**How to integrate FAST API into Keil**

**Table of Contents [RIGHT CLICK TEXT BELOW TO UPDATE FIELD]**

[**1**](#_heading=h.gjdgxs) **Introduction 3**

[**2**](#_heading=h.30j0zll) **Hardware and firmware requirement for demo 3**

[**3**](#_heading=h.1fob9te) **Integration steps 3**

[3.1](#_heading=h.3znysh7) Include header files & library 3

[3.2](#_heading=h.2et92p0) Convert .bin into an array 3

[3.3](#_heading=h.tyjcwt) Integrate I2C(Nordic calls TWI) 4

[3.4](#_heading=h.3dy6vkm) Implement internal functions 4

[3.5](#_heading=h.1t3h5sf) Implement API functions 5

[3.6](#_heading=h.4d34og8) Firmware download and test 5

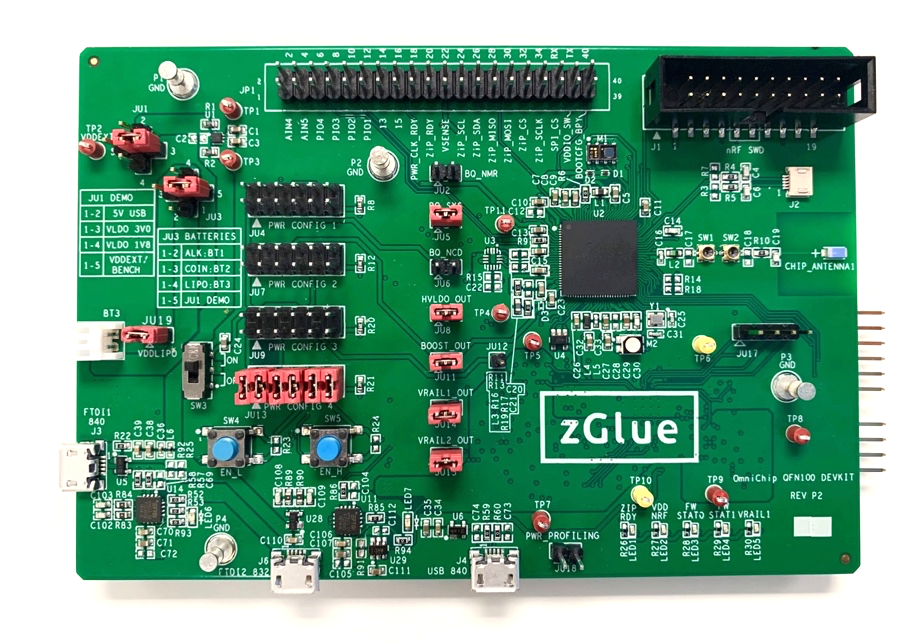
[**4**](#_heading=h.17dp8vu) **Revision History 6**

# Introduction

Fast API is a library which contains functions controlling zglue smart fabric and peripherals. This document intends to guide you through how to integrate FAST API into a keil project.

# Hardware and firmware requirement for demo

The hardware used for this demo is Omnichip. Please refer to the link <https://zglue.com/products/omnichip>.



*Figure 1: Omnichip*

The firmware used for this demo is Nordic SDK nRF5\_SDK\_13.0.0-1 with Softdevice s132. The project used for this demo is ble\_app\_uart. The IDE integrated is Keil5.

# Integration steps

## Include header files & library

User needs to include fast\_api.h and fast\_api\_support.h into the project.

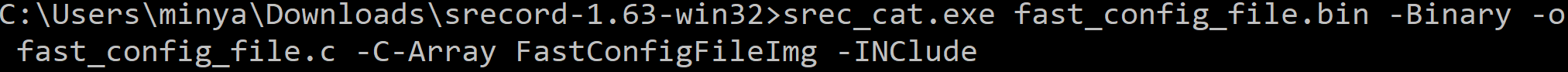
fast\_api.h contains the declaration of fast api functions, enum parameters and error codes.

fast\_api\_support.h contains 5 function declarations. Users need to implement these functions because these functions are mcu dependent.

## Convert .bin into an array

fast\_config\_file.bin is a binary file which includes system information and smart fabric routing information. Some FAST APIs use data in this binary file. There are couple ways to integrate this binary file into firmware. Here we provide a generic method in which we convert .bin into an array using a third-party tool and include the array into firmware.

To convert .bin into an array, download srecord third party tool which is included in this folder for windows user. Go to the directory and type the commands shown below to generate fast\_config\_file.c which contains an array FastConfigFileImg[]



## Integrate I2C(Nordic calls TWI)

Zglue smart fabric and peripherals can be controlled via I2C or SPI interface. Only I2C interface was integrated in this example. Users will need to integrate I2C or SPI driver into the firmware depending on which driver is chosen.

## Implement internal functions

As mentioned earlier, there are 5 functions that need to be implemented by users to fully use our API. User might implement part of the functions if he/she decides to use our API partially. These 5 functions are implemented in zglue\_fast.c in this demo.

/\* Read from a specific register in the FAST \*/

fast\_status\_e fast\_read\_register(uint8\_t dev\_id, uint32\_t reg\_addr, uint8\_t \*read\_data, uint16\_t len);

User needs to integrate I2C master driver into firmware. In this case, I2C read should be implemented.

Zglue’s register is in the format of 32bits address and 32bits data. While Zglue I2C slave driver expects big-endian format for the i2C transmission. User might need to swap the byte order if the default configuration is little-endian format.

/\* Write to a specific register in the FAST \*/

fast\_status\_e fast\_write\_register(uint8\_t dev\_id, uint32\_t reg\_addr, uint8\_t \*read\_data, uint16\_t len);

User needs to integrate I2C master driver into firmware. In this case, I2C write should be implemented.

/\* Read the configuration file from specific place in memory of the HOST system \*/

fast\_status\_e fast\_config\_file\_read(uint32\_t offset, uint32\_t length, uint16\_t \*buffer);

fast config file format is in a 16bits width format. Therefore, users have to take two bytes from the array and convert into a 16 bits format.

/\* Toggle the ULPM Wake up pin of the fast \*/

fast\_status\_e fast\_toggle\_ulpm\_wake(void);

The Smart Fabric has five power states; they are Full Programming Mode, Limited Programming Mode, Low Power Mode, Ultra Low Power Mode, and Ship Mod.

Changing between Full Power Mode, Limited Programming Mode, and Low Power Mode is accomplished by simply writing to a register. Switching to Ultra-Low Power Mode, is done by writing to a register, and switching out of this mode is accomplished by toggling the ULPM\_WAKE pin.

/\* Sleep in uS \*/

void fast\_sleep(uint32\_t time\_us);

/\*Debug feature\*/

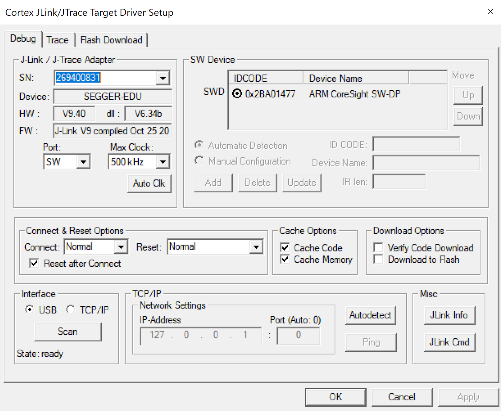
Inside our library, we define dbg\_printf as printf. If user’s current firmware doesn’t support printf function, he/she needs to implement printf function to view debug message.

## Implement API functions

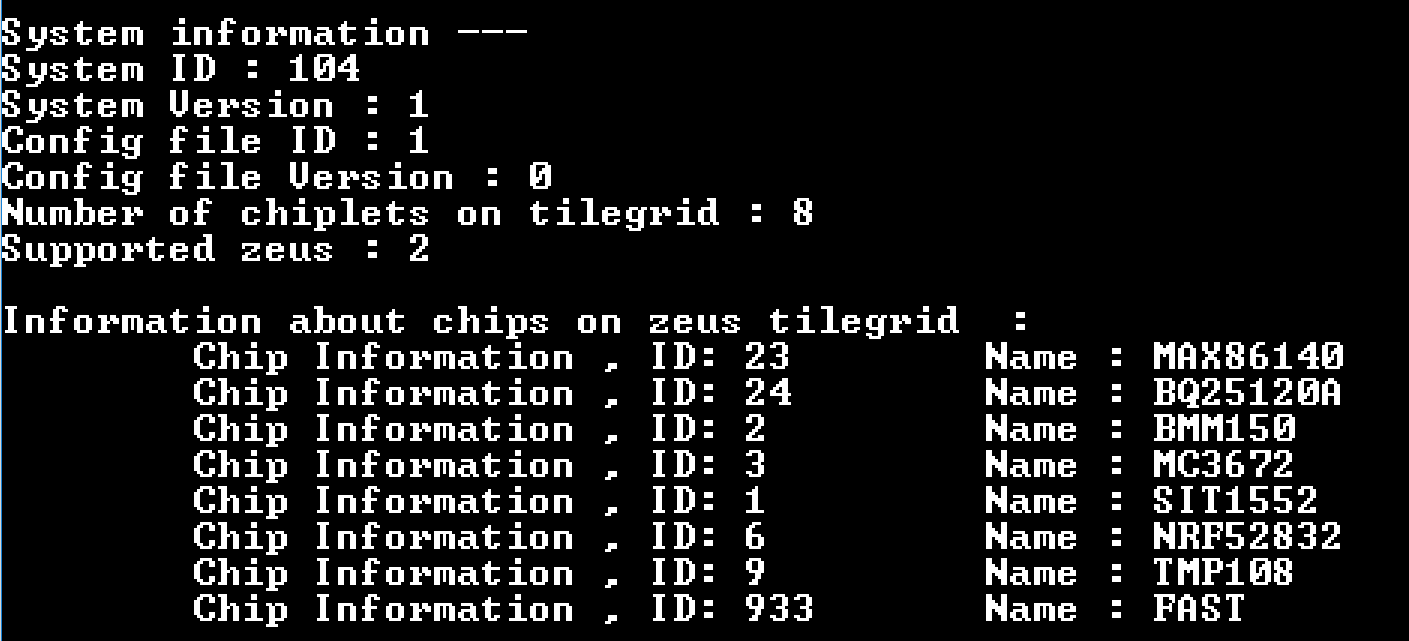
User can start to call Fast API in the application. There are a few API calls implemented in main.c as a demo.

## Firmware download and test

After porting, user can compile and download the image into Omnichip platform. If you have trouble downloading the image to Omnichip platform, try to lower the downloading speed.



To see the demo, open UART terminal in your computer and configure the baud rate to 115200. After downloading, you should be able to see below message.



# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **By** | **Changes** |
| 3/6/2019 | 1.0 | Min | Initial version |
|  |  |  |  |