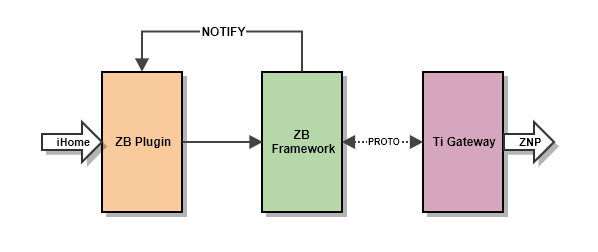
# System overview



## Zstack

Zstack 3.0 is the solution from Texas Instruments that supports the latest zigbee 3.0 specification and contains everything needed for a zigbee compliant solution.

With ZNP configuration is supported, Zstack can be used in two-chip architectures where Zigbee PRO stack runs on a SoC and the application runs on an external microcontroller.

## ZbPlugin

ZbPlugin implements a common interface to communicate with iHome framework. The logic in the interface should be as less as possible in order to easily maintenance and adding new features in the future.

## ZbFramework

ZB Framework is introduced to separate the concerns between the plugin and gateway. Thanks to this, ZB module can be easily portable to other platforms.

ZbFramework consists of the device lists which are recognized by ZNP, the commands to control endpoint, and the flow control to communicate with TiGateway in asynchronous mode.

The ZbFamework should be designed to be low-coupling to ZbPlugin and TiGateway. This can be obtained by socket communication to the TiGateway and notification mechanism to ZbPlugin.

## TiGateway

TiGateway features an abstracted socket API for application developers to access the Zigbee functionality, implemented through a set of Linux-based servers

TiGateway is provided by Texas instruments with many layers (Gateway applications, gateway subsystem, ZNP). Unfortunately, gateway application layer is not well-designed and not stable in high load working, therefore, this layer will be redesigned (ZbFramework)

## Repository

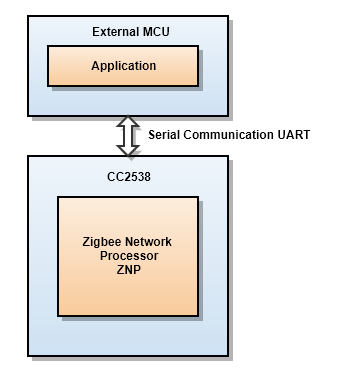
http://brtsgsvn1.brtchip.local:8080/svn/FP\_SW\_000155/branches/zigbee3/

# Detail design

## Zstack

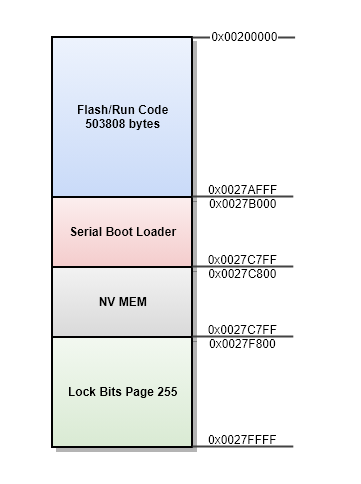
### Zigbee network processor

In order to support two chip architecture, where the Zigbee PRO stack runs on a SoC and the application runs on an external microcontroller, ZNP solution must be applied. This solution comes with an ZNP firmware image that would be programmed on SoC and interfaces to any controller through a range of serial interfaces.

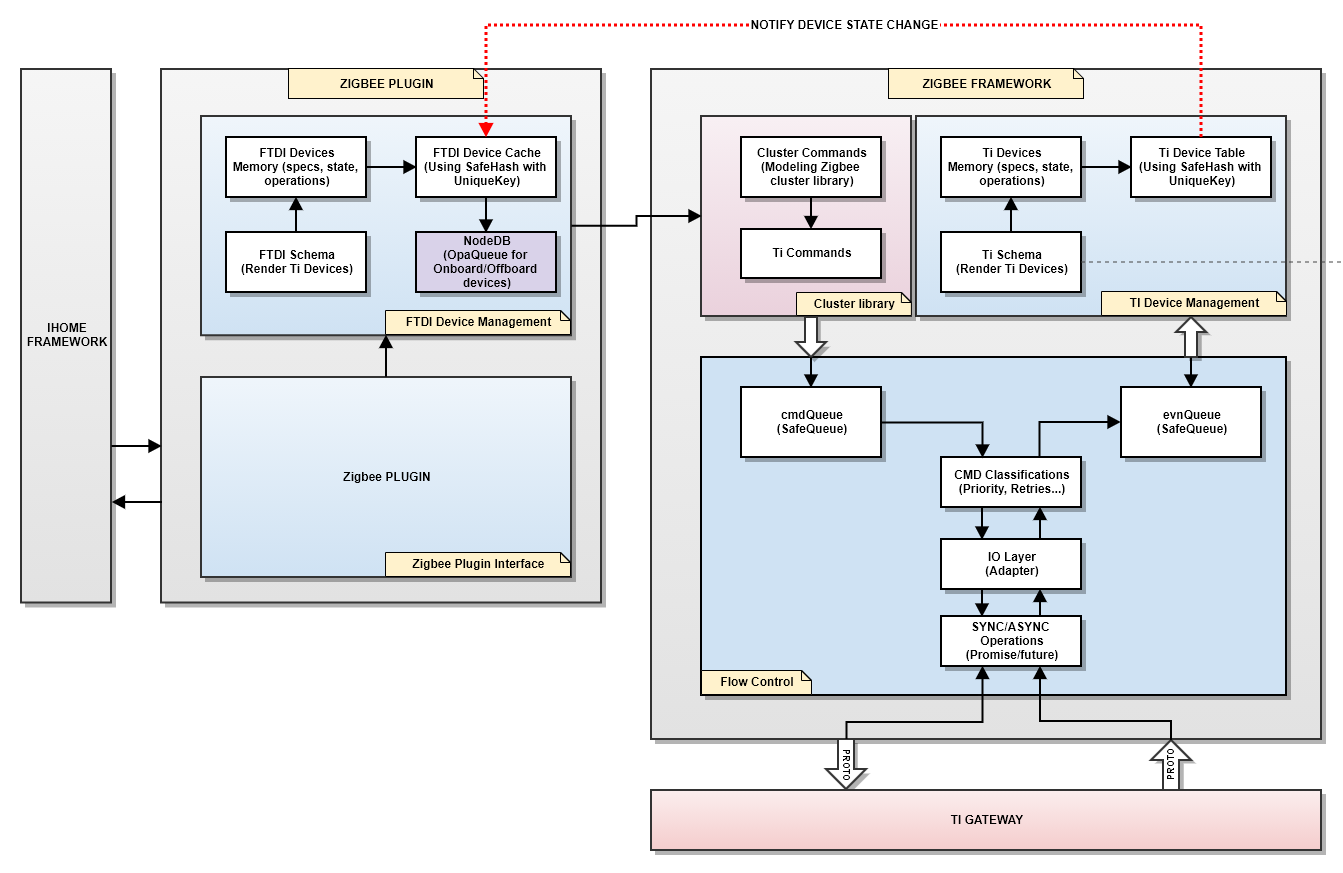


### ZNP Firmware

Firmware is a single image containing both serial bootloader and application code so that the part must only be programmed once. Where application code is Zigbee 3.0 stack and serial bootloader is to handle receiving a new application code and write it to flash memory, as well as check the integrity of the active code image before jumping to it, this check guards against an incomplete or incorrect programming of the active image area.



## Zigbee framework

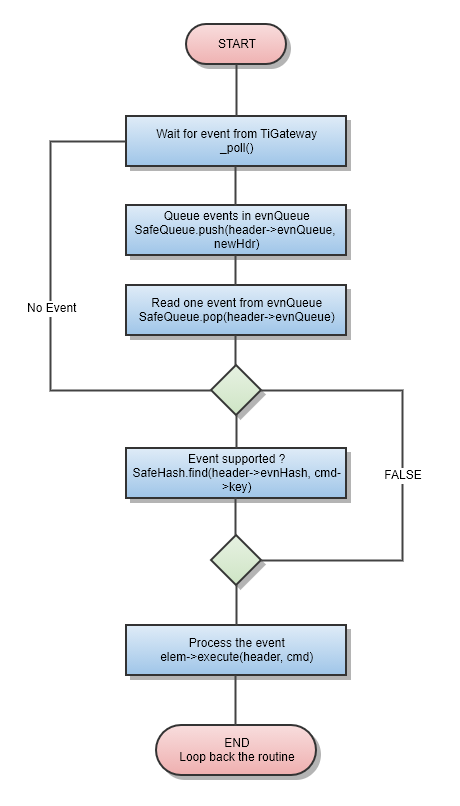


### Async operations

In order to flatten async operations, commands and events will be queued to process later. For the less impact to ingress and egress flow, two queues were used, cmdQueue for queuing egress commands and evnQueue for queuing ingress commands.

Thanks to these queues, no need to use any locking machanism for asynchronous operations. In order to take the advantages of free locking, state in Ti Device managment should be modified by the evnQueue handler only, other accesses to this zone should be read/notify operations.

The asynchronous operation could be simplified by the flow chart below



### Cluster libraries

Cluster library is a modeling of “Ti cluster library specification” for manipulating attributes and other general tasks that are not specific to an individual cluster. In addition, this library provided a local library to query local data in TiGateway

Thanks to cluster library, requests/events could be encapsulated as a command object that can be queued in cmdQueue and envQueue.

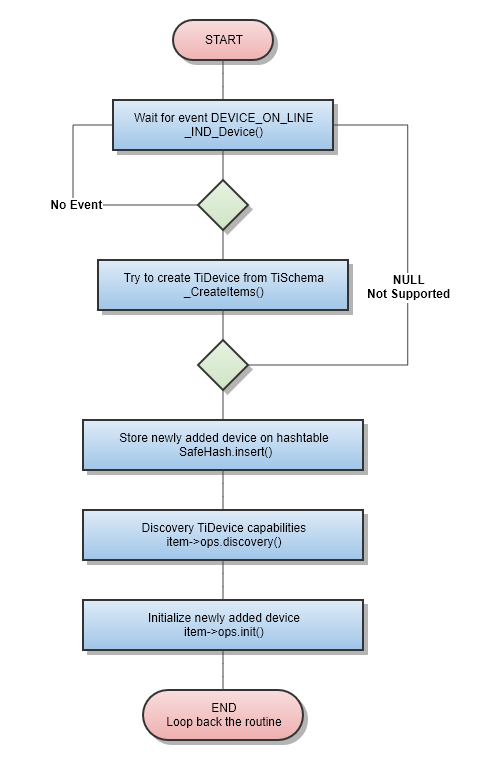
### Ti Device management

Ti Device Management is a trusted place for querying device states and information to minimize the request to endpoint devices. In order to obtain this, it must keep tracking all events relating device states and have it updated synchronously.

Since the events are asynchronous operations and to be queued before processing, the update/query to Ti Device management should be as fast as possible to have efficiency in flow control. To obtain this, hash table is used to store device states and information. In addition, no locking mechanism will be used, hence, make sure that evnQueue handler is the only one thread that have write access to the device list.

Ti Device management is designed to manage a diversity of device types which were identified by profile ID and device ID. Hence, it should be expanded easily and follow strictly the rules “closed for modifications but open for extensions". Blueprint (Schema) is applied to obtain this. A device blueprint is a design of a specific endpoint which predefined the operations and behaviors of devices. If you are familiar with OOP, think it as a class with methods (device operations) and properties (device states). Thanks to blueprint, developing new devices is very fast, efficient and minimize the code change to the framework

Ti Device management operations could be simplified by the flow chart below



## Zigbee Plugin

### FTDI device management

Devices managed by iHome framework are modeled by predefined attributes and states (iHome\_object\_model.xlsx). Hence, a plugin that want to interface with iHome must provide an adapter between endpoint model and iHome device model (FTDI Device model). It requires to have one adapter for each kind of device based on profile and device ID.

As the target is to build an ecosystem, the new type of devices will be added to the system frequently. Therefore, it’s necessary to have an architect that can help in adding/removing new adapters easily, transparently and minimize the code change as less as possible. For that reason, FTDI device management is introduced.

The architecture of FTDI device management is very similar to TI device management. The difference is the device model (states, specifications) stored in hash table.

# Implementations

## Firmware

### Prerequisite

* Firmware source code …\zigbee3\Firmware\v0.1\Z-Stack 3.0.2
* TI Smart RF Flash Programmer v2
* TMS320-XDS100V3 Debugger
* Zigbee module EMB-Z2538PA
* IAR Embedded Workbench IDE - Arm 8.22.1

### Firmware design

#### Pin mapping

Texas instruments provides a reference design for CC2538 – CC2592 (AN130 - Using CC2592 frontend with CC2538). This reference design is applied for almost manufacture but EMB-Z2538PA. Hence, it required some changes in firmware which was designed for reference design to make it works with EMBIT module.

As described in datasheets, PA1\_UART1\_RX (GPIO\_PIN\_1) should be UART RX port instead of GPIO\_PIN\_0 as reference design. This change in chip layout required the switch between GPIO\_PIN\_1 and GPIO\_PIN\_0 in firmware

#### Enable SBL\_CLIENT

This helps the application firmware can be combined with serial bootloader, hence firmware upgrade could be able to execute via SBL script

#### Disable Child aging

Child aging must be disable, otherwise Xiaomi device are being kicked off the network

#### Revision number

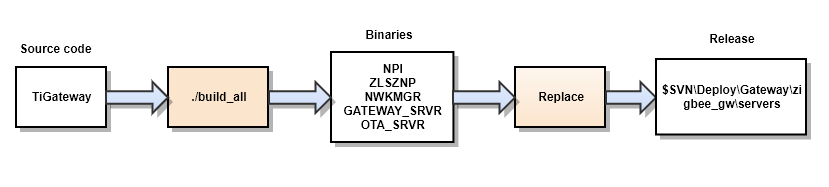
## TiGateway

### Prerequisite

* Z-STACK Linux Gateway v1.0.1 Release date: 09-Jun-2014
* Compiler: arm-linux-gnueabihf-gcc
* TiGateway source code “…\zigbee3\Gateway\Zigbee\_3\_0\_Linux\_Gateway\_1\_0\_1”
* TiGateway release “…\zigbee3\Deploy\Gateway”

### Workflow

The workflow of TiGateway implementation could be simplified by the flow



### Build Ti Gateway binaries

Access to TiGateway source code and build with arm-linux-gnueabihf-gcc

$ cd $GW/src

$ ./build\_all

The output consists of scripts, configurations and binaries files and should be found at $GW/source/out.

### Deploy TiGateway

The configuration and script files are generated by build script and are not the same as SVN version since some modifications have been introduced to make it works on PANLhub. Hence, only binaries files below will be used.

NPI - UART transport

ZLSZNP - MT message to protobuf message

NWKMGR - Network manager

GATEWAY\_SRVR – Gateway

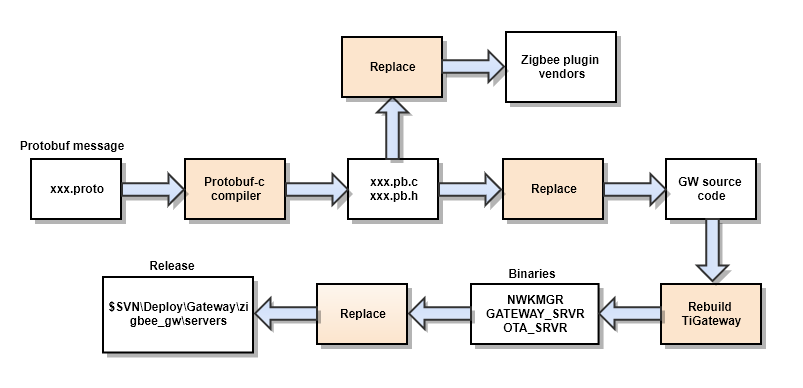
OTA\_SRVR – OTA server

For release, make sure that TiGateway release directory “…/zigbee3/Gateway/zigbee\_gw/servers” is updated with the new binaries listed above.

## Register new TiGateway message

Protocol buffer is using to discribe the data exchanges between ZbFramework and TiGateway and has been defined at “$PATH\Proto\_files”. The exchange messages are enough for almost cases, but it’s flexible to add new messages whenever it’s necessary as long as the steps below are followed strictly.

### Workflow



### Prerequisite

* Proto message “: …\zigbee3\Gateway\Zigbee\_3\_0\_Linux\_Gateway\_1\_0\_1\Proto\_files”
* TiGateway source code GW: …\zigbee3\Gateway\Zigbee\_3\_0\_Linux\_Gateway\_1\_0\_1
* TiGateway release …\zigbee3\Deploy\Gateway
* Proto compiler

### Implementation

#### Install protobuf-c compiler

$ sudo apt-get -f install protobuf-c-compiler

#### Generate protobuf API from protobuf messages (.proto)

Once the package is installed, proceed to generate the structures/functions to manage protobuffers in .c/.h files using protoc compiler. The

$ protoc-c --c\_out=. gateway.proto

$ protoc-c --c\_out=. gateway.proto

$ protoc-c --c\_out=. gateway.proto

The result should be: gateway.pb.cc, gateway.pb.h, nwkmgr.pb.cc, nwkmgr.pb.h, otasrvr.pb.c, otasrvr.pb.h

#### Rebuild TiGateway

Update TiGateway with the new protobuf API

GW = ...\zigbee3\Gateway\Zigbee\_3\_0\_Linux\_Gateway\_1\_0\_1\ source\Projects\zstack\linux

GW\hagateway

GW\nwkmgr

GW\otaserver

Rebuild TiGateway, make sure that the binaries must be updated

…\out\Precompiled\GATEWAY\_SRVR\_arm

…\out\Precompiled\NWKMGR\_SRVR\_arm

…\out\Precompiled\OTA\_SRVR\_arm

#### Deploy TiGateway

Make sure that TiGateway release directory (…\zigbee3\Deploy\Gateway\zigbee\_gw\servers

) is updated with the new binaries at previous step.

#### Rebuild zigbee plugin

The protobuf API should be updated for both TiGateway and Zigbee plugin. Make sure that the plugin vendor (“…/app\_plugins/zigbeev3/API/libs/vendors”) is updated.