Compa Family CPLDs Device Loaded Platform Software Application Guide

(AN03008, V1.2.3)

(15.09.2021)

Revisions History

Document Revisions

Version	Date of Release	Revisions
V1.2.3	15.09.2021	Initial release.

About this Manual

Terms and Abbreviations

Terms and Abbreviations	Meaning
CPLD	Complex Programmable Logic Device
JTAG	Joint Test Action Group
pef	Pango Embedded File

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 Overview

1.1 Introduction

This document serves as the application manual for the loading solution of the Compa family CPLD device platform, provided by Shenzhen Pango Microsystems Co., Ltd. It focuses on the parsing of loaded header files and the parsing descriptions of function codes for the Compa family CPLD device platform.

1.2 Design Information

Table 1-1 Information on Loaded Products

Information on Loaded Products	
Supported Devices	Compa family CPLD
Supported User Interface	JTAG
Provided Design Files	
Remote Upgrade Reference Designs	C file source codes
CPU Reference Designs	1. Linux development project 2. Linux kernel version 4.1.15
Development Tools	
Design Tools	1. PDS2020.3-SP2.2 2. Ubuntu16.04x64

Chapter 2 Function Description

The provided source code has the following functions: identify the chip ID and link number on the board; parse and read the loaded file; and download the bit stream to the corresponding device on the board by using GPIO to simulate JTAG, etc.

The Compa family CPLD device loading platform block diagram is shown in Figure 2-1.

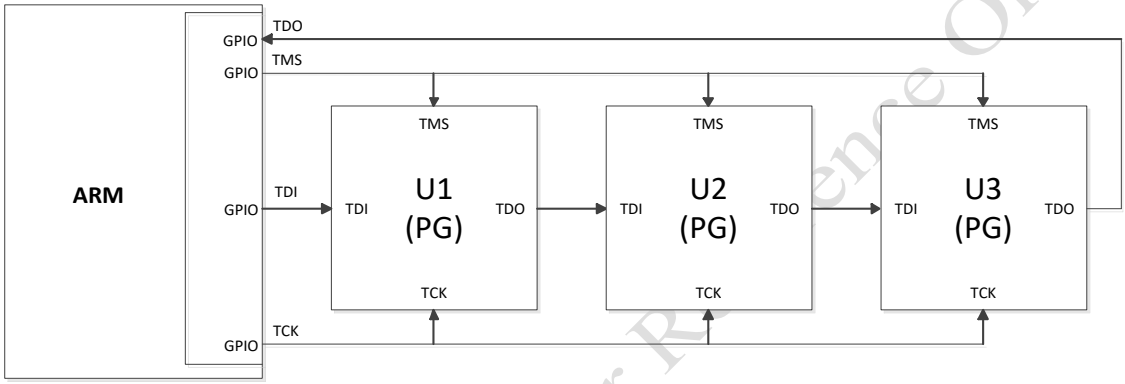

Figure 2-1 Compa Family CPLD Device Loading Platform System Block Diagram

2.1 JTAG Driver

The block diagram of the JTAG ports on the Compa family CPLD device is shown in Figure 2-2:

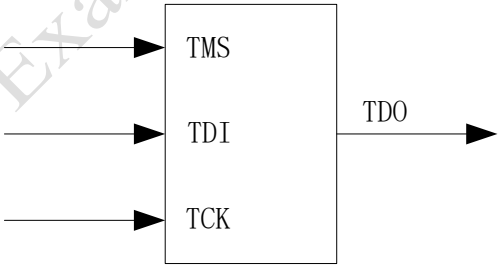

Figure 2-2 JTAG Port Block Diagram

2.1.1 JTAG Hardware Port

Table 2-1 JTAG Hardware Port

Item	Direction	Description
TCK	Input	Test Clock: TCK provides the clock for chip test logic and is a dedicated input pin. It can operate independently of the chip system clock or be synchronized with the operating mode clock
TMS	Input	Test Mode Select: Used to control the status switching of the test access port controller state machine on the rising edge of TCK to move test instructions and test data. When TMS is not driven, it defaults to logic 1.

Item	Direction	Description
TDI	Input	Test Data In: 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. When TDI is not driven, it defaults to logic 1.
TDO	Ootput	Test Data Out: 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. In other states, no data is moved out, and TDO is in a high impedance state.

Please refer to the relevant information regarding the JTAG protocol, as the length of the document does not allow for details.

2.1.2 JTAG Interface Timing

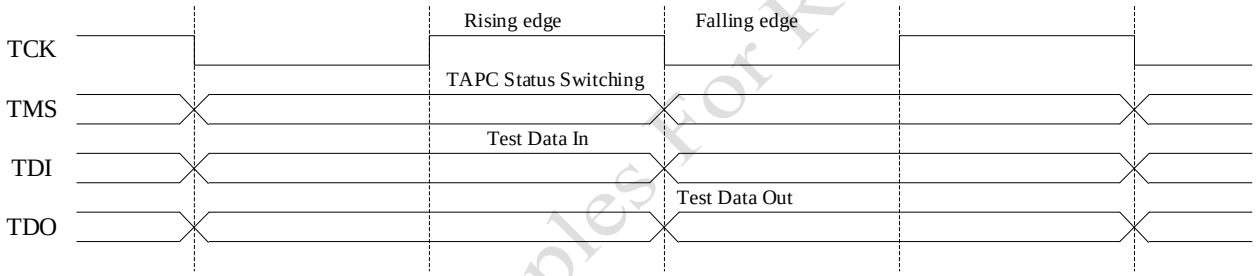

Figure 2-3 JTAG Timing

2.1.3 JTAG Driver Porting

The driver file is used to initialize the GPIO and control the GPIO level switching. Before executing the application, users need to add the driver to the Linux kernel.

Before adding the driver file, users need to determine the GPIO port number of JTAG, as shown in Figure 2-4 below:

```
#define Jtag_TMS_Gpio_Num   31   /*  GPIO pin number corresponding to the jtag MS signal   */
#define Jtag_TCK_Gpio_Num   30   /*  GPIO pin number corresponding to the jtag TCK signal  */
#define Jtag_TDO_Gpio_Num   27   /*  GPIO pin number corresponding to the jtag TDO         */
#define Jtag_TDI_Gpio_Num   3    /*  GPIO pin number corresponding to the jtag TDr signal  */
```

Figure 2-4 JTAG GPIO Port Number

To ensure that the GPIOs used by the JTAG driver are not being used by other drivers, users need to modify the GPIO number based on your specific project requirements.

Then start adding the driver file to the Linux kernel source file (Linux kernel version 4.1.15).

Step 1: Copy the "../driver/jtag_gpio_ctrl_driver.c" file to the GPIO directory of the Linux kernel source file ("../drivers/gpio/").

Step 2: Modify the Kconfig file in the GPIO directory ("../drivers/gpio/"), as shown in Figure 2-5 below:

```
978
979 config GPIO_VIPERBOARD
980         tristate "Viperboard GPIO a & b support"
981         depends on MFD_VIPERBOARD && USB
982         help
983           Say yes here to access the GPIO signals of Nano River
984           Technologies Viperboard. There are two GPIO chips on the
985           board: gpioa and gpiob.
986           See viperboard API specification and Nano
987           River Tech's viperboard.h for detailed meaning
988           of the module parameters.
989
990 endmenu
991
992 config GPIO_JTAG
993         tristate "Jtag Gpio Driver"
994         help
995           Jtag Gpio Port Inital
996
```

Figure 2-5 Kconfig Configuration

Add the following:

config GPIO_JTAG

tristate "Jtag Gpio Driver"

help

Jtag Gpio Port Initial

Step 3: Modify the Makefile file in the GPIO directory ("../drivers/gpio/"), as shown in Figure 2-6 below:

```
112 obj-$(CONFIG_GPIO_XTENSA)        += gpio-xtensa.o
113 obj-$(CONFIG_GPIO_ZEVIO)         += gpio-zevio.o
114 obj-$(CONFIG_GPIO_ZYNQ)          += gpio-zynq.o
115 obj-$(CONFIG_GPIO_JTAG)          += jtag_gpio_ctrl_driver.o
116
```

Figure 2-6 Makefile Configurations

Add the following:

obj-$(CONFIG_GPIO_JTAG) += jtag_gpio_ctrl_driver.o

Step 4: Use the "make ARCH=ARM menuconfig" command to access the Linux kernel menu configuration interface to enable the driver, as shown in Figure 2-7 below:

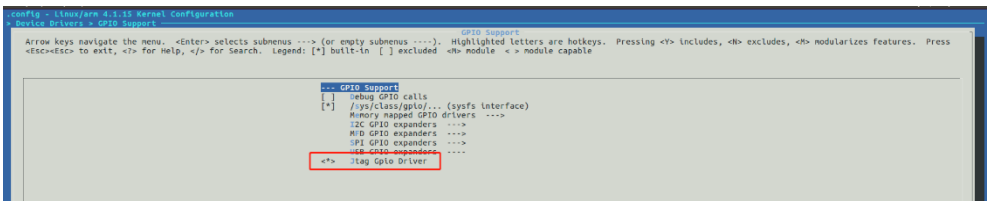

Figure 2-7 Enabling jtag gpio Driver

Step 5: Confirm that the driver is enabled and check the Linux kernel configuration file:.config file, as shown in Figure 2-8 below:

Figure 2-8 Opening .config File

The presence of the content in the red box as shown indicates that the driver is enabled.

Step 6: Compile the Linux kernel file to generate a new zImage file and download it to the development board.

Step 7: Boot up the Linux target board and inspect the mounted driver files, as shown in Figure 2-9 below:

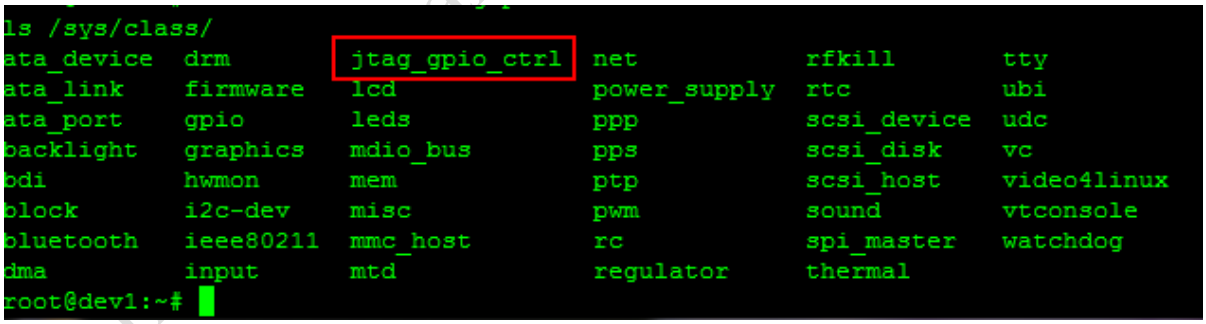

Figure 2-9 Mounted Driver File Directory

A folder with the name highlighted in the red box in the above figure appears in the "/sys/class" directory, which indicates a successful addition of the JTAG driver.

2.2 Function Design

2.2.1 Function Design Block Diagram

The design primarily utilizes a GPIO to simulate JTAG loading, where a compressed pef file is parsed and loaded. The function flowchart is shown in Figure 2-10 below:

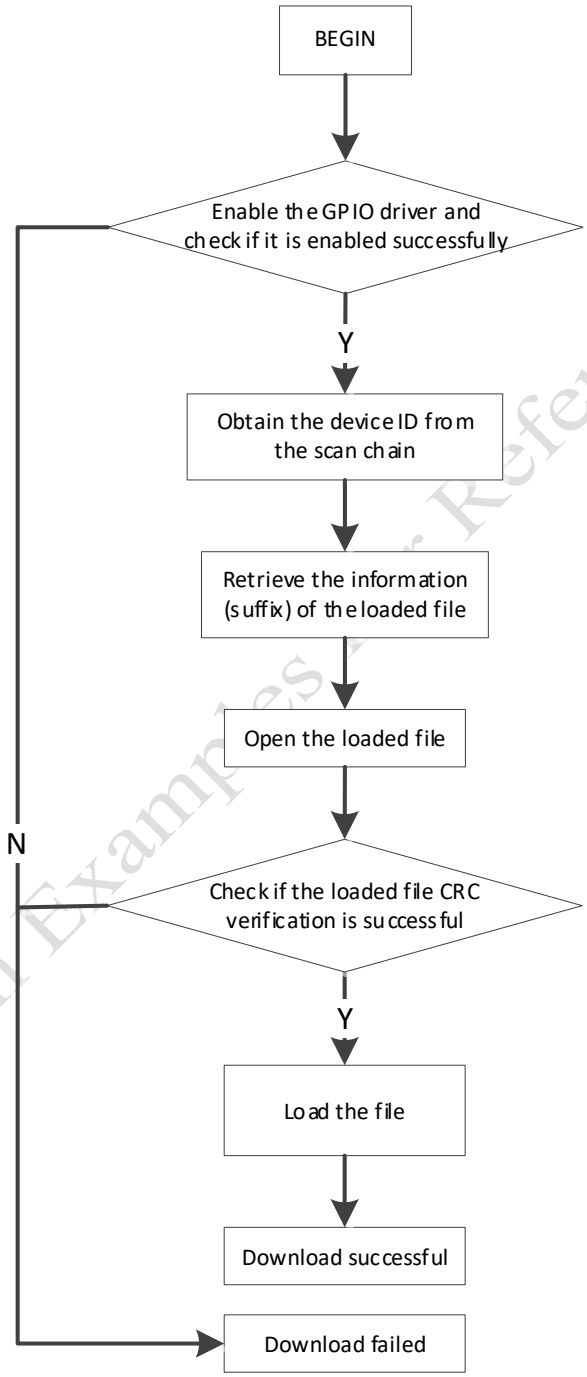

Figure 2-10 Platform Function Loading Flowchart

Step 1: Enable the GPIO driver.

Step 2: Obtain the device ID from the scan chain.

Step 3: Retrieve the information (suffix) of the loaded file.

Step 4: Open the loaded file.

Step 5: Check the CRC of the loaded file.

Step 6: Download the loaded file

Chapter 3 Reference Designs

This section primarily explains the directory structure of the reference design files and the main function flow of platform loading for multiple devices and models. It also provides detailed explanations of the functions called within the main function, including their functionalities and usage details.

3.1 Reference Design File Directory

The reference design file directory is shown in Figure 3-1 below:

```
pef_parser Reference design Directory Structure:

├─doc                          //Reference design document
├─app                          //Application folder
|  ├─includes                  //*.h header file directory
|  |  ├─baseDefine.h           //Instruction encoding file
|  |  ├─gpio_bsp.h             //Header file defining GPIO function calls
|  |  ├─jtag_chain.h           //Header file defining the scan chain
|  |  ├─pefParser.h            //Header file defining the parsing of pef files
|  |  ├─pgutil.h               //Header file defining error messages, decompression, and CRC
|  |  ├─ports.h                //Header file defining GPIO port numbers
|  ├─sources                   //Source code file directory
|  |  ├─gpio_bsp.c             //Definitions for GPIO open function, close function, and other functions
|  |  ├─jtag_chain.c           //jtag scan chain
|  |  ├─main.c                 //Main function
|  |  ├─pefParser.c            //pef file parsing function
|  |  ├─pgutil.c               //Decompression and CRC function
|  |  ├─ports.c                //Port level conversion function
|  ├─Makefile                  //Makefile compiling files
├─driver                       //JTAG driver directory
|  ├─jtag_gpio_ctrl_driver.c   //Driver source files
```

Figure 3-1 Reference Design File Directory

The reference design files include application documents, code source files, and driver files. For details, please refer to the specific file source codes.

3.2 Reference Design Board Validation

3.2.1 pef File Generation

Step 1: Use Cable to scan for the device, then place the mouse over the blue device and right-click, and then a popup will appear as shown below:

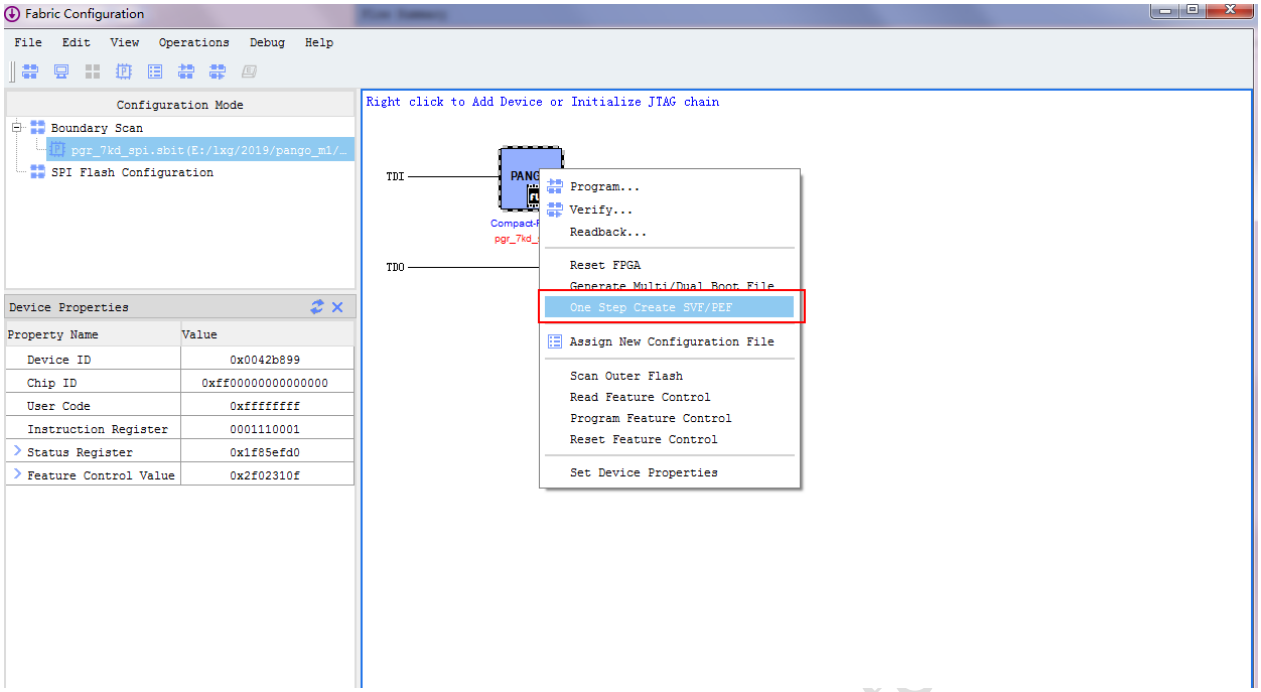

Figure 3-2 Generating a pef File

Step 2: Upon completion of Step 1, the following interface appears:

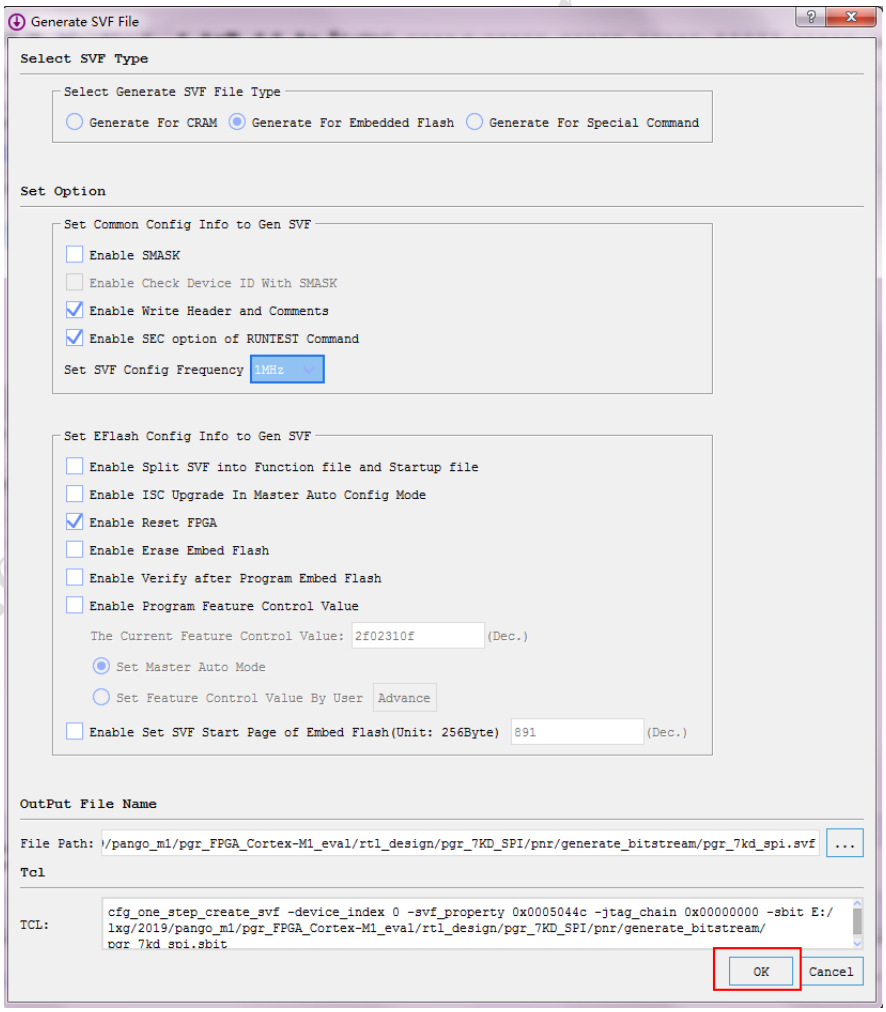

Figure 3-3 Configuring svf File Interface

If no other configuration is needed, simply click OK to generate svf and pef files in the "../pnr/generate_bitstream" directory.

3.2.2 Application Code Porting

Step 1: Replace JTAG port functions JtagSetTMSHigh, JtagSetTMSLow, JtagSetTCKHigh, JtagSetTCKLow, JtagSetTDIHigh, JtagSetTDILow, and JtagReadTDO with level control functions for JTAG GPIO simulation ports.

```
void JtagSetTMSHigh()
{
    jtag_file_info.TMSFileinfo._info.value = 1;
    ioctl(jtag_file_info.TMSFileinfo.fd, GPIO_SET_DATA, &jtag_file_info.TMSFileinfo._info);
}

void JtagSetTMSLow()
{
    jtag_file_info.TMSFileinfo._info.value = 0;
    ioctl(jtag_file_info.TMSFileinfo.fd, GPIO_SET_DATA, &jtag_file_info.TMSFileinfo._info);
}

void JtagSetTCKHigh()
{
    jtag_file_info.TCKFileinfo._info.value = 1;
    ioctl(jtag_file_info.TCKFileinfo.fd, GPIO_SET_DATA, &jtag_file_info.TCKFileinfo._info);
}

void JtagSetTCKLow()
{
    jtag_file_info.TCKFileinfo._info.value = 0;
    ioctl(jtag_file_info.TCKFileinfo.fd, GPIO_SET_DATA, &jtag_file_info.TCKFileinfo._info);
}

void JtagSetTDIHigh()
{
    jtag_file_info.TDIFileinfo._info.value = 1;
    ioctl(jtag_file_info.TDIFileinfo.fd, GPIO_SET_DATA, &jtag_file_info.TDIFileinfo._info);
}

void JtagSetTDILow()
{
    jtag_file_info.TDIFileinfo._info.value = 0;
    ioctl(jtag_file_info.TDIFileinfo.fd, GPIO_SET_DATA, &jtag_file_info.TDIFileinfo._info);
}

unsigned char JtagReadTDO()
{
    ioctl(jtag_file_info.TDOFileinfo.fd, GPIO_GET_DATA, &jtag_file_info.TDOFileinfo._info);
    return jtag_file_info.TDOFileinfo._info.value;
}
```

Figure 3-4 Functions for JTAG Port Replacement

Step 2: Cache the contents of loaded files

```
int OpenFile(char* file_name)
{
    //step1: open pef file to read
    _byte_nums = 0;
    if ((_pg_pef_file = fopen(file_name, "rb")) == NULL)
    {
        return FILE_OPEN_ERROR;
    }
    fseek(_pg_pef_file, 0, SEEK_END);
    _file_total_Len = ftell(_pg_pef_file);
    _one_percentnum = _file_total_Len/100;
    fseek(_pg_pef_file, 0, SEEK_SET);
    file_data_buffer=(char *)malloc(_file_total_Len + 1);

    if (!file_data_buffer)
    {
        fprintf(stderr, "Memory error!\n");
        fclose(_pg_pef_file);
        return -1;
    }

    fread(file_data_buffer, 1, _file_total_Len+1, _pg_pef_file);
    fclose(_pg_pef_file);
    return 1;
}
```

Figure 3-5 Caching the Contents of Loaded Files

If loaded by file transfer, this step can be skipped.

Note: If the user doesn't load the interface function by loading a file, they can directly cache the loaded file by assigning the data to file_data_buffer:

1. Cache the content of the pef file into the global "buffer: file_data_buffer" (request caching before using)
2. Obtain the total number of bytes in the loaded file and assign it to the global variable _file_total_Len (used as the basis for requesting the file cache size)
3. Assign a value to the loading percentage global variable counter: _one_percentnum

Step 3: Call the interface function that loads *.pef files

```
int gpio_ctrl_fd = -1;

int main(int argc, char* argv[])
{
    char re_val = 0;
    char* error_info = NULL;
    gpio_ctrl_fd = open_gpio_ctrl();
    if(gpio_ctrl_fd > 0)
    {
        if(argc == 3)
        {
            error_info = PefParseFunc(argv[1], argv[2]);
            printf(error_info);
        }
        else
        {
            printf("Input parameter error\n");
            printf("Example:./pef *.pef 0\n");
            printf("Explain:executable file;download file;device num\n");
        }
    }
    return 0;
}
```

Figure 3-6 Calling the Interface Function That Loads *.pef Files

An interface function for loading *.pef file: PefParseFunc

This function requires two incoming parameters:

2. The name of the pef file to be loaded (including the suffix)
3. The device link number to be loaded (the current device's position in the link)

3.2.3 Description of System Delay Functions

System delay function: void waitTime(long microsec);

Internal function calling: void pgDelay(unsigned short delay_time);

The unit of time for the delay (1us) is usleep(1);

Delay time calculation:

- When the value of delay_time & 0x8000 is less than or equal to 0
  (delay_time / 1000)*Emb_PEF_Delay_Cnt*((Cpu_Frequency / 8) + (+((Cpu_Frequency % 8) ? 1 : 0)))*usleep(1))
- When the value of delay_time & 0x8000 is greater than 0
  (delay_time &(~0x8000))*Emb_PEF_Delay_Cnt*((Cpu_Frequency / 8) + (+((Cpu_Frequency % 8) ? 1 : 0)))*usleep(1))

The content of the pgDelay function is shown in Figure 3-7:

```
void pgDelay(unsigned short delay_time)
{
    unsigned short loop_index = 0;
    unsigned short ms_index = 0;
    unsigned short us_index = 0;

    if (delay_time & 0x8000) /*Test for unit*/
    {
        delay_time &= ~0x8000; /*unit in milliseconds*/
    }
    else
    {
        delay_time = (unsigned short)(delay_time / 1000); /*convert to milliseconds*/
        if (delay_time <= 0)
        {
            delay_time = 1; /*delay is 1 millisecond minimum*/
        }
    }
    for (ms_index = 0; ms_index < delay_time; ms_index++)
    {
        for (us_index = 0; us_index < Emb_PEF_Delay_Cnt; us_index++)
        {
            loop_index = 0;
            do
            {
                usleep(1);
            } while (loop_index++ < ((Cpu_Frequency / 8) + (+((Cpu_Frequency % 8) ? 1 : 0))));
        }
    }
}
```

Figure 3-7 Content of pgDelay Function

3.2.4 Board Mounting

The execution of the demo is based on an embedded development environment, where JTAG is emulated through GPIO to generate level transitions for TCK, TMS, TDI, and read data from TDO. The compiled code is executed as shown in Figure 3-8:

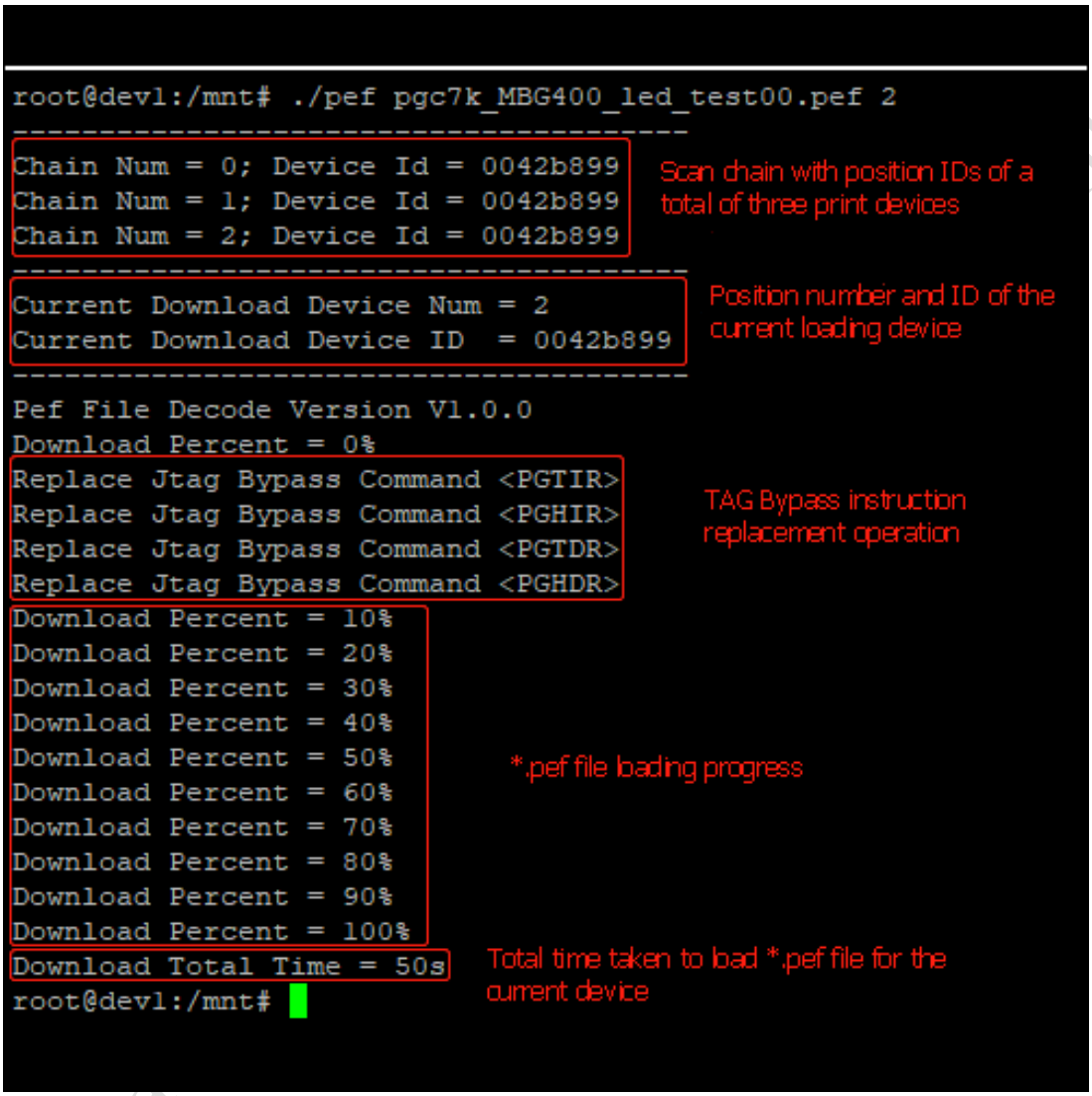

Figure 3-8 Project Demo Operation

For PGC device ID query, please refer to Section 7.2.11 of Chapter 7, titled "Device Identification Register (IDR)" in the document "UG030004_Compa Family CPLDs Configuration User Guide".

Chapter 4 Supported Versions

Supported PDS software versions: PDS2020.3-SP2.2 and above.

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.