Gowin goConfig IP(I2C)示例代码

使用说明

TN715-1.0,2024-02-02

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止发言或其它方式授予任何知识产权许可。除高云半导体在其产品的销售条款和条件中声明的责任之外，高云半导体概不承担任何法律或非法律责任。高云半导体对高云半导体产品的销售和／或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等，均不作担保。高云半导体对文档中包含的文字、图片及其它内容的准确性和完整性不承担任何法律或非法律责任，高云半导体保留修改文档中任何内容的权利，恕不另行通知。高云半导体不承诺对这些文档进行适时的更新。

版本信息

日期	版本	说明
2024/02/02	1.0	初始版本。

目录

图目录

表目录

1 关于本手册

1.1 手册内容

Gowin® goConfig IP(I2C)示例代码使用说明主要内容包括示例代码简介、功能描述、API 接口描述及使用方法。

注！

- 本示例代码使用 C 语言编写，提供的代码是基于嵌入式平台（STM32F429）的完整工程包，使用的 IDE 是 keil_V5。
- 适用于配置 GW1N-1P5C/GW1N(R)-2C 系列。
- 在微控制器（MCU）访问 goConfig I2C IP 之前，我们需要加载一个包含 goConfig I2C IP 的比特流文件。

1.2 相关文档

通过登录高云®半导体网站 www.gowinsemi.com 可以下载、查看以下相关文档。

- DS100，GW1N 系列 FPGA 产品数据手册
- DS117，GW1NR 系列 FPGA 产品数据手册
- IPUG795，Gowin goConfig I2C IP 用户指南
- UG290，Gowin FPGA 产品编程配置手册

1.3 术语、缩略语

表 1-1 中列出了本手册中出现的相关术语、缩略语及相关释义。

表 1-1 术语、缩略语

术语、缩略语	全称	含义
ACK	Acknowledge	响应
FPGA	Field Programmable Gate Array	现场可编程门阵列
I^2C(I2C)	Inter-Integrated Circuit	两线式串行总线
IP	Intellectual Property	知识产权
NACK	Not Acknowledge	不响应
SRAM	Static Random Access Memory	静态随机存储器

1.4 技术支持与反馈

高云半导体提供全方位技术支持，在使用过程中如有任何疑问或建议，可直接与公司联系：

网址：www.gowinsemi.com

E-mail：support@gowinsemi.com

Tel：+86 755 8262 0391

2 示例代码介绍

2.1 示例代码简介

用户移植使用这份示例代码时，可以直接应用我们封装好的上层 API 接口（位于 i2c_jtag.c），底层 I2C 发送数据接口（位于 bsp_i2c.c）可以根据自己的平台做修改。

2.2 文件目录

```
————————————————————————————————————
├──goConfigIP(I2C)示例代码说明.docx
├──stm32_I2C_PORG_FLASHorSRAM(I2CIP)
│       │   brief introduction.pptx
│       │   readme.txt
│       ├──Project
│       │   └──RVMDK（uv5）
│       │           BH-F429.uvprojx
│       └──User
│           │   bitstream.h
│           │   main.c
│           │
│           ├──gw_i2c_xxx
│           │       i2c_jtag.c
│           │       i2c_jtag.h
│           ├──i2c
│           │       bsp_i2c.c
│           │       bsp_i2c.h
```

```
|           └──usart
|                  ├── bsp_debug_usart.c
|                  └── bsp_debug_usart.h
```

2.2.1 主要文件介绍

表 2-1 主要文件

文件	描述
User/main.c	main file
User/bitstream.h	the bitstream data in a hpp file
User/User/i2c_jtag	i2c timing file(thru IP)
User/User/i2c/bsp_i2c.c	i2c bsp API

3 功能描述

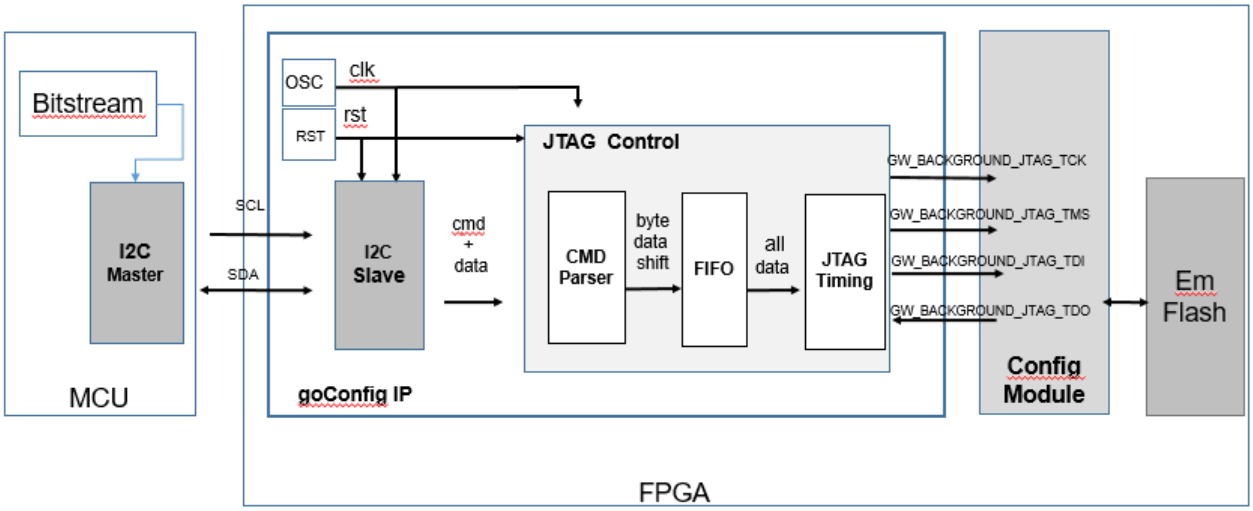
3.1 功能介绍

本实例的示例代码是在 MCU 中执行，通过 I²C 接口借助 goConfigIP 可以实现对内嵌 Flash 的擦除与编程操作也可以读取 FPGA 的 ID code、user code、status code。

注！

开始使用 goConfig IP 之前，FPGA 需要加载包含 goConfig I2C IP 的比特流文件。

图 3-1 结构框图



3.2 代码执行流程

3.2.1 检验 MCU 与 IP 通信否正常

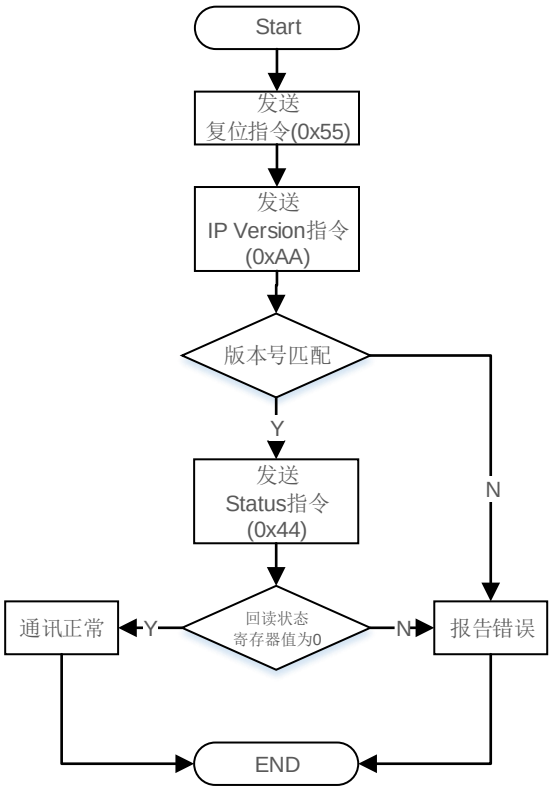
在开始进行背景升级前，先确认 MCU 与 IP 间的通信是否正常，检验流程如下：

1. 发送复位指令（0x55）确保 IP 处于初始状态（API：gw_i2c_send_reset）；
2. 发送 IP Version 指令 (0xAA) 来检查读回版本值 (如 Version Data=0x20)以确保 IP 版本与预期的一致（API:gw_i2c_read_version）；

3. 发送 status 指令（0x44），查看读回的状态值（API: gw_i2c_read_reg_status）；
4. 当操作完成并且读回的数据符合预期，表明与 IP 的通信可以开始。

检验 MUC 与 IP 通信的流程图如图 3-2 所示。

图 3-2 检验 MUC 与 IP 通信的流程图



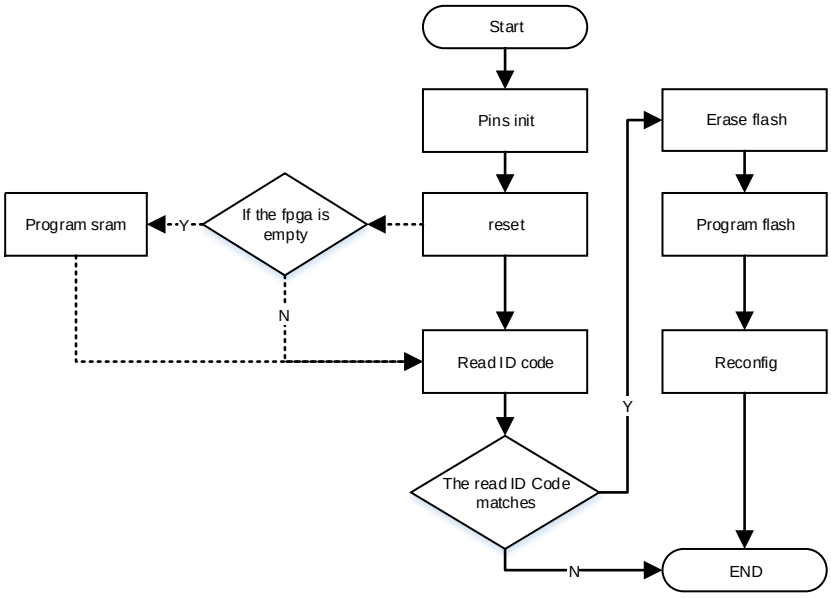
3.2.2 背景升级整体流程

背景升级流程如下所示：

1. MCU 平台引脚初始化（I2C,debug_uart）；
2. IP 复位 (API:gw_i2c_send_reset)；
3. 如果芯片不是空白片则忽略此步，如果芯片是空白片，先配置 SRAM (API: i2c_configmode_program_sram)；
4. 比对 MCU(I2C)读取的 IDcode 与数据流文件的 IDcode 是否一致(API：mach_code)，若一致则继续执行，不一致结束背景升级；
5. 擦除 Flash (API: gw_i2c_erase_flash_tsmc)；
6. 编程 Flash (API: gw_i2c_prog_flash_tsmc)；
7. 重新配置 (gw_i2c_send_reconfig_n)。

背景升级流程图如图 3-3 所示。

图 3-3 背景升级流程图



读取 FPGA Code

读取 FPGA code 使用 API: gw_i2c_read_code。当 MCU 与 IP 通讯正常后，可以使用 I2C 读取 FPGA 的 code。

- ID code 即 JEDEC ID Code，是 FPGA 器件的一个基本标识；
- user code 是用户为自己所使用的 FPGA 器件进行的身份标识；
- status code 是 FPGA 器件完成加载数据流文件后的一个状态标识。

例如：

- 读取 ID Code，API 接口为：gw_i2c_read_code(0x11)；
- 读取 user Code，API 接口为：gw_i2c_read_code(0x13)；
- 读取 status Code，API 接口为：gw_i2c_read_code(0x41)。

空白片通过 I2C 配置 SRAM

通过 I2C 配置空白片的 SRAM 使用 API: i2c_configmode_program_sram.

注！

FPGA 空白片没有 IP，这里配置 SRAM 是通过 FPGA 内部原生的硬件配置模块配置 SRAM，原生的硬件配置模块仅支持配置 SRAM，如果编程内嵌 Flash 需要使用 goConfig I2C IP。

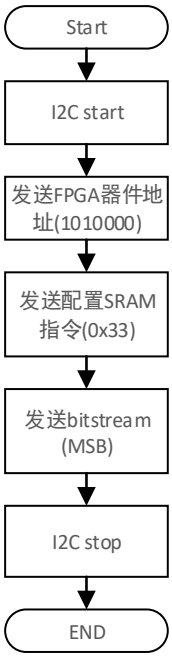
I2C 配置 SRAM 流程如下所示：

1. 发送 start 信号
2. 发送器件地址
3. 发送配置 SRAM 指令（0x33）
4. 发送 Bitstream（MSB 格式）

5. 发送 stop 信号

配置 SRAM 流程图如图 3-4 所示。

图 3-4 配置 SRAM 流程图



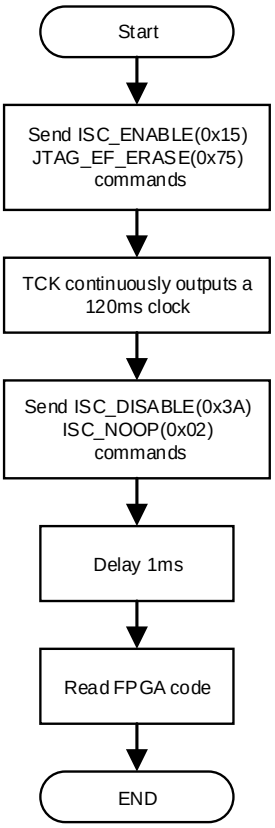
擦除内嵌 Flash

擦除内嵌 Flash 使用 API: gw_i2c_erase_flash_tsmc。擦除内嵌 Flash 流程如下所示：

1. 发送 ISC_ENABLE(0x15)和 JTAG_EF_ERASE(0x75)；这两个指令是通过 API（write_inst_and_data）打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；
2. TCK 持续输出 120ms clock（API：gw_i2c_send_runtest_xus）；
3. 发送 ISC_DISABLE(0x3A)和 ISC_NOOP(0x02)指令；这两个指令是通过 API（write_inst）打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；
4. delay 1ms（API：gw_i2c_send_runtest_xus）；
5. 读取 FPGA code（API：gw_i2c_read_code）。

擦除内嵌 Flash 流程图如图 3-5 所示。

图 3-5 擦除内嵌 Flash 流程图



编程内嵌 flash

编程内嵌 Flash 使用 API: gw_i2c_prog_flash_tsmc。编程内嵌 Flash 流程如下所示：

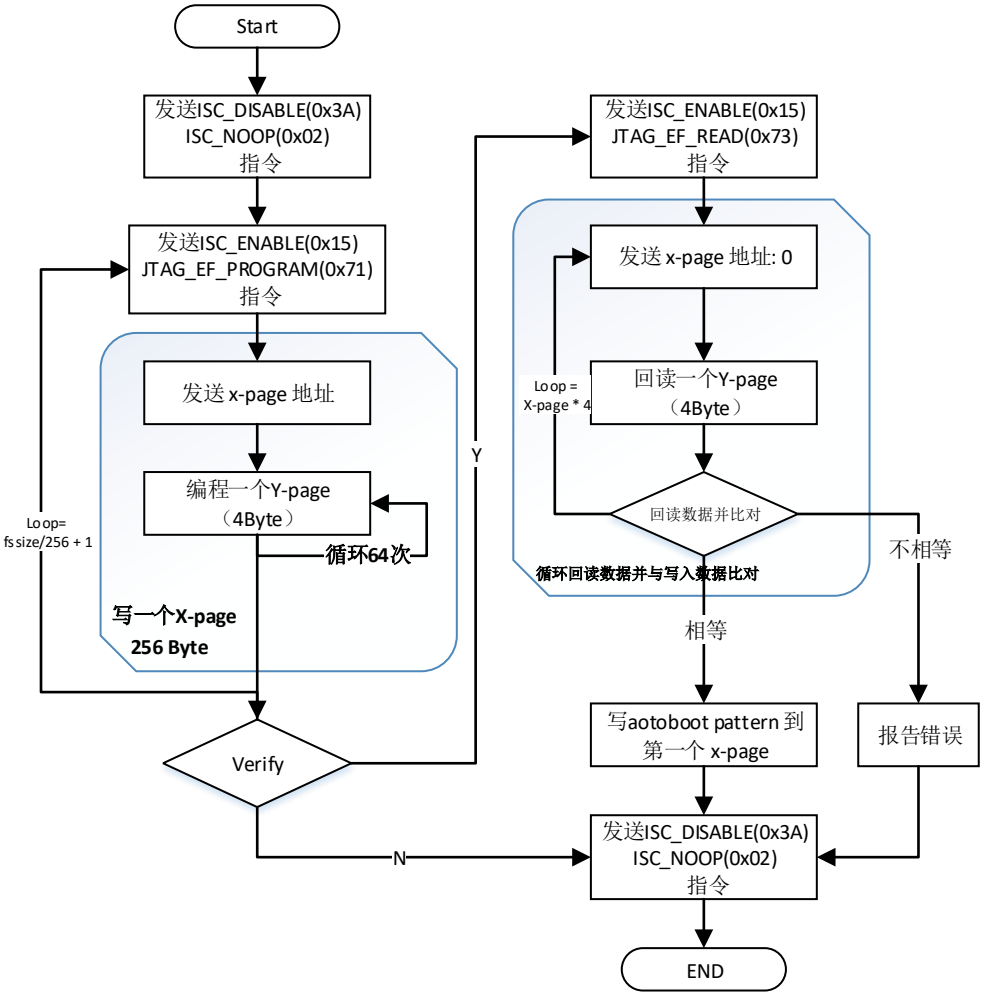
1. 发送 ISC_DISABLE(0x3A)和 ISC_NOOP(0x02)；这两个指令是通过 API（write_inst_and_data）打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；
2. 发送 ISC_ENABLE(0x15)和 JTAG_EF_PROGRAM (0x71)；这两个指令是通过 API（write_inst_and_data）打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；
3. 发送 X-pgae 地址，这个地址数据通过 API (write_one_x_address)打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；
4. 编程一个 Y-page，一个 X-page 包含 64 个 Y-page, 这个 Y-page 数据通过 API（write_one_y_page）打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；
5. 如果打开回读校验的话需要发送 ISC_ENABLE(0x15) 和 JTAG_EF_READ (0x73)；这两个指令是通过 API（write_inst_and_data）打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；
6. 发送 X-pgae 地址数据 0x00，这个地址数据通过 API

（write_one_x_address）打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；

7. 循环从 Flash 回读数据，并与写入数据流文件做对比（API：recv_one_y_page_and_verify）；
8. 循环回读对比结束再写 autoboot-pattern 数据到第 1 个 X-page，autoboot-pattern 数据通过 API(i2c_prog_xpage) 打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA；
9. 发送 ISC_DISABLE(0x3A)和 ISC_NOOP(0x02)指令；这两个指令是通过 API（write_inst_and_data）打包后由 MCU 的 I2C 底层接口（API：i2c_write_buffer）发送到 FPGA。

编程内嵌 Flash 流程图如图 3-6 所示。

图 3-6 编程内嵌 Flash 流程图



4 API 接口描述

4.1 常用 API 汇总

表 4-1 常用 API 汇总

名称	功能	备注
gw_i2c_read_code()	读取FPGA芯片Code	可以读取ID Code、Status Code、User Code
gw_i2c_prog_flash_tsmc()	编程Em-Flash	–
gw_i2c_erase_flash_tsmc()	擦除Em-Flash	
gw_i2c_send_reset()	复位IP	
gw_i2c_read_version()	读取IP版本信息	

4.2 IP 基本指令

4.2.1 指令定义

Gowin goConfig I2C IP 2C 系列支持的指令如表 4-2 所示。

表 4-2 2C 系列指令定义

序号	指令名称	指令地址(CMD)	描述	指令格式	示例代码中定义	备注
1	Status	0x44	读状态寄存器指令	CMD	READ_STATUS_INST	–
2	Reset	0x55	复位IP指令	CMD	IP_RESET_INST	
3	Send (data back)	0x66	发送数据且回读Flash接口数据指令	CMD + Length + Data +Check_sum	SEND_DATA_BACK_INST	
4	Waiting	0x77	等待指令	CMD + Length + Check_sum	WAITING_INST	
5	Send (No data back)	0x88	发送数据且不回读Flash接口数据	CMD + Length + Data +Check_sum	SEND_NO_DATA_BACK_INST	
6	Reconfign	0x99	Reconfign指令	CMD	RECONFIG_N_INST	

序号	指令名称	指令地址(CMD)	描述	指令格式	示例代码中定义	备注
7	IP Version	0xAA	IP版本指令	CMD	IP_VERSION_INST	–

指令格式中，名称的定义与说明如下：

- CMD: 支持的指令参考表 4-2；
  - 定义：指令地址；
  - 位宽：8 比特（bit）。
- Length：最小值为 2
  - 定义：发送的有效字节长度[1]；
  - 位宽：16 比特（bit），低 8 比特（Length_L）先发，高 8 比特（Length_H）后发；
  - 计算公式： Length =有效数据长度+2。

注!

- [1] 有效字节长度即有效数据（Data）的字节数+2；以 0x66 指令为例，Length= Data 的字节长度 +2；
- 例如预期输入字节长度为 10，则 Length=10+2=12。

- Data：
  - 定义：发送的有效数据；
  - 位宽：一个数据 8 比特；
  - 长度：实际发送的数据长度根据 Length-2 决定；Data_0 表示第一个有效数据，Data_1 表示第二个有效数据，... Data_N 表示最后一个有效数据。
- Check_sum：
  - 定义：校验
  - 位宽：16 比特，高 8 比特（Check_sum_H）先发, 低 8 比特（Check_sum_L）后发；
  - 计算公式：Check_sum =CMD+ Length_L + Length_H +Data_0+...+ Data_N；

注！

当 Check_sum 计算溢出时，只取低 16 位，其余位舍弃。假设计算结果，Check_sum=0x10F11；则 Check_sum_H=0x0F；Check_sum_L=0x11.其余的舍弃。

Status(0x44)

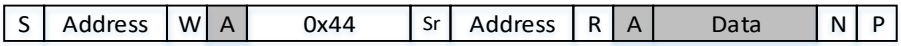
读状态寄存器指令，只读。读回的 Data 为状态寄存器值，定义如下表 4-3 所示。

表 4-3 状态寄存器

比特位	默认值	描述	备注
Bit[7:3]	0	保留	–
Bit [2]	0	1：Flash接口工作中 0：Flash接口空闲	
Bit [1]	0	1：校验出错 0：校验正确	
Bit [0]	0	1：I²C状态出错 0：I²C状态正常	–

读状态寄存器指令示例如图 4-1 所示。

图 4-1 读状态寄存器指令示例



注！

下文中，相似图例，未作特殊说明，则含义与本例中一致。

- 图中“白色底纹”为“I2C 主机”发送至“I2C 从机”，“灰色底纹”为“I2C 从机”发送至“I2C 主机”。
- 图示中会使用到的名称定义如下：
  - S：I²C 总线的“开始（START）”；
  - Address：I²C 总线的“从机地址（Slave Address）”；
  - W：I²C 总线的“写（WRITE）”；
  - A：I²C 总线的“响应（ACK）”；
  - Sr：I²C 总线的“再开始（Repeated START）”；
  - R：I²C 总线的“读（READ）”；
  - Data：I²C 总线中，传输的有效数据；
  - N：I²C 总线的“不响应（NACK）”；
  - P：I²C 总线的“停止（STOP）”；
  - Length_L：Length 的低 8 bits；
  - Length_H：Length 的高 8 bits；
  - Data_0：第一个有效数据；
  - Data_N：最后一个有效数据；
  - Check_sum_H：Check_sum 的高 8 bits；
  - Check_sum_L：Check_sum 的低 8 bits。

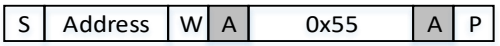
Reset(0x55)

复位 IP 指令。将复位 IP 内部的状态，以及清空内部的 FIFO。

注！

此过程只涉及 IP，不涉及背景升级流程

图 4-2 复位 IP 指令示例



Send(0x66,data back)

发送数据且回读 Flash 数据指令。I2C 接口发送将要读回数据的字节（byte）数目（Length），同时发送数据；若校验无误 IP 将转换为 Flash 接口，并将数据回读到 FIFO 中（图 4-3 步骤）。此时，若用户侧若需要此数据，可通过 I2C 接口发送读指令，可以将 FIFO 内的数据读出（图 4-4 步骤）。

简单来说，在回读的过程中，分为两步：

1. 将待读数据读至 IP 的 FIFO 内；
2. 将 FIFO 内的数据通过 I^2C 总线读出。

注！

- 第 1 步与第 2 步中间可以插入部分指令，例如读状态指令。
- 在图 4-3 中，Data_0~Data_N，可以为任意值，但一般建议写 0x00 或 0xFF。
- 在图 4-4 中，Data_0~Data_N 是读回的实际值。

图 4-3 回读指令发送示例

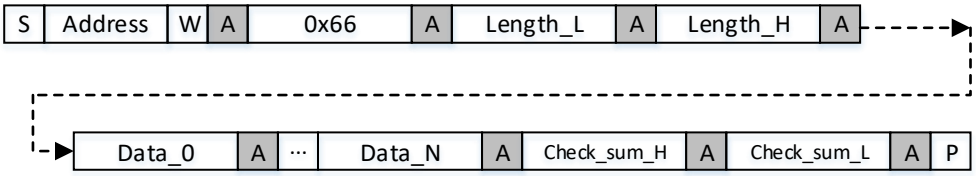
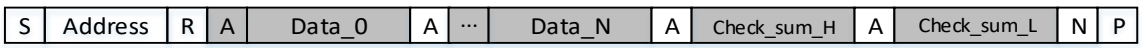


图 4-4 读 FIFO 数据指令示例



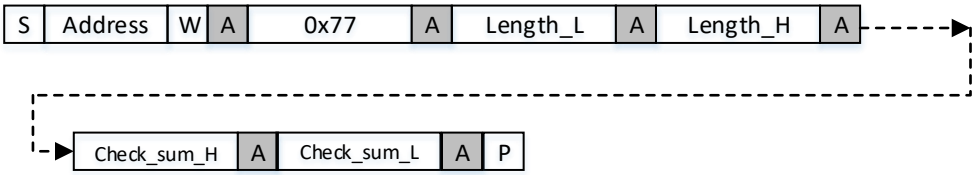
Waiting(0x77)

等待指令。I2C 接口发送等待时长的字节数目（Length），校验无误，Flash 接口将会持续发送空闲的时钟（将不会有数据传输），直到“等待时长的字节数目”计数完成。

注！

2C 系列中，Flash 接口时钟为 2.5MHz，即一个周期 400ns。1 字节为 8*400 ns =3200ns=3.2us。例如，等待 150ms，计算出有效的字节数为 46875= 0xB71B；输入的字节数为 0xB71B+2=0xB71D，则 Length_L = 0x1D；Length_L = 0xB7。

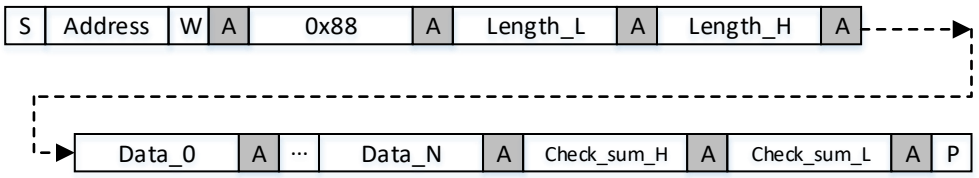
图 4-5 等待指令示例



Send(0x88,No data back)

发送数据且不回读 Flash 数据指令。I2C 接口先将待发送数据存放到 FIFO，在数据计数完成后，且校验无误，FIFO 内的数据一次性通过 Flash 接口发出。

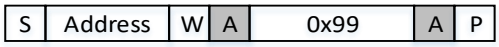
图 4-6 发送数据且不回读 Flash 数据指令示例



Reconfign(0x99)

Reconfign 指令。I2C 接口发送此指令，可实现触发（拉低）“Reconfig_N”管脚的功能。

图 4-7 Reconfign 指令示例



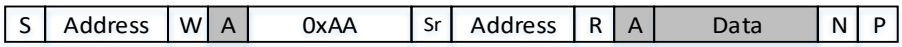
IP Version(0xAA)

IP 版本指令。I2C 接口发送此指令，读回的 Data 为固定值。

注！

当前版本，Version_Data=0x20。

图 4-8 IP 版本指令



4.3 读取芯片 Code API

4.3.1 API 介绍

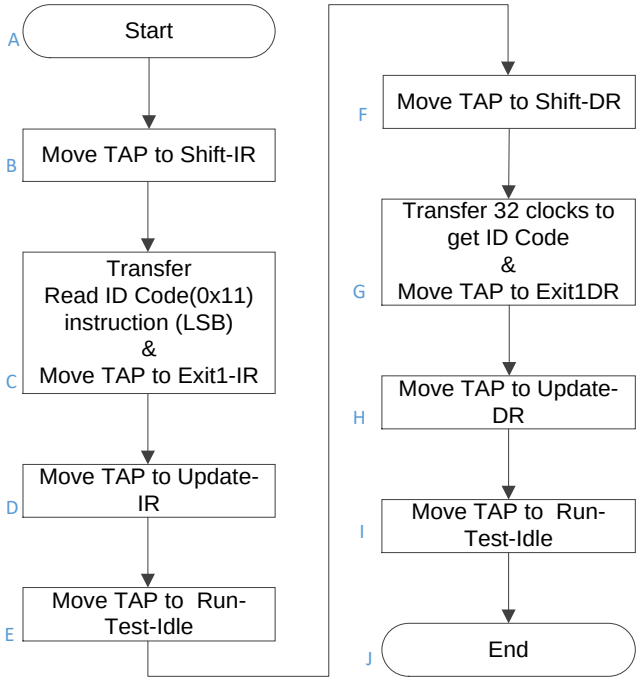
读取 FPGA 芯片的 ID Code、Status Code、User Code；例如：code = gw_i2c_read_code(0x11); code 值返回：0x10281B

4.3.2 读取 ID Code 原理

下图是 JTAG 读取 ID Code 状态机流程图，分别编号为 A、B、C、D、E、F、G、H、I、J；IP 读取 ID Code 时，需要构造如下图 JTAG 状态机转换的数据包，然后通过 0x88 指令和 0x66 指令发送数据包给 IP，IP 在内部转到 JATG 接口实现 IDcode 的读取。

数据包的构造如图 4-9 所示。

图 4-9 数据包的构造图



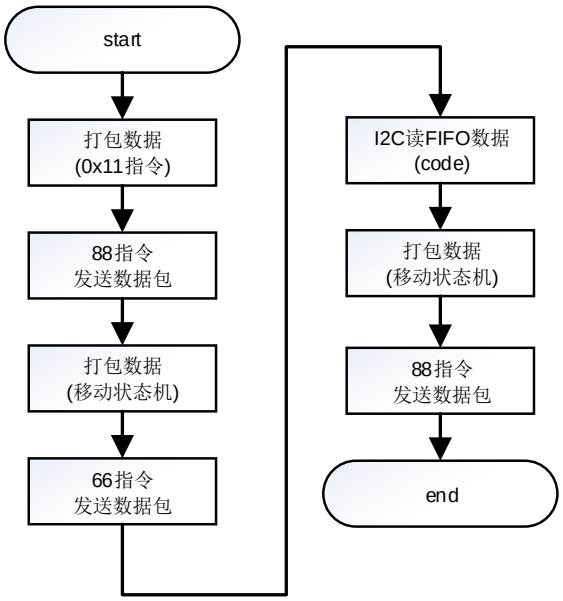
4.3.3 I2C 读取 Code 流程

I2C 读取 Code 流程如下所示：

1. 打包数据：打包 A、B、C、D、E、F 状态机转换及 0x11 数据；
2. 通过 0x88 指令发送数据包；0x88 指令数据包格式参考章节 4.2 IP 基本指令；
3. 打包数据：打包 G、H 状态机跳转数据；
4. 通过 0x66 指令发送数据包；0x66 指令数据包格式参考章节 4.2 IP 基本指令；
5. I2C 读 FIFO 数据(4 Byte code 数据)；
6. 打包数据：打包 I、J 状态机跳转数据；
7. 通过 88 指令发送数据包。

I2C 读取 Code 流程图如图 4-10 所示。

图 4-10 I2C 读取 Code 流程图



5 示例代码移植

5.1 代码接口介绍

表 5-1 代码接口介绍

接口类型	API	备注
上层功能接口	gw_i2c_send_reset gw_i2c_send_reconfig_n gw_i2c_read_version gw_i2c_read_code gw_i2c_read_reg_status gw_i2c_erase flash_tsmc gw_i2c_prog_flash_tsmc i2c_configmode_program_sram gw_i2c_send_runtest_xus match_code get_idcode_from_file	直接调用，无需修改。
	delay_ms delay_us	需要根据自己的平台进行修改 这些接口在delay.c 文件中
中间层接口（数据打包、指令组合等）	write_inst write_inst_and_data i2c_send_pack_tdi_tms write_move_tapdata write_data_at_tap_shiftdr i2c_recv_pack_tdo i2c_jtag_waiting_count i2c_prog_xpage write_one_x_address write_one_y_page	直接调用，无需修改。
底层I2C读写接口	i2c init i2c_write_buffer i2c_write_buffer_2 i2c_write_byte i2c_read_buffer i2c_read_bufferxx	需要根据自己的平台进行修改，实现I2C的初始化和I2C基本的读写； 这些接口在：bsp_i2c.c 文件中。

5.1.1 上层功能接口

上层功能接口在 main.c 中直接调用，上层接口会调用中间层接口从而实现一些功能，上层接口也会直接调用底层读写接口实现一些简单的功能。

5.1.2 中间层接口

中间层接口主要实现数据的组合和打包操作，在中间层会调用底层 I2C 读写接口发送打包数据。

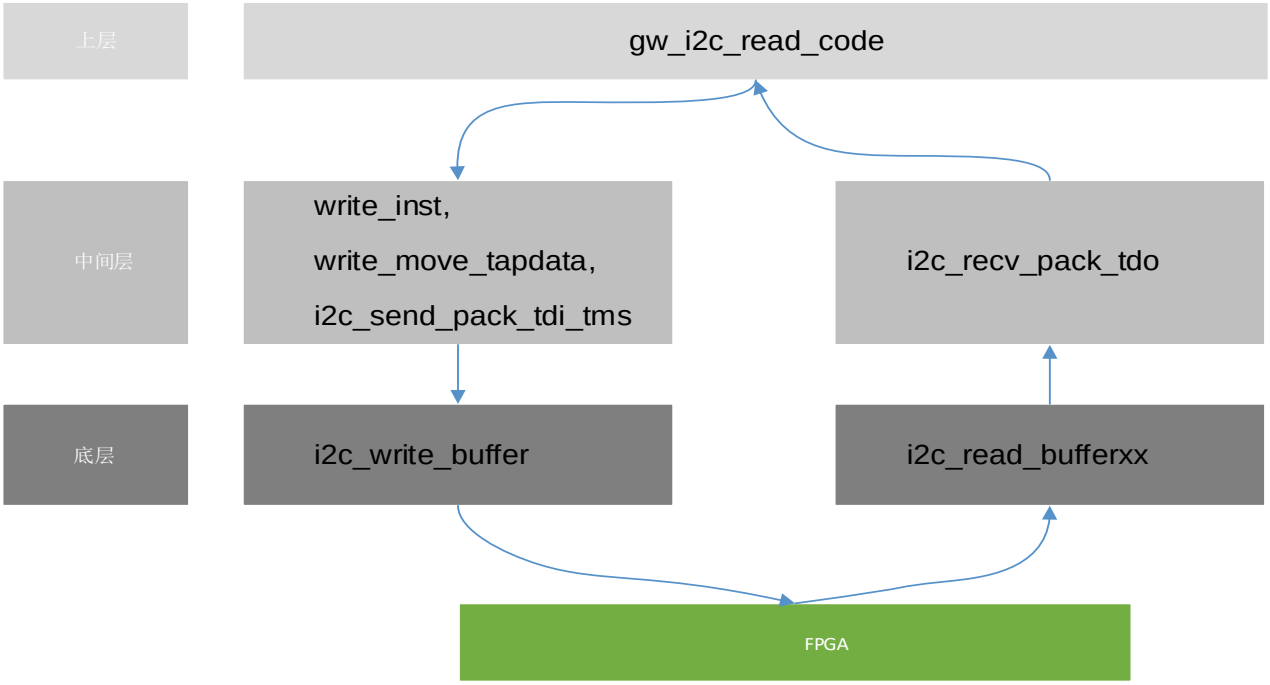
5.1.3 底层 I2C 读写接口

底层读写接口会被上层功能接口和中间层接口调用，因此这个底层接口的名称最好不要改动，只需要在对应的函数内实现 i2c 初始化、读、写功能，如：i2c_init,i2c_read_buffer,i2c_read_bufferxxx, i2c_write_buffer，需要注意 i2c_write_buffer_2 这个接口是针对配置 SRAM 使用，这里配置 SRAM 是通过 FPGA 自带的硬件配置模块实现。goConfigIP-I2C 支持标准 I2C 协议，I2C 的读、写过程是固定的步骤。

我们提供的示例代码底层接口包含两种模式，一种是 MCU 自带的硬件 I2C，另外一种是 GPIO 模拟 I2C，通过宏 USE_HARDWARE_I2C 来进行 I2C 模式选择；如果#define USE_HARDWARE_I2C 则选择使用硬件 I2C。

5.1.4 接口调用关系

图 5-1 接口调用关系



5.2 底层接口替换说明

5.2.1 i2c_init

i2c_init 函数接口主要实现管脚定义、i2c 模式配置、i2c 速率设置，需

要根据自己的平台进行改写。

如：

```
void i2c_init(void)
{
  I2C_GPIO_Config();
  I2C_Mode_Config();
}
```

下文描述为简写，例如：

- i2c start 表示为 Start
- i2c stop 表示为 Stop
- fpga device IP address 表示为 ADDRESS（如地址为 7bit：1011000（0x58））
- write control bit 表示为 WR
- read control bit 表示为 RD
- 检查 ACK 表示为 check ACK
- Command Address 表示为 CMD ADDRRESS

5.2.2 i2c_write_buffer

这个函数接口主要实现 i2c 写数据，根据自己的平台修改，实现以下步骤即可实现 i2c 写数据的功能。

实现步骤是：

1. Start
2. send (ADDRESS<<1) | WR
3. check ACK
4. send CMD ADDRRESS
5. check ACK
6. send n byte data
7. check ACK（每发送一 byte 数据都需要进行一次 checkACK）
8. Stop

5.2.3 i2c_read_buffer

这个函数接口主要实现 i2c 读数据（主要用于 0x44 指令 read status register 和 0xAA 指令读取 IP Version），根据自己的平台修改，实现以下步骤即可实现 i2c 读数据的功能；

1. Start
2. send (ADDRESS<<1) | WR
3. check ACK
4. send CMD ADDRRESS
5. check ACK
6. Start
7. send (ADDRESS<<1) | RD
8. check ACK
9. i2c receive n byte data
10. NACK（最后 1byte 发送 NACK）
11. Stop

5.2.4 i2c_read_bufferxx

这个函数接口主要实现 i2c 读数据（主要用于 0x66 指令 Send data and read back Flash interface data 之后，去读取 FIFO 中缓存的数据），根据自己的平台修改，实现以下步骤即可实现 i2c 读数据的功能；

1. Start
2. send (ADDRESS<<1) | RD
3. check ACK
4. i2c receive n byte data
5. NACK（最后 1byte 发送 NACK）
6. Stop

注！

i2c_read_bufferxx 与 i2c_read_buffer 的差异是省略了 send ADDRESS | WR 步骤；如果使用的是 linux kernel 库，可能需要修改增加这个接口函数的实现。

5.3 移植要点

5.3.1 准备工作

1. 准备一个基本的 i2c 读写工程(basic_i2c_project)，能在目标平台下运行；
2. i2c 读写工程(basic_i2c_project)，实现封装可以多字节写入的函数；
   - 传入参数及命名为：uint32_t i2c_write_buffer(u8 *pBuffer, u8 WriteAddr, u32 NumByteToWrite)；（参照 5.2 底层接口替换说明实

现）

- 封装一个可以实现多字节读取的函数：uint32_t i2c_read_buffer(u8 *pBuffer, u8 ReadAddr, u16 NumByteToRead)，（参照 5.2 底层接口替换说明实现）
- 封装一个可以实现多字节读取的函数：uint32_t i2c_read_bufferxx(u8 *pBuffer, u8 ReadAddr, u16 NumByteToRead)，（参照 5.2 底层接口替换说明实现）

3. 准备好示例代码，主要使用这几个文件：main.c i2c_jtag.c,i2c_jtag.h，bsp_i2c.h, bsp_i2c.c;
4. 将上层、中间层函数接口拷贝到目标平台下的 basic_i2c_project 工程中。

5.3.2 编译与验证

1. 移植完成后先编译一下，确保编译正确无警告及错误提示；
2. 将目标平台与 FPGA 的 I2C 连接；
3. 并执行 gw_i2c_send_reset、gw_i2c_read_version 接口，观察打印信息能读到版本号则说明工作正常。

5.3.3 问题排查

1. 调用测试接口 gw_i2c_read_version 如果不能读到版本号：

   a) 需要排查一下接线是否正常，SDA、SCL、GND 连接是否良好；

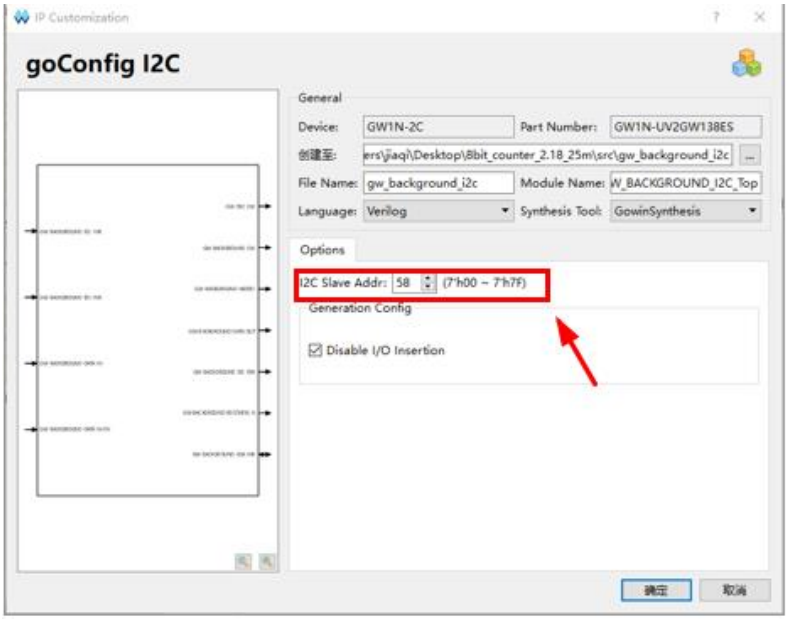
   b) MCU 的 I2C 是否有信号输出，可以使用示波器或逻辑分析仪抓一下信号看一下；

   c) FPGA 是否有 IP 功能运行，FPGA 不能是空片，必须下载 goConfig IP 到芯片；

2. 如果发送了器件地址，没有 ACK 响应：

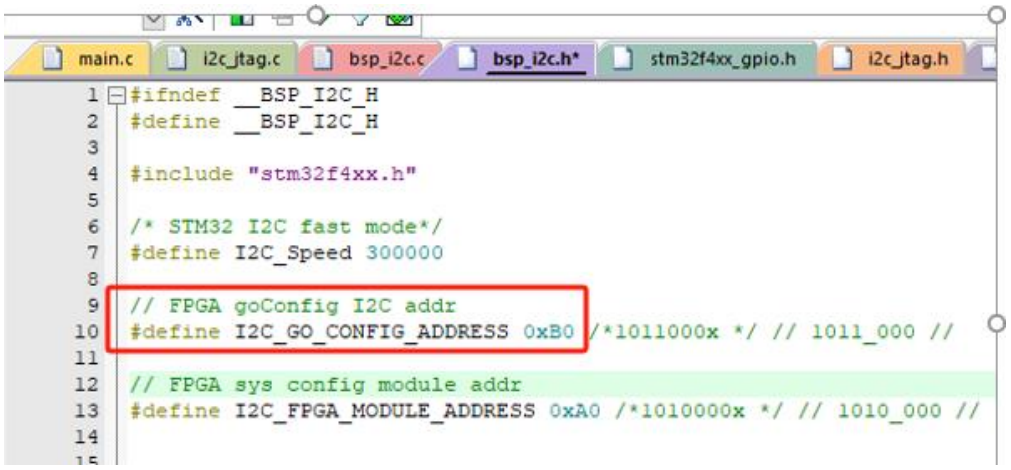
   a) 查看一下 goConfigIP 设置的地址是否匹配：The i2c address of goconfigIP is set to 0x58 (1011 000);

图 5-2 查看地址



b) In the C code, the macro is defined as: #define I2C_GO_CONFIG_ADDRESS 0xB0（1011 0000），0xB0 是 0x58 向左移 1 位得到（0xB0 = 0x58<<1;嵌入式端 7bit 也是按照 8bit 运算处理）。

图 5-3 宏定义



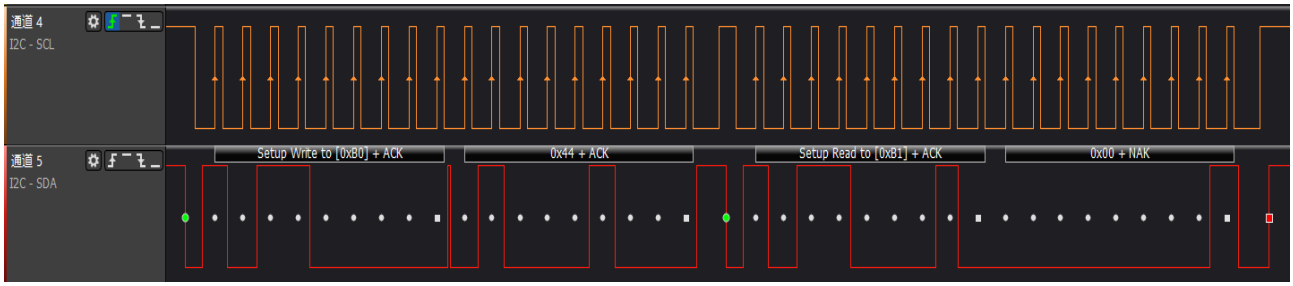
5.3.4 参考时序图

下图是执行 gw_i2c_send_reset 波形时序（器件地址为：7bit,1011000，再发送 0x55 指令）。

图 5-4 波形时序图

下图是执行 gw_i2c_read_reg_status 波形时序（发送一个 0x44 指令，后面回读数据为 0x00）。

图 5-5 波形时序图




GOWIN高云
智慧逻辑 定制未来