AN4420
Application note

M24SR Programmer's Application Note (based on M24SR-DISCOVERY Firmware)

Introduction

Prerequisite

It is supposed that the developer has already read the M24SR-DISCOVERY firmware user manual. No information from user point of view will be given in this document.

Description

This document describes the firmware of the M24SR-DISCOVERY board. This document helps the developer to understand how works this firmware and how tailored its own application.

M24SR can communicate with a MCU, this make M24SR a dynamic tag. Indeed the reader (for instance a smartphone) can retrieve different content staying close to the M24SR.

The reader communicates with the M24SR-DISCOVERY board through ISO14443A protocol. The MCU communicates with the M24SR by I2C bus. The MCU of the M24SR-DISCOVERY is a STM32F103.

For Standard and Premium edition the firmware is the same. Detection at boot time is done to enable or not some features of the demonstration.

Note 1: All information content in this document is applicable for M24SR-DISCOVERY standard and premium edition, unless otherwise specified.

Note 2: This documentation was written with M24SR discovery firmware version v1.1.1, board is delivered with the SW version v1.1.0, the differences between these releases are only cleaning of code mainly for Doxygen documentation. No functional modifications have been added.

Reminder

The present firmware description which is for guidance only aims at providing customers with coding information regarding their products in order for them to save time. As a result, STMicroelectronics shall not be held liable for any direct, indirect or consequential damages with respect to any claims arising from the content of such firmware and/or the use made by customers of the coding information contained herein in connection with their products.

Contents

List of tables

List of figures

1 Overview

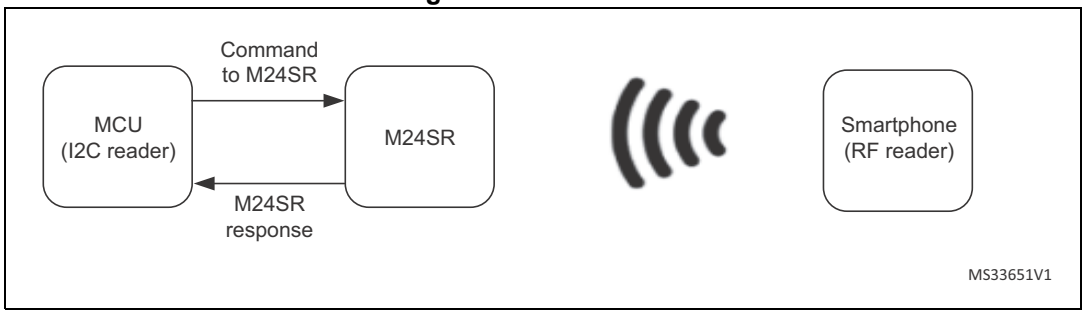
1.1 M24SR overview

M24SR is a dynamic tag IC for contactless application (ISO 14443A). It manages the RF communication with reader. It's including trame coding, RF modulation and manages anti-collision process itself.

The M24SR works as a NFC Forum Type 4 tags, it supports the detection, reading and writing operation.

M24SR can communicate with a RF reader without any external control.

Figure 1. M24SR basic



For more details concerning the M24SR device, please refer to its datasheet

1.2 STM32F103 overview

The STM32F103xx incorporates the high-performance ARM Cortex™-M3 32-bit RISC core operating at a 72 MHz frequency, high-speed embedded memories (Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I2Cs and SPIs, three USARTs, an USB and a CAN.

These features make the STM32F103xx microcontroller suitable for a wide range of applications such as motor drives, application control, medical and handheld equipment, PC and gaming peripherals, GPS platforms, industrial applications, PLCs, inverters, printers, scanners, alarm systems, video intercoms, and HVACs.

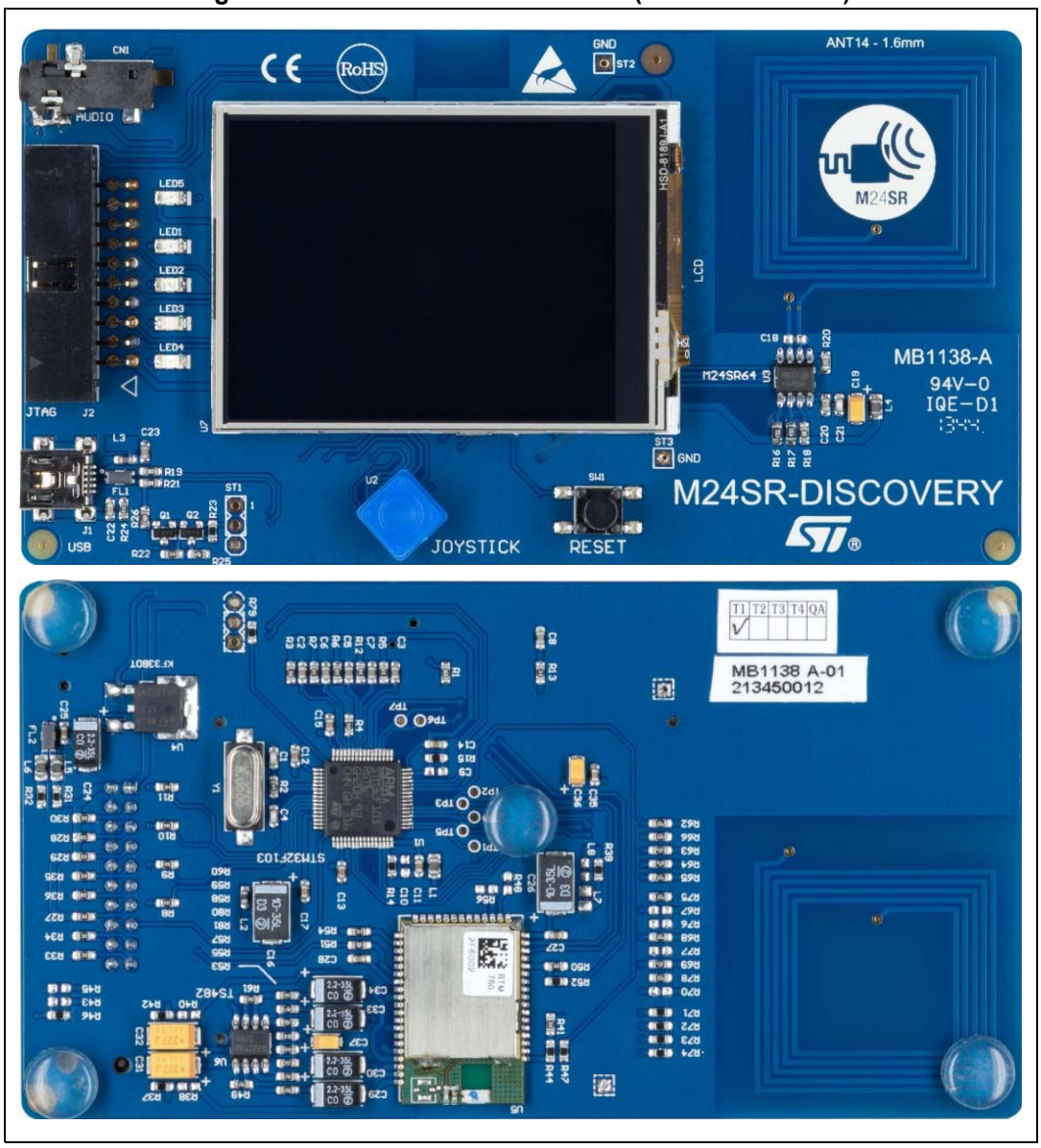
1.3 M24SR-DISCOVERY board

The M24SR-DISCOVERY is an evaluation kit which allows evaluating the performances of ST M24SR dynamic tag.

The M24SR-DISCOVERY is powered through the USB bus and no external power supply is required. It includes a M24SR tag, a 31 x 30 mm 13.56 MHz double layer inductive etched antenna (no need for tuning components).

By default, the M24SR communicates with the STM32F103RG 32-bit MCU via the I2C bus.

Figure 2. M24SR-DISCOVERY board (Premium edition)



2 Firmware overview

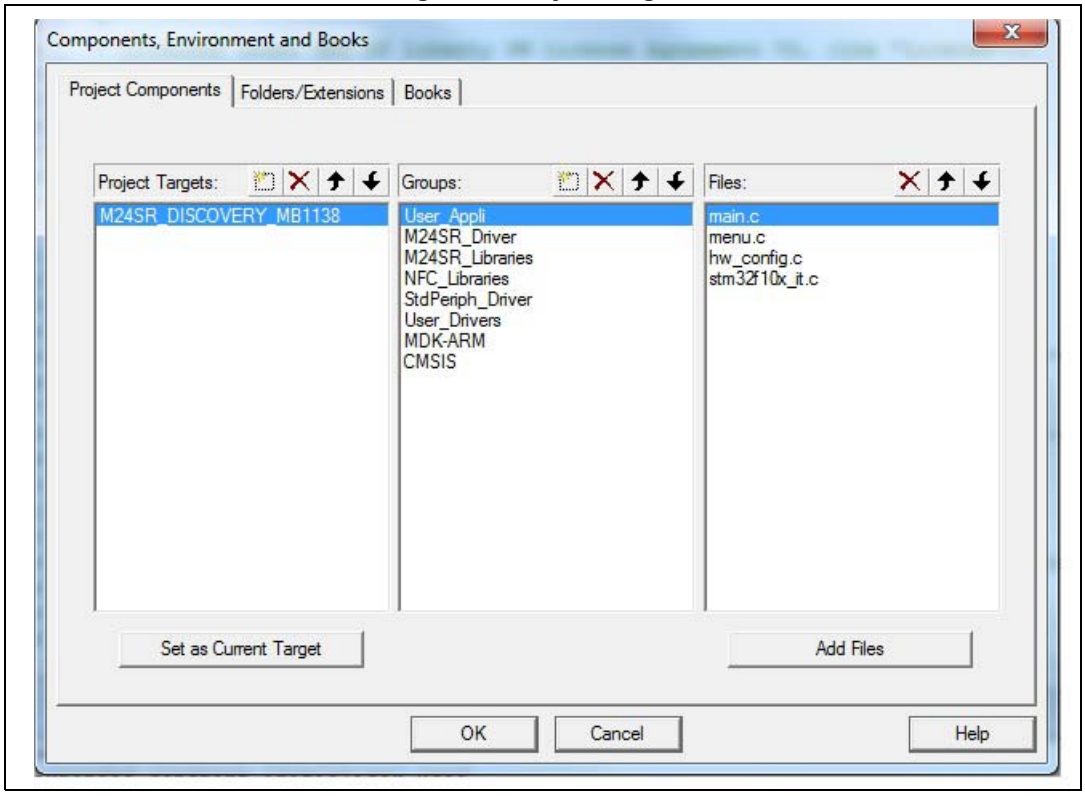
2.1 The project target

The firmware was developed on the Keil µvision and can be used on the M24SR-DISCOVERY. Thus the Keil project contains the project:

- M24SR_DISCOVERY_MB1138

The capture below shows these project targets

Figure 3. Project targets

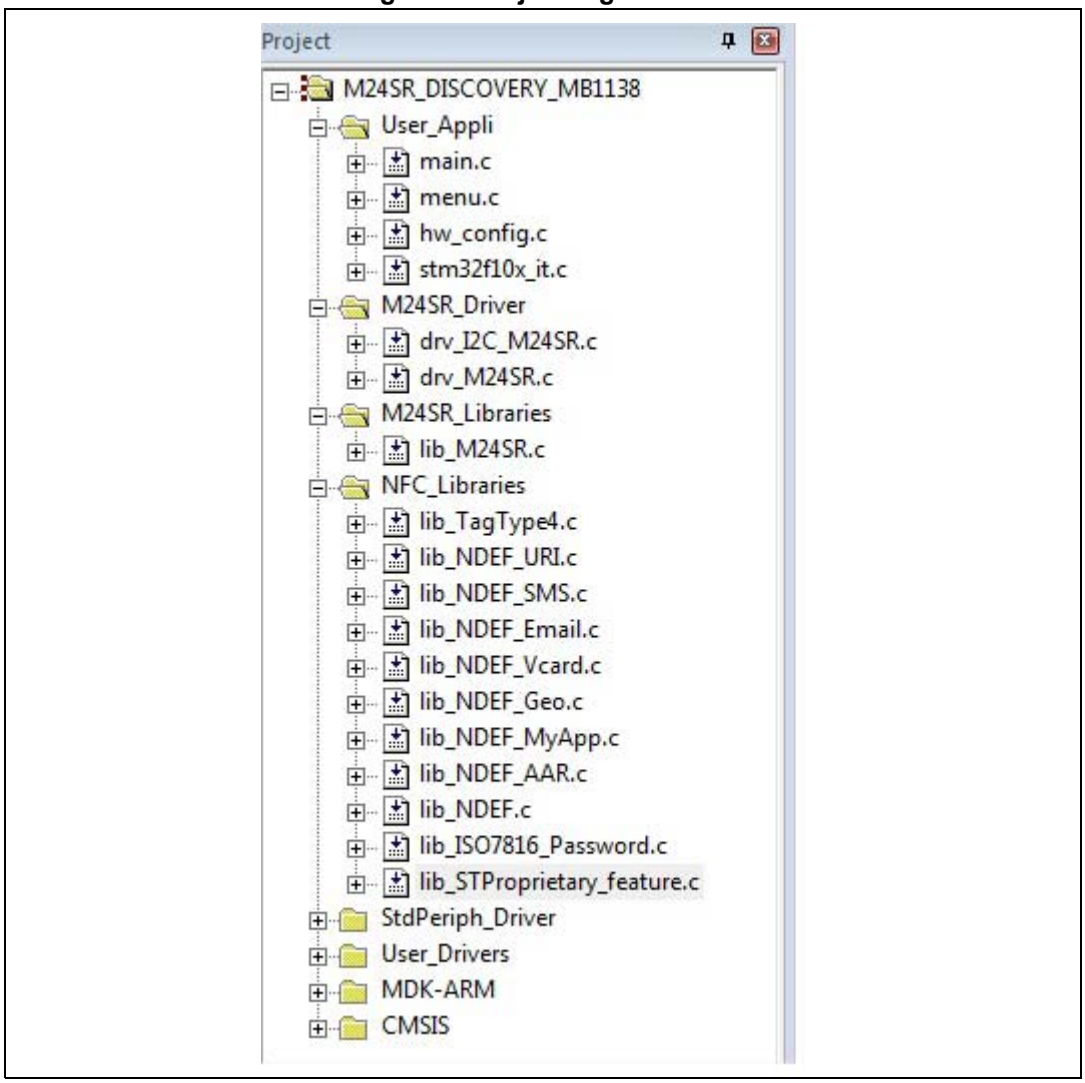


2.2 The firmware package

The firmware, developed using KEIL µvision 4.71, is delivered in a ZIP file and contains all the subdirectories and .h and .c source code files that make up the core of the application. Also the related KEIL workspace/project files are included.

The KEIL project is organized in project folders,

Figure 4. Project organization



The firmware contains all the application task source files and the related modules files and consists of the following project folders:

- **User_Appli**: the application layer
- **M24SR_Driver**: the drivers manages the GPIO of the MCU to communicate to the M24SR dynamic tag via the I²C bus. All the commands accepted by M24SR are developed in this part.
- **M24SR_Libraries**: this library includes the function to provide the service needed by the NFC_Libraries. This is an abstraction layer between NFC_libraries and M24SR_driver.
- **NFC_Libraries**: This library contains files to manage operation that can be realized by NFC. These operations can be compliant with NFC forum protocol, or ISO7816 protocol or be proprietary…
- **StdPeriph_Driver**: is the STM32 MCU standard library
- **User_Drivers**: this is linked to the feature supported by the board hardware. This folder contains the source file that manages the LCD of the M24SR-DISCOVERY board and

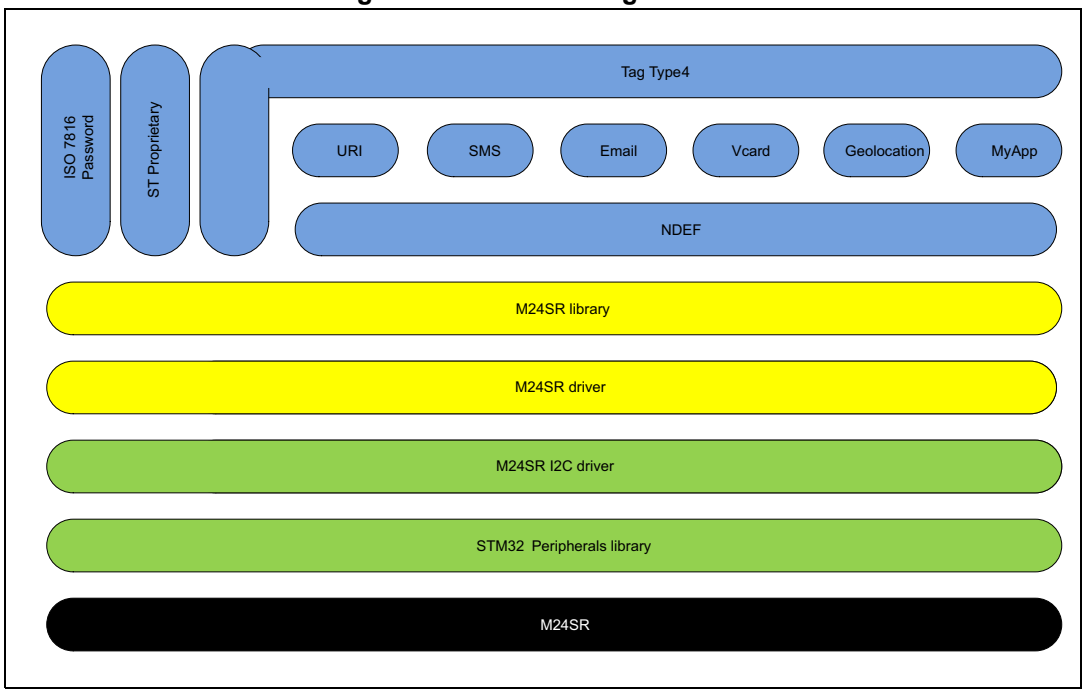
also the management of the LED. You will also find the driver of the Bluetooth module (BT module is only available on the premium edition).

2.3 Application firmware architecture

This layer architecture improves the code reusability splitting the application programming interface code (portable and reusable) provided by the libraries layer from the hardware abstraction layer code (hardware dependent and written in the STM32F10xxx libraries).

Figure 5. Firmware architecture

Figure 6. Source file organization



In Blue: You will find files not directly linked to M24SR, but to the NFC protocol. They have been creating for the purpose of the demonstration. You can keep them like this, improve or replace by your own development.

In Yellow: This is the heart of the firmware, these file are directly linked to the M24SR. You have the M24SR driver and its associated library that simplify the use of M24SR by adding some logical glue.

In Green: That's the files you have to modify regarding your HW platform, unless you are using on your design a STM32F1 to drive the M24SR, in this case you can keep the file without any modification.

3 Principle of the application

In the firmware the STM32 drives the M24SR dynamic NFC Type 4 tag.

The M24SR can receive a command from STM32 or from a RF Reader. If the command has been sent by STM32 it has been done through I2C channel, if it comes from an external reader it has been sent by RF.

To secure transfer and avoid data corruption during RF transfer, there is also a CRC. As I2C works like RF, the CRC is also used during I²C transfer.

To simplify the set of command, M24SR has been designed to support the same command whatever the interface used (I2C or RF). Few specific commands are used for only one mode.

The M24SR support basic NFC command like Select and Read CC file, Read and Update Binary for NDEF Files and it also supports few other commands to allow password and GPO management.

To illustrate M24SR NFC forum application, the firmware can configure the M24SR with URI, SMS, Email, Vcard, Geolocation and Bluetooth pairing information (BT pairing can only be done with premium edition)

To illustrate M24SR proprietary (but NFC forum protocol compliant) application, a demonstration has been developed to let user take the control over the M24SR-DISCOVERY leds and LCD with an external application running on a smartphone thanks to M24SR RF capabilities.

To illustrate M24SR product feature the firmware allows to disable/enable RF, drives M24SR GPO, and manages security content thanks to a password strategy available on M24SR.

3.1 Low layer

This part is covered by the M24SR datasheet, here is only a short reminder.

3.1.1 Data exchange protocol

The exchange of data between the RF or the I²C host and the M24SR uses three kinds of data formats, called blocks:

- I-Block: to exchange the command and the response
- R-Block: to exchange positive or negative acknowledgment
- S-Block: to use either the Deselect command or the Frame Waiting eXtension (WTX) command or response

For more information please see M24SR datasheet.

In the M24SR driver the private function M24SR_BuildIBlockCommand() is responsible to format data in I-Block

3.1.2 Data exchange synchronization

The data exchange synchronization between M24SR and host can be done in 3 ways:

- a time-out
- a polling
- an interrupt mode

The time-out is not recommended as it will impact system performance. By default you will wait the maximum time M24SR could take to answer (M24SR is programmed with FWI=5)

TimeOut = (256x16/fc)*2^FWI, see chapter 11.7.1 of the NFCForum-TS-DigitalProtocol-1.0.

If you select the polling mode you will keep the I2C bus busy. Most of systems use a dedicated I2C bus by component, so this is not an issue.

The last solution is to use the interruption provided by M24SR GPO, but your system must have one more GPIO available to connect to M24SR and an interruption controller that can detect falling edges. (As M24SR can only drive GPO to low state)

In the demonstration the last mode is used, selected by a compiling flag (I2C_GPO_SYNCHRO_ALLOWED) in drv_I2C_M24SR.h file.

3.1.3 CRC Computation

The CRC is as defined in ISO/IEC 13239. The initial register content shall be 0x6363 and the register content shall not be inverted after calculation.

To avoid reading the full ISO/IEC 13239 spec you will find in annex 1, the code implementation in C language to compute the CRC.

3.2 Driver and Lib M24SR layer (general part)

M24SR cannot be accessed by I2C and RF at the same time.

For this reason there is an open session phase at the beginning of any operation between host and M24SR.

When the host send the open session command to the M24SR:

If M24SR has already a session opened, the requester will receive an error response. If, on the other hand, M24SR does not have sessions opened, the request will be fulfilled and a session will be opened.

Note: On I²C side there is a specific Kill Session command, with this command I²C host will be sure to open a session with M24SR but if a session was already opened between M24SR and RF host, the session will be closed with RF without any chance for RF host to close properly. This means that M24SR content can be corrupted.

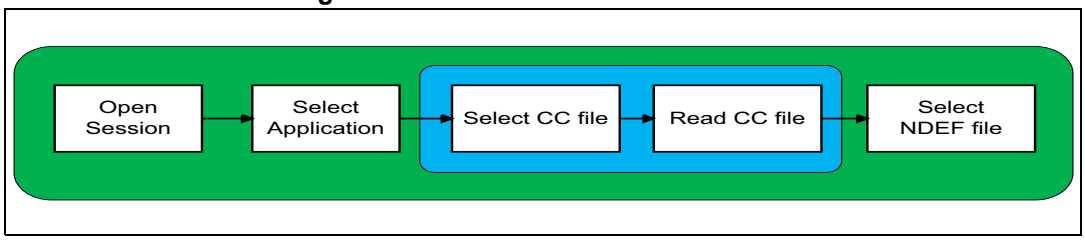
It is not recommended to use this command if RF host have write permission.

M24SR is NFC forum compliant; this means that user must follow NFC forum recommendation to access M24SR by RF.

As a consequence the I²C host can use the same mechanism to access M24SR, but some optimization has been done in this firmware as explain here after.

For any operation it's required by NFC forum to proceed like this:

Figure 7. RF Select File Command Flow



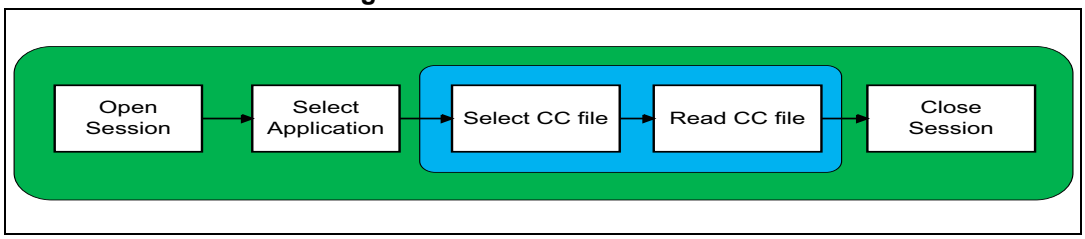
When an RF reader opens a session it does not know which tag it will access and if it will be the same tag each time it will open a session, so reading CC file is mandatory.

With our dynamic M24SR tag this is not the same case. The I2C host will always have the same tag to drive. For this reason and to improve performance, the select CC file and read CC file operation is done during initialization phase and not at each access.

So in the firmware this strategy has been adopted:

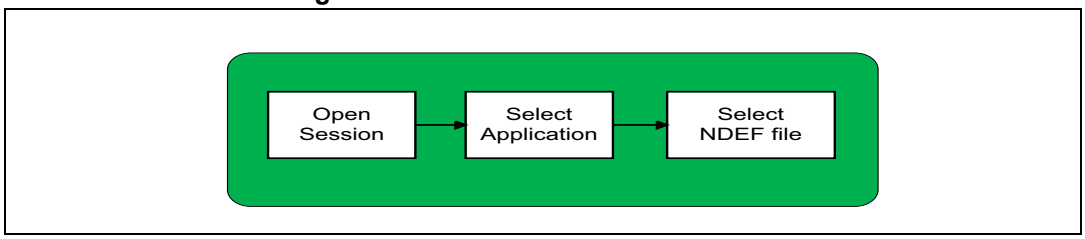
- Init done at power-up
- CC file information stored in a CCFile Structure.

Figure 8. I2C Init Command Flow



For all the following accesses done to M24SR by I^2C CC file won't be ready anymore.

Figure 9. I2C Select File Command Flow



This sequencing of command sent to M24SR is managed by the function M24SR_OpenNDEFSession available in lib_M24SR.c file.

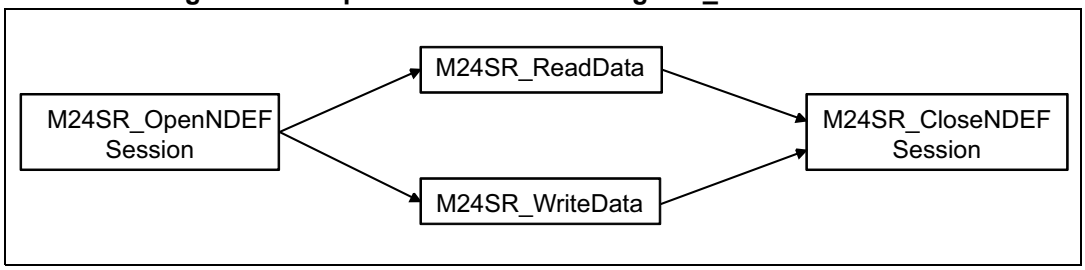
The lib_M24SR.c file also implements the loop to read or write data with a number of data that exceed the M24SR buffer size.

At the end the lib_M24SR.c file simplify a lot the integration of the M24SR in your design.

Indeed after a call to the M24SR_Initialization function during your boot sequence:

1. you only have to call M24SR_OpenNDEFSession to access M24SR.
2. then M24SR_ReadData or M24SR_WriteData regarding the action you want to do.
3. and finally M24SR_CloseNDEFSession to release the session and let RF reader access the M24SR.

Figure 10. Simple M24R access through lib_M24SR interface



3.3 NFC library layer (general part)

For the purpose of the demonstration with M24SR-DISCOVERY a “NFC library” has been developed to manage tag content. Indeed the content of the M24SR must be formatted in respect to the NFC forum standard to enable natively interaction with smartphone (for more information please refer to NFCForum-TS-NDEF_1.0 documentation).

This library has been created to manage this content Read, Parse and Identify content type or Format and Write NDEF message.

So this part of the software is not directly linked to M24SR but to any NFC Forum tag. That’s the reason why a lib_wrapper.h file has been created. Inside the NFC library all the function call made to the lower layer are done through this API:

```
TagT4Init();
OpenNDEFSession();
CloseNDEFSession();
ReadData();
WriteData();
```

The file lib_wrapper.h allows redirecting function call to M24SR functions.

For the purpose of the demonstration the NFC library manage NDEF message that represent:

- URI
- SMS
- Email
- Vcard
- Geolocation
- And a private NDEF message structure (see Annex 2 for more information)
- It’s also possible to add AAR record.

The entry point of this library is the file lib_TagType4.c with the following functions:

```
u16 TT4_Init( void );
u16 TT4_ReadNDEF(u8 *pNDEF);
u16 TT4_WriteNDEF(u8 *pNDEF);
u16 TT4_ReadURI(sURI_Info *pURI);
u16 TT4_WriteURI(sURI_Info *pURI);
u16 TT4_ReadSMS(sSMSInfo *pSMS);
u16 TT4_WriteSMS(sSMSInfo *pSMS);
u16 TT4_ReadEmail(sEmailInfo *pEmailStruct);
```

```
u16 TT4_WriteEmail(sEmailInfo *pEmailStruct);
u16 TT4_ReadVcard(sVcardInfo *pVcard);
u16 TT4_WriteVcard(sVcardInfo *pVcard);
u16 TT4_ReadGeo(sGeoInfo *pGeo);
u16 TT4_WriteGeo(sGeoInfo *pGeo);
u16 TT4_ReadMyApp(sMyAppInfo *pMyAppStruct);
u16 TT4_WriteMyApp(sMyAppInfo *pMyAppStruct);
u16 TT4_AddAAR(sAARInfo *pAAR);
```

3.4 Password and specific M24SR feature

M24SR embeds a password feature to protect M24SR content.

It's also possible to drive a GPO.

This feature are not described by NFCForum, the password are describe by ISO7816 standard.

The GPO is a complete proprietary feature of the M24SR.

To keep consistency at a software point of view in the NFC_Libraries you will find 2 files:

Lib_ISO7816_Password.c and libSTProprietary_feature.c to manage

They are used to manage password, read/write only mode and the GPO.

As we are in the NFC_libraries and even if the code seems to be more specific to the driven component, the choice was made to keep the same strategy. So to keep NFC_libraries not tied to the HW.

For this reason the lib_wrapper.h also helps to redirect the function call to the M24SR driver.

3.4.1 Password

Here is short resume about M24SR password strategy:

M24SR has always 2 128bits passwords saved inside the chip (one read password and one write password). By default the passwords are 0x00000000000000000000000000000000.

Then the password can be activated or not. This means that the password exists but it is not necessary required for accessing M24SR data.

If the password is activated it will be required when trying to access M24SR in read or write mode (depending on the password activated).

To activate a password, the password saved in M24SR must be presented.

To change read password, write password saved in M24SR must be presented (as it has been considered as a write operation)

To change write password, write password saved in M24SR must be presented.

These read and write passwords can be used through I2C and RF access.

On top of that implementation a super user mode has been put in place by creating a super user password named I2C password (because it can be used only through M24SR I2C interface)

```
SuperUserPassword = I2CPassword
```

This password can be used to modify read and write passwords.

For more Password information please refer to the M24SR datasheet.

Inside lib_ISO7816_Password.c file this is managed by the functions:

```
u16 ISO7816_EnableReadPwd(u8* pCurrentWritePassword, u8* pNewPassword);
u16 ISO7816_DisableReadPwd(u8* pCurrentWritePassword);
u16 ISO7816_EnableWritePwd(u8* pCurrentWritePassword, u8* pNewPassword);
u16 ISO7816_DisableWritePwd(u8* pCurrentWritePassword);
u16 ISO7816_DisableAllPassword(u8* pSuperUserPassword);
```

3.4.2 Specific M24SR feature

Read & Write Only mode

It's also possible to configure M24SR in Read or Write only mode.

This configuration can be done only using write password.

A set of function to perform this action are available in the file lib_STProprietary_feature.c :

```
u16 STProprietary_EnableReadOnly(u8* pCurrentWritePassword);
u16 STProprietary_DisableReadOnly(u8* pCurrentWritePassword);
u16 STProprietary_EnableWriteOnly(u8* pCurrentWritePassword);
u16 STProprietary_DisableWriteOnly(u8* pCurrentWritePassword);
```

GPO feature

M24SR has a GPO that can be used in your design to give some information to the host.

GPO has 3 different configurations regarding the M24SR state.

1) No RF & I2C session open on M24SR side:

The GPO is not driven, GPO is high-Z but pull-upped externally on M24SR-DISCOVERY board.

2)An I2C session is opened:

The GPO behavior follows the configuration set for I2C session. In this firmware demonstration is used to know when M24SR responses are available.

3)A RF session is opened:

The GPO behavior follows the configuration set for RF session. In this menu is used to send interruption to MCU when RF decides to do it.

For more GPO information please refer to the M24SR datasheet.

It's also in the file lib_STProprietary_feature.c that you will find the API to manage GPO:

```
u16 STProprietary_GPOConfig(uc8 GPO_RFconfig, uc8 mode);
```

4 Firmware functions

You will find documentation (in html format) in the directory “Public document/doxygen” inside the firmware delivery. Launch the file OpenM24SRDocumentation.bat, your browser will display the documentation.

Extract of the Doxygen documentation to illustrate available content in the figures below:

Figure 11. Doxygen function documentation example

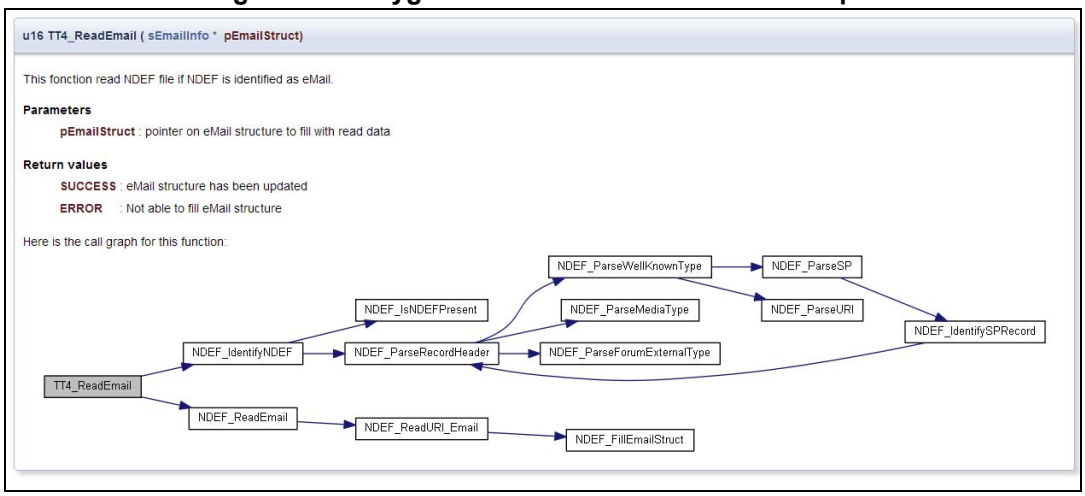
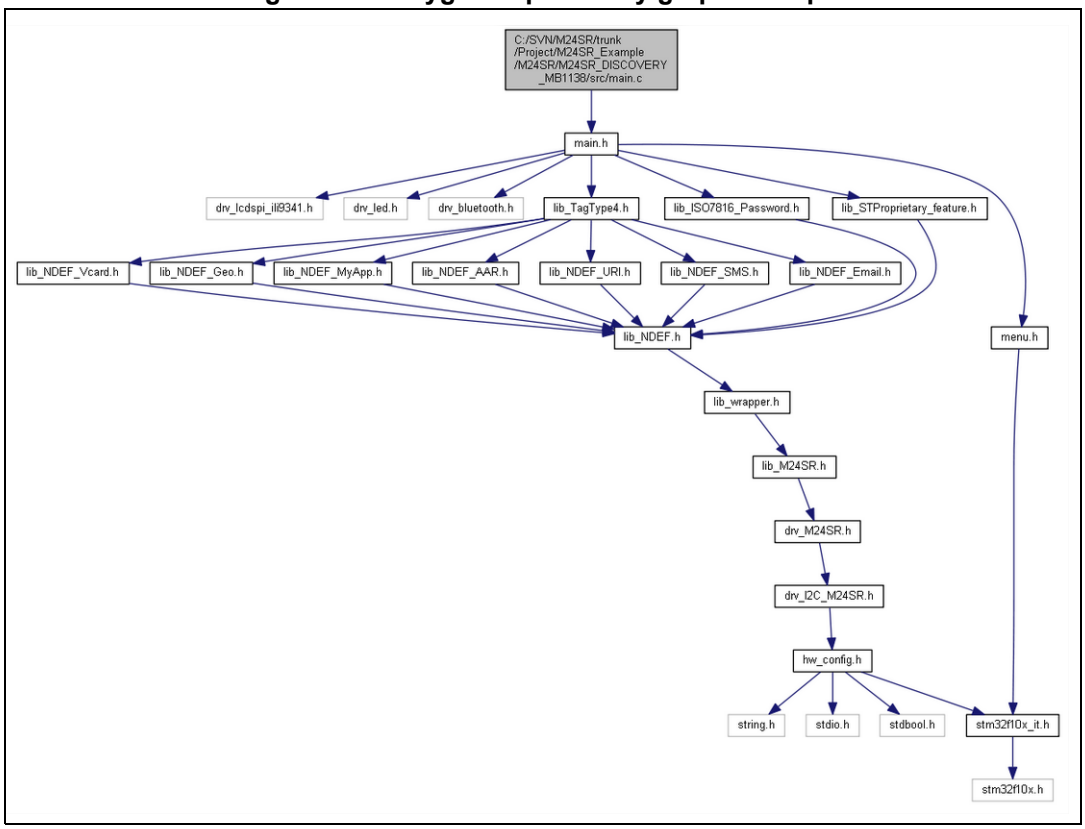


Figure 12. Doxygen dependency graph example



Appendix 1: Acronyms and Notational Conventions

Acronyms

ISO: International Organization for Standardization

IEC: International Electrotechnical Commission

IAP: In Application Programming

HID: Human interface device

MCU: Micro Controller Unit

NFC: near field communication

RF: radio frequency

RFID: Radio Frequency Identification

USB: Universal Serial Bus

Representation of Numbers

The following conventions and notations apply in this document unless otherwise stated.

Binary number representation

Binary numbers are represented by strings of digits 0 and 1 shown with the most significant bit (MSB) on the left, the least significant bit (LSB) on the right, and a “0b” added at the beginning.

Example: 0b11110101

Hexadecimal number representation

Hexadecimal numbers are represented by using the numbers 0 to 9 and the characters A – F, and adding an “0x” at the beginning. The Most Significant Byte (MSB) is shown on the left and the Least Significant Byte (LSB) on the right.

Example: 0xF5

Decimal number representation

Decimal numbers are represented as is without any trailing character.

Example: 245

Annex 1: CRC calculation

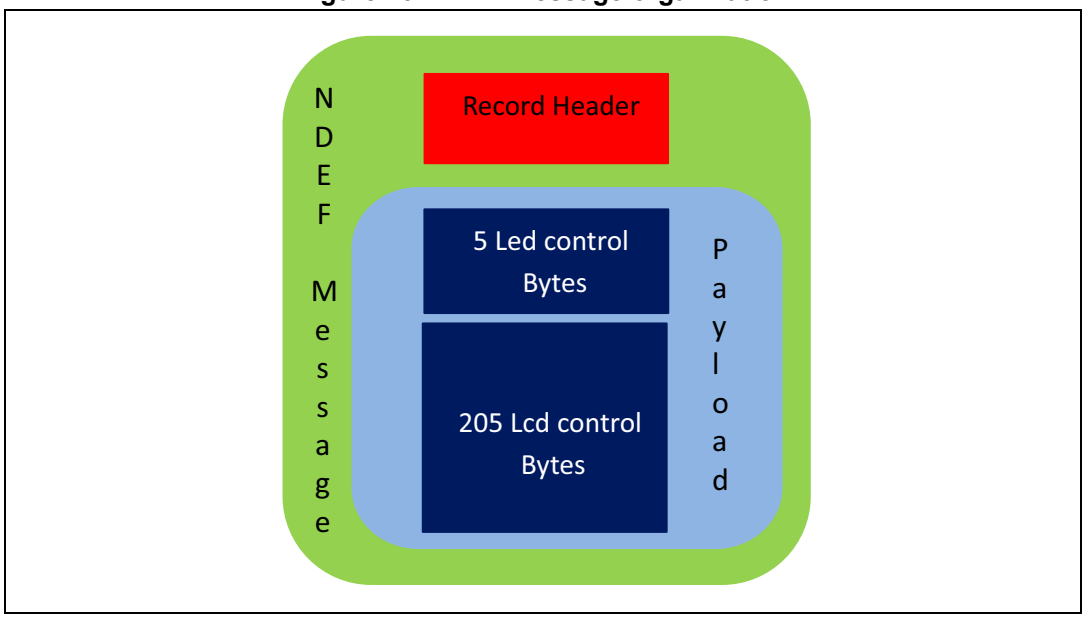
```
/**
  * @brief  This function updates the CRC
  * @param  None
  * @retval None
  */
static uint16_t M24SR_UpdateCrc (uint8_t ch, uint16_t *lpwCrc)
{
    ch = (ch^(uint8_t)((*lpwCrc) & 0x00FF));
    ch = (ch^(ch<<4));
    *lpwCrc = (*lpwCrc >> 8)^((uint16_t)ch <<
8)^((uint16_t)ch<<3)^((uint16_t)ch>>4);

    return(*lpwCrc);
}

/**
  * @brief  This function returns the CRC 16
  * @param  Data : pointer on the data used to compute the CRC16
  * @param  Length : number of byte of the data
  * @retval CRC16
  */
static uint16_t M24SR_ComputeCrc(uc8 *Data, uint8_t Length)
{
uint8_t chBlock;
uint16_t wCrc;

 wCrc = 0x6363; // ITU-V.41

 do {
   chBlock = *Data++;
   M24SR_UpdateCrc(chBlock, &wCrc);
 } while (--Length);

return wCrc ;
```

Annex 2: Private application example

The example illustrates the use of NFC forum specification in developing your own application. The benefit of this choice is seen with an Android smartphone. By adding an AAR record, the android smartphone that will read M24SR, will automatically launch your application to manage M24SR content.

For this purpose the external type: “**st.com:m24sr_discovery_democtrl**” is used.

Figure 13. NDEF message organization



The record header can be representing like this:

```
/* External Type Record Header */
/***********************************/
/*  7 |  6 |  5 |  4 |  3 | 2  1  0 */
/*---------------------------------*/
/* MB   ME   CF   SR   IL    TNF   */  /* <---- IL=0 and CF=0 TNF=4 NFC Forum external type*/
/*---------------------------------*/
/*          TYPE LENGTH            */
/*---------------------------------*/
/*        PAYLOAD LENGTH 3         */  /* <---- Not Used  */
/*---------------------------------*/
/*        PAYLOAD LENGTH 2         */  /* <---- Not Used  */
/*---------------------------------*/
/*        PAYLOAD LENGTH 1         */  /* <---- Not Used  */
/*---------------------------------*/
/*        PAYLOAD LENGTH 0         */  /* The payload will always be 5 + 25*8=205 bytes */
/*---------------------------------*/
/*          ID LENGTH              */  /* <---- Not Used  */
/*---------------------------------*/
/*             TYPE                */  /* st.com:m24sr_discovery_democtrl */
/*---------------------------------*/
/*              ID                 */  /* <---- Not Used  */
/***********************************/
```

The payload is organized like this:

Table 1. Payload organization (Led config part)

Example of settings	LED CONFIG				BLINK CONFIG
	Led1 Init state off No blink	Led2 Init state on No blink	Led3 Init state off Blink enabled	Led4 Init state on Blink enabled	Blink speed: LOW
payload address 0	0x00	0x01	0x02	0x03	0x01

LED CONFIG, 1 byte used like this:
The 2 lower bits are used to indicate initial led state and if the led will blink or stay fixed
bit0: if 0 led is OFF, if 1 led is ON
bit 1: if 0 Led won't blink, if 1 led will blink

BLINK CONFIG, 1 byte with this value:
0x00 Disable blink (stay in init config), 0x01 Low speed, 0x02 Medium Speed, 0x03 High speed

1st Byte Led1 config, 2nd Byte Led2 config, 3rd Byte Led3 config, 4th Byte Led4 config, 5th Byte Blink config

The following bytes are used to retrieve information that will be displayed on the LCD:

Table 2. Payload organization (Lcd config part)

Example of settings	LINE NUMBER	BACKGROUND COLOR	FONT COLOR	ASCII STRING
	Line 1	Black	White	abcdefghijklmnopqrst
payload address 5	0x01	0x????	0x????	20 bytes, ASCII string
payload address 30	0x02	0x????	0x????	20 bytes, ASCII string
payload address 55	0x03	0x????	0x????	20 bytes, ASCII string
payload address 80	0x04	0x????	0x????	20 bytes, ASCII string
payload address 105	0x05	0x????	0x????	20 bytes, ASCII string
payload address 130	0x06	0x????	0x????	20 bytes, ASCII string
payload address 155	0x07	0x????	0x????	20 bytes, ASCII string
payload address 180	0x08	0x????	0x????	20 bytes, ASCII string

LINE NUMBER, 1 byte to indicate line to update on the target screen
value can be 0x01 to 0x08, other are RFU
BACKGROUND COLOR, 2 bytes used like this:
RGB value of color to display on screen in big endian
FONT COLOR, 2bytes used like this:
RGB value of color to display on screen in big endian
ASCII STRING, 20 bytes used like this:
Should always be 20 bytes (no more, no less) (add space to complete line if needed)

5 Revision history

Table 3. Document revision history

Date	Revision	Changes
13-Feb-2014	1	Initial release.