

Defence Lighting

Components – EMEA

Power

Management



ESC – Software Solutions



Quick Start Guide

©2017 by ARROW

All rights reserved. No part of this manual shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, desktop publishing, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this document, the publisher and author assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein. All terms mentioned in this manual that are known to be trademarks or service marks are listed below. In addition, terms suspected of being trademarks or service marks have been appropriately capitalized. ARROW cannot attest to the accuracy of this information. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Revision History

Revision , Date	Editor	Subject (major changes)
Revision 1.0, 10.04.2017	Quang Hai Nguyen	Initial release

Table of Contents

Revision History	3
Table of Contents	4
List of Figures	5
Introduction	6
Requirements for the demo	6
Running the demo	6
Creating the demo from scratch	6
Running the demo	7
Flashing the board with STM32 – ST LINK Utility	7
Running the Demo	9
Getting started from the scratch	11
Getting started with CubeMX	
Editing the code	

List of Figures

Figure 1: Folder structure	6
Figure 2: Hardware for running the demo	7
Figure 3: Connect the board to STM32 ST-LINK Utility	7
Figure 4: Loading the binary into the STM32 ST-LINK Utility	8
Figure 5: Programming the board with STM32 ST-LINK Utility	9
Figure 6: Terminal Program configuration	9
Figure 7: Running the demo 1	10
Figure 8: Running the demo 2	
Figure 9: Running the demo 3	10
Figure 10: Board selection for new project	11
Figure 11: Clearing the pinout	12
Figure 12: SPI configuration	12
Figure 13: TIM1 configuration	13
Figure 14: USART2 configuration	13
Figure 15: Setting the pins	13
Figure 16: Setting the clock	
Figure 17: Configuration Tab	14
Figure 18: Setting for SPI1	
Figure 19: Setting for USART 2	15
Figure 20: Setting for TIM1	16
Figure 21: Setting for NVIC	16
Figure 22: Project setting 1	17
Figure 23: Project setting 2	17
Figure 24: Generate project 1	
Figure 25: Generate Project 2	
Figure 26: Adding additional source file	
Figure 27: Include and define section	
Figure 28: Private variables section	19
Figure 29: Function prototype section	
Figure 30: Peripherals initialize	
Figure 31: Check ID and check bad blocks	20
Figure 32: Enable using printf	
Figure 33: CheckBadBlock function	
Figure 34: FlashTest function	
Figure 35: Call back function for push button	
Figure 36: Build and Download	23

Introduction

This document provides the information about the hardware and software requirements for running the demo. It also guides the user how to run the X-NUCLEO-SPINAND-TOSH demo and how to create one from scratch.

Folder structure

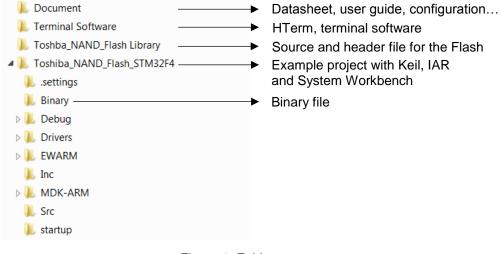


Figure 1: Folder structure

Requirements for the demo

Running the demo

- STM32F401 Nucleo kit
- X-NUCLEO-SPINAND-TOSH
- Mini USB cable
- STM32 ST LINK Utility
- Terminal Program (HTerm, TeraTerm, Putty...)

Creating the demo from scratch

- STM32CubeMX
- Preferred IDE (Keil, IAR, System Workbench)

Running the demo

Flashing the board with STM32 - ST LINK Utility

Plug the X_NUCLEO onto the STM32F4 Nucleo and power it with the mini USB cable

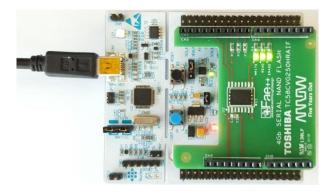
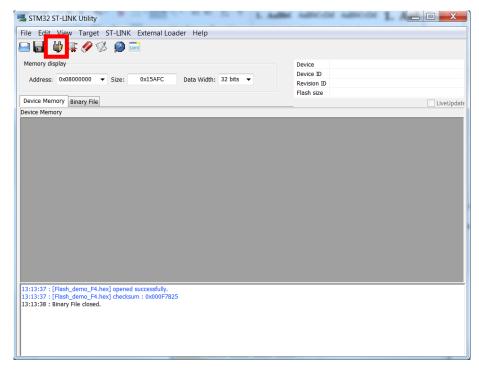
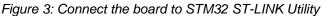


Figure 2: Hardware for running the demo

Open STM32 ST-Link Utility and connect it to the board by pressing the connect button or go to **Connect** \rightarrow **Target**.





Open the binary file by clicking **File** \rightarrow **Open File**... then navigate to the folder which the binary file is stored (Figure 1) or simply drag and drop the file into the program.

Binary find can be found in the Binary folder (Figure 1)

3 🖬 🖕	G 🔗 🖗	Swv						
Memory display		Size: 0x	2996743	Data Width:	32 bits 🔻	Device Device ID Revision ID	STM32F401xD/E 0x433	
Device Memory						Flash size		
Foshiba_NAND_ Address	0	4. nexj, Addres	8	C	ASCII			
0x08000000	20003E88	080001A9	080001B1	080001B3	^>. ©±	³		-
0x08000010	080001B5	080001B7	080001B9	00000000	μ [±]			
0x08000020	00000000	00000000	00000000	08001B19				
0x08000030	080001BD	00000000	08001A61	08001C3B	½a;.			
0x08000040	080001C3	080001C3	080001C3	080001C3	ÃÃÃÂ	ί		
x08000050	080001C3	080001C3	080001C3	080001C3	ÃÃÃÃ	ί		
0x08000060	080001C3	080001C3	080001C3	080001C3	ÃÃÃÃ	(
•		1						۴
0:20:23 : ST-L 0:20:23 : Conn			M14					
0:20:23 : SWD	Frequency = 1 ection mode : (00 KHz.	Basat					
0:20:23 : Com		Jonnect onder	Reset.					

Figure 4: Loading the binary into the STM32 ST-LINK Utility

Finally the board can be programmed by clicking **Target** \rightarrow **Program & Verify** \rightarrow **Start** or clicking on the Program Verify button, then press start.

When the program is finished, the console will show the complete message as below and the board is ready.

Memory display Address: 0x0	08000000 👻	Size: 0	x24E8	Data Width:	32 bits 💌	Device Device ID Revision ID Flash size	
Device Memory Target memory,				ash_STM32F4.	hex		📃 LiveUpdat
Address	0	4	8	с	ASCII		
0x08000000	20003E88	080001A9	080001B1	080001B3	^ >. ©± ³		=
0x08000010	080001B5	080001B7	080001B9	00000000	μ		
0x08000020	00000000	00000000	00000000	08001B19			
0x08000030	080001BD	00000000	08001A61	08001C3B	½a;		
0x08000040	080001C3	080001C3	080001C3	080001C3	ÃÃÃ		
0x08000050	080001C3	080001C3	080001C3	080001C3	ÃÃÃ		
0x08000060	080001C3	080001C3	080001C3	080001C3	ÃÃÃ		
0x08000070	080001C3	080001C3	080001C3	080001C3	ÃÃÃ		
0x08000080	080001C3	080001C3	080001C3	00000000	ÃÃÃ		
0x08000090	00000000	00000000	00000000	080001C3	Ã		
•	1				~ ~ ~ ~		*
15:07:59 : SWD							
15:07:59 : Conn 15:07:59 : Devic	e ID:0x433		Reset.				
15:07:59 : Devic	e flash Size : 5 e family :STM3						

Figure 5: Programming the board with STM32 ST-LINK Utility

Running the Demo

Open the preferred Terminal program. HTerm is used in this document and can be found in the Terminal Software folder. Configure it as following (*Note that the port is different on your computer*):

b HTerm 0.8.1beta	
ile Options View Help	
Connect Port COM75 R Baud 115200 Data 8 Stop 1 Parity None CTS Flow control	
Rx 0 Reset Tx 0 Reset Count 0 🔹 0 Reset Newline at LF 🕶 Show newline characters	
	lewline after ms 0 🕞 CTS DSR RI DCD eceive pause (0=off) 0
equence Overview X Received Data	

Figure 6: Terminal Program configuration

By pressing **Connect** button on HTerm and the **reset button** on the controller (black one), there will be a message displayed to show the address of the flash, also running the process of checking bad blocks. Please note that, every memory is shipped with some bad blocks, and the bad blocks are different from one memory to the others.

In this example, bad blocks occur at block #3, #1536, #1537, #1561, and #1789.

Received Data													
1 5 10 15 20 25 Flash ID: 39117 w Checking for Bad Block w Bad block at block #1536 w Bad block at block #1537 w Bad block at block #1561 w Bad block at block #1561 w Bad block at block #1789 w Total of bad blocks: 5 w Please press user button to	30 35 start the	40 45 9 flash t	55	60	65	70	75	80	85	90	95	100	Ш

Figure 7: Running the demo 1

Then press the **blue button** to continue the demo. The controller will check if the current block is bad block, then it erases the data inside the block and write new data to the first page of each block.

Received Data	
1 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 10	• 0
Suceesfully writing to first page of block #2033w	
Suceesfully writing to first page of block #2034m	
Successfully writing to first page of block $#2035_{M}$	
Suceesfully writing to first page of block #2036m	
Succesfully writing to first page of block $\#2037_{V_{R}}$	
Suceesfully writing to first page of block #2038v	
Succesfully writing to first page of block $#2039_{M}$	
Succesfully writing to first page of block $\pm 2040_{ m m}$	
Successfully writing to first page of block $\pm 2041_{ m m}$	
Successfully writing to first page of block $\pm 2042_{ m in}$	
Succesfully writing to first page of block $#2043_{M}$	
Successfully writing to first page of block $\pm 2044_{ m m}$	
Succesfully writing to first page of block $\#2045_{Vn}$	
Succesfully writing to first page of block $#2046_{M}$	
Successfully writing to first page of block $\pm 2047_{ m m}$	
Finish!!! \m	
	E
Selection (-) Timestamp: 09:30:42	.675.5

Figure 8: Running the demo 2

Bad blocks at block number 3, 1536, 1537, 1561, 1789 are reported again during the run so they are not overwritten.

Block #3: bad block. Move to next one \n Block #1536: bad block. Move to next one \n Block #1537: bad block. Move to next one \n Block #1561: bad block. Move to next one \n Block #1789: bad block. Move to next one \n Figure 9: Running the demo 3

Getting started from the scratch

Getting started with CubeMX

Open CubeMX and choose New Project.

In the new project window, go to Board Selector tab. In the field **Type of Board**, **Nucleo64** is chosen and in the **MCU Series STM32F4** is picked. Finally in the **Board Lists field**, **NUCLEO-F401RE** is chosen.

Peripherals/Connectors N Max Type Accelerometer Im N/A Nuclea Analog I/O 0 0 Nuclea Audio Line În N/A Nuclea Audio Line În N/A Nuclea Button 0 1 Camera N/A N/A Oigital I/O 0 76 Digital I/O 0 76 Echernet N/A Ethanet N/A Flash Memory 0 Cyroscope N/A	STM32F4 st: 4 Items Reference NUCLEO-F401RE NUCLEO-F411RE NUCLEO-F416RE NUCLEO-F410RB	MCU STM32F401RETX STM32F411RETX STM32F446RETX STM32F410RBTX
eripherals/Connectors Selection Board Peripherals/Connectors Nb Max Accelerometer NAA Accelerometer NAA Audio Line in NAA Audio Line in NAA Button 0 1 Carmera NAA Corngass NAA Corngass NAA Ethermet NAA Ethermet NAA Ethermet NAA Ethermet NAA Ethermet NAA Ethermet NAA Ethermet NAA Ethermet NAA	Reference NUCLEO-F401RE NUCLEO-F411RE NUCLEO-F446RE	MCU STM32F401RETx STM32F411RETx STM32F446RETx
Peripherals/Connectors N Max Type Accelerometer In N/A Nuclear Analog I/O 0 0 Nuclear Audio Line Out N/A Nuclear Nuclear Audio Line Out N/A Nuclear Nuclear Gutton 0 1 N/A Nuclear Opigital J/O 0 76 Eprom N/A Experiment N/A N/A N/A N/A Pash Memory 0 0 0 70/A	Reference NUCLEO-F401RE NUCLEO-F411RE NUCLEO-F446RE	STM32F401RETx STM32F411RETx STM32F446RETx
Accelerometer IVA Nuclei Analog V/O 0 0 Nuclei Analog V/O 0 0 Nuclei Audio Line Out IVA Nuclei Sution 0 1 CAN 0 0 Campass IVA N/A Digital V/O 0 76 Esprom IVA N/A Ethernet IVA JOA IVA JOA IVA	NUCLEO-F411RE NUCLEO-F446RE	STM32F411RETx STM32F446RETx
Analog V/O 0 0 0 Nucle Audio Line In N/A N/A Nucle Button 0 1 N/A Button 0 0 0 Camera 1 N/A Compass N/A 0 Oigital I/O 0 76 Ebrornet N/A N/A Flash Memory 0 0 OrDA 1 N/A	NUCLEO-F411RE NUCLEO-F446RE	STM32F411RETx STM32F446RETx
Audio Line In In/A Nuclei Audio Line Out In/A Nuclei Button 0 1 CAM 0 0 Camera N/A Compass N/A Oigital 10/0 0 Ethernet N/A Flash Memory 0 IrOA N/A	NUCLEO-F446RE	STM32F446RETx
Judio Line Out N/A Button 0 CAN 0 Camera N/A Objetal I/O 76 Digital I/O 76 Eprom N/A Ethernet N/A Fish Memory 0 Grossope N/A		
Button 0 1 CAN 0 0 Camera N/A Compass N/A Comprom N/A Digital I/O 0 Eeprom N/A Ethermet N/A Flash Memory 0 Großocope N/A	, nocito i navio	
CAN 0 0 Camera N/A Compass N/A Digital I/O 76 Eprom N/A Ethermet N/A Flash Memory 0 Syroscope N/A IrbA NA		
Camera N/A Compass N/A Jojatal I/O 0 Jeprom N/A Eprom N/A Flash Memory 0 Groscope N/A IrbA NA		
Compass N/A Objetal I/O 0 76 Exprom N/A N/A Ethernet N/A N/A Flash Memory 0 0 Gyroscope N/A N/A IrbA N/A N/A		
Digital I/O 0 76 Eeprom N/A Ethernet N/A Flash Memory 0 Gyroscope N/A IrloA N/A		
Eeprom N/A Ethernet N/A Flash Memory 0 Gyroscope N/A IrDA N/A		
Ethernet N/A Flash Memory 0 0 Gyroscope N/A IrDA N/A		
Flash Memory 0 0 Gyroscope I N/A IrDA I N/A		
Gyroscope N/A		
IrDA N/A		
Joystick N/A		
Lcd Display (Graphics)		
Lcd Display (Segemnt)		
Led 0 1		
Light Sensor		
Memory Card		
Memory Card N/A		
Potentiometer N/A		
Pressure Sensor N/A		
RS-232 0 0		
RS-485		
SRAM/SDRAM 0 0		
Speaker 0 0		
Temperature Sensor		
Touch Key Sensing N/A		
Motor control connector		

Figure 10: Board selection for new project

By pressing OK, a new window appears for project configuration. First thing need to be done is cleaning the pinout. Choosing the option **Pinout** \rightarrow **Clear Pinouts**

• STM32CubeN	1X Untitled: STM32F401RETx N	UCLEO-F4	401RE	AA	A state	Paston Wag 12 Institution Inc.	
File Project Pin	out Window Help						
🖪 🗁 🖿 🤊	Undo	Ctrl+Z	hent 🤊 🤉 🔙 🗕 🥝 🔶 Find	-	🔍 🔍 🔍 📝 Show user Label 🛛 🦻 🎐		
Pinout Clock	Redo	Ctrl+Y	Calculator				
Configuration MiddleWa B • FAT	Clear Pinouts Clear Single Mapped Signals						
🗄 💿 FRE	Set unused GPIOs	Ctrl+G					
B- O USB	Reset used GPIOs	Alt+G					
⊕-● USB ⊖-Periphera ⊕-1 ADC	Generate CSV pinout text file Pins/Signals Options	Ctrl+O	•			ě	
⊕ © CRC → ⊕ © 12C ⊘ ⊕ © 12C + ⊕ © 12C	Collapse All Disable Modes Expand All	Alt+E Ctrl+D Alt+X		VB/ B1 (Blue PushButton)	-10	VDD VSS	
• 125 • 125	Best fit	Alt+I Alt+B Alt+O		RCC_OSC32_IN PC RCC_OSC32_OUT PC RCC_OSC_IN PH RCC_OSC_OUT PH		PA13 PA12 PA11 PA10	
RTC SDI0 SPI1 SPI2 SPI3				kar Por Por Por Por	STATZ	PA5 PA8 PC5 PC5 PC7	
				VSS VRE PAI		PC5 P815 P814 P813 P813	
				USART_TX PA	WCA PB10 PB2 PB1 PB0 PC4 PC4 PC4 PA5 PA5 PA5 PA5 PA3		
• • TIM10 • • TIM11 • • USART1					LD2 [Green Led]		
B 💿 USART2					-		
USART6		*					

Figure 11: Clearing the pinout

In the **Pinout** tab, go to **SPI1**, we configure *Mode* as *Full-Duplex Master*, and *Hardware NSS Signal* as *Disable*

🗄 🚹 SDIO	
🖨 💿 SPI1	
Mode Full-Duplex Master	•
Hardware NSS Signal Disable	•
🗄 💿 SPI2	
🖶 💿 SPI3	

Figure 12: SPI configuration

In the **TIM1** configuration, we set *Clock Source* as *Internal Clock* to activate the General purpose timer 1

🗄 🚹 SYS							
🖨 🔥 ТІМ1							
Slave Mod	le Dis	able 💌					
<mark>Trigger So</mark>	ource	Disable 💌					
Clock Sou	rce	Internal Clock					
Channel1	Disab	Disable					
Channel2	Disab	Disable 💌					
Channel3	Disab	Disable 💌					
Channel4	Disab	Disable 🔹					
Combined	Combined Channels Disable						
Activa	Activate-Break-Input						
····· 📃 Use E	Use ETR as Clearing Source						
🔲 XOR a	activat	ion					
🔲 One P	ulse N	Node					
🖽 🚹 ТІМ2							
🗄 💿 ТІМЗ							

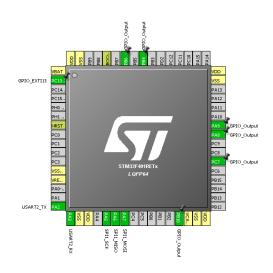
Figure 13: TIM1 configuration

Next, we activate UART by going to **USART2** configuration and set *Mode* as *Asynchronous*, and *Hardware Flow Control (RS232)* as *Disable*



Figure 14: USART2 configuration

In the window on the right, configure pins *PB6, PB4, PA9, PA8, PC7*, and *PB10* as *GPI0_Output* and *PC13* as *GPI0_EXT13*





Move to Clock Configuration tab and make sure that the clock is set to 84MHz

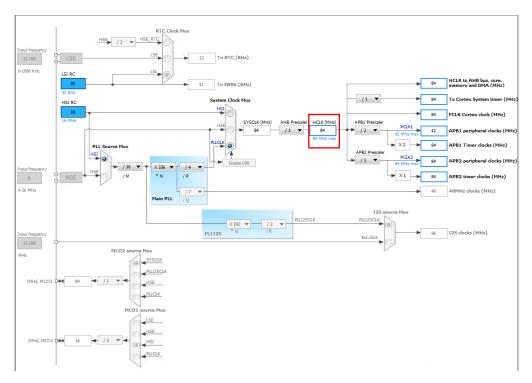


Figure 16: Setting the clock

Move to **Configuration** tab. If we have configured everything correct, we should have the picture as below

		Middlewares		
Multimedia	Connectivity	Analog	System	Control
	SPIL 2			TIM1 ⊘

Figure 17: Configuration Tab

We start to configure the SPI by clicking in the tab **SPI1**. The setting in **SPI1** tab is configured as follow

🖉 Parameter Settings 🗹 User Constants 🔹	NVIC Settings 🖋 DMA Settings 🖋 GPI) Settings
onfigure the below parameters :	······································	· · · · · · · · · · · · · · · · ·
Search : Search (Crtl+F)	* *	
Basic Parameters		
Frame Format	Motorola	
Data Size	8 Bits	
First Bit	MSB First	
Clock Parameters		
Prescaler (for Baud Rate)	8	
Clock Polarity (CPOL)	High	
Clock Phase (CPHA)	2 Edge	
Advanced Parameters		
CRC Calculation	Disabled	
NSS Signal Type	Software	

Figure 18: Setting for SPI1

Then the **USART2** is configured as followed:

✓ Parameter Settings ✓ User Consti Configure the below parameters :	ants 🖋 NVIC Settings 🖋 DMA Settings 🖋 GPIO Se	ettings
Search : Search (Crtl+F)	• •	
Basic Parameters		
Baud Rate	115200 Bits/s	
Word Length	8 Bits (including Parity)	
Parity	Odd	
Stop Bits	1	
Advanced Parameters		
Data Direction	Receive and Transmit	
Over Sampling	16 Samples	

Figure 19: Setting for USART 2

Here is the configuration for the timer:

TIM1 Configuration	X
🖋 Parameter Settings 🗹 User Constants 🖋 NVIC	Settings 🦪 DMA Settings
Configure the below parameters :	
Search : Search (Crtl+F)	•
Counter Settings	
Prescaler (PSC - 16 bits value)	84
Counter Mode	Up
Counter Period (AutoReload Register - 16	pit 0xFFFF
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits value)	0
Trigger Output (TRGO) Parameters	
Master/Slave Mode	Disable (no sync between this TIM (Master) and its Sla
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
	Apply Ok Cancel

Figure 20: Setting for TIM1

Finally, we go to **NVIC** tab to enable the interrupt for the push button.

Priority Group 0 bits for pre-emption priority 4	bits f 🔻 🔲 S	Sort by Premption Prior	ity and Sub Pro	rity
Search (Crtl+F)		Show only enabled inte	errupts	
Interrupt Table	Enabled	Preemption Priority	Sub Priority	
Non maskable interrupt	1	0	0	
Hard fault interrupt	1	0	0	
Memory management fault	1	0	0	
Pre-fetch fault, memory access fault	1	0	0	
Undefined instruction or illegal state	1	0	0	
System service call via SWI instruction		0	0	
Debug monitor		0	0	
Pendable request for system service		0	0	
Time base: System tick timer		0	0	
PVD interrupt through EXTI line 16		0	0	
Flash global interrupt		0	0	
RCC global interrupt		0	0	
TIM1 break interrupt and TIM9 global interrupt		0	0	
TIM1 update interrupt and TIM10 global interrupt		0	0	
TIM1 trigger and commutation interrupts and TI	M11 gl	0	0	
TIM1 capture compare interrupt		0	0	
SPI1 global interrupt		0	0	
UCADT2 global interrupt		10	-	
EXTI line[15:10] interrupts	1	0	0	
n o giobar interrupt		19 19	0	
i ro gioda interrupt		0	U	
				Ŧ
Enabled Pre	emption Priority	Sub Priority	¥	
	, , ,	Apply Ok	Cancel	

Figure 21: Setting for NVIC

Then we go to **Project** → **Setting** to configure the final setting for our project before generating it.

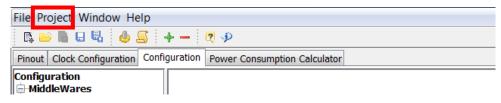


Figure 22: Project setting 1

In the **Project Setting** window, we name our project, choose a location to place it and, the most important, choose the Toolchain/IDE to writing the code (in this case, Keil is used).

oject Code Generator A	avanced Settings
Project Settings	
Project Name	
Flash_demo_F4	
Project Location	
C:\MyFiles\SoftwareTea	m\Technology\Flash -Arrow\X-Nucleo\Projects
Toolchain Folder Location	•
C:\MyFiles\SoftwareTea	m\Technology\Flash -Arrow\X-Nucleo\Projects\Flash_demo_F4
Toolchain / IDE	
MDK-ARM V5	Generate Under Root
Linker Settings Minimum Heap Size Minimum Stack Size	0x200 0x400
Mcu and Firmware Packa	ge
Mcu Reference STM32F401RETx	
Firmware Package Name	
STM32Cube FW_F4 V1.1	13.1

Figure 23: Project setting 2

Press Ok to close the window, then the project can be generated by choosing **Project** -> Generate Code or clicking on the cog wheel button

File Project Window	n Help
🕼 🐸 📗 🖯 🖏	🎂 😼 🛨 — 🔍 🌮
Pinout Clock Configur	ation Configuration Power Consumption Calculator
Configuration	

Figure 24: Generate project 1

After the project is successfully generated, a window will pop up to ask for further action. From here we can open the project in preferred IDE

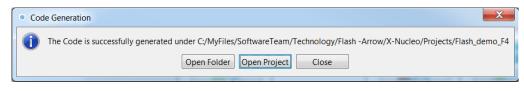


Figure 25: Generate Project 2

Editing the code

Keil is used in this guide but it can be easily tailored to other IDE.

First thing we do is adding all the necessary source file into the project, which is *TC58_FPP_CMD.c* in this case. Please also remember to add the header files to the *Inc folder*. All the related source and header files can be found in the Toshiba_NAND_Flash Library folder.

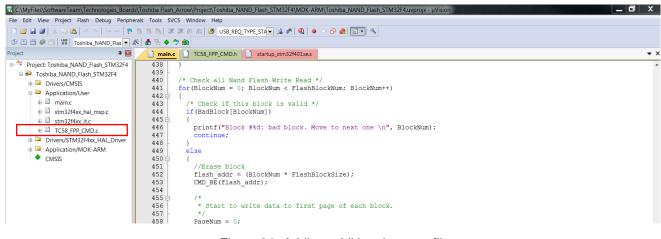


Figure 26: Adding additional source file

First step is done, we go to the main.c and enter these lines between the /* USER CODE BEGIN Includes */ and /* USER CODE END Includes */

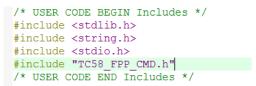


Figure 27: Include and define section

Those codes include all the headers that needed. Next, we add all the needed variables to the USER CODE BEGIN PV section.

```
/* USER CODE BEGIN PV */
/* Private variables --
/* Enable printf on Keil and IAR*/
#ifdef __GNUC______idefine PUTCHAR_PROTOTYPE int ____io_putchar(int ch)
           GNUC
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC_
#define TRANS_BAD_LENGTH 4224
#define FLASH_TARGET_ADDR 0x00000000
#define Error inc(x) x = x + 1;
uint8_t st_reg1 = 0;
uint8_t memory_addr[TRANS_BAD_LENGTH];
uint8_t memory_addr_cmp[TRANS_BAD_LENGTH];
uint8_t memory_addr_t[TRANS_BAD_LENGTH];
uint8_t memory_addr_t_cmp[TRANS_BAD_LENGTH];
uint8_t pTxData[16];
uint8 t pRxData[16];
uint8_t BadBlock[FlashBlockNum];
char buff[50];
uint16_t BlockNum;
uint16_t PageNum;
uint16_t i=0;
uint16_t col_address = 0;
uint16_t BadBlockTot = 0;
uint16_t BadBlockCot;
uint16 t flash id = 0;
uint16 error_cnt = 0;
uint32_t PageAddress;
uint32_t BlockAddress;
uint32_t FlashAddressTest;
uint32_t FlashAddressShift1;
uint32_t FlashAddressShift2;
uint32_t flash_addr;
HAL_StatusTypeDef status = HAL OK;
ReturnMsg message = Flash Success;
/* USER CODE END PV */
```

Figure 28: Private variables section

Then we add the code for the function prototype to the USER CODE BEGIN PFP section. The names of the functions self-explaining their purposes. *FlashTest* tests the operation of the flash memory and *SearchBadBlock* looks for the bad block in the flash.

```
/* USER CODE BEGIN PFP */
/* Private function prototypes ------*/
void SearchBadBlock(void);
void FlashTest(void);
/* USER CODE END PFP */
```

Figure 29: Function prototype section

Inside the main function, we navigate to the USER CODE BEGIN 2 section and add this block.

/* USER CODE BEGIN 2 */
__HAL_DBGMCU_FREEZE_TIM1();
HAL_TIM_Base_Start_IT(&htim1);
Initial_Spi();

Figure 30: Peripherals initialize

The block above initializes the timer and SPI and also starts the timer. After that, we add the following code to the application. This block reads the address of the flash memory and compares it with the pre-defined one. If there is a mismatch, the program will print the error message and go to the *Error_Handler* function, which is an infinite loop. If there is no error, it will run the *SearchBadBlock* function.

```
/* Read flash id */
CMD_RDID((uint16*)&flash_id );
/* Read flash id */
CMD RDID((uint16*)&flash id );
/* Compare to expected value */
if( flash_id != FlashID )
-{
 Error inc(error_cnt);
 printf("Wrong Flash ID \n");
 Error_Handler();
printf("Flash ID: %d \n", flash id);
if(!error_cnt)
{
 printf("Checking for Bad Block \n");
 SearchBadBlock();
 printf("Total of bad blocks: %d \n", BadBlockTot);
}
printf("Please press user button to start the flash test n");
/* USER CODE END 2 */
```

Figure 31: Check ID and check bad blocks

Scrolling down to the USER CODE BEGIN 4, we add this block to the program. This block allows us to use the *printf* function to write the data to the UART communication.

```
/* USER CODE BEGIN 4 */
/* Enable printf on System Workbench*/
int _write(int file, char *ptr, int len)
{
    HAL_UART_Transmit(&huart2, (uint8_t *)ptr, len, 10);
    return len;
}
/* Enable printf on IAR or Keil*/
PUTCHAR_PROTOTYPE
{
    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
    return ch;
}
```

Figure 32: Enable using printf

Next thing we add is the body of the function *SearchBadBlock*. According to the datasheet, information about the bad block is stored in column 0 or 4096 of the first and second page of the block. Therefore, this function scans through the column 4096 of the first page of each block and checks if they are equal to 0xFF. If it is not equal to 0xFF then we have a bad block.

```
void SearchBadBlock(void)
{
  BadBlockCnt = 0;
  for(BadBlockCnt = 0; BadBlockCnt < FlashBlockNum; BadBlockCnt++)</pre>
  {
    /* Check First Byte Spare Area of Page0 Block N */
flash_addr = BadBlockCnt * FlashBlockSize;
    FlashAddressTest = (BadBlockCnt << 6);</pre>
    FlashAddressShift1 = (flash_addr >> 12);
#if 0
    if(flash addr \geq 0 \times 10000000)
      BadBlock[BadBlockCnt] = 0;
    1
#endif
    BadBlock[BadBlockCnt] = 0;
    /* Read flash memory data to memory buffer */
    message = CMD_READ( flash_addr );
    if(message != Flash_Success)
    {
      while(1)
        printf("Read error \n");
    }
    col address = 0;
    memset(memory_addr, 0, TRANS_BAD_LENGTH);
    message = CMD_READ_CACHE( col_address, memory_addr, TRANS_BAD_LENGTH, 0 );
    if (message != Flash_Success)
    {
      while(1);
    }
    if(memory_addr[FlashPageSize + 1] != 0xFF)
    {
      BadBlockTot++;
      BadBlock[BadBlockCnt] = 1;
      printf("Bad block at block #%d \n", BadBlockCnt);
      continue;
    }
 }
}
```



After that we add the body of the function *FlashTest*. The main task of this function is erasing the block, writing data into the first page of the first block, then reading the data again and comparing them with the data it has written before to see if there is a match.

```
void FlashTest(void)
{
  /* Clear the block protection bit*/
 CMD_GET_FEATURE( 0xa0, &st_reg1 );
 if (st_reg1 & 0x38)
 {
   CMD_SET_FEATURE( 0xa0, (st_reg1&0x87) );
 }
 /* Check all Nand Flash Write Read */
 for(BlockNum = 0; BlockNum < FlashBlockNum; BlockNum++)</pre>
  {
    /* Check if this block is valid */
   if (BadBlock [BlockNum])
    {
     printf("Block #%d: bad block. Move to next one \n", BlockNum);
      continue;
    else
    {
      //Erase block
      flash_addr = (BlockNum * FlashBlockSize);
      CMD_BE(flash_addr);
      * Start to write data to first page of each block.
      */
      PageNum = 0;
      col_address = 0;
      flash_addr = (BlockNum * FlashBlockSize) + (PageNum * FlashPageSize);
      /* Read flash memory data to memory buffer */
      CMD_READ(flash_addr);
      CMD_READ_CACHE(col_address, memory_addr_t_cmp, FlashPageSize, 0);
      if(memory_addr_t_cmp[0] != 0xFF)
      {
        memory_addr_t_cmp[0] = 0;
      }
      /* Write one page at time */
memory_addr_t[0] = (BlockNum & 0xFF);
      memory_addr_t[1] = ((BlockNum >> 8) & OxFF);
      memory_addr_t[2] = PageNum;
      memory_addr_t[3] = 0x0F;
      for(i = 4; i < FlashPageSize; i+=2)</pre>
      {
        memory_addr_t[i] = (i & 0xFF);
                                                  /* generate sequential byte data */
        memory_addr_t[i+1] = ((i >> 8) & 0xFF);  /* generate sequential byte data */
      1
      /* Program data to flash memory */
      CMD_PP_LOAD(col_address, memory_addr_t, FlashPageSize, 0);
      CMD_PROGRAM_EXEC(flash_addr);
      /* Read flash memory data to memory buffer */
      CMD_READ(flash_addr);
      CMD_READ_CACHE(col_address, memory_addr_t_cmp, FlashPageSize, 0);
        * Compare Data Write with data Readed */
      if(memcmp(memory_addr_t, memory_addr_t_cmp, FlashPageSize) != 0)
      {
        Error_inc( error_cnt );
        printf("Error writing to first page of block #%d\n", BlockNum);
      3
      else
        printf("Succesfully writing to first page of block #%d\n", BlockNum);
      }
    }
  printf("Finish!!! \n");
```

Figure 34: FlashTest function

From the code block above, the first thing the function *FlashTest* does is clearing the protection bit. Then it will run through block by block, check if the block is bad block. If current block is not a bad block, the function will generate random data, write it to the first page of the block, read the data again and compare with the original one. If the data is matching, the function will move to the next block. If the data is not matching, it will prompt an error message.

Finally we add the call back function for the push button. Whenever the button is pressed, the call back function calls the *FlashTest*.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
}{
FlashTest();
}
Figure 35: Call back function for push button
```

Now we can build our program, download it into the board and our application is ready to run.

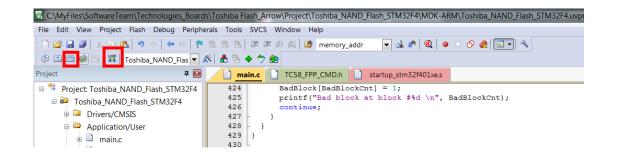


Figure 36: Build and Download

THE END