Compa Family CPLDs Device JTAG Interface Configuration and Programming Application Guide

(AN03013, V1.0)

(19.04.2022)

Revisions History

Document Revisions

Version	Date of Release	Revisions
V1.0	19.04.2022	Initial release.

About this Manual

Terms and Abbreviations

Terms and Abbreviations	Meaning
JTAG	Joint Test Action Group
CCS	Configuration Control System
JTAG	Serial Peripheral Interface
UID	Unique ID
FSM	Finite State Machine

Related Documentation

The following documentation is related to this manual:

1. ***UG030004_Compa Family CPLDs Configuration User Guide***

Table of Contents

Tables

Figures

Chapter 1 Document Introduction

Compa family document focuses on the application of the Pango Microsystems CPLDs JTAG configuration mode, the software and hardware environment required for JTAG configuration, and the setup of the internal special function registers of the CPLDs. The document concludes with an actual application example of configuring CPLDs with MCU, providing users with a complete configuration flow for reference.

Chapter 2 Configuration Details

2.1 Boundary Scan and JTAG

The device supports IEEE1149.1 (Boundary Scan Test) and IEEE1532 (In-System Configuration of Programmable Devices) standards.

2.1.1 System Block Diagram

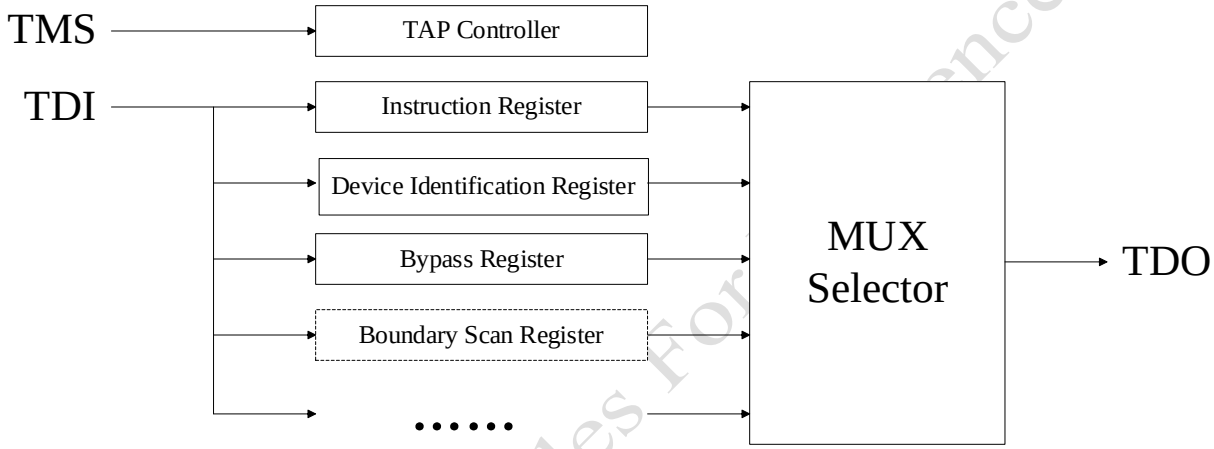

Figure 2-1 Block Diagram of JTAG Boundary Scan System

The hardware structure of the boundary scan consists of four parts:

- Test Access Port (TAP)
- Test Access Port Controller (TAPC)
- Instruction Register (IR)
- Test Data Register (TDR)

2.1.2 Test Access Port

The JTAG interface is shown in the following figure:

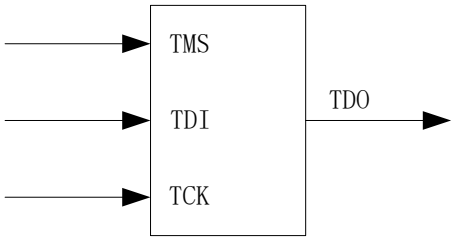

Figure 2-2 JTAG Interface Diagram

The list of JTAG interfaces is as follows:

Table 2-1 List of JTAG Interfaces

Item	I/O	Description
TCK	I	Test Clock. For the maximum supported frequency, please refer to the "DS03001_Compa Family CPLDs Device Datasheet"
TMS	I	Test Mode Select. Used to control the status switching of the TAPC (Test Access Port Controller) state machine on the rising edge of TCK to move test instructions and test data.
TDI	I	Test Data Input and serial input pin. Used to move the test instructions into the instruction register and the test data into the test data register on the rising edge of TCK.
TDO	O	Test data output and serial output pin. During the instruction shift state, it is used to shift the test instructions out from the instruction register on the falling edge of the TCK signal. During the data shift state, it is used to shift out the test data stored in the test data register that is placed on the scan chain from TDI to TDO on the falling edge of the TCK signal.

Interface timing is illustrated in the figure below.

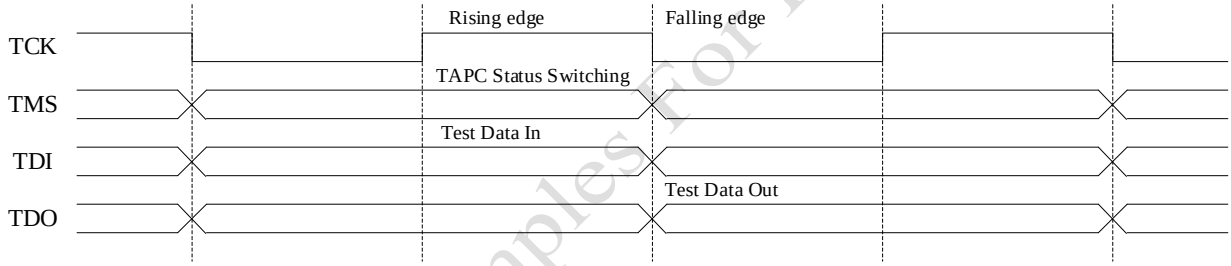

Figure 2-3 JTAG Interface Timing

2.1.3 TAP Controller

The TAP controller is the core that controls the entire JTAG operation. The TAP controller, based on TCK and TMS signal changes, outputs various timings and modes required for the instruction register and test data register. It also generates control signals for the capture, shift, and update of the instruction register and test data register.

The TAP controller is a 16-state synchronous Finite State Machine (FSM). As shown in Figure 2-4, the state machine is divided into three columns corresponding to the reset operation, test data register operation and instruction register operation. The data on the status switching lines is determined by the TMS signal level. The status switching of the TAP controller occurs on the rising edge of TCK.

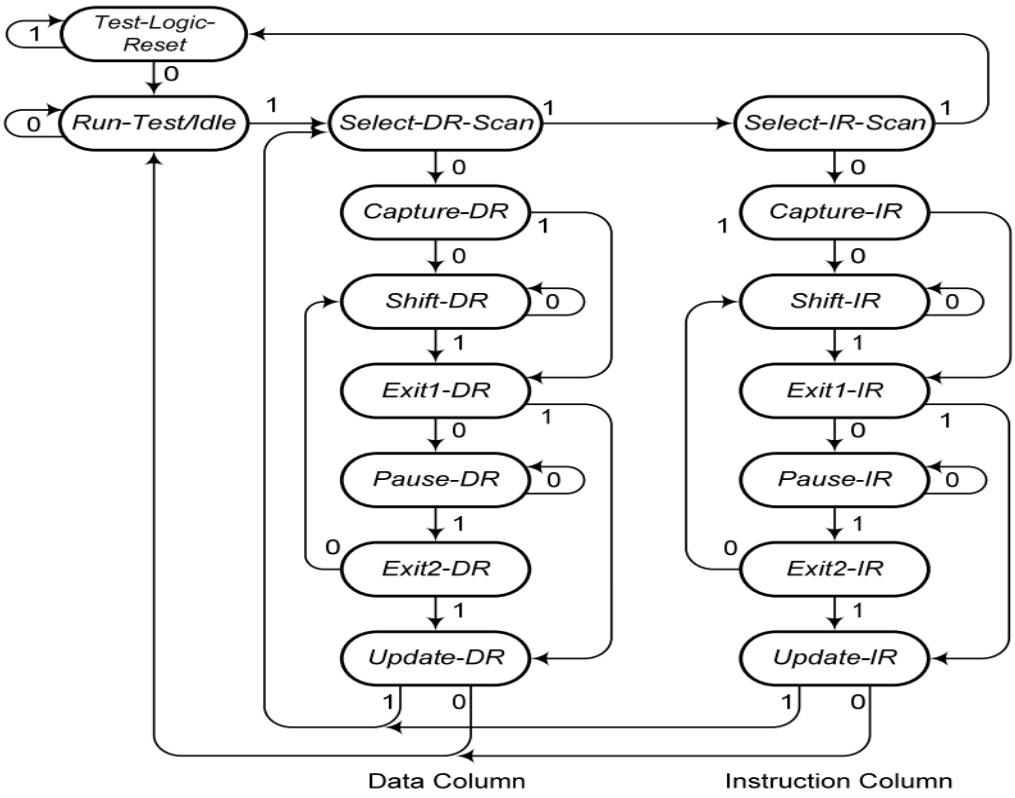

Figure 2-4 TAP Synchronous Finite State Machine (FSM)

2.1.4 Instruction Register

The instruction register is 10 bits in length.

The instruction register receives and decodes instructions, generating control signals to select the test data register to be placed on the scan chain from TDI to TDO. It also controls the loading source and driving destination of the test data register.

When the TAPC enters the instruction capture state (Capture-IR), the instruction register captures data {1'b0, busy, wakedown_over, wakeup_over, init_n, init, isc_enabled, isc_done, 2'b01} on the rising edge of TCK. "busy" indicates the embedded FLASH is in a busy state; "wakedown_over" indicates the completion of the wake-up shutdown process; "wakeup_over" indicates the completion of the wake-up process; "init_n" indicates the state of the INIT_FLAG_N signal; "init" indicates the completion of device initialization; "isc_enabled" indicates the enablement of JTAG ISC operation; and "isc_done" indicates the completion of JTAG ISC operation. The two lowest bits are fixed at 01 to detect the integrity of the scan chain and to locate faults.

When writing instructions to the instruction register, start with the lower bits.

When the TAPC enters the instruction update state (Update-IR), the currently executed instruction is updated on the falling edge of TCK.

2.1.5 Instruction Set

Table 2-2 JTAG Instruction Set

Instruction	Types	Operation Code	Description
BYPASS	1149.1 Non-test instruction	1111111111	Bypass instruction
SAMPLE/PRELOAD	1149.1 Non-test instruction	1010000000	Sample/preload instruction
EXTEST	1149.1 Test instruction	1010000001	External test instruction
INTEST	1149.1 Test instruction	1010000010	Internal test instruction
IDCODE	1149.1 Non-test instruction	1010000011	Identification instruction
HIGHZ	1149.1 Test instruction	1010000101	High-Z instruction
JRST	Only for design	1010001010	Reset instruction
CFGI	Only for design	1010001011	Configuration instruction
CFGO	Only for design	1010001100	Readback instruction
JWAKEUP	Only for design	1010001101	Wakeup instruction
READ_UID	Only for design	0101001100	Read-UID instruction
RDSR	Only for design	0101011001	Read-Status-Register instruction
WADR	Only for design	0101011010	Write-Address instruction
ERASE	Only for design	0101011101	Erase instruction
ERASE_PAGE	Only for design	0101011110	Page-Erase instruction
ERASE_CTL	Only for design	0101011111	Control-Erase instruction
PROGRAM	Only for design	0101100000	Program instruction
PROGRAM_CTL	Only for design	0101100001	Program-Control instruction
READ	Only for design	0101100010	Read instruction
READ_CTL	Only for design	0101100011	Control-Read instruction
PROGRAM_LOCK	Only for design	0101100100	Lock-Embedded-Flash instruction
READ_LOCK	Only for design	0101100101	READ-Embedded-Flash-Lock-Flag instruction
EFlash_SLEEP	Only for design	0101100110	Embedded-Flash-Sleep instruction
EFlash_WAKEUP	Only for design	0101100111	Embedded-Flash-Wakeup instruction

2.1.6 Test Data Register (TDR)

Table 2-3 List of Test Data Registers

Property	Item	Bit width
Data register	Device identification register	32
	Bypass register	1

Property	Item	Bit width
	Boundary scan register	2
	UID register	64
	Feature control register	32
	Program register	32
	Status register	32
	Address register	24
	LOCK register	12/13
Internal data register	READ register	-
	Configuration register	-

Table 2-4 Instruction and Test Data Register Mapping Table

Instruction	Data register
BYPASS	Bypass register
SAMPLE/PRELOAD	Boundary scan register
EXTEST	Boundary scan register
INTEST	Boundary scan register
IDCODE	Device identification register
USERCODE	Device identification register
HIGHZ	Bypass register
JRST	Bypass register
CFGI	Bypass register
CFGO	Configuration register
JWAKEUP	Bypass register
READ_UID	UID register
RDSR	Status register
WADR	Address register
ERASE	Bypass register
ERASE_PAGE	Bypass register
ERASE_CTL	Bypass register
PROGRAM	Program register
PROGRAM_CTL	Feature control register
READ	READ register
READ_CTL	Feature control register
PROGRAM_LOCK	LOCK register
READ_LOCK	LOCK register
EFlash_SLEEP	Bypass register
EFlash_WAKEUP	Bypass register

2.1.6.1 Device Identification Register

The device identification register is 32 bits in length.

The device identification register is capable of capturing and shifting data without parallel output.

When the current instruction is IDCODE and the TAPC state machine is in the data capture state, the device identification register captures the device flags in parallel on the rising edge of TCK.

2.1.6.2 Bypass Register

When certain chips need isolation, a bypass register can be used to achieve short-circuiting and shorten the length of the entire boundary scan chain.

The bypass register is 1 bit in length.

Bypass registers are capable of capturing and shifting data.

When the TAPC state machine is in the data capture state, the bypass register captures a logical 0 on the rising edge of TCK.

When the TAPC state machine is in the data shift state, the bypass register captures TDI on the rising edge of TCK.

2.1.6.3 Boundary Scan Register

The boundary scan register is a collection of boundary scan units. Boundary scan units are placed at the input port, output port, bidirectional port, and tri-state port of the device signal. Combining the boundary scan units forms a boundary scan register.

Pin connections and placement constraints determine the connection order of the boundary scan units.

The multi-function TCK, TMS, TDI, and TDO of PADs have no boundary scan register.

All other PADs have the same boundary scan register as the bi-directional DC PAD.

The boundary scan register of bi-directional DC PAD uses BC_2 control + BC_7 data

2.1.6.4 UID Register

The UID register is used for reading the UID.

The UID register is 64 bits in length.

The UID register is capable of capturing, shifting, and updating data.

The current instruction is READ_UID:

When the TAPC state machine is in the data capture state, the UID register captures the UID stored

in the embedded FLASH on the rising edge of TCK.

When the TAPC state machine is in the data shift state, the UID register captures the value of TDI on the rising edge of TCK.

2.1.6.5 Feature Control Register

The feature control register is used for programming and reading feature control bits from the embedded FLASH.

The feature control register is capable of capturing, shifting, and updating data.

The current instruction is PROGRAM_CTL:

When the TAPC state machine is in the data capture state, the feature control register captures the feature control bits stored in the embedded FLASH on the rising edge of TCK.

When the TAPC state machine is in the data shift state, the feature control register captures the value of TDI on the rising edge of TCK.

When the TAPC state machine is in the data update state, the data from the shift register path is latched to the parallel output of the feature control register on the falling edge of TCK.

The current instruction is READ_CTL:

When the TAPC state machine is in the data capture state, the feature control register captures the feature control bits stored in the embedded FLASH on the rising edge of TCK.

When the TAPC state machine is in the data shift state, the feature control register captures the value of TDI on the rising edge of TCK.

2.1.6.6 Program Register

The PROGRAM register is used for programming the embedded FLASH.

The PROGRAM register is 32 bits in length.

The PROGRAM register is capable of shifting and updating data.

The current instruction is PROGRAM:

When the TAPC state machine is in the data shift state, the PROGRAM register captures the value of TDI on the rising edge of TCK.

When the TAPC state machine is in the data update state, the data from the shift register path is latched to the parallel output of the PROGRAM register on the falling edge of TCK.

2.1.6.7 Status Register

The status register is used for reading the status register of the configuration control system.

The status register is 32 bits in length.

The status register is capable of capturing and shifting data, without parallel output.

The current instruction is RDSR:

When the TAPC state machine is in the data capture state, the status register captures the status register of the configuration control system on the rising edge of TCK.

When the TAPC state machine is in the data shift state, the status register shifts right by one bit on the rising edge of TCK.

2.1.6.8 Address Register

The address register is used for reading and writing the 24-bit starting address of the embedded FLASH.

The address register is 24 bits in length.

The address register is capable of capturing, shifting, and updating data.

The current instruction is WADR:

When the TAPC state machine is in the data capture state, the address register captures the current starting address of the embedded FLASH on the rising edge of TCK.

When the TAPC state machine is in the data shift state, the address register shifts right by one bit on the rising edge of TCK.

When the TAPC state machine is in the data update state, the data from the shift register path is latched to the parallel output of the address register on the falling edge of TCK.

For the meaning of each bit of the address register, see Address.

2.1.6.9 LOCK Register

The LOCK register is used for programming and reading the embedded FLASH lock flags and the end page address of the Master Self Configuration bitstream.

The 1K/2K/4K/7K LOCK register is 12 bits in length (lock flag: lock; heap address: lock_ba[1:0]; page address: lock_ra[8:0]).

The 10K LOCK register is 13 bits in length (lock flag: lock; heap address: lock_ba[1:0]; page address lock_ra[9:0]).

For 1K devices, lock_ba[1:0] is the reserved bit.

For 2K devices, lock_ba[1:0] is the reserved heap address and should be set to 2'b11.

The "lock" flag set to 1 indicates locked, with the locking range from address 0 to the end page address (inclusive). If EFlash has been locked, then it cannot be read and write, but can still be booted normally by loading bitstream from EFlash.

The LOCK register is capable of capturing, shifting, and updating data.

The current instruction is PROGRAM_LOCK:

When the TAPC state machine is in the data capture state, the LOCK register captures the embedded FLASH lock flag and the end page address of the Master Self Configuration bitstream on the rising edge of TCK.

When the TAPC state machine is in the data shift state, the LOCK register shifts right by one bit on the rising edge of TCK.

When the TAPC state machine is in the data update state, the data from the shift register path is latched to the parallel output of the LOCK register on the falling edge of TCK.

The current instruction is READ_LOCK:

When the TAPC state machine is in the data capture state, the LOCK register captures the embedded FLASH lock flag and the end page address of the Master Self Configuration bitstream on the rising edge of TCK.

When the TAPC state machine is in the data shift state, the LOCK register shifts right by one bit on the rising edge of TCK.

2.1.6.10 READ Register

The READ register is used to read the contents of embedded FLASH.

The READ register is an internal data register that is not placed on the TDI to TDO scan chain.

When the current instruction is READ and the TAPC state machine is in the data shift state, the PROGRAM register shifts left by one bit on the rising edge of TCK.

2.1.6.11 Configuration Register

The JTAG configuration register is used to read back the configuration register and CCS configuration register.

The configuration register is an internal data register that is not placed on the TDI to TDO scan chain.

The current instruction is CFGO. When the TAPC state machine is in the data shift state, the configuration register shifts left by one bit on the rising edge of TCK.

2.2 Feature Control Bits

The feature control bits of the Compa family CPLDs are stored in the embedded FLASH (Non-Volatile Memory) and are read by the chip CCS each time the chip is initialized. The CCS selects the function of the corresponding configurable multi-function PIN based on the feature control bits.

The CPLDs contains a feature control group CTL which is composed of 32-bit feature control bits. For details on the feature control bits, refer to the document "UG030004_Compa Family CPLDs Configuration User Guide".

The enablement of the JTAG interface is controlled by the feature control bits cfg_jtag_en_n (CTL [5]) and persist_jtag_n (CTL[14]). When cfg_jtag_en_n (CTL[5]) is 0, the JTAG interface is enabled in configuration mode; when persist_jtag_n (CTL[14]) is 0, the JTAG interface is enabled in user mode.

Additionally, if users choose to store the bitstream in the embedded FLASH and desire for the CPLDs to automatically load the bitstream into CRAM from the embedded FLASH after a reboot/reset, then users must enable the Master Self Configuration by setting the feature control bit cfg_msd_en (CTL[0]) to 1.

The chip feature control bits can be modified in the following ways:

1. When the PDS (Pango Design Suite) download tool (Fabric Configuration) downloads a sbit file, it writes the feature control bits contained in the sbit file into the chip.
2. Separately programming these features control bits of device in PDS download tool.
3. If the JTAG interface is enabled, use the JTAG interface instruction PROGRAM_CTL.
4. If the salve SPI interface is enabled, use the slave SPI interface instruction PROGRAM_CTL.
5. If the slave I2C interface is enabled, use the slave I2C interface instruction PROGRAM_CTL.
6. Use the internal slave APB interface instruction PROGRAM_CTL.

2.3 Configuration Flow

2.3.1 CRAM Configuration

Configuration/Reconfiguration

Configuration refers to the process of configuring PGC chip from the start until it enters user mode; Reconfiguration refers to the process of resetting (JTAG soft reset or externally triggered RSTN hard reset), clearing all CRAMs, exiting user mode, and then reconfiguring after the PGC chip enters user mode. The flow is as follows:

Table 2-5 Configuration/Reconfiguration Flowchart

Steps	TAP Controller State Transitions and Descriptions	TDI	TMS	TCK Cycles
1	Set TMS to 1 and provide 5 TCKs to ensure entry into the Test Logic Reset (TLR) state	X	1	5
2	Jump to the Run-Test/Idle (RTI) state	X	0	1
3	Jump to the SELECT-IR (SLIR) state	X	1	2
4	Enter the SHIFT-IR (SIR) state	X	0	2
5	Begin loading the JRST instruction into IR, starting with the lower bits	010001010	0	9
6	Load the highest bit of the JRST instruction and exit the SIR state	1	1	1
7	Jump to the Update-IR (UIR) state	X	1	1
8	Enter the Run-Test/Idle state and keep at least 1 TCK cycle	X	0	1
9	Jump to the SELECT-IR state	X	1	2
10	Enter the SIR state	X	0	2
11	Begin loading the CFGI instruction into IR, starting with the lower bits	010001011	0	9
12	Load the highest bit of the CFGI instruction and exit the SIR state	1	1	1
13	Jump to the Update-IR (UIR) state	X	1	1
14	Enter the Run-Test/Idle state and keep at least 75,000 TCK cycles (1)	X	0	75000
15	Jump to the SELECT-DR (SLDR) state	X	1	1
16	Enter the SHIFT-DR (SDR) state	X	0	2
17	Begin loading bitstream, starting with bit_1 or the first bit of the bitstream	$bit_{n-1}…bit_1$	0	n-1(2)
18	Load the last bit of the bitstream bit_n and exit the SDR state	bit_n	1	1
19	Enter the Test Logic Reset state	X	1	4
20	Jump to the Run-Test/Idle state	X	0	1
21	Jump to the SELECT-IR state	X	1	2
22	Enter the SHIFT-IR state	X	0	2
23	Begin loading the JWAKEUP instruction into IR, starting with the lower bits	010001101	0	9
24	Load the highest bit of the JWAKEUP instruction and exit the SIR state	1	1	1
25	Jump to the Update-IR state	X	1	1
26	Enter the Run-Test/Idle state and keep at least 1000 TCK cycles (3)	X	0	1000
27	Enter the Test Logic Reset state	X	1	3

Notes:

1. After the execution of the JRST instruction, the device reinitializes at a maximum of 1.5ms. Here, a delay is achieved by utilizing TCK. As the maximum frequency supported by JTAG is 50MHz, the delay shall be 75000 cycles. However, the number of cycles can be reduced proportionally based on the actual TCK toggle frequency.

2. "n" is the number of bits in the bitstream, which is a binary stream; here, the shift into the device starts from the beginning of the bitstream.

3. The device wake-up clock can be set to the configuration interface clock or the user clock. It defaults to the main SPI clock (i.e., Master Configuration Clock, generated by internal OSC), where a minimum continuing delay of 1000 TCK cycles is required. If

users choose another clock such as the JTAG clock or user clock as the wake-up clock, ensure that there is clock input on the corresponding interface and that JTAG delays at least 30 wake-up clock cycles in that state.

2.3.2 Embedded Flash Programming

The JTAG interface allows direct operations on the embedded FLASH.

The complete process for JTAG programming of embedded FLASH is as follows:

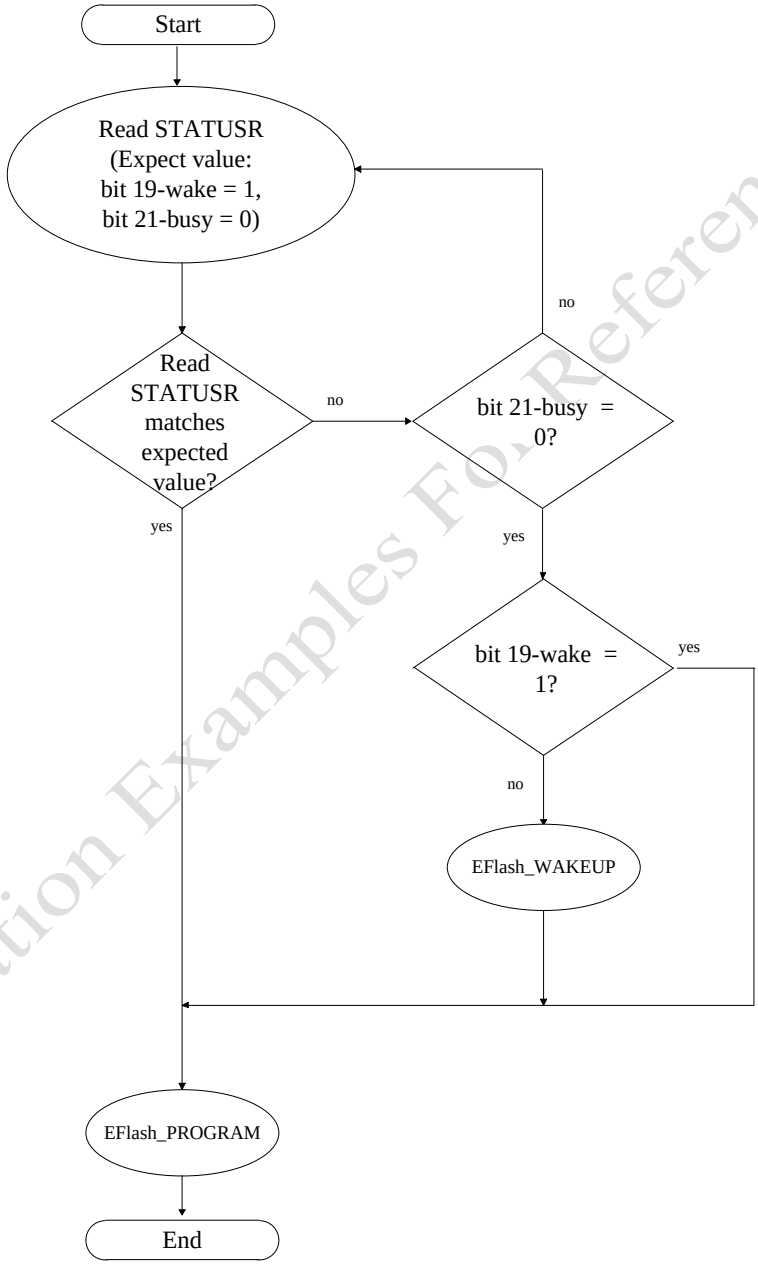

Figure 2-5 Programming Embedded Flash

These sub-processes include reading the status register (STATUSR), waking up EFlash, etc.., which are presented below in the same order as shown in Table 6. For all JTAG instruction flows, refer to Appendix 2 of the "***UG030004_Compa Family CPLDs Configuration User Guide***".

2.3.2.1 Reading Status Register (STATUSR)

Table 2-6 Status Register Reading Flowchart

Steps	TAP Controller State Transitions and Descriptions	TDI	TMS	TCK Cycles
1	Set TMS to 1 and provide 5 TCKs to ensure entry into the Test Logic Reset (TLR) state	X	1	5
2	Jump to the Run-Test/Idle (RTI) state	X	0	1
3	Jump to the SELECT-IR (SLIR) state	X	1	2
4	Enter the SHIFT-IR (SIR) state	X	0	2
5	Begin loading the RDSR instruction into IR, starting with the lower bits	010001010	0	9
6	Load the highest bit of the RDSR instruction while exiting the SIR state	1	1	1
7	Jump to the SELECT-DR (SLDR) state	X	1	2
8	Enter the SHIFT-DR (SDR) state	X	0	2
9	Begin reading the status register, starting with the lower bits	STATUSR[30:0]	0	31
10	Read the highest bit of the status register while exiting the SDR state	STATUSR[31]	1	1
11	Enter the Test Logic Reset (TLR) state	X	1	4

2.3.2.2 Waking Up Eflash

Table 2-7 Waking Up Eflash Flowchart

Steps	TAP Controller State Transitions and Descriptions	TDI	TMS	TCK Cycles
1	Set TMS to 1 and provide 5 TCKs to ensure entry into the Test Logic Reset (TLR) state	X	1	5
2	Jump to the Run-Test/Idle (RTI) state	X	0	1
3	Jump to the SELECT-IR (SLIR) state	X	1	2
4	Enter the SHIFT-IR (SIR) state	X	0	2
5	Begin loading the EFLASH_WAKEUP instruction into IR, starting with the lower bits	101100111	0	9
6	Load the highest bit of the EFLASH_WAKEUP instruction while exiting the SIR state	0	1	1
7	Jump to the Update-IR (UIR) state	X	1	1
8	Enter the Run-Test/Idle state for at least 27000 TCK cycles (1)	X	0	27001
9	Enter the Test Logic Reset (TLR) state	X	1	3

Notes: Here, a delay is achieved by utilizing TCK. As the maximum frequency supported by JTAG is 50MHz, the delay shall be 27000 cycles. However, the number of cycles can be reduced proportionally based on the actual TCK toggle frequency.

2.3.2.3 Program Eflash

Programming Eflash requires writing by page; if the data written is less than one full page, it must be filled with a default 0 to complete the page.

For each page, first use the WADR instruction to write to the address register in the page address

format, then use the PROGRAM instruction to write one page of data and wait for a certain period till the write is completed; if readback verification is needed, users can use the READ instruction to perform readback at this point, or after all programming is completed.

To finish programming Eflash, simply write the data to be written in a loop, one page at a time. PGC10KD differs slightly from other devices, the flow for other devices is as follows:

Table 2-8 Eflash Programming Flowchart

Steps	TAP Controller State Transitions and Descriptions	TDI	TMS	TCK Cycles
1	Set TMS to 1 and provide 5 TCKs to ensure entry into the Test Logic Reset (TLR) state	X	1	5
2	Jump to the Run-Test/Idle (RTI) state	X	0	1
3	Jump to the SELECT-IR (SLIR) state	X	1	2
4	Enter the SHIFT-IR (SIR) state	X	0	2
5	Begin loading the WADR instruction into IR, starting with the lower bits	101011010	0	9
6	Load the highest bit of the WADR instruction while exiting the SIR state	0	1	1
7	Jump to the SELECT-DR (SLDR) state	X	1	2
8	Enter the SHIFT-DR (SDR) state	X	0	2
9	Begin loading the Eflash page address, starting with the lower bits	addr[0:22]	0	23
10	Load the highest bit of the page address while exiting the SDR state	addr[23]	1	1
11	Jump to the Update-DR (UDR) state	X	1	2
12	Jump to the Run-Test/Idle (RTI) state	X	0	1
13	Jump to the SELECT-IR (SLIR) state	X	1	2
14	Enter the SHIFT-IR (SIR) state	X	0	2
15	Begin loading the PROGRAM instruction into IR, starting with the lower bits	101100000	0	9
16	Load the highest bit of the PROGRAM instruction while exiting the SIR state	0	1	1
17	Jump to the SELECT-DR (SLDR) state	X	1	2
18	Enter the SHIFT-DR (SDR) state	X	0	2
19	Begin shifting in one word of data, starting with the higher bits	data[31:1]	0	31
20	Shift in the lowest bit of that word of data while exiting the SDR state	data[0]	1	1
21	Repeat steps 17 to 20 until the page data is fully written, that is, 64 cycles	-	-	-
22	Jump to the Update-IR (UIR) state	X	1	1
23	Enter the Run-Test/Idle state for at least 470000 TCK cycles (1)	X	0	470001
24	Reap steps 3 to 23 until all data is written	-	-	-
25	Enter the Test Logic Reset (TLR) state	X	1	3

Notes: Here, a delay is achieved by utilizing TCK. As the maximum frequency supported by JTAG is 50MHz, the delay shall be 470,000 cycles. However, the number of cycles can be reduced proportionally based on the actual TCK toggle frequency.

The flow for PGC10KD is as follows:

Table 2-9 PGC10KD Eflash Programming Flowchart

Steps	TAP Controller State Transitions and Descriptions	TDI	TMS	TCK Cycles
1	Set TMS to 1 and provide 5 TCKs to ensure entry into the Test Logic Reset (TLR) state	X	1	5
2	Jump to the Run-Test/Idle (RTI) state	X	0	1
3	Jump to the SELECT-IR (SLIR) state	X	1	2
4	Enter the SHIFT-IR (SIR) state	X	0	2
5	Begin loading the WADR instruction into IR, starting with the lower bits	10101101 0	0	9
6	Load the highest bit of the WADR instruction while exiting the SIR state	0	1	1
7	Jump to the SELECT-DR (SLDR) state	X	1	2
8	Enter the SHIFT-DR (SDR) state	X	0	2
9	Begin loading the Eflash page address, starting with the lower bits	addr[0:22]	0	23
10	Load the highest bit of the page address while exiting the SDR state	addr[23]	1	1
11	Jump to the Update-DR (UDR) state	X	1	2
12	Jump to the Run-Test/Idle (RTI) state	X	0	1
13	Jump to the SELECT-IR (SLIR) state	X	1	2
14	Enter the SHIFT-IR (SIR) state	X	0	2
15	Begin loading the PROGRAM instruction into IR, starting with the lower bits	10110000 0	0	9
16	Load the highest bit of the PROGRAM instruction while exiting the SIR state	0	1	1
17	Jump to the Update-IR (UIR) state	X	1	1
18	Enter the Run-Test/Idle state for at least 600,000 TCK cycles (1)	X	0	600001
19	Jump to the SELECT-DR (SLDR) state	X	1	1
20	Enter the SHIFT-DR (SDR) state	X	0	2
21	Begin shifting in one word of data, starting with the higher bits	data[31:1]	0	31
22	Shift in the lowest bit of that word of data while exiting the SDR state	data[0]	1	1
23	Repeat steps 17 to 20 until the page data is fully written, that is, 64 cycles	-	-	-
24	Jump to the Update-IR (UIR) state	X	1	1
25	Enter the Run-Test/Idle state for at least 600,000 TCK cycles (1)	X	0	600001
26	Reap steps 3 to 23 until all data is written	-	-	-
27	Enter the Test Logic Reset (TLR) state	X	1	3

Notes: Here, a delay is achieved by utilizing TCK. As the maximum frequency supported by JTAG is 50MHz, the delay shall be 600,000 cycles. However, the number of cycles can be reduced proportionally based on the actual TCK toggle frequency.

2.3.2.4 Address

The minimum addressing unit of PGC EFlash is a 32-bit word, with each page containing 256Bytes or 64 words. Different devices have different memory sizes, thus the total number of pages varies. For larger devices, EFlash is divided into multiple heaps, hence the Address is non-continuous. The memory size and address mapping of devices are shown below.

Table 2-10 Ordinary Memory Size of Embedded Flash

Device	Number of Piles	Number of Pages Per Pile	Page Size (Bytes)	Total Capacity (Bytes)	Total Pages of Embedded Flash
PGC1KG/L	1	332	256	84992	332
PGC2KG/L	1	332	256	84992	332
PGC4KD/L	4	320	256	327680	1280
PGC7KD	4	452	256	462848	1808
PGC10KD	4	640	256	655360	2560

Table 2-11 Meaning of 1K/2K/4K/7K Addresses

Address	Item	Description
[23:19]	Reserved	Must be set to 5'b00000
[18:17]	Ba[1:0]	Heap Address For 1K and 2K devices, it is reserved bit, while for the 2K devices, it must be set to 2'b11
[16:8]	Ra[8:0]	Page Address
[7:6]	Reserved	Must be set to 2'b00
[5:0]	Ca[5:0]	32-bit data page internal offset address The PROGRAM instruction should be set to 6'b000000

Table 2-12 Meaning of 10K Address

Address	Item	Description
[23:20]	Reserved	Must be set to 4'b0000
[19:18]	Ba[1:0]	Heap Address
[17:8]	Ra[9:0]	Page Address
[7:6]	Reserved	Must be set to 2'b00
[5:0]	Ca[5:0]	32-bit data page internal offset address The PROGRAM instruction should be set to 6'b000000

Chapter 3 Configuration Examples

This section introduces the application of PGC JTAG interface configuration, using the example of controlling the embedded FLASH that reads/writes PGC and loading CRAM via STM32F103.

This example is based on the Wildfire STM32F103MINI hardware platform, with the software development environment being Keil5.

The supported functions and devices in the example are shown in the following table:

Table 3-1 List of Supported Functions

S. No.	Supported Functions
1	Read device information, including IDCode, UID, status register, and feature control bits
2	Program UID
3	Configure CRAM
4	Program EFlash

Table 3-2 List of Supported Devices

S. No.	Supported Devices
1	PGC1KG
2	PGC1KL
3	PGC2KG
4	PGC2KL
5	PGC4KD
6	PGC4KL
7	PGC7KD
8	PGC10KD

3.1 Hardware Connection

The hardware connection consists of two parts: the JTAG connection between STM32 and CPLDs using Dupont lines, and the hardware connection between STM32 and the FLASH chip via a PCB circuit (Wildfire STM32MINI on-board STM32F103 and W25Q64 chips).

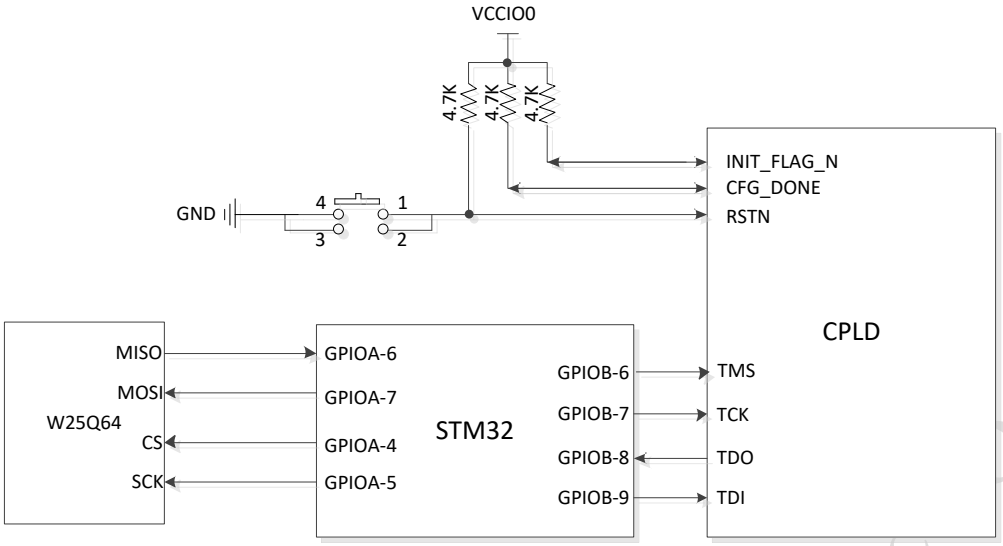

Figure 3-1 Example Hardware Circuit Diagram

During the configuration process, it is necessary to maintain the connection between the microcontroller's GPIO Pin for the simulated JTAG interface and the CPLDs' JTAG interface.

Table 3-3 Connections between JTAG Pins

STM32 Pins	**CPLDs Pins**
GPIOB-6	TMS
GPIOB-7	TCK
GPIOB-8	TDO
GPIOB-9	TDI

In the example program, the bitstream of CPLDs is stored on the FLASH chip of the STM32 circuit board and the FLASH chip uses a standard SPI interface. The pin connection between the microcontroller and the FLASH chip is shown in the following table.

Table 3-4 FLASH Chip JTAG Pins

FLASH Chip Pins	**STM32 Pins**
CS	GPIOA-4
SCK	GPIOA-5
MISO	GPIOA-6
MOSI	GPIOA-7

When downloading the demonstration bitstream, it is necessary to connect the FLASH pins from Table 5 to the outside, writing the CPLDs bitstream starting from Address 0. Note that the reset button on the board must be kept pressed during the FLASH programming. Users can use Pango Fabric Configuration to download FLASH, simply by connecting these corresponding FLASH pins to the Pango USB Cable.

3.2 Bitstream Setting

3.2.1 Setting Feature Control Bits

As described in Section 2.2, the feature control bits determine whether the JTAG is enabled. To use the JTAG, ensure that the JTAG interface is enabled in the feature control bits settings. The chip is shipped with JTAG enabled by default. If the JTAG interface is disabled, users can restore it by keeping the JTAGEN pin pulled high.

When using the download tool Fabric Configuration to download the bitstream to CRAM or EFlash, the feature control bits are written by default. In the bitstream settings, set the feature control bits to maintain the JTAG interface as the configuration interface (select Dedicated IO), as shown in the figure below.

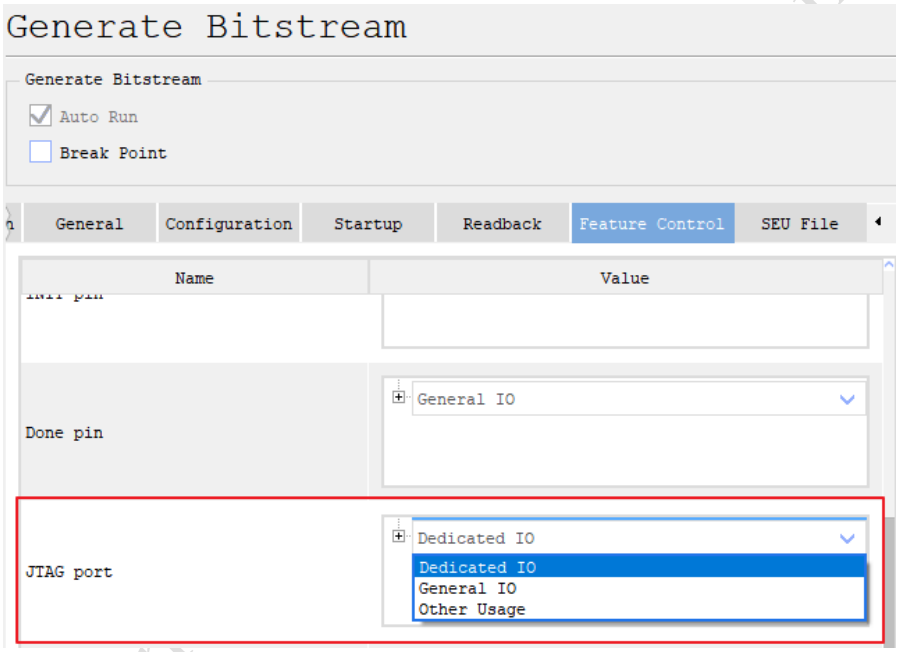

Figure 3-2 Setting Feature Control Bits in Bitstream Settings

3.3 Project Notes

Table 3-5 Project Overviews

Project Name	
CFG_DEMO_JTAG	
Application Range	
Supported Devices	The routine is run by STM32F103 and can be ported to other microcontroller platforms with minor modifications.
Language Used	C
Provided Design Files	
C Language Source Program	.c and .h files
keil5 Project File	RAR compressed file

Development Tools Provided	
Design Tools	Keil5 uVision5 version

The project directory can be used for the following purposes:

Table 3-6 Project Directory Classification

Doc	Documents
Project	Location where Keil5 project files are placed
Libraries	Platform-related codes, including the library files and module drivers such as spi, uart, etc. required by the example platform
Source	Program key source codes It includes the following parts: pgc_def.h: Definitions of parameters related to the PGC family devices pango_jtag.c/h: Implementation of the JTAG interface protocol pgc_jtag_basic.c/h: Implementation of basic operations for the PGC family JTAG interface pgc_jtag_demo.c: Implementation of practical example functions for the PGC family JTAG interface hardware.c: Implementation of platform-related function functions

3.4 Porting

This example project can be ported to other platforms with simple modifications. Porting mainly refers to modifying the "Source" or the core code in the project directory for use on other platforms. The following explains the main concerns during the porting process.

During the porting process, first add the platform-independent files (pgc_def.h, pango_jtag.c/h, pgc_jtag_basic.c/h) from the "Source" directory to the target project; then call the functions by referring to pgc_jtag_demo.c file, or just add some of the functions to the target project; finally, modify the hardware.c file with the following main points in mind:

3.4.1 Frequency Setting

The frequency of CPUs across various platforms differs significantly, affecting some functions of the example code, mainly the IO toggling rate of the JTAG interface, the delay in certain processes (achieved by using TCK toggling), and the implementation of delay functions.

Considering the impact on the JTAG interface rate, the PGC family JTAG interface supports a rate up to 50MHz. It is necessary to ensure that the interface rate does not exceed 50MHz, which can be determined by observing the TCK toggling. If the interface rate exceeds 50MHz, a delay must be added to the IO control interface. For example, the IO control function:

```
void pango_JtagSetTMSHigh()
{
```

```
    GPIO_SetBits(pango_JTAG_TMS_PORT, pango_JTAG_TMS_PIN);
}
```

The toggling rate can be reduced by setting the IO to the same value in a loop:

```
void pango_JtagSetTMSHigh()
{
        unsigned int IdleTime = 5; // Loop 5 times for delay
        unsigned int i = 0;
        for(i = 0; i < IdleTime; i++)
                GPIO_SetBits(pango_JTAG_TMS_PORT, pango_JTAG_TMS_PIN);
}
```

Considering the delay implemented by using TCK toggling in the process, the TCK toggling frequency should not exceed 50MHz after the adjustment in the last step. In the example project, for the delay implemented by using TCK toggling in some parts of the process, the number of TCK cycles is adjusted based on the global variable g_user_clk. This parameter can simply be modified based on the actual TCK toggling frequency.

Considering the implementation of the delay function, the delay function can be modified based on the actual situation. In this project, only the function that checks the status uses delay as an interval. If the delay is too short, the function will frequently check the status, without affecting the actual functionality.

3.4.2 GPIO Definition

To control IO toggling across different platforms, it generally requires defining the IO location and initializing the IO, as shown in the hardware.c example:

```
/****************************************** JTAG connected pin define *****************
#define           pango_JTAG_TMS_APBxClock_FUN          RCC_APB2PeriphClockCmd
#define           pango_JTAG_TMS_CLK                   RCC_APB2Periph_GPIOB
#define           pango_JTAG_TMS_PORT                  GPIOB
#define           pango_JTAG_TMS_PIN                   GPIO_Pin_6

#define           pango_JTAG_TCK_APBxClock_FUN          RCC_APB2PeriphClockCmd
#define           pango_JTAG_TCK_CLK                   RCC_APB2Periph_GPIOB
#define           pango_JTAG_TCK_PORT                  GPIOB
#define           pango_JTAG_TCK_PIN                   GPIO_Pin_7

#define           pango_JTAG_TDO_APBxClock_FUN          RCC_APB2PeriphClockCmd
#define           pango_JTAG_TDO_CLK                   RCC_APB2Periph_GPIOB
#define           pango_JTAG_TDO_PORT                  GPIOB
#define           pango_JTAG_TDO_PIN                   GPIO_Pin_8

#define           pango_JTAG_TDI_APBxClock_FUN          RCC_APB2PeriphClockCmd
#define           pango_JTAG_TDI_CLK                   RCC_APB2Periph_GPIOB
#define           pango_JTAG_TDI_PORT                  GPIOB
#define           pango_JTAG_TDI_PIN                   GPIO_Pin_9
```

```
void pango_JtagIOInit(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    /*!< Configure JTAG pins: TMS */
    pango_JTAG_TMS_APBxClock_FUN(pango_JTAG_TMS_CLK, ENABLE);
    GPIO_InitStructure.GPIO_Pin = pango_JTAG_TMS_PIN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(pango_JTAG_TMS_PORT, &GPIO_InitStructure);

    /*!< Configure JTAG pins: TCK */
    pango_JTAG_TCK_APBxClock_FUN(pango_JTAG_TCK_CLK, ENABLE);
    GPIO_InitStructure.GPIO_Pin = pango_JTAG_TCK_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(pango_JTAG_TCK_PORT, &GPIO_InitStructure);

    /*!< Configure JTAG pins: TDO */
    pango_JTAG_TDO_APBxClock_FUN(pango_JTAG_TDO_CLK, ENABLE);
    GPIO_InitStructure.GPIO_Pin = pango_JTAG_TDO_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(pango_JTAG_TDO_PORT, &GPIO_InitStructure);

    /*!< Configure JTAG pins: TDI */
    pango_JTAG_TDI_APBxClock_FUN(pango_JTAG_TDI_CLK, ENABLE);
    GPIO_InitStructure.GPIO_Pin = pango_JTAG_TDI_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(pango_JTAG_TDI_PORT, &GPIO_InitStructure);

    /* Init the Jtag: TCK, TDO low ,TMS high */
    pango_JtagSetTCKLow();
    pango_JtagSetTDILow();
    pango_JtagSetTMSHigh();
}
```

Figure 3-3 Example Code

Users can modify this part based on the specific platform and then execute the read/ write interface functions (pango_JtagSetTMSHigh, pango_JtagReadTDO, etc.).

3.4.3 Bitstream Storing

The example project stores the bitstream in an external SPI Flash and utilizes the driver implemented in the bsp_spi.c file to read the bitstream. During the porting process, just modify the function pango_BufferRead based on the bitstream acquisition method supported by the actual

platform. As shown in the definition, the function needs to support two key inputs: ReadAddr, which specifies the read address; and NumByteToRead, which is the number of bytes to read at one time.

3.5 Example Demo

The Demo program is a routine of data interaction based on serial port control, where users control the embedded system by inputting instructions through the serial port to configure the CPLDs. The use of it requires the installation of a serial port debugging assistant, which is configured as shown in the table below.

Table 3-7 Parameter Configuration of Serial Port

Configuration Item	**Parameter**
Baud Rate	115200
Checksum Bits	None
Data Bits	8
Stop Bits	1
HEX Display	Off
HEX Send	Off

After the STM32 is reset, the serial port debugging assistant appears as shown in the figure. Enter the number for the corresponding function in the sending box and then send, then the microcontroller will execute the corresponding operation.

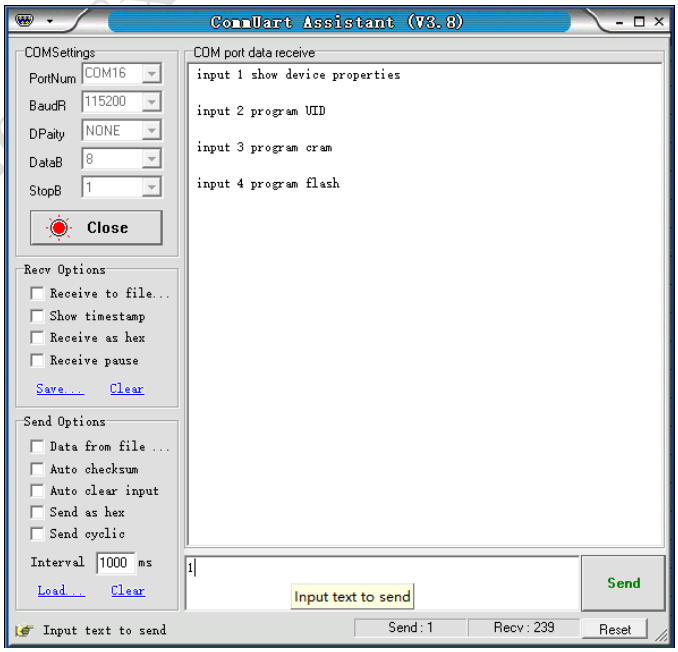

Figure 3-4 Serial Port Debugging

Disclaimer

Copyright Notice

Disclaimer

1. This document only provides information in stages and may be updated at any time based on the actual situation of the products without further notice. The Company assumes no legal responsibility for any direct or indirect losses caused by improper use of this document.
2. This document is provided "as is" without any warranties, including but not limited to warranties of merchantability, fitness for a particular purpose, non-infringement, or any other warranties mentioned in proposals, specifications, or samples. This document does not grant any explicit or implied intellectual property usage licence, whether by estoppel or otherwise.
3. The Company reserves the right to modify any documents related to its family products at any time without prior notice.
4. The information contained in this document is intended to assist users in resolving application-related issues. While we strive for accuracy, we cannot guarantee that the document is entirely free from flaws. Should any functional abnormalities and performance degradation arise due to deviation from the prescribed procedures outlined herein, our company will neither be held liable nor concede that such issues stem from product deficiencies. The solutions presented in this document are just one of the feasible options and cannot cover all application scenarios. Consequently, if users encounter functional abnormalities or performance degradation despite adhering to the prescribed procedures outlined herein, we cannot assure that such issues are indicative of product deficiencies.