AN4500
Application note

How to display size-optimized pictures on a 4-grey level E-Paper from STM32 embedded memory

Introduction

This application note describes how to optimize the size of the black and white pictures to be stored into the embedded flash memory of the STM32 microcontrollers and how to display them on a E-Paper display.

The application note presents how to prepare and encode the black and white picture and presents the software solution to decompress the picture to display it on a 4-grey level E-Paper display.

The STM32 microcontrollers allow to interface the E-Paper display using specific peripherals to send data/command to the E-Paper display controller, and to drive specific GPIOs to manage the E-Paper control pins.

The application note and the associated software (STSW-STM32152) are based on STM32L053 discovery kit (32L0538DISCOVERY) offering an embedded E-Paper display. They can be reused easily for any STM32 microcontroller customer board with minor changes (clock configurations, GPIOs definition according to the board schematics).

E-Paper display is a 2,1 inches active area containing 172x72 pixels, with 2-bit full display capabilities. For more details about the E-Paper functionalities which would not be presented into this application note, please refer to the GDE021A1 specification available on the ST web.

Table 1. Applicable products, tools & software

Type	Reference products
STM32 embedded software	STSW-STM32152
STM32 MCU evaluation tools	32L0538DISCOVERY

Contents

List of tables

List of figures

1 Implementation example

1.1 General overview

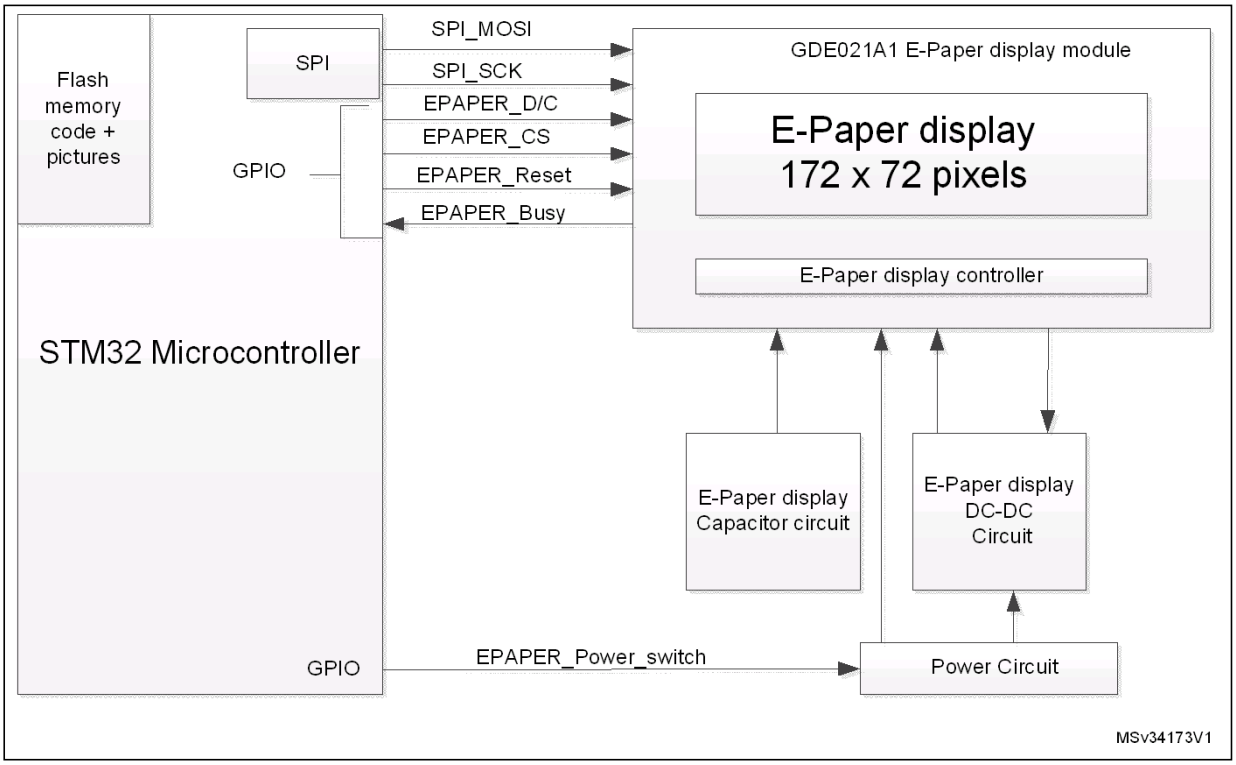
The example presented in this application note provides typical hardware and software implementation basics to interconnect with some E-Paper with one STM32 microcontroller.

Typically, the system embeds:

– One STM32 microcontroller

– An E-Paper display with external components used by the charge pump of the E-Paper display driver embedded into the GDE021A1 E-Paper display module.

Figure 1. Typical implementation design description



The E-Paper display module is connected to the STM32 MCU via the SPI interface receiving the data and commands to configure the display and to transfer the pictures into the E-Paper module internal RAM buffer.

The pictures are stored into the internal flash program memory to minimize external resources. It consists of four pictures displayed to promote the STM32L053 main features. It is an endless application rolling back to the first picture when the 4th one has been displayed on the E-Paper module.

Note: The pictures could be stored into external memories (like SD cards or external flash memory) if the embedded flash memory size is too small to content the application code + the pictures library. In such case, there is probably no need to reduce the picture size and to apply the expansion algorithm present into this application note. Data processing time will be reduced since the picture is sent to the EPD's buffer without any pre-processing.

1.2 STM32 configuration

General requirements

The developed example is mainly based on the STM32L053 discovery kit but the functional and structural description is similar for most applications and platforms.

1.2.1 SPI peripheral

The communication between the MCU and the E-Paper display is served by the SPI protocol. The MCU configures the SPI in master 8-bit mode with NSS managed by software. There is no CRC needed here. The E-Paper display module can be written only through SPI channel. This is why the MOSI line is defined and not the MISO line.

The frequency used for the communication is 2 MHz starting from the HSI set to 16 MHz from which the prescaler by 8 is applied into the SPI baud rate generator.

1.2.2 System clock

For the application note, the high speed internal oscillator has been set to be the system clock. There is no divider on the clock path, meaning that the APB and AHB bus frequencies are 16MHz.

1.2.3 Specific GPIOs to control the E-Paper display

Some specifics signals are used to control E-Paper display:

- EPAPER_Reset: This signal is generated by the MCU to reset the E-Paper Registers and to clear any on-going refresh.
- EPAPER_D/C: Data/Command line. This output generated by the MCU allows the E-Paper display module to know if the value sent by the SPI identifies a command or a data.
- EPAPER_CS: This is the Chip Select pin. This output generated by the MCU is used to enable the SPI Slave embedded in the E-Paper display module.
- EPAPER_Busy: This signal comes from the E-Paper display module to inform the MCU about the module status. When the software launches a refresh, the busy bit will be set and no more action has to be done on the E-Paper display (no more command or data) to avoid any corruption on the display.
- EPAPER_Power_switch: This GPIO is used to control an analog switch to power on/ power off the E-Paper display module to save power consumption depending the application.

1.3 E-Paper display configuration

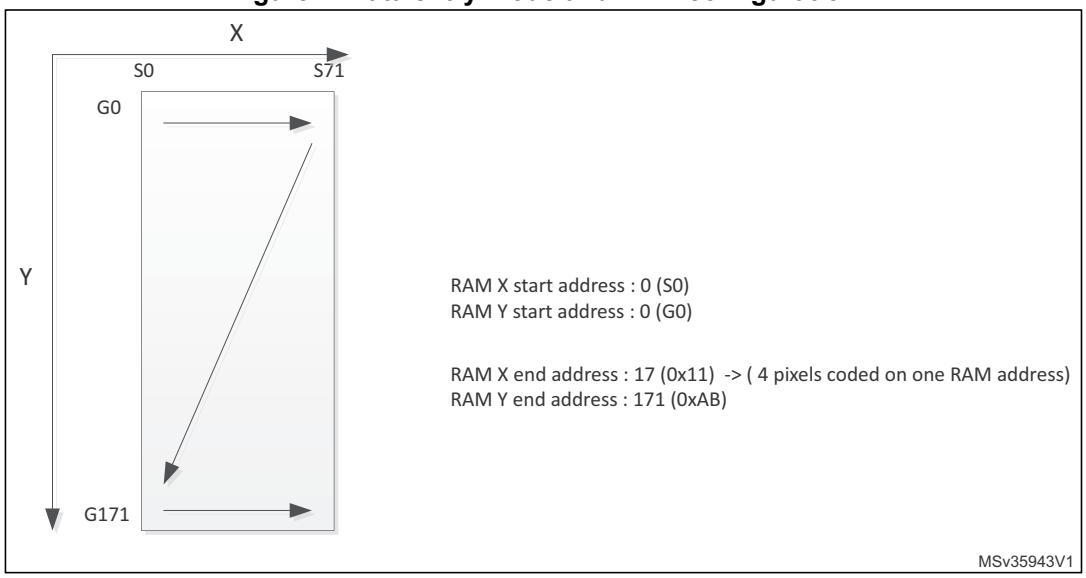
The E-Paper display used in the STM32L053 discovery kit is very configurable. It is recommended to refer to the E-Paper specification (Model GDE021A1) to better assimilate the way this application note is going to deal with the E-Paper module.

The E-Paper module internal RAM has to be filled with data coding the picture frame through the MCU SPI interface (refer to *Table 2: Ram address map* for the RAM mapping).

The RAM address will be incremented or decremented after each write operation depending on the E-Paper module configuration. The address counter may be updated in X or Y direction. Each of the axis can be configured independently concerning the start/end address for X and Y coordinates.

For this application note, the module has been configured to increment the address counter in X direction from 0 to 71, then incrementing the Y position from 0 to 171.

Figure 2. Data entry mode and RAM configuration



The configuration used in this application note does not require the E-Paper display module to enter in deep sleep mode after update operation. It means that the RAM data is retained in between two update refresh cycles.

If the application requires to decrease consumption when there is no on-going update, it could be possible to configure the E-Paper to reach the deep sleep mode. The consumption is then reduced by a factor 10 (down to 2 µA), but the RAM content is not retained in such case. As a drawback, it will no more be possible to refresh some part of the picture to display in such case, and all the pixels of the full picture have to be reloaded in the RAM through the SPI before refreshing the display.

The Waveform Look Up Table (LUT) used here (default one, without any temperature range). For more details, please refer to the GDE021A1 specification.

Figure 3. Waveform look up table

```
/* Look-up table for the epaper (90 bytes) */
const unsigned char init_data[]={
0x82,0x00,0x00,0x00,0xAA,0x00,0x00,0x00,
0xAA,0xAA,0x00,0x00,0xAA,0xAA,0xAA,0x00,
0x55,0xAA,0xAA,0x00,0x55,0x55,0x55,0x55,
0xAA,0xAA,0xAA,0xAA,0x55,0x55,0x55,0x55,
0xAA,0xAA,0xAA,0xAA,0x15,0x15,0x15,0x15,
0x05,0x05,0x05,0x05,0x01,0x01,0x01,0x01,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x41,0x45,0xF1,0xFF,0x5F,0x55,0x01,0x00,
0x00,0x00,};//waveform
```

The E-Paper RAM mapping for this application note is organized like described in the *Table 2: Ram address map*.

Table 2. Ram address map

			X-ADDR (SOURCE)													
			S0	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**			**S68**	**S69**	**S70**	**S71**
			00h				**01h**				**...**		**11h**			
Y- ADDR (Gate)	G0	00h	DB0 [7.6]	DB0 [5.4]	DB0 [3.2]	DB0 [1.0]	DB1 [7.6]	DB1 [5.4]	DB1 [3.2]	DB1 [1.0]	...	...	DB17 [7.6]	DB17 [5.4]	DB17 [3.2]	DB17 [1.0]
	G1	01h	DB18 [7.6]	DB18 [5.4]	DB18 [3.2]	DB18 [1.0]	DB19 [7.6]	DB19 [5.4]	DB19 [3.2]	DB19 [1.0]	...	...	DB35 [7.6]	DB35 [5.4]	DB35 [3.2]	DB35 [1.0]
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	G1 70	AAh	DB3060 [7.6]	DB3060 [5.4]	DB3060 [3.2]	DB3060 [1.0]	DB3061 [7.6]	DB3061 [5.4]	DB3061 [3.2]	DB3061 [1.0]	...	...	DB3077 [7.6]	DB3077 [5.4]	DB3077 [3.2]	DB3077 [1.0]
	G1 71	ABh	DB3078 [7.6]	DB3078 [5.4]	DB3078 [3.2]	DB3078 [1.0]	DB3079 1[7.6]	DB3079 [5.4]	DB3079 [3.2]	DB3079 [1.0]	...	...	DB3095 [7.6]	DB3095 [5.4]	DB3095 [3.2]	DB3095 [1.0]

Each RAM address stores 4-pixels, each pixel coded in 4-grey level. The SPI of the STM32 is configured to set the transfer with the MSB first feature, like requiered by the E-Paper display module.

The picture needs to be coded following these constraints. For more details, please refer to *Section 1.4: Picture creation and size compression*.

1.4 Picture creation and size compression

The picture to display on the E-Paper module has to be built in a smart way in order to get a corresponding software constant well formatted. It would be easier to process it to display it on the E-Paper module.

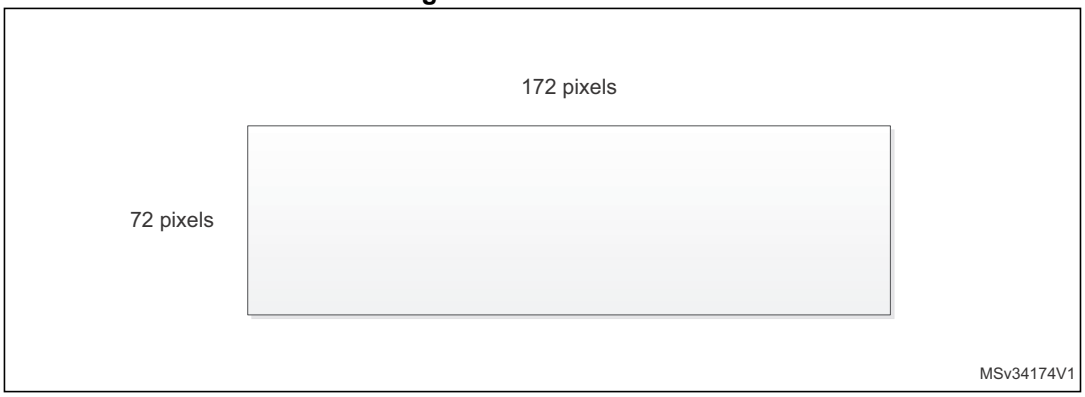
This picture is embedded into the internal non-volatile flash program area of the STM32L053. The picture has in consequence to be compressed in a specific format allowing to code 1-bit by pixel to save space memory. In consequence, the picture will be coded in black and white and no more as a 4-grey level picture.

As the E-Paper display RAM module needs to receive a 2-bit by pixel encoding mode, an expansion processing starting from the picture stored into the non-volatile flash program memory is requiered. This operation will be managed by the software (please refer to the section *Section 1.5: Picture data expansion to load the embedded E-Paper RAM*)

1.4.1 Picture frame

Picture frame can be edited using the well-known “PAINT” software from Windows for instance. Blank picture has to be edited at first (like in *Figure 4: Picture frame*). The blank frame is 172x72 pixels.

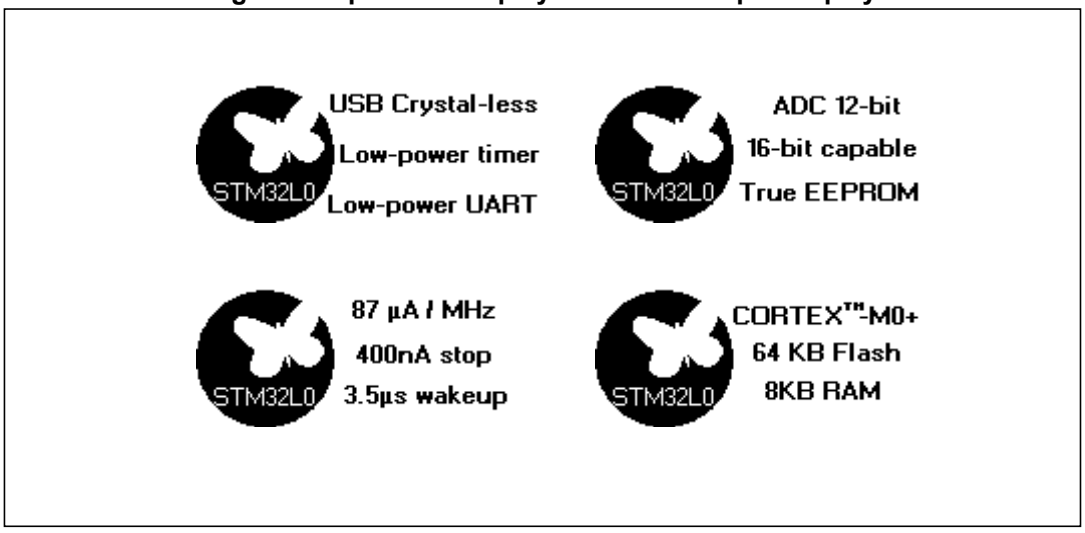
Figure 4. Picture frame



Then the picture to display has to be put inside this frame. If a text has to take place in the picture, the dedicated font “small fonts” may be used to offer a convenient display for the text.

The picture displayed to illustrate this application note are presented in *Figure 5: 4 pictures displayed on the E-Paper display*.

Figure 5. 4 pictures displayed on the E-Paper display



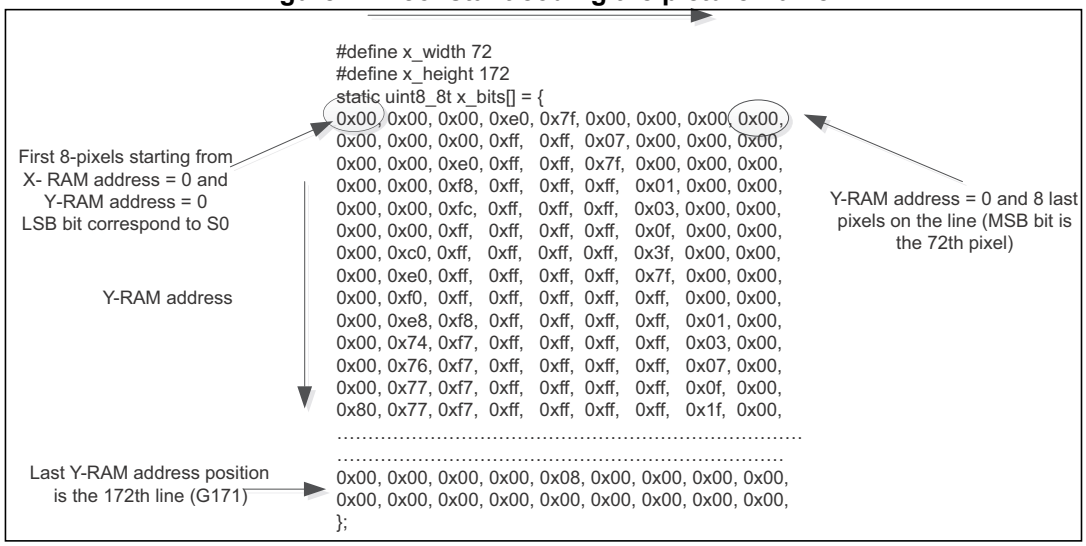
To obtain a constant coding the picture in a convenient way to minimize the data processing, it is interesting to rotate the picture by 90° right (*Figure 6: 90° right picture rotation*). In such a way, as illustrate into the *Figure 7*, the number of bytes into the constant will code all the pixels and will be an integer multiple of the total number of the pixels in the full frame.

Figure 6. 90° right picture rotation

Then, each picture has to be saved with PAINT with the format .bmp picture with a monochrome bitmap type.

In order to get the C-constant directly encoded with the picture to display, it is interesting to save the picture in XBM-X11 format, using for instance the freeware Xnview. Here is the file format obtained after this operation (*Figure 7: C constant coding the picture frame*). The constant can be directly used and declared in the C project.

Figure 7. C constant coding the picture frame



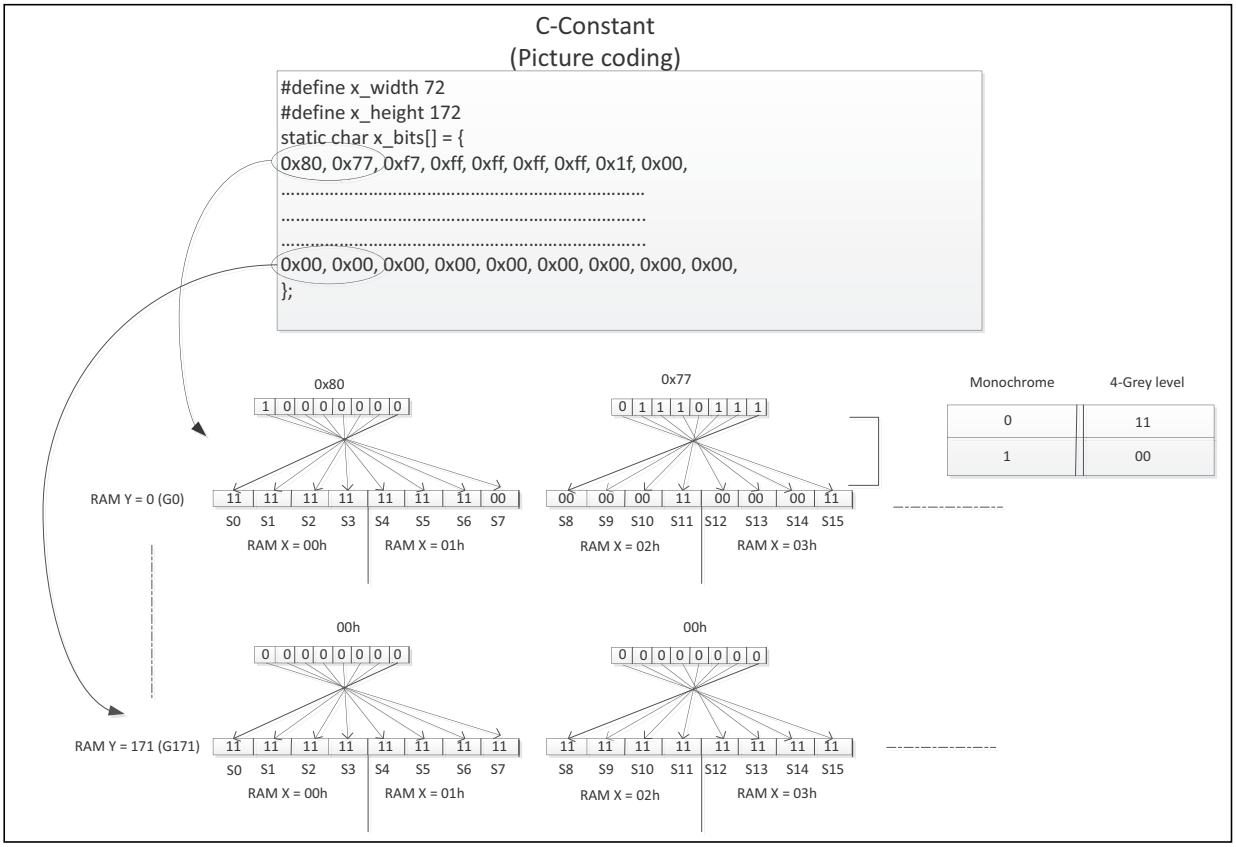
Using this picture formatting allows to reduce by 2 the size of each picture, meaning that the picture's weight is 1.5KB against 3KB in 4-grey level mode.

1.5 Picture data expansion to load the embedded E-Paper RAM

The software has been developed to read efficiently the constant generated into the xbm file and to display it. Depending on the application, there are different optimizations axis for the software and the E-Paper display configuration. Some of them are described in the *Section 2: Firmware description*.

Figure 8: Data expansion to load the E-Paper RAM illustrates the way the C-constant (coding the picture in monochrome 1-bit format) is processed to be loaded into the E-Paper RAM module.

Figure 8. Data expansion to load the E-Paper RAM



The software is minimizing the data processing time to perform this data expansion operation. If the data read from the C-Constant is read equal to 0, the software writes directly 2 bytes at FFh in the corresponding RAM Address pointed out by X and Y RAM address pointers. Indeed, the white level is coded 11b in 4-Grey Level in the RAM whereas it is coded 0b into the C-constant. Each byte coded into the C-Constant represents 8 pixels and so 2 RAM addresses, each of them coding 4 pixels in 4-grey levels.

In case of there is at least one pixel defined as black pixel from the C-Constant value, the software data processing is launched to extend the 1-bit monochrome coding into the 4-grey level coding like represented in *Figure 8: Data expansion to load the E-Paper RAM*.

X and Y RAM address pointers are managed by the hardware of the E-Paper display module each time a data is written in SPI mode when it is in data entry mode (for more details, please refer to the GDE021A1 E-Paper display specification).

2 Firmware description

The application note is based on a software (reference STSW-STM32152) which runs on a STM32L053 discovery kit. This section describes the main functions and software features to address the E-Paper display module and to manage picture size compression/ expansion to save memory space for pictures storage.

The software displays 4 pictures in rolling mode, with a temporisation of 5s between each display refresh cycle.

2.1 System configuration

The STM32L053 discovery kit has been configured to run at 16MHz from the internal HSI16 RC-oscillator.

The SPI is running at 2MHz to communicate with the E-Paper display module for command and data sending.

The systick is used to control delays between each picture display refresh, and to manage the E-Paper initialization.

2.2 Interrupt source

The only interruption source used in the software is the systick interrupt which allows to increment a counter to manage the HAL_Delay() function.

2.3 E-Paper power supply

The E-Paper power supply is switched ON thanks to the GPIO PB10 in the main.c file. Then it always remains powered in this application.

2.4 Major software functions description

High level software functions used by this application note are presented and described in the *Table 3: High level software functions*.

The CS (Chip Select) pin of the E-Paper is controlled by a MCU GPIO each time a display processing phase is launched. The HAL macro allows to control it directly:

EPD_CS_LOW() or EPD_CS_HIGH()

Table 3. High level software functions

Function name	Description
BSP_EPD_DrawImage(uint16_t Xpos, uint16_t Ypos, uint16_t Xsize, uint16_t Ysize, uint8_t *pdata)	This function calls the others high level functions needed to display one specific picture on the E-Paper display.
BSP_EPD_RefreshDisplay(void)	This function is used to launch the E-Paper refresh command.It waits for the BUSY signal deassertion from the E-Paper module.
EPD_RESET_HIGH() EPD_RESET_LOW()	This macro is used to generate the reset signal to the E-Paper display thanks to a MCU GPIO.
BSP_EPD_Init()	Configure the E-Paper: - Sleep mode - RAM X start/end address: 00h/11h - RAM Y start/end address: 00h/ABh - RAM X counter: 00h - RAM Y counter: 00h - Disable RAM bypass and GS0 to GS3 for the display transition - Display update: CLK ON + Charge Pump ON + pattern display

Only the most relevant low level functions are detailed below.

- **EPD_IO_WriteReg function**

  This function implements all the needed operations to send a command to the E-Paper display module.

Table 4. EPD_IO_WriteReg function

Function name	EPD_Write_Com
Prototype	void EPD_IO_WriteReg(uint8_t Reg)
Behavior description	Send the command to the E-Paper display module thanks to the SPI MOSI line
Input parameters	The command to send to the E-Paper display module.
Output parameter	None
Return value	None
Required preconditions	None
Called functions	SPIx_Write(), EPD_CS_LOW(), EPD_CS_HIGH(), EPD_DC_LOW()

This function is going to transmit a single byte identifying the command (for more details about the command values and their meaning, please refer to the GDE021A1 E-Paper display specification)

- **EPD_IO_WriteData function**

  This function implements all the needed operations to send a data to the E-Paper display module.

Table 5. EPD_IO_WriteData function

Function name	EPD_Write_Com
Prototype	void EPD_IO_WriteData(uint8_t RegValue)
Behavior description	Send the data to the E-Paper display module thanks to the SPI MOSI line
Input parameters	The data to send to the E-Paper display module for display configuration, or to load the RAM module with the picture to display.
Output parameter	None
Return value	None
Required preconditions	None
Called functions	SPIx_Write(), EPD_CS_LOW(), EPD_CS_HIGH(), EPD_DC_HIGH()

This function is going to transmit a single data byte which could be either a data used to configure the E-Paper register or a data corresponding to the picture to display (for more details about the command values and their meaning, please, refer to the GDE021A1 E-Paper display specification).

- **gde021a1_DrawImage function**

This function prepares the 2 expanded bytes data from the 8-bit data read from the 1-bit XBM files. It reorders the 2-bit pixels in the byte in a proper way to fill-in the RAM (refer to *Figure 8: Data expansion to load the E-Paper RAM*).

Table 6. Gde021a1_DrawImage function

Function name	Processing_8_pixels
Prototype	gde021a1_DrawImage(uint16_t Xpos, uint16_t Ypos, uint16_t Xsize, uint16_t Ysize, uint8_t *pdata)
Behavior description	2-bytes data preparation (expansion) to load the E-Paper RAM display and byte reordering to fill-in the RAM with the pixels information placed at the right place
Input parameters	Data to send to the E-Paper display module, the start and the end addresses of the matrix X/Y as well as the size of each of them.
Output parameter	None
Return value	None
Required preconditions	None
Called functions	EPD_IO_WriteReg(uint8_t Reg), EPD_IO_WriteData(uint8_t RegValue)

3 Possible firmware optimization

There are few axis to optimize the code and/or to reduce the power consumption.

3.1 E-Paper consumption management

In this application note, the software switches ON the power for the E-Paper permanently. Depending on the application, power consumption may be critical and the software may want to act for a drastic power consumption reduction. Two modes are offered by the E-Paper like described into *Table 7: E-Paper display module low power modes*.

Table 7. E-Paper display module low power modes

E-Paper low power modes	Consumption (typical) @3,3V	Contribution	Drawback
Sleep mode	35 uA	DC/DC converter OFF No more clock, No output load, RAM retention guaranty	The consumption
Deep sleep mode	2 uA	DC/DC converter OFF No more clock, No output load, No RAM retention	RAM is no more retained, it means that partial picture refresh is not possible after wake-up. The RAM content needs to be completely reloaded with the new picture to display.

Normally in most of the application, the E-Paper refresh rate should be long enough to prefer to let the E-Paper entering in deep sleep mode when it is not busy. It could cost less in term of power consumption to reload the RAM with the picture to display through the SPI (3096 bytes). In opposite, if the refresh rate is quite fast, it could be better to enter in sleep mode rather the deep sleep mode to maintain the RAM content to allow partial display area refresh with a minor runtime for the MCU. The user has to evaluate the best solution in term of power consumption and the time to refresh the picture according to his goals.

The STM32L053 discovery kit used to run this application note software allows to pilot a MOSFET from a dedicated GPIO (PB10) to switch OFF totally the power consumption, disconnecting the E-Paper main power supply. In such case, the screen for sure is still displaying the picture but the module doesn’t consume anymore current. It is a kind of deep sleep mode “plus”. In such case, the module may consume more when the application will switch it ON and when the charge pump will be reactivated.

3.2 Partial update of the E-Paper display RAM

The application may require time to time to refresh partially the display. In such case, the digital interface of the E-Paper display module may configure the X and Y RAM address pointers and the corresponding counters to load only the part of the display which changes.

The remaining data to display are still in the E-Paper RAM if the sleep mode is set during the E-Paper initialisation phase rather than the deep sleep mode (and the power supply maintained on the E-Paper). It has two effects, to minimize the number of bytes to transfer by the MCU to the E-Paper RAM and so to minimize the MCU/CPU runtime to do this operation.

4 Revision history

Table 8. Document revision history

Date	Revision	Changes
15-Oct-2014	1.0	Initial release