



Gowin goConfig IP(I2C) Example Code

Application Note

TN715-1.0E,2024-02-02

Copyright © 2024 Guangdong Gowin Semiconductor Corporation. All Rights Reserved.

GOWIN is a trademark of Guangdong Gowin Semiconductor Corporation and is registered in China, the U.S. Patent and Trademark Office, and other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders. No part of this document may be reproduced or transmitted in any form or by any denotes, electronic, mechanical, photocopying, recording or otherwise, without the prior written consent of GOWINSEMI.

Disclaimer

GOWINSEMI assumes no liability and provides no warranty (either expressed or implied) and is not responsible for any damage incurred to your hardware, software, data, or property resulting from usage of the materials or intellectual property except as outlined in the GOWINSEMI Terms and Conditions of Sale. GOWINSEMI may make changes to this document at any time without prior notice. Anyone relying on this documentation should contact GOWINSEMI for the current documentation and errata.

Revision History

Date	Version	Description
2024/02/02	1.0E	Initial version published.

Contents

Contents	i
List of Figures	iii
List of Tables	iv
1 About This Note	1
1.1 Purpose	1
1.2 Related Documents	1
1.3 Terminology and Abbreviations	1
1.4 Support and Feedback	2
2 Introduction to the Sample Code	3
2.1 Introduction to the Sample Code	3
2.2 File Directory	3
2.2.1 File Description	4
3 Functional Description	5
3.1 Function Overview	5
3.2 Code Execution Flow	5
3.2.1 Check MCU-IP Communication for Normalcy	5
3.2.2 Overall Process of Background Programming	6
4 API Interface Description	11
4.1 Summary of Commonly Used API	11
4.2 IP Commands	11
4.2.1 Command Definition	11
4.3 API for Reading Chip Code	15
4.3.1 API Introduction	15
4.3.2 Principle of Reading ID Code	15
4.3.3 Process of Reading Code via I2C	16
5 Example Code Porting	18
5.1 Code Interface Introduction	18
5.1.1 Upper-level Function Interface	19

5.1.2 Middle-level Interface	19
5.1.3 Underlying I2C Read/Write Interface	19
5.1.4 Interface Calling Relationship	19
5.2 Underlying Interface Replacement Description.....	20
5.2.1 i2c_init	20
5.2.2 i2c_write_buffer	20
5.2.3 i2c_read_buffer.....	21
5.2.4 i2c_read_bufferxx	21
5.3 Key Points for Porting.....	21
5.3.1 Preliminary.....	21
5.3.2 Compilation and Verification	22
5.3.3 Troubleshooting	22
5.3.4 Reference Timing Diagram	23

List of Figures

Figure 3-1 Block Diagram	5
Figure 3-2 Flowchart of Checking MCU-IP Communication	6
Figure 3-3 Flowchart of Background Programming	7
Figure 3-4 Flowchart of Configuring SRAM	8
Figure 3-5 Flowchart of Erasing Embedded Flash.....	9
Figure 3-6 Flowchart of Programming Embedded Flash	10
Figure 4-1 Example of Read Status Register Command	13
Figure 4-2 Example of Reset IP Command	14
Figure 4-3 Example of Sending Read-back Command.....	14
Figure 4-4 Example of Read FIFO Command	14
Figure 4-5 Example of Waiting Command	15
Figure 4-6 Example of Sending Data and No Read Back Flash Data Command	15
Figure 4-7 Example of Reconfign Command.....	15
Figure 4-8 IP version Command	15
Figure 4-9 Construction of the Packet	16
Figure 4-10 Flowchart of Reading Code via I2C	17
Figure 5-1 Interface Calling Relationship.....	19
Figure 5-2 View Address.....	23
Figure 5-3 Macro Definition	23
Figure 5-4 Waveform Timing Diagram	24
Figure 5-5 Waveform Timing Diagram	24

List of Tables

Table 1-1 Terminology and Abbreviations	1
Table 2-1 File Description	4
Table 4-1 Summary of Commonly Used API.....	11
Table 4-2 2C Series Commands	11
Table 4-3 Status Register	13
Table 5-1 Code Interface Introduction.....	18

1 About This Note

1.1 Purpose

Gowin goConfig IP(I2C) Example Code Application Note include an introduction to the sample code, a description of its functions, a description of the API interfaces, and instructions on how to use it.

Note!

- This example code is written in C language, and a complete project package based on the embedded platform (STM32F429) is provided, using the keil_V5 IDE.
- It is applicable for configuring the GW1N-1P5C/GW1N(R)-2C series.
- We need to load a bitstream which contains goConfig I2C IP before MCU can access goConfig I2C IP.

1.2 Related Documents

The latest user guides are available on the GOWINSEMI Website. You can find the related documents at www.gowinsemi.com.

- [DS100, GW1N series of FPGA Products Data Sheet](#)
- [DS117, GW1NR series of FPGA Products Data Sheet](#)
- [IPUG795, Gowin goConfig I2C IP User Guide](#)
- [UG290, Gowin FPGA Products Programming and Configuration Guide](#)

1.3 Terminology and Abbreviations

The terminology and abbreviations used in this manual are as shown in Table 1-1.

Table 1-1 Terminology and Abbreviations

Terminology and Abbreviations	Meaning
ACK	Acknowledge
FPGA	Field Programmable Gate Array
I ² C(I2C)	Inter-Integrated Circuit
IP	Intellectual Property
NACK	Not ACKnowledge
SRAM	Static Random Access Memory

1.4 Support and Feedback

Gowin Semiconductor provides customers with comprehensive technical support. If you have any questions, comments, or suggestions, please feel free to contact us directly by the following ways.

Website: www.gowinsemi.com

E-mail: support@gowinsemi.com

2 Introduction to the Sample Code

2.1 Introduction to the Sample Code

When users port and use this sample code, they can directly use our pre-packaged upper-level API interface (located in `i2c_jtag.c`). Modifications to the low-level I2C data transmission interface (located in `bsp_i2c.c`) can be made according to their own platform requirements.

2.2 File Directory

```

├──goConfigIP(I2C) Example Code Application Note.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

```

```
|          └─┬─uart
|              └─ bsp_debug_uart.c
|              └─ bsp_debug_uart.h
```

2.2.1 File Description

Table 2-1 File Description

File	Description
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 Functional Description

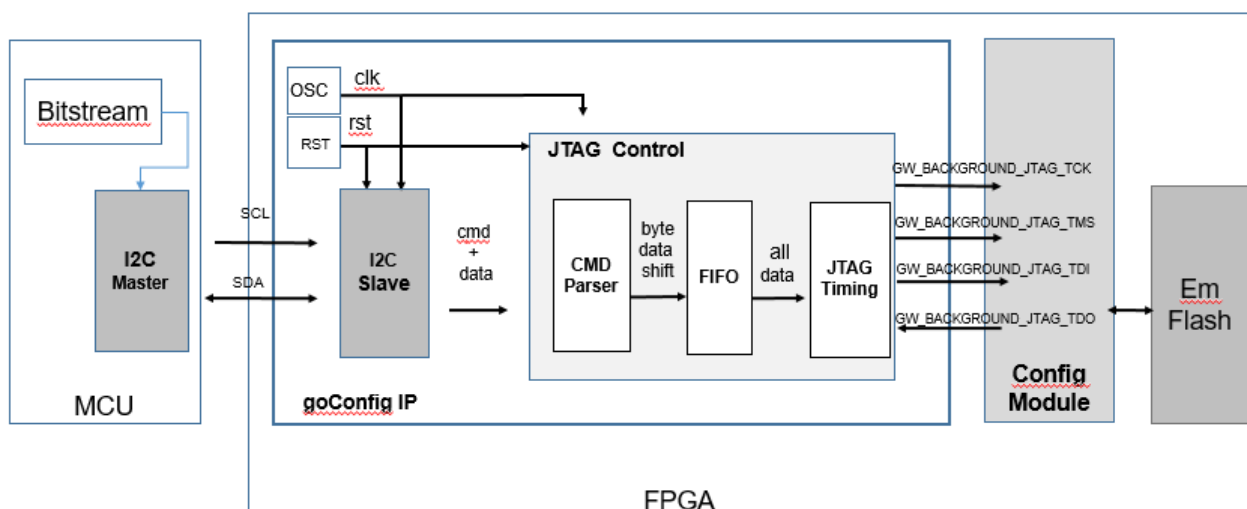
3.1 Function Overview

This example code is executed in an MCU, and with the help of goConfigIP through the I²C interface, it can perform erasing and programming operations on the embedded Flash. It can also read the FPGA's ID code, user code, and status code.

Note!

Before you use the goConfig IP, the FPGA needs to load the bitstream containing the goConfig I2C IP.

Figure 3-1 Block Diagram



3.2 Code Execution Flow

3.2.1 Check MCU-IP Communication for Normalcy

Before starting the background programming, it is important to ensure that communication between the MCU and IP is functioning properly. The check process is as follows:

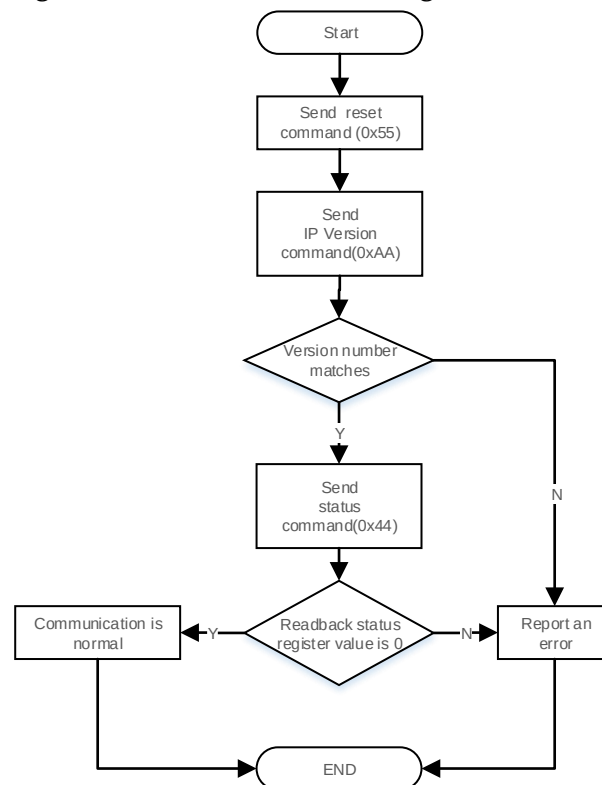
1. Send a reset command (0x55) to ensure that the IP is in its initial status (API: gw_i2c_send_reset).
2. Send an IP Version command (0xAA) to check the read-back version value (e.g., Version Data=0x20) to ensure that the IP version matches

the expected one (API: gw_i2c_read_version).

3. Send Status command (0x44) and check the read-back status value (API: gw_i2c_read_reg_status).
4. When the operation is completed and the read-back data matches the expectation, it indicates that the communication with the IP is ready to start.

The flowchart of checking MCU-IP communication is shown in Figure 3-2.

Figure 3-2 Flowchart of Checking MCU-IP Communication

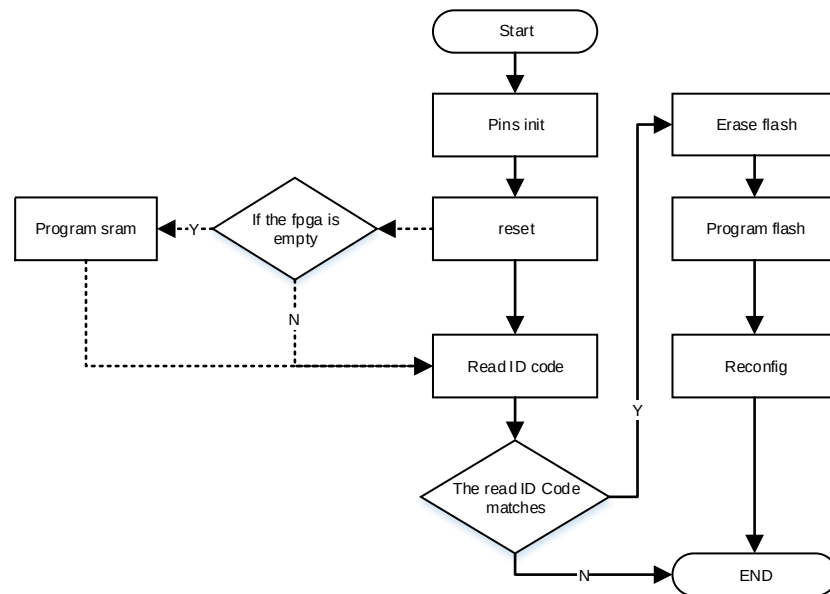


3.2.2 Overall Process of Background Programming

The background programming process is as follows:

1. Initialize MCU platform pins (I2C, debug_uart).
2. Reset the IP (API: gw_i2c_send_reset).
3. If the chip is not blank, skip this step. If the chip is blank, configure SRAM (API: i2c_configmode_program_sram).
4. Compare the ID code read by the MCU (I2C) with the ID code in the data stream file (API: mach_code). If they match, continue; if not, terminate the background programming.
5. Erase Flash (API: gw_i2c_erase_flash_tsmc).
6. Program Flash (API: gw_i2c_prog_flash_tsmc).
7. Reconfig (gw_i2c_send_reconfig_n).

The flowchart of background programming is shown in Figure 3-3.

Figure 3-3 Flowchart of Background Programming

Read FPGA Code

Use API: `gw_i2c_read_code` to read FPGA code. Once communication between the MCU and IP is normal, you can use I2C to read FPGA code.

- ID code, also known as JEDEC ID Code, is a fundamental identifier for the FPGA device.
- User code is a user's identifier for the FPGA device they are using.
- Status code is a status identifier for the FPGA device after loading the data stream file.

For example:

- To read the ID Code, use the API: `gw_i2c_read_code(0x11)`.
- To read the User Code, use the API: `gw_i2c_read_code(0x13)`.
- To read the Status Code, use the API: `gw_i2c_read_code(0x41)`.

Configure SRAM in a Blank Chip via I2C

To configure SRAM in a blank chip, use the API: `i2c_configmode_program_sram`.

Note!

Blank FPGA chips do not have IP. Configuring SRAM is done through the FPGA's internal native hardware configuration module, which only supports SRAM configuration. If you want to program the embedded Flash, use the `goConfig` I2C IP.

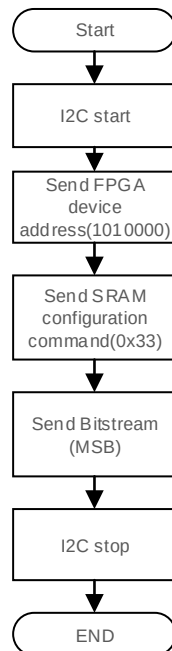
The process for configuring SRAM in a blank chip via I2C is as follows:

1. Send start signal
2. Send device address
3. Send SRAM configuration command (0x33)
4. Send Bitstream (in MSB format)

5. Send stop signal

The flowchart of configuring SRAM is shown in Figure 3-4.

Figure 3-4 Flowchart of Configuring SRAM

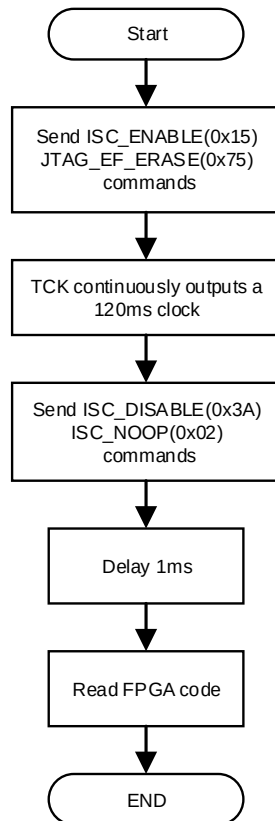


Erase Embedded Flash

Use the API: `gw_i2c_erase_flash_tsmc` to erase the embedded Flash. The process for erasing the embedded flash is as follows:

1. Send `ISC_ENABLE (0x15)` and `JTAG_EF_ERASE (0x75)` commands. These two commands are packaged using the API (`write_inst_and_data`) and sent to the FPGA through the MCU's I2C low-level interface (API: `i2c_write_buffer`).
2. TCK continuously outputs 120ms clock (API: `gw_i2c_send_runtest_xus`).
3. Send `ISC_DISABLE (0x3A)` and `ISC_NOOP (0x02)` commands. These two commands are packaged using the API (`write_inst_and_data`) and sent to the FPGA through the MCU's I2C low-level interface (API: `i2c_write_buffer`).
4. Delay 1ms (API: `gw_i2c_send_runtest_xus`).
5. Read FPGA code (API: `gw_i2c_read_code`).

The flowchart of erasing embedded Flash is shown in Figure 3-5.

Figure 3-5 Flowchart of Erasing Embedded Flash**Program Embedded Flash**

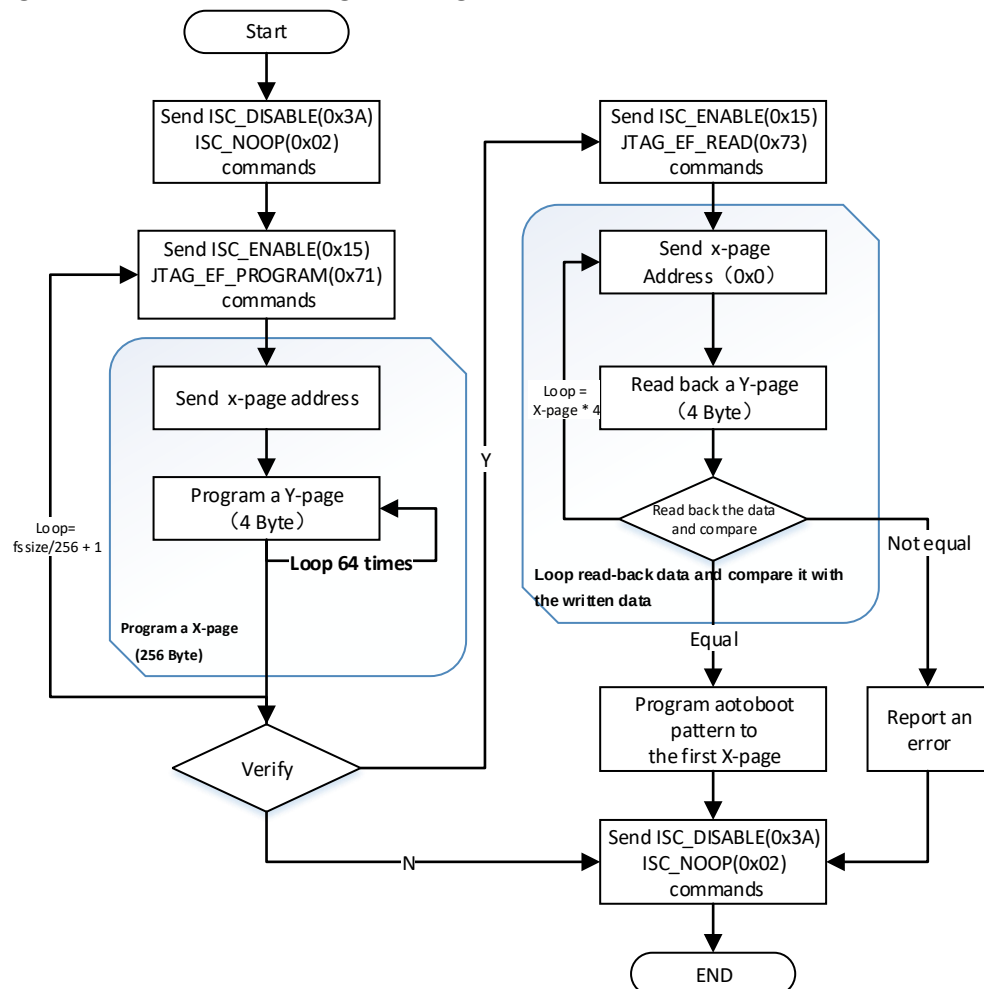
Use the API: `gw_i2c_prog_flash_tsmc` to program the embedded Flash. The process for programming the embedded flash is as follows:

1. Send `ISC_DISABLE (0x3A)` and `ISC_NOOP (0x02)`. These two commands are packaged using the API (`write_inst_and_data`) and sent to the FPGA through the MCU's I2C low-level interface (API: `i2c_write_buffer`).
2. Send `ISC_ENABLE (0x15)` and `JTAG_EF_PROGRAM (0x71)`. These two commands are packaged using the API (`write_inst_and_data`) and sent to the FPGA through the MCU's I2C low-level interface (API: `i2c_write_buffer`).
3. Send the X-page address. This address data is packaged using the API (`write_one_x_address`) and sent to the FPGA through the MCU's I2C low-level interface (API: `i2c_write_buffer`).
4. Program a Y-page. One X-page contains 64 Y-pages, and this Y-page data is packaged using the API (`write_one_y_page`) and sent to the FPGA through the MCU's I2C low-level interface (API: `i2c_write_buffer`).
5. If read-back verification is enabled, send `ISC_ENABLE (0x15)` and `JTAG_EF_READ (0x73)`. These two commands are packaged using the API (`write_inst_and_data`) and sent to the FPGA through the MCU's I2C low-level interface (API: `i2c_write_buffer`).

6. Send the X-page address data 0x00. This address data is packaged using the API (write_one_x_address) and sent to the FPGA through the MCU's I2C low-level interface (API: i2c_write_buffer).
7. Loop the read-back data from Flash and compare it with the data written in the data stream file (API: recv_one_y_page_and_verify).
8. After the read-back loop comparison, write the autoboot-pattern data to the first X-page. This autoboot-pattern data is packaged using the API (i2c_prog_xpage) and sent it to the FPGA through the MCU's I2C low-level interface (API: i2c_write_buffer).
9. Send ISC_DISABLE (0x3A) and ISC_NOOP (0x02). These two commands are packaged using the API (write_inst_and_data) and sent to the FPGA through the MCU's I2C low-level interface (API: i2c_write_buffer).

The flowchart of programming embedded Flash is shown in Figure 3-6.

Figure 3-6 Flowchart of Programming Embedded Flash



4 API Interface Description

4.1 Summary of Commonly Used API

Table 4-1 Summary of Commonly Used API

Name	Function	Note
gw_i2c_read_code()	Read FPGA chip code	Can read ID Code, Status Code, User Code
gw_i2c_prog_flash_tsmc()	Program Em-Flash	—
gw_i2c_erase_flash_tsmc()	Erase Em-Flash	
gw_i2c_send_reset()	Reset IP	
gw_i2c_read_version()	Read IP version	

4.2 IP Commands

4.2.1 Command Definition

Gowin goConfig I2C IP 2C series commands are as shown in Table 4-2.

Table 4-2 2C Series Commands

No.	Name	Command Address	Description	Command Format	Definition in the Example Code	Note
1	Status	0x44	Read status register	CMD	READ_STATUS_INST	—
2	Reset	0x55	Reset IP	CMD	IP_RESET_INST	
3	Send (data back)	0x66	Send data and read back Flash interface data	CMD + Length + Data + Check_sum	SEND_DATA_B_ACK_INST	
4	Waiting	0x77	Waiting	CMD + Length + Check_sum	WAITING_INST	
5	Send (No data back)	0x88	Send data and no read back Flash interface data	CMD + Length + Data + Check_sum	SEND_NO_DATA_BACK_INST	

No.	Name	Command Address	Description	Command Format	Definition in the Example Code	Note
6	Reconfig	0x99	Reconfiguration	CMD	RECONFIG_N_I NST	
7	IP Version	0xAA	IP Version	CMD	IP_VERSION_IN ST	—

The command is defined and described as follows:

- **CMD:** See Table 3-1 for the command support.
 - Definition: Command address
 - Width: 8 bits
 - **Length:** Minimum 2
 - Definition: Valid byte length to be sent ^[1]
 - Width: 16 bits, low 8 bits (Length_L) are sent first, high 8 bits (Length_H) are sent later.
 - Formula: Length = valid data length + 2.
- Note!**
- [1] Valid byte length is the valid data (Data) length + 2; take 0x66 command as an example, Length = Data length + 2.
 - For example, if the expected input byte length is 10, then Length = 10 + 2 = 12.
- **Data:**
 - Definition: Sent valid data
 - Width: 8 bits for one data
 - Length: The actual length of the data sent is determined according to Length-2; Data_0 indicates the first valid data; Data_1 indicates the second valid data, ... Data_N means the last valid data.
 - **Check_sum:**
 - Definition: Check
 - Width: 16 bits, high 8 bits (Check_sum_H) are sent first, low 8 bits (Check_sum_L) are sent later.
 - Formula: Check_sum = CMD + Length_L + Length_H + Data_0 + ... + Data_N

Note!

When Check_sum calculation overflows, only the lower 16 bits are taken and the rest are discarded. Suppose Check_sum=0x10F11, then Check_sum_H=0x0F; Check_sum_L=0x11, and the rest are discarded.

Status(0x44)

Read status register command, read only. The data read back is the value of the status register, and the status register is defined as shown in Table 4-3.

Table 4-3 Status Register

Bit(s)	Default	Description	Note
Bit[7:3]	0	Reserved	—
Bit [2]	0	1: Flash interface operating 0: Flash interface idle	
Bit[1]	0	1: Check error 0: Check correct	
Bit[0]	0	1: I ² C status error 0: I ² C status normal	—

An example of read status register command is as shown in Figure 4-1.

Figure 4-1 Example of Read Status Register Command

S	Address	W	A	0x44	Sr	Address	R	A	Data	N	P
---	---------	---	---	------	----	---------	---	---	------	---	---

Note!

In the following, unless otherwise specified, the names in similar diagrams have the same meanings as this example.

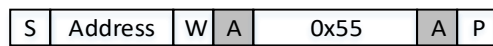
- For white background in the figure, it means I²C master sends to I²C slave; for gray background, it means I²C slave sends to I²C master.
- The names shown in the figure are defined as follows:
 - S: START in I²C bus
 - Address: Slave Address in I²C bus
 - W: WRITE in I²C bus
 - A: ACK in I²C bus
 - Sr: Repeated START in I²C bus
 - R: READ in I²C bus
 - Data: Transferred valid data in I²C bus
 - N: NACK in I²C bus
 - P: STOP in I²C bus
 - Length_L: Lower 8 bits of Length
 - Length_H: High 8 bits of Length
 - Data_0: First valid data
 - Data_N: Last valid data
 - Check_sum_H: High 8 bits of Check_sum
 - Check_sum_L: Low 8 bits of Check_sum

Reset(0x55)

Reset IP command. It can reset internal status of the IP, as well as clear the internal FIFO.

Note!

This process only involves the IP, not the background programming.

Figure 4-2 Example of Reset IP Command**Send(0x66,data back)**

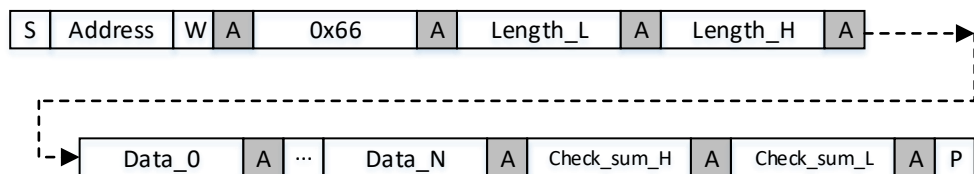
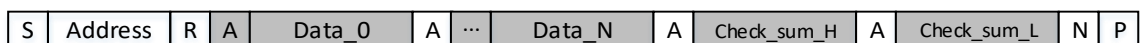
Send data and read back Flash data command. The I²C interface sends the byte length of the data to be read back, and sends the data at the same time; if there is no check error, the IP will be converted to the Flash interface and the data will be read back into the FIFO, as shown in Figure 3-6. At this time, if users need this data, a read command can be sent through the I²C interface, you can read the data in the FIFO, as shown in Figure 3-7.

In brief, the process of read back includes two steps.

1. Read the data into the FIFO of the IP.
2. Read the data in the FIFO through the I²C bus.

Note!

- Some commands can be inserted between step 1 and step 2, such as read status command.
- In Figure 3-6, Data_0~Data_N can be any value, but it is recommended to write 0x00 or 0xFF.
- In Figure 3-7, Data_0~Data_N is the actual read back value.

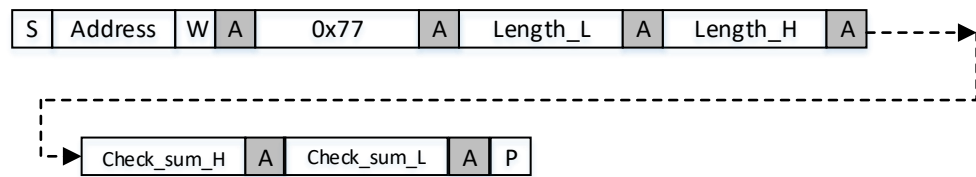
Figure 4-3 Example of Sending Read-back Command**Figure 4-4 Example of Read FIFO Command****Waiting(0x77)**

Waiting command. The I²C interface sends the byte length of waiting; if there is no check error, Flash interface will keep sending the idle clock (no data transmission) until the bytes waiting count is completed.

Note!

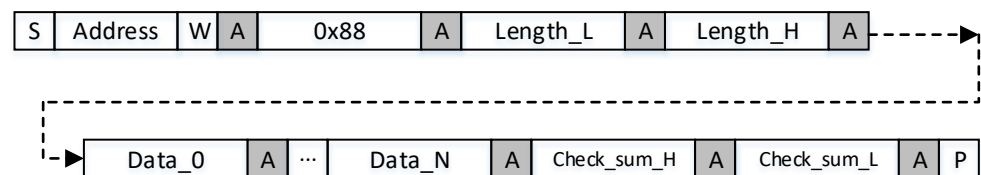
In 2C series, Flash interface clock is 2.5MHz, which means one cycle is 400ns. 1 byte is $8 \times 400 \text{ ns} = 3200 \text{ ns} = 3.2 \mu\text{s}$. For example, if waiting 150ms, the number of valid bytes is calculated as $46875 = 0xB71B$; the number of input bytes is $0xB71B + 2 = 0xB71D$, then Length_L = 0x1D; Length_H = 0xB7.

Figure 4-5 Example of Waiting Command

**Send(0x88, No data back)**

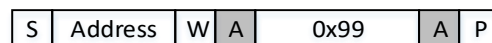
Send data and no read back Flash data command. The I²C interface first stores the data to be sent into the FIFO, after the data count is completed and there is no check error, the data in the FIFO is sent out once through the Flash interface.

Figure 4-6 Example of Sending Data and No Read Back Flash Data Command

**Reconfig(0x99)**

Reconfig command. When the I²C interface sends this command, "Reconfigure" is triggered; for example, "Reconfig_N" pin is pulled down.

Figure 4-7 Example of Reconfig Command

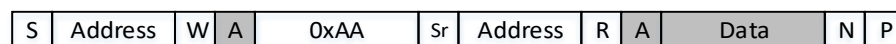
**IP Version(0xAA)**

IP version command. The I²C interface sends this command and read-back data is a fixed value.

Note!

Version_Data=0x20 for the current version.

Figure 4-8 IP version Command



4.3 API for Reading Chip Code

4.3.1 API Introduction

Read FPGA's ID Code, Status Code, and User Code. For example, code = gw_i2c_read_code(0x11); returns code value: 0x10281B.

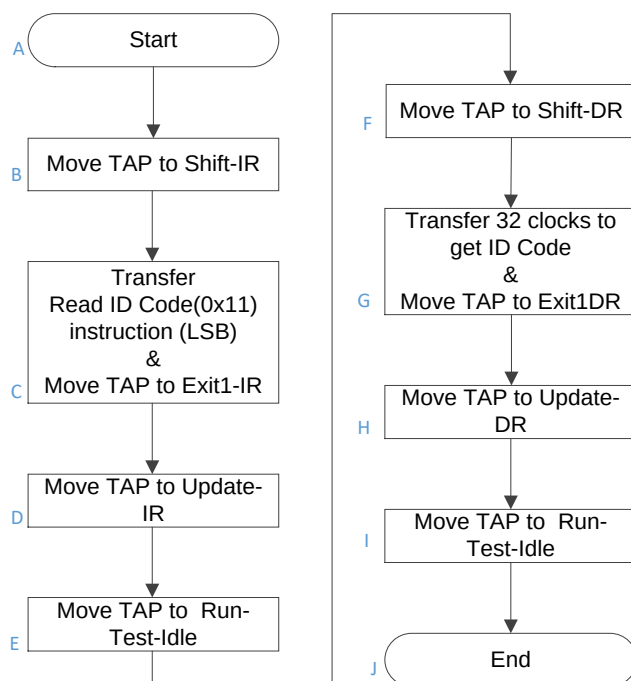
4.3.2 Principle of Reading ID Code

The following is the state machine flowchart for reading the ID Code via JTAG, labeled as A, B, C, D, E, F, G, H, I, and J. When the IP reads the ID Code, it needs to construct packets that follow the JTAG state machine

transition as shown in the diagram below. These packets are then sent to the IP using the 0x88 and 0x66 commands. The IP internally transfers them to the JTAG interface to perform the ID code reading.

The construction of the packets is illustrated in Figure 4-9.

Figure 4-9 Construction of the Packet



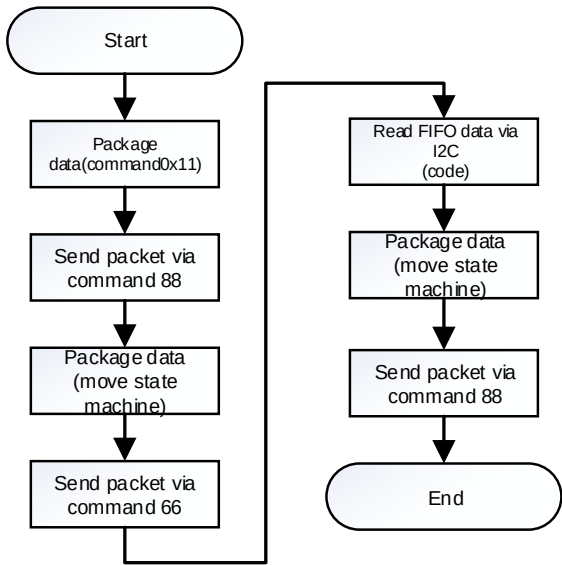
4.3.3 Process of Reading Code via I2C

The process of reading code via I2C is shown below:

1. Package data: Package A, B, C, D, E, F state machine transition and 0x11 data.
2. Send the packet via 0x88 command; for 0x88 packet format, you can see 4.2 IP Commands.
3. Package data: Package G, H state machine transition data.
4. Send the packet via 0x66 command; for 0x66 packet format, you can see you can see 4.2 IP Commands.
5. Read FIFO data via I2C (4-byte code data).
6. Package data: Package I, J state machine transition data.
7. Send the packet via 88 command.

The flowchart of reading code via I2C is shown in Figure 4-10.

Figure 4-10 Flowchart of Reading Code via I2C



5 Example Code Porting

5.1 Code Interface Introduction

Table 5-1 Code Interface Introduction

Interface Type	API	Note
Upper-level function interface	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	Called directly without modification
	delay_ms delay_us	Needs to be modified for your platform, and these interfaces are in the delay.c file.
Middle-level interface (Data packing, command combination, etc.)	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	Called directly without modification
Underlying I2C read/write interface	i2c_init i2c_write_buffer i2c_write_buffer_2 i2c_write_byte i2c_read_buffer i2c_read_bufferxx	Needs to be modified for your platform to implement I2C initialization and basic I2C read/write, and these interfaces are in the: bsp_i2c.c file.

5.1.1 Upper-level Function Interface

The upper-level function interface functions are directly called in main.c. The upper-level interface will call the middle-level interface to realize some functions, and it will also directly call the underlying read/write interface to realize some simple functions.

5.1.2 Middle-level Interface

The middle-level interface mainly realizes the combination and packing of the data, and it will call the underlying I2C read/write interface to send the packed data.

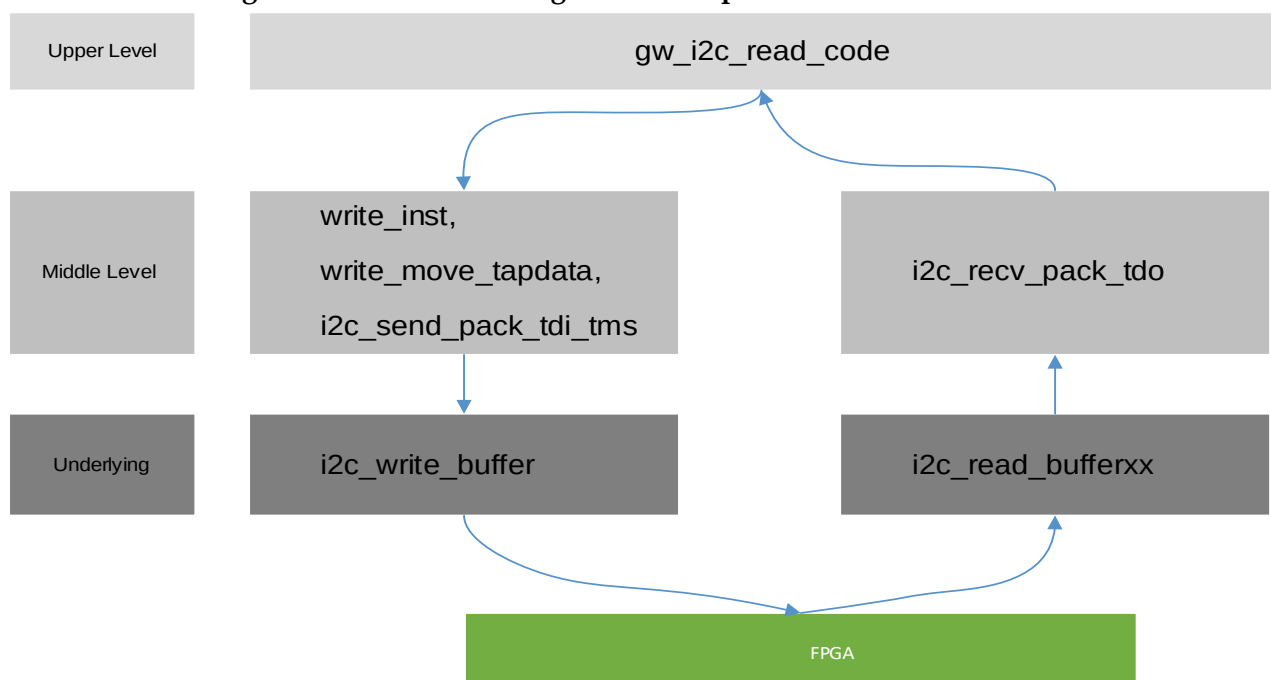
5.1.3 Underlying I2C Read/Write Interface

The underlying read/write interface will be called by the upper-level function interface and the middle-level interface, so it is better not to change the name of the underlying interface, and it only needs to implement the I2C initialization, read and write within the corresponding functions, such as: i2c_init, i2c_read_buffer, i2c_read_bufferxxx, i2c_write_buffer. Note that the interface i2c_write_buffer_2 is for configuring the SRAM, and here the SRAM configuration is realized by the hardware configuration module that comes with the FPGA. goConfigIP-I2C supports the standard I2C protocol, and the I2C read and write process is fixed.

The underlying interface of the sample code we provide contains two modes, one is the hardware I2C that comes with the MCU, and the other is the GPIO-based simulated I2C; the I2C mode selection is controlled by the macro USE_HARDWARE_I2C; If #define USE_HARDWARE_I2C is used, the hardware I2C will be chosen.

5.1.4 Interface Calling Relationship

Figure 5-1 Interface Calling Relationship



5.2 Underlying Interface Replacement Description

5.2.1 i2c_init

The i2c_init function interface mainly realizes pin definition, i2c mode configuration, i2c rate setting, which needs to be modified for your platform.

For example:

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

The following descriptions are in abbreviated form.

For example:

- i2c start is abbreviated as Start
- i2c stop is abbreviated as Stop
- fpga device IP address is abbreviated as ADDRESS (e.g. address 7bit: 1011000 (0x58))
- The write control bit is abbreviated as WR
- The read control bit is abbreviated as RD
- Check acknowledgement is abbreviated as check ACK
- Command Address is abbreviated as CMD ADDRESS

5.2.2 i2c_write_buffer

This function interface mainly realizes i2c write data, which needs to be modified for your platform, and you can follow the steps below to realize the function of i2c write data.

The steps are:

1. Start
2. send (ADDRESS<<1) | WR
3. check ACK
4. send CMD ADDRESS
5. check ACK
6. send n byte data
7. check ACK (a checkACK is required for every byte of data sent)
8. Stop

5.2.3 i2c_read_buffer

This function interface mainly realizes i2c read data, and mainly used for 0x44 command (read status register) and 0xAA command (read IP Version). It also needs to be modified for your platform, and you can follow the steps below to realize the function of i2c read data.

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

5.2.4 i2c_read_bufferxx

This function interface mainly realizes i2c read data, and mainly used to read the data cached in the FIFO after 0x66 command (Send data and read back Flash interface data), It also needs to be modified for your platform, and you can follow the steps below to realize the function of i2c read data.

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

Notes!

The difference between i2c_read_bufferxx and i2c_read_buffer is that the "send ADDRESS | WR" step is omitted; if you are using the linux kernel library, you may need to modify to implement the interface function.

5.3 Key Points for Porting

5.3.1 Preliminary

1. Prepare a basic i2c read/write project (basic_i2c_project) that can run on the target platform.
2. In the i2c read-write project (basic_i2c_project), implement the encapsulation of a function that enables multi-byte writes.

- Input parameters and named as: `uint32_t i2c_write_buffer(u8 *pBuffer, u8 WriteAddr, u32 NumByteToWrite)`, and you can see Section [5.2 Underlying Interface Replacement Description](#) for implementation.
 - Encapsulate a function for multi-byte reads: `uint32_t i2c_read_buffer(u8 *pBuffer, u8 ReadAddr, u16 NumByteToRead)`, and you can see Section [5.2 Underlying Interface Replacement Description](#) for implementation.
 - Encapsulate a function for multi-byte reads: `uint32_t i2c_read_bufferxx(u8 *pBuffer, u8 ReadAddr, u16 NumByteToRead)`, and you can see Section [5.2 Underlying Interface Replacement Description](#) for implementation.
3. Prepare the sample code, and the following files are used, including `main.c` `i2c_jtag.c`, `i2c_jtag.h`, `bsp_i2c.h`, `bsp_i2c.c`.
 4. Copy the interfaces of the upper and middle level functions to the `basic_i2c_project` project on the target platform.

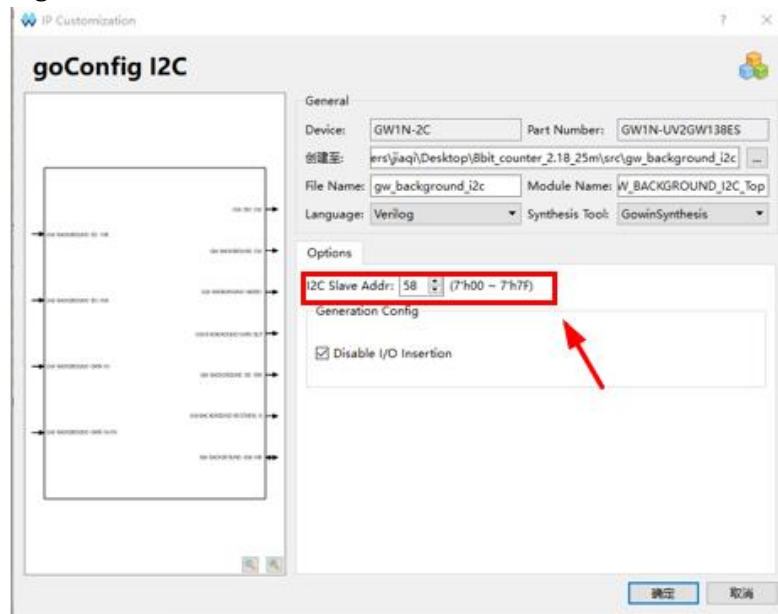
5.3.2 Compilation and Verification

1. After the porting is complete, compile it first to make sure it compiles correctly without warnings or errors.
2. Connect the target platform to the FPGA's I2C.
3. Execute `gw_i2c_send_reset` and `gw_i2c_read_version` interfaces and observe the print information. If it can read the version number, it means that the operation is functioning properly.

5.3.3 Troubleshooting

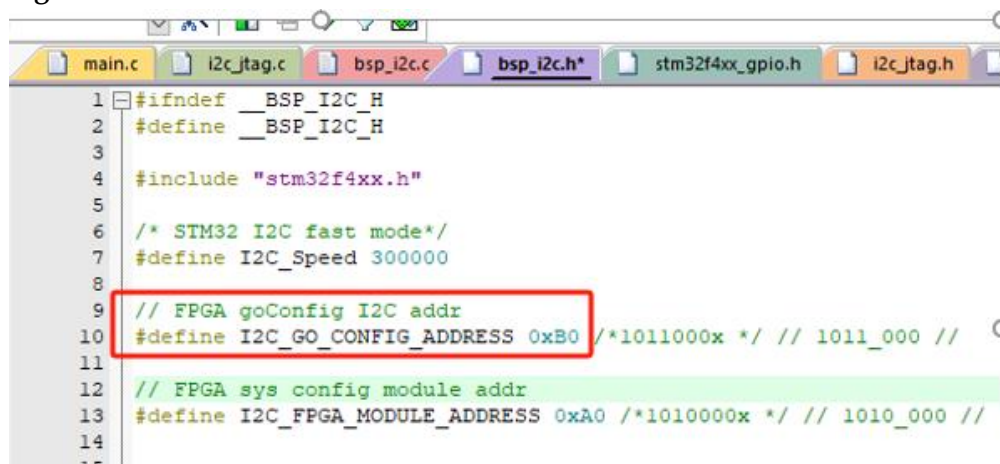
1. If you call the test interface `gw_i2c_read_version`, the version number cannot be read.
 - a) You need to troubleshoot the wiring to make sure the SDA, SCL, and GND connections are good.
 - b) Verify if the MCU's I2C is generating signals, and you can use an oscilloscope or logic analyzer to capture the signal.
 - c) Check if the FPGA is running with IP function; FPGA should not be an empty chip, the chip must be programed with bitstream containing goConfig I2C IP.
2. If the device address is sent, there is no ACK response.
 - a) Check if the address of goconfigIP matches: The i2c address of goconfigIP is set to 0x58 (1011 000).

Figure 5-2 View Address



- b) In the C code, the macro is defined as: `#define I2C_GO_CONFIG_ADDRESS 0xB0` (1011 0000). The value 0xB0 is obtained by shifting 0x58 to the left by 1 bit ($0xB0 = 0x58 \ll 1$; In embedded system, the 7 bits is also processed according to 8-bit operation).

Figure 5-3 Macro Definition



5.3.4 Reference Timing Diagram

The following diagram illustrates the waveform timing for the execution of `gw_i2c_send_reset` (device address: 7-bit, 1011000, followed by sending the 0x55 command).

Figure 5-4 Waveform Timing Diagram



The following diagram illustrates the waveform timing for the execution of `gw_i2c_read_reg_status` (send a 0x44 command, and the read back data is 0x00).

Figure 5-5 Waveform Timing Diagram

