基于 STM32WB 系列创建制造特定集群

引言

本应用笔记旨在指导最终用户在 STM32WB 系列上实现特定的 ZCL（Zigbee®集群库）制造。

Exegin ZSDK（Zigbee 软件设计套件）包含适于大多数现有集群的模板，可提供广泛的功能。但某些应用仍需开发自定义集群模板。本文档描述了此类自定义集群模板的开发过程，以及新 ZCL 集群的构建，构建方式与 Exegin ZSDK 集群的相同。

其中，假设最终用户熟悉一般 Zigbee®组网[R1]、Exegin ZSDK 协议栈参考[R3]以及 Exegin ZCL 集群模板的使用[R5]（见表 1. 参考文档）。

1 概述

本文档适用于基于 ARM®的 STM32WB 系列器件。

注意： *Arm 是 Arm Limited（或其子公司）在美国和/或其他地区的注册商标。*

2 参考文档

下面的资源是公开的，可以从意法半导体的网站或第三方网站上获得。

表 1. 参考文档

参考编号	文件标题	文档编号
[R1]	《Zigbee PRO 规范》第 22 版：www.zigbeealliance.org/(1)	05-2374-22
[R2]	《Zigbee 集群库》第 7 版：www.zigbeealliance.org/(1)	07-5123-07
[R3]	《ZSDK API 在 STM32WB 系列上实现 ZigBee®》(2)	AN5500
[R4]	《Zigbee 智能能源标准》1.4 版：www.zigbeealliance.org/(1)	07-5356-21
[R5]	《如何在 STM32WB 系列上使用 ZigBee®集群模板》(2)	AN5498

1. *该 URL 属于第三方。它在文档发布时处于活动状态，但意法半导体对 URL 或参考材料的任何变更、转移或停用不承担责任。*
2. *可在 www.st.com 获得。如需详细信息，请与意法半导体联系。*

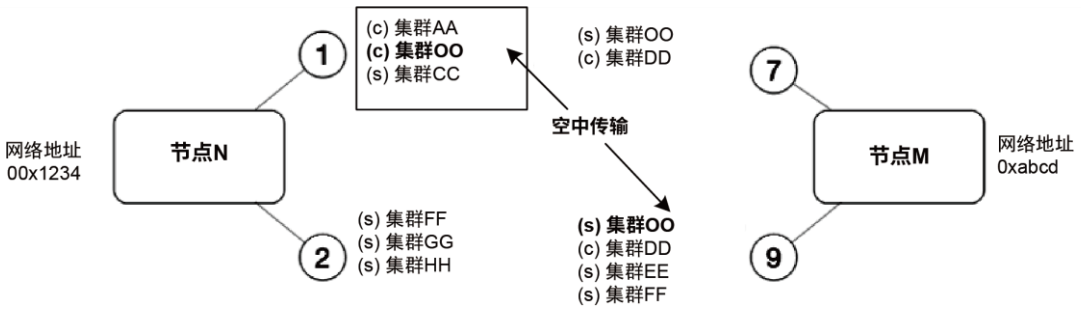
3 ZCL 集群架构

在介绍实现新 ZCL 集群的机制之前，有必要回顾一些 ZCL 的基础知识和展示一个可供全文引用的具体示例。

Zigbee 集群库（ZCL）定义了应用在节点之间通过网络和空中进行交互的机制。用于特定目的的功能被整理到“集群”中，该集合定义了一组相关的属性和指令。例如，“开/关”集群定义了可开启/关闭器件的功能。

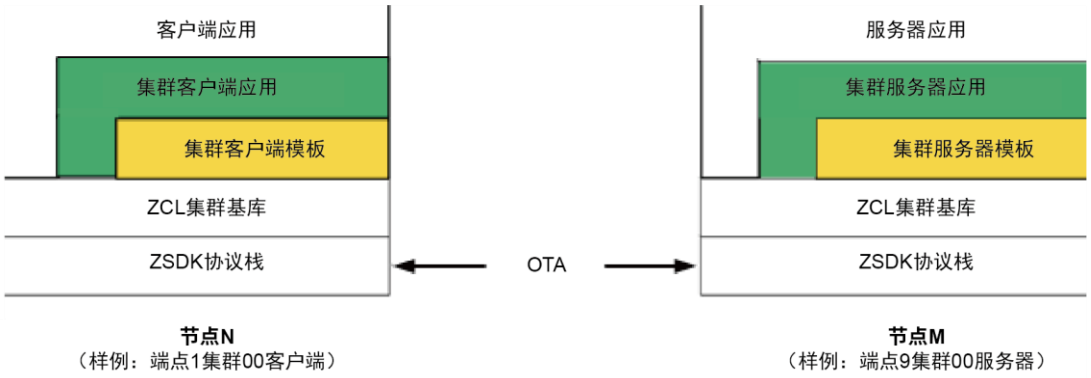
下图展示了一个网络的两个节点。节点 N 和 M 二者都支持位于端点 1 和 9 上的集群 OO（例如：开/关）。该功能在客户端和服务器端之间进行拆分。在“开/关”样例中，开关可作为客户端，而灯可作为服务器。利用节点 N 端点 1 上的开关，控制节点 M 端点 9 上的灯。

图 1. 双节点网络样例



ZCL 基于 APS 消息构建，APS 消息又使用 NWK 层功能。若需更多信息，请参考[R3]。ZSDK 包括可在其上构建 ZCL 集群的通用服务。ZCL 集群模板[R5]在此类通用服务的基础上构建。下图展示了与应用和 ZSDK 协议栈相关的 ZCL 结构。

图 2. ZCL 应用结构



ZCL 集群特定功能（例如在 ZCL [R2]或 ZSE [R4]规范的集群定义中定义的功能）在集群模板中实现，如图 2. ZCL 应用结构黄色部分所示。

集群客户端或者服务器应用，可通过在模板中填充设备特定的细节来完成。例如，在节点 N 端点 1 中填充如何访问物理开关的细节，和在节点 M 端点 9 中填充一些如何访问灯泡的细节。

4 属性端到端操作顺序

如下所示的客户端应用样例通过调用 ZCL 集群基库函数 ZbZclReadReq()，从远程服务器读取属性。由此提供了需要调用的客户端集群、读取请求结构和回调函数，结果如下所示。

```
static void
read_onoff_callback(const ZbZclReadRspT *rsp, void *arg)
{
	struct application *app = (application *)arg;

	if(rsp->status == ZCL_STATUS_SUCCESS) {
		if(rsp->attr[0].status == ZCL_STATUS_SUCCESS) {
			app->remote_onoff_state = pletoh16(rsp->attr[0].value[0]);
		}
		else { /* handle error */
		…
	}
	else { /* handle error */
	…
}

.
.
.

	enum ZclStatusCodeT status;
	ZbZclReadReqT req;

	memset(&req, 0, sizeof(req));
	req.dst = ZbApsAddrBinding;
	req.attr[0] = ZCL_ONOFF_ATTR_ON_TIME;
req.count = 1;
status = ZbZclReadReq(app->client_cluster, &req, read_onoff_callback, &application)
```

调用 ZbZclReadReq()会将 ZCL 属性请求通过空中发送到服务器。该样例使用绑定，并假设节点 N 在 OnOff 集群上包含从其本地端点 1 到节点 M 端点 9 的绑定。如果服务器集群支持所请求的属性，则会在 ZCL 属性读取响应指令中返回该属性的值。如果该属性不存在，则以 UNSUPPORTED_ATTRIBUTE 状态(0x86)返回 ZCL 默认响应。属性在初始化期间添加到集群（强制属性），或使用 ZbZclAttrAppendList()进行添加。在收到 ZCL 读取属性响应时，回调会被调用。

定义属性的结构采用 struct ZbZclAttrT。属性被初始化为这些结构的列表，并提供给 ZbZclAttrAppendList()以附加到集群上。下面展示了服务器集群应用或服务器模板的样例。

```
static const struct ZbZclAttrT attr_list[] = {
	…
	{
		ZCL_ONOFF_ATTR_ON_TIME, ZCL_DATATYPE_UNSIGNED_16BIT,
		ZCL_ATTR_FLAG_NONE, 0, NULL, {0, 0}, {0, 0}
},
…
```

使用以下指令将其添加到集群中：

```
ZbZclAttrAppendList(cluster, attr_list, ZCL_ATTR_LIST_LEN(attr_list))
```

注意： *在某些情况下，客户端集群有其自己单独的属性。此时将服务器、客户端的名字对调即可，而上述的论不变。*

前面关于服务器集群应用或服务器模板的示例中，将属性回调函数设置为了 NULL。在这种情况下，ZCL 集群基库使用该属性在收到那一刻的值，对读取请求进行响应。服务器应用通过 ZCL 集群基库调用（如 ZbZclAttrIntegerWrite()）更新本地值。服务器应用先前设置的值为响应读取请求时使用的值。服务器应用通过集群基库实现了和客户端访问的解耦。

另一种方法是为服务器提供一个回调函数，如以下服务器集群应用样例中所示。

```
/* hardware register, little endian, two bytes*/
extern volatile uint8_t *on_time;

enum ZclStatusCodeT on_time_cb(struct ZbZclClusterT *clusterPtr, struct ZbZclAttrCbInfoT
*info)
{
	uint8_t *data = info->attr_data;

	switch(info->type) {
		case ZCL_ATTR_CB_TYPE_READ:
			/* read from hardware */
			data[0] = on_time[0];
			data[1] = on_time[1]
			break;

		case ZCL_ATTR_CB_TYPE_WRITE:
			/* write to hardware */
			on_time[0] = data[0];
			on_time[1] = data[1];
			break;

		case ZCL_ATTR_CB_TYPE_NOTIFY:
			on_time[0] = data[0];
			on_time[1] = data[1];
			break;
	}

	return ZCL_STATUS_SUCCESS;
}

static const struct ZbZclAttrT attr_list[] = {
	...
	{
		ZCL_ONOFF_ATTR_ON_TIME, ZCL_DATATYPE_UNSIGNED_16BIT,
		ZCL_ATTR_FLAG_NONE, 0, on_time_cb, {0, 0}, {0, 0}
	},
	…
```

在提供回调时，每当读取或写入属性时都会调用服务器应用。实现方法是：在上述属性定义中，设置 ZCL_ATTR_FLAG_CB_READ 和 ZCL_ATTR_FLAG_CB_WRITE 标志。建议应用同时设置读取标志和写入标志并提供回调来处理所示的读取和写入，或者将回调设置为 NULL 但不设置这些标志。在没有回调的情况下，该值完全在协议栈中进行管理。在设置回调和两种标志的情况下，该值完全在应用中进行管理。

尽管可行，但不建议在应用中对仅读取或仅写入请求进行处理，因为如果不仔细注意，读取和写入的值很可能不同。

一旦服务器调用完成，集群基库就准备好响应消息，并通过无线方式将其发送回客户端。在这个例子中，基库发送的响应消息内容是读的状态和值，或者是写的状态。客户端协议栈接收响应消息。客户端的集群基库处理这个响应消息，并且，如果有回调（比如上述例子中的 read_onoff_callback 函数）则将状态、和读过来的值一并包含在回调函数中，传递到上层。

5 指令端到端演示

当客户端向服务器发送命令请求时，该过程类似于第 4 节“属性端到端操作顺序”中详细介绍的端到端操作序列。然而，与使用 ZCL 基本函数发送命令不同，客户端模板提供了一个特定于命令的请求函数，能够处理该命令的 ZCL 负载。

表 2、表 3 和表 4 展示了客户端向服务器发送 GetCalendar 指令请求的演示样例。如果该操作成功，则会生成 PublishCalendar 响应返回到客户端。有关这些日历集群指令的详细信息，请参阅[R4]。

5.1 客户端指令生成

第 D.9.2.4.1.1 节[R4]定义了 GetCalendar 指令，如下表所示。

表 2. GetCalendar 指令

八位字节	数据类型	字段名称
4	UTC 时间	最早开始时间（M）
4	无符号 32 位整数	最小颁发者事件 ID（M）
1	无符号 8 位整数	日历数量（M）
1	8 位枚举	日历类型（M）
4	无符号 32 位整数	提供者 ID（M）

D.9.2.3.1.1 [R4]中定义的来自服务器的 PublishCalendar 响应如下所示。

表 3. PublishCalendar 响应 - 第一部分

八位字节	数据类型	字段名称
4	无符号 32 位整数	提供者 ID（M）
4	无符号 32 位整数	颁发者事件 ID（M）
4	无符号 32 位整数	颁发者日历 ID（M）
4	UTC 时间	开始时间（M）
1	8 位枚举	日历类型（M）

表 4. PublishCalendar 响应 - 第二部分

八位字节	数据类型	字段名称
1	无符号 8 位整数	日历时间参考（M）
1..13	八位字节字符串	日历名称（M）
1	无符号 8 位整数	季节数量（M）
1	无符号 8 位整数	周配置文件数量（M）
1	无符号 8 位整数	日配置文件数量（M）

当客户端应用决定请求日历时，会发送 Get Calendar。这在以下集群客户端应用样例中展示。

```
static void get_cal_cb (struct ZbZclCommandRspT *rsp, void *arg)
{
	struct application app = (struct application *)arg;


}

...

	enum ZclStatusCodeT status;
	struct ZbZclCalClientGetCalendarT req;

	memset(&req, 0, sizeof(req));
	req.earliestStartTime = app->calendar_start;
	req.minIssuerEventId = ZCL_INVALID_UNSIGNED_32BIT; /* all ids */
	req.numCalendars = 1;
	req.calendarType = 0x02; /* delivered and received */
	req.providerId = app->provider_id;

status = ZbZclCalClientCommandGetCalReq(cluster, ZbApsAddrBinding, &req, get_cal_cb, app);
```

如果 ZbZclCalClientCommandGetCalReq()的参数有效，则返回的状态为 ZB_STATUS_SUCCESS，并尝试发送 GetCalendar 请求，如表 2. GetCalendar 指令所示。否则，返回错误状态，而且不会发送请求。

当返回 ZB_STATUS_SUCCESS 时，回调函数将会被调用以指示客户端尝试的结果；如果结果是成功的，回调函数的参数中将包含服务器的响应消息。

该回调需要考虑规范中定义的服务器行为。对于某些 ZCL 请求指令，所需的响应为详述的响应，如对上述 GetCalendar 请求的 PublishCalendar 响应。在其他情况下，ZCL 响应主要为一种状态，有时带有附加信息。当未定义 ZCL 消息响应时，ZCL 规范要求服务器返回通用默认响应消息。

此外，由于响应通过无线传输，因此客户端请求可能存在以下状况：

- 服务器未收到，
- 服务器可能拒绝该请求或无法处理该请求，
- 或者该响应未到达客户端

回调可以检测到其中每种状况，并根据需要采取适当的操作。应用的操作取决于这些可能状况中的每一种以及应用的特定要求。

客户端可采取以下操作：

- 即发即弃
- 重试
- 或采取更激进的操作，如酌情重新加入或启动频率敏捷程序。

实际的选择完全取决于应用的要求。

下面通过前述样例，展示在回调中发送 ZbZclCalClientCommandGetCalReq()的可能客户端结果。

1. 当请求成功并且服务器响应并成功发送了空中 ZCL 消息时，rsp.status 的值为 ZCL_STATUS_SUCCESS。另外：
   a. 帧类型为“cluster”，即（rsp.hdr.frameCtrl.frameType & ZCL_FRAMECTRL_TYPE）== ZCL_FRAMETYPE_CLUSTER
   b. ZCL 指令 ID 为响应的预期指令 ID。在这种情况下，预期为 Publish Calendar：rsp.hdr.cmdId == ZCL_CAL_SVR_PUBLISH_CALENDAR
   c. 执行这些检查后，应用程序可使用提供的帮助程序函数来解析响应。在这种情况下，为了解析 Publish Calendar 响应，应用程序可使用有效负载 rsp->payload 和 rsp->length 来调用 ZbZclCalClientParsePublishCalendar()

注意： *某些 ZCL 指令响应包含嵌入式状态代码，这些代码也必须予以检查。*

2. 当请求成功，且无来自服务器的已定义 ZCL 消息响应，或者指令出现错误时，服务器将以“默认响应”消息进行响应。与前述情况一样，rsp.status 的值为 ZCL_STATUS_SUCCESS。另外：
   a. 帧类型改为 ZCL_FRAMETYPE_PROFILE
   b. 并且指令 ID 为 ZCL_COMMAND_DEFAULT_RESPONSE
   c. 可使用 ZbZclParseDefaultResponse()解析默认响应
   d. 如果默认响应状态并非 ZCL_STATUS_SUCCESS，则说明了来自服务器的指令响应失败。

注意： *ZCL 消息的帧控制的“禁用默认响应”标志通常置为 1，使得仅在无其他 ZCL 响应时，默认响应才会“成功”。然而，如果置为 0，则始终发送默认响应，即使响应为 ZCL 消息时也是如此。*

3. 如果请求传输错误或如果服务器无响应，则 rsp.status 的值为 ZCL_STATUS_FAILURE。在这种情况下，必须检查 rsp.aps_status 的值，该值包含 APS 或 NWK 层状态代码。一些样例包括：
   a. aps_status 为 ZB_APS_STATUS_NO_ACK。这表明客户端已成功发送请求，并且是在网络级收到该请求，但服务器并未在 APS 层确认收到。当服务器可以解释请求但拒绝请求时，通常会发送带有上述错误状态的默认响应。当服务器无法解释消息时，例如：
      - 服务器上不存在目标端点
      - 或者消息不符合最低安全/加密要求
      - 服务器不确认或无法确认收到
      - 未发送 APS 层确认。

      然后发生超时，并向应用返回 ZB_APS_STATUS_NO_ACK aps_status。
   b. 如果在发送请求时，客户端甚至无法到达路由中的下一跳，则 aps_status 变为 ZB_WPAN_STATUS_NO_ACK。协议栈有几个层次都内置了重传机制，如果出现这种 aps_status，则原因很可能是客户端完全失去了与网络的连接。

5.2 服务器端指令接收和处理

当客户端无线发送了 GetCalendar 请求并且服务器收到该请求时，协议栈收到消息（参见第 3 节中的节点 M），通过 NWK 和 APS 层传递到 ZCL 集群基库。当应用创建该集群的实例时，集群模板中的代码注册模板级指令处理程序和一组集群服务器应用回调，每个回调对应一个指令。

```
static enum ZclStatusCodeT
get_calendar(struct ZbZclClusterT *clusterPtr, void *arg,
      struct ZbZclCalClientGetCalendarT *req,
      struct ZbZclAddrInfoT *srcInfo)
{
      struct ZbZclCalServerPublishCalendarT rsp;

      memset(&rsp, 0, sizeof(rsp));
      /* fill in response */
      ZbZclCalServerSendPublishCalendar(clusterPtr, srcInfo, &rsp);
      return ZCL_STATUS_SUCCESS_NO_DEFAULT_RESPONSE;
}

…
callbacks.get_calendar = get_calendar;
…
cluster = ZbZclCalServerAlloc(zb, endpoint, &callbacks, arg)
…
```

```
struct ZbZclClusterT *
ZbZclCalServerAlloc(struct ZigBeeT *zb, uint8_t endpoint, struct ZbZclCalServerCallbacksT
*callbacks, void *arg)
{
      ...
      clusterPtr->cluster.command = zcl_calendar_server_command;
      ...
      ZbZclCalServerConfigCallbacks(&clusterPtr->cluster, callbacks);
```

在上述集群模板样例中，ZCL 集群基库调用 zcl_calendar_server_command()回调，其中包含代码可解包每个支持的命令。对于不支持的指令，回调必须返回 ZCL_STATUS_UNSUPP_MFR_CLUSTER_COMMAND（或对于 ZCL 规范[R2]中定义的集群而言，则返回 ZCL_STATUS_UNSUPP_CLUSTER_COMMAND）。这些返回码触发 ZCL 集群基库将相应的默认响应发送回客户端。

否则，当支持该指令时，集群模板指令处理程序会将 ZCL 指令有效负载解析为 struct ZbZclCalClientGetCalendarT 数据结构，并调用所提供的集群服务器应用 get_calendar()回调。

对于 GetCalendar 指令，get_calendar() 回调负责确定哪些日历与请求 ZbZclCalClientGetCalendarT *req 中的条件匹配，并通过调用 ZbZclCalServerSendPublishCalendar()返回匹配的日历。实施时会形成 ZCL PublishCalendar 消息并将其返回给客户端。然后，该服务器回调退出，以 ZCL_STATUS_SUCCESS_NO_DEFAULT_RESPONSE 结束服务器端业务，并通知 ZCL 集群基库其已发送响应（ZCL PublishCalendar），且必须仅在需要时发送“成功”默认响应。

如果集群服务器应用回调成功，但未发送“ZCL 消息”响应，则回调应以 ZCL_STATUS_SUCCESS 结束，生成“成功”默认响应。否则，必须返回不成功 ZCL_STATUS，并始终生成具有相应状态代码的默认响应。然而，在实施 ZCL 规范时，只有规范所述的状态代码才必须返回。

版本历史

表 5. 文档版本历史

日期	版本	变更
2020 年 7 月 17 日	1	初始版本

目录

表格索引

图片目录

重要通知 - 请仔细阅读

意法半导体公司及其子公司（“意法半导体”）保留随时对 ST 产品和/或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于意法半导体产品的最新信息。意法半导体产品的销售依照订单确认时的相关意法半导体销售条款。

买方自行负责对意法半导体产品的选择和使用，意法半导体概不承担与应用协助或买方产品设计相关的任何责任。

意法半导体不对任何知识产权进行任何明示或默示的授权或许可。

转售的意法半导体产品如有不同于此处提供的信息的规定，将导致意法半导体针对该产品授予的任何保证失效。

ST 和 ST 标志是意法半导体的商标。关于意法半导体商标的其他信息，请访问 www.st.com/trademarks。其他所有产品或服务名称是其各自所有者的财产。本文档中的信息取代本文档所有早期版本中提供的信息。