Fingerprint Identification Algorithm

SmackFinger SDK 3.0

(for WEDS-Embedded Linux)

2009. 7. 27

Contents

1. Fingerprint Engine Library Interface

This part describes about library interface of fingerprint algorithm “SmackFinger3.0”.

1.1 Fingerprint Template

[Syntax]

```
typedef struct _tag_FPINFO
{
    unsigned long   lParam;         // Available to the application.
    unsigned short  wParam;         // Available to the application.
    unsigned char   Valid;          // Valid flag or reserved for SmackFinger.
    unsigned char   Reserved[1397]; // Reserved for SmackFinger.
} FPINFO, *P_FPINFO;// all size = 1404byte
```

[Description]

The size of a fingerprint template is 1404 bytes.
The application can use the member ***lParam***, ***wParam*** for own purpose.

1.2 Main Entry Point

[Syntax]

```
int SB_fp(int FuncNo, long Param1, long Param2, long Param3, long Param4, long Param5);
```

[Description]

The library only exports above function as main entry point.
The meaning of return value and parameters depends on ***FuncNo***. For more detail, see helper sources (***_fp.c*** and ***_fp.h***).
The summary of available ***FuncNo*** is as below.

Function Name	Value	Description
FP_GETVERSION	0	Get version and some license information.
FP_OPEN	1	Initialize the engine.
FP_CLOSE	2	Terminate the engine.
FP_INDEXDBALLOC	3	Allocate index database.
FP_INDEXDBFREE	4	Free index database.
FP_FPDBALLOC	5	Allocate template database.
FP_FPDBFREE	6	Free template database.
FP_ENROLLSTART	21	Start an enrollment.
FP_ENROLLNTH256	22	Process a fingerprint image for an enrollment.
FP_ENROLLNTHFPDATA	23	Process a fingerprint template for an

		enrollment.
FP_ENROLLMERGE	24	Merge results of three calls of FP_ENROLLNTH* and generate a template for an enrollment.
FP_PROCESSIMAGE256	31	Process a fingerprint image and generate a template, should only use for matching (transmission), not for enrollment.
FP_IDENTIFYIMAGE256	41	Identify a fingerprint image from the database (1: N).
FP_IDENTIFYFPDATA	42	Identify a template from the database (1: N).
FP_VERIFYIMAGE256	51	Verify a fingerprint image with an enrolled template (1: 1).
FP_VERIFYFPDATA	52	Verify a template with an enrolled template (1: 1).
FP_CMOSINIT	61	Initialize (or reset) the sensor.
FP_CMOSCHECKADJUST	62	Check and adjust the consistency of the sensor.
FP_ISPRESSFINGER	63	Check whether a finger placed on the sensor.
FP_CAPTUREFINGER	64	Capture a fingerprint image, 256x256.
FP_CAPTUREONEFRAME	65	Capture live image, 640x480.

[Errors]

If *SB_fp* returns an error, the value is always less than zero, and the meaning of the error value is as below.

Error Name	Value	Description
FP_ERR_SUCCESS	0	Success.
FP_ERR_PARAM	-1	Invalid parameter.
FP_ERR_NOT_ENROLLED_POS	-2	Not enrolled position.
FP_ERR_BAD_FINGER	-3	Bad fingerprint image.
FP_ERR_MERGE	-4	Cannot merge three fingerprints.
FP_ERR_IDENTIFY	-5	Cannot identify.
FP_ERR_VERIFY	-6	Cannot verify.
FP_ERR_SENSOR	-7	Sensor error.
FP_ERR_NOT_PRESSED	-8	A finger isn't placed.
DEV_ERR	-10	Device error.

1.3 Using the library

Smackbio always releases the library (*fp.so*) with helper source *_fp.c* and *_fp.h*.
These helper sources have detailed comments to help developers.
The application developer must not change the helper sources, and must use only helper functions rather than to call to main entry point (***SB_fp***) directly.
Below description is based on the helper sources.

1.3.1 Initialization and Termination

The helper function SB_FP_GETVERSION gets the version, release date and database capacity of the library. These are all read-only.

The helper function SB_FP_OPEN initializes the engine, especially checks Crypto-Memory's consistency. See below.

```
int SB_FP_OPEN(
    IN int nSensorType,
    OUT OPTIONAL BYTE **pIndexDB,
    OUT OPTIONAL BYTE **pFingerprintDB
)
{
    long params[5];
    params[0] = nSensorType; //reserved for sensor type
    params[1] = 0; //reserved
    params[2] = 0; //reserved
    params[3] = 0; //reserved
    params[4] = 0; //reserved

    engLastError = SB_fp(FP_OPEN, (long)params, (long)pIndexDB, (long)pFingerprintDB, 0, 0);

    engTemp     = (void*)params[0];     //engTemp, temporary buffer used by the engine.
    engTempSize = (int)params[1];       //engTemp Size, always is greater than 640x480 bytes.
    engCmosInitResult   = (engcmos*)params[2]; //engCmosInitResult, used for debug.
    engImage256 = (void*)params[3];
        //engImg256, converted fingerprint image (256x256), used for displaying.
    engFirstImage   = (void*)params[4];
        //engFirstImage, first captured image (640x480), used for debug.

    return engLastError;
};
```

**pIndexDB* is allocated by the library and array of the flag (1 byte) that indicates valid state of an enrolled template. **pFingerprintDB* is also allocated by the library and array of the fingerprint

template (1404 bytes).
So these two buffers construct the fingerprint database.
If *(*pIndexDB)[position]* is nonzero (TRUE), then *(*pFingerprintDB)[position]* is valid.
If *(*pIndexDB)[position]* is 0 (FALSE), then *(*pFingerprintDB)[position]* is invalid.
Of course, *position* must be in range from 0 to database capacity -1 (For example, if database capacity is 10000, valid range is 0-9999.).
If these optional parameters are NULL (0), the library doesn't allocate, the application might use SB_FP_INDEXDBALLOC, SB_FP_INDEXDBFREE, SB_FP_FPDBALLOC and SB_FP_FPDBFREE.

engTemp points the temporary buffer for the library; *engTempSize* indicates the size of the temporary buffer. The size is always greater than 640x480 bytes, so this buffer used for temporary storage of captured image. This interface is only historical relic; the application developer should not use this buffer.
engCmosInitResult points the data structure that indicates sensor init result. This is used only for debug purpose.
engImage256 is another temporary buffer of the library; the application developer should not use this buffer, but this buffer has last converted fingerprint image (256x256), so the application can display the fingerprint image on LCD.
After successful call of SB_FP_CAPTUREONEFRAME, *engTemp* has last live image (640x480), therefore *engTemp* defined as SB_FP__LIVEIMAGE.
After successful call of SB_FP_CAPTUREFINGER, *engImage256* has last converted fingerprint image (256x256), therefore *engImage256* defined as SB_FP__256IMAGE.
And, *engFirstImage* defined as SB_FP__FIRSTIMAGE, this is the first image after call of SB_FP_CMOSINIT or SB_FP_CMOSCHECKADJUST, used only for debug purpose.

If the sensor driver is trouble (because of LINUX kernel's malfunction), or Crypto-Memory (AT88SC0104C) is trouble, SB_FP_OPEN returns with failure (DEV_ERR).
If SB_FP_OPEN returns with failure, there is no guarantee for normal working state of the library.

The library doesn't perform any file system operation, so the application must load, save the database. Besides, the library doesn't perform any writing operation on the memory database.
Only exception is the consolidation of a template (intelligent adapting mechanism) while matching, in this case, the application must save consolidated template to the persistent file. See the topic about matching.

SB_FP_CLOSE terminates the engine, frees all allocated resources, therefore the application must call SB_FP_OPEN and SB_FP_CMOSINIT again to restart the engine.

1.3.2 Sensor initialization

SB_FP_CMOSINIT initializes the sensor, especially sets the correct CMOS register values.
This function has two optional parameters (*dwMechanical*, *dwExpose*), see the topic about SB_FP_CMOSCHECKADJUST.
While initializing, the library captures first image and save it into *engFirstImage*.

If SB_FP_CMOSINIT returns with failure, the application must prompt “Check Sensor” message to the user.
SB_FP_CMOSCHECKADJUST also initializes the sensor, captures one image (first image, saved in *engFirstImage*), and checks the first captured image to verify whether the sensor is trouble or not, and adjusts the brightness of the sensor. The result is saved in *engCmosInitResult*, used for debug.
This function has two optional (output) parameters (*pdwMechanical*, *pdwExpose*).
If SB_FP_CMOSCHECKADJUST returns with success (FP_ERR_SUCCESS), these two output values are valid; the application should save these values to the persistent setting file.
After then, the application should call SB_FP_CMOSINIT with pre-saved these two parameters.
If never called SB_FP_CMOSCHECKADJUST, the application must call SB_FP_CMOSINIT with default parameter values (both default values are 0).
If SB_FP_CMOSCHECKADJUST returns with failure, the application must prompt “Check Sensor” message to the user.

NOTE: SB_FP_CMOSCHECKADJUST isn’t necessary step for sensor initialization.
Legacy products have “Adjust Brightness” (“采集仪亮度校正”) menu item, the application should call SB_FP_CMOSCHECKADJUST in this menu item routine.
And, the application must check return value of SB_FP_CMOSCHECKADJUST, if it was failed, must prompt “Check Sensor” message to the user.
Furthermore, while calling of SB_FP_CMOSCHECKADJUST, the user (or manufacturer) must clean the sensor, and must obstruct any lights. (That is, must check and adjust in dark environment - this is IMPORTANT.)

If the application implemented “Adjust Brightness” menu item with calling of SB_FP_CMOSCHECKADJUST, it is recommended to do as below.

- The manufacturer should press “Adjust Brightness” menu item (or issue equivalent OCX command) for every product, just before packing.
- If the sensor changed, or persistent setting file is invalid (for example, after “Restore Def” (“恢复出厂设置”), the setting file may become to be invalid), the user should press “Adjust Brightness” menu item.

That is, SB_FP_CMOSCHECKADJUST is optional function; it will be used for checking and adjusting of the sensor.

1.3.3 Capture of the fingerprint image

SB_FP_ISPRESSFINGER checks whether a finger placed on the sensor.
If it returns with success (FP_ERR_SUCCESS), this means that a finger is placed on the sensor.
This function used especially while enrolling.

SB_FP_CAPTUREFINGER captures a fingerprint image (256x256), if a finger isn't placed on the sensor, it returns with FP_ERR_NOT_PRESSED.
If this function returns with success, *engImage256* is valid for displaying on LCD, but if the application calls other library function, *engImage256* will be invalid.

SB_FP_CAPTUREONEFRAME captures a live image (640x480), it doesn't check whether a finger placed on the sensor, this function used for debug or displaying.
If this function returns with success, *engTemp* is valid, but if the application calls other libr function, *engTemp* will be invalid.

1.3.4 Identifying and Verifying

SB_FP_IDENTIFYIMAGE256 and SB_FP_IDENTIFYFPDATA perform 1: N matching operation.
SB_FP_VERIFYIMAGE256 and SB_FP_VERIFYFPDATA perform 1: 1 matching operation.

Just before calling of image-related matching functions (SB_FP_IDENTIFYIMAGE256, SB_FP_ VERIFYIMAGE256), the application must call SB_FP_CAPTUREFINGER

SB_FP_PROCESSIMAGE256 only extracts a fingerprint template from previous captured fingerprint image (by SB_FP_CAPTUREFINGER), usually this template used for transmission.

It is recommended the application should call template-related matching functions (SB_FP_IDENTIFYFPDATA SB_FP_VERIFYFPDATA) with the result of SB_FP_PROCESS IMAGE256, not with pre enrolled template.

SmackFinge uses intelligent adapting mechanism while matching, if the matched template was adapted, the corresponding flag (**pbAdapted*) will be set.
In this case, the application must save consolidated template to the persistent file.

1.3.5 Enrollment

If the application enrolls with three fingerprint images, should call as below.

1. SB_FP_ENROLLSTART
2. SB_FP_CAPTUREFINGER
3. SB_FP_ENROLLNTH256 (1)
4. Wait to take off the finger using SB_FP_ISPRESSFINGER
5. SB_FP_CAPTUREFINGER
6. SB_FP_ENROLLNTH256 (2)
7. Wait to take off the finger using SB_FP_ISPRESSFINGER
8. SB_FP_CAPTUREFINGER
9. SB_FP_ENROLLNTH256 (3)
10. SB_FP_ENROLLMERGE

If the application enrolls with three templates, should call as below.

1. SB_FP_ENROLLSTART
2. SB_FP_ENROLLNTHFPDATA (1) with 1st template
3. SB_FP_ENROLLNTHFPDATA (2) with 2nd template
4. SB_FP_ENROLLNTHFPDATA (3) with 3rd template
5. SB_FP_ENROLLMERGE

Of course, it is recommended the application should call SB_FP_ENROLLNTHFPDATA with the result of SB_FP_PROCESSIMAGE256, not with pre-enrolled template.

After a successful enrollment, the application must save the merged template to the persistent file.

2. Sample

Because the library doesn't manipulate user ID, only manages the database by position, the sample source demonstrates how to manipulate user ID.
And, the sample source demonstrates how to call library functions, in detail.
Current library supports OPT16-OV7648, OPT16-HV7131R and OPT16-EB6048.
Furthermore, current library supports four types of database capacity. (200, 2000, 3000 and 10000)

The command line syntax of sample console application is as below.

```
./test 2k 2
```

First argument indicates library suffix, second argument indicates sensor type.
For more detail, see source.

ImageView windows application will be used to see the images by TCP/IP.