ZigBee Device Objects, ZDO)
- the management platform of the ZigBee device objects

Application framework layer (AF)

- The port mainly deals with receiving data and sending data
- Specifies a series of standardized data types, descriptors to assist service discovery, frame formats for transmitting data

- When using an endpoint, you must first use **afRegister** to register an endpoint, otherwise, the endpoint will not receive data
- The application layer sends wireless data through the **AF_DataRequest** function
- After receiving the data, the **afIncomingData** function sends the data to the corresponding port

*extern void afIncomingData(aps_FrameFormat_t *aff, zAddrType_t *SrcAddress,*
uint8 LinkQuality, byte SecurityUse, uint32 timestamp);

APS: Application Support Sublayer

- The main function of the APS sublayer is to save the binding table and transfer information between corresponding devices

The application support sublayer is divided into two parts:

- APS data entity (**APSDE**) (receive device requests and save device information)
- APS management entity (**APSME**) (contains a database of management objects)

ZDO (Zigbee device object layer)

ZDO is a special application layer endpoint (Endpoint), ZDO occupies the 0 terminal (Endpoint0)

- The application can communicate with other layers of the Zigbee stack through endpoint 0
- Endpoint 255: used to broadcast to all endpoints.

ZDO defines the role of a device in the network (coordinator, router, or terminal node), initiates or responds to binding and discovery requests, and establishes a security relationship between network devices

- An endpoint corresponds to a task, and a task has a unique task number
- ZDO_RegisterForZDOMsg() registers events in the corresponding task

The sending and receiving of wireless data:

- The sending end: first send the wireless data packet to the lower layer, until the physical layer, the physical layer sends the data packet in the form of radio waves
- Receiving end: When a wireless data packet arrives, the lowest layer of the node, that is, the physical layer, will receive the wireless data packet, and then upload it layer by layer until the application layer

Events / messages: (SYS_EVENT_MSG)

- A task can have 16 events, and each event can have 256 messages, from **0x00-0xff**

NWK: Network layer

The network layer provides interfaces for MAC and application layer and manages network structure and routing (routing is to choose a path to transmit data to the destination device)

- ensures that the MAC layer has the ability to work normally

The network layer includes two service entities:

- the network-layer data entity (**NLDE**)
- the network management entity (**NLDE**)

MAC: data link layer

Responsible for providing a reliable communication link between a node and its direct neighbors

- responsible for generating beacon frames and controlling device synchronization through beacon frames

MAC layer mainly provides two kinds of services:

- MAC **data service**
- MAC **layer management service**

PHY: physical layer

The physical layer is mainly responsible for data modulation and demodulation, sending and receiving

- radio frequency module is on this layer

PHY provides two kinds of services:

- **PHY data service**
- **PHY management**

Main workflow of the Program

1. system startup
2. driver initialization
3. OSAL initialization and startup
4. entering the task cycle stages, which will be analyzed in detail below

After the system is powered on, the hardware is initialized by executing the ZSEG int **main()** function of ZMain.c in the ZMain folder:

- includes the general interrupt **osal_int_disable(INTS_ALL)**
- the initialization board hardware setting **HAL_BOARD_INIT()**
- the initialization I/O port InitBoard(**OB_COLD**)

- initialize the HAL layer driver **HalDriverInit()**
- initialize the non-volatile memory **sal_nv_init(NULL)**
- initialize the MAC layer **ZMacInit()**
- allocate a 64-bit address **zmain_ext_addr()**
- initialize the operating system **osal_init_system()**

When the above initialization is successfully completed, execute the **osal_start_system()** function to start the OSAL system

---

Code

GitHub: https://github.com/SlavkoPrytula/home-automation-system

- **Coordinator branch - coordinator**
- **Web - app**
- **CC-tool branch - flasher**
- **Router branch - router + enddevice**

На гілці cc-tool можна знайти папку із cc-tool, яки використовується для прошивки самої плати. Насамперед, варто зауважити, що сам cc-tool підтримуються на Linux. Для Windows існують робочі альтернативи.

Для перепрошивки коду, потрбіно його скормпілювати у .hex файл та розмістити у зручній дерикторії.
Даля перейти у папку **./CC-tool/cc-tool**
І запустити: sudo **./cc-tool -e -w ../<firmware dir>**

На нашому GitHub мможна знайти робочі версії прошивок. Вони знаходяться у гілці **router**, далі Firmwares:

- May_2021 - last update

Сам проект знаходиться у гілці **router**, у папках
Projects/zstack/HomeAutomation/SampleLight/

В папці source можна знайти основний код для програми. (zcl_samplelight.c)

```
static zclGeneral_AppCallbacks_t zclSampleLight_CmdCallbacks =
{
  zclSampleLight_BasicResetCB,                          // Basic
  zclSampleLight_IdentifyCB,
  zclSampleLight_IdentifyQueryRspCB,                    // Ident
  zclSampleLight_OnOffCB,
  NULL
```

zclSampleLight_Init містить старт програми та ініціалізацю усіх тасків по їх id
Також мітиться там ініціалізація на запит для приєднання до координотора

```
bdb_StartCommissioning(BDB_COMMISSIONING_MODE_NWK_STEERING | BDB_COMMISSIONING_MODE_FINDING_BINDING);
```

zclSampleLight_event_loop - посстійне виконання команд посліовно.
Тут міститься всі операції викликів та слідкування за взамодією із контролером.

Попри це важливою є ініціалізація Bind Request, яка дозволяє приєднуватися новому End device

```
  // Initiate an End Device Bind Request, this bind request will
  // only use a cluster list that is important to binding.
  zcl_dstAddr.addrMode = afAddr16Bit;
  zcl_dstAddr.addr.shortAddr = 0;   // Coordinator makes the match
  ZDP_EndDeviceBindReq( &zcl_dstAddr, NLME_GetShortAddr(),
                        SAMPLELIGHT_ENDPOINT,
                        ZCL_HA_PROFILE_ID,
                        ZCLSAMPLELIGHT_BINDINGLIST, bindingInClusters,
                        0, NULL,   // No Outgoing clusters to bind
                        TRUE );
```

Також там є відповідне приєднання до координатора

```
bdb_StartCommissioning(
                        BDB_COMMISSIONING_MODE_NWK_FORMATION |
                                BDB_COMMISSIONING_MODE_NWK_STEERING |
                                        BDB_COMMISSIONING_MODE_FINDING_BINDING |
                                                BDB_COMMISSIONING_MODE_INITIATOR_TL
                                                        );
```

При надсиланні запиту на приєднання виконується відлік, який визначить чи координатор доступний, якщо ні, тоді він знову надішле request на приєднання

```
if ( events & SAMPLELIGHT_IDENTIFY_TIMEOUT_EVT )
{
        if ( zclSampleLight_IdentifyTime > 0 )
        {
                zclSampleLight_IdentifyTime--;
        }
        zclSampleLight_ProcessIdentifyTimeChange();
        return ( events ^ SAMPLELIGHT_IDENTIFY_TIMEOUT_EVT );
}
```

zclSampleLight_HandleKeys - функція для перевірки натискання на кнопки на платі

```
static void zclSampleLight_HandleKeys( byte shift, byte keys )
{
//  UI_MainStateMachine(keys);

//  HalLedSet( HAL_LED_ALL, HAL_LED_MODE_BLINK);

 (void)shift;  // Intentionally unreferenced parameter

 if ( keys & HAL_KEY_SW_6 )
 {
        LED(1, 7);
        osal_start_timerEx(zclSampleLight_TaskID, SAMPLEAPP_EVT_LONG, 2000);
 }
 else
 {
        LED(0, 7);
        osal_stop_timerEx(zclSampleLight_TaskID, SAMPLEAPP_EVT_LONG);
 }
```

zclSampleLight_ReportOnOff - повертає чи роутер зараз включений чи виключений

```
void zclSampleLight_ReportOnOff(void) {
  const uint8 NUM_ATTRIBUTES = 1;

  zclReportCmd_t *pReportCmd;

  pReportCmd = osal_mem_alloc(sizeof(zclReportCmd_t) +
                              (NUM_ATTRIBUTES * sizeof(zclReport_t)));
  if (pReportCmd != NULL) {
    pReportCmd->numAttr = NUM_ATTRIBUTES;

    pReportCmd->attrList[0].attrID = ATTRID_ON_OFF;
    pReportCmd->attrList[0].dataType = ZCL_DATATYPE_BOOLEAN;
    pReportCmd->attrList[0].attrData = (void *)(&RELAY_STATE);

    zclSampleLight_DstAddr.addrMode = (afAddrMode_t)Addr16Bit;
    zclSampleLight_DstAddr.addr.shortAddr = 0;
    zclSampleLight_DstAddr.endPoint = 1;

    zcl_SendReportCmd(SAMPLELIGHT_ENDPOINT, &zclSampleLight_DstAddr,
                      ZCL_CLUSTER_ID_GEN_ON_OFF, pReportCmd,
                      ZCL_FRAME_CLIENT_SERVER_DIR, false, SeqNum++);
  }

  osal_mem_free(pReportCmd);
}
```

Тут складається сам пакед для надсилання

```
pReportCmd->attrList[0].attrID = ATTRID_ON_OFF;
pReportCmd->attrList[0].dataType = ZCL_DATATYPE_BOOLEAN;
pReportCmd->attrList[0].attrData = (void *)(&RELAY_STATE);

zclSampleLight_DstAddr.addrMode = (afAddrMode_t)Addr16Bit;
zclSampleLight_DstAddr.addr.shortAddr = 0;
zclSampleLight_DstAddr.endPoint = 1;
```

І відправляється (бачимо, що для відповідного Cluster ID)

```
zcl_SendReportCmd(SAMPLELIGHT_ENDPOINT, &zclSampleLight_DstAddr,
                  ZCL_CLUSTER_ID_GEN_ON_OFF, pReportCmd,
                  ZCL_FRAME_CLIENT_SERVER_DIR, false, SeqNum++);
```

zclSampleLight_ReportTemp (beta version) - функція, яка має надсилати панети даних про температуру із датчика. Наразі ще не вийшло повністю розібратися із його підключенням до плати.

У файлі zcl_samplelight_data.c знаходяться усі ініціалізації Claster IDs, також там присутні дані про координатор

Дані про координатор

```
// Basic Cluster
const uint8 zclSampleLight_HWRevision = SAMPLELIGHT_HWVERSION;
const uint8 zclSampleLight_ZCLVersion = SAMPLELIGHT_ZCLVERSION;
const uint8 zclSampleLight_ManufacturerName[] = { 10, 'S','i','m','p','l','e','L','i','n','k' };
const uint8 zclSampleLight_ModelId[] = { 8, 'R','o','u','t','e','r','0','1' };
const uint8 zclSampleLight_DateCode[] = { 8, '2','0','2','2','0','4','2','0' };
const uint8 zclSampleLight_PowerSource = POWER_SOURCE_MAINS_1_PHASE;

uint8 zclSampleLight_LocationDescription[17];
uint8 zclSampleLight_PhysicalEnvironment;
uint8 zclSampleLight_DeviceEnable;
```

Cluster IDs

```
  ZCL_CLUSTER_ID_GEN_BASIC,
  { // Attribute record
    ATTRID_BASIC_ZCL_VERSION,
    ZCL_DATATYPE_UINT8,
    ACCESS_CONTROL_READ,
    (void *)&zclSampleLight_ZCLVersion
  }
},
```

- Використовується при пересиланні даних про **zclSampleLight_ZCLVersion**

```
{
  ZCL_CLUSTER_ID_GEN_BASIC,
  { // Attribute record
    ATTRID_BASIC_MANUFACTURER_NAME,
    ZCL_DATATYPE_CHAR_STR,
    ACCESS_CONTROL_READ,
    (void *)zclSampleLight_ManufacturerName
  }
},
```

- Використовується при пересиланні даних про **zclSampleLight_ManufacturerName**

Також файл містить Cluster IDs для пересилання даних про температуру

```
// *** Temperature Measurement  ***
{
  ZCL_CLUSTER_ID_MS_TEMPERATURE_MEASUREMENT,
  { // temperature value
    ATTRID_MS_TEMPERATURE_MEASURED_VALUE,
    ZCL_DATATYPE_INT16,
    ACCESS_CONTROL_READ | ACCESS_REPORTABLE,
    (void *)&zclSampleLight_MeasuredValue
  }
},

{
  ZCL_CLUSTER_ID_MS_TEMPERATURE_MEASUREMENT,
  { // min temperature
    ATTRID_MS_TEMPERATURE_MIN_MEASURED_VALUE,
    ZCL_DATATYPE_INT16,
    ACCESS_CONTROL_READ | ACCESS_REPORTABLE,
    (void *)&zclSampleLight_MinMeasuredValue
  }
},
{
  ZCL_CLUSTER_ID_MS_TEMPERATURE_MEASUREMENT,
  { // max tempeerature
    ATTRID_MS_TEMPERATURE_MAX_MEASURED_VALUE,
    ZCL_DATATYPE_INT16,
    ACCESS_CONTROL_READ | ACCESS_REPORTABLE,
    (void *)&zclSampleLight_MaxMeasuredValue
  }
},
{
  ZCL_CLUSTER_ID_MS_TEMPERATURE_MEASUREMENT,
  {  // cluster version
    ATTRID_CLUSTER_REVISION,
    ZCL_DATATYPE_UINT16,
    ACCESS_CONTROL_READ | ACCESS_REPORTABLE,
    (void *)&zclSampleLight_clusterRevision_all
  }
},
```

Для читання/передачі даних для координатора ми можемо використати ACCESS_CONTROL_READ | ACCESS_CONTROL_WRITE відповідно. Або можемо робити і те і інше, це залежить від вимог того, що ми робимо. Для передачі температури нам важливо її надсилати на сам координатор із End device через router