ChatGPT Refrences Taken

[Skip to content](https://chatgpt.com/?utm_source=google&utm_medium=paidsearch_brand&utm_campaign=DEPT_SEM_Google_Brand_Acquisition_APAC_India_Consumer_CPA_BAU_Mix_English&utm_term=chatgpt&gad_source=1&gad_campaignid=22370388714&gclid=CjwKCAjw3MXBBhAzEiwA0vLXQSI4vNIwL6utrRLTiGI8mefUu62gsJheTgqLGhJJghObjwBdk93MUhoCY5IQAvD_BwE" \l "main)

##### You said:

bit banging code for AT24C512C for stm32f041re

###### ChatGPT said:

Here's a basic bit-banging implementation in C to interface with the AT24C512C EEPROM using STM32F041RE, without using the I2C peripheral. This implementation manually toggles GPIO pins to simulate I2C (SCL and SDA lines).

### ✅ Assumptions:

* You are using STM32 HAL library.
* You have two GPIO pins assigned as:
  + SCL (Clock line)
  + SDA (Data line)
* EEPROM is operating in standard I2C mode (7-bit addressing).
* 512Kb EEPROM, i.e. 64KB of data.

## 📌 Pin Setup Example

// Define SCL and SDA pins (change as per your setup)

#define I2C\_GPIO\_PORT GPIOB

#define I2C\_SCL\_PIN GPIO\_PIN\_6

#define I2C\_SDA\_PIN GPIO\_PIN\_7

## 📌 GPIO Helper Functions

void I2C\_Delay(void) {

for (volatile int i = 0; i < 10; i++); // adjust for timing

}

void SDA\_Output(void) {

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

GPIO\_InitStruct.Pin = I2C\_SDA\_PIN;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_OD;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(I2C\_GPIO\_PORT, &GPIO\_InitStruct);

}

void SDA\_Input(void) {

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

GPIO\_InitStruct.Pin = I2C\_SDA\_PIN;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(I2C\_GPIO\_PORT, &GPIO\_InitStruct);

}

void I2C\_Start(void) {

SDA\_Output();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, GPIO\_PIN\_RESET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

}

void I2C\_Stop(void) {

SDA\_Output();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, GPIO\_PIN\_SET);

}

uint8\_t I2C\_WriteByte(uint8\_t byte) {

SDA\_Output();

for (int i = 0; i < 8; i++) {

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, (byte & 0x80) ? GPIO\_PIN\_SET : GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

byte <<= 1;

}

SDA\_Input();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

uint8\_t ack = HAL\_GPIO\_ReadPin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

return ack == 0;

}

uint8\_t I2C\_ReadByte(uint8\_t ack) {

uint8\_t byte = 0;

SDA\_Input();

for (int i = 0; i < 8; i++) {

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

byte = (byte << 1) | HAL\_GPIO\_ReadPin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

I2C\_Delay();

}

SDA\_Output();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, ack ? GPIO\_PIN\_RESET : GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

return byte;

}

## 📌 AT24C512C EEPROM Read/Write Functions

### EEPROM Addressing:

AT24C512C uses 16-bit word addressing.

### Write Byte:

uint8\_t EEPROM\_WriteByte(uint16\_t mem\_address, uint8\_t data) {

I2C\_Start();

if (!I2C\_WriteByte(0xA0)) return 0; // Device write address

if (!I2C\_WriteByte((mem\_address >> 8) & 0xFF)) return 0; // MSB

if (!I2C\_WriteByte(mem\_address & 0xFF)) return 0; // LSB

if (!I2C\_WriteByte(data)) return 0;

I2C\_Stop();

HAL\_Delay(5); // EEPROM write cycle time

return 1;

}

### Read Byte:

uint8\_t EEPROM\_ReadByte(uint16\_t mem\_address, uint8\_t \*data) {

I2C\_Start();

if (!I2C\_WriteByte(0xA0)) return 0;

if (!I2C\_WriteByte((mem\_address >> 8) & 0xFF)) return 0;

if (!I2C\_WriteByte(mem\_address & 0xFF)) return 0;

I2C\_Start(); // Repeated start

if (!I2C\_WriteByte(0xA1)) return 0; // Device read address

\*data = I2C\_ReadByte(0); // No ACK on last byte

I2C\_Stop();

return 1;

}

### ✅ Notes:

* This is a minimal implementation; you may want to add error handling, retries, and optimize delays.
* You must configure your GPIOs in main() or CubeMX before using them.

Would you like a full example project with main.c and CubeMX pin setup?

##### You said:

yes

###### ChatGPT said:

Certainly! Here's a step-by-step guide to setting up a bit-banging I2C interface between the STM32F041RE microcontroller and the AT24C512C EEPROM using STM32CubeMX and STM32CubeIDE.

## 🛠 Hardware Setup

### **STM32F041RE GPIO Pinout**

For this example, we'll use the following GPIO pins:

* SCL (Clock): PB6
* SDA (Data): PB7

Ensure these pins are connected to the corresponding pins on the AT24C512C EEPROM:

* SCL: Pin 6
* SDA: Pin 5
* VCC: 3.3V
* GND: Ground

Note: The AT24C512C EEPROM operates at 3.3V logic levels, which is compatible with the STM32F041RE.

## 🧰 STM32CubeMX Configuration

1. Create a New Project:
   * Open STM32CubeMX.
   * Select the STM32F041RE microcontroller.
2. Configure GPIO Pins:
   * Navigate to the Pinout & Configuration tab.
   * Set PB6 and PB7 as GPIO\_Output.
3. Configure GPIO Settings:
   * Go to the Configuration tab.
   * For PB6 and PB7, set:
     + Mode: GPIO\_MODE\_OUTPUT\_OD (Open Drain)
     + Pull-up/Pull-down: GPIO\_NOPULL
     + Speed: GPIO\_SPEED\_FREQ\_LOW
4. Generate Code:
   * Go to the Project tab.
   * Set the project name and toolchain (e.g., STM32CubeIDE).
   * Click Generate Code.

## 🧪 STM32CubeIDE Code Implementation

1. Open the Project:
   * Launch STM32CubeIDE.
   * Open the project generated by STM32CubeMX.
2. Implement Bit-Banging Functions:

In main.c, implement the following functions:

#include "main.h"

#define I2C\_SCL\_PIN GPIO\_PIN\_6

#define I2C\_SDA\_PIN GPIO\_PIN\_7

#define I2C\_GPIO\_PORT GPIOB

void I2C\_Delay(void) {

for (volatile int i = 0; i < 10; i++);

}

void SDA\_Output(void) {

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

GPIO\_InitStruct.Pin = I2C\_SDA\_PIN;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_OD;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(I2C\_GPIO\_PORT, &GPIO\_InitStruct);

}

void SDA\_Input(void) {

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

GPIO\_InitStruct.Pin = I2C\_SDA\_PIN;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

HAL\_GPIO\_Init(I2C\_GPIO\_PORT, &GPIO\_InitStruct);

}

void I2C\_Start(void) {

SDA\_Output();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, GPIO\_PIN\_RESET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

}

void I2C\_Stop(void) {

SDA\_Output();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, GPIO\_PIN\_SET);

}

uint8\_t I2C\_WriteByte(uint8\_t byte) {

SDA\_Output();

for (int i = 0; i < 8; i++) {

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, (byte & 0x80) ? GPIO\_PIN\_SET : GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

byte <<= 1;

}

SDA\_Input();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

uint8\_t ack = HAL\_GPIO\_ReadPin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

return ack == 0;

}

uint8\_t I2C\_ReadByte(uint8\_t ack) {

uint8\_t byte = 0;

SDA\_Input();

for (int i = 0; i < 8; i++) {

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

byte = (byte << 1) | HAL\_GPIO\_ReadPin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

I2C\_Delay();

}

SDA\_Output();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SDA\_PIN, ack ? GPIO\_PIN\_RESET : GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_SET);

I2C\_Delay();

HAL\_GPIO\_WritePin(I2C\_GPIO\_PORT, I2C\_SCL\_PIN, GPIO\_PIN\_RESET);

return byte;

}

1. Implement EEPROM Read/Write Functions:

uint8\_t EEPROM\_WriteByte(uint16\_t mem\_address, uint8\_t data) {

I2C\_Start();

if (!I2C\_WriteByte(0xA0)) return 0;

if (!I2C\_WriteByte((mem\_address >> 8) & 0xFF)) return 0;

if (!I2C\_WriteByte(mem\_address & 0xFF)) return 0;

if (!I2C\_WriteByte(data)) return 0;

I2C\_Stop();

HAL\_Delay(5);

return 1;

}

uint8\_t EEPROM\_ReadByte(uint16\_t mem\_address, uint8\_t \*data) {

I2C\_Start();

if (!I2C\_WriteByte(0xA0)) return 0;

if (!I2C\_WriteByte((mem\_address >> 8) & 0xFF)) return 0;

if (!I2C\_WriteByte(mem\_address & 0xFF)) return 0;

I2C\_Start();

if (!I

::contentReference[oaicite:0]{index=0}

##### You said:

which clock speed to set for timing

###### ChatGPT said:

When bit-banging I2C, you control the timing, so the I2C clock speed (SCL frequency) depends entirely on the delays you introduce in your code (e.g., in the I2C\_Delay() function).

## ✅ Recommended I2C Speed for AT24C512C

The AT24C512C EEPROM supports:

* Standard Mode: up to 100 kHz
* Fast Mode: up to 400 kHz

## ⚙️ Bit-Bang Timing Setup

Your I2C\_Delay() function sets the SCL frequency. The I2C clock period is roughly twice the time it takes to toggle SCL up and down with delays.

### 🧮 Formula:

T\_clock = 2 \* delay\_time

F\_clock = 1 / T\_clock

### ⏱ Example: 100 kHz SCL (Standard Mode)

To achieve ~100 kHz clock:

T\_clock = 10 µs → Delay = 5 µs

Modify the delay function like this:

void I2C\_Delay(void) {

// For ~100kHz I2C, use a 5 µs delay (adjust depending on CPU)

for (volatile int i = 0; i < 50; i++); // Calibrate as needed

}

⚠ Important: This loop-based delay is affected by CPU speed. You might need to empirically adjust the loop count or use HAL\_Delay() or DWT\_Delay\_us() for more precise timing.

## ⛏ If You Want to Go Faster (e.g., 400 kHz)

You'll need to:

* Reduce delay loop size (e.g., i < 10)
* Ensure signal rise/fall times are fast (pull-ups, trace length)
* Validate EEPROM and wiring can handle it

But for reliability, stick with 100 kHz (Standard Mode).

## ✅ Summary

| Target I2C Speed | Delay per toggle | I2C\_Delay() Loop (Approx.) |
| --- | --- | --- |
| 100 kHz | 5 µs | i < 50 |
| 400 kHz | 1.25 µs | i < 10 |

Let me know your STM32's clock speed (e.g. 48 MHz), and I can give you an exact cycle-based delay function using SysTick or DWT.

##### You said:

what happens to output status if i change the pin direction from output to input in stm32 controllers

###### ChatGPT said:

Great question — this is very relevant when doing bit-banging protocols like I²C, especially with open-drain signaling like on the SDA line.

## ✅ What Happens When You Change a GPIO Pin from Output to Input on STM32?

### When you **change a GPIO pin from output to input**:

1. The pin no longer drives the line.
2. Its output driver is disabled, and it enters a high-impedance (Hi-Z) state.
3. The pin can now be read without affecting the line.
4. Pull-up/down resistors (external or internal) now determine the voltage level on the pin unless another device is driving it.

## 👇 Practical Implication (Especially for I²C Bit-Banging)

In I²C, both SDA and SCL lines are open-drain:

* Devices can pull the line low, but not drive it high.
* A pull-up resistor pulls the line high when no device pulls it low.

So:

| Pin Mode | Behavior |
| --- | --- |
| Output + LOW | Drives line LOW |
| Output + HIGH | Not valid for open-drain; avoid (or configure as input) |
| Input Mode | Hi-Z; external pull-up or other device defines voltage |

### ✅ STM32 GPIO Configurations for I²C

When bit-banging:

* To drive SDA/SCL low, set pin to Output Open-Drain and write LOW.
* To let SDA/SCL go high, set pin to Input (Hi-Z) or Output-OD + write HIGH.

Example behavior for SDA line:

// Pull SDA low

SDA\_Output();

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_7, GPIO\_PIN\_RESET);

// Release SDA (let it go high via pull-up)

SDA\_Input(); // Now pin is Hi-Z

## ❗️ Warning: What NOT to Do

* Don’t set the pin as Push-Pull Output and write HIGH — that can cause bus contention if another device pulls the line LOW (could damage hardware).
* Always use Open-Drain output for I²C or bit-bang emulation of it.

## 🧪 Summary Table

| Direction | Pin State | Line Level | Notes |
| --- | --- | --- | --- |
| Output | LOW | LOW (driven) | Used to pull line low |
| Output | HIGH (OD) | FLOATING | Not useful unless external PU |
| Input | N/A | FLOATING or PU | Used to "release" line |
| Push-Pull | HIGH | HIGH (driven) | ❌ Dangerous for I²C |

Let me know if you want to simulate or visualize this on a logic analyzer or oscilloscope.

##### You said:

how to un ignore in stm32cube ide a file

###### ChatGPT said:

To un-ignore a file in STM32CubeIDE, it depends on how it was ignored in the first place. Here are the typical cases and solutions:

## 🔍 Case 1: **File is ignored via** .gitignore

If you're using Git and the file was ignored using a .gitignore file:

### 🛠 Steps to un-ignore:

1. Open .gitignore file in your project root.
2. Remove or comment out the line that matches your file/folder.

* # Ignore main.c

#Core/Src/main.c <-- comment or delete this line

* If the file was already ignored, Git will still ignore it unless you force it to track:

Open terminal (or Git Bash) in your project folder:

* git add -f Core/Src/main.c
* Then commit:

1. git commit -m "Un-ignore main.c"

## 🔍 Case 2: **File was excluded from build in STM32CubeIDE (not Git related)**

STM32CubeIDE can exclude files from compilation using IDE settings.

### 🛠 Steps to re-include (un-exclude) the file:

1. Right-click on the file in Project Explorer (e.g., main.c).
2. Go to Properties.
3. Navigate to:

* C/C++ General > Paths and Symbols > Source Location
* OR directly:

1. C/C++ Build > Settings > Tool Settings > MCU GCC Compiler > Includes
2. Make sure the file is not excluded from the build. Alternatively:
   * Right-click the file → Resource Configurations → Exclude from build
   * Uncheck the box.

## 🔍 Case 3: **File is filtered in Project Explorer**

If a file doesn't appear because of a filter (not ignored from build):

### 🛠 Fix:

1. Click the small down arrow on the Project Explorer tab.
2. Choose Customize View...
3. Go to Filters tab and ensure nothing relevant (like \*.c or .\*) is hiding your file.

## ✅ Summary

| Situation | Solution |
| --- | --- |
| Ignored by .gitignore | Edit .gitignore + git add -f |
| Excluded from build | Right-click → Resource Configurations → Uncheck "Exclude from build" |
| Hidden in Project Explorer | View menu → Customize View → Remove filters |

Let me know if you want to check for a specific file or can’t find where it’s being ignored.

##### You said:

i ignored from Team -> ignore

###### ChatGPT said:

Thanks for the clarification!

If you ignored a file via Team → Ignore in STM32CubeIDE, that means you're using Git integration in the IDE, and the file was added to the .gitignore list.

## ✅ How to Un-Ignore a File Ignored via Team → Ignore in STM32CubeIDE

### 🛠 Steps:

### 1. **Open the** .gitignore **File**

* In STM32CubeIDE:
  + Locate the .gitignore file (usually in the root of your project).
  + Open it.
  + Look for the line corresponding to the file or folder you ignored (e.g., Core/Src/main.c or Core/Src/).

### 2. **Remove or Comment Out the Line**

# Remove this line to un-ignore the file

Core/Src/main.c

You can also comment it out:

#Core/Src/main.c

### 3. **Force Git to Track the File Again**

Git still considers the file ignored unless you tell it to track it again.

* Open Git Bash or STM32CubeIDE's Git terminal, and run:

git add -f Core/Src/main.c

### 4. **Commit the Change**

git commit -m "Un-ignored main.c"

### 🧠 Why This Happens

When you choose Team → Ignore, STM32CubeIDE adds the file path to the .gitignore. Git then stops tracking the file unless forced (git add -f), even if you modify it.

### ✅ Optional: Refresh in STM32CubeIDE

* After editing .gitignore, right-click your project → Refresh (or press F5) to update the IDE’s view.

Let me know the file path you ignored if you'd like exact .gitignore and git add commands!

##### You said:

how to know the user name in github

###### ChatGPT said:

To find your GitHub username, here are a few easy ways depending on what you’re doing:

## ✅ 1. **If You're Logged into GitHub in a Browser**

1. Go to [https://github.com](https://github.com/).
2. In the top right, click your profile picture.
3. You'll see your username next to your avatar in the dropdown.
4. Or, go to your profile:
   * Your GitHub URL will be:  
     https://github.com/your-username

## ✅ 2. **From Git Command Line (Terminal or Git Bash)**

To see your GitHub username configured for commits (not necessarily the logged-in account):

git config --global user.name

Or for a specific repo:

git config user.name

This shows the commit author name, not your GitHub account name (but often the same).

## ✅ 3. **From GitHub Remote URL (in your local repo)**

If you've cloned a repository via HTTPS or SSH:

git remote -v

You might see something like:

origin https://github.com/your-username/repo-name.git (fetch)

Here, your-username is your GitHub username.

## ✅ 4. **If You Use GitHub CLI (**gh**)**

If you’ve set up the GitHub CLI:

gh auth status

You’ll see your logged-in GitHub username.

If you're trying to find someone else's GitHub username from a commit or email, let me know and I can help with that too.

##### You said:

stm32cubeide is asking for user and passward of github, what to put

###### ChatGPT said:

STM32CubeIDE (when using Git features via Team → Push/Pull/Clone) may prompt for your GitHub username and password — but GitHub no longer allows password-based login for Git over HTTPS. Instead, you must use a Personal Access Token (PAT).

## ✅ What to Use When STM32CubeIDE Asks for "Username" and "Password"

### 🔑 Username:

* Your GitHub username

### 🔐 Password:

* A Personal Access Token (PAT) from GitHub — not your actual GitHub password

## 🔧 How to Create a Personal Access Token (PAT)

1. Log in to GitHub:  
   [https://github.com](https://github.com/)
2. Go to:  
   <https://github.com/settings/tokens>
3. Click “Generate new token” → Choose “Fine-grained token” or "Classic token":
   * For STM32CubeIDE, a Classic token is simpler.
4. Set permissions:
   * repo → full access (for private repos)
   * read:org (optional if accessing org repos)
5. Click Generate token
6. Copy the token — you will not be able to see it again!

### 📌 When STM32CubeIDE Prompts:

* Username: your GitHub username (e.g., johnsmith)
* Password: paste the token you just created

You can check “Remember password” to avoid doing this every time.

## 🔒 Optional: Use Credential Manager

If you’re on Windows and using Git, you can also cache credentials:

git config --global credential.helper manager

This will securely store your token.

## ✅ Summary

| Prompt | Enter This |
| --- | --- |
| Username | Your GitHub username |
| Password | GitHub Personal Access Token (PAT) |

Let me know if you’d like help generating the token or setting it up securely in STM32CubeIDE.

##### You said:

how to make first line in README.md in github big

###### ChatGPT said:

To make the first line in your README.md big on GitHub, use a Markdown heading.

## ✅ Syntax for a Big Header (Heading Level 1)

# This is a Big Heading

This creates the largest standard header in Markdown (like <h1> in HTML). It will be bold and large on GitHub.

## 🧪 Example README.md:

# My Awesome Project

This project does amazing things with STM32 and EEPROMs!

### Result on GitHub:

My Awesome Project (Big bold text as title)

## 🧠 Tip:

* Use only one # for the biggest size.
* You can use ##, ###, etc. for smaller headings (H2, H3, ...).

=======================================================================