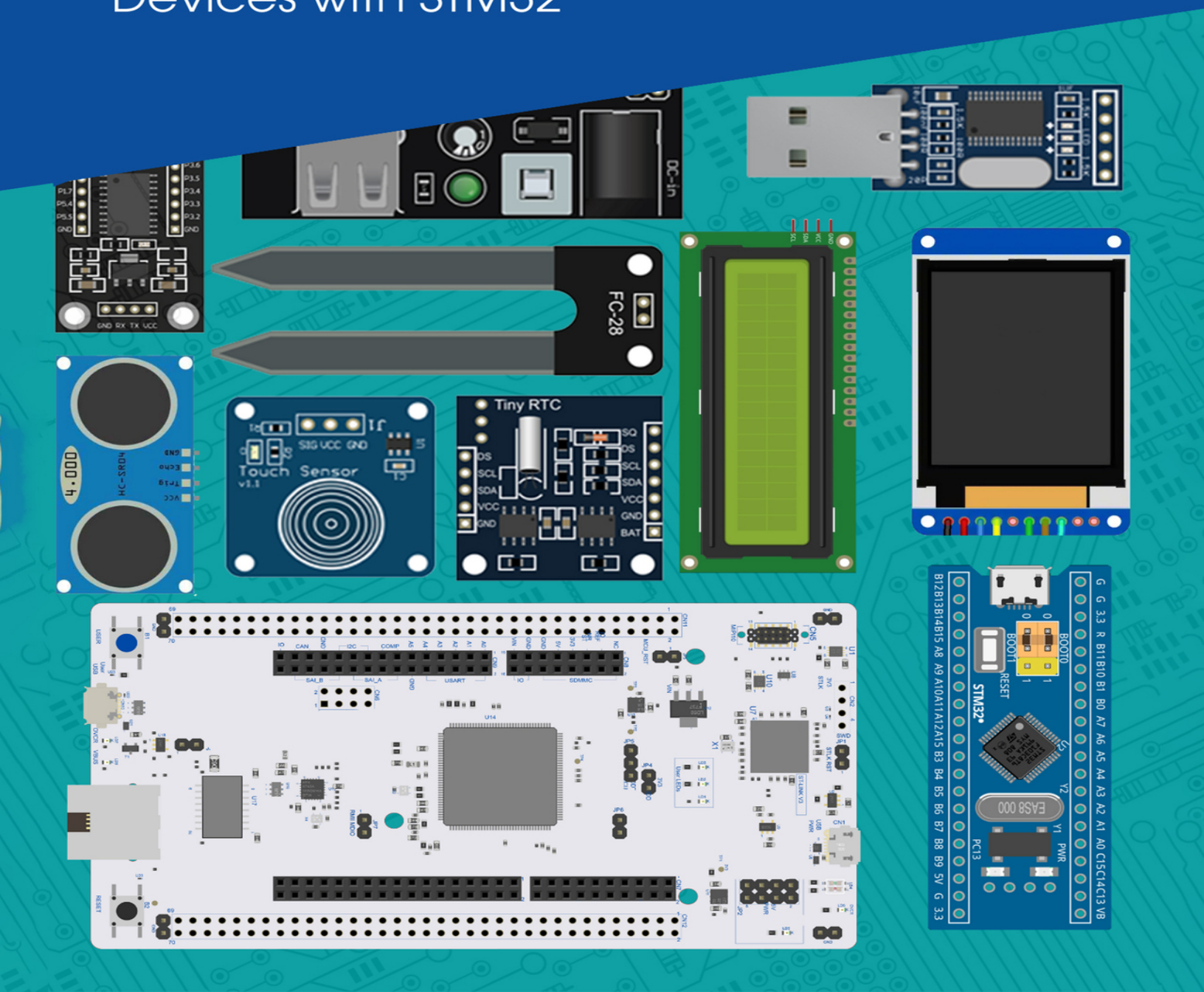
STM32 IOT PROJECTS FOR BEGINNERS

A Hands-On Guide to Connecting Sensors, Programming Embedded Systems, Build IoT Devices with STM32



STM32 IOT PROJECTS FOR BEGINNERS

A Hands-On Guide to Connecting Sensors, Programming Embedded Systems, Build IoT Devices with STM32

By

Aharen-san

TABLE OF CONTENTS

DF PLAYER MINI AND STM32

EXAMPLE DUMMY CODE

GPS MODULE AND STM32 NEO 6M GET COORDINATES DATE TIME SPEED

EXAMPLE DUMMY CODE

HCSR04 AND STM32 USING INPUT CAPTURE PULSE WIDTH CUBEIDE

EXAMPLE DUMMY CODE

INCREMENTAL ENCODER AND SERVO ANGLE CONTROL IN STM32 PWM

EXAMPLE DUMMY CODE

INPUT CAPTURE USING DMA MEASURE HIGH FREQUENCIES AND LOW WIDTH

EXAMPLE DUMMY CODE

JOYSTICK MODULE WITH STM32 ADC MULTI CHANNEL HAL

NEXTION GUAGE AND PROGRESS BAR STM32

NUMBERS FLOATS QR CODE HOTSPOTS IN NEXTION DISPLAY STM32

PRINTF DEBUGGING USING SEMIHOSTING IN STM32 SW4STM LIVE VARIABLE CHANGE

QSPI IN STM32 BOOT FROM EXT MEMORY XIP N25Q

QSPI IN STM32 WRITE AND READ N25Q

RIVERDI STM32 DISPLAY HOW TO CONTROL LED USING BUTTONS ON THE DISPLAY

RIVERDI STM32 DISPLAY HOW TO SEND DATA FROM UART TO UI

ROTARY ANGLE SENSOR AND STM32 ADC

SD CARD USING SDIO IN STM32 UART RING BUFFER 4-BIT MODE CUBEMX

SD CARD USING SPI IN STM32 CUBE-IDE FILE HANDLING UART

SDRAM IN STM32 MT48LC4

SLEEP MODE IN STM32F103 CUBEIDE LOW POWER MODE CURRENT CONSUMPTION

SSD1306 OLED AND STM32 128X64 SW4STM CUBEMX

ST7735 TFT DISPLAY AND STM32 HAL

STANDBY MODE IN STM32 LOW POWER MODES CUBEIDE

STEPPER MOTOR AND STM32 ANGLE RPM AND DIRECTION CONTROL CUBEIDE

STOP MODE IN STM32 CUBEIDE LOW POWER MODE

STORE DATA INTO SD CARD FREERTOS STM32 ADC DHT

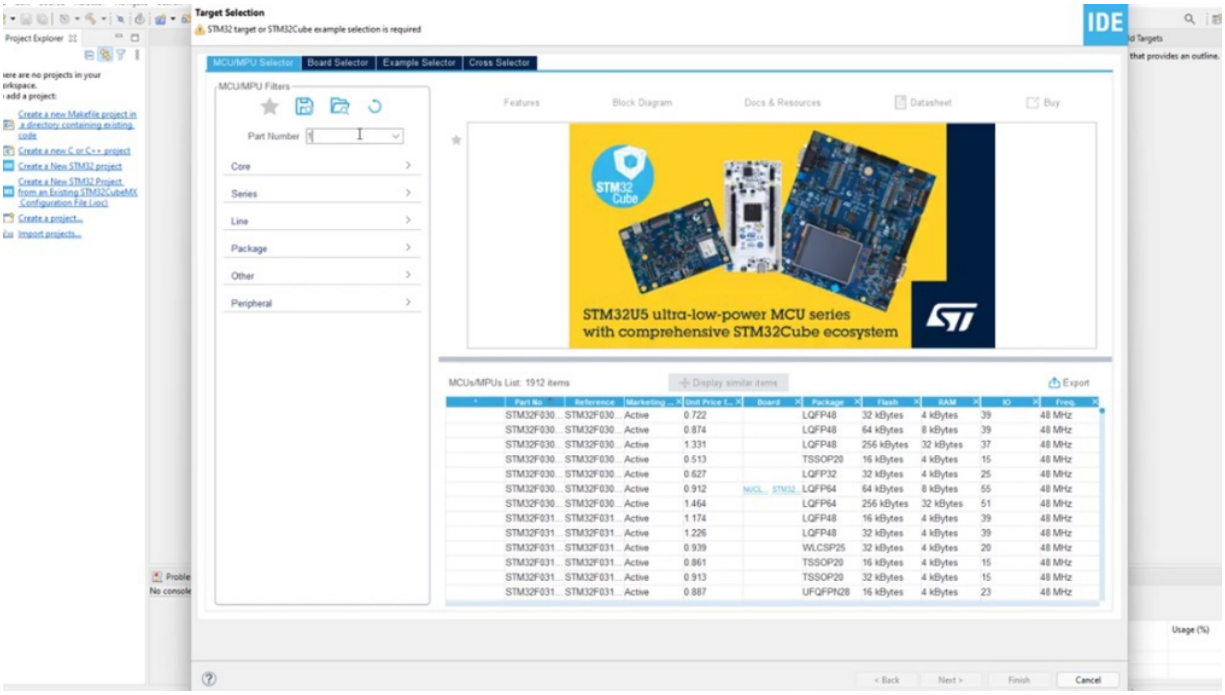
UART RING BUFFER USING HEAD AND TAIL IN STM32 CUBEIDE

USE STM32 AS A KEYBOARD F103C8 USB DEVICE HID

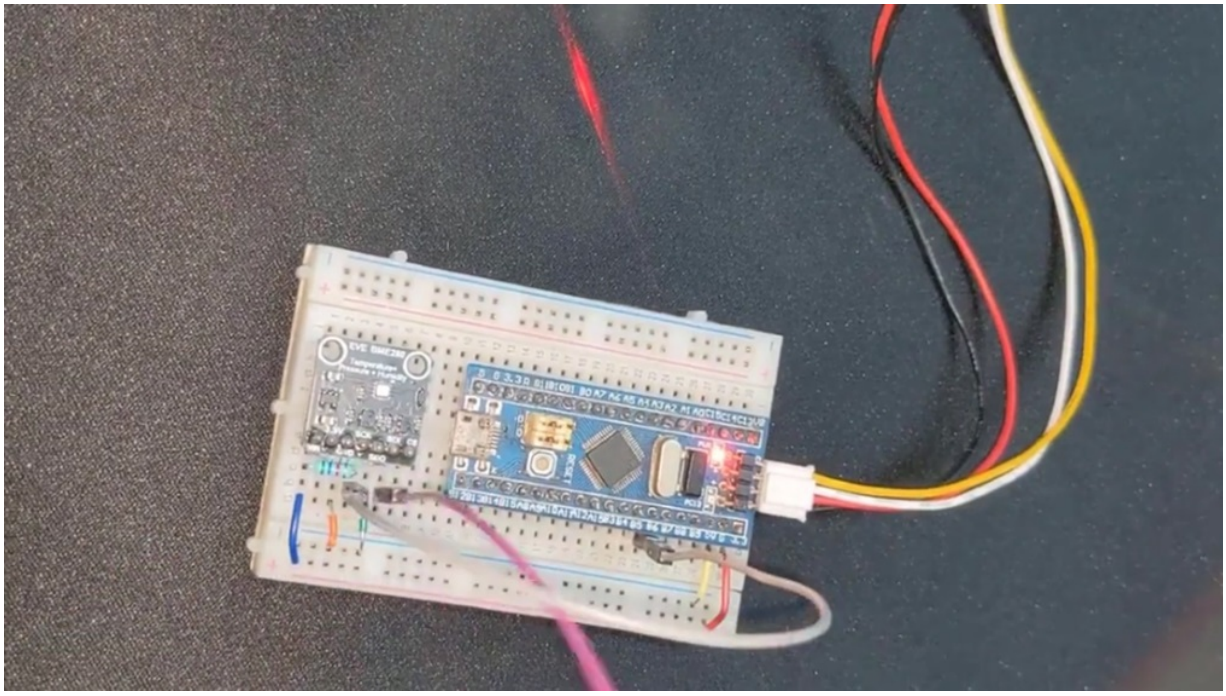
USING PRINTF DEBUGGING SWV TRACE IN CUBEIDE ITM SWV

BME280 WITH STM32 I2C TEMP PRESSURE HUMIDITY

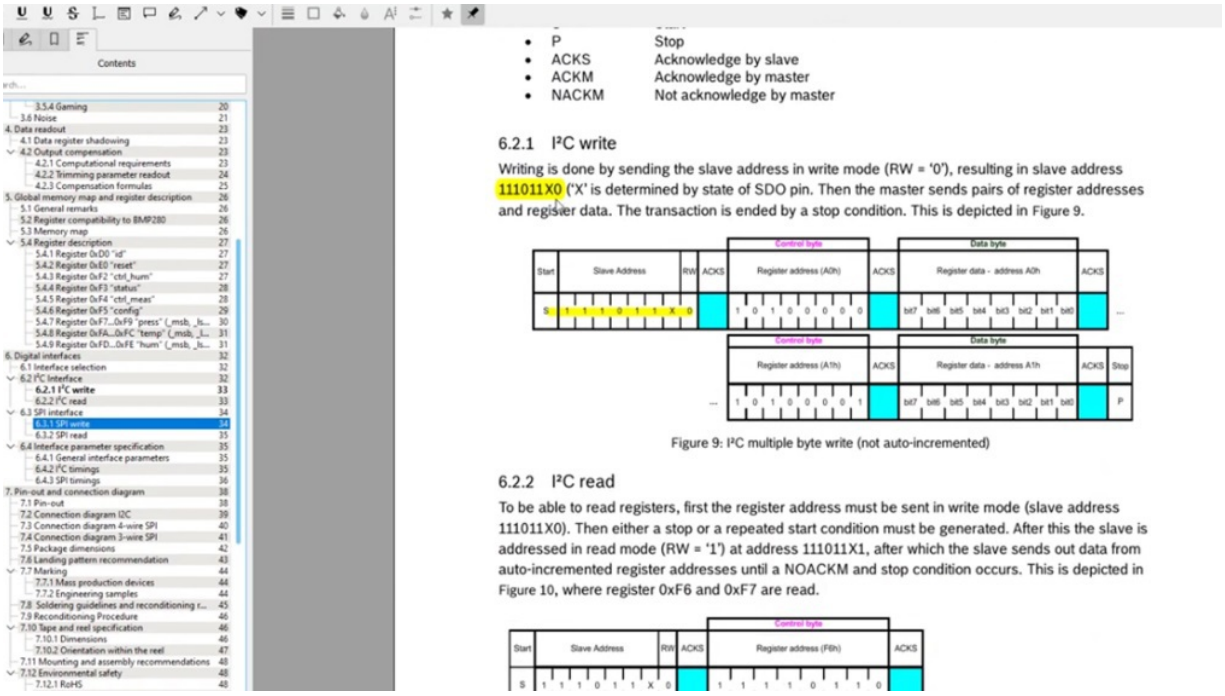
We will see how to interface BM e 280 sensor with STM 32. This sensor can measure the temperature, pressure and relative humidity. I have written a library for it, which I will upload on the GitHub and you can get it from there. As we progress along the project, I will also explain the code and how you can write one yourself using the datasheet. The library covers a lot of things, but there are still few things which you need to manually implement. So watch the project carefully as you might need to make changes in the library based on what requirements you have from the sensor. This is the datasheet for the device. Here I have highlighted few important things that I will cover in today's project. I will leave the link to this data sheet in the description.



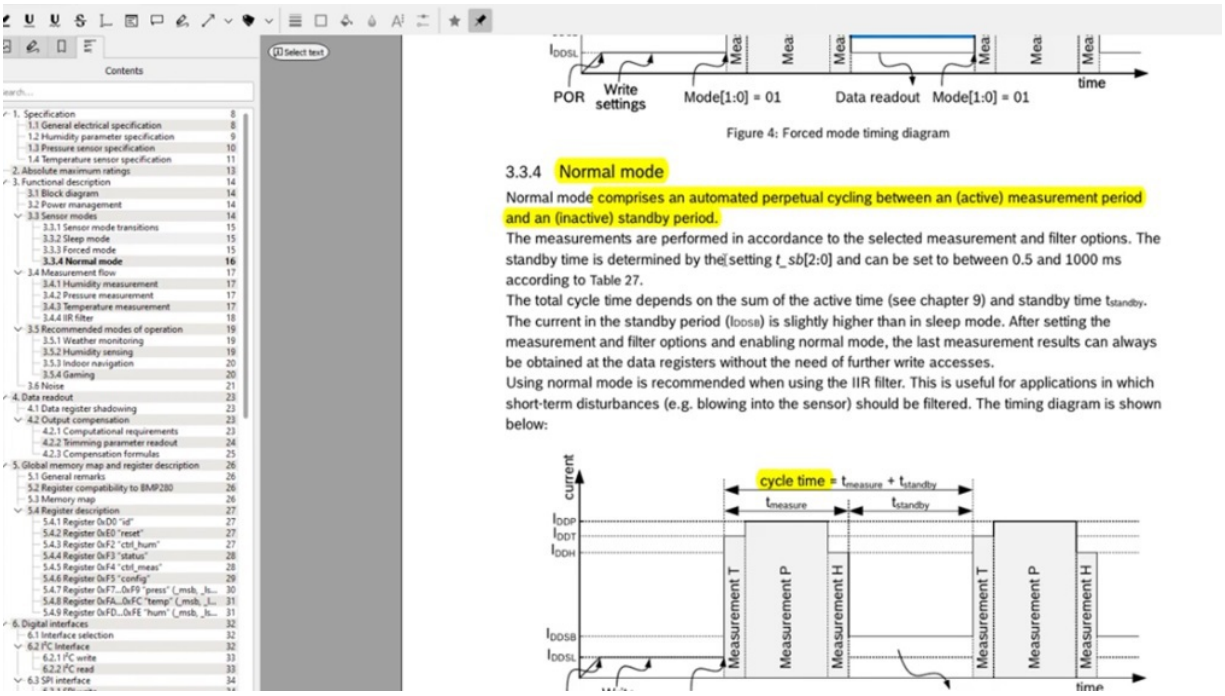
Let's start with cube ID and create a new project I am using STM 32 F 103 controller give some name to the project and click finish first of all I am enabling the external crystal for the clock. The blue pill have eight megahertz crystal on board and I want the system to run at maximum 72 megahertz clock. Enable the serial wire debug. The sensor can use both the eye to C and SPI for communication. You can use either of those but I am going to go with the eye to see enable the eye to see interface and leave everything to default. We have the two pins for data and clock. Before going any further in the project. Let's see the sensor and the connection with blue pill. Here is the BM e 280. And as you can see it has the pinout for both SPI and eye to see. Here I am connecting it with the blue pill. It's powered with 3.3 volts and there are two pull up resistors each 4700 ohms connected between the clock and data pins and the 3.3 volts. Poor resistors must be used while using the eye to see communication. Also one very important thing I have grounded the SDO pin. Keep this in mind as it will be used in the addressing of the device. Now connect the PV six to the clock pin and PV seven to the SDI pin that is data pin.



That completes the connection. Let's generate the project now. First thing we will do is copy the library files into our project. So copy the C file into the source directory and header file into the include directory. Let's take a look at the source file. Here first we have to define the eye to see we are using as I set up the i two c one so I am leaving it unchanged. The next thing is the 64 bit support. If your configuration supports 64 bit integers, then leave this as one or else to use the 32 bit integers uncomment the 32 bit support and comment out 64 bit the next is the address of the device. As mentioned in the datasheet the seven bits of the address are these. Here x depends on the SDO pin. And if you remember I grounded the pin and therefore the X is zero in my case the slave address will consist of these seven address bits along with the read or write bit. So the address will be 11101100 which makes up zero Crossy see. These variables will store the corresponding values and they are externally defined here. So you should define them in the main file. The rest of the code should be unchanged for default configuration Let me explain how this works.

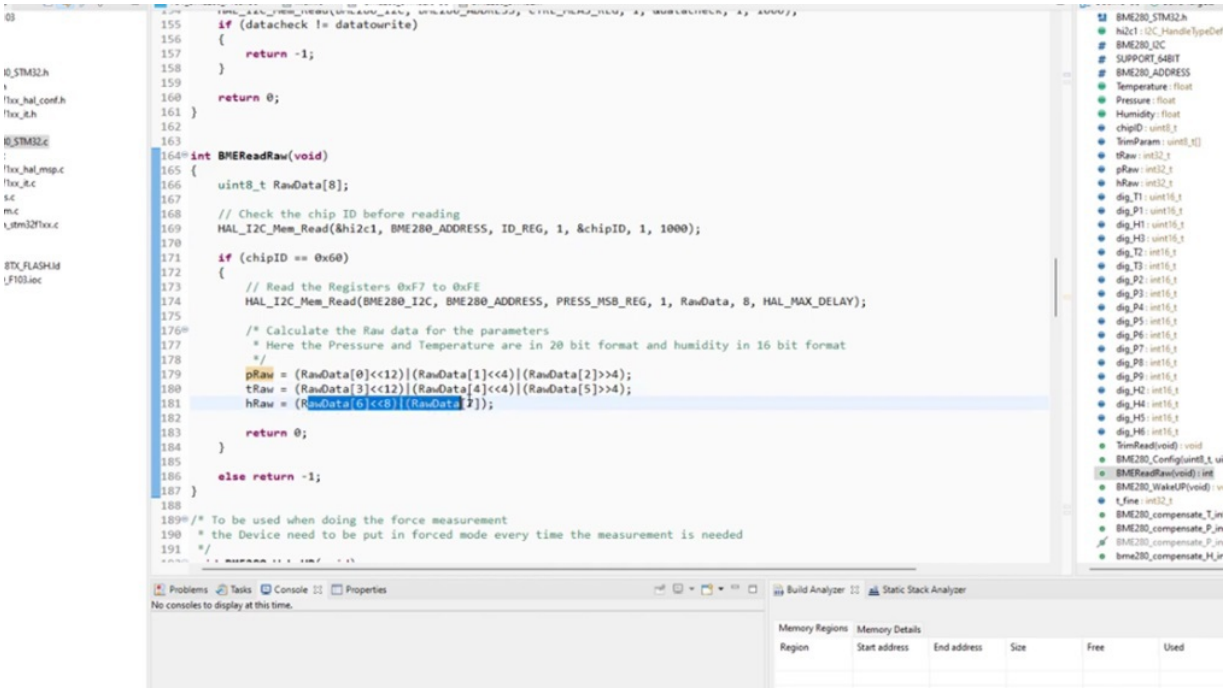


The sensor can work with three different modes. in sleep mode, no measurements are performed, but the registers are accessible and therefore, you can wake the sensor and perform the measurement. Then comes the forced mode. Here the sensor performs a single measurement and goes into the sleep mode. For the next measurement, you need to wake the sensor again, I have added a function for this you need to call this wakeup function before doing the measurement. This is useful in situations like weather monitoring, where the data does not need to be read very frequently. Basically, it will measure all three parameters and then go back to sleep mode. The next is the normal mode.



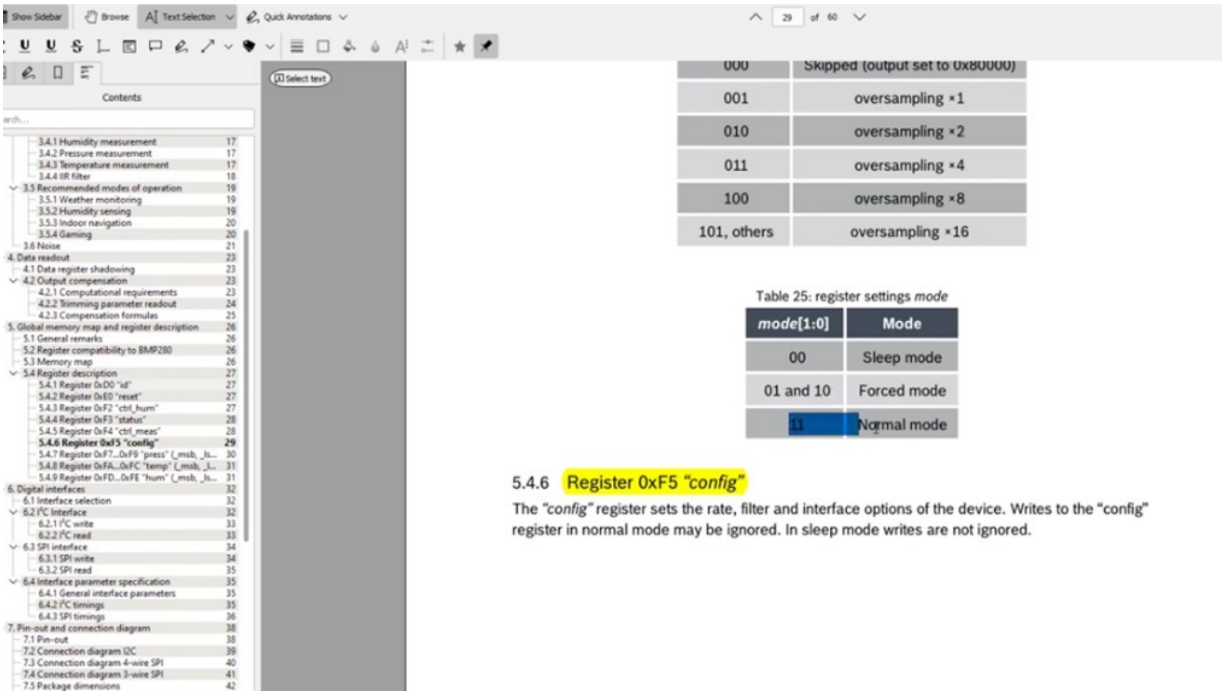
Here the sensor does the measurement and goes into the standby. The data rate depends on the measurement time and standby time, the current consumption will obviously be higher, but it allows you to continuously monitor the data. In this tutorial, I will be using the normal mode. There are few important things to note about the measurements, the humidity measurement have a fixed resolution of 16 bits. The resolution for the temperature and pressure depends on the fact that if you are using the IR filter or not, if using the filter, then the resolution will be 20 bits. Otherwise, it depends on the oversampling setting as shown here. IR filter can be used to avoid the fluctuations in the pressure and temperature measurements. I will be using the IR filter in this tutorial. And this library does not support the measurements without filter, at least for now. Then we have some examples for the settings which we will see later. Let's check the source file again. Here the first function is the trim read. This reads the trimming values that are stored in the non volatile memory of the sensor. Every sensor comes pre programmed with these values, and they don't change with reset or anything. We need to read these values and then use them in the calculations ahead as shown in the datasheet we must read the values from these registers. And I am going to name them same as its named here.

Also note that some of these are unsigned and others are signed values. So that's why I have defined them separately. Then we start reading from zero by 88 address and we burst read 25 registers. This means we read up to zero XA one which is Digg h one. Again, we have to read from E one to E seven. This is done here we are reading seven bytes from E one. And finally we will arrange the data just how it's arranged in the data sheet. Then comes the configuration which we will see later. Next is the reading of raw data. It's mentioned in the datasheet that we must first read the data in order to avoid the possible mix ups between the measurements. We will read the registers F seven to f e. This is done here. We are reading from this register. It is defined in the header file and its addresses F seven and we will read eight bytes from here which will include the registers up to F E. As I mentioned in the beginning, this library is using the filter and this is why the pressure and temperature are 20 bits in resolution. We will calculate the row values for all three parameters using the registers we just read.



After this, we have the compensation formulas on page 25 of the datasheet. These formulas uses the row values for temperature, pressure and humidity and gives us the refined results. We need to use them exactly in the same way. So this is what that is, I just

copied them from the datasheet and put them here. Notice that the pressure uses the 64 bit integer here. But in case your machine doesn't support it, there is a 32 bit alternative also, I have included that in the library. So all you need to do is define the support as I mentioned in the beginning. All right, now we will take a look at the registers. The first register is the ID register, it's a read only register, and it returns the ID of the device, which should be zero by 60. I have included it in the code. Before reading the raw values, the code checks for the ID. If the ID is zero by 60, only, then it goes for the measurement. The next register is reset. If we write zero XP six to this register, the device will soft reset. The next register is for the humidity control. Here we need to select the oversampling for the humidity. If you want to skip the humidity calculation, just set the oversampling to zero. This configuration is controlled by the BM e 280 config function. Here are the parameters or the oversampling settings for all three parameters. Then we have the mode, the standby time and finally the filter configuration. First of all, we will read the trimming parameters as it only needs to be done once and then I am performing the soft reset then write the oversampling data for the humidity these oversampling parameters are defined in the header file and you can use them instead of writing a hexadecimal value. After writing the data, we will read the same register to make sure the changes were done in the register. The next register is control measure register. It controls the oversampling of temperature and pressure along with the mode of the sensor. Here I am shifting the temperature data by five the pressure data by two and the first two bits are for the mode.



The next register is the config register. It controls the standby time for the normal mode along with the filter coefficients for the IR filter. You can also use three wire SPI and enable it from here below the tables for both standby time and filter coefficients. I have defined them in the header file also. And at last, we have the data registers where we read the data from the pressure, temperature and humidity has mentioned here. This is how the 20 bit data is arranged in these registers. This is enough explaining I hope let's write the code now that I have defined these configurations as per my setup, I am using i two c one and also my system supports 64 bit variables. Let's copy this in the main file, we will include the BM e 280 header file, and now define the variables to store the data. Inside the main function we will call the BM e 280 config function. There are some examples provided in the datasheet. For example, for weather monitoring, we can use the Force mode with one sample per minute the pressure, temperature and humidity oversampling all should be set to one. Similarly there are other examples but in this tutorial, I will use the indoor navigation. Here I will use the normal mode With standby time of 0.5 milliseconds, the pressure oversampling is 16. And that for temperature is two and one for humidity. So let's set the temperature over sampling to to pressure to 16 and humidity to one the mode

should be set to normal mode, the standby time is set to 0.5. And at last the filter coefficient is 16. With this configuration, the output data rate is 25 Hertz. inside the while loop, we will call the BM e 280 Measure function. This will handle all the measurement and store the values in the variables that we defined earlier. All right, everything is done. Now build the code and debug it. Let's run it you can see the values of temperature in degree Celsius pressure in Pascal's and relative humidity as a percentage, I am going to put my finger on the sensor see the value of the temperature rising. I kept the sensor in the refrigerator for a while and see the temperature and humidity values. The temperature is rising and humidity is decreasing. They both are slowly tending towards the room condition. This is it for this project.

EXAMPLE DUMMY CODE

I can provide you with an example code snippet for interfacing the BME280 sensor with an STM32 microcontroller using the I2C interface. This code assumes you have already set up the necessary hardware connections for the I2C communication. Here's a basic example using the STM32 HAL (Hardware Abstraction Layer) library:

C

```
#include "stm32f4xx_hal.h"
#include "bme280.h"  // BME280 library header

I2C_HandleTypeDef hi2c1;  // I2C handle
BME280_HandleTypedef hbme280;  // BME280 handle

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);

int main(void)
{
  HAL_Init();
```

```
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();

  // Initialize BME280 sensor
  hbme280.i2c_handle = &hi2c1;
  BME280_Init(&hbme280);

  while (1)
  {
    float temperature, pressure, humidity;

    // Read sensor data
    BME280_ReadTemperature(&hbme280,
&temperature);
    BME280_ReadPressure(&hbme280,
&pressure);
    BME280_ReadHumidity(&hbme280,
&humidity);

    // Process and use sensor data as needed

    HAL_Delay(1000); // Delay for 1 second
  }
}

void SystemClock_Config(void)
{
  // Configure the system clock here
}
```

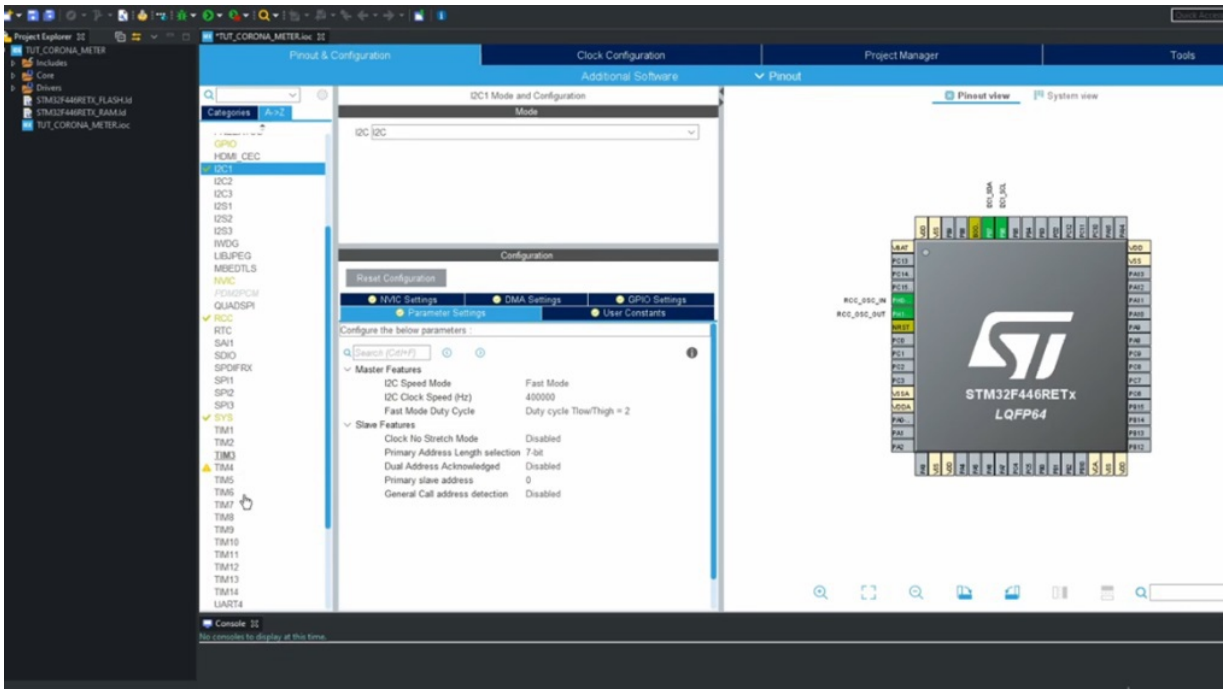
```
static void MX_I2C1_Init(void)
{
  // Initialize I2C1 peripheral here
}

void HAL_I2C_MspInit(I2C_HandleTypeDef* i2cHandle)
{
  // Initialize I2C MSP (GPIO, NVIC, etc.) here
}

void HAL_I2C_MspDeInit(I2C_HandleTypeDef* i2cHandle)
{
  // Deinitialize I2C MSP here
}
```

Please note that you need to include the appropriate BME280 library and modify the code according to your specific STM32 microcontroller and development environment. Also, make sure that you have configured the I2C peripheral and corresponding GPIO pins correctly in the MX_I2C1_Init and HAL_I2C_MspInit functions.

Remember to refer to the BME280 datasheet and STM32 reference manual for accurate pin mappings, register settings, and other details.

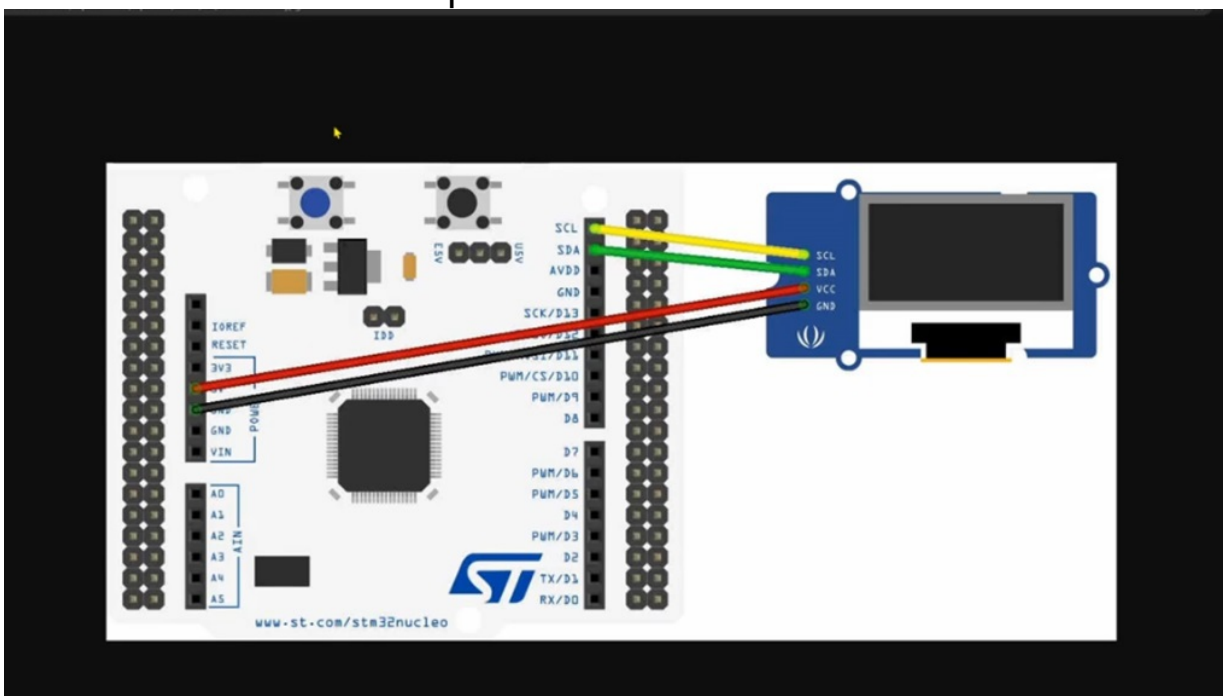
CORONA DISPLAY METER USING STM32 ESP8266 OLED LCD16X2

I will also demonstrate this on LCD 16 cross two. Let's start by creating a project in cube ID I am using STM 32 F 446 R E but you can use any other controller to First thing first I am selecting external crystal for the clock select our to see for connecting the displays we need to use the fast mode for OLED display LCD display can work in either mode.



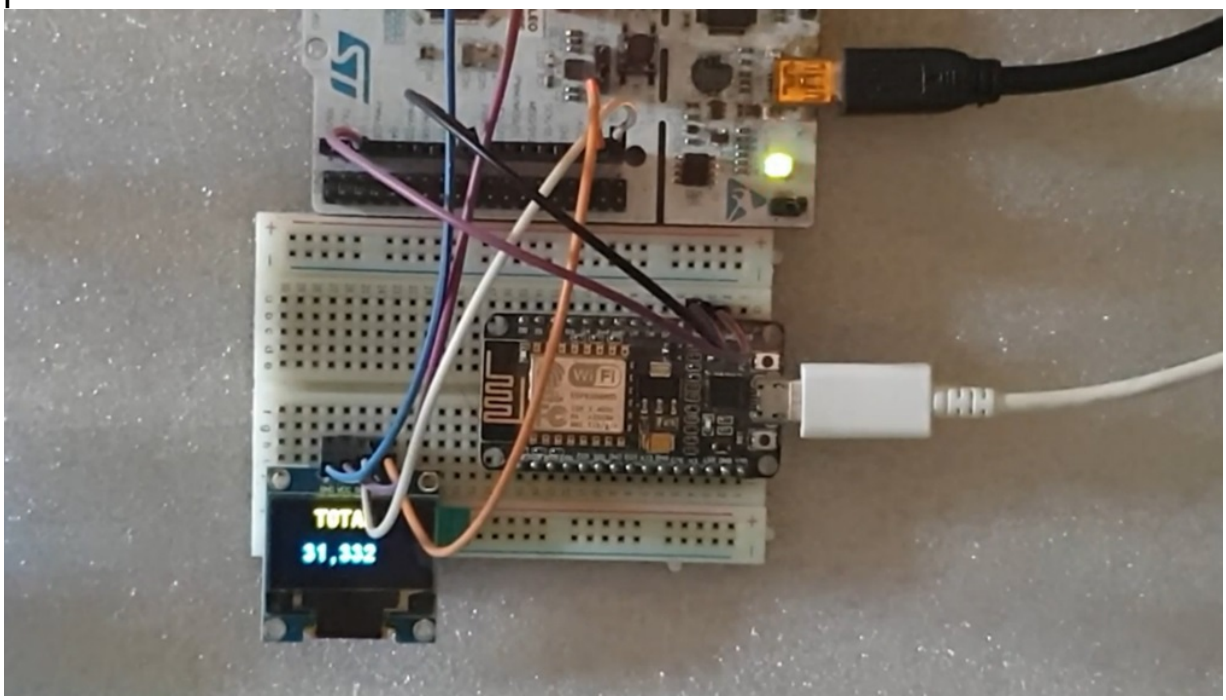
Next select the UART in asynchronous mode and enable the interrupt next the clock setup I am selecting external crystal which is eight megahertz and I want the system to run at 100 megahertz also I am changing these pins for convenience click Save to generate the

project this is our main file let's see the connections first. This is how E SP is connected to the controller connector our x pins a TX and TX two RX connect Vcc and C H P d to the five volts and ground to ground Oh led is connected straight forward now we need to copy these library files. You can get them after you download the code from the link in the description.



Copy the C files in the source directory and header files in the include directory. Now include the functions dot h file and SSD 1306 dot h file. If you are using any other eye to see you need to update it here also update the UART if you are using any other let's copy this function and we will paste it in the interrupt file also change the UART interrupt handler. These are all the basic setup we have used in previous projects two functions file only have two functions. This is to initialize the E SP. You need to enter your Wi Fi credentials here. This function is to get the data it requires the API key to get it, go to thingspeak.com and create an account. After logging into your account, go to Apps thing HTTP new thing HTTP give some name to this now we have to get data from somewhere go to well domitor dot info select your country. Now copy this address and paste it in the thingspeak URL. Leave everything as it is go back to well domitor Right click on the total cases click Inspect Right Click here, copy,

copy XPath and paste this in the past string remove the span and click Save. That's it now let's write the main program. Initialize the OLED display. print some data to show the status. Initialize the ESP and enter the Wi Fi credentials, clear the display in the while loop, print some info.



Now we will get the data copy the API key from the thingspeak and paste it here we also need to create some arrays to save this data to enter the arrays to their respective positions. And now we will print this results on the OLED This will print the total number of cases deaths and recovered cases let's build this and flush it into the board you can see the data being displayed to show that it's working properly I will change the country here you can see the data is also updated let's change it again and data is updated again. I will change this back to my country. If you are using LCD we need to copy the LCD related files into the source and include folders in code include the ITU c I c, d dot h file now we will initialize the LCD. print some string to display status initialize the ESP eight to six six. get data from the server and print it on the LCD.

EXAMPLE DUMMY CODE

It seems you're asking for an example code that combines a STM32 microcontroller, an ESP8266 module, and an OLED LCD 16x2 display to create a corona display meter. However, there might be a mix-up in your question. The STM32 microcontroller and ESP8266 are two separate components.

Assuming you want to create a corona display meter using an STM32 microcontroller and an OLED LCD 16x2, I'll provide you with an example code snippet for interfacing the STM32 and OLED display to create a simple meter-like animation. If you're specifically looking to integrate an ESP8266 into the project, you'd need to modify the code accordingly to communicate with the ESP8266.

C

```
#include "stm32f4xx_hal.h"
#include "ssd1306.h"  // OLED library header

I2C_HandleTypeDef hi2c1;  // I2C handle
```

```
SSD1306_HandleTypeDef holed1;  // OLED
handle

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void SSD1306_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();

  SSD1306_Init();

  uint8_t progress = 0;  // Progress of the corona
display

  while (1)
  {
    SSD1306_ClearBuffer(&holed1);

    // Draw the corona meter
    for (int i = 0; i < progress; i++) {
        SSD1306_DrawPixel(&holed1, i, 15,
SSD1306_COLOR_WHITE);
    }
```

```
    SSD1306_UpdateScreen(&holed1);

    progress++;
    if (progress > 128) {
        progress = 0;
    }

    HAL_Delay(100); // Adjust delay for
animation speed
  }
}

void SystemClock_Config(void)
{
  // Configure the system clock here
}

static void MX_I2C1_Init(void)
{
  // Initialize I2C1 peripheral here
}

void HAL_I2C_MspInit(I2C_HandleTypeDef*
i2cHandle)
{
  // Initialize I2C MSP (GPIO, NVIC, etc.) here
}

void HAL_I2C_MspDeInit(I2C_HandleTypeDef*
i2cHandle)
```

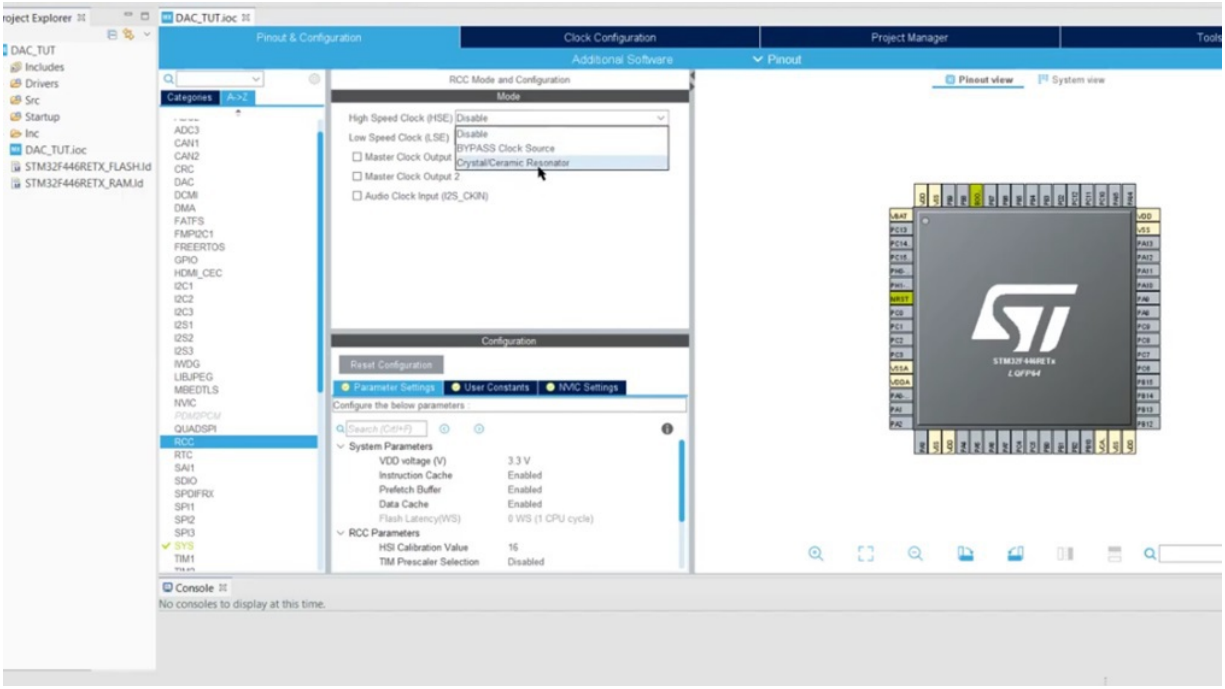
```
{
  // Deinitialize I2C MSP here
}

static void SSD1306_Init(void)
{
  // Initialize the OLED display here using the SSD1306 library
}
```

This code uses the SSD1306 OLED library (assumed to be named ssd1306.h) to communicate with the OLED display. The loop increments a progress variable, which is used to draw the corona meter-like animation on the OLED screen.

Please make sure to set up the I2C communication and the SSD1306 OLED initialization according to your hardware configuration. If you need to integrate an ESP8266 for any specific purpose, you'll need to adapt the code to manage the ESP8266 communication and functionality as well.

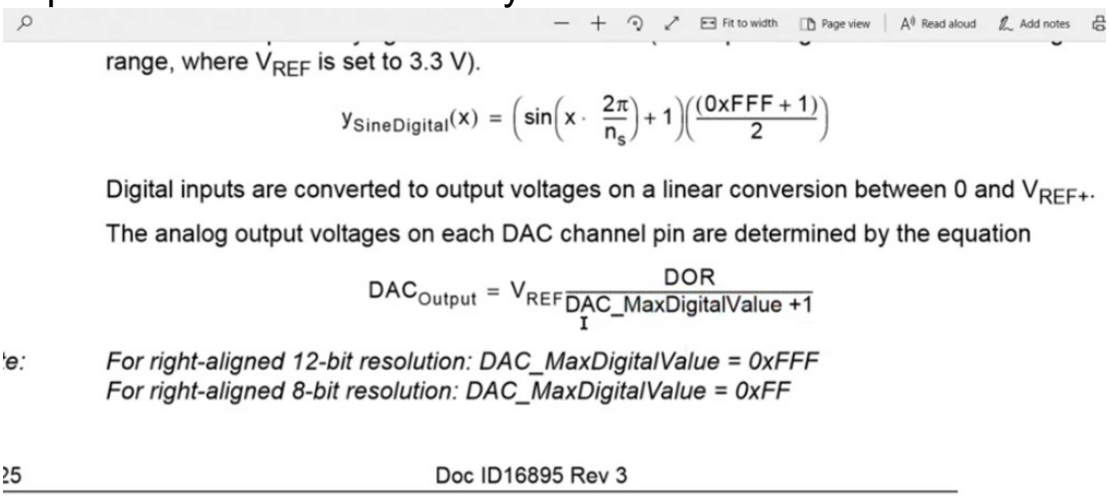
DAC IN STM32 SINE WAVE HAL CUBEIDE

We will create a sine wave using D AC and STM 32. digital to analog converters are basically opposite to ADC, they convert the digital signal to the analog signal and as the sine wave is the best representation of an analog signal, that's what we will create today. Let's start with the cube Id first I am using STM 32 F 446 R E, F 103 ca turns of Da C so I have to use this one let's set up the cube mix.



First things first I am using the external crystal for the clock. Next select the DSC configuration as you can see here, the pin PA four is selected as the ACF pin. By default output buffer is enabled and there is no trigger we will keep it like this for the first part of this project. Next, set up the clock and once done click Save to generate the code. Here is the generated main dot c file. Before starting let's

see the document provided by st on da C as you can see the formula here to calculate the D AC output the ref is 3.3 volts D O R is the value that we don't know Max digital values are given below depends on what resolution you use.



range, where V_{REF} is set to 3.3 V).

$$y_{SineDigital}(x) = \left(\sin\left(x \cdot \frac{2\pi}{n_s}\right) + 1\right)\left(\frac{(0xFFF + 1)}{2}\right)$$

Digital inputs are converted to output voltages on a linear conversion between 0 and V_{REF+}.

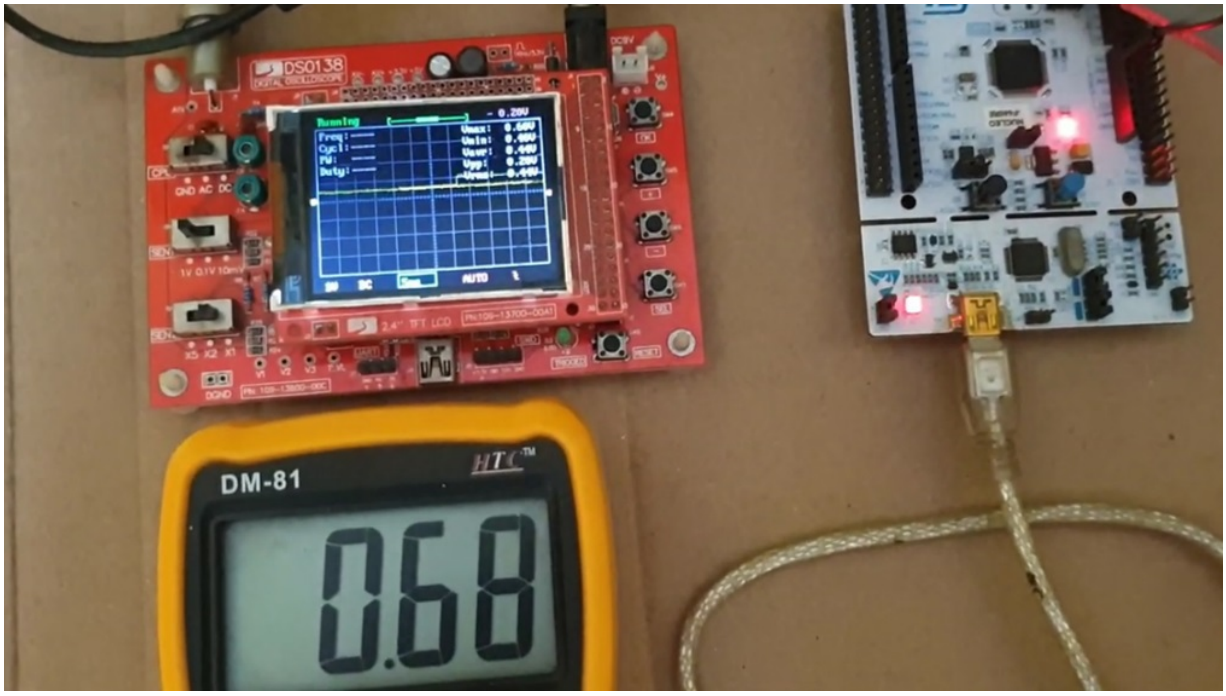
The analog output voltages on each DAC channel pin are determined by the equation

$$DAC_{Output} = V_{REF} \frac{DOR}{DAC\_MaxDigitalValue + 1}$$

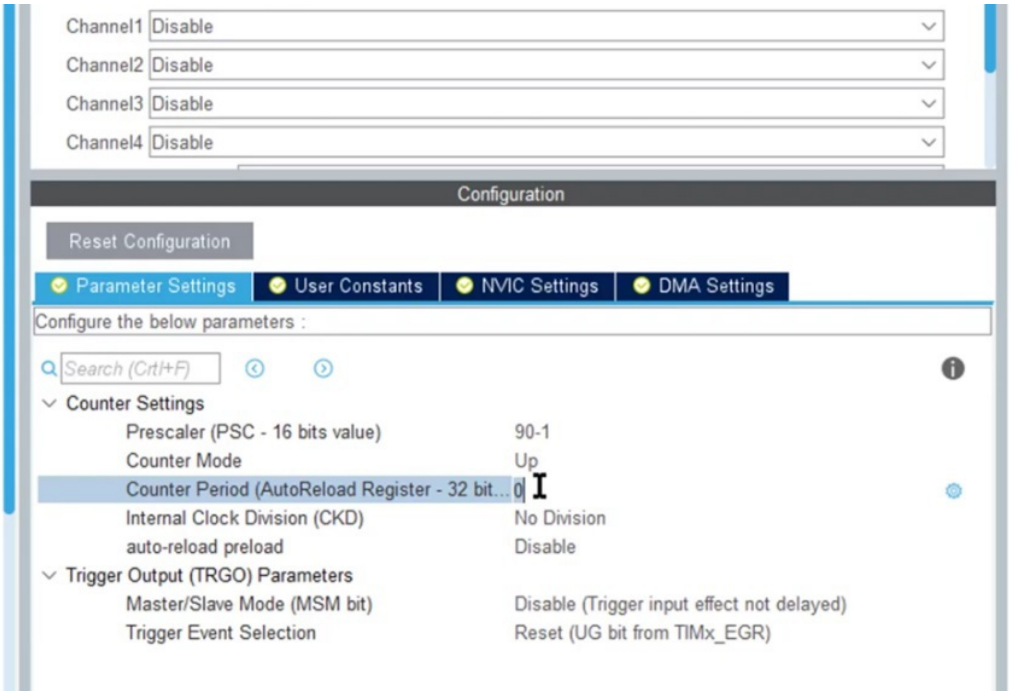
e: For right-aligned 12-bit resolution: DAC_MaxDigitalValue = 0xFFF
For right-aligned 8-bit resolution: DAC_MaxDigitalValue = 0xFF

25 Doc ID16895 Rev 3

Let's start programming now first of all I am creating a variable 0.2 is going to be the output voltage that is d AC output value. We need to create another variable to store the respective digital value inside the main function first start the D AC. In this while loop, I am going to do the conversion from voltage to the digital value. To do so, we need to make do our as the subject of this formula. Once done, we need to set this value to the DA see and then increment value this process will continue with some delay remember that the value can't be more than reference voltage and that is 3.3 volts looks like we got some errors let's compile it again.



So, the code compiled successfully time to flash it to the board just create a debug configuration observe the reading on the voltmeter and on the oscilloscope the voltage output from the pin is changing every 750 milliseconds and it doesn't go higher than three volts let's increase the time delay so that you can observe the change properly here we go the voltage is increasing every two seconds now so, this was the basic idea about how to use the D AC and STM 32. Now, we will create a sine wave using the same this process is mentioned in the same note but before this we need to make some changes to our setup first we need to select a trigger timer I am using timer to also we will use DMA for this entire process this will keep the CPU free Next, go to the timer setting that you chose. This setup will decide the frequency of the wave and therefore is critical. I am using timer two which is connected to the APB one clock and this is running at 90 megahertz. I am selecting the prescaler as 90 This will reduce the timer clock to one megahertz using a Rs 100 will further divide the clock to 10 kilo hertz I will explain this particular setup in a while.



Select Update events here. This is it for the setup. Let's generate the code now. include math dot h, so that we can use the sine function I am creating an array to store the digital values of 100 samples of sine function we need pi value this function here will do the conversion let's take a look at the PDF again this is the formula that we are going to use number of samples is going to be 100 in our case f f f is for the 12 bit resolution now in the main function, we first need to start the timer and then start the DA C with D M A. Now let's talk about the frequency of this wave as I mentioned he APB clock is at 90 megahertz and using a prescaler of 90 We'll divide that clock by 90 making it one megahertz using the ARR value 100 will further divide the clock by 100 making it 10,000 hertz according to this application note the frequency of the sine wave is equal to time a frequency divided by number of samples. In our case we have to further divide this value by 100 Because we are doing 100 samples which gives us a frequency of 100 hertz This is the frequency that I am expecting for the sine wave to have. Oh sorry, I forgot to include the function to convert the values to digital form Let's build and run the code you can see the sine wave being produced with the frequency of 100 hertz let's increase the frequency now. I am dividing it by 10 that will increase the frequency by multiple of 10.

EXAMPLE DUMMY CODE

Here's an example code snippet for generating a sine wave using the STM32's DAC (Digital-to-Analog Converter) module, HAL (Hardware Abstraction Layer) library, and CubeIDE. This example generates a sine wave using a lookup table and outputs it through one of the DAC channels. Please note that this example assumes you have configured your hardware and peripherals properly in CubeIDE.

```
#include "main.h"
#include "math.h"

#define DAC_CHANNEL  DAC_CHANNEL_1
#define DAC_TRIGGER  DAC_TRIGGER_NONE

#define SINE_WAVE_SAMPLES  128

DAC_HandleTypeDef hdac;

// Sine wave lookup table
const uint16_t sineWave[SINE_WAVE_SAMPLES] = {
    2048, 2447, 2831, 3185, 3495, 3750, 3939, 4056,
    4095, 4056, 3939, 3750, 3495, 3185, 2831, 2447,
    2048, 1648, 1264, 910, 600, 345, 156, 39,
    0, 39, 156, 345, 600, 910, 1264, 1648,
    2048
};

void SystemClock_Config(void);
```

```
static void MX_DAC_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_DAC_Init();

  while (1)
  {
    for (uint16_t i = 0; i < SINE_WAVE_SAMPLES; i++)
    {
      HAL_DAC_SetValue(&hdac, DAC_CHANNEL,
DAC_ALIGN_12B_R, sineWave[i]);
      HAL_Delay(10); // Adjust delay for desired
frequency
    }
  }
}

void SystemClock_Config(void)
{
  // Configure the system clock here
}

static void MX_DAC_Init(void)
{
  hdac.Instance = DAC;
  if (HAL_DAC_Init(&hdac) != HAL_OK)
  {
    Error_Handler();
  }
```

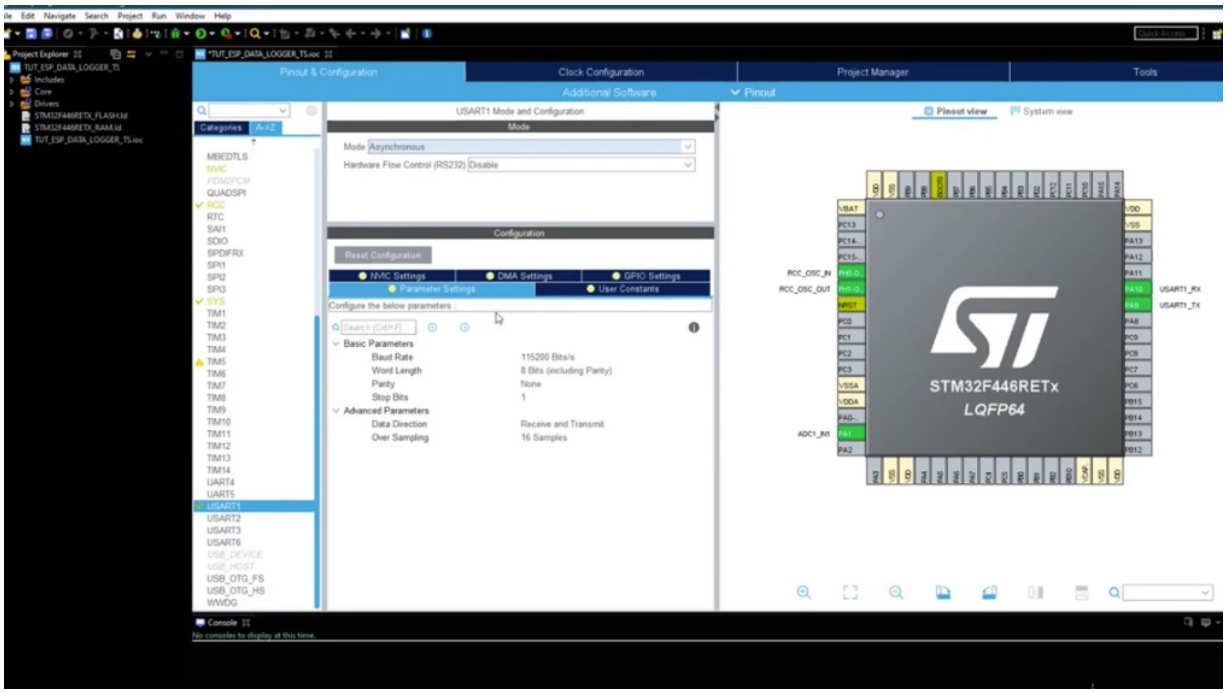
```
  DAC_ChannelConfTypeDef sConfig = {0};
  sConfig.DAC_SampleAndHold =
DAC_SAMPLEANDHOLD_DISABLE;
  sConfig.DAC_Trigger = DAC_TRIGGER;
  sConfig.DAC_HighFrequency =
DAC_HIGH_FREQUENCY_INTERFACE_MODE_DISABL
E;
  sConfig.DAC_OutputBuffer =
DAC_OUTPUTBUFFER_ENABLE;
  if (HAL_DAC_ConfigChannel(&hdac, &sConfig,
DAC_CHANNEL) != HAL_OK)
  {
    Error_Handler();
  }
}
```

In this code, a sine wave lookup table is defined, containing 128 samples for a half cycle of a sine wave. The code then loops through these samples and sets the DAC output value accordingly. The delay inside the loop controls the frequency of the generated sine wave. Adjust the delay value as needed to achieve the desired frequency.

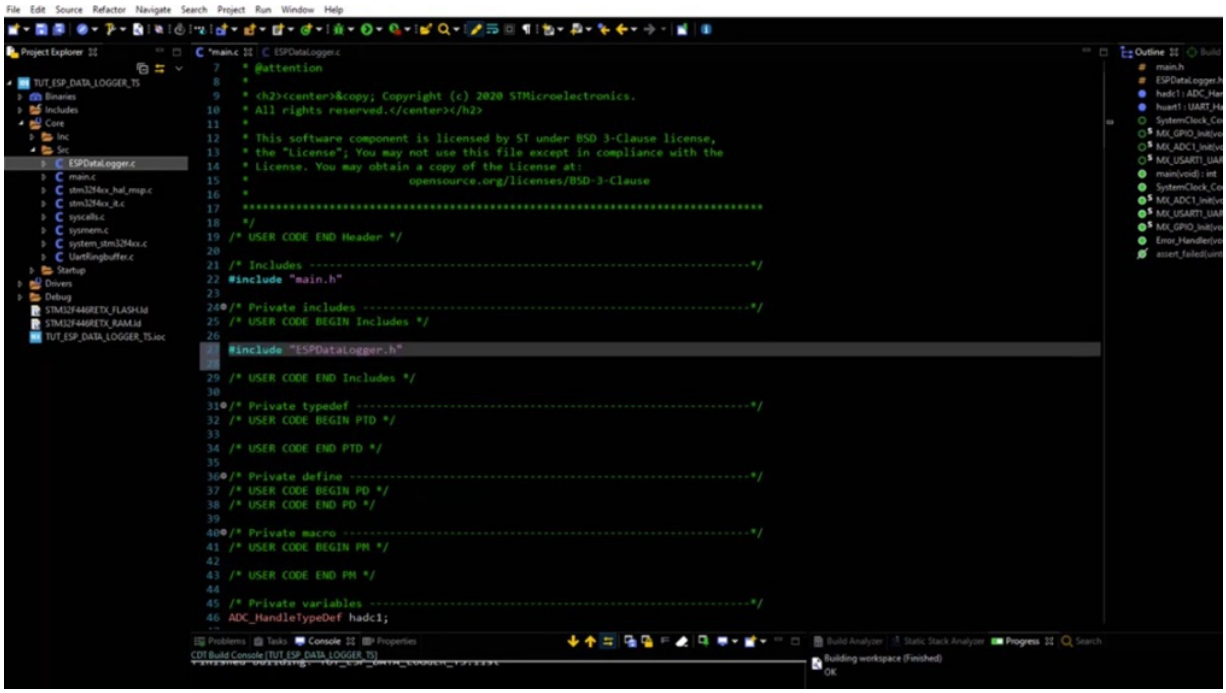
Please note that this example assumes you've configured the DAC peripheral and corresponding GPIO pins correctly in the CubeIDE interface. Also, remember to configure the system clock appropriately using the SystemClock_Config function.

DATA LOGGER USING STM32 ESP8266 THINGSPEAK

We will see how to use STM 32 to log data to the thingspeak server using ESP H 266. Let's first start by creating the project in cube ID I am using nuclear 446 R E for this project give some name and click finish here is our cube MX, first I am selecting external crystal for the clock next, I am selecting an ADC which will be used to read the potentiometer data changing the resolution to 10 bits even if you don't enable continuous conversion, it should be fine because I am going to use the pole for conversion method. So continuous conversion is not required. Leave everything else to default now, enable the UART with the interrupt. This is where the ESP 8266 will be connected. Let's set up the clock now.



I have eight megahertz Crystal and I want the system to run at maximum frequency click Save to generate the project let's include the libraries first. I am going to use the ring buffer library and data logger library. Put the source and the header files in the respective folders click Refresh and you can see the files here in our project. Let's take a look at the ring buffer dot c file. Here you can change the URL that you are using copy this function from here and paste it in the interrupt dot c file also we need to modify UART handler and comment out the original one. Let's take a look at the data logger dot c file now ESP and its text the SSID and password as the parameter ESP send data sends the data to a particular filed label. The parameters are self explanatory here. Here you have to give the label of the field such as either one or two or three etc. This is the one I am going to use. It sends data to multiple fields at once. Let's first go to the things register yourself or log in. Click on channels then click New Channel give some name to this channel. I am going to use four fields in this channel we can modify these names also. I will quickly modify others too. You can see the modified names.



Remember that field label number is the one that we need while sending data Let's start include the data logger dot h i will write a function to get the ADC value this includes start ADC pole for

conversion, get the value and stop ADC. Let's define the variables that we are going to use also a buffer to store the four values inside the main function. First of all initialize the ESP with the SSID and password now in the while loop, we will get the value of ADC, then convert this value to the voltage and at last increment the count variable now, we will store these values in the respective position in value buffer and send these values to the server. Now here we need the API key go to your channel, click API key and copyright API key I have four number of fields so, enter four here and pass the value buffer give a delay of 15 seconds minimum things speed server will not update the data if we send it within 15 seconds let's build and debug our program I am adding the variables in the live expression let's run it now. You can see the first entry in the thingspeak Now I am rotating the potentiometer so the ADC value will change you can see the new entries now also note the values of count and count times two. Everything is as par with the values we have in the live expression this data will keep updating every 15 seconds.

EXAMPLE DUMMY CODE

Below is an example code for creating a data logger using an STM32 microcontroller and an ESP8266 module to send data to ThingSpeak. This example assumes that you have set up the STM32 microcontroller, ESP8266, and the necessary sensors for data measurement.

```
#include "stm32f4xx_hal.h"
#include <string.h>
#include <stdio.h>

UART_HandleTypeDef huart2;  // UART handle for ESP8266 communication

// Configure your ThingSpeak channel and API key
#define THINGSPEAK_CHANNEL_ID   "YOUR_CHANNEL_ID"
#define THINGSPEAK_API_KEY      "YOUR_API_KEY"

// Function to send data to ThingSpeak
void SendDataToThingSpeak(float temperature, float humidity)
{
  char buffer[128];
```

```
  sprintf(buffer, "GET /update?
api_key=%s&field1=%.2f&field2=%.2f\r\n",
THINGSPEAK_API_KEY, temperature, humidity);

  HAL_UART_Transmit(&huart2, (uint8_t*)buffer,
strlen(buffer), HAL_MAX_DELAY);
}

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_USART2_UART_Init();

  // Initialize sensors and other peripherals here

  while (1)
  {
    // Read temperature and humidity from sensors
    float temperature = ReadTemperature();
    float humidity = ReadHumidity();

    // Send data to ThingSpeak
    SendDataToThingSpeak(temperature, humidity);

    HAL_Delay(10000);  // Send data every 10
seconds
  }
}

void SystemClock_Config(void)
{
```

```
  // Configure the system clock here
}

void MX_USART2_UART_Init(void)
{
  // Initialize USART2 peripheral for ESP8266 communication here
}

void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
  // Initialize UART MSP (GPIO, NVIC, etc.) here
}

void HAL_UART_MspDeInit(UART_HandleTypeDef* huart)
{
  // Deinitialize UART MSP here
}

// Implement functions to read temperature and humidity from sensors
// and other necessary hardware initialization as needed
```

This code demonstrates a basic data logging functionality where it reads temperature and humidity data from sensors (you need to implement the functions ReadTemperature() and ReadHumidity()), then sends the data to

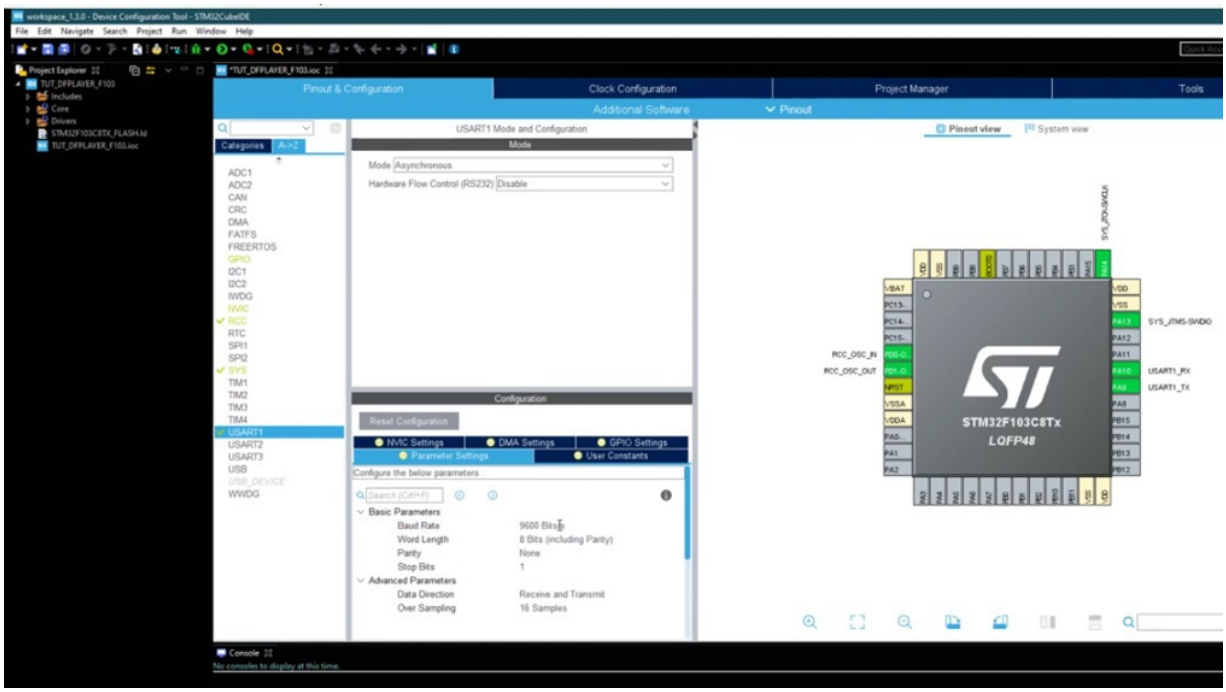
ThingSpeak using the ESP8266 module through UART communication. Remember to initialize and configure your UART communication with the ESP8266 properly.

Before using this code, make sure to replace "YOUR_CHANNEL_ID" and "YOUR_API_KEY" with your actual ThingSpeak channel ID and API key.

Please note that this is a simplified example, and you may need to adapt it to your specific STM32 microcontroller, ESP8266 module, sensors, and overall project requirements. Additionally, ensure that you have the necessary libraries and drivers set up in your STM32 project.

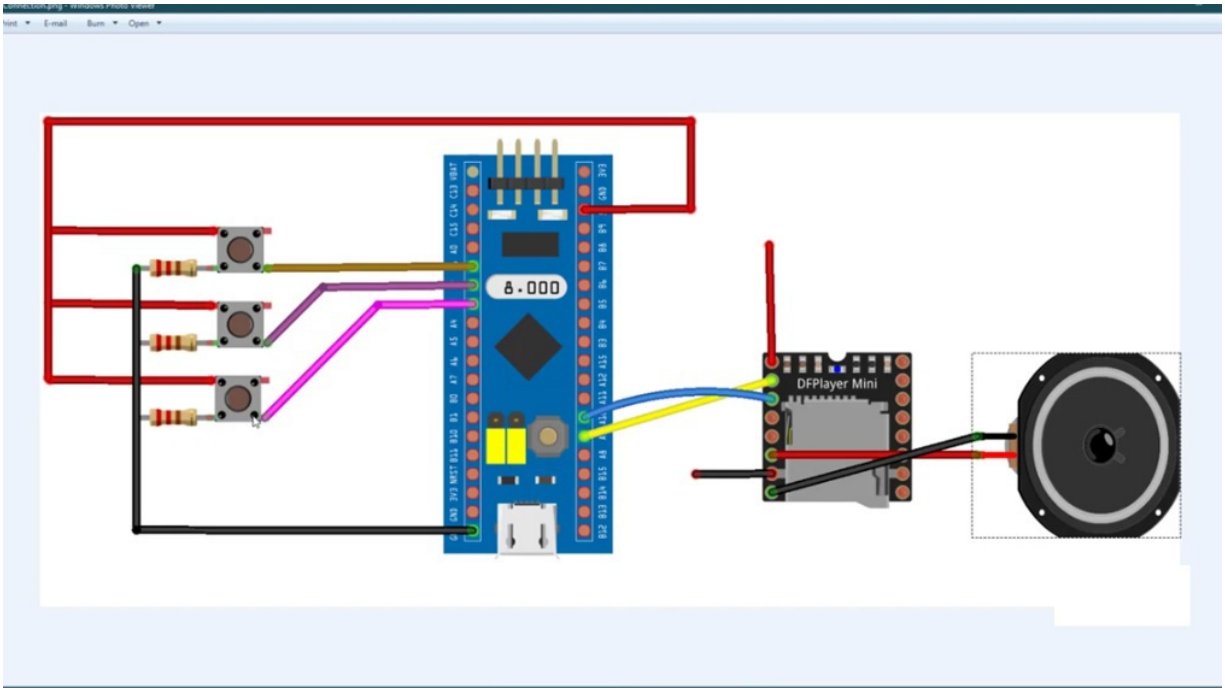
DF PLAYER MINI AND STM32

D F player works with very simple commands which needs to be transmitted using UART let's start by creating a project in cube Id first I am using STM 32 F 103 For this purpose give some name to the project and click finish here is our cube MX I am selecting external crystal for the clock select serial wire debug I am using you f1 To send commands make sure the baud rate is 9600 bits per second I am going to use three buttons to control the playlist.



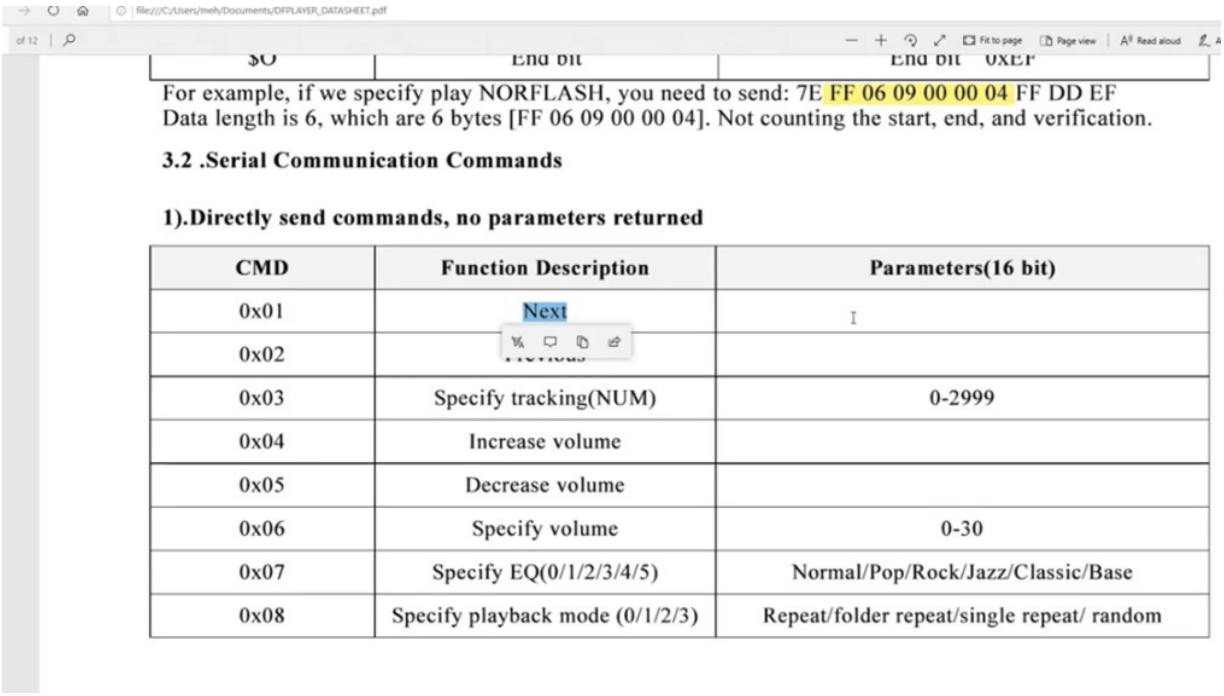
That's why I am selecting these pins as input. Let's go to the clock setup now I have eight megahertz external Crystal and I want the controller to run at 72 mega hertz click Save to generate the project first of all we need to copy the library files. So copy the C file to source and header file to include directory click Refresh and you

can see the files in the project. Let's take a look at the data sheet first. As you can see here, it supports fat 16 Fat 32 formats for the SD card there are 30 levels of volume and other things are there too you can read them as mentioned here, the default baud rate is 9600. And now the very important thing the standby current is 20 milliamp pairs I will tell you more about this in a while this is the diagram of DF player mini we are going to use only this side let's take a look connection now. I am using external power supply for the DF player. RX pin from the DF is connected to Tx pin of the controller which is pin PA nine and Tx pin from the DF is connected to Rx pin of the controller which is pin p a 10. Speaker One and speaker two pins are connected to the positive and negative of the speaker these three buttons are connected to pin PA one pa two and Pa three This button is for previous selection. This one is for pause or play and this is for next song selection.



Coming back to the external power supply. While testing I found that DF player needs around 25 milliamp pair of current during the song is playing F 103 was not able to supply this much current even with the USB connected. So I switched to the external power supply. Even in the datasheet it's mentioned that it takes 20 milli amperes of idle current. Let's go back to our main file we need to include the

library in our main file let's see the DF player dot c file now. Here you can change the URL According to the one UI using the source is defined as from the SD card I will get back to this in a while define the pins that you are using for the buttons let's see these defines now stalked by two zero cross seven e and Vita zero cross E FF, the command sent to the DF player have a sequence and that is staff byte version command length command feedback Parameter one Parameter 216 Vic checksum and the end bite command length is six as mentioned here, because it does not count the start checksum and the end bytes we are not going to use feedback. So, I have defined it zero here send command function will send the commands to the DF player it first calculates the checksum the command sequence is same here too we have staffed by its version command length command feedback Parameter one Parameter to check some high byte lower byte and the end byte and then it sends this array to the UART DF in it is to initialize the player and it takes volume as the parameter basically we send zero cross three F with the parameters you can see here this is the commands to initialize the player and we need to send the parameters also if you scroll down the example is provided in the datasheet This is the sequence of commands need to be sent for the TF card we have command feedback parameter one and parameter to the parameter two is zero cross 02 for the TF card that's why I have defined it as source here you can change it and see the other parameters in the data sheet.



$O	End bit	End bit 0xEF

For example, if we specify play NORFLASH, you need to send: 7E FF 06 09 00 00 04 FF DD EF
Data length is 6, which are 6 bytes [FF 06 09 00 00 04]. Not counting the start, end, and verification.

3.2 .Serial Communication Commands

1).Directly send commands, no parameters returned

CMD	Function Description	Parameters(16 bit)
0x01	Next	
0x02	Previous	
0x03	Specify tracking(NUM)	0-2999
0x04	Increase volume	
0x05	Decrease volume	
0x06	Specify volume	0-30
0x07	Specify EQ(0/1/2/3/4/5)	Normal/Pop/Rock/Jazz/Classic/Base
0x08	Specify playback mode (0/1/2/3)	Repeat/folder repeat/single repeat/ random

Then it sets the volume zero cross 06 is the command to set the volume along with the parameter which acts as a level of the volume the F next function is to play the next song this command does not need any parameter. This is commands to play previous song we have pause and play also. Check key function checks if the key is pressed. If the song is playing, then it will be paused or if it is paused then it will resume the play. Then we have a button for previous and a button for next song. Now let's start programming. Initialize the def player. I am keeping the volume as 10 stop the play from the first song in the while loop. We will keep checking for the button pressed. Let's build and debug this before starting I want to show you my SD card. There are six mp3 files. You should also number them like I have done it here.

EXAMPLE DUMMY CODE

Below is an example code for interfacing a DFPlayer Mini MP3 module with an STM32 microcontroller. This code demonstrates how to control the DFPlayer Mini to play specific MP3 files using UART communication. Make sure you have the necessary connections between the STM32 and the DFPlayer Mini (TX-RX and RX-TX).

C

```
#include "stm32f4xx_hal.h"

UART_HandleTypeDef huart2; // UART handle for DFPlayer Mini communication

void SystemClock_Config(void);
static void MX_USART2_UART_Init(void);

void SendCommand(uint8_t cmd, uint8_t param1, uint8_t param2)
{
    uint8_t buffer[10];
    buffer[0] = 0x7E; // Start byte
    buffer[1] = 0xFF; // Version
    buffer[2] = 0x06; // Command length
```

```
    buffer[3] = cmd;  // Command
    buffer[4] = 0x00; // Feedback
    buffer[5] = param1;
    buffer[6] = param2;
    buffer[7] = 0xEF; // End byte

    HAL_UART_Transmit(&huart2, buffer, 8,
HAL_MAX_DELAY);
}

void PlayTrack(uint16_t trackNumber)
{
    SendCommand(0x03, highByte(trackNumber),
lowByte(trackNumber));
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_USART2_UART_Init();

    while (1)
    {
        // Play a specific track when a condition is
met
        if (/* your condition here */) {
            PlayTrack(1); // Play track number 1
```

```
            HAL_Delay(2000); // Delay to avoid
continuous triggering
        }
    }
}

void SystemClock_Config(void)
{
    // Configure the system clock here
}

static void MX_USART2_UART_Init(void)
{
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 9600;
    huart2.Init.WordLength =
UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl =
UART_HWCONTROL_NONE;
    huart2.Init.OverSampling =
UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
```

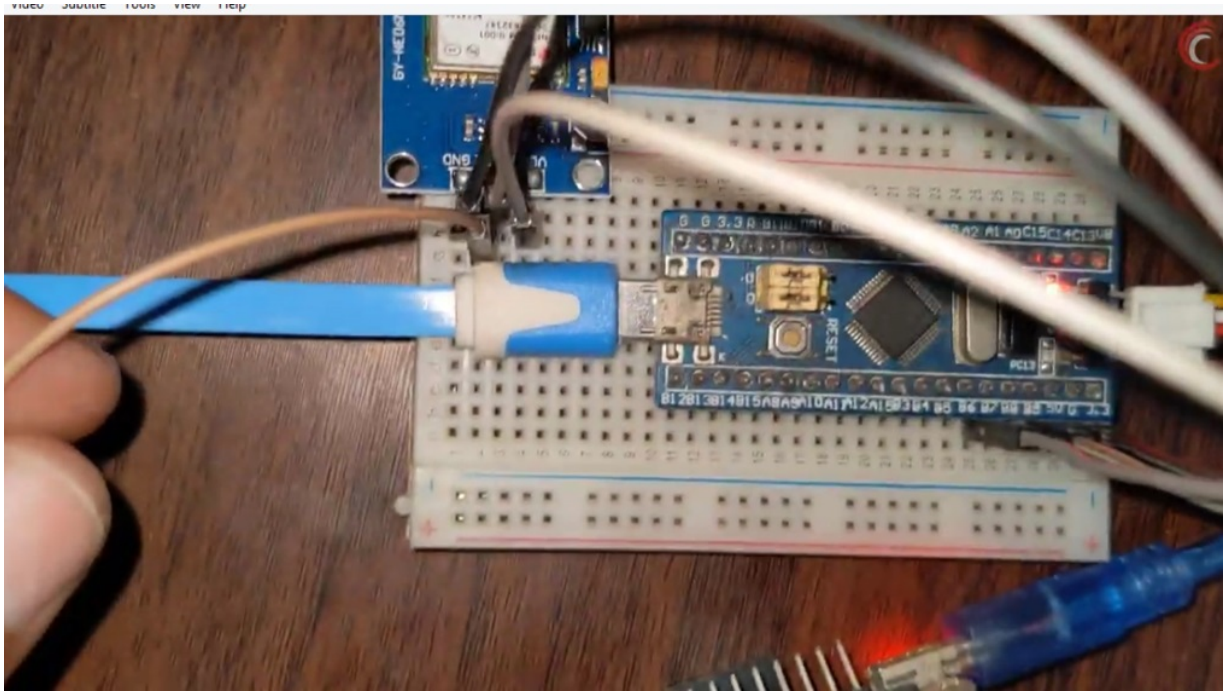
```
}

void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
    // Initialize UART MSP (GPIO, NVIC, etc.) here
}

void HAL_UART_MspDeInit(UART_HandleTypeDef* huart)
{
    // Deinitialize UART MSP here
}

uint8_t lowByte(uint16_t x) {
    return x & 0xFF;
}

uint8_t highByte(uint16_t x) {
    return (x >> 8) & 0xFF;
}
```

Please note that this is a simplified example and you may need to adapt it to your specific STM32 microcontroller model, hardware connections, and requirements. The provided SendCommand function prepares and sends the necessary commands to the DFPlayer Mini over UART.

Ensure that you have the UART communication properly configured and connected between the STM32 and the DFPlayer Mini. Also, replace the condition in the loop with the logic that determines when to play the desired track.

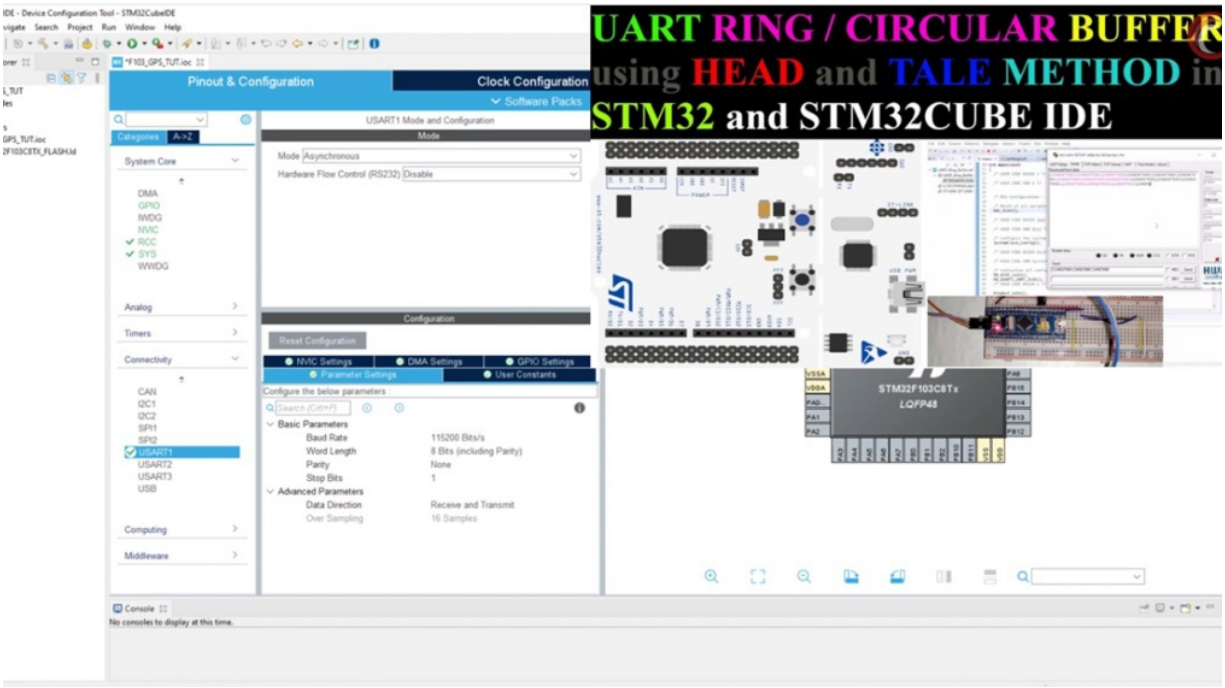
GPS MODULE AND STM32 NEO 6M GET COORDINATES DATE TIME SPEED

I brought to you another project on the GPS module where we will interface the module with the STM 32 micro controller. As many of you might be using the Neo GPS modules, I have also switched to Neo six M for this project, but it does not matter. As I have mentioned before, the output from all the GPS modules is pretty much the same. They all outputs the data in the NME a format and that is why you can use this method for pretty much every GPS module out there. If you remember from my previous project, this is the output of the GPS module in the NME of format. Before starting this project, I would want you guys to get familiar with the ring buffer code. If you've got the ring buffer code to work, you can go ahead with this one or else first try to make that work.

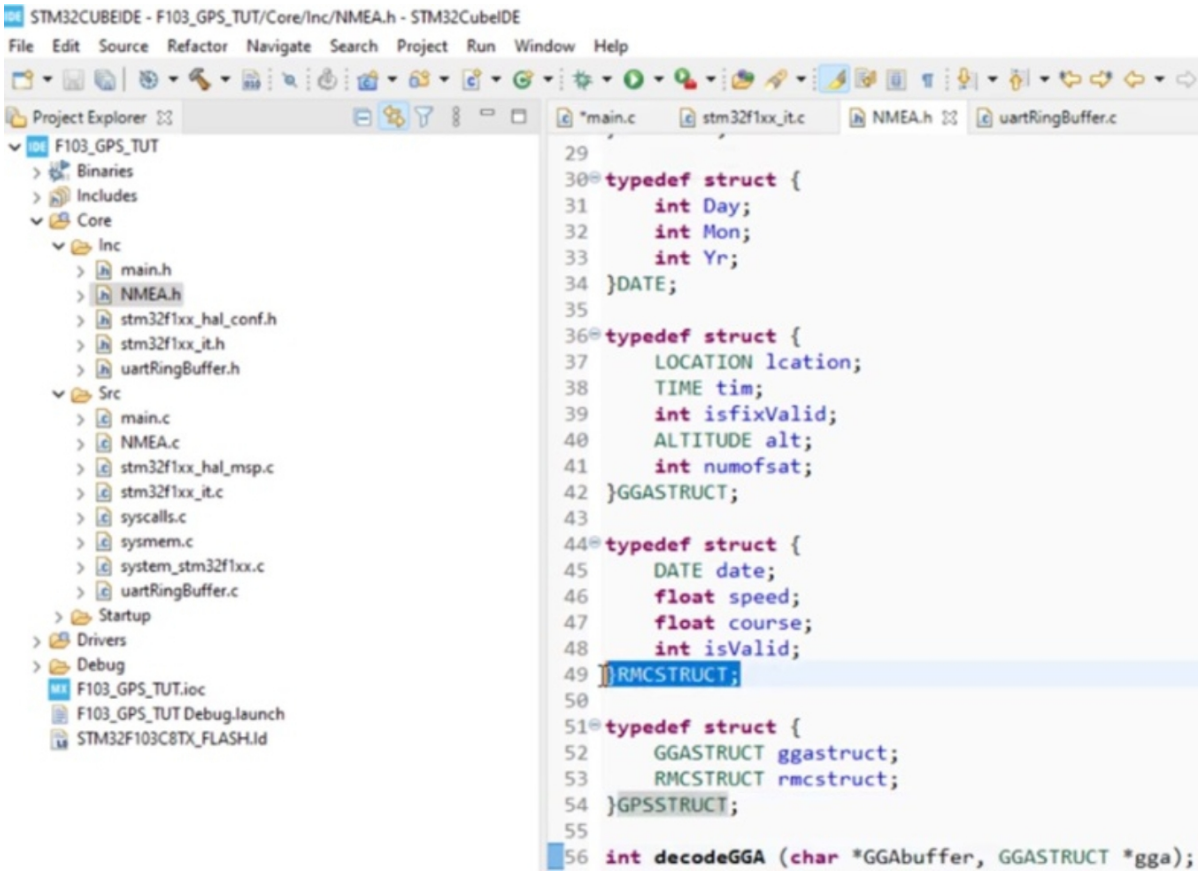


Like I said I have switched to Neo six M and I have connected it with the ft 232 UART TTL device you can see the connection the LED on the six M is blinking this means that we have the fix on the signal the TX pin from the six M is connected to the RX pin of the ft 232. It's also connected to the RX pin of the UART one of the blue pill but for now let's focus on the ft 232. The module is powered with five volts supply from the blue pill you can see the data it is transmitting via the UART it's basically the same format as the previous GPS module. Now I am going to connect the module to 3.3 volts supply and we will see the UART output you can see the output if you are getting something like this make sure you provide proper voltage supply to the module. Here I have switched back to five volts and the transmission begins again. Let's keep these things in mind and create a new STM 32 project I am going to use STM 32 F 103 controller for this give some name to the project and click Finish enable the external high speed crystal for the clock. Also we need to enable the serial wire debug. There is eight megahertz crystal onboard and let's run the system at maximum 72 megahertz clock All right, now we will enable the UART since the data incoming is of uncertain length, we need to use some circular or ring buffer for the

UART I already did a project on it few years ago and if you have made it work, you are good to go with this tutorial.

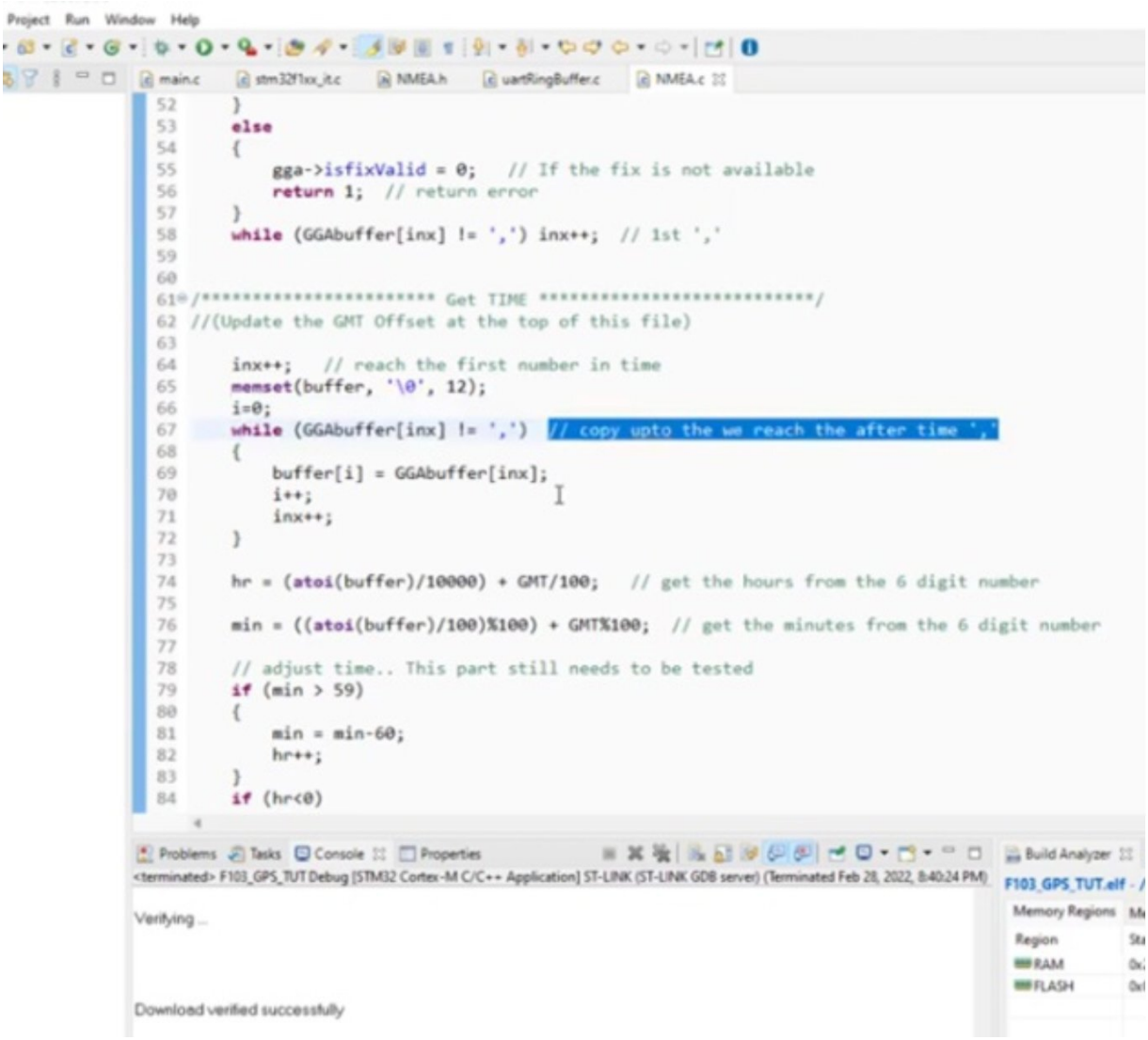


Also, I would recommend that you watch that project first. If you haven't did it yet, as this code will heavily depend on that project. Let's enable the UART interrupts now. Nero six M works with 9600 baud by default. I am relocating the UART pins for my convenience. PV seven is the RX pin and PV six is the TX pin. This is it click Save to generate the project this here is the ring buffer code that I am going to use. Here are the steps provided for this particular code to work. I have already downloaded the ringbuffer files and now we will copy them into the project along with the NN e files I have created specifically for this tutorial put the C files into the source directory and header files into the include directory let's see the ring buffer dot c file, we need to put this in the interrupt file first include the UART ring buffer header file. Now define the UART is our function as extern we need to also define the timeout as external variable.



This is needed for the timeout feature to work inside the SysTick interrupt handler, decrease the timeout value. And finally, inside the UART interrupt handler, replace the interrupt handler with UART is our function. Make sure you update the correct UART instance you are using. I have modified the RX buffer size to 512 Bytes as there is large data in the incoming buffer. Other than this, the ring buffer code is pretty much untouched. Let's include the UART ring buffer header file also include the n n e a header file in the main file. This is how the nm e a header file looks like. Notice that it only have two functions to decode G G, A and R MC. If you have watched my previous project on GPS, you will know what these terms means. These two gives enough information for now. All right, let's initialize the ring buffer now and build the code to check for any errors. If there are no errors, we can go ahead let's debug the code once. This is to make sure if we are receiving the data in the RX buffer or not, let's add the RX buffer to the live expression. Run the code now. Let's see the buffer itself. Here you can see we got the RM C

string or if you get any other string also, it's all right. The main thing that we are checking is if we are able to receive data or not. All right, so the RX buffer is receiving data and we are good to go. Let's define an array to store the G G A data and another one for the RMC data. Next we will define this GPS structure in the main file. Actually the decode functions take respective structures as the parameters but GPS structure contains both of them and therefore we will only define it give some delay after initializing the ring buffer. Now we will wait for the G G a string in the incoming buffer if the string is detected, the function will return one. This function also includes the timeout feature, just in case if the string is not detected, it will timeout and the execution will continue to the next statement. If the string is detected in the incoming buffer, we will copy up to the star Mark and we will save this data in the GG a buffer we created earlier. So basically it will wait for this GGA and Once found, it will copy up to this star Mark. This entire data in the middle will be saved in our G G a buffer which we will process later.

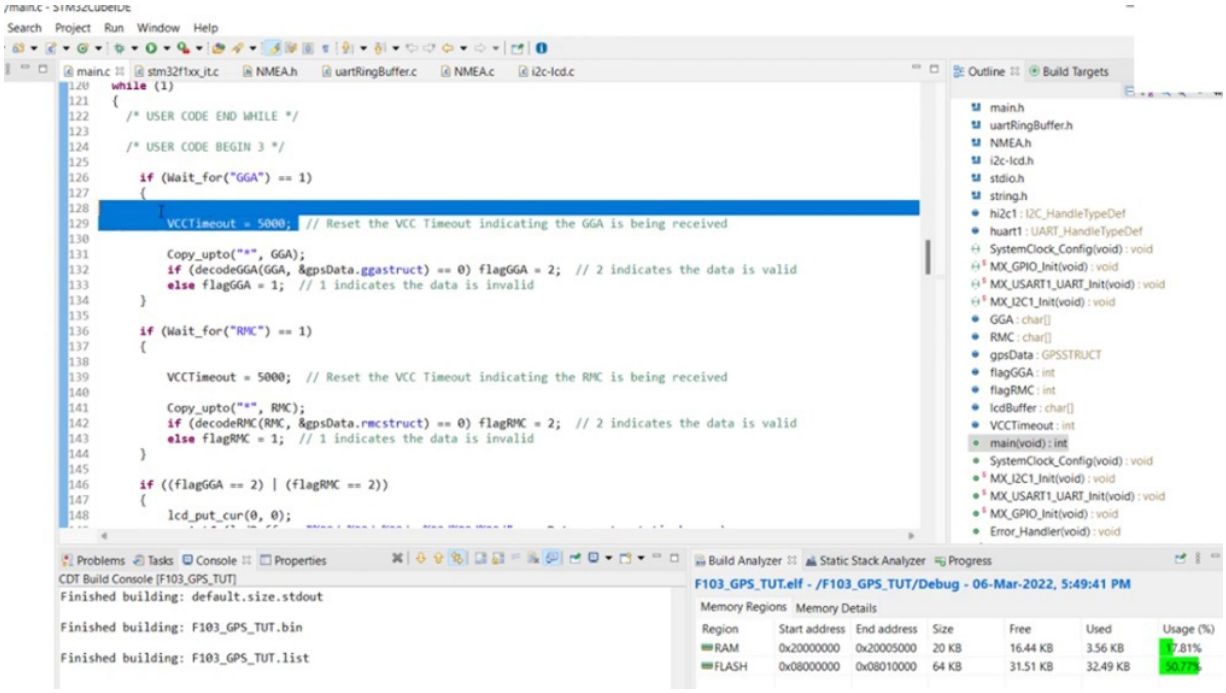


After copying the data. We will call the decode G G a function. The first parameter is the GG a buffer where the data was saved. And the second parameter is the G G A structure. Here we will pass the address of the G G a struct element of the GPA structure. Let's build the code to check for any errors. All right, we are good to go. So let's debug this much part first, add the GPS data in the live expression, G, G, A structure have location, time, altitude, and the number of satellites, it also have the means to check if the fix is valid or not. We have time here, then the coordinates. This one indicates that the fix is valid for is the number of satellites. This is the altitude. All right, let's run the code we got the hard fault. Let's put a breakpoint inside the wait for function and run it two three times. Now remove the

breakpoint and let it run freely. Here you can see is fixed valid is set to one. And if you see the coordinates, they are pretty much accurate time is also correct. For the time conversion, I have included this variable and you need to edit it in the code. The time part is still needs to be tested. So if you are getting wrong time, let me know in the comments. So everything is working well. And we were able to decode the GG a parameter from the NN e a data let's write the same code for the RMC parameter also this needs to be decode our MC. All right, let's debug the code you can see all the parameters are pretty much working fine. The date today is 28 February, things are working fine. And the GPS data we are getting is also accurate. Let me explain the code so that you guys can decode other things also. Let's see the decoding of the G G A. We first script six commerce. If you check the collected data, you can see this one comes after the six commas. We look at the data at this position. If the value is one, two or six, that means the fix is valid. For any other value, the fix will be invalid and we will return with some error code. By the way, this function returns zero on success and other numbers based on where the error returned. If the fix is valid, we will continue with the decoding. And first of all, we will copy the data between first and second commas. This data represents the time in UTC. Next step would be to extract the time and adjust it according to the offset. This part is still untested. And if you get any error in time, let me know in the comments. next task is to get the latitude. Since we have already hit the comma in the previous copying, we will increment the index to reach the first digit of the latitude and basically here now we will copy this entire data into our buffer. After copying this data, we will convert the string into the number and do the necessary calculation to extract the latitudes from the data. We will increment the index again and check for the north or south parameter. Similarly, we will get the longitude data, convert it into the number and extract the longitudes from it. Next, we will skip the position fixed checking since we already checked it and directly calculate the number of satellites being used. And finally get the altitude data and convert it to number format. Similarly, we can decode the RMC data and extract the date, speed and other things from it. I will comment this code before uploading it so that

you can understand it better. Now I know there will be some of you who will still ask about how to display it on the LCD. So I have decided to include that part in the project itself. I am going to use the eye to see to connect with the LCD 16 02. This is why I have enabled the eye to see one. Everything is set to default here, you need to include the eye to see LCD libraries in the project. Now in the main file, if the GG a string is detected, we will set the V cc timeout to 5000. This V c c timeout is basically for checking if we are receiving the data or not. You need to define it externally in the interrupt file, just like we did for the timeout. Also similar to timeout, we need to decrease its counter in the SysTick interrupt handler. This is where it's defined. Also note that I have defined the flags for the GGA and our MC. These flags will be set if we receive the valid data from the GPS module. Also, there is a LCD buffer which will contain the character data respective to numerical values. All right after the GG a string is detected, we will reset the VCC timeout and copy the GG a data into the buffer. If the decoding results in a success, we will set the G G a flag to to or else set it to one indicating that the string was received, but the fix is still invalid. similar steps for the RMC data also. Now if either of the received data is valid, we will put the LCD cursor to 00 then use S printf. To convert the numerical data into the characters and save it in the LCD buffer. The first half of this will be hours, minutes and seconds from the GGS structure. And the later half will be the date from the RMC structure once the converted data is saved into the LCD buffer, we will send it to the LCD, then reset the buffer, set the cursor to the second row and convert the location data. This point to f is to convert the float data in two decimal places, which is basically the latitude. The C here indicates the character which indicates whether it's north or south. Similarly, we have for the longitude and the character for east or west. Finally, we will send this data to the LCD. Now if the either of the flags is one indicating that the string was received, but the fix was invalid, we will print the respective information on the display. Notice here that I am not clearing the display, but instead I choose to go with the spaces. This is to ensure that every time we are waiting for the fix, the screen shouldn't feel like it's refreshing. Finally, if the VCC timeout is zero, that means the GG a or RMC

strings were not detected in the incoming buffer. This could only mean that either the voltage is insufficient, or there is some problem with the connection between the module and the microcontroller. So here we will print the respective information on the display again. Here you can see I have taken some pictures for each case.



This is when I provided 3.3 volts to the module. This is when the voltage was fine, but the fix was invalid. And this is when everything was all right. So this is it for the project. I hope you understood the concept well. Decoding the NME A could be better. And I know there are some libraries available to just pass this data. But I like to use my own code for whatever I can. And that's the only reason I wrote this.

EXAMPLE DUMMY CODE

Here's an example code that demonstrates how to interface a GPS module (specifically the NEO-6M) with an STM32 microcontroller to obtain coordinates, date, time, and speed information. This code assumes you have the GPS module connected to the STM32's UART interface.

C

```
#include "stm32f4xx_hal.h"
#include <stdio.h>
#include <string.h>

UART_HandleTypeDef huart2;  // UART handle for GPS communication

void SystemClock_Config(void);
static void MX_USART2_UART_Init(void);

void ProcessGPSData(char* gpsData)
{
    char *token;

    // Search for specific GPS NMEA sentences
    if (strstr(gpsData, "$GPGGA")) {
```

```
        // Parse GPGGA sentence for coordinates
and fix status
        token = strtok(gpsData, ",");
        for (int i = 0; i < 2; i++) {
            token = strtok(NULL, ",");
        }
        float latitude = atof(token);

        token = strtok(NULL, ",");
        if (strcmp(token, "S") == 0) {
            latitude = -latitude;
        }

        token = strtok(NULL, ",");
        float longitude = atof(token);

        token = strtok(NULL, ",");
        if (strcmp(token, "W") == 0) {
            longitude = -longitude;
        }

        // Use latitude and longitude

        // ... (add your code here)
    }
    else if (strstr(gpsData, "$GPRMC")) {
        // Parse GPRMC sentence for date, time, and
speed
        token = strtok(gpsData, ",");
```

```
        for (int i = 0; i < 1; i++) {
            token = strtok(NULL, ",");
        }
        char time[10];
        strcpy(time, token);

        token = strtok(NULL, ",");
        char fixStatus = token[0];

        token = strtok(NULL, ",");
        float speed = atof(token);

        token = strtok(NULL, ",");
        char date[10];
        strcpy(date, token);

        // Use time, fixStatus, speed, and date

        // ... (add your code here)
    }
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_USART2_UART_Init();

    char gpsBuffer[256];
    int bufferIndex = 0;
```

```
    while (1)
    {
        char receivedChar;
        if (HAL_UART_Receive(&huart2, (uint8_t
*)&receivedChar, 1, HAL_MAX_DELAY) ==
HAL_OK) {
            if (receivedChar == '\n' || bufferIndex >=
sizeof(gpsBuffer) - 1) {
                gpsBuffer[bufferIndex] = '\0';
                ProcessGPSData(gpsBuffer);
                bufferIndex = 0;
            } else {
                gpsBuffer[bufferIndex++] =
receivedChar;
            }
        }
    }
}

void SystemClock_Config(void)
{
    // Configure the system clock here
}

static void MX_USART2_UART_Init(void)
{
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 9600;
```

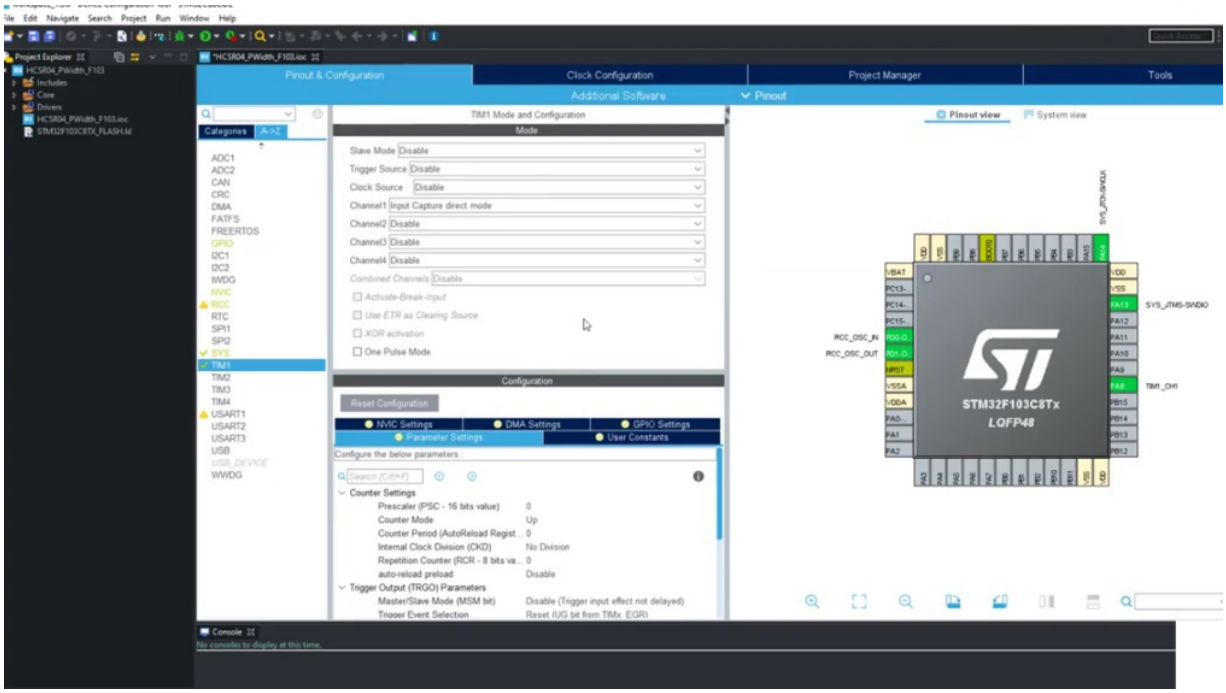
```
    huart2.Init.WordLength =
UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_RX;
    huart2.Init.HwFlowCtl =
UART_HWCONTROL_NONE;
    huart2.Init.OverSampling =
UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
}

void HAL_UART_MspInit(UART_HandleTypeDef*
huart)
{
    // Initialize UART MSP (GPIO, NVIC, etc.) here
}

void
HAL_UART_MspDeInit(UART_HandleTypeDef*
huart)
{
    // Deinitialize UART MSP here
}
```

This example code listens to the UART2 interface for NMEA sentences transmitted by the GPS module. When a full sentence is received (indicated by the newline character \n or when the buffer size is reached), it processes the sentence to extract the desired information. The code specifically processes GPGGA and GPRMC sentences to obtain latitude, longitude, date, time, and speed information.

Keep in mind that this code is a basic example and might need further adaptation based on your specific requirements and STM32 microcontroller model. Additionally, ensure that the UART communication is correctly configured and the GPS module is properly connected to the STM32 microcontroller.

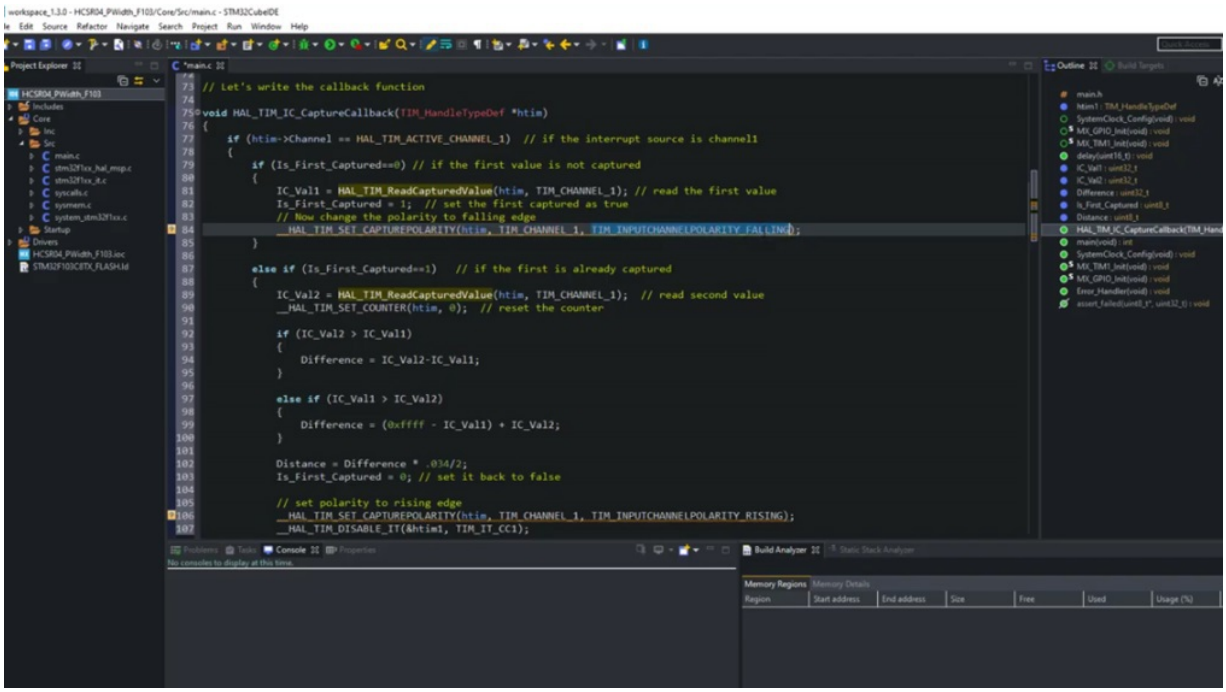
HCSR04 AND STM32 USING INPUT CAPTURE PULSE WIDTH CUBEIDE

This project will cover the HCS a 04 sensor, but this time we will use a different but better approach towards it. We will see how can we use input capture to measure the pulse width and which will be used for find the distance. Let's start with cue ID I am using STM 32 F 103 ch controller give the name to the project and click Finish let's fix up the clock first. I am selecting external crystal.



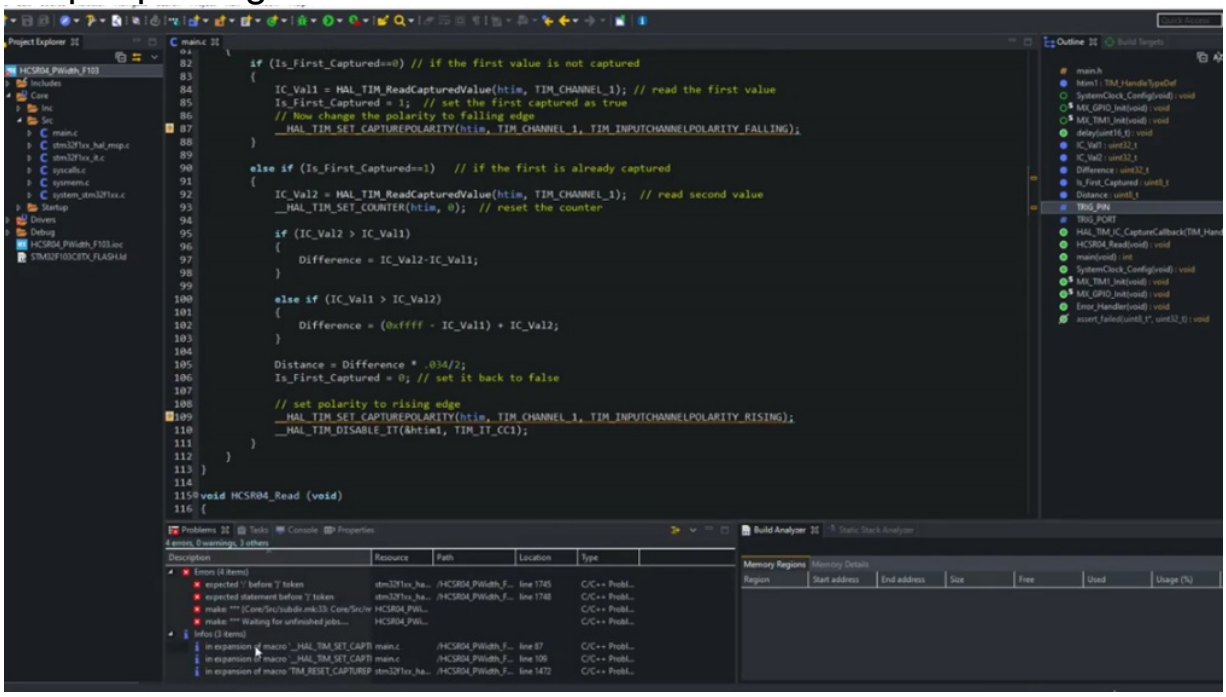
Do this if you have Cortex M three only. Let's run the system at maximum clock I am going to use timer one for this project. Select the Input Capture mode. I have made a project on input capture few months ago. You can watch it by clicking the top right corner. Timer

one is connected to APB two clock and which is clocked at 72 megahertz. If you are using any other timer, check the datasheet to know where it is connected to prescaler in my case will be 71 this will bring the timer clock to one megahertz and we will get a time a period of one microsecond per count. This is important because HCSS 04 sends the pulse a few microseconds. So in order to read the width of the pulse, we need the timer working in microseconds range. Now a R will be the maximum possible value for a 16 bit timer it would be 65,535. Make sure you enable the timer capture compare interrupt. Let's assume that you used another timer like timer two and you don't see the capture compare interrupt then in that case, enable the global interrupt pin PAA it is selected by default and we will connect the echo pin here select the pin P A nine as the output for the trick pin. Everything is set now. Click Save to generate the project let's start now first of all I will create a delay function which can provide us the microsecond delay I have already made a project on it check the top right corner.



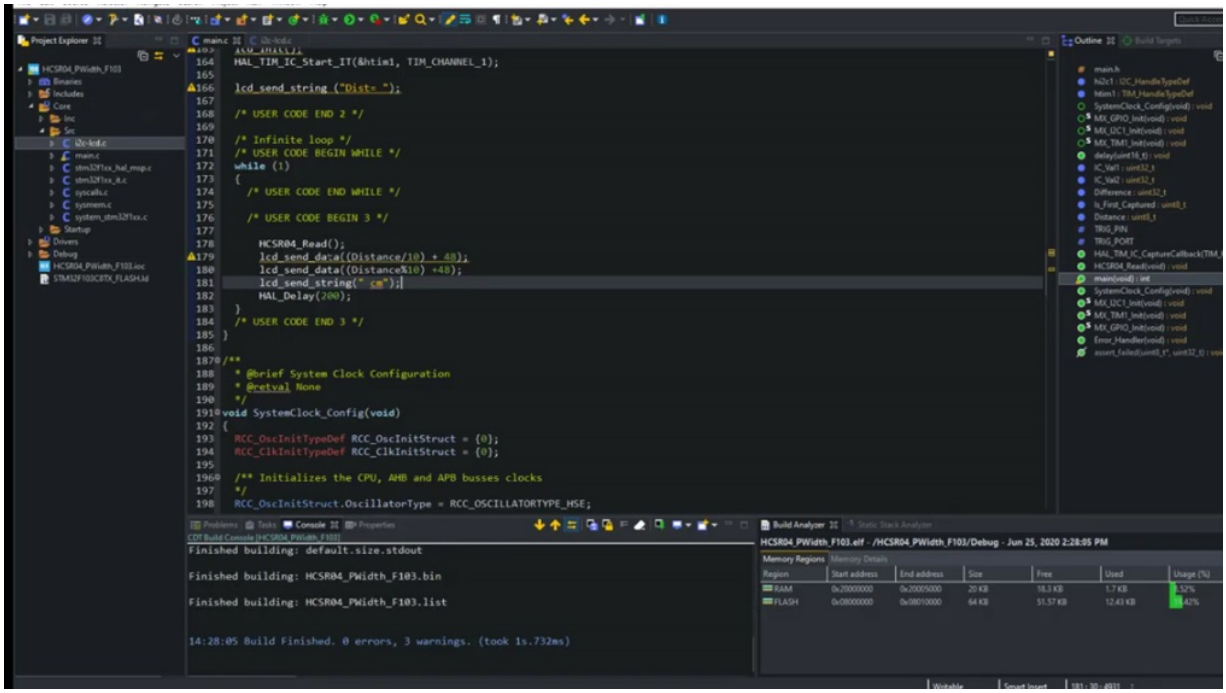
Here are the variables defined which will be used. This callback function will be called whenever a rising or falling edge will be captured. First, on detecting a rising edge the timestamp will be stored in the ice Val one variable. Now we will set the polarity for

capturing the falling edge. When the falling edge is captured, another timestamp will be stored in the IC Val two variable. Now the counter will be reset and we will calculate the difference between two captured timestamps. This difference will be in microseconds as our timer is running in that range. Based on this difference, the distance will be calculated by using the formula provided in the sensors data sheet. After finishing up all the calculations, we will change the polarity in order to capture the rising edge again and this loop will keep repeating.



If you are using any other time channel, make sure you update it at all the respective positions. This is the function to trigger the HCSS 04 to start capturing data. Let me quickly define the trig pin and port to trigger the HCSS 04 We have to pull the trigger pin high for around 10 microseconds and then pull it low for this purpose only I created a delay function to give delay in microseconds now in the main function we will start the timer in input capture interrupt mode and in the while loop we will read the HC SS zero for every 200 milliseconds we got a lot of errors here I have defined the pin in a wrong way we still got few errors this error is actually present in the default generated code you can also click the error in the console and you will directly reach here click at this end and delete this

bracket I didn't got this error in a four series controllers you may or may not get this based on your software version.



If you are not looking for the LCD This is it you will get the distance accurately within plus minus two centimeters range lexan include the eye to see for the LCD I am using eye to see one for this purpose that arrow got generated again let me quickly fix it now let's include the LCD library files refresh the code so that you can see them if you are using another eye to see make sure you change here initialize the LCD first let's print this string on the display now inside the while loop, we will print the distance values on the display let's say the distance is 23 then 23 divided by 10 will give us two and plus 48 is to convert to the respective character similarly 23% 10 will give us three let's build it looks like I forgot to include the header file include itu CLC d dot h Everything is good now. Let's flush it to the board.

EXAMPLE DUMMY CODE

Here's an example code that demonstrates how to interface an HC-SR04 ultrasonic distance sensor with an STM32 microcontroller using the Input Capture feature and CubeIDE. This code will measure the distance using the pulse width of the echo signal from the sensor.

Configure the Hardware Connections:

Connect the HC-SR04 VCC to a 5V power source, GND to GND, Trig to a GPIO pin, and Echo to a GPIO pin. Additionally, connect the Trigger pin to a timer channel's output (typically TIMx_CHy).

Configure the Timer in CubeIDE:

Open CubeIDE and configure your microcontroller peripherals using the graphical interface.
Configure a general-purpose timer (e.g., TIM2) and set its mode to Input Capture Direct mode.
Set the timer's prescaler and counter mode based on your desired measurement range and accuracy.
Generate the Code:

In the main.c file, include the necessary headers. Initialize the HAL library, timer, and GPIO pins. Here's the code snippet:

C

```
#include "main.h"

TIM_HandleTypeDef htim2;
volatile uint32_t start_tick = 0;
volatile uint32_t end_tick = 0;
volatile uint8_t measurement_done = 0;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();

  while (1)
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1,
GPIO_PIN_SET);  // Trigger the sensor
    HAL_Delay(1);  // Wait for a short pulse
```

```
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1,
GPIO_PIN_RESET);  // Reset trigger

    // Wait for the rising edge of the echo signal
(start of pulse)
    while (HAL_GPIO_ReadPin(GPIOA,
GPIO_PIN_0) == GPIO_PIN_RESET);
    HAL_TIM_IC_Start_IT(&htim2,
TIM_CHANNEL_1);  // Start Input Capture

    // Wait for the falling edge of the echo signal
(end of pulse)
    while (HAL_GPIO_ReadPin(GPIOA,
GPIO_PIN_0) == GPIO_PIN_SET);
    HAL_TIM_IC_Stop_IT(&htim2,
TIM_CHANNEL_1);  // Stop Input Capture

    float pulse_width_us = (float)(end_tick -
start_tick) * 0.5f;  // Calculate pulse width in
microseconds
    float distance_cm = pulse_width_us * 0.0343f;
// Convert to distance in centimeters

    // Use the distance value as needed

    HAL_Delay(1000);  // Delay between
measurements
  }
}
```

```
void
HAL_TIM_IC_CaptureCallback(TIM_HandleTypeD
ef *htim)
{
  if (htim->Instance == TIM2 && htim->Channel ==
HAL_TIM_ACTIVE_CHANNEL_1)
  {
    if (measurement_done == 0)
    {
      start_tick =
HAL_TIM_ReadCapturedValue(&htim2,
TIM_CHANNEL_1);  // Capture start time
      measurement_done = 1;
    }
    else
    {
      end_tick =
HAL_TIM_ReadCapturedValue(&htim2,
TIM_CHANNEL_1);  // Capture end time
      measurement_done = 0;
    }
  }
}

void SystemClock_Config(void)
{
  // Configure the system clock here
}
```

```
static void MX_TIM2_Init(void)
{
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode =
TIM_COUNTERMODE_UP;
  htim2.Init.Period = 0xFFFFFFFF;
  htim2.Init.ClockDivision =
TIM_CLOCKDIVISION_DIV1;
  HAL_TIM_Base_Init(&htim2);

  TIM_IC_InitTypeDef sConfigIC = {0};
  sConfigIC.ICPolarity =
TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection =
TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  HAL_TIM_IC_ConfigChannel(&htim2,
&sConfigIC, TIM_CHANNEL_1);
}

static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();

  GPIO_InitTypeDef GPIO_InitStruct = {0};
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

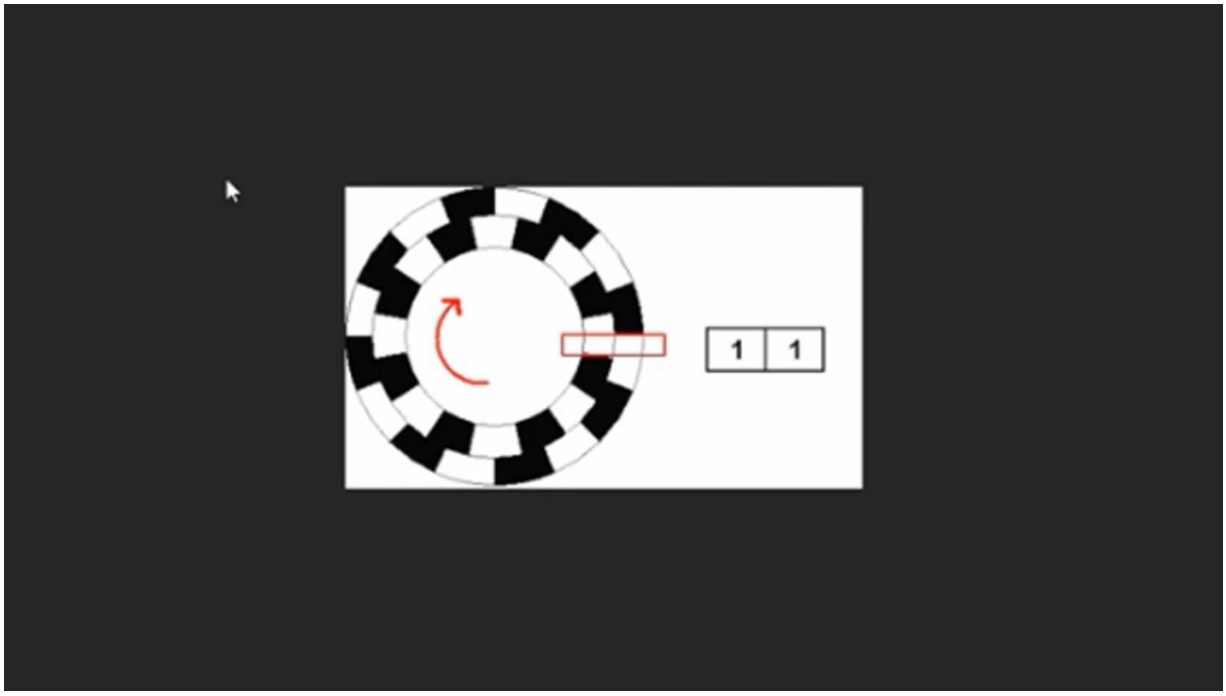
```
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = GPIO_PIN_1;
  GPIO_InitStruct.Mode =
GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

This code initializes Timer2 as an Input Capture timer. It configures GPIO pins for the trigger and echo signals. The main loop triggers the sensor, captures the echo signal's pulse width using Input Capture, and calculates the distance using the speed of sound.

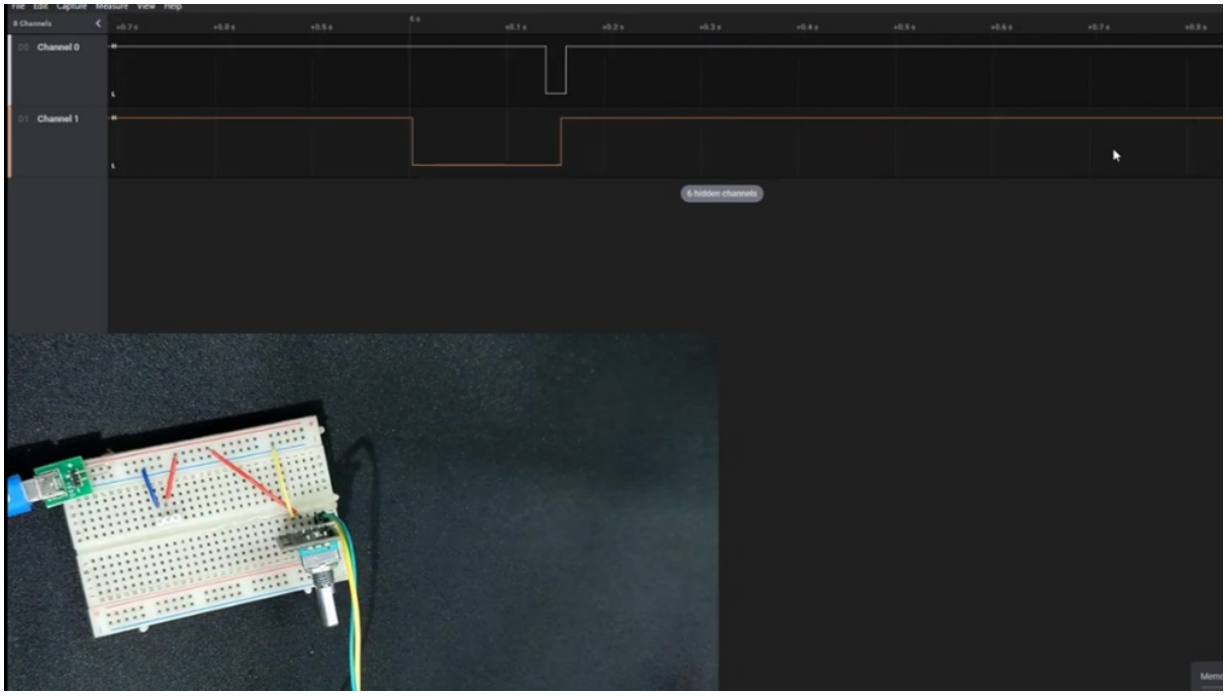
Please note that this code is a simplified example and might need further adaptation based on your specific STM32 microcontroller model and project requirements.

INCREMENTAL ENCODER AND SERVO ANGLE CONTROL IN STM32 PWM

We will see how to use the incremental encoder. And in the second half, we will use this encoder to control the positioning of the servo motor. By controlling the position, I mean the angle by which the servo rotates. So let's start with the encoder first. I have this encoder here, and it's rotary as you can rotate the shaft the shaft is free to rotate in either direction, as there is no limit on the rotation. He'd have five pins, but we are interested in the bottom two pins. The pin names do not justify their functions. So we will call the clock pin is pin A and D T pin is pin B. The middle one is the switch and it represents the push button on the shaft. That's it about the encoder.

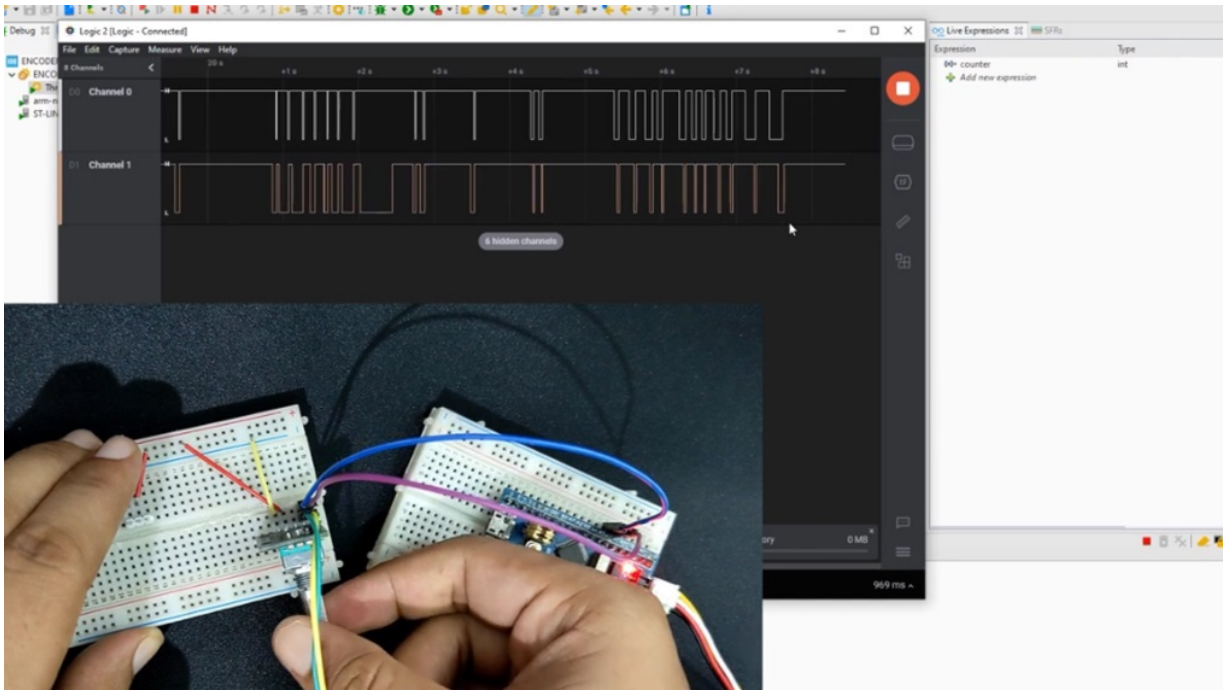


Now let's see how it works. I got this gift from the Wikipedia and it shows exactly what happens think of the black region as ground and white region as VCC. So it starts with both the pins in contact with the white region. So both of the pins are high. Now let's say we move the shaft clockwise so the outer pin goes low, while the inner one is still high. Now the inner pin goes low and both the pins are low. And now they are going back to high again. If we move it counterclockwise, the inner pin goes low first, then the outer one and then they both goes back to high. This is the entire working. Basically, we just have to check which pin goes low first, and based on that we can figure out whether the shaft moves clockwise or counterclockwise. Let's see one more time with the logic analyzer. Here I have connected the channels to the pin A and pin v sub supply is five volts let's start this pay attention I am rotating the shaft counterclockwise. Here we got the signal on both pins. Let's zoom in. We have some unwanted signals here but we will take care of them in the code itself. Let's focus on the main part. As you can see here, the first pin goes low and after some time, the second one goes low now I am rotating it in the other direction this time the second one goes to low first and after some time the first one goes low this is exactly what's shown in this animation.



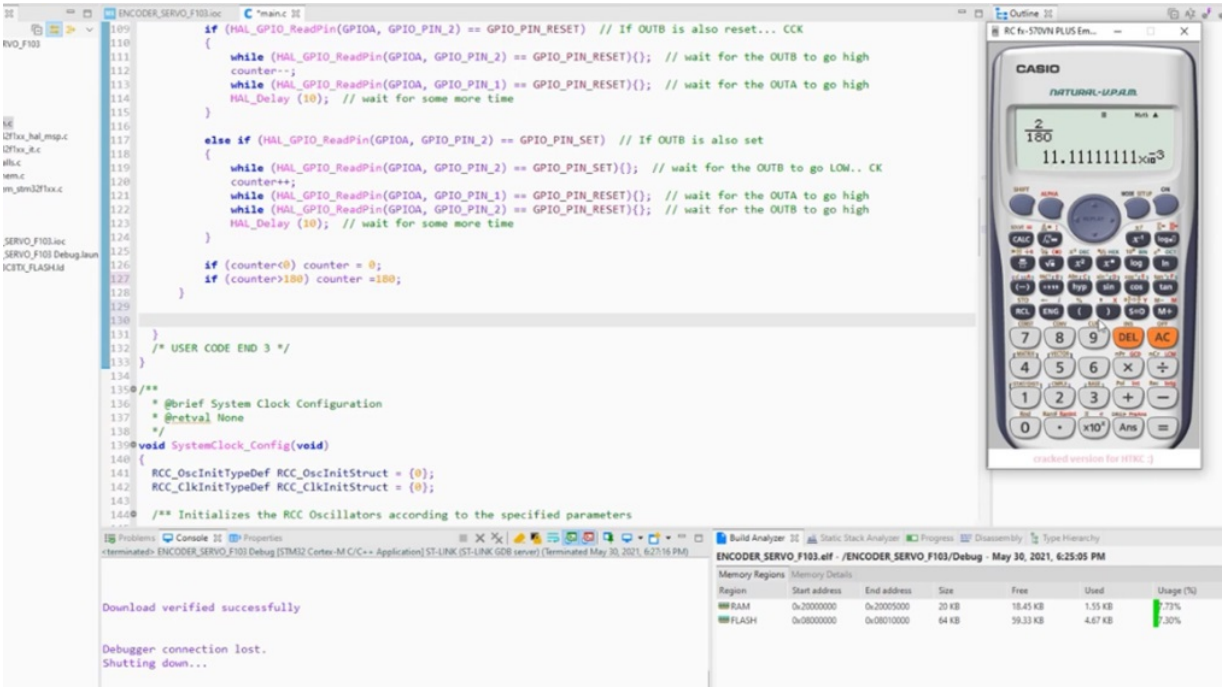
This is the entire working of this encoder and we will use these pins to identify the direction of rotation let's create a project in cube ID I am using f1 03 c eight. Give some name to the project and click Finish I am selecting the external crystal for the clock. Select serial wire debug. I have eight megahertz Crystal and I want the system to run at maximum 72 megahertz next select two input pins which will be connected to the encoder click Save to generate the project here first we will read the pin A if this pin A is low, then we will check for pin V actually I am considering this situation when the pin goes low at the moment pin V was already low and if this happens, we will increment the counter now, before incrementing this counter we will wait for the pin V to go back to high this is necessary to avoid those unwanted fluctuations. Also, we need to do the same for pin A as there are fluctuations in pin A also. So, we will wait for the pin A to go high as you can see the pin goes low again and it will change the counter value. So, to avoid this let's wait for some time before reading the pins again. Similarly, when the shaft is rotated in another direction, we will use the same code with minor changes of course this time we will check if the pin V is set this is the part where pin A goes low and at this particular time if the pin B is high that means the movement was different from the last time now, we will wait for the

pin V to go low This is to avoid these fluctuations now, when the pin V is low, we will decrease the counter then we will wait for the pin A to go high and then for the pin V let's say I want to restrict this counter value between zero and 20 this should be integer let's debug it now.



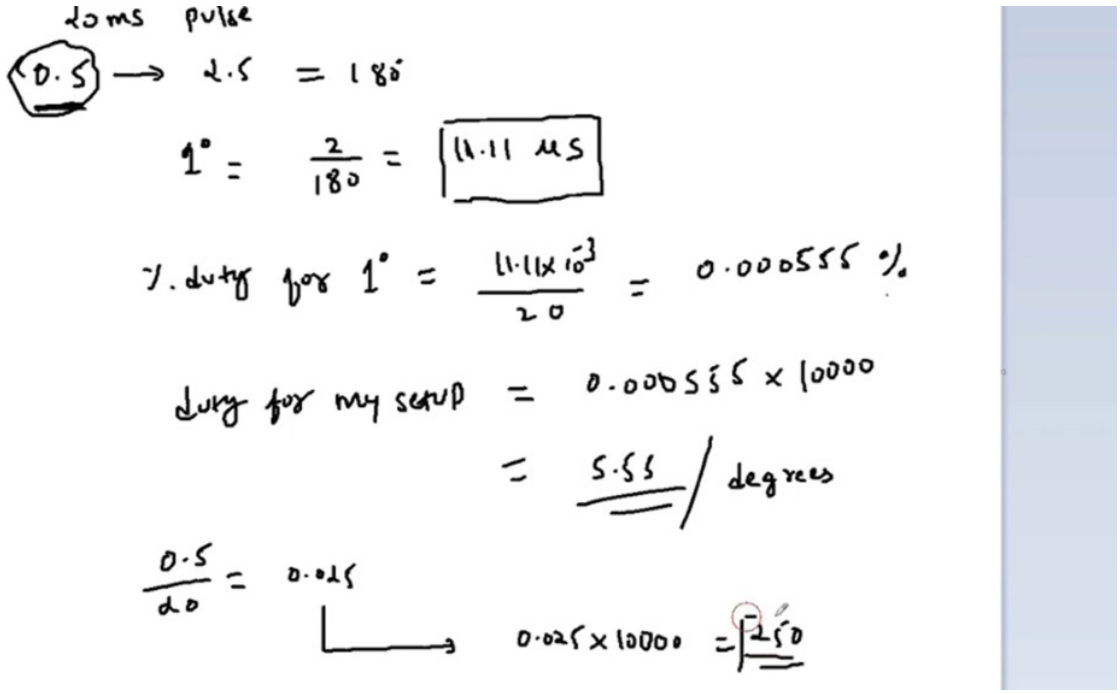
Note the counter value in the live expression I am rotating in counter clockwise direction and you can see the value is increasing now, if I rotate in clockwise direction the value starts decreasing the value is limited between zero to 20. So, the encoder is working well but I would like the value to increase in the clockwise direction. So, I am changing the counter operation here this is it about the encoder part. Now we will use this encoder to control the movement of servo motor. I have already covered the servo motor a long time ago and this is the tutorial for that I am taking this as the reference for today's project. Servo Motor Works with the pulse width modulation and like it's mentioned here that a pulse of one millisecond will move the motor to zero degree angle. Then 1.5 millisecond pulse will move it to 90 degrees. See and two milliseconds pulse will move it to 180 degrees, the pulse width needs to be 20 milliseconds which translates to 50 hertz frequency I have also mentioned that some Motorworks between 0.5 millisecond to 2.5 millisecond and I have

that kind so, first of all we will enable the PWM I am using timer one channel one for the PWM output signal you can see what the timer clocks are at 72 megahertz but if you want to know where the timer is connected, you can check the controller datasheet. Here you can see the timer one is connected to the APB to clock and if you see the clock set up the APB two timer clock is running at 72 megahertz. Now we need to bring this clock down to 50 hertz, if we divide 72 megahertz by 50 hertz, we get 1,440,000 We need to split this value between prescaler and the auto reload register. I am using the prescaler of 144 and auto reload of 10,000.



Keep the auto reload value high as this is where the percentage for the duty cycle will be calculated. So if we keep it high, the percentage values will be high and hence more accurate. Let's save this to generate the project. Now like I said, the motor needs the pulse width between 0.5 milliseconds to 2.5 milliseconds This means that the motor covers the angle of 180 degrees in the difference of two milliseconds. This would translate to 11.11 microseconds per degree rotation let's start the timer in PWM mode with channel one. We will limit the count of between zero to 180 As per the servo rotation. Now we have 11.11 microseconds for one degree rotation and the pulse width is 20 milliseconds. This is the duty cycle for one

degree rotation. Let me explain this properly in between 0.5 millisecond to 2.5 milliseconds the motor covers 180 degrees. This means the motor will take 11.11 microseconds for one degree rotation again we have 20 milliseconds pulse width for the servo. So the percentage duty cycle for one degree rotation is 0.000555 and so on. This is the percentage from 100 obviously but we have the auto reload of 10,000. And hence we will calculate the 0.00055% of 10,000. This will come around 5.55. This is the value which corresponds to the same duty cycle as zero Point five from 20. Let's define a variable to store the PWM value, now we will multiply this value to the counter value, this will give us the duty cycle corresponding to the counter value. And now we will feed this value to the capture compare register. I am using channel one and that's why CCR one if you are using any other channel use the respective capture compare register. There is one last thing the motor timing starts from the 0.5 milliseconds when the pulse width is 0.5 the motor will be at zero degrees. So we need to also take this into consideration the duty percentage for 0.5 is 0.025 which in our case will become 250. So we will add this 250 to each of our PWM values. This will make sure that when the counter is zero, the motor must be supplied with 0.5 milliseconds pulse.



And with the increment in the angle, the time will be added to this 0.5. This is the servo I am using it's the SG 90 It have three pins, the red and brown are for VCC and Ground and the yellow is for the signal let's debug it now. Check the counter value as it is the angle by which the servo is rotating. clockwise rotation is increasing the angle it have reached 90 and the servo has also rotated by the same amount. Now we are at the maximum 180 degrees. If we reset the controller here, the counter will go back to zero and the server will go back to its original position. clockwise rotation is increasing the angle counterclockwise rotation is decreasing the angle and the servo is also rotating in the reverse direction. If you want to debug the registers, you can add the SFR. Here check the value of the CCR one register. It's not changing in the real time, but if you refresh it, you can see the value. This is it for this tutorial. I hope things were clear. Before using the code, check the server you are using as it might not work within the 0.5 to 2.5 range. Most of the servo motors work between one milliseconds to two milliseconds range, so make sure to check this part.

EXAMPLE DUMMY CODE

Below is an example code that demonstrates how to interface an incremental encoder and control a servo motor's angle using PWM with an STM32 microcontroller. The code uses an STM32 HAL (Hardware Abstraction Layer) and CubeIDE for easy setup. It assumes you have an incremental encoder connected to two GPIO pins and a servo motor connected to a PWM output pin.

Configure the Hardware Connections:

Connect the incremental encoder's A and B outputs to two GPIO pins and the servo motor's control input to a PWM-enabled GPIO pin.

Configure the Timer and PWM in CubeIDE:

Open CubeIDE and configure the microcontroller peripherals using the graphical interface.
Configure a general-purpose timer (e.g., TIM3) in Encoder mode for reading the incremental encoder.

Configure another timer (e.g., TIM2) in PWM mode for controlling the servo motor.
Generate the Code:

In the main.c file, include the necessary headers.
Initialize the HAL library, timers, GPIOs, and PWM channels.
Here's the code snippet:

C

```
#include "main.h"

TIM_HandleTypeDef htim2;  // PWM timer for servo control
TIM_HandleTypeDef htim3;  // Encoder timer

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_TIM3_Init();
```

```
  int32_t encoderValue = 0;
  int32_t encoderValuePrev = 0;
  uint16_t servoPulseWidth = 1500;  // Initial
pulse width for servo (center position)

  HAL_TIM_Encoder_Start(&htim3,
TIM_CHANNEL_ALL); // Start encoder

  HAL_TIM_PWM_Start(&htim2,
TIM_CHANNEL_1); // Start PWM

  while (1)
  {
    encoderValue =
__HAL_TIM_GET_COUNTER(&htim3); // Read
encoder value

    // Update servo angle based on encoder
value change
    if (encoderValue != encoderValuePrev)
    {
      if (encoderValue > encoderValuePrev)
      {
        servoPulseWidth += 10; // Increment angle
        if (servoPulseWidth > 2000) {
          servoPulseWidth = 2000; // Maximum
angle
        }
      }
```

```
      else
      {
        servoPulseWidth -= 10; // Decrement
angle
        if (servoPulseWidth < 1000) {
          servoPulseWidth = 1000; // Minimum
angle
        }
      }

      // Update servo PWM pulse width
      __HAL_TIM_SET_COMPARE(&htim2,
TIM_CHANNEL_1, servoPulseWidth);

      encoderValuePrev = encoderValue;
    }
  }
}

void SystemClock_Config(void)
{
  // Configure the system clock here
}

static void MX_TIM2_Init(void)
{
  // Configure TIM2 for PWM output for servo
control
}
```

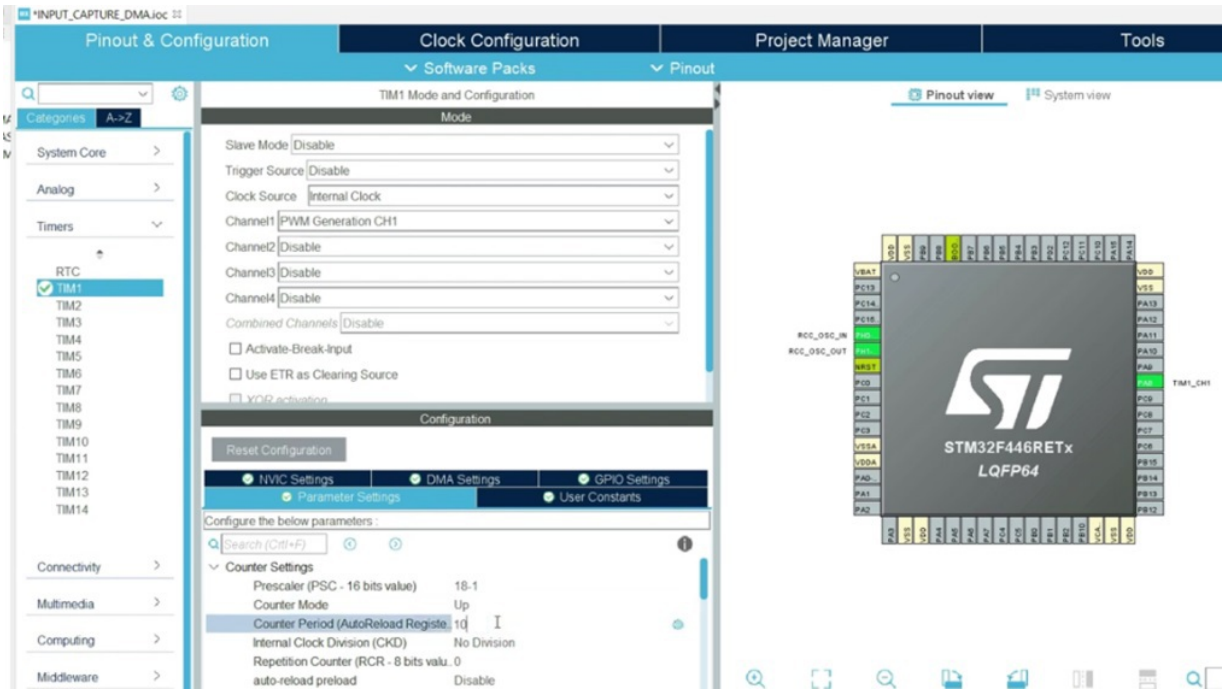
```
static void MX_TIM3_Init(void)
{
  // Configure TIM3 for encoder input
}

static void MX_GPIO_Init(void)
{
  // Configure GPIO pins for encoder and PWM
}
```

This code initializes two timers: TIM2 for PWM output to control the servo motor and TIM3 in Encoder mode to read the incremental encoder's A and B outputs. The main loop reads the encoder value, compares it to the previous value, and adjusts the servo angle accordingly by changing the PWM pulse width.

Please note that this code is a simplified example and might need further adaptation based on your specific STM32 microcontroller model, encoder type, servo specifications, and project requirements. Additionally, ensure that the timers, GPIO pins, and PWM channels are properly configured in CubeIDE.

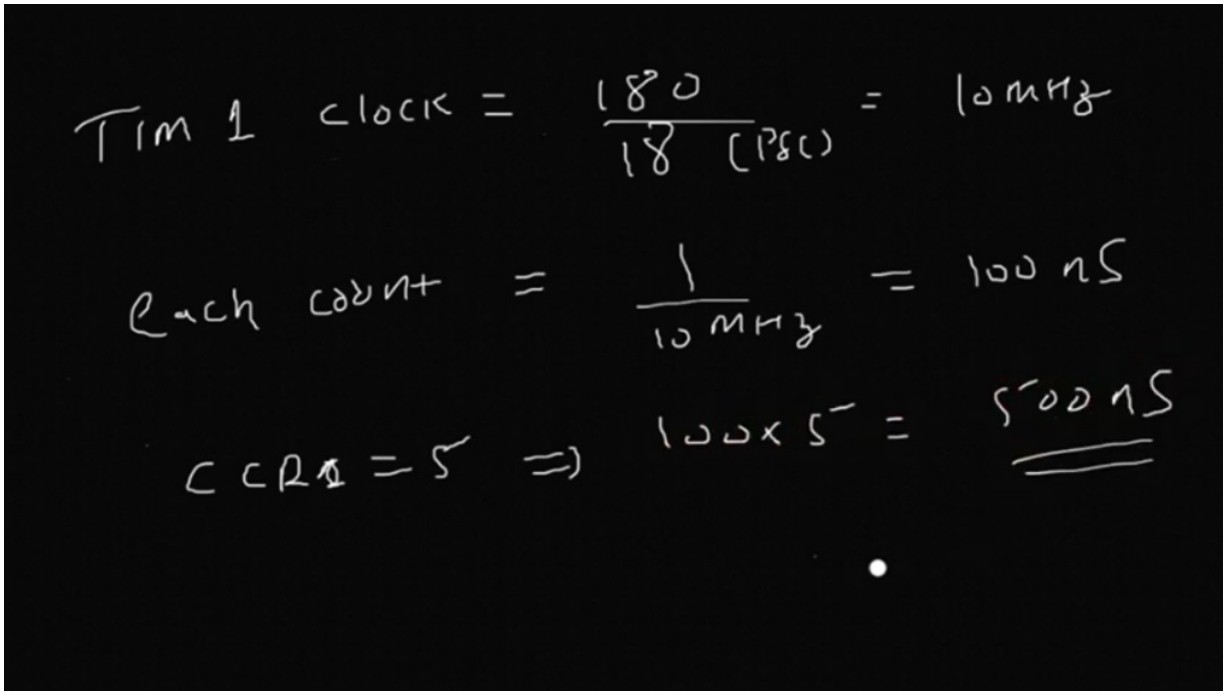
INPUT CAPTURE USING DMA MEASURE HIGH FREQUENCIES AND LOW WIDTH

We will look into the method of DMA. In the previous project about the input capture, we saw how to measure the frequency and width of the incoming signal. The method was pretty simple to implement, but we could only measure the frequencies up to few 100 kilohertz range. Today we will use the DMA to do the same, but we will be able to measure frequencies up to few megahertz range. I have already uploaded the code on my GitHub and I will show how to implement it.



The IOC file here is for the setup reference. The entire code is in the main file. Let's start the Q ID and create a new project I am using STM 32 F 446 controller for this tutorial. Give some name to the project and click Finish. First thing we will do is set up the clock. I am using external high speed crystal for the clock I have eight megahertz crystal on board and I want to run the system of maximum 180 megahertz frequency. Make sure you input the correct crystal frequency for the input or else the project won't work well check the crystal on your controller and use that has the input clock now we will configure the timer. I am going to use the timer one for the PWM. The timer one is connected to the APB two clock, which is running at 180 megahertz. Also the timer two is connected to the APB one clock which is running at the 90 megahertz clock I am setting the timer so that the PWM frequency would be equal to one megahertz. If you don't understand this part, check the previous project about the PWM. Now we will set up the timer to for the input capture clock source is the internal clock set the channel one as the input capture direct mode. You can see this pin here got selected as the timer to channel one pin. Here we will connect our signal select the channel two is the input capture indirect mode and we don't have any additional pin for it keep the prescaler to zero and set the auto reload to the maximum value. Now set the polarity selection to the rising edge for the channel one and falling edge for the channel two this is it for the parameters now we will configure the DMA at the DMA request for the channel one makes sure it's set from peripheral to the memory data with his word and the mode is set to normal mode add another request for the channel to and same configuration for it also. Now the final thing would be to enable the interrupt for the timer that's it for the configuration click Save to generate the project. Now we will copy this entire code and paste it in our main file and inside the main function, we will start the PWM here the timer one auto reload is set to 10 So I am setting the capture compare value to five. Now start the PWM for channel one. This is the wrong function and I will correct it afterwards. Next we will start the input capture in the DMA mode for both the channels of timer two To hear timer to channel one is set to capture the rising edge and it will store the values in Rise data. Similarly, channel two will store in the fall data

buffer they will capture these number of pulse data and store them in the buffers. We will then do the calculation using the data collected in these buffers inside the while loop, we will start the input capture whenever required. This way we can also use it in the RT o s and we don't have to worry about the interrupts being generated all right now some important things define the timer clock here. Basically the APB timer clock frequency also defined the prescaler you are using I have used the prescaler of zero for the time of two which is running at 90 megahertz. Since we are capturing lower frequency for now, I am only taking 100 captures you can go through this code here it is commented properly so you will understand it the pulse width is measured in nanoseconds let's build and debug this now I have added the values in the live expression All right, we are getting something weird and that's because the PWM function I used is wrong. This is the correct one let's measure it again. So here we have the frequency of one megahertz and the pulse width is 500 nanoseconds let's understand this 500 nanoseconds first. As we know the timer one clock is 180 megahertz and we use the prescaler of 18 This bring the timer clock to 10 megahertz so each count of the counter takes 100 nanoseconds we have used the capture compare of five so the pulse width remain high for 500 nanoseconds. Now let's try to increase the frequency of the PWM the prescaler is zero and let's use the auto reload of nine. So the timer one PWM frequency would be 20 megahertz now now the timer clock is at 180 megahertz. So each count would take 5.55 nanoseconds CCR of five would give the pulse width of around 27 nanoseconds let's test this one now. The frequency is not accurate anymore also the width is highly inaccurate Let me reduce the frequency. Now the frequency would be 15 megahertz All right, the frequency is accurate now but the width is still inaccurate. I guess it can't measure the very low wits.



I will try increasing the capture compare value so that the signal would remain high for a little bit more time all right now we are getting the accurate with. Actual li the see see our value is eight now, and the signal remains high for 44 nanoseconds. And this is what we saw in the debugger. So this is it for this project, measuring up to 20 megahertz and the width of 40 nanoseconds isn't that bad after all, but if you want to go even higher in terms of frequencies, I will make another project for it. We will not use input capture in that but using some other timer mode.

EXAMPLE DUMMY CODE

Below is an example code that demonstrates how to use the DMA and Input Capture features of an STM32 microcontroller to measure high frequencies and low pulse widths of an external signal. This code captures input signal timings and processes them to calculate frequency and pulse width. The example assumes you're using a general-purpose timer like TIM2 and DMA1.

Configure the Hardware Connections:

Connect the input signal you want to measure to a GPIO pin configured for Input Capture.

Configure the Timer and DMA in CubeIDE:

Open CubeIDE and configure the microcontroller peripherals using the graphical interface.
Configure a general-purpose timer (e.g., TIM2) in Input Capture mode.
Configure DMA1 in your microcontroller's configuration.
Generate the Code:

In the main.c file, include the necessary headers.
Initialize the HAL library, timer, DMA, and GPIO pin.
Here's the code snippet:

C

```
#include "main.h"

TIM_HandleTypeDef htim2;  // Timer handle
DMA_HandleTypeDef hdma_tim2_ch2;  // DMA handle for TIM2

uint32_t capturedData[2];  // Store captured values (rising and falling edges)

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
static void MX_DMA_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_DMA_Init();
```

```
  HAL_TIM_IC_Start_DMA(&htim2,
TIM_CHANNEL_2, (uint32_t *)capturedData, 2);

  while (1)
  {
    uint32_t risingEdge = capturedData[0];
    uint32_t fallingEdge = capturedData[1];

    if (fallingEdge > risingEdge) {
      uint32_t period = fallingEdge - risingEdge;
      float frequency =
(float)HAL_RCC_GetHCLKFreq() / period;

      // Use frequency value as needed

      // Calculate pulse width
      uint32_t pulseWidth = risingEdge -
capturedData[0];

      // Use pulse width value as needed
    }
  }
}

void SystemClock_Config(void)
{
  // Configure the system clock here
}

static void MX_TIM2_Init(void)
```

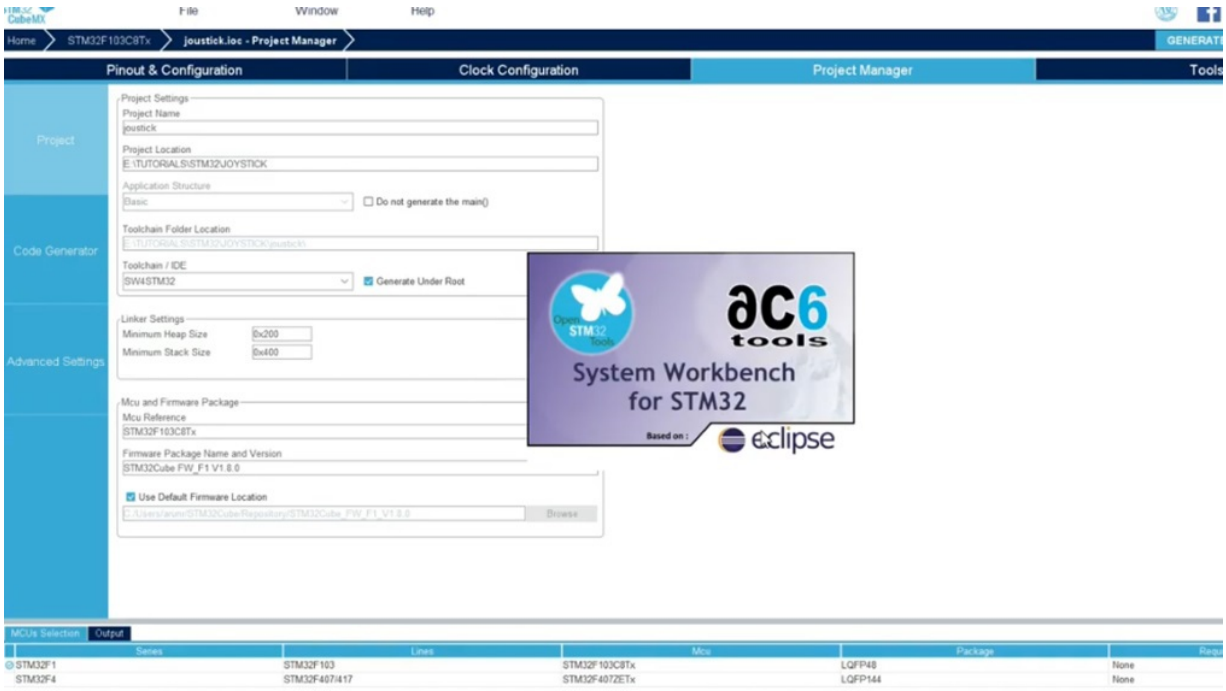
```
{
  // Configure TIM2 for input capture
}

static void MX_DMA_Init(void)
{
  // Configure DMA for TIM2 input capture
}

static void MX_GPIO_Init(void)
{
  // Configure GPIO pins for input capture
}
```

This code sets up TIM2 for input capture using the rising edge on channel 2. DMA is used to automatically transfer the captured values to the capturedData array. In the main loop, the captured rising and falling edge values are used to calculate the period, frequency, and pulse width of the input signal.

Please note that this code is a simplified example and might need further adaptation based on your specific STM32 microcontroller model, input signal characteristics, and project requirements. Ensure that the timers, GPIO pins, and DMA channels are properly configured in CubeIDE.

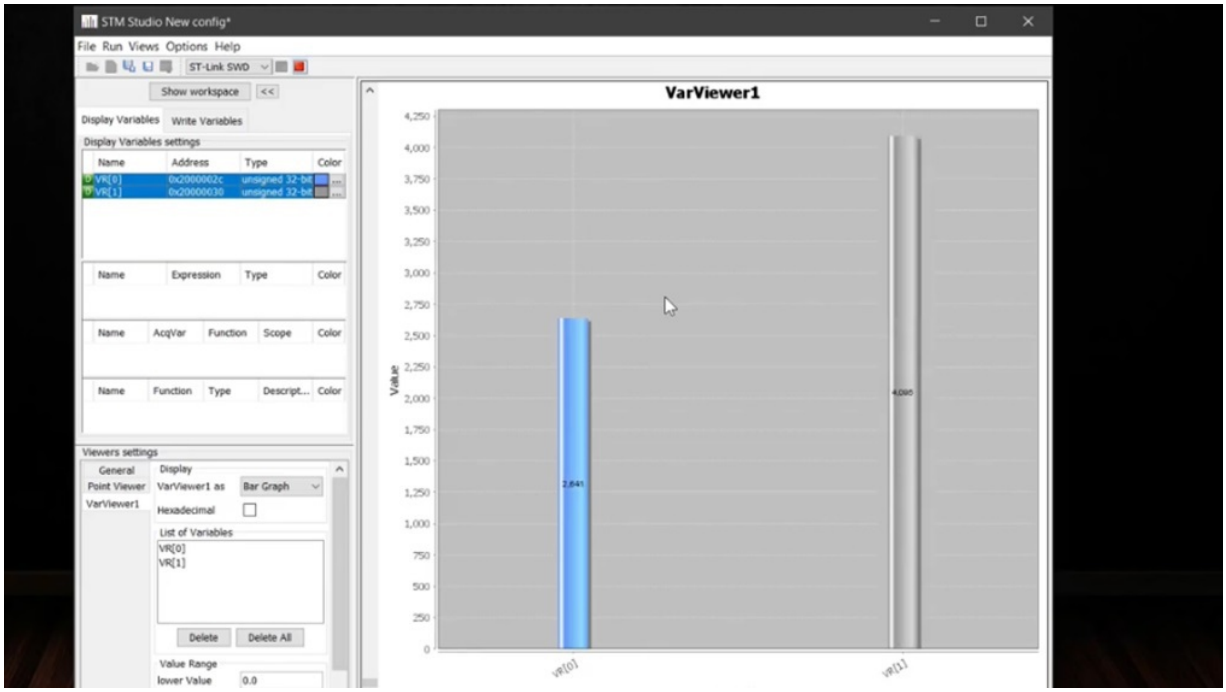
JOYSTICK MODULE WITH STM32 ADC MULTI CHANNEL HAL

We are going to interface joystick with STM 32. I am using STM 32 F 103 micro controller joystick can be used for the purpose of controlling the robot or a camera or anything else the output from the joystick is in the form of an analog signal and so we have to use the ADC to read this data here I am using channels one and two of ADC one the pins used are a one and a two there is a little bit of setup needs to be done in the ADC.



First select the continuous conversion mode and enable it this is because we want the conversion to be taking place continuously. Next select the number of conversions as to because we have two

channels to do the conversion for rank one is for channel one and I have changed the sampling time to 239.5 cycles to make the conversion fastest possible rank two is for Channel Two with same settings. Next thing we need to do is enable the DMA in circular mode. This is important since in order to use multi-channel we have to use DMA next I am going to connect four pins to the four LEDs to demonstrate the working of the joystick and so I am selecting these four pins as output the rest is our usual sets up. Here I am creating an array of two variables to store the ADC result it must be a 32 bit unsigned integer because that's the argument of the ADC de ma function. Next we need to start ADC in DMA mode and the argument will be the array that we created above please note that do not write this inside the while loop write it before the wire loop begins. Now let's compile it and flashing the micro controller. Let me just quickly fix this one okay so the flashing is done.

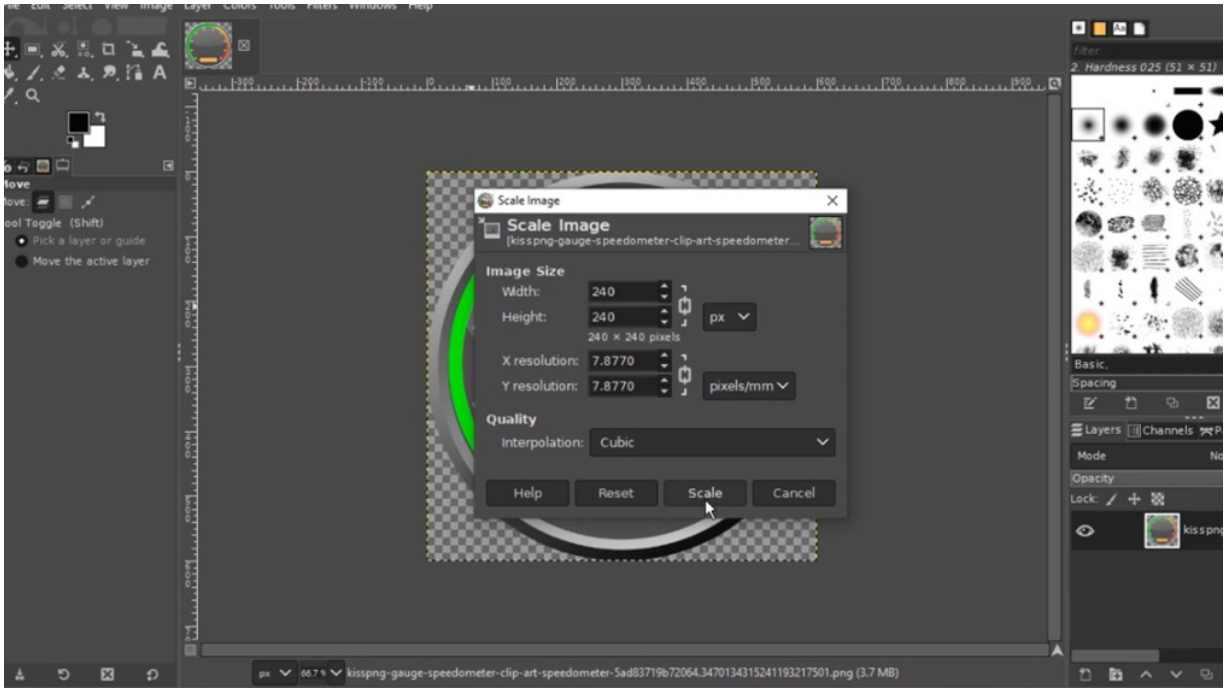


Now, I will show you the result in the STM studio first go to the project folder in the debug folder open the Dotty ELF file now send both of these variables to the variable Will viewer and change it to the bar graph now notice the bar graph as I move the stick around no please values carefully especially the extreme values this is varying between zero to 4095 because the ADC and STM 32 will

have 12 bit of resolution we will write our rest of the program based on these values. In my case VR zero is connected to the Wii RX and VR one to the VR why this is the case if the stick is in the middle then all the LEDs will be off. Rest we can write individual cases for each position turning on individual LEDs.

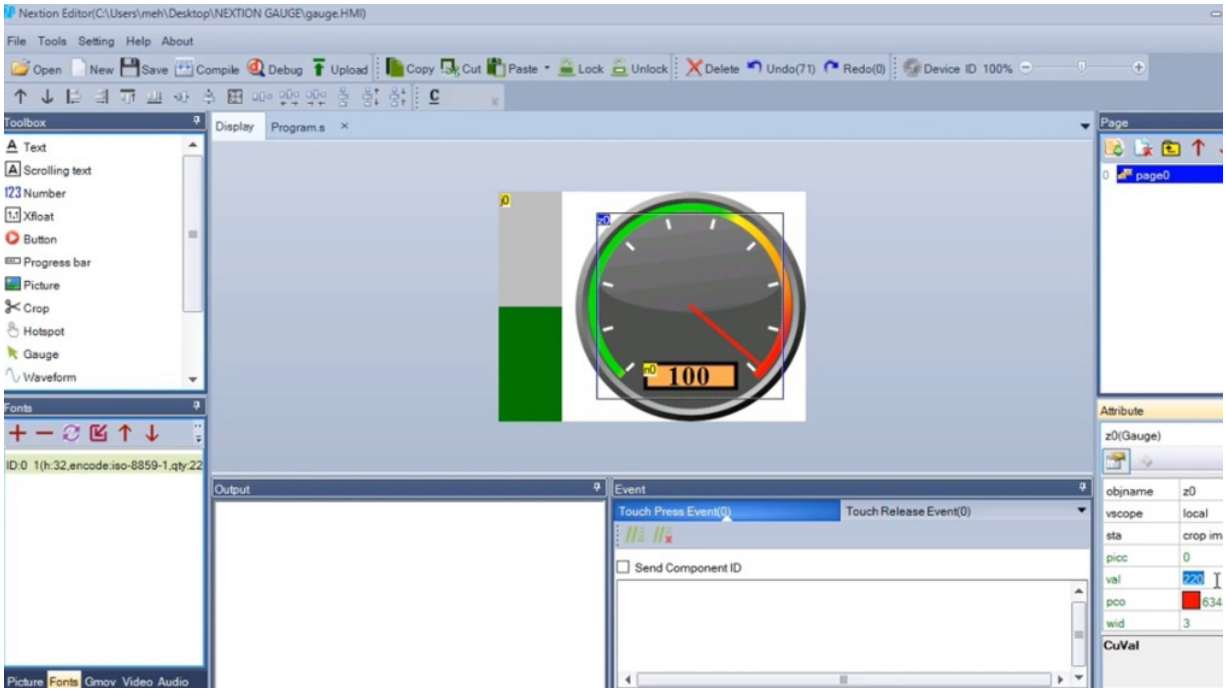
NEXTION GUAGE AND PROGRESS BAR STM32

Let's start by creating the project in next and editor first create the project folder here give some name and click Save select your next in model and select the display orientation here first of all we need to download the gauge background Let's google for some This one seems good for our job. Let's go to this website this is 600 by 600 pixels and it's perfect. Let's download it we need to reduce its size so it can fit the display.



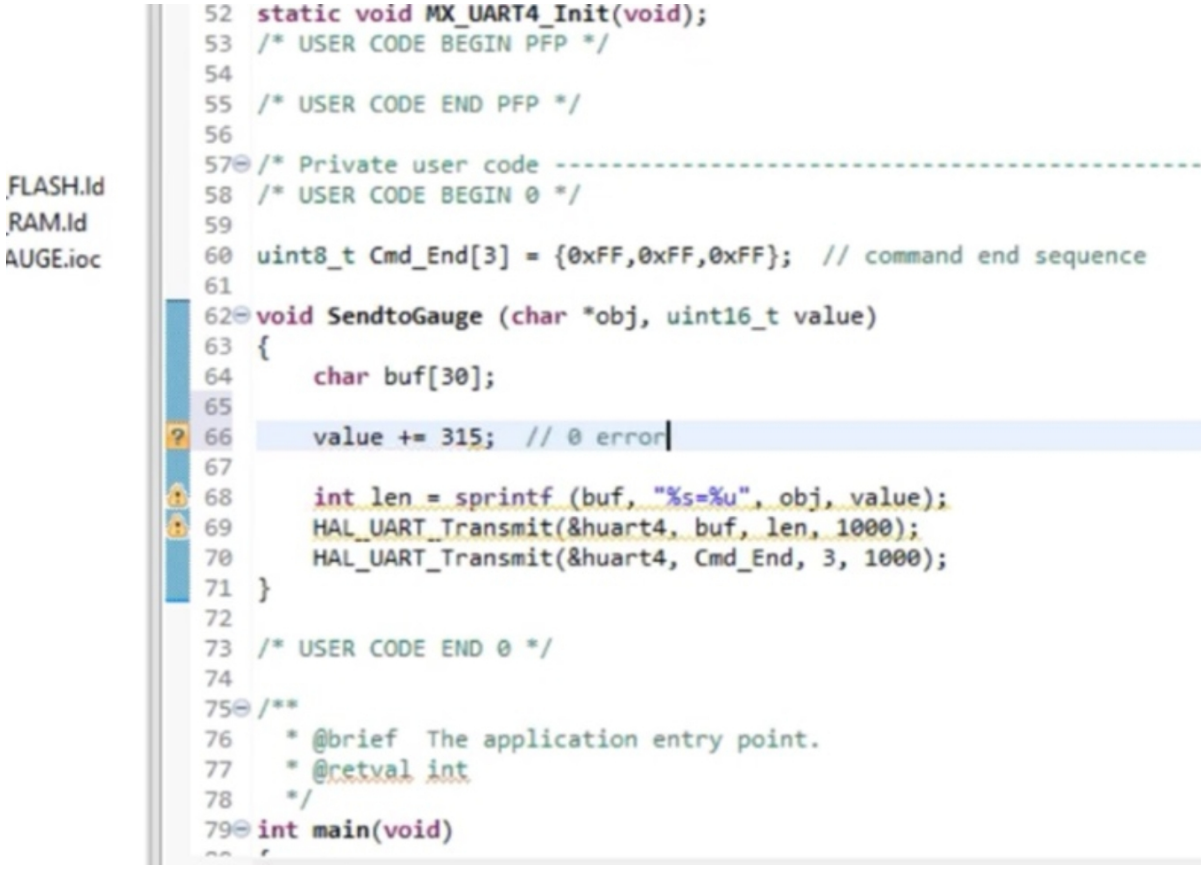
Let's open it with GIMP I have the height here as 240 pixels so I will reduce it to the same size go to image scale image and reduce the size here let's export it in the same PNG format I am going to create a background now the size should be same as the display size and that image is a layer here I am going to keep a progress

bar on the left here let's export this final image we need to add the image to our project add the image box to the display and include the gauge image so this will be our background on the display. Let's set a progress bar I am going to select the vertical type now the gauge if you don't want this white background, select the background as cropped image and select the same image that is in the background. Let's change that pointer color and its thickness adjusted also try with different values is to check if the pointer fits the gauge okay, it's perfect fit now let's add the number box to this use the cropped image as the background we also need to add fonts here you can generate them if you don't have let's try and put a value to see if it fits the number box great it fits perfectly let's test some values on the gauge here we don't want the pointer to go here so 315 indicates the minimum value on the gauge but remember, this gauge can only rotate between zero to 360.



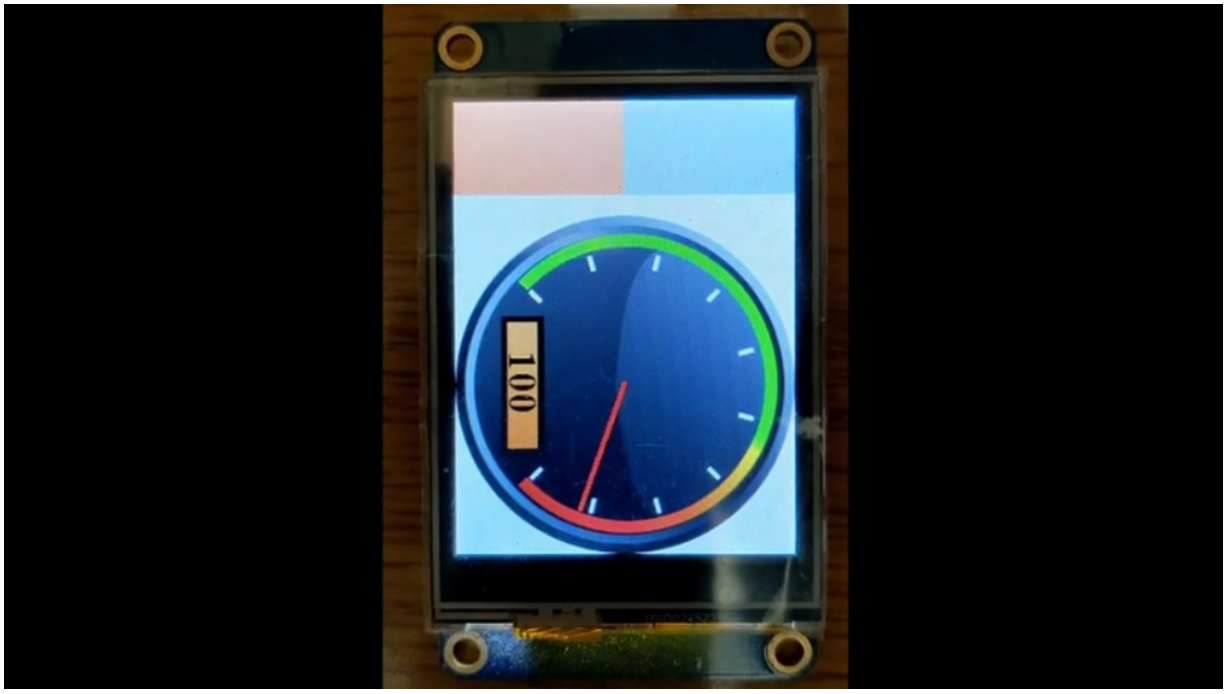
If I tried to go higher than 360 it will simply not take the value. This 360 and zero are the same positions let's try to look for the highest value so 225 is the maximum value we can have. Keeping that in mind let's create the project with cube ID I am using STM 32 F 446 R II controller give some name to the project and click finish first of all I am selecting the external crystal for the clock I will be

connecting the display to the UART for make sure you keep the baud rates to 9600. Let's do the clock setup now. I have eight megahertz oscillator and I want the system to run at maximum clock click Save to generate the project now.

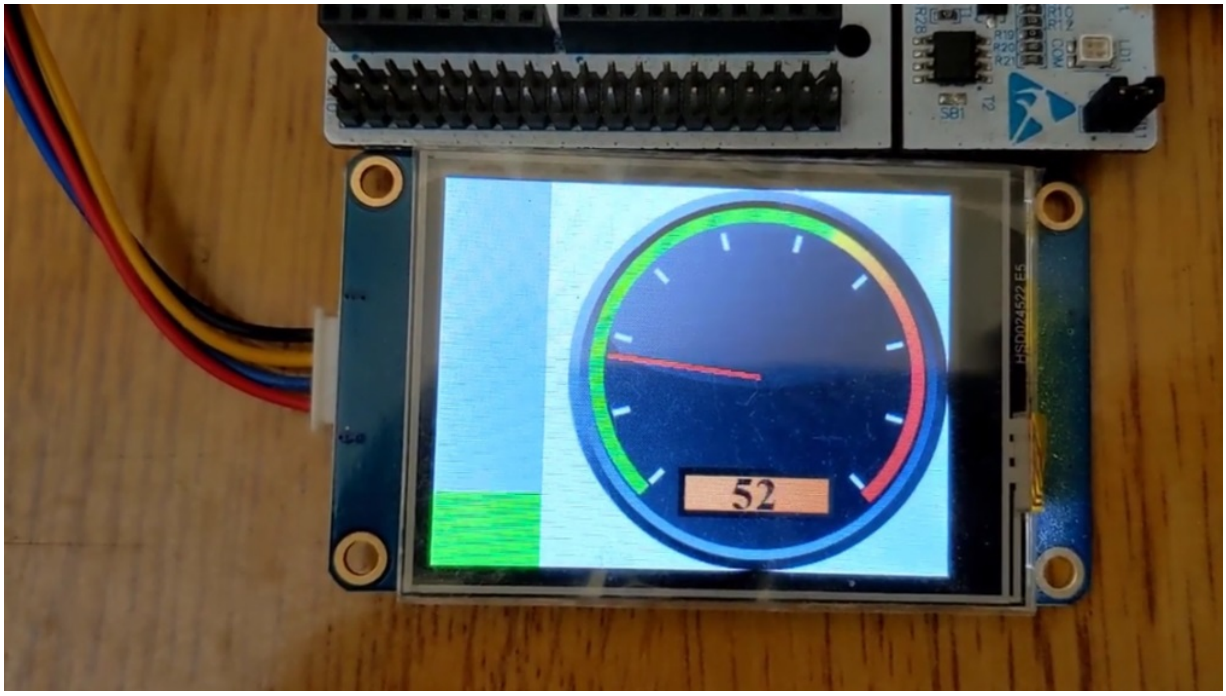


Let's first define the end commands these must be sent after every command to the next in create a function to send the value to the gauge here first define a buffer to hold some values. Now copy the data to be sent into the buffer and now we will send the buffer to the UART. And finally, send the end commands. Let's build it once we need to include stdio for the S printf function. Now if you remember that the initial value for the gauge was 315. So we will add 315 to every value. And since the gauge can have the maximum value of 360 if the value is higher than 360 we will subtract 360 from the value now another function to send the data to the number box we basically have the same data here the same steps for sending the data to the progress bar also basically you can just create a single function for all these values now in the while loop, we will send the

values to all the elements I am using 270 Because that's the limit for my gauge set up first I will send it to the gauge the object name is Sid zero and the Value field is vowel.



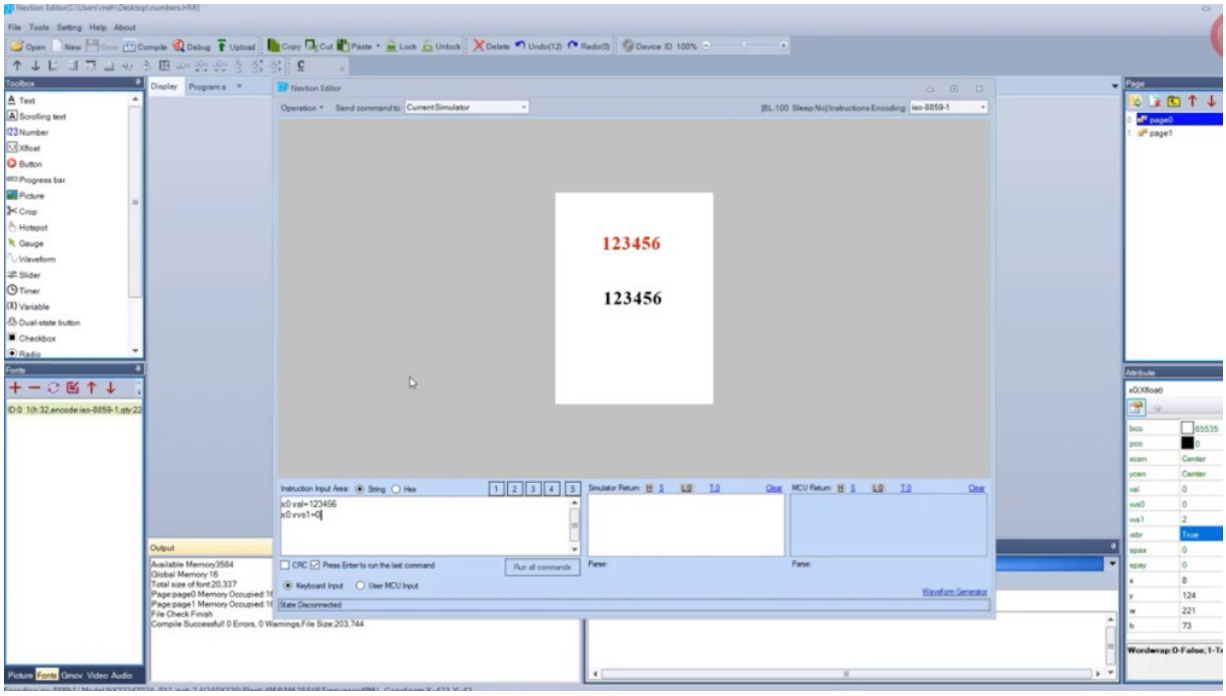
Next send to the number here the object field is n zero and the Value field is Val. Now before sending the value to the progress bar, I am going to modify the color of the bar based on the value colors will vary from green to yellow to orange to red based on the value if the value is less than 160, we will send 2016 to this field the object name for the bar is J zero and the color field is PCO 2016 is the code for the green color. Here you can get different codes for the different colors and then based on different values, we will send the commands to change the color of the bar now we will convert the value to the percentage so that we can send it to the progress bar and finally send the value to the value filed in the progress bar I will add a small delay here we need to upload this to the next and display first I have connected the display to the F TDI which is connected to comm four let's start the upload you can see the data is being written to the display and we are done and the images are loaded successfully.



Let's upload the code to the board to here we go this flickering effect remains with the default gauge. If you want to remove it you need to make your own gauge with different set of pictures. I will do that in another project we can also simulate the gauge in the editor. But for that we need to send the serial data to the computer and in my nucleo board I have to use you out too for that purpose and I will change these you up for to you are too in the editor select the debugging and select User MCU input select the comport and press Start Now I will load the code into the board you can see it's pretty smooth here compared to the actual display let me just reduce the delay to 10 milliseconds.

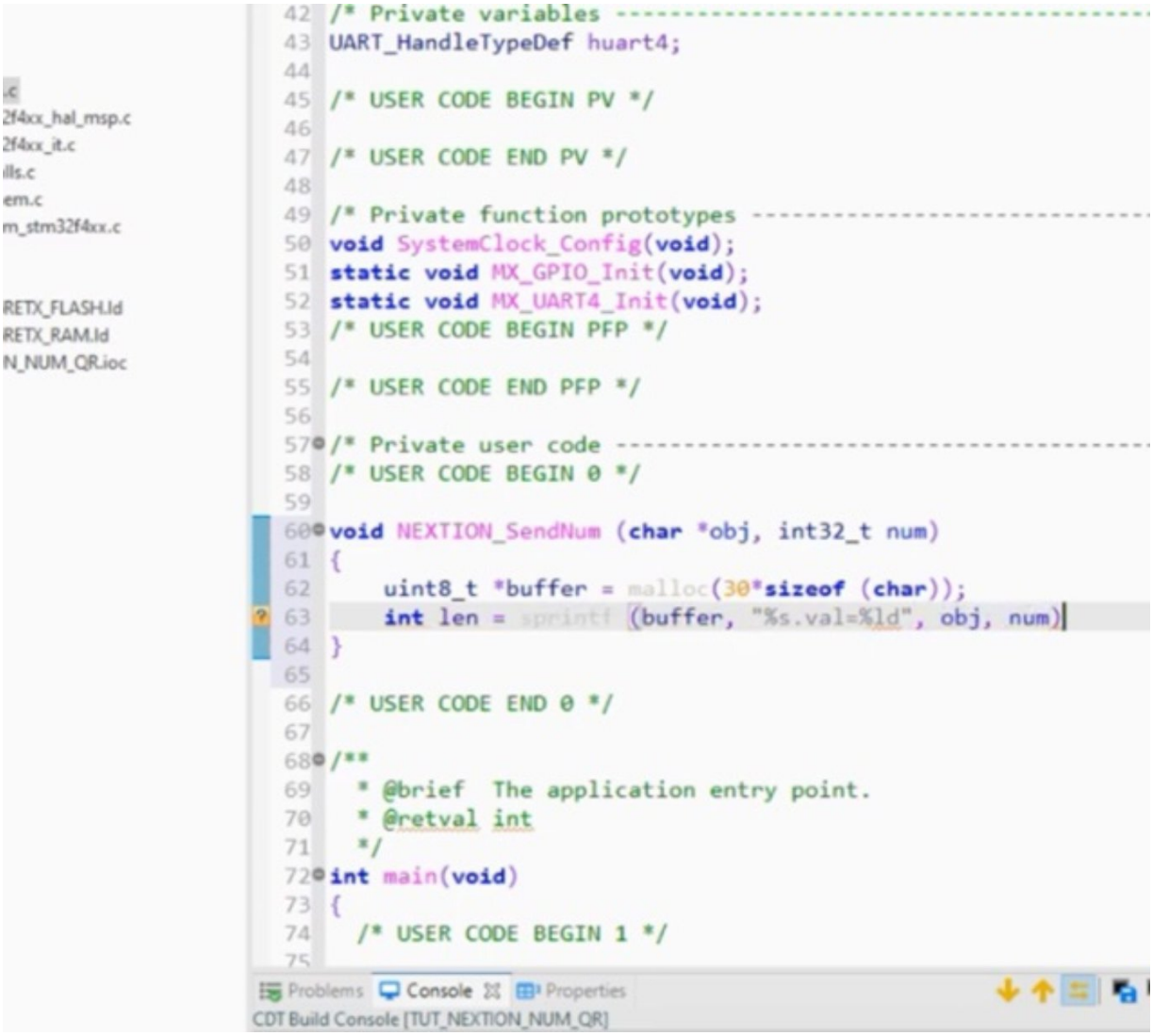
NUMBERS FLOATS QR CODE HOTSPOTS IN NEXTION DISPLAY STM32

today I will cover the remaining features in the next in display and that would be sending numbers sending floats hotspots, and finally the QR code. So let's start by creating the project in nexty and editor give some name to this file and select the Display Type I will use the vertical orientation today let's add one more page to make it more interesting. We will add the hotspot first and I am going to turn the entire display into a hotspot. So basically I can touch anywhere on this display. And what happens when we touch this hotspot well we will make it send its ID let's test this quickly. As you can see it's sending 01 When we touch it, we will make use of it when we write the program. Let's go to page one and add one more hotspot here. Now we can't make it send its ID again or else we won't be able to differentiate which one of them is sending the value one. So we will make it send zero to let's test it to this is that first page and it's sending a one by the way to change the page just type page and space followed by the page number and we are on page one now as you can see it's ascending to followed by some zeros. But we will program the microcontroller to receive only one byte from the UART so we will be getting to from this page. This completes the hotspot setup. Now on the page zero, I am going to add the blocks for the number and the float we need to add some fonts also. I have already created them in my previous projects.



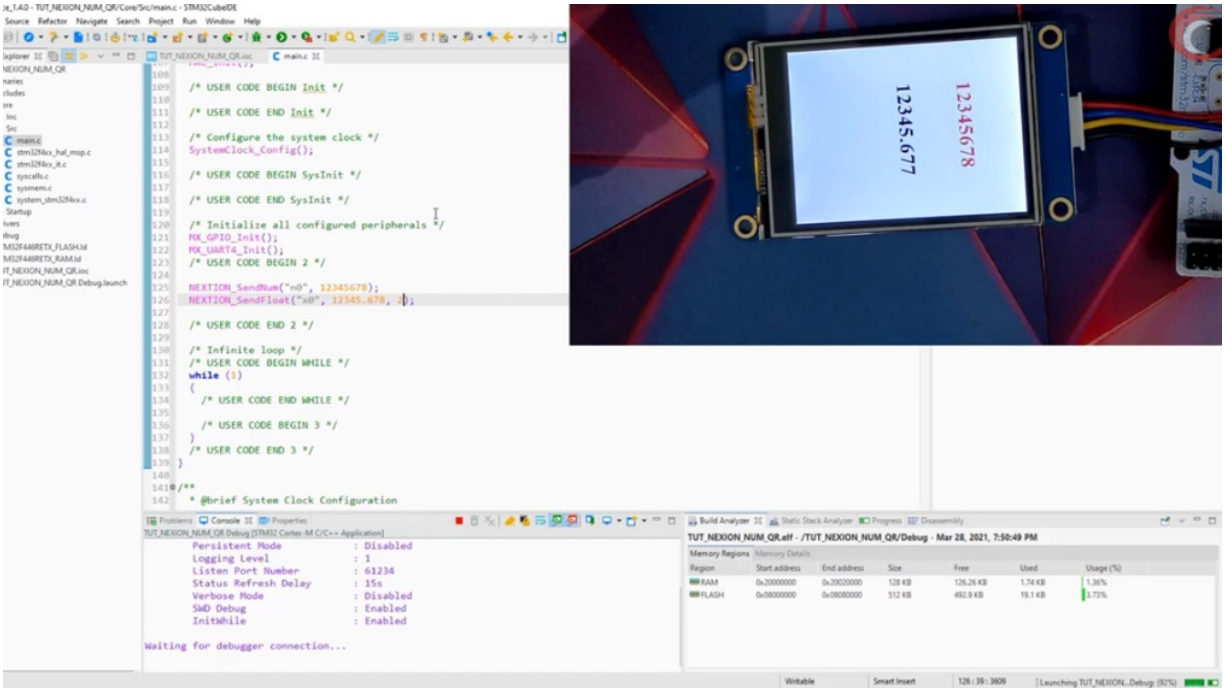
But anyway you can create the fonts from here now you can see the default values that is zero the number value is assigned 32 bit integer and you can see the limit for the values you can also change the format to other than decimal values. Now about the float values. As you can see here, the value is again a 32 bit signed integer. This means we can't directly write the float values into the display but we need to send the integer value instead. This parameter v v s one decides the position of the decimal point if I change this to to see the decimal point has been shifted to two places, and yes, this is the key to write the float values. Let's test these first I will write the number and you can see it's being displayed. Now let's write to the float. As you can see, it's displaying with two decimal places. And if I change the values of the VVS one, the decimal place shifts respectively. I hope you understood how these things are going to work. Now let's go to page one and add the QR code. Here we have the text field which the QR code will respond to and this is the maximum length of the characters, let's increase it to 50. Now remember one very important thing. As you see on the right, some fields are green and the rest are black. The green fields are the ones whose values can be modified during the runtime. But the blank fields should be fixed before loading the code into the display. This means that we can't

modify this length field once it is set, but we can modify the text anytime. So this is it. Now let's upload this to our display I am using you opt to do the uploading it's finished successfully. Now let's go to the cube ID and create a new project I will be using a 446 r e give some name to this and click Finish. In the cube MX first of all I am setting the crystal for the clock. Here I have eight megahertz Crystal and I want the controller to run at 180 megahertz I am going to use you out for to transfer data to the next one. Set the baud rate at 9600. Let's enable the UART interrupt also. This is because we will be receiving data from the next en display and it's better to receive it in the interrupt mode. Click Save to generate the project let's start writing the program now. Let's create a function to send the number to the display. The parameters are the object name and a 32 bit signed integer that you want to send create a buffer to store the string that we will send now as printf will copy the data into the buffer we will be sending the Value field the Value field is the value of the number and the object name is n zero for this number block. And now we will use the hall you ops transmit to send the buffer to the display.



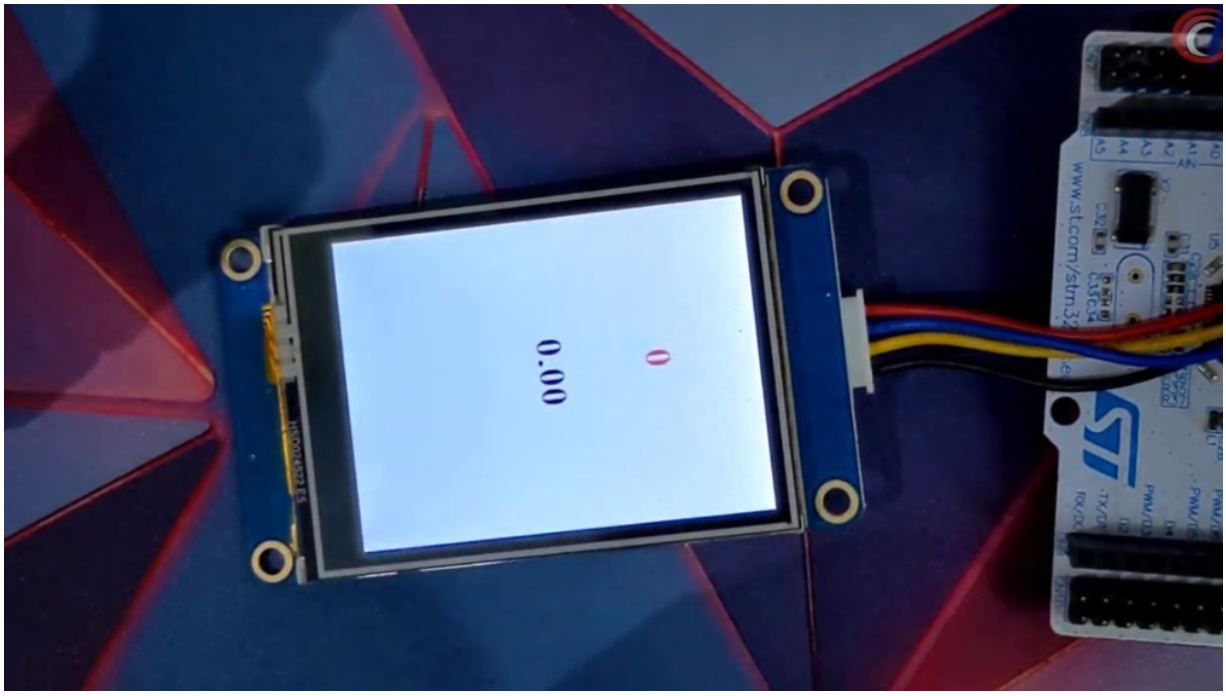
These end commands will be sent after every data to indicate the end of data transfer. So send the end commands to indicate the end of transfer. And finally free the memory from the buffer. Similarly, this is the function to send the float to the display here as you can see, the float also accepts the 32 bit integer as the value and v v s one sets the decimal places we will make certain changes in this function so that this function can take the float value as the parameter. But of course, you have to specify the decimal places you need in that value. Here first of all, we will convert the float to the integer by multiplying with 10 to the power of the decimal places. Then we will send to the display how many decimal places we need. This can be done by updating this v v s one field. And we will send this buffer to the display. Now send the actual number by using the same steps

but instead of VVS one we will be sending the Value field. Before going any further. Let's test this much path first. We have a lot of warnings because we haven't included the libraries. Here you can see the ID is asking us to include all these. We still have three warnings. This is because s printf accepts the character array so we can tie casted here so we are good now in the main function, let's send the number first, the object name is n zero and the value is 12345678. Similarly in the float function, the object name is zero and input some float value, I am looking for three decimal places let's run it now you can see the values are being displayed in their respective positions, the float value is not so accurate. As we are not taking any drastic measures for high accuracy, I will advise you to not use very high values in float. Keep your values in seven digits and it will work well. Now I'm changing the decimal places to two. And you can see it reflects on the display.



So this part is fine let's write another function for the QR code. Here also, the parameter will be the object's name and the pointer to the string that you want the QR code for we will use the same format like the others here instead of value field, we have the text field. Now we need to receive data also. To do that, we will use the interrupt and once the data is received, the interrupt callback will be called.

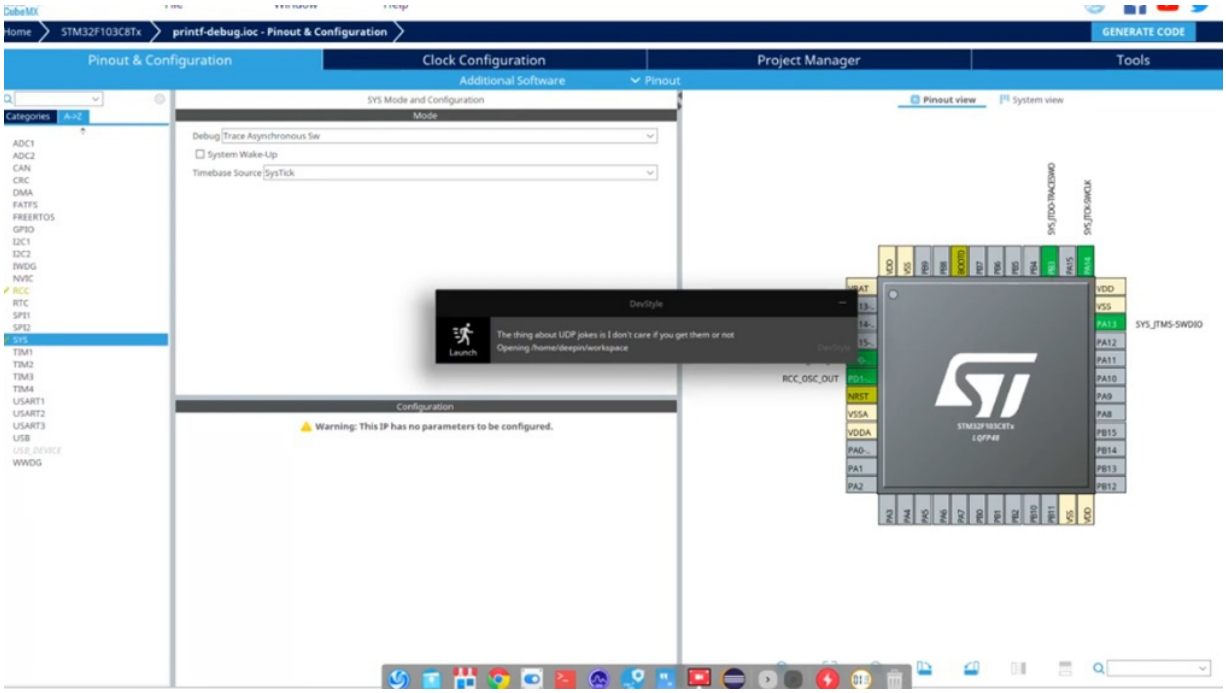
Let's define a variable to store the received data. Inside the callback function, we will first check if the data received is from the first hotspot or from the second hotspot. If it is from the first hotspot, which is on page zero, we will send the command to go to the page one. Now remember that we have to send the end commands also to indicate the end of transfer. Similarly, if the received data is from second hotspot, we will go to the page zero inside the main function, call the UART receive it and receive only one byte. Since this pre generated code disables the interrupt after each execution, we need to put it in the callback function also to manually enable it again. Okay, let's check if the hot spots are working. As you can see when I am touching the empty space which is the hotspot the pages are changing, so everything is fine till now. Now let's take it to the next level define some variables we will update these variables inside the while loop and then send them to the display this process will repeat every 500 milliseconds. Let's run it again. You can see the values are updating but the float is updating really fast. Let's modify this a little more. In the callback function, I am resetting their values before the while loop begins, let's send some string to the QR also. Now it is perfect the hotspots are working well. The values are also resetting whenever we move between the pages. So I tried to test the QR code and it's not working. After debugging a little more, I have found that we need to do some correction in our code. Let's see it. Right now we are on the page zero. And if we send the string to the QR, it is returning the error. So let's go to the page one.



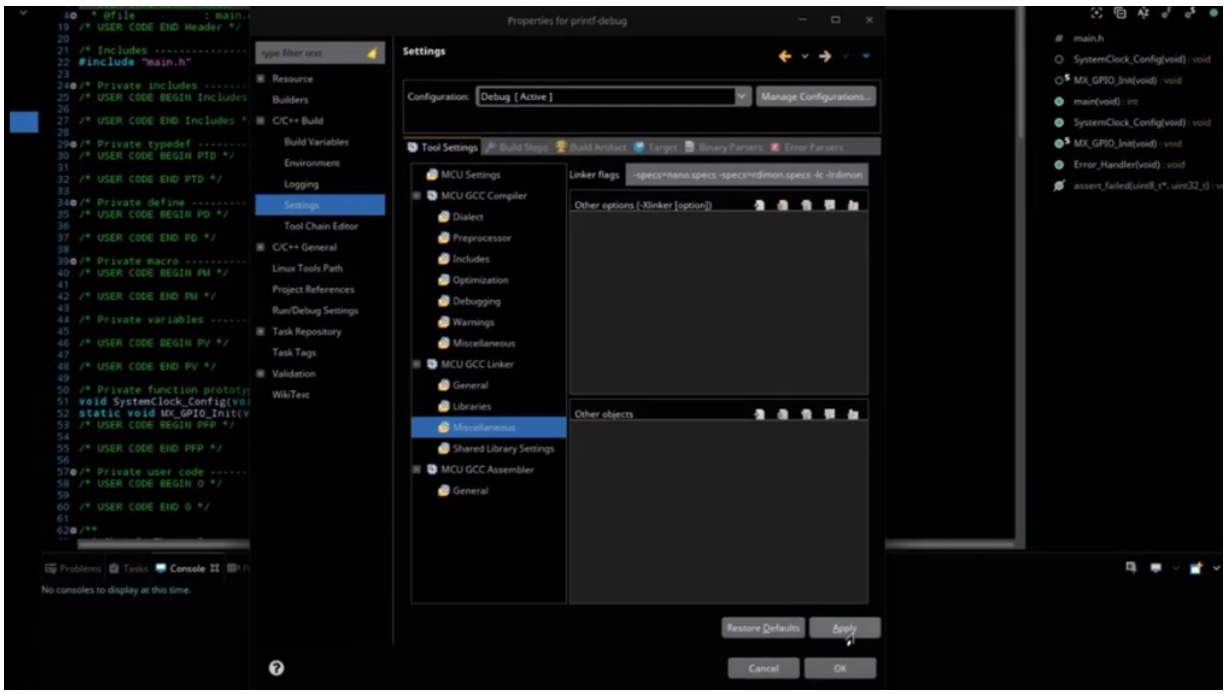
And now send the string you can see it's updated. So this means that before sending the string to the QR, we need to be on the page that have the QR also, the string should be in the double quote. And before sending this string, we need to go to the page one. Let's run it now. I am using my phone to scan the code and you can see the string here. So the QR code works well. You can enter any string here and it not necessarily needs to be a website. Just remember to go to the page where the QR code is. This is basically it for the next tn display. I will make one or two more projects about multiple page menu bar some fancy buttons, but this is it for the feature related projects. I have the basic model and it only supports these features.

PRINTF DEBUGGING USING SEMIHOSTING IN STM32 SW4STM LIVE VARIABLE CHANGE

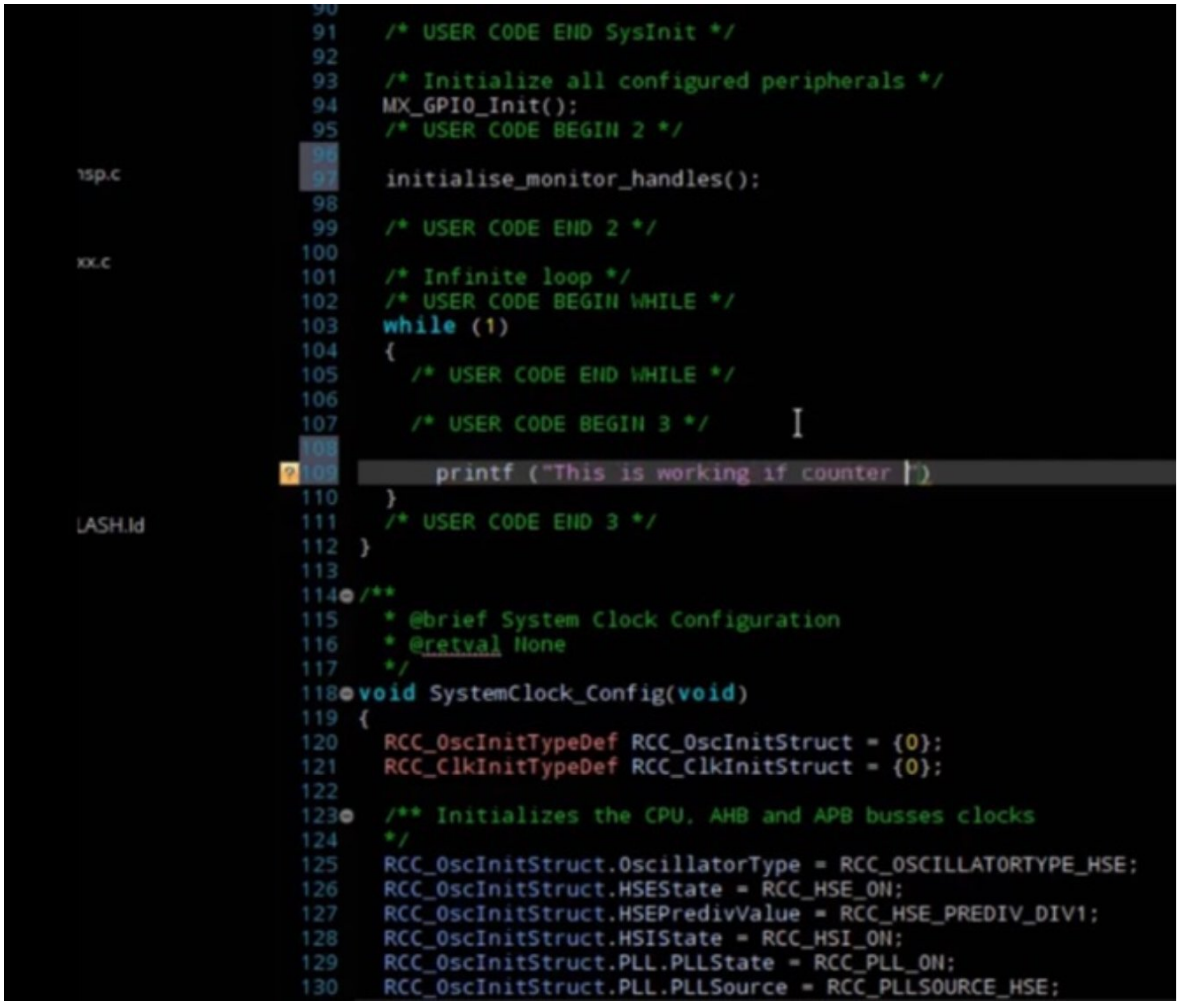
I will show you how to use prin tf debugging in STM 32. I am using SW for STM 32 ide. I am not sure about any other IDE, though you can test it yourself prin tf debugging is very useful in Eclipse space tidy ease. As these IDs don't support live variable change, we have to pause the debugger to observe the change in the values using prin tf we can see the variable changes without halting the CPU I am using STM 32 F 103 controller but others should be fine too.



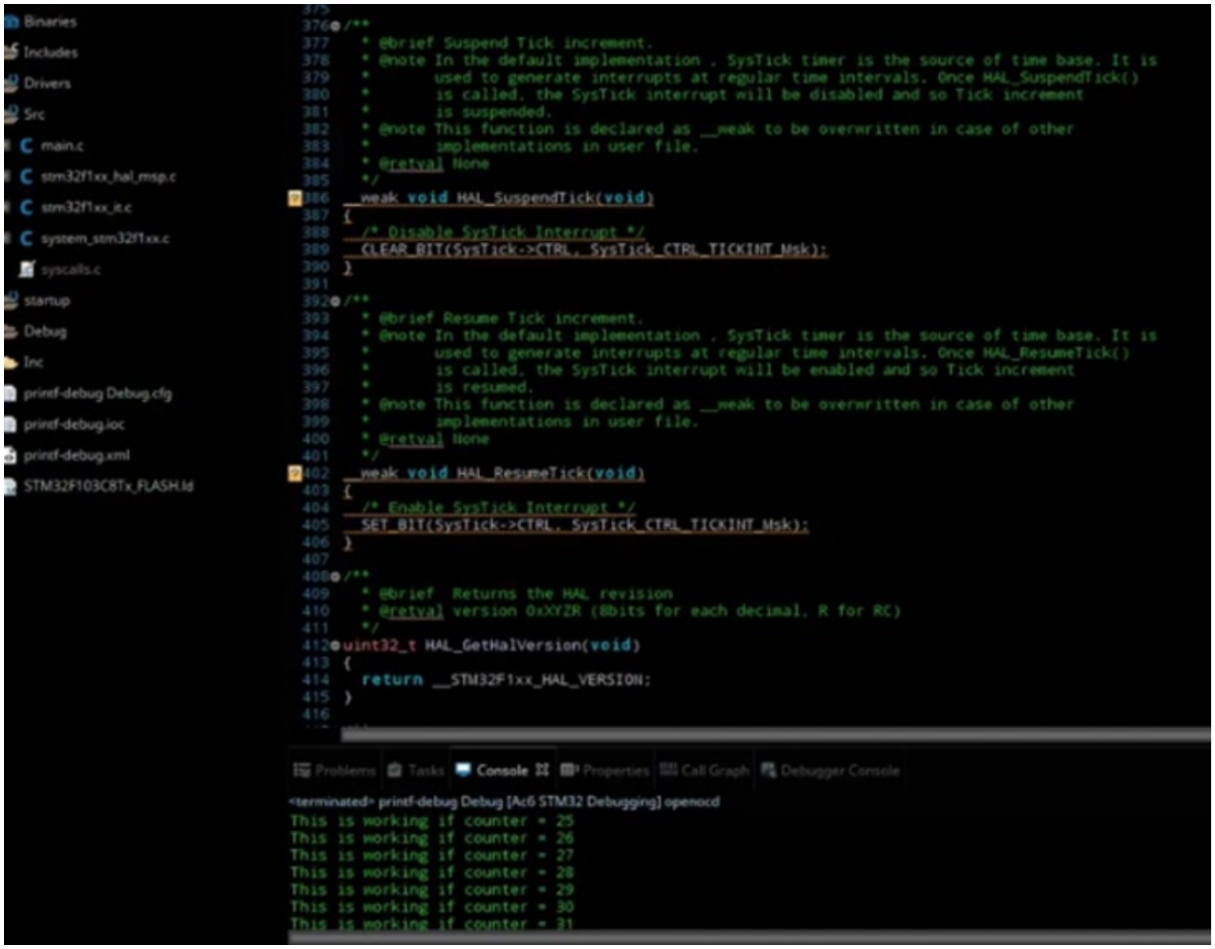
First of all in the serial debug we have to enable the trace a synchronous SW This is it for the setup part for cube MX rest is our usual setup for the clock. Also make sure that this PB three pin here is not connected to anything or else this will not work now generate the project and open it here are the instructions that we need to follow first of all copy this command here go to Project Properties CC plus plus spilled click Settings under GCC linker click miscellaneous we need to copy that command here.



So add a space and paste it next we need to exclude sis call C from the build so right click on it go to resource configuration and click exclude from build Next, we need to put this function in our code make sure you put it outside the main function Next we need to initialize the monitor handle before we make any call to prin tf function. Now I will create a variable which I can increment during the prin tf and I will name it counter.



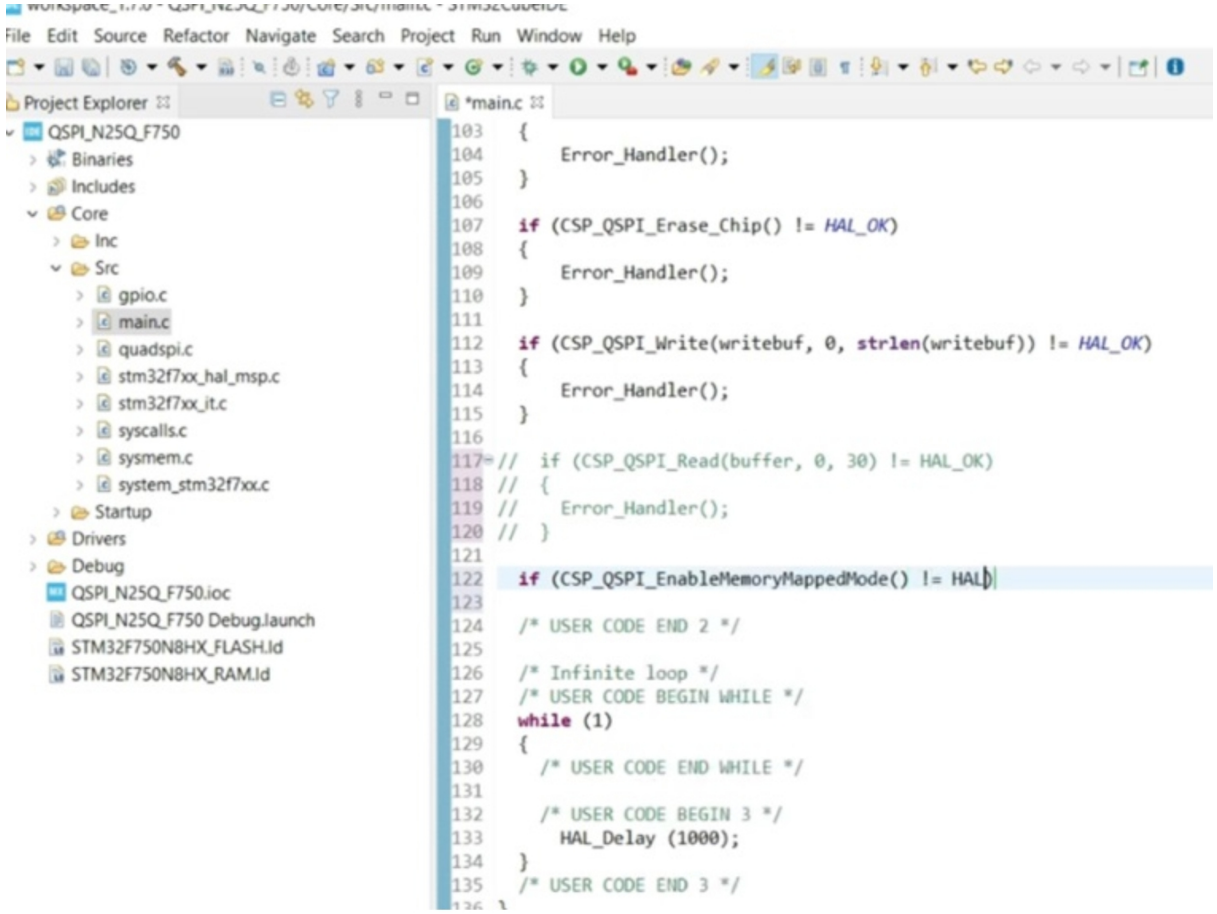
Next, I am writing printf inside the while loop so the variable can auto increment and adding a delay of 500 milliseconds Next compile the code and the bucket of course we will have our usual problem of resetting we need to do one last thing though copy of the monitor arm semi hosting command go to debug configuration startup tab and paste it go to generate a script and set the reset mode to software system reset so we are done now hit the start button and observe the printf values.



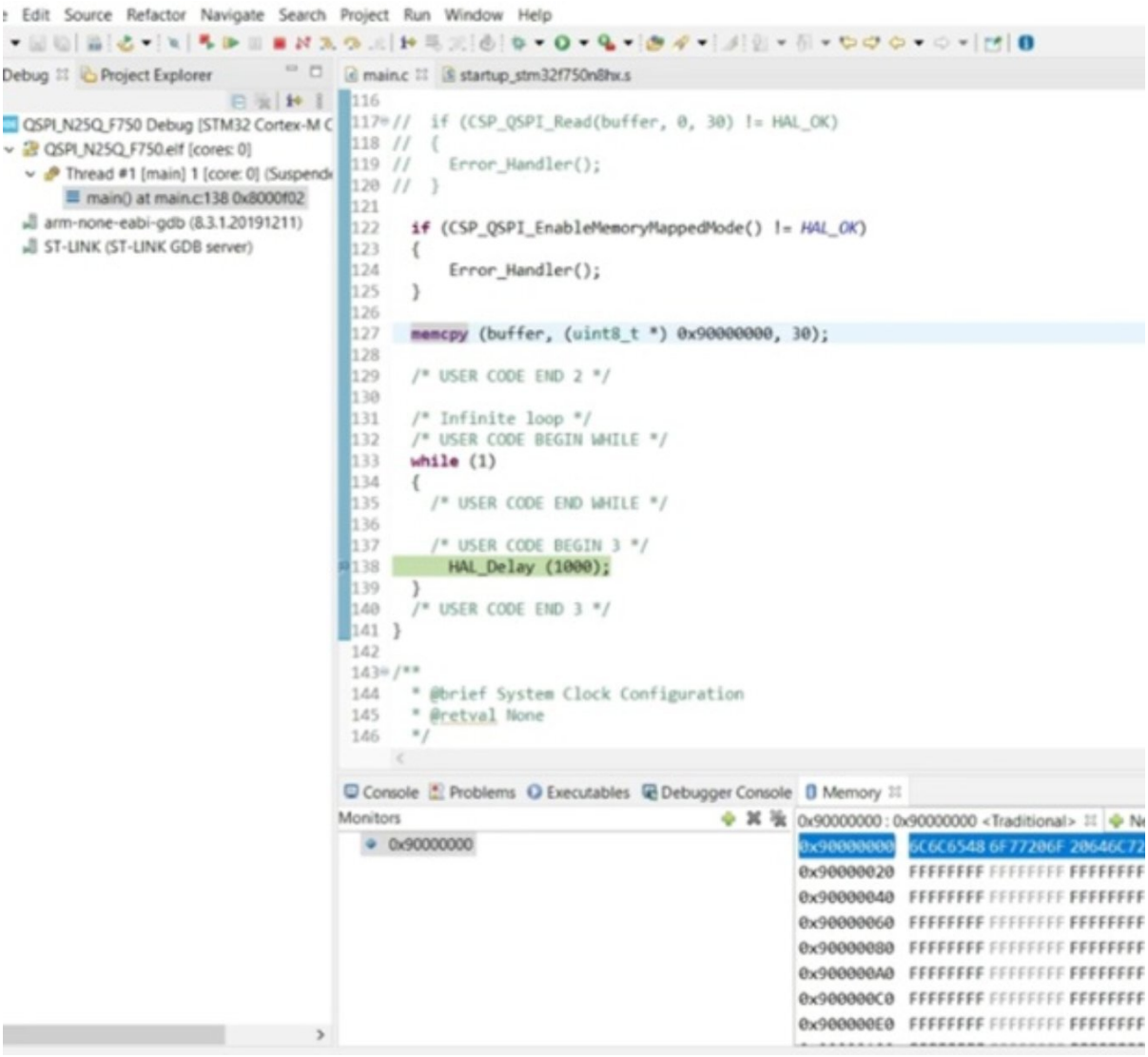
As you can see, we are getting the results on the debugger console itself. The counter variable is incrementing and everything is as expected. Let's reduce the delay and make it a little faster.

QSPI IN STM32 BOOT FROM EXT MEMORY XIP N25Q

We will see the memory mapped mode. We will also see how to use the external flash memory as the internal and how to run the application from the external flash. This is called execute in place x IP. Watch the previous Qasba project if you haven't watched it yet, I am going to start with the same project that I created last time. So let's open the cube ID and import the previous project here you can see I have the same code that was present last time. Here we will not perform the read from cue SPI.

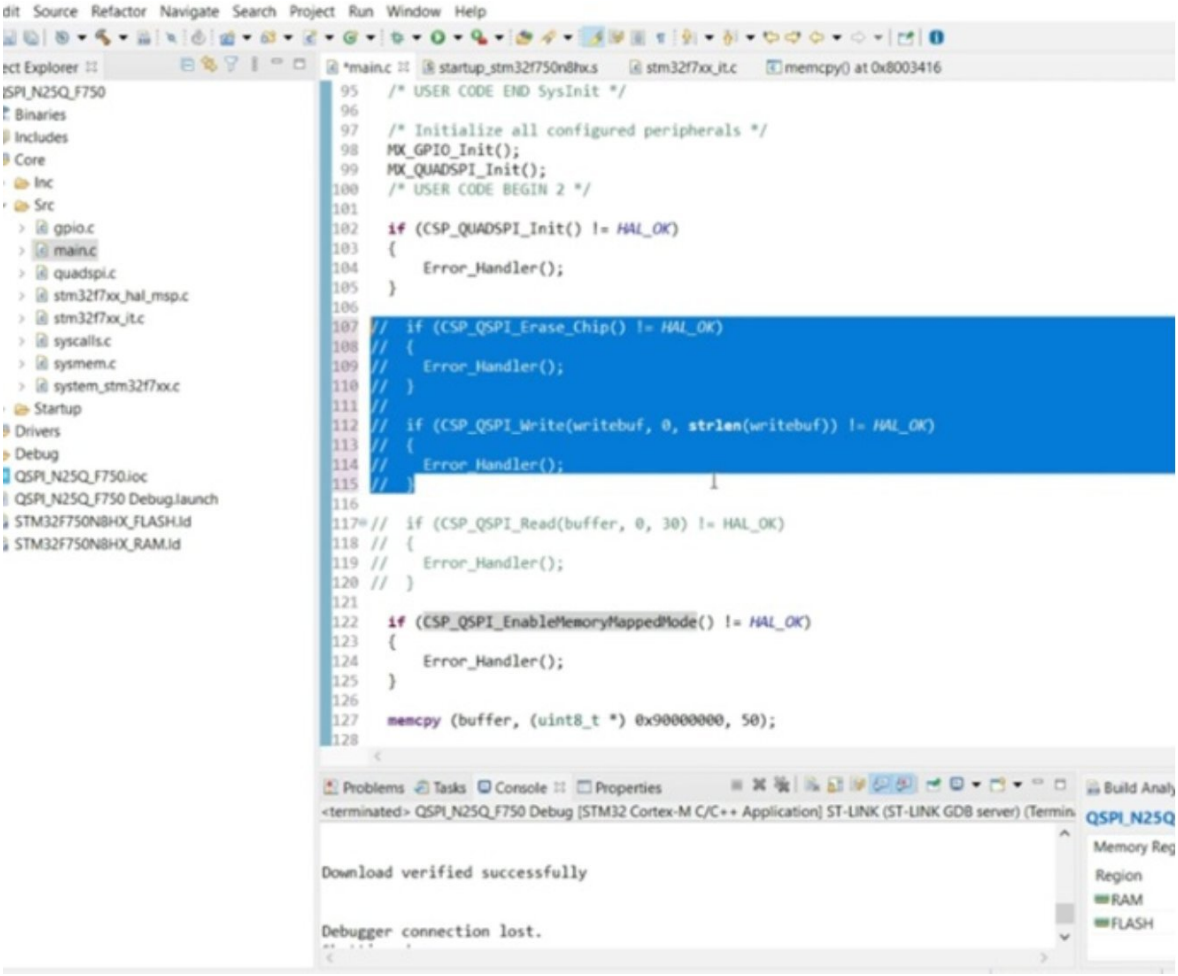


But instead we will enable the memory math mode once the memory is mapped, we can treat it as just like the internal memory. And in order to read the data, now I am simply using the mem copy function let's build and test it let's put a breakpoint in the while loop we hit the breakpoint, that means all the functions were executed properly.



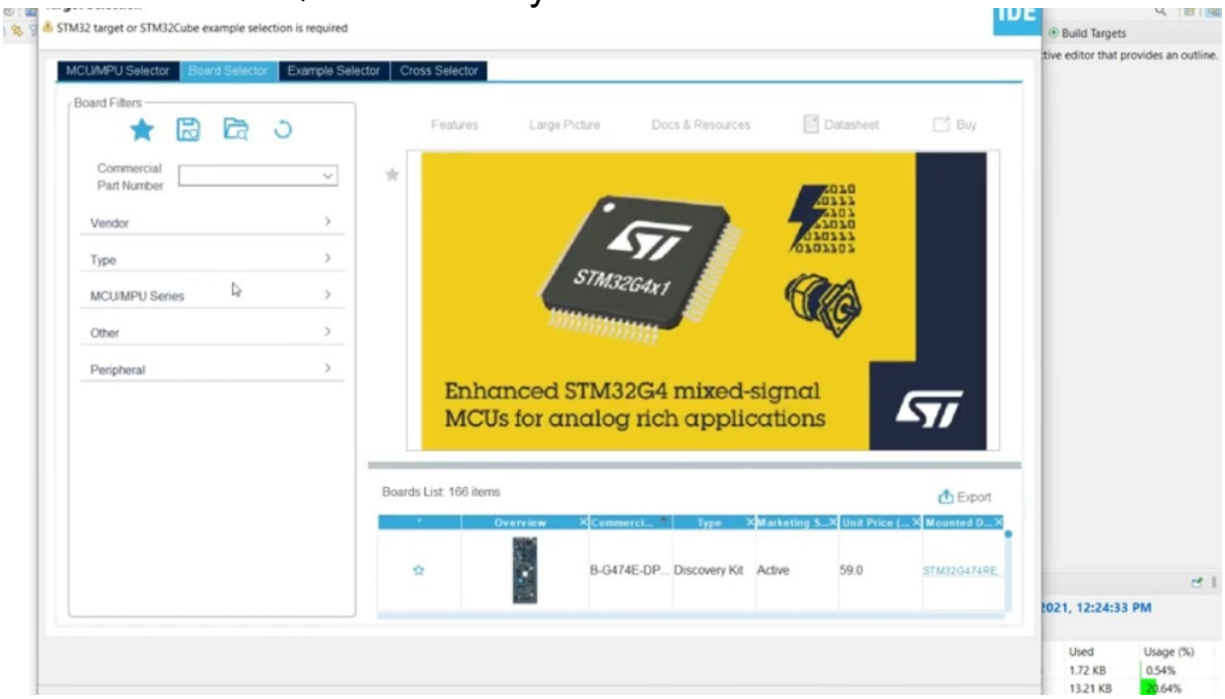
If we check the buffer here you can see the data that we wrote in The Flash memory the fact that we can use the mem copy function means that the external flash is an internal memory now and we can also view it in the memory window here you can see the data just to verify this whole process, I will not enable the memory mapped mode. And let's see what happens now. So we got the heart fault and it is caused by the mem copy function. This is because we are trying to access the internal memory at a location which is not internal. If you try to browse the memory region, you can see we can't access it so we need to enable the memory mapped mode let me change this data a bit here. And this time we got the data in into the buffer also, we can access the memory in the memory window.

But remember one thing the memory map mode is for read only. Once the memory mapped mode is enabled, you can not write the data to the flash. We will be using this mode to either run the application from the external flash or to increase the internal flash memory for our controller. Let's see how to use it. First of all, we will disable these function make sure you disable the chip arrays function.



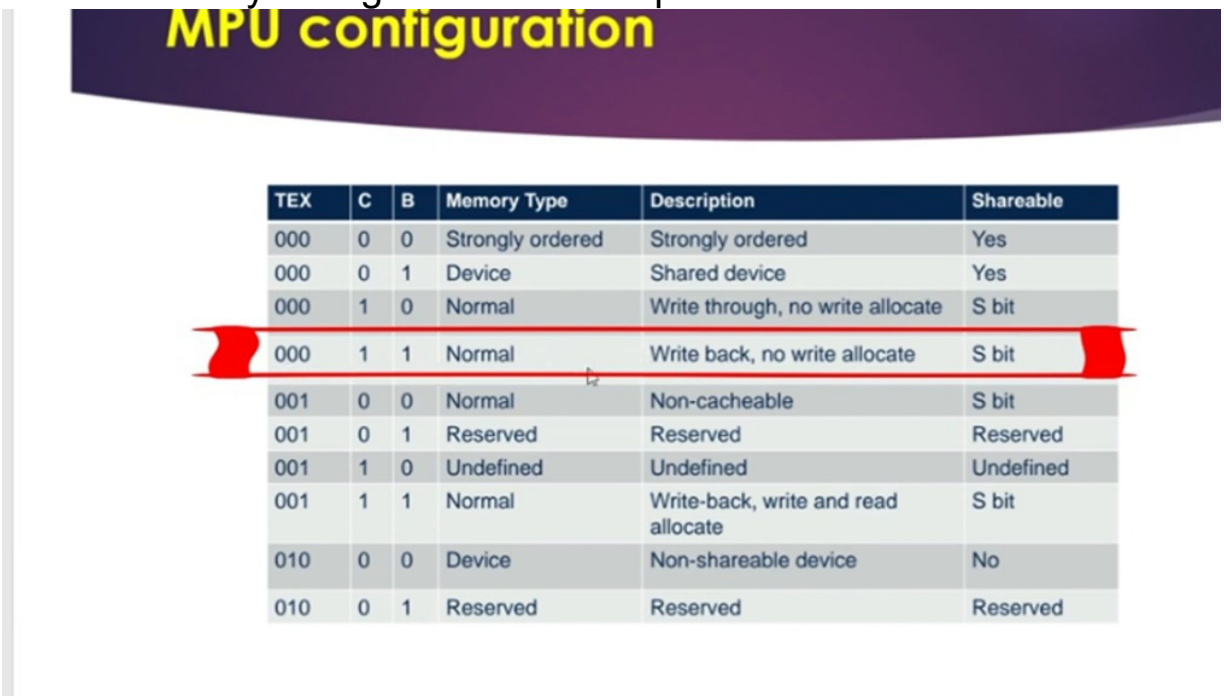
This is because when we will load the application into the external flash first this code will run and then it will make a jump to the application. So if the chipper raise is enabled, every time this code will run, it will erase the application code from the external flash. So make sure that chippy arrays it commented out all right here we are defining a function that will make the Jump define the address of the Qasba here. Now after the memory mapped mode is enabled, disable the cache if you are using them. Next we will disable the

SysTick interrupt and finally the code to make the jump. Here we are setting the address for the reset handler. This one is the address for the stack pointer. And then we call jump to application to make the jump. Let's try to debug this to see what happens I am putting a breakpoint at the jump function. As you can see, we got into the hard fault and this is okay, since there is no executable code at the provided location yet. All right, now we need to write some code and store it into the Qasba memory.



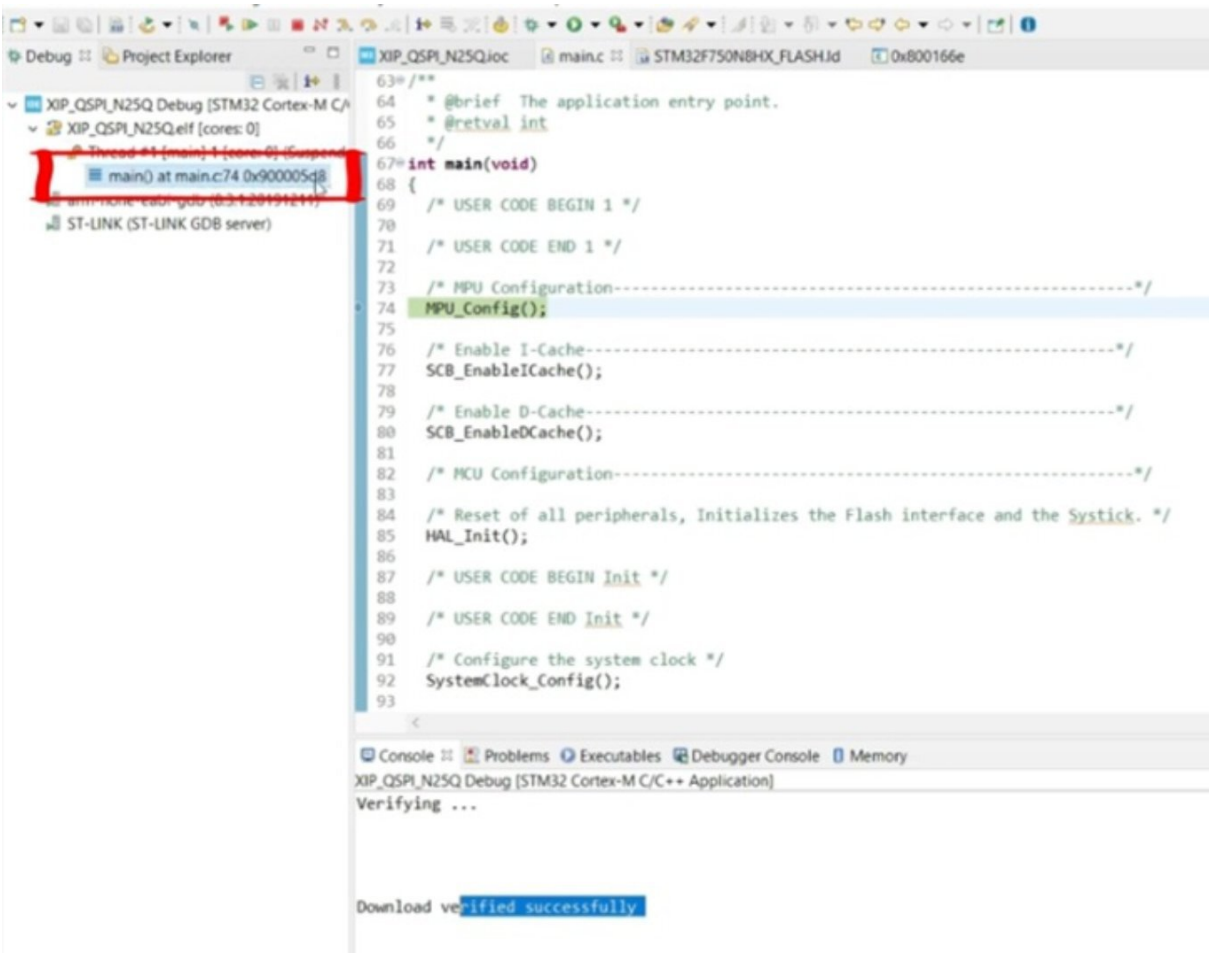
I am creating a new project for that. So choose the board give the name and click Finish. First things first select the proper clock. This application will execute from cue SPI flash. So make sure the cue SPI clock is same as the previous clock. Now we will enable the cache. Let's configure the MPU also enable the region the address of the USB flash. As I mentioned in the previous project, the size is 16 megabytes. If you remember the memory configuration project from the Cortex M seven playlist here I have explained the memory setups we will set the normal region for the cue SPI flash memory. So it should be cashable and buffer will with texts equal to zero first disable the instructions and then set cacheable and buffer will we will create one more region in the same address the size will only be one megabyte. And we will keep the same settings just enabled the

instruction access. This is because our code is going to execute from this flash so we need the instruction access. So that's it for the MPU configuration. Now before setting anything up, make sure you enable the quad SPI Also make sure the pins are correct. We don't need to do any configuration for the queue SPI.



We are doing this so that we don't configure these pins for some other use, or else the cue SPI communication will fail. Once all the setup is finished, disable the cue SPI as it is already enabled in the first part of the code. All right, all the setup is finished. So click Save to generate the project. Now here we will first edit this system file. Come to this system and it function and here we will reset the RCC configuration register. Next we need to relocate the vector table and we will relocate it to the queue SPI base. You can also provide the flash memory address here. That's it for this let's go back to our main file, I will just write a simple code where the counter will increment every second. So create a counter variable and incremented inside the while loop. We need to do one last thing and that is to change the flash memory address to the cue SPI flash memory. The size is 16 megabytes. Let's build it once to check for any errors no errors and we are good to go. Here you can see the flash has been relocated, and so are the others things now if we want to debug this,

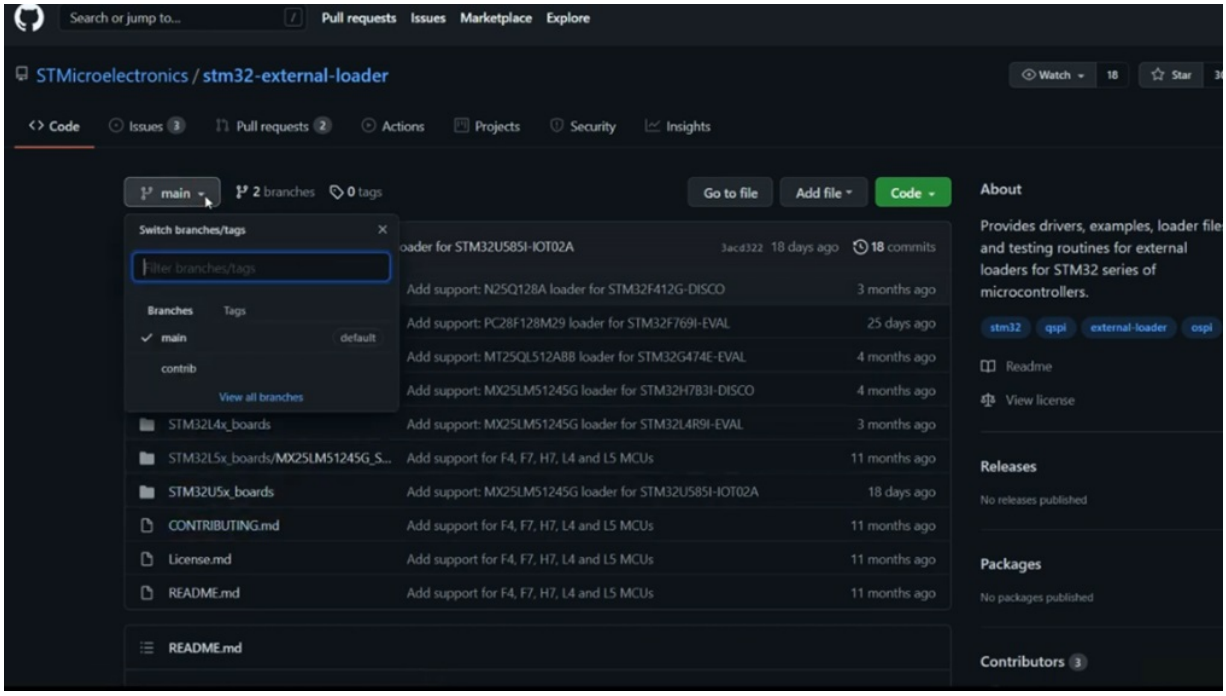
we need to use the external loader. So go to the debug configuration, create a new configuration for this project. Now go to the debugger tab, check the external loader and click Scan. Here you can see the list of all the loaders available. Here it is n 25 q four F 7508. Click Apply and click debug. Notice that the debugger will not halt in the main function. But here we see the download is verified you can just press the reset once and now it came inside the main function. You can see the memory address here corresponds to the cue SPI flash memory let's add the counter to the live expression you can see the counter value is increasing. Also note the memory address we can also browse the memory here. So the cue SPI flash memory is working as our internal flash and also we were able to execute a piece of code from it. This is very useful for some microcontrollers with low flash memory like this one here itself, it only have 64 kilobytes of flash, which is not enough for majority of the things. So this way we can increase the flash memory.



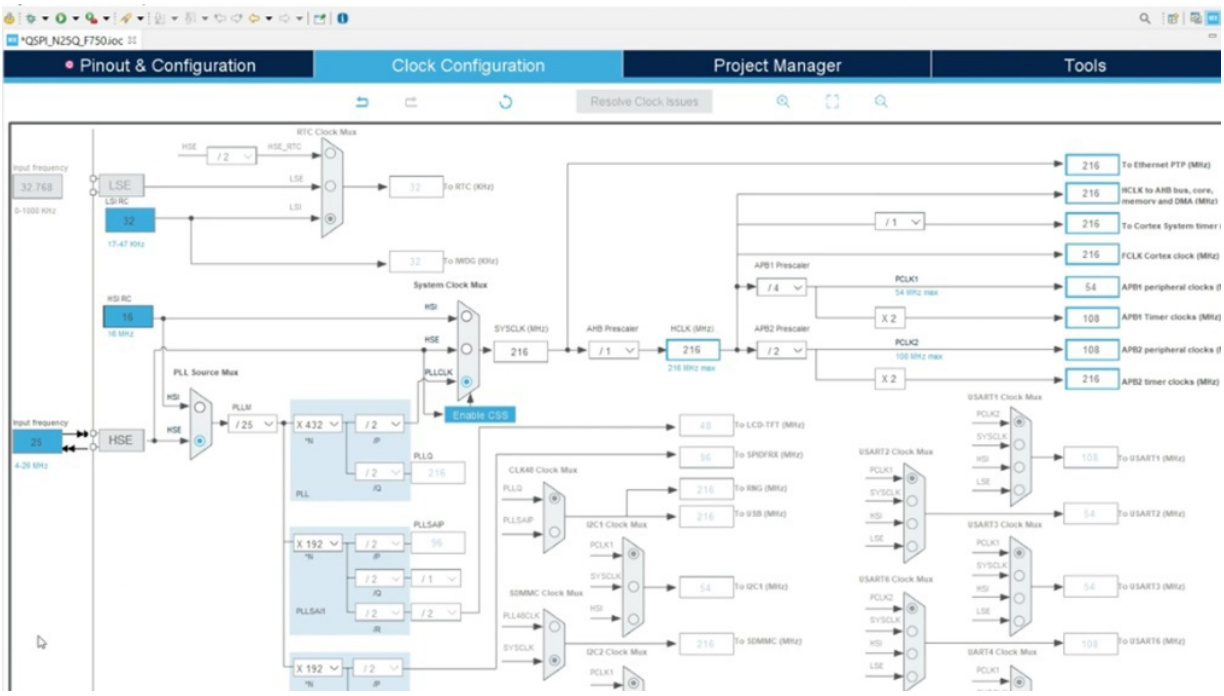
We saw that executing the code from external flash involves two things. The first is to initialize the cue SPI and set it in the memory mapped mode. And the second one is the code itself which will execute from the flash memory. You need two different projects for these. But I think there is some way to make it work in a single project itself. I will try to look for it. And if I do succeed, I will release another project on it too. In the meantime, do let me know if you want to see the Q SBR tutorial for Mt. 25 flash memory on the H 745.

QSPI IN STM32 WRITE AND READ N25Q

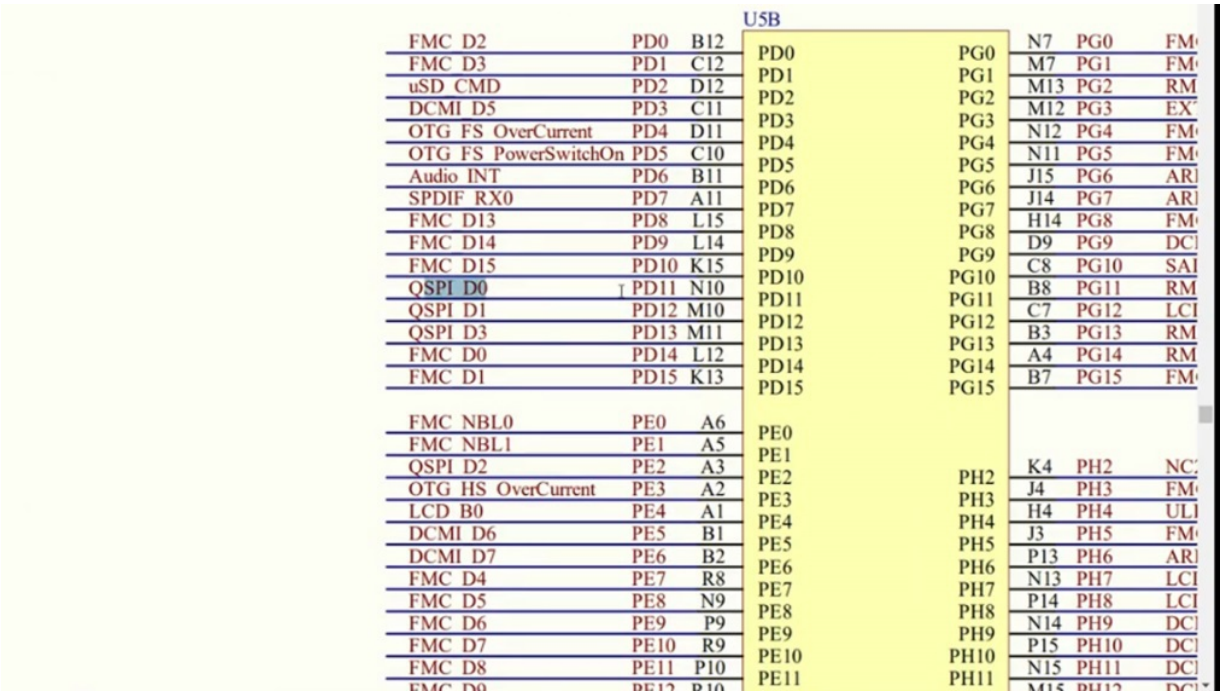
Quad SPI is mainly used to interface the external flash memory and it's very fast compared to the regular SPI chips. For this reason, the quad SPI flash memories can also be used as the internal flash memory of the microcontroller. Our focus will be on this part how to use the flash memory as the internal flash. In this project, I will focus on the setup process and how to read and write data to the external flash. The next project will cover the mapping of the external flash as the internal memory and also how to load the application from it. STM 32 microcontrollers mainly have three different types of flash memories, and 25 Q, Mt 25 and MX 25. Out of these three memories, I have access to n 25 q and Mt 25. So I have written the codes for these two memories, but I will show you how to make it work for the MX 25. Also, as I don't have the access to MX 25 series, I can't say if it will work for it.



So let's start the project now. SD Micro electronics already have a GitHub tutorial on these memories. I will leave the link to this in the description. Here you can see there are three Q SPI drivers. I have also uploaded the drivers for two of them on my GitHub page. These files are slightly different, but I have kept the formatting same as that of the SDS drivers. Let's see how to make these work. Create a new project in cube ID. I am using F 7508 Discovery Board for this tutorial. Give some name to this project and click finish first thing we are going to do is set up the clock select the external high speed crystal for the clock. If you configure anything wrong here, the project won't work. So make sure you configure the right crystal frequency as per your board as you can see, the n 25 q can reach up to 108 megahertz we will take advantage of it I am running the system at 216 megahertz. Even if you run into a bit lower frequency, it's alright since 108 is the maximum it can also perform well at slightly lower frequencies. So the clock has been configured to run at 216 megahertz. As you can see here in the datasheet the quad SPI is connected to the HB one which runs at 216 megahertz also.

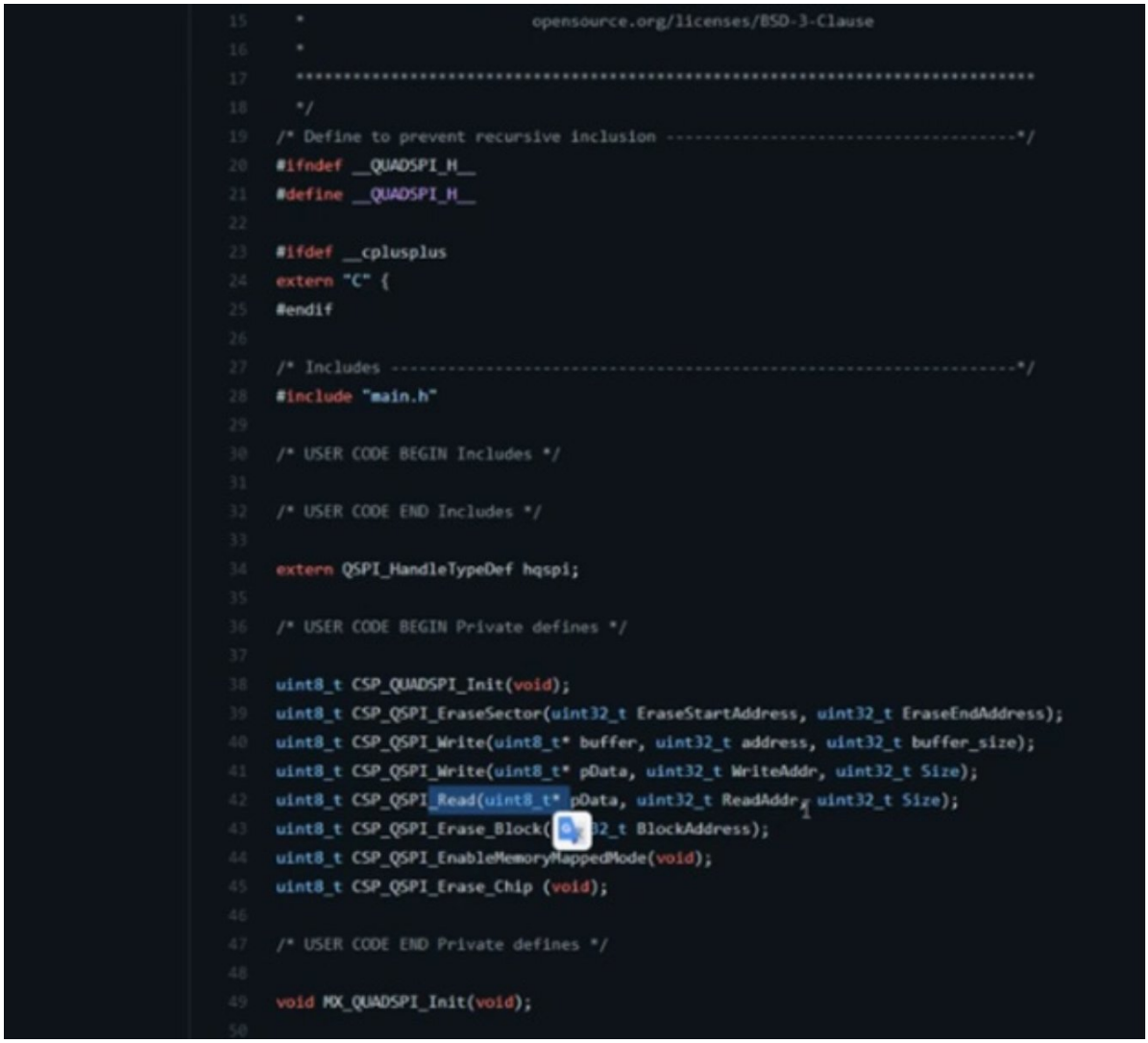


So right now the quad SPI clock is also 216 megahertz which we will reduce later by using the prescaler All right the clock is configured now if you are using Cortex M seven enable the cache. Now we will go to the quad SPI and enable the quad SPI lines for communication this here is the schematics for the F 7508 board. And here you can see it only have one Q SPI memory chip and this is why I can only use bank one while enabling the quad SP I. Next we need to make sure that the pin connection is correct as per the schematics. I have said it a lot of times during the Ethernet projects, and some people still configured the wrong pins. Cube MX sometimes configures the wrong pins for these modules, so you need to cross check the pins with the schematics. Then we need to make sure that the pins are configured to run at highest possible frequency.



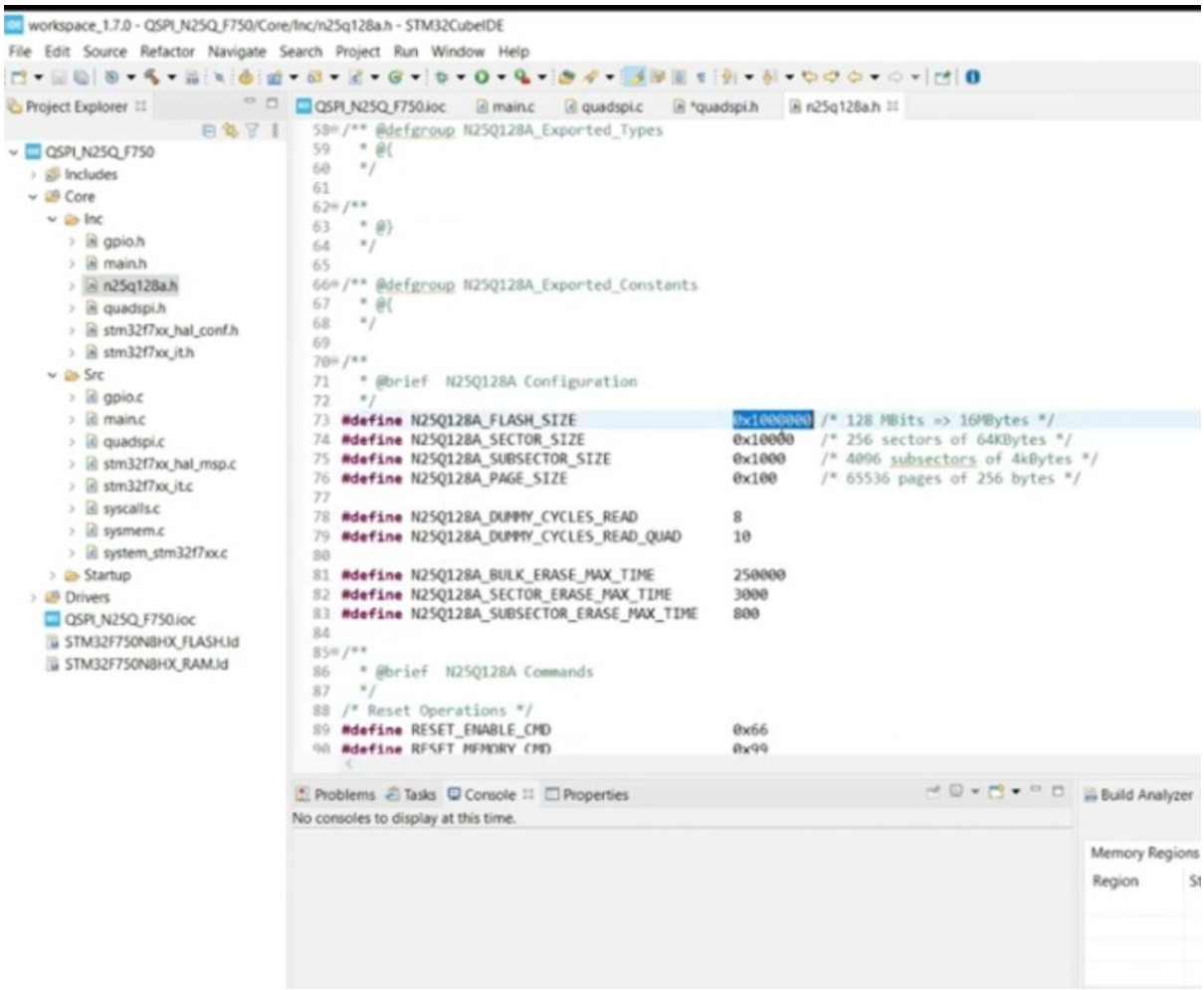
Press Ctrl A to select all the pins and go below and change the output speed to very high. Now let's configure the parameters. The cue SPI clock is 216 Mega hertz right now, and this en 25 Q can run at 108 megahertz max. So, we need to use the prescaler of two to bring the Q SPI clock to 108 megahertz, here we will enter to minus one that is one, this configuration we are doing is for the single transfer rate mode, just set the FIFO threshold to four. Next is the sample shifting. It also depends on the mode we are using. For the dual transfer rate, there will be no shifting and for single transfer rate there will be shifting by half cycle. Flash size depends on the memory size. For example, I have the 128 megabits which is 16 megabytes of flash storage, two to the power 24 gives us the 16 megabytes. So, the flash size will be 24 minus one that is 23. Chips select high time, let's keep it to six cycles. clock mode should be low flash ID is one since there is only one flash memory. dual flash is disabled as there is no second flash available. That's it for the configuration. One last thing we need to do is create separate files for the peripherals. Like I mentioned in the beginning, I am trying to keep it as for the STS QSP I drivers and that's how they have put it enable it here. Now click Save to generate the project if you check the queue SPI drivers by the s t they have separate files for Q SPI

dot c and dot h. So if you want to use these drivers, you need to put this code in the user code zero part and the rest of them in the user code one part and the same goes for the header file also you need to put these functions wherever they are defined in these files. Of course you need to change this memory size now if you take a look at my GitHub, the functions are almost the same. The only change I have made is I have added the read function so that we can read the data from a particular address in the memory now to make it work, you need to download these files first I have them here.



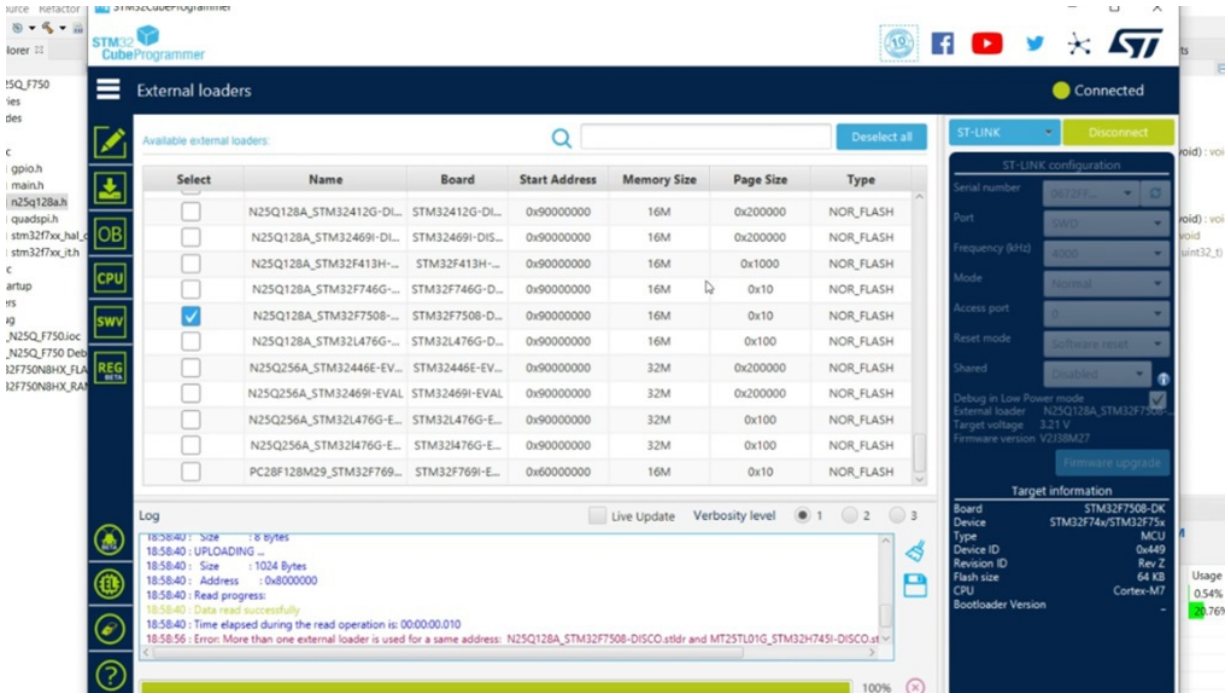
Now we will copy the n 25. Q header file in the project folder the rest we will modify let's start with the QS p i dot c file All right, first we will put these functions in the user code zero. And then we will copy all

these in the user code one. Now let's see the header file the right function seems to be defined twice. Let me delete one and I will update the GitHub for this we need to copy these private defines only So that's it. I forgot to include the n 25. Q header file here in the n 25 Queue header file, you can see the memory size is defined here, you can change it as per the memory size for your cue SPI device. Let's build it once to check for any errors. All right, we are good to go, we will initialize the queue SPI first then erase the entire chip, then we will write data to the memory, we need to define the data first. Let's create another variable to store the data read from the memory.



So, we will write this data into the QS p i at the address zero, then we will read the data from the same address. The write and read address is zero, which means that it will be the start of the QS pi

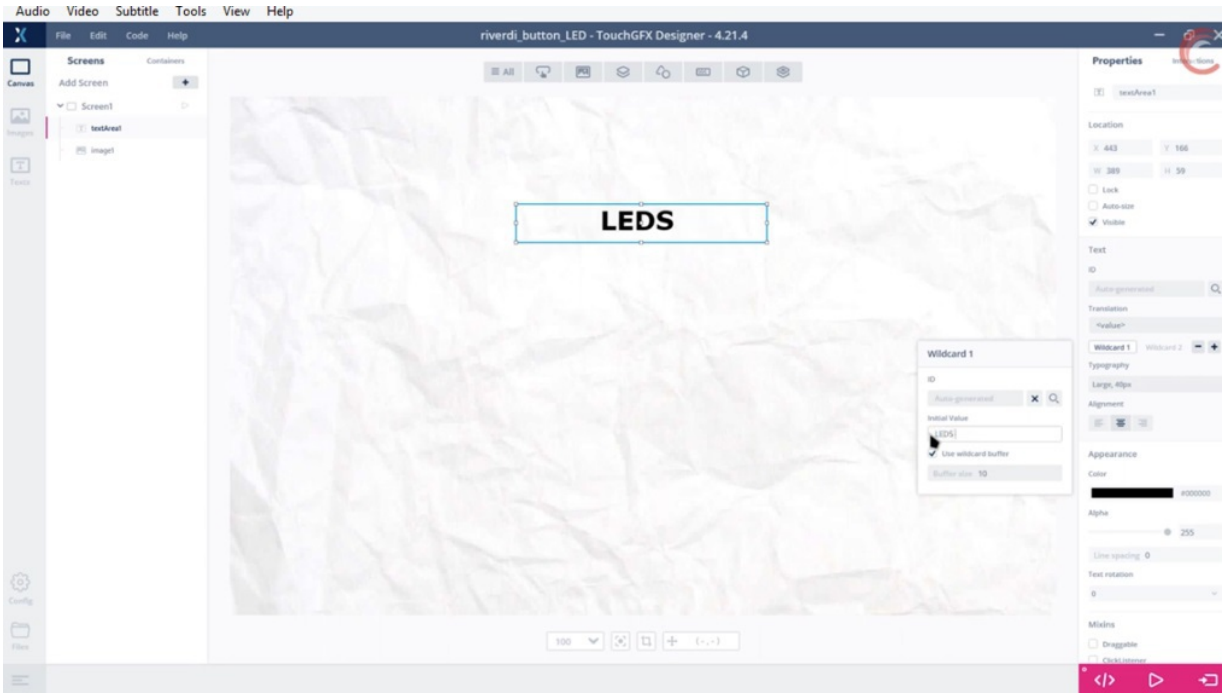
address, which in most cases is 90 million hexa. I am saying most cases because there are very few microcontrollers where the QS P I address starts at a 0 million hexa. Few warnings, but that's all right. Let's debug it. Now. I am putting a breakpoint at while and one in the error handler. Just in case if any of the functions fail, it will hit the error handler. Let's run it now. We hit the while loop which means the functions were executed all right. Read buffs do contain the data that we stored in the memory we can also check the data in the cube programmer just connect the controller. Now click the external loader and search for your controller here you can see that the QS P I address starts at 90 million hexa. I didn't uncheck the others. Let me do that quickly. All right, now it's fine. Now we will enter the cue SPI address here and hit enter and here you can see the data that we stored in the flash memory. Some of you guys always have problems while storing the numerical values. I will show you the simplest approach to it. Let's say the number is 1234 create one buffer also. Now we will use the S printf to convert the number to the string. And we will store this string into the flash memory. Include the stdio for the S print F, just ignore these warnings. And let's debug it. Here we hit the breakpoint and you can see the string of the number that we stored you can later convert it back to the integer format. I know this is not the effective way of storing the numbers. But the purpose of these Q SPI tutorials is not to store the data into the flash anyway, we will use this flash memory as the internal flash memory so as to compensate for the low flash in some microcontrollers. For storing data into the external flash, I will make another project using the wideband ICS as they are widely available, and we can simply use SPI for that purpose. Still, if you want to store the numbers into the flash, you need to use the union method. You can check out the E P rom code on my GitHub and see how I have used you In to store the number and floats.



Now, similar to this n 25 q, there is the code for m t 25 You need to follow the same steps here too if you check the initialization function, I have kept everything similar to what I did in the end 25 Q and in the header file to the similar function are being used. I have h 745 Discovery Board, which have the two flash memories each 512 megabytes in size. So, I have modified this particular code to work with dual flash also, the steps are similar to what you saw in this project, just in case if you want to see the working for Mt 25. Also, let me know in the comment section. I will make another project for it. Or else in the next project, we will see the memory mapped mode where these external flash memories will be treated as the internal flash. We will also see how to load the application from these external flash memories. That's it for today. The code is on the GitHub. So get it from there. Leave comments in case of any doubt. Keep watching and have a nice day ahead.

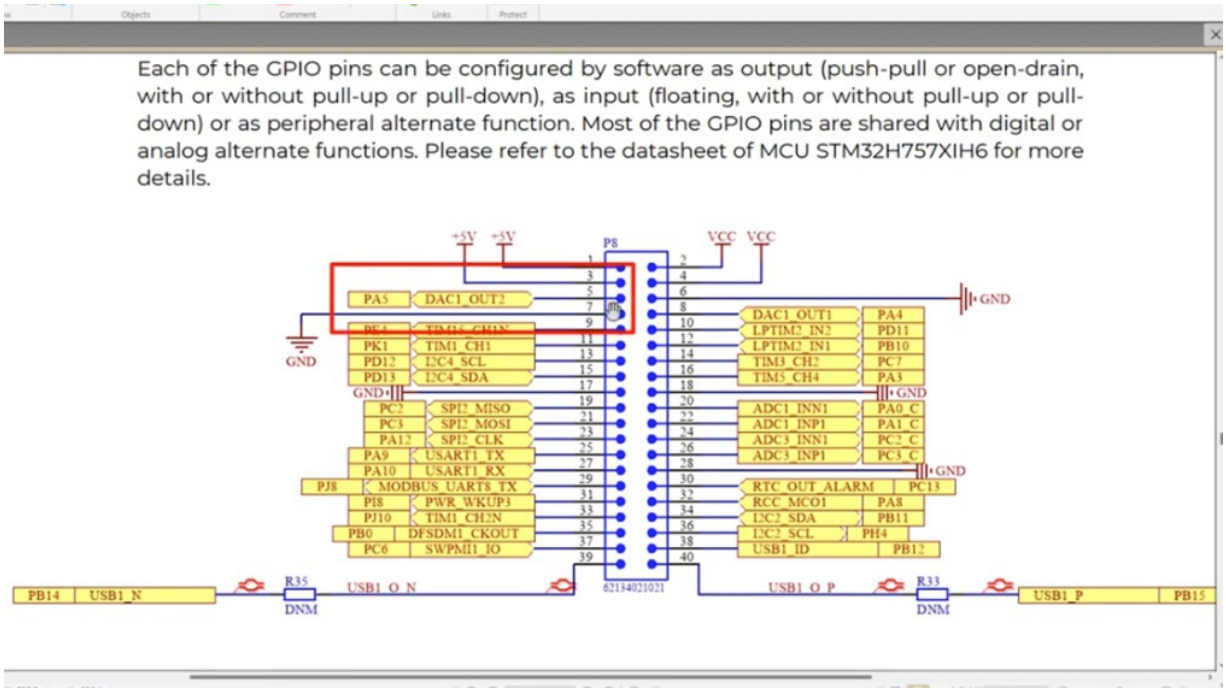
RIVERDI STM32 DISPLAY HOW TO CONTROL LED USING BUTTONS ON THE DISPLAY

Today we will see how to use the button on the display to control the LED attached to the MCU. This seems like a very easy topic, but if you have a riveted display, you know how challenging it is. The method I am going to use isn't going to be an easy one, but it will be a universal method for a lot of things. You can use the same method to display the data from UART any sensor pretty much anything I have covered so far in the touch GFX I am using the latest version of the touch GFX that is four point 21.4 But remedy has mentioned that they only support the version four point 20 I have already tested this version and it works fine. Although you won't see the BSP folder generated like it used to generate on the version 20 I am not going to use any BSP replated files, so this is fine with me. Here I am creating a new project with the activity display. First of all I am adding a background image to this project. I already have a 1280 by 800 Ping image for the background. Let's add a text area to display the LED status. I don't want the auto size, choose the alignment, the typography, et cetera. Now go to Text typography and add the wildcard range to this typography. We will use the wildcard for the text area and the buffer size will be 10 Bytes the default status for the LED is set to off.



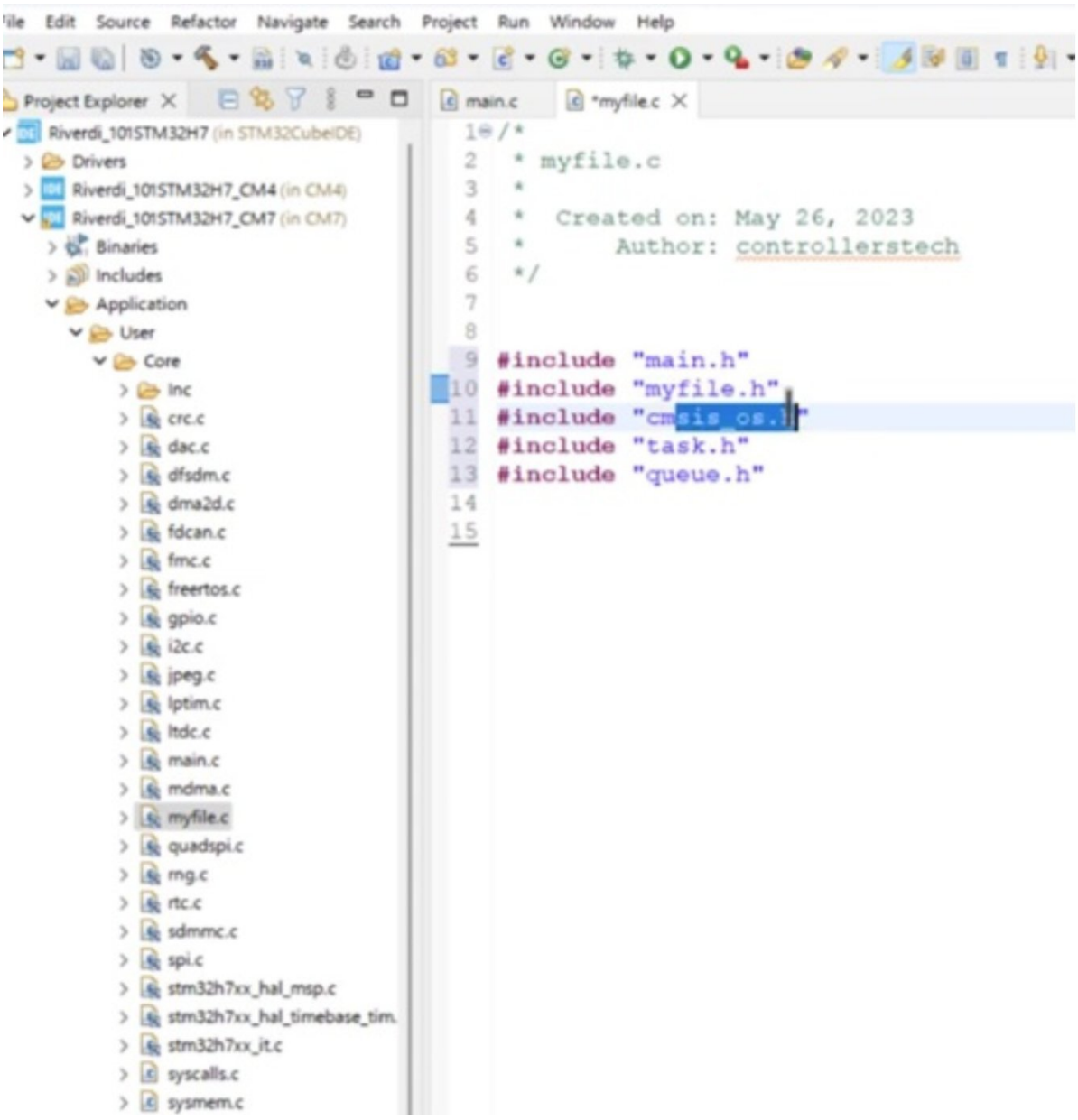
Now let's add the buttons to turn the LED on and off. I am using the flex button so that I can resize it as per my need. I will modify this button to look a little more attractive. So under the visual element, click Add and select icon. And now in the icon option, we will add some button icon to our button. I am using this clay theme as it has some good circular icons. The image size should be less than the button size. Choose the images for the button released and button pressed. Now remove the button with border element so that the background is removed. Let me adjust the button size so that the image fits a little more perfectly. All right, let's name this button as the on button we need to create one more button the off button. So we will just copy this button and change some things now the buttons have been created. So we will also define the interactions. When the on button is clicked, it will call a new virtual function the On Clicked. Similarly, when the off button is clicked, the off clicked function will be called. That is it for the designing. Let's generate the project now. Open the project in the cube ID. Here is the project structure. Let's take a look at the G P i O dot c file. Here some pins are already configured for certain purposes. So when you choose a certain pin to be an input or output pin, make sure not to choose from these. I need one pin to be configured as output where I can

connect the LED to let's quickly see the datasheet to see the availability of the pins.



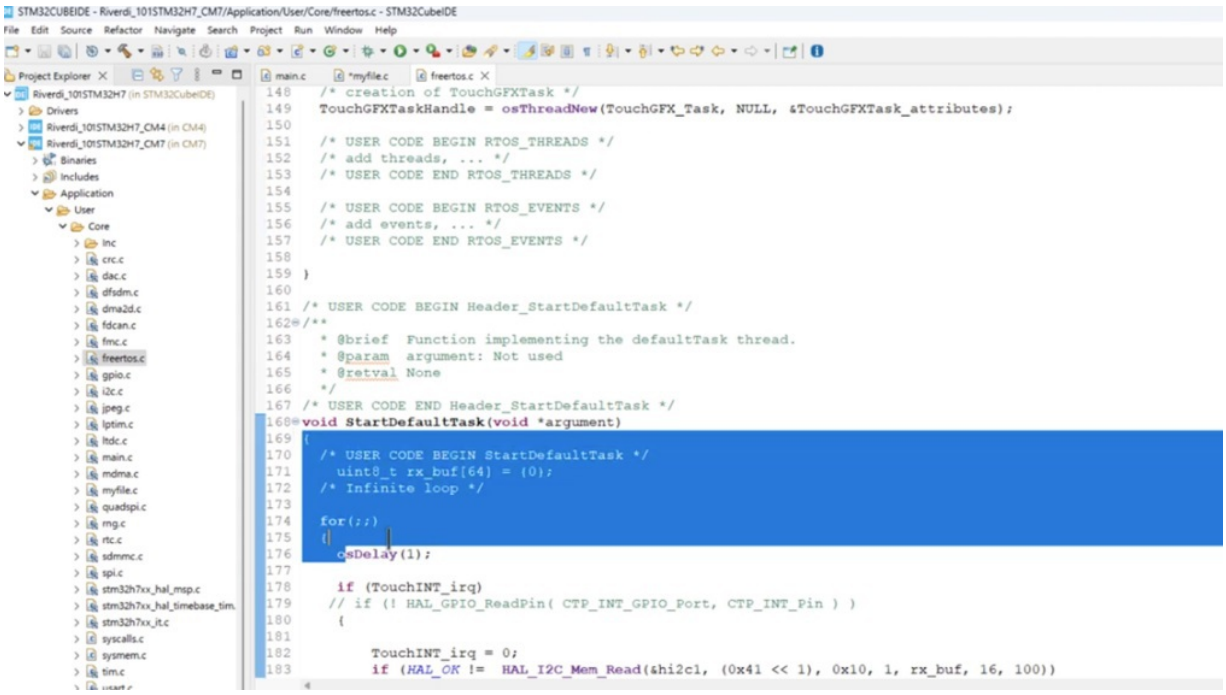
I have this particular display from reverdy. Here is the link to download the datasheet Alright, as you already know, we have the expansion connector on the display which can be used to connect GPIO i to see SPI et cetera. I will use one of the pins from this connector to be used as the output pin. I am going to use this pa five but this is already He being used as the DA C pen. So we need to first disable the initialization of this pin as the DA C pin and then initialize it as the output pin. Gia in the DA C source file, you can see the pin PA five has been used as the DA C output pin. So go to the main file and comment out the DA C initialization function. Now, we will initialize the pin in the output mode. If you know the initialization code, write it or you can refer to the GP I O dot c file. Here this pin is being initialized in the output push pull mode. So I will copy this part. Let's write a function led in it. We also need the definition of initialization structure. Also reset the pin before initializing it. Now change the GP i o port to GPIO A and the pin to pin five. Make sure the clock for the respective port is enabled. If not, then enable yourself. Now in the main function, call the LED init function to initialize the LED pin as output. Let's build the code once. We have

few warnings, but that's alright. As I mentioned in the beginning, I will use a harder but more common method to control this LED, I am going to create a new task to blink the LED and cue to transfer the data from greed to the MCU. A much better way of approaching this is to create a new source file to write our code. This way we will have our code separated from the predefined one. And whenever needed, we will only make changes in that particular file. So I am creating a new source file, my file dot c and a new header file my file dot h now we need to move these files to the main project folder where the rest of the files are stored. Open the newly created file in the system explorer. Now we will move these files to a different location. Go to the main project folder cm seven core source, move the dot c file here and dot h file in the include folder. Now both the files have been removed from the ID, so we will link the source file to the core directory. Make sure you link relative to the project location. Now we have the file here and we can write it we will start with certain inclusions. So here I have included the main header file, my file header, C M sis o s for the RTO s task header file for the task related functions and Q header file for the queue related functions. Next, we will create a task and a queue, we need to first create the attributes for the task and the queue. If you don't know the functions for the same, you can refer to the free RT o s dot c file. Let's copy the attributes of the default task. Now we will change the word default with the LED.



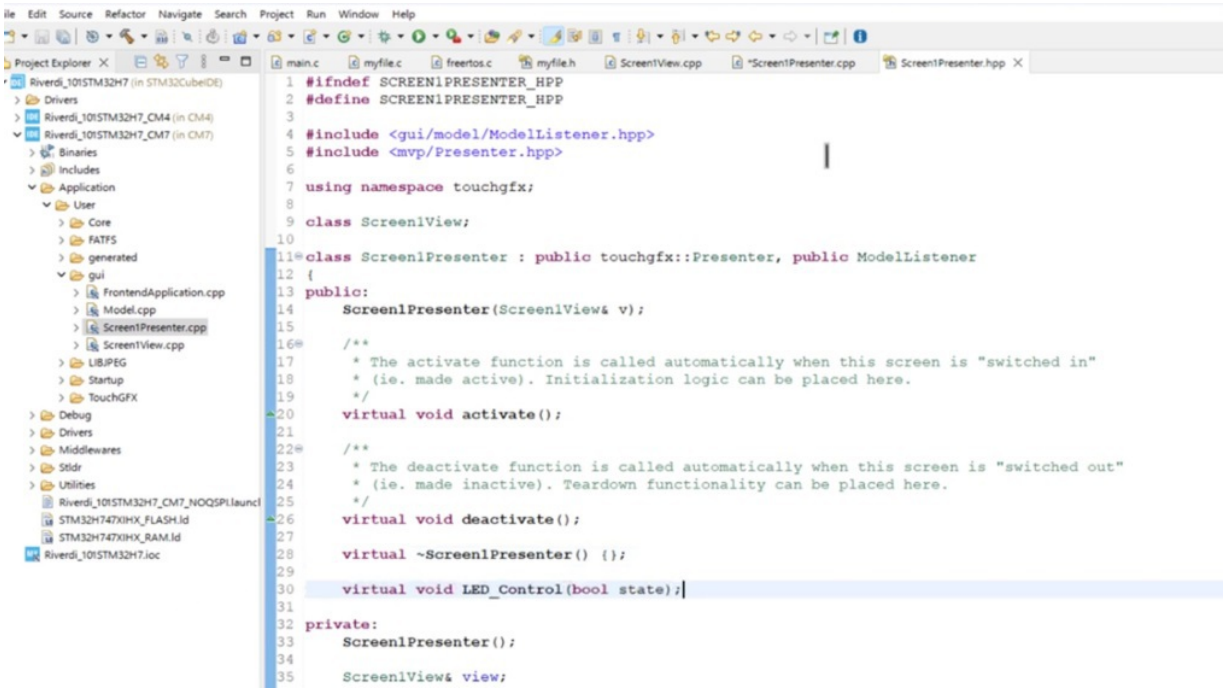
So we have the LED task handle and the LED task attributes now creates the attributes for the Q. Let's define the LED Q handle the Q attributes just require the name parameter Next, we need to create the task and the queue. Let's write a separate function to create the LED task. The Start led task function will be defined later. All right, let's create the queue now. The size of the queue should be one element and the element size is the size of the integer. Basically, this is going to be an integer Q with one element. This is because we only need to transfer with a one or a zero. Now we will write the start led task. Again I am taking the reference from the start default task. I

have covered this part in detail in the previous touch GFX projects. We will first check if there is some message in the queue If there is we will copy the message into the variable led state let's define the integral variable led state now the LED state will be set as per the value of the LED state variable, this task will run every 10 milliseconds, but the LED will only set when there is some data in the queue. That is when the button is pressed, define the start led task at the beginning. So, we create the task attributes and the queue attributes. Then we create the task and the queue and in the end we write the task function. We need to define these initialization functions in the my file header file now go to the main file and include the my file header and initialize the task and the queue after the free R T O 's initialization function now we will write the code for the GUI, open the GUI folder and we will start with the screen one view source file. We have defined the interactions when the buttons are clicked.

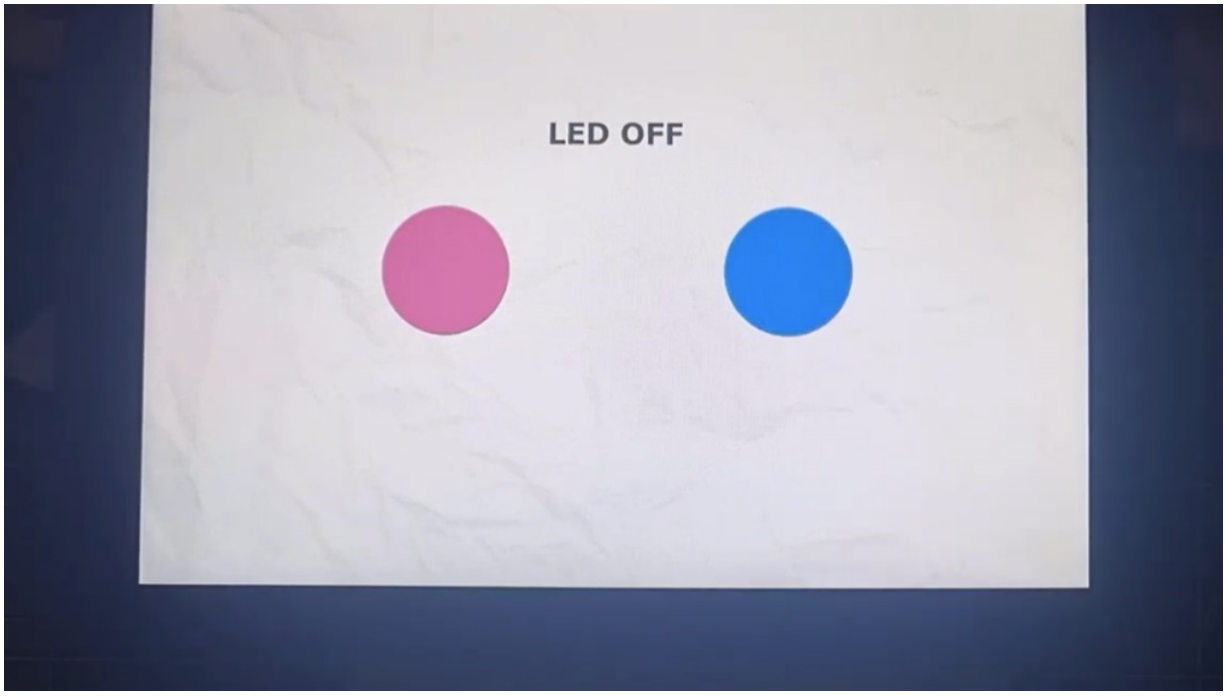


So we first need to define these functions in the view header file. Now when the on button is pressed, the on click function will be called. And here we will call the LED control function in the presenter and pass the parameter true. We also need to display the LED status in the text area. So first use the string copy function to copy

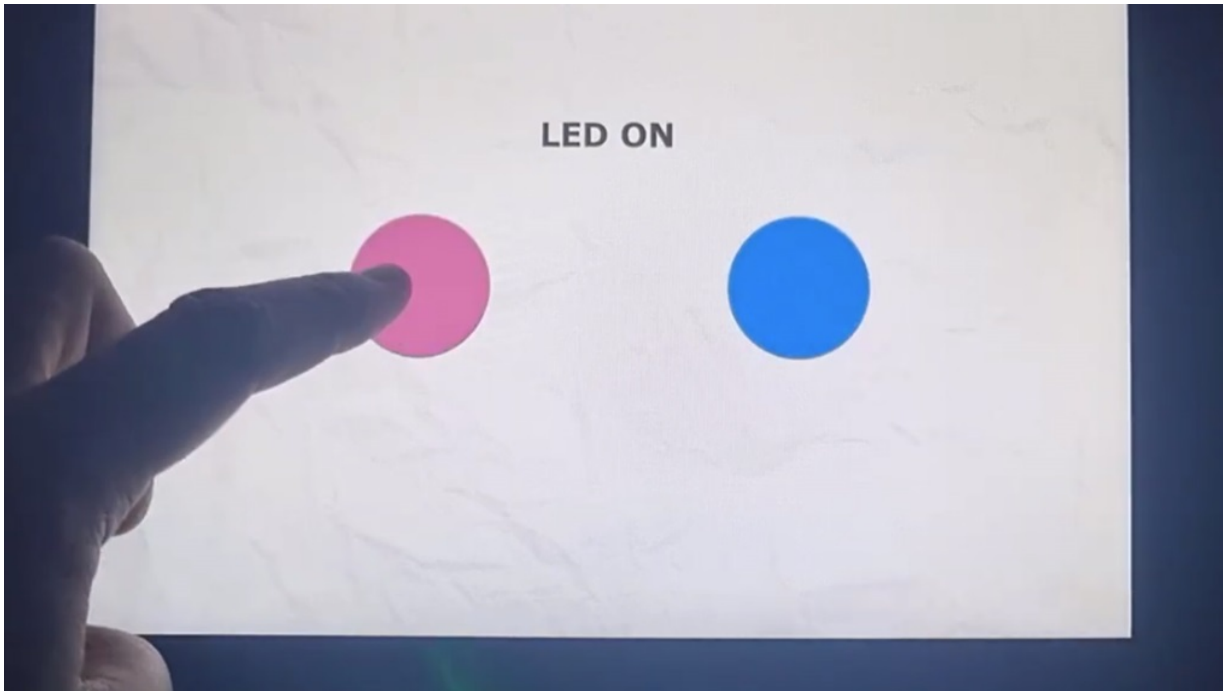
the LED on string to the text area buffer and then invalidate the text area. Similarly, when the off clicked function is called we will pass the false parameter to the LED control function and update the text area with the string led off. Now we need to define the LED control function in the presenter.



Here we will call the same function in the model and also pass the state as the parameter. We also need to define this function in the presenter header file finally we will write the main code in the model file first include the CM sis OS header file, so that we can write the Q functions include the my file header also. Now define the LED queue handle as the external variable in the LED control function, first check if there is some space available in the queue. If it is then put the message in the queue. The message is the state variable that has been passed from the view file. Let's build the code now. We have to define this in the model header file also. All right, the code builds fine but it will show an error when you flush it via the touch GFX so let's fix that to open the project folder and go to the root folder. Here go inside the GCC folder and edit the make file for cm seven. Scroll down till you see the board C files section we need to add one more file for the compilation. So add the my file dot c here. Save the make file and open the touch GFX again.



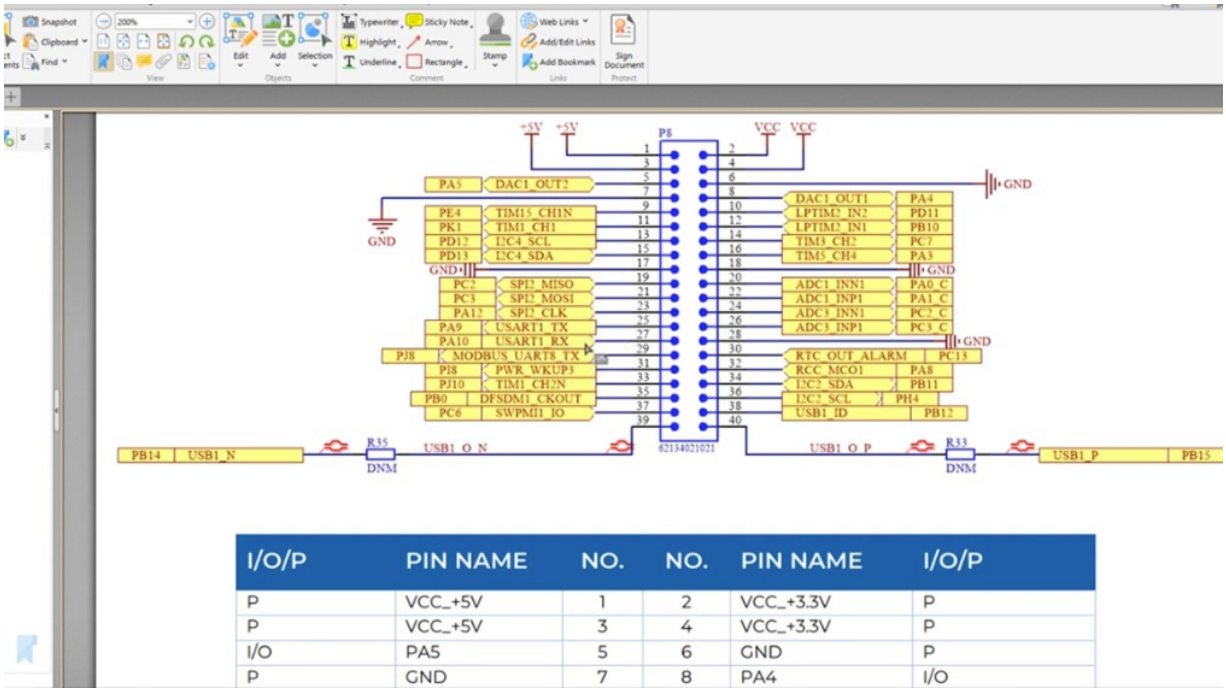
Let's see the simulator first the project is running fine on the touch GFX so let's mash it to the board target has been flashed successfully. Here you can see in logs my file dot c was also compiled and therefore there was no error. Let's see the project working on the board you can see the LED is responding well and the text area is showing the status so everything is working well. I will continue with the activity display and make more projects covering the UART and some sensor.



We will continue with this file we created today and add more functions to it.

RIVERDI STM32 DISPLAY HOW TO SEND DATA FROM UART TO UI

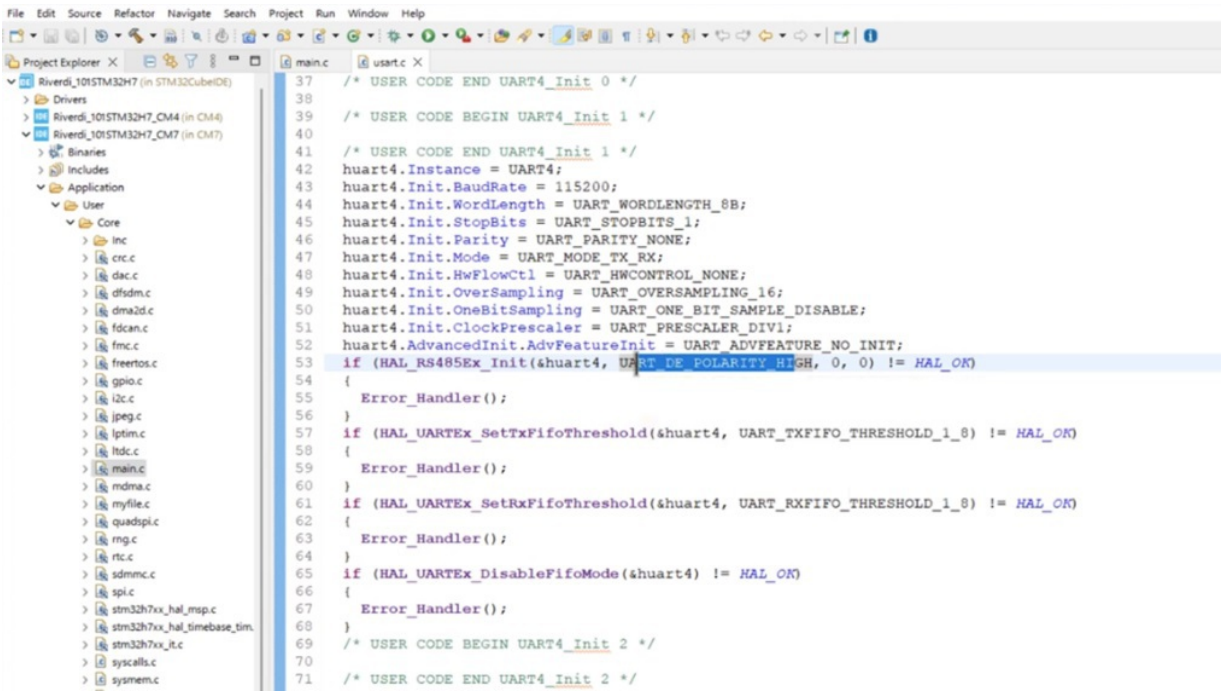
And today we will send the data from you on to the UI, we will simply receive something from the UART and then display it on the GUI. I will continue where I left the previous project. So the ad functionality will still be present in the tutorial. Here is the previous project we worked on. Let's add another text area to it. I am using the same large typography and the name should be text area you aren't the wildcard range is already set for it. So this is it for the configuration. Let's generate the project and open it in the cube ID.



I/O/P	PIN NAME	NO.	NO.	PIN NAME	I/O/P
P	VCC_+5V	1	2	VCC_+3.3V	P
P	VCC_+5V	3	4	VCC_+3.3V	P
I/O	PA5	5	6	GND	P
P	GND	7	8	PA4	I/O

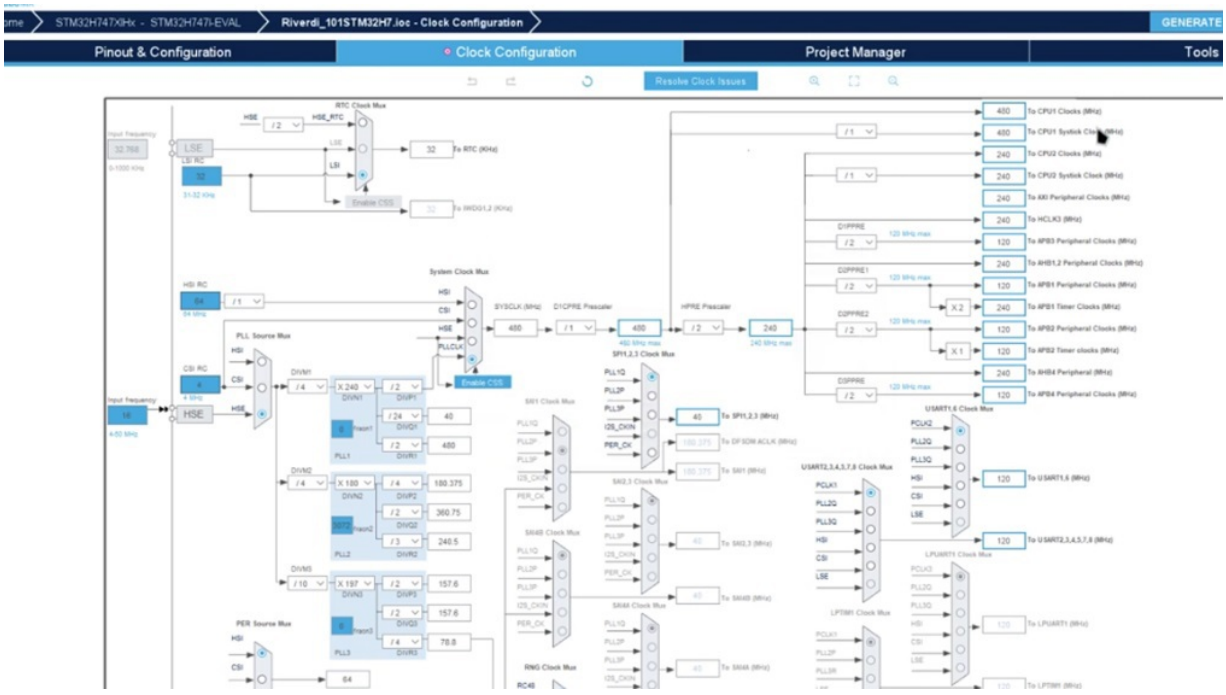
In the meantime, let's see the datasheet again. Here you can see the board has Rs 485 interface and the RS 232 interface. Along with

these the expansion connector also has the Yousaf one pins and one UART a TX pin for the Modbus for convenience, you can use the RS 232 but since I don't have a converter for this, I am going to use the RS 485 All of them work on the UART so using either is fine, you can check out my previous project explaining the RS 485 communication. Here we need only two pins pin A and pin V i am using a USB to Rs 485 converter module with the computer the pin A from the module connects to the pin A of the board and the pin B connects with each other. The board already has Rs 485 to UART converter, so we don't need to worry about the signal level. Let's see the project now. This is the same project we made for LED control and I will continue with the same here in the main file you can see the you SOT header file is already included. In the initialization section, the UART one is being initialized along with you art four and you aren't eight. Let's see the code for it this section initializes the UART four and if you note here there is a function to initialize the RS 485. This means that the UART four is connected to Rs 485 and we will work with it if you know about the RS 485 module there is a driver enable pin this pin can be used to set the module in the receive mode or the transmit mode here that the pins polarity is set to high which means that the pin is active high. So to pull the signal high we pull the pin high and to pull the signal low we pull the pin low the UART eight has no initialization here, the UART one is initialized normally in the initialization of you are to the hardware flow control is enabled. So this must be connected to the RS 232 port.

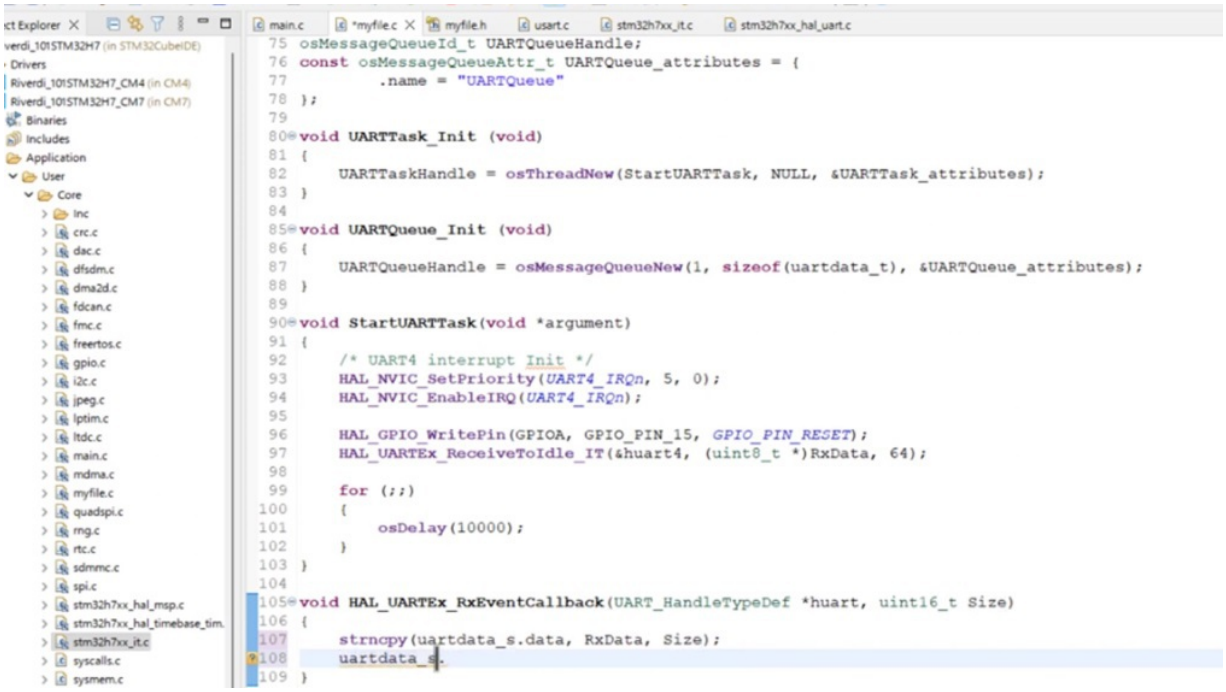


As I mentioned, I am going to use the RS 485 port. So I will use the UART for irrespective of whatever you are to use, the code will remain the same. All you need to do is change the UART instance. Let's quickly see the MSP initialization for the UART for here you can see the D pin is actually the pin p of 15. Keep this in mind as we would need it to put the port in the receive mode. The UART for initialization is already called so we are good here. We don't need anything in the main file. Let's head over to my file dot c, which we created in the last project. Here we already have the LED related code from the previous project. So I will write the UART code separately. Most of the things will remain the same. So let's copy this complete code and paste in the UART section. And now we will change the word led to the UART. All right, let's understand things. First we define the task handle and the Task Entry function. Then we have the attributes for the UART task. The stack size Is 512 words and priority is set to normal. Next define the attributes for the UART queue, which will be used to send the data to the UI. Then we write the functions to initialize the UART task, and you are to queue. And finally, we have the task function, which we will write from scratch. Since we will be sending the data in the string format, we need to create a structured queue. I have covered this in my touched GFX

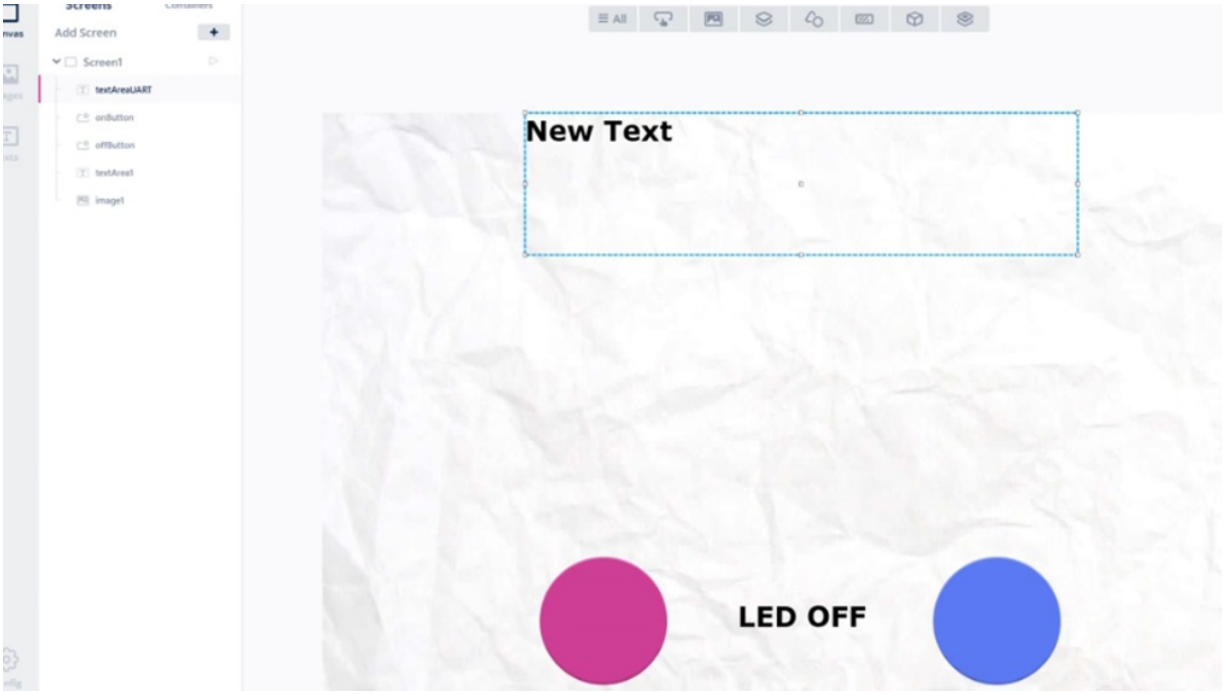
project, you can see on the top right corner here, the structure consists of two elements, a character array to store the data and an integral variable to store the length of the data. Let's define the structure you asked data S, which will be used to store the information, the queue will only have one element and the size of this element will be the size of the structure. We also need to define the UART handle. So copy it from the UART dot c file and define it as an external variable. Now I could simply use the basic UART functions to receive the data here. But let's say that I want to use the idle line interrupt to do so. Let's also assume that I don't know how to initialize the interrupt for the UART. So far, we have used the cube MX to initialize the interrupts for us, but we know that we can't use cube MX with the reverdy display. So here is what we will do. Copy the project to another location. Now open the IOC file and cube MX. Let's see the UART four. Here you can see the pins being used for the UART four. I want to initialize the interrupt for the UART four. So let's enable it. Don't make any unnecessary changes in the file generates the code now, don't open the project. Instead open the folder. Now go to the CM seven core source and open the UART dot c file. Here you can see the interrupt has been initialized.



So let's copy this, you can put it in the UART source file itself. But since it's initialized after the UART I am just putting it here in the task function. Another addition that would have happened is in the interrupt file, let's copy the UART handle and paste it in the interrupt file. The UART for interrupt handler is also created so we will copy it in our interrupt file that's all the changes happened when we added the interrupt. Let's write the rest of the function now. First, we will set the D pin low so that the RS 485 can be put in the receiver mode. Now we will receive data in the receive to idle interrupt mode, the data will be stored in the RX buffer and the maximum we can receive 64 bytes at once. Let's define this RX buffer. Since we don't need anything else from this task, let's give a bigger delay here. Now when the UART has received 64 bytes, or it detects an idle line before that an interrupt will trigger. This will call the RX event callback function. Here we will first copy the data from the RX buffer into our structure. I am setting the last data byte as the string Terminator and updating the length parameter with the new size. Now we will send the structure to the queue. So first we will check if there is some space in the queue. If there is then put the message the priority and timeout both are set to zero here and finally call the receive function again.



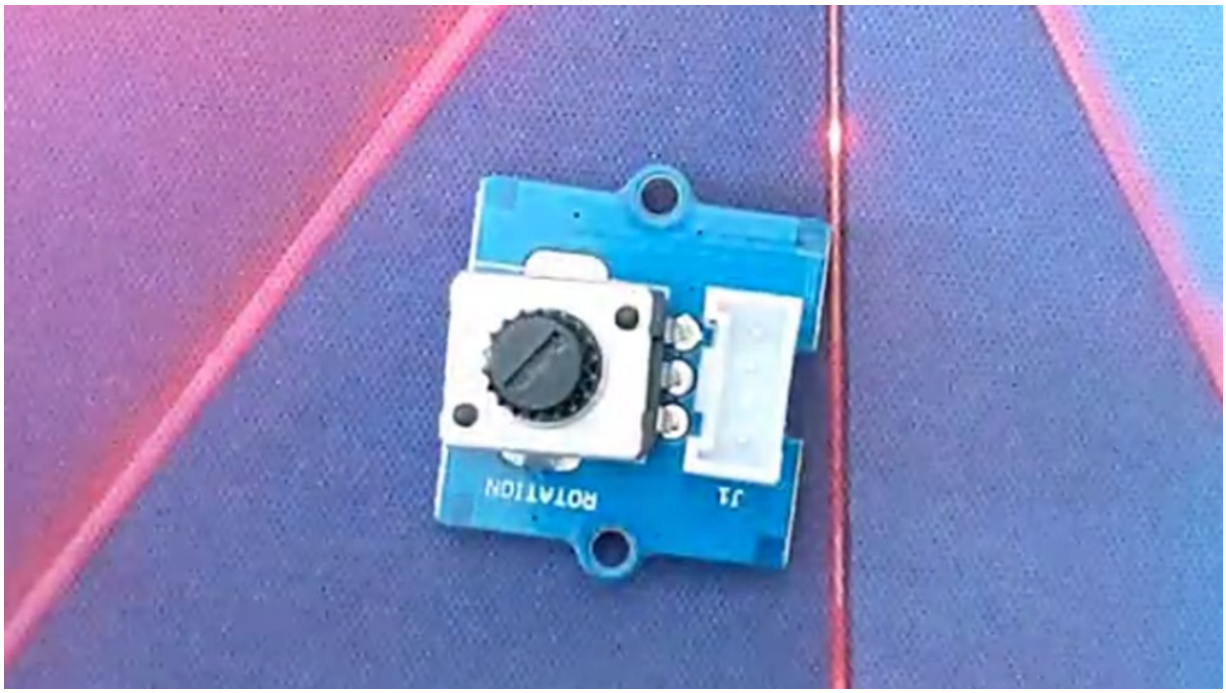
So that is all we need to do for sending the data. Let's build the code now. The string header file needs to be included here. Now define these newly created functions in the my file header so that we can call them in the main file all right now we need to receive this data in the model and finally display it on the UI So let's start with the model. First, let's define the UART cue handlers the external variable and also define a new structure to save the received data. The model tick function is called every time the frame is refreshed on the UI. So we will write our code inside it, we will first check if there is some data available in the queue. If there is then we will get the data and store it in the structure we defined. Now copy the data from the structure and store it in an array we need to define the character array in the model header file. Once the data has been stored Paul the function you are to display in the model listener, we need to define this function in the model listener. Make sure the function has an empty implementation. Since the function has an empty implementation, it will look for the same function in the presenter now. So we will define the same function in the presenter header file and write the code for it in the source file. Here we will call the same function in the view. So now define it in the view header file. Finally, we will write the code in the view source file. Here we will use the Unicode string copy function to copy the data into the text area you are to buffer. I guess I forgot to implement the wildcard for the text area. Let me do this quickly. The buffer should have one higher byte than the maximum data you are displaying on it regenerates the code. Let's refresh the project once it is still not showing in the autocomplete. So let me copy it from the screen view base file. After copying the data into the buffer, we will invalidate the text area so the changes can take effect. The string copy inclusion is missing.



So let's include the C string or write the code builds fine. Go to touch GFX and flash the code to the board I have connected the RS 485 module to the computer and it is connected to comport seven you can see the Ewoks data is being displayed on the text area. Here the data has exceeded the space available for the text area. This is because I forgot to include the word wrapping the code here in the view file, set the wide text action to Word Wrap. This will fix that issue and the text will automatically display on the next line the LED is working fine. And now we have the UART also. So this completes today's project about sending the data from UART to the UI. I hope you understood the procedure.

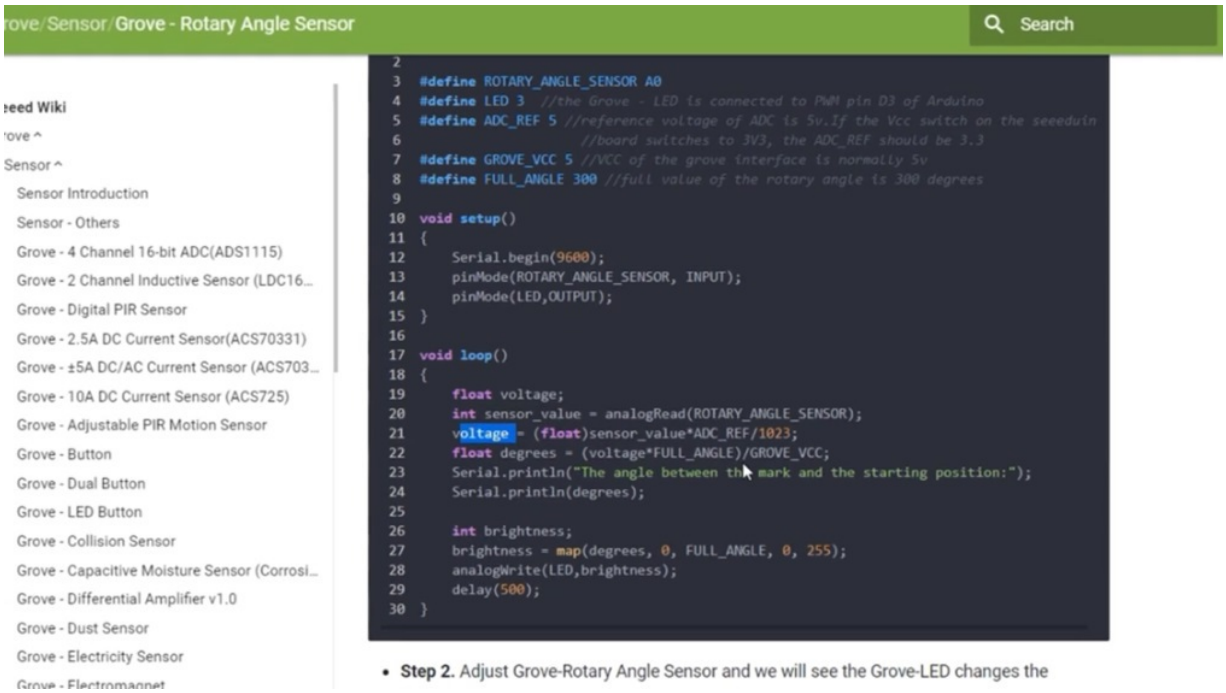
ROTARY ANGLE SENSOR AND STM32 ADC

We will cover how to interface rotary angle sensor with STM 32. This here is the rotary angle sensor and it works with analog to digital converter it works like a potentiometer so the resistance changes with rotation and we get a variable voltage at the output data for pins round VCC and signal the third pin is not connected we will use the reading from it and change them to the angle by which the shaft has been rotated.

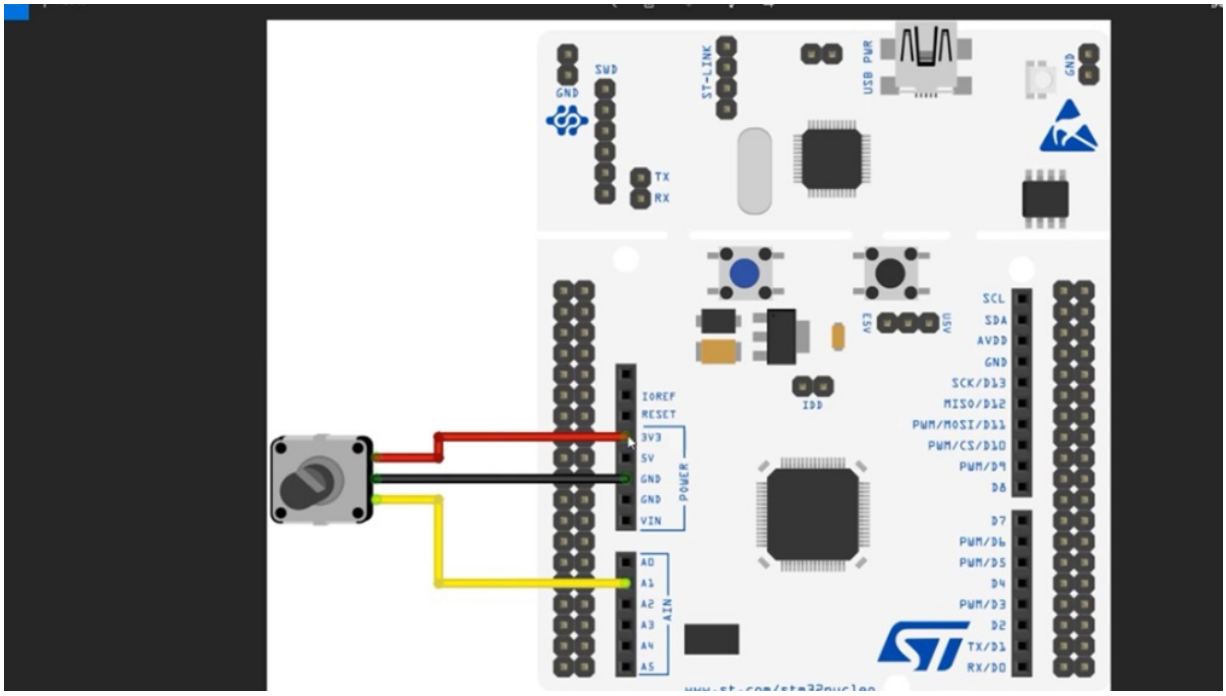


Let's start by creating the project with cube ID. By the time the ID loads let's see the details about this rotary angle sensor. As we can see here, the sensor can rotate between zero to 300 degrees here

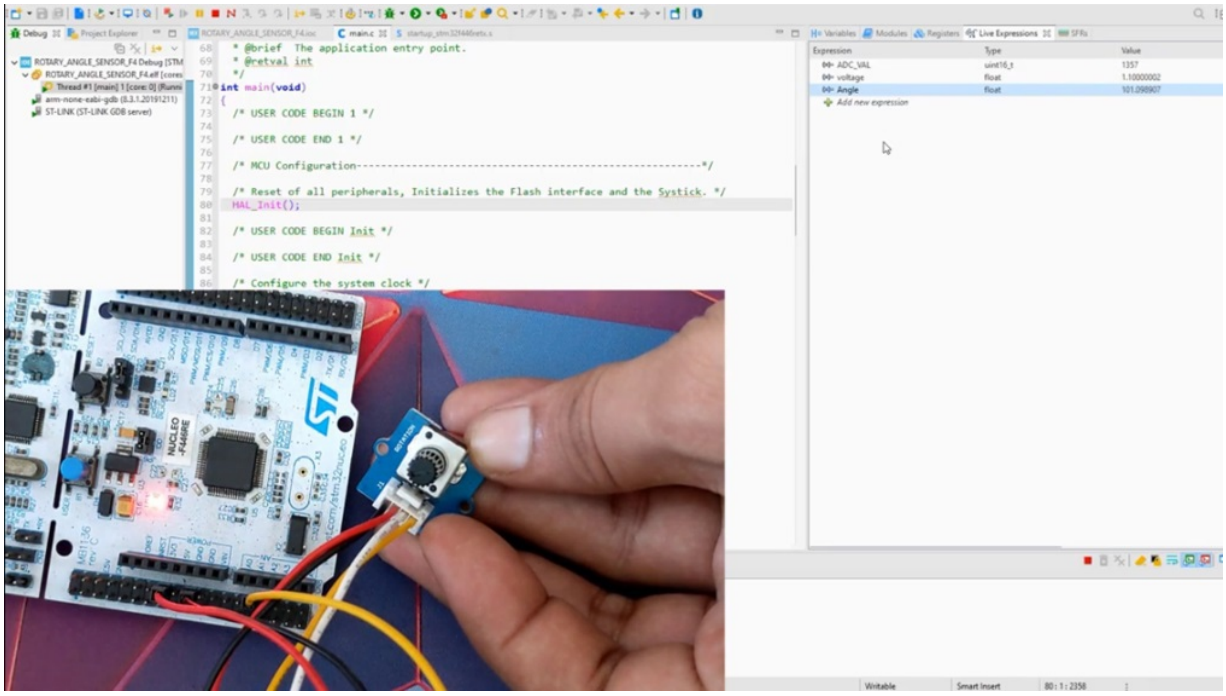
they have a sample code. It is an Arduino code but it's simple to understand. Let's go through it. Here first of all we will read the ADC value next convert this value to the voltage and using this voltage find the angle that's all as I said it's pretty simple to use now let's create the project in cube Id choose your device here give some name to this project and click finish first of all I am selecting external crystal for the clock as this sensor works with ADC, I am enabling channel one we will keep using 12 bits for the ADC here we will enable the continuous conversion.



That's all for the ADC setup. Now let's go to the clock I have eight megahertz external Crystal and I want the system to run at 180 megahertz click Save to generate the project let's see the connection once. VCC is connected to 3.3 volts ground to ground and the signal pin is connected to the pin P A one the channel one of ADC Let's start our program now. First of all we will define a variable that can store the ADC value. Next define the voltage and the angle variables. To read the data from ADC we will use the poll for conversion method.



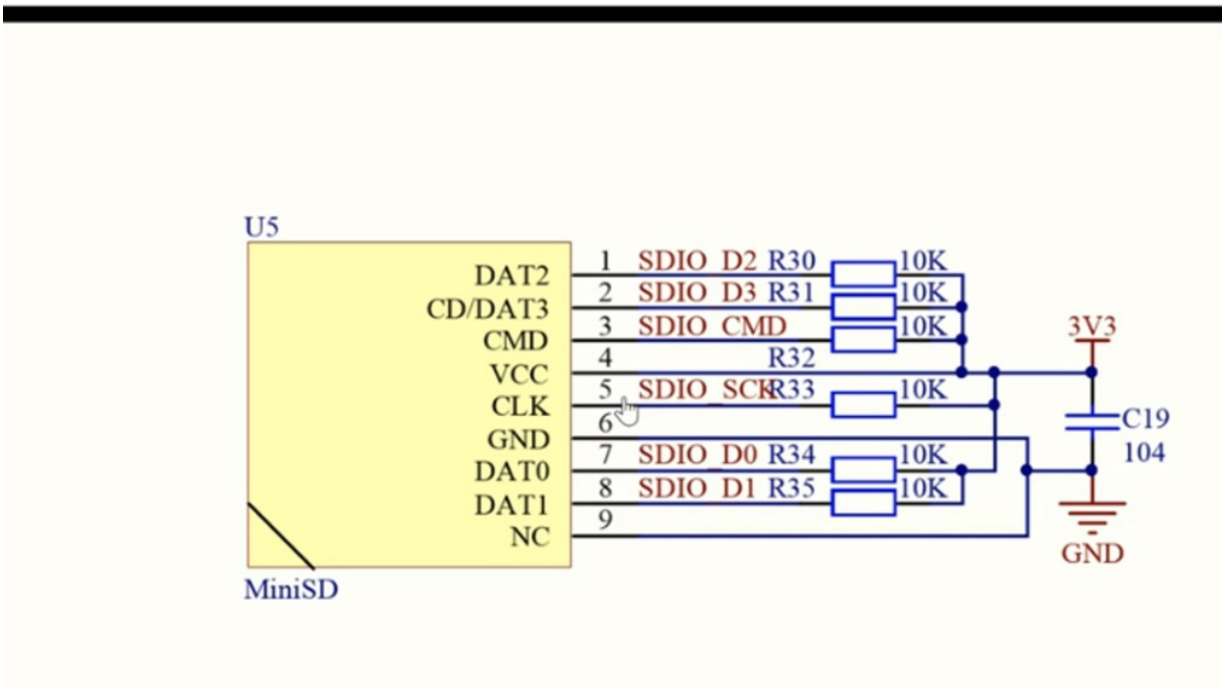
So first start the ADC poll for the conversion to finish. Read the value and store in the variable we created and finally stopped the ADC. Now we will convert this ADC value to the respective voltage. Here 4095 is used because I am using 12 bit resolution in ADC And 3.3 volts is the reference voltage in STM 32 F 446. After getting the respective voltage we will convert this to the angle here 300 is the maximum angle and 3.3 volts is the supply to the sensor. Let's give a delay of 10 milliseconds in this continuous loop, change the type here to float. Let's build this code now.



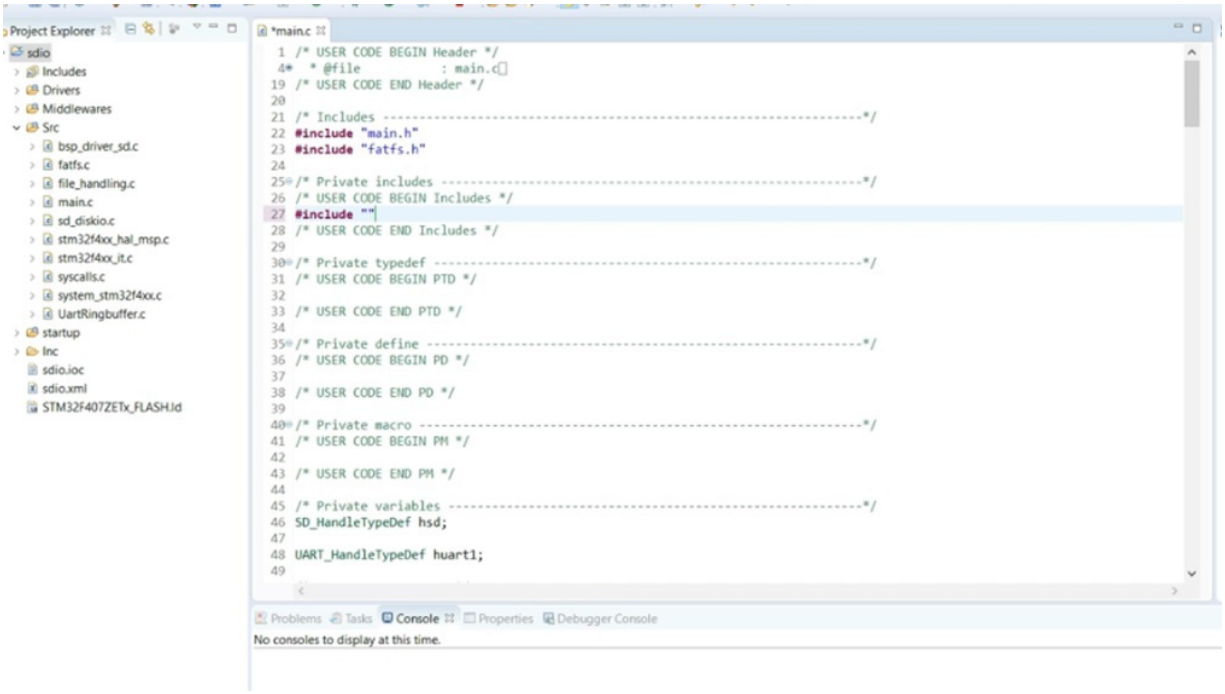
We don't have any errors. So let's debug it I have added all the three variables to the live expression. Let's run it as you can see the values of zero right now as I rotate the potentiometer the angle starts increasing the maximum angle is 300 degrees now it stopped reducing and we are back to zero this working is very simple as the sensor relies on ADC This is it for this project. I will try to make another project where this sensor can be used to control the position of the stepper motor.

SD CARD USING SDIO IN STM32 UART RING BUFFER 4-BIT MODE CUBEMX

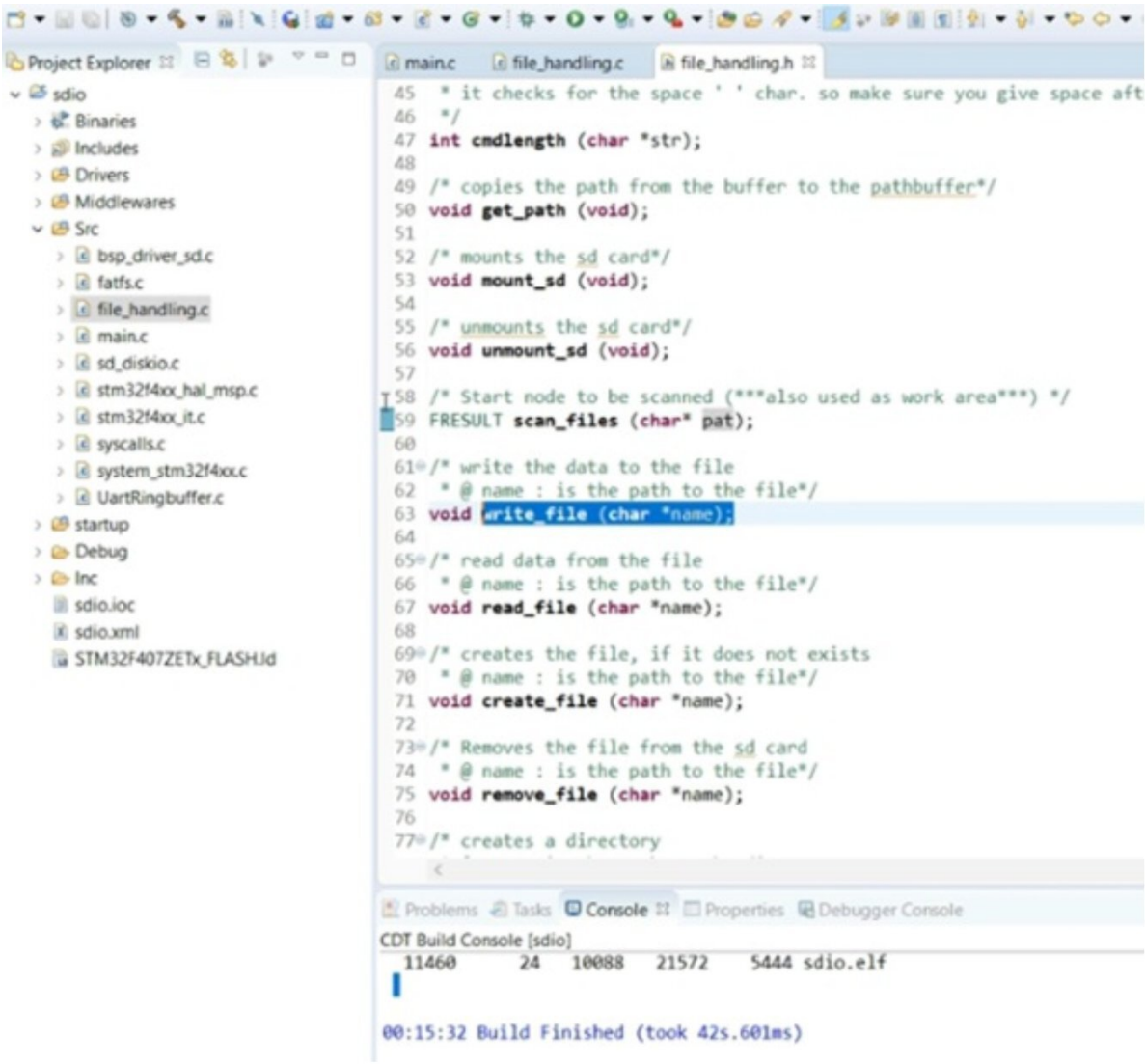
We will interface the SD card using SD i o four bit mode. Please keep in mind that all STM 32 series does not support SD I O, especially those who were requesting it for STM 32 F 103 controller, we can't use SD IO in F 103 controller. I am using STM 32 F 407 controller and it has onboard SD card reader with SD IO support. So I will start with cube MX first select the SD IO four bit mode and leave everything as it is I will be using UART to control the entire process. So select the UART interrupt in the fat Fs select the SD card we are going to keep everything same except increase the size here as you can see these are the pins for the SD i o four bit mode and these are how they are connected to the onboard SD card reader you can pause the project and take a better look let's set up the clock now.



I will keep the clock of maximum possible frequency here this finishes up the setup let's generate the code now. First of all copy the library files in the project folder. I will be using some functions from you authoring buffer. I already made a project about it you can watch it if you don't understand something I will leave the link below. Next we need to include these functions in our project let's build it once to check for any errors Okay, so everything is good till now. Let's proceed To open the file handling dot h and here you can modify the buffer size and path length. These are the commands that you can use for different operations on the SD card. These include ls to list all files and directories. MK dir to create a directory, MK Phil to create file, read and write for the data operation on the file. RM removes the file or directory update is for updating the contents of a file check file gives the information about the file and check st gives the memory details of SD card. If you don't want to use with the UART you can use the functions directly. They are as follows.

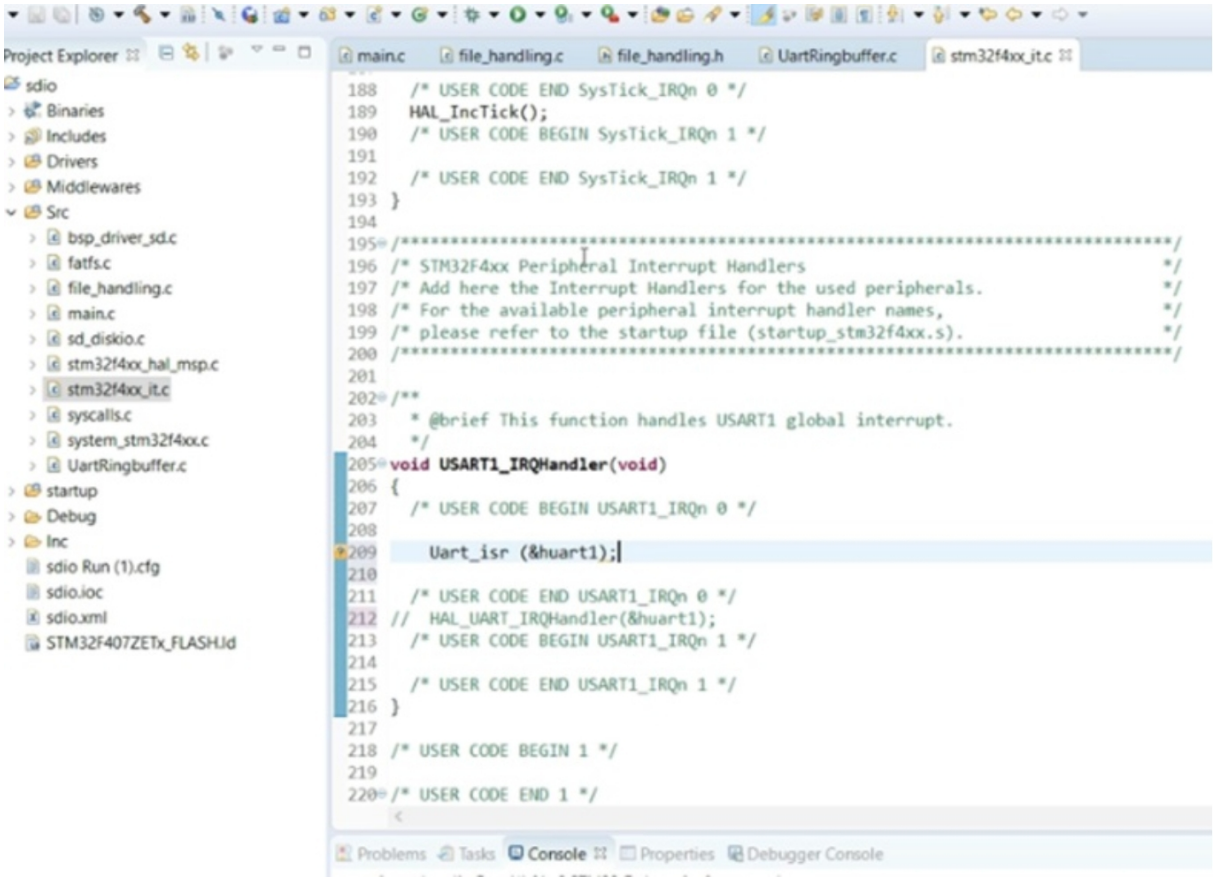


Send you art sends the string to the UART you won't need CMD length and get path functions than mountain Dunmer unmount the SD card scan file gives all the directories and files from the input path. Write, read, create and remove works as their name suggests. In this tutorial, I will use the UART to control the file handling in SD card. First of all, we need to initialize the ring buffer Next, mount the SD card using the function mount SD. Now the rest of the code will be written inside the while loop. Till the data is available to be read in the RX buffer we will get the string sent by the user data from the RX buffer will be written to this input buffer only when the user press enter or send a terminating character at the end of the string. Next, we are going to measure the length of the command within the string. This works on the fact that there will be a space after the command next we are going to get the path this path will be copied in the path buffer. Next, we are going to compare the input commands to the saved commands. I forgot to define this CMD variable based on the comparison we are going to assign different characters to this CMD variable.



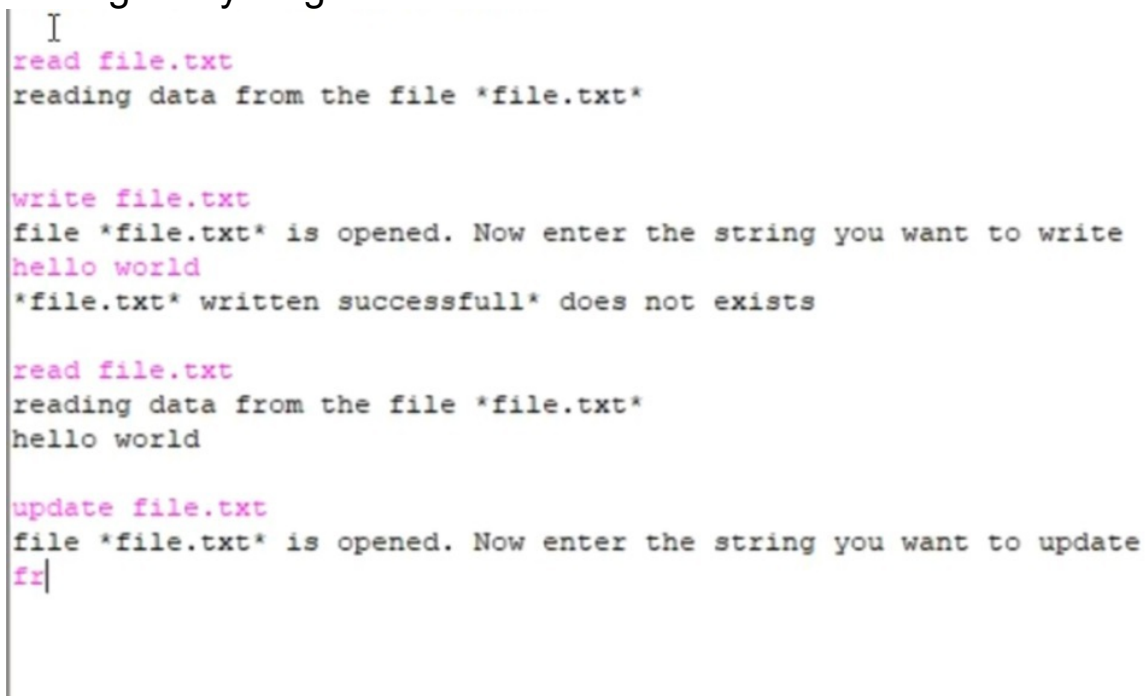
And finally use the switch and case function to perform different operations according to the command. For example, if command is Ls than CMD variable will be assigned character L. And inside the case function we can scan files for the entered path. Just like this we can write the appropriate functions for the other commands also M for creating directory C for creating file are for reading a file and other characters as you wish. One last thing we need to do is we need to change the default interrupt routine to the one we have in the library file. So open the URL to ring buffer dot c file and copy the ISR definition and paste it in the STM 32 F four xx it dot c file then

locate the UART IRQ handler and comment the default function in it and add the function that we just defined above.



Now let's spill this code and test it I am getting an error about resetting the target go to debug configuration or run configuration under the debugger tab, select Show generate a script and change it to software system reset so the flashing part is success, I am using Hercules for the serial data transmission in the computer on resetting the controller I got the following message on the terminal let's try by listing all the directories and files present in the SD card at this moment as you can see there is a directory and there is a file inside this directory let's first try to remove the directory and see what happens. So there is an error this is because the directories can only be removed if they are empty so I will remove the file first this is successful now let's try to remove the directory so now the directory is removed and there are no files present in the SD card right now let's create a directory in the root you can see this now let's create a file in the root itself I made one more file inside the

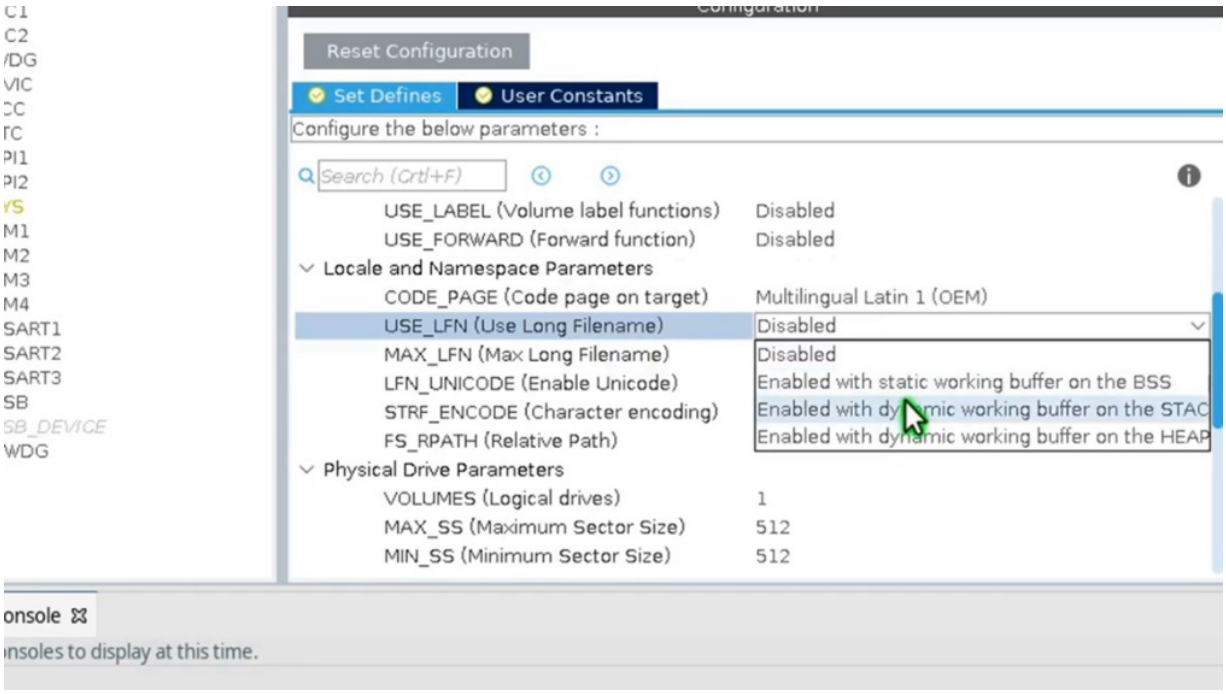
directory as you can see here if we read the file at this moment we won't get anything so let's first write the data into this file.



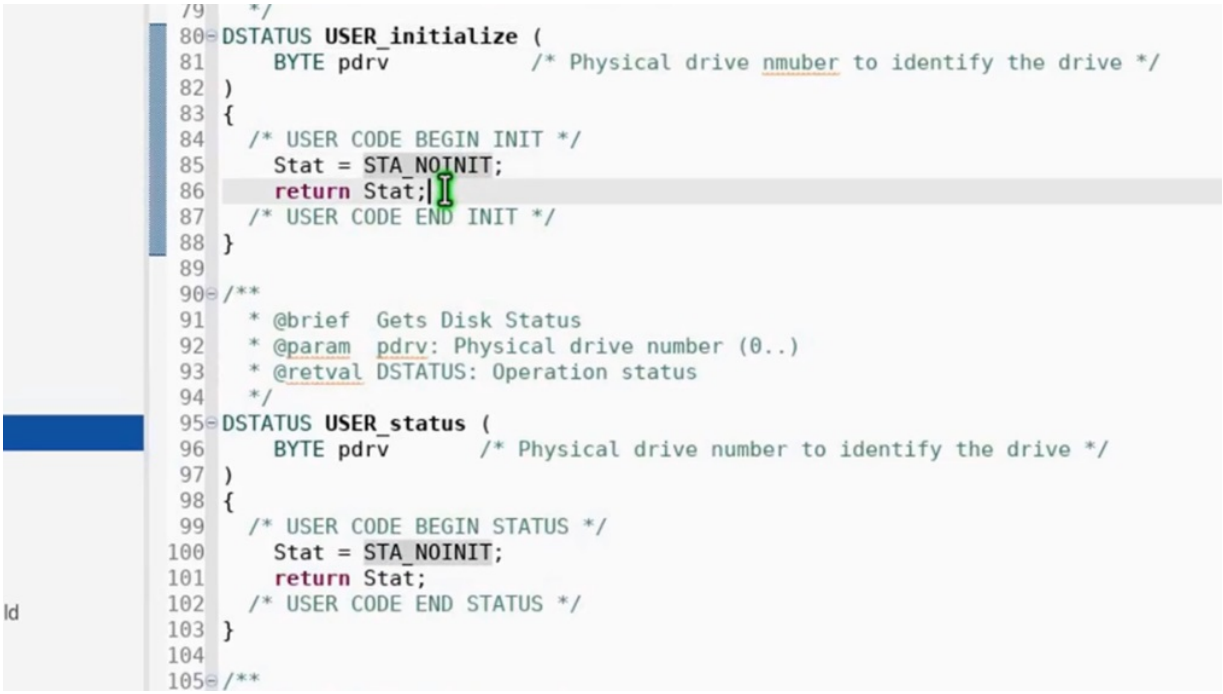
I will write hello world into the file and got the confirmation to now let's read the same file and yes, we got the string that we wrote into the file we can also update the file but this data will be written to the end of the file and I'm reading by got the updated file check file gives the details about the file if we check the another file that we created, you can see its details. Check SD gives the memory details of SD card.

SD CARD USING SPI IN STM32 CUBE-IDE FILE HANDLING UART

We will do some basic file handling this includes creating opening Read Write, update, and deleting the files, other complex functions such as directory related and file system related functions will be covered in the upcoming projects I am using STM 32 F 103 controller and STM 32 Cube IDE So, let's start by creating the project in cube MX. Select the fat EFS file system and pay attention to the settings here I will be using SPI one in full duplex Master Mode also I will use you off to transmit data and rest is the usual settings.

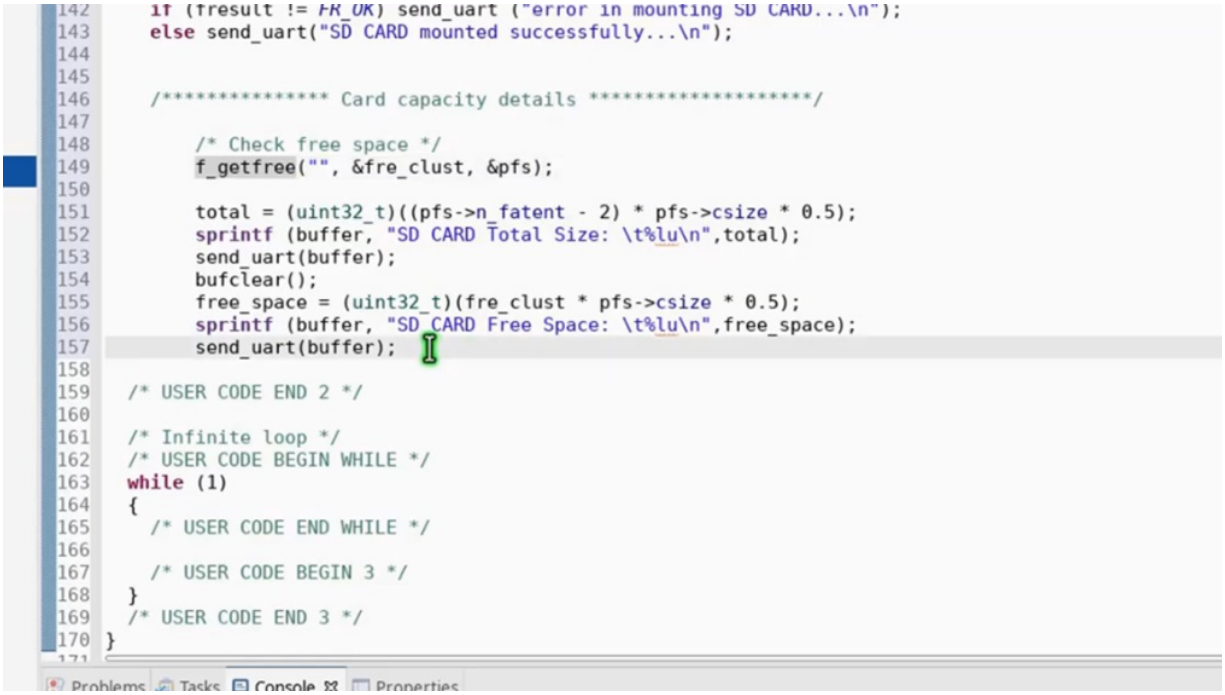


So the project is created I will just quickly build it once so the errors can be avoided copy of the library files that we are going to need to the respective folders we need to include this file that we just copied also to use some string functions next opens That fssd dot h file and use a disk i o h file here we need to link the file system to our driver just follow me note that the arguments here should be same as they are in the function to which this is a part of let's build it again to check for errors.



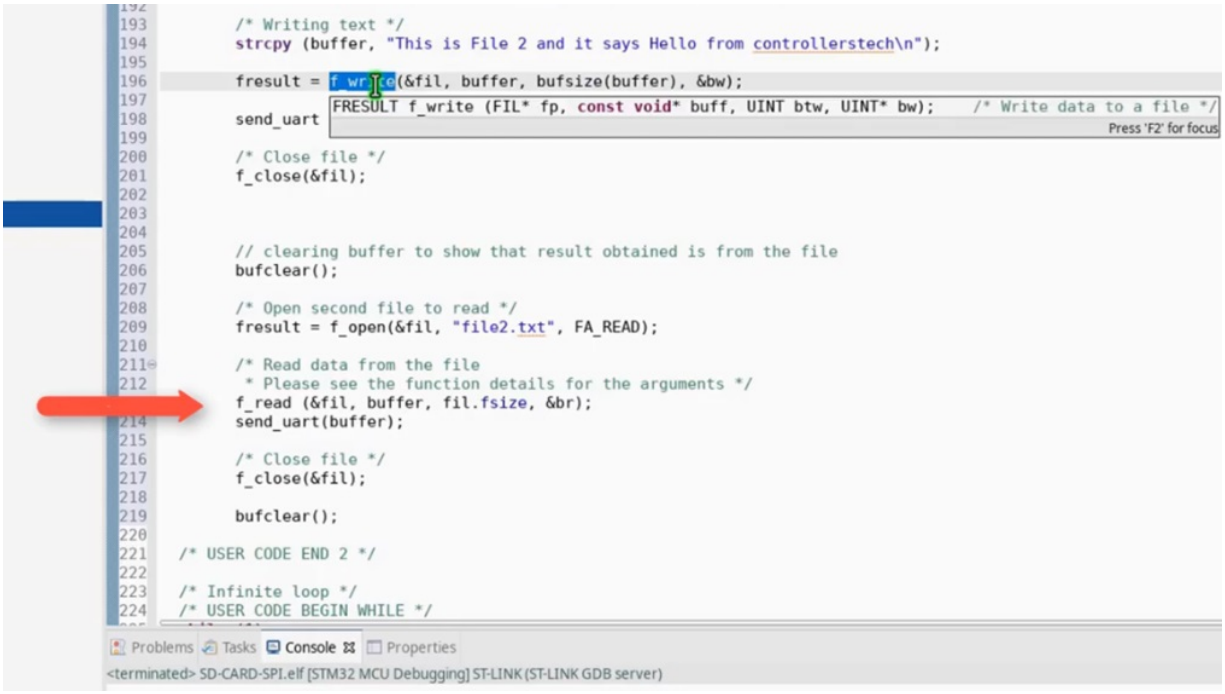
And yes, we do have some errors, but don't worry, we will get rid of them in our next step in the fat fssd dot c file make sure you define the CS port and CS pin that you are using. Also the SPI handler the default here is b one let's close these as they are done and open the interrupt handler. Here we need to define our timer functions these timers should run independently and that's why it's best if we define them in the SysTick handler. Note here all arrows are gone. Also this is it for the configuration. Let's move to our main function now. I am creating some variables that I am going to use these include FS for file system fill for the file F result for storing the result of each operation and buffer to store the data which we can read or write. B R and B W stores the counter to the read and write in the file and at last some variables to check the capacity of the card I am writing

this function send you out to make it easy to write the data to the UART. Buffer Size is used to determine the size of the buffer up to which it is filled and at last a function to clear the buffer Okay, let's start our main program by mounting the SD card first and we are going to use F mount function to do so there is some verbose in case of a failure.



Next, we are going to check the capacity of the card. We are going to use F get free function to do so. Though that total size and the free space will be sent via the UART. Now it's time to create our first file, F open function is used to open a file, and if the file does not exists, it will create it and then opens it here F open always does that job of creating file. Also f read and f the right are the access given to the file. Next, we are going to write some string to the file using F protests function and then close the file now in order to read it, we need to open it with Read Access and then copy the string from the file to the buffer using F get s function here filled dot F size is the size of the data in the file. Now we need to close the file let's build our code and test it take a look at the output of the UART it shows us the total size and the free space in the card. It also shows that the file was created and the string that was written the file this function used F get S and F potesse to read and write string. Next, we are

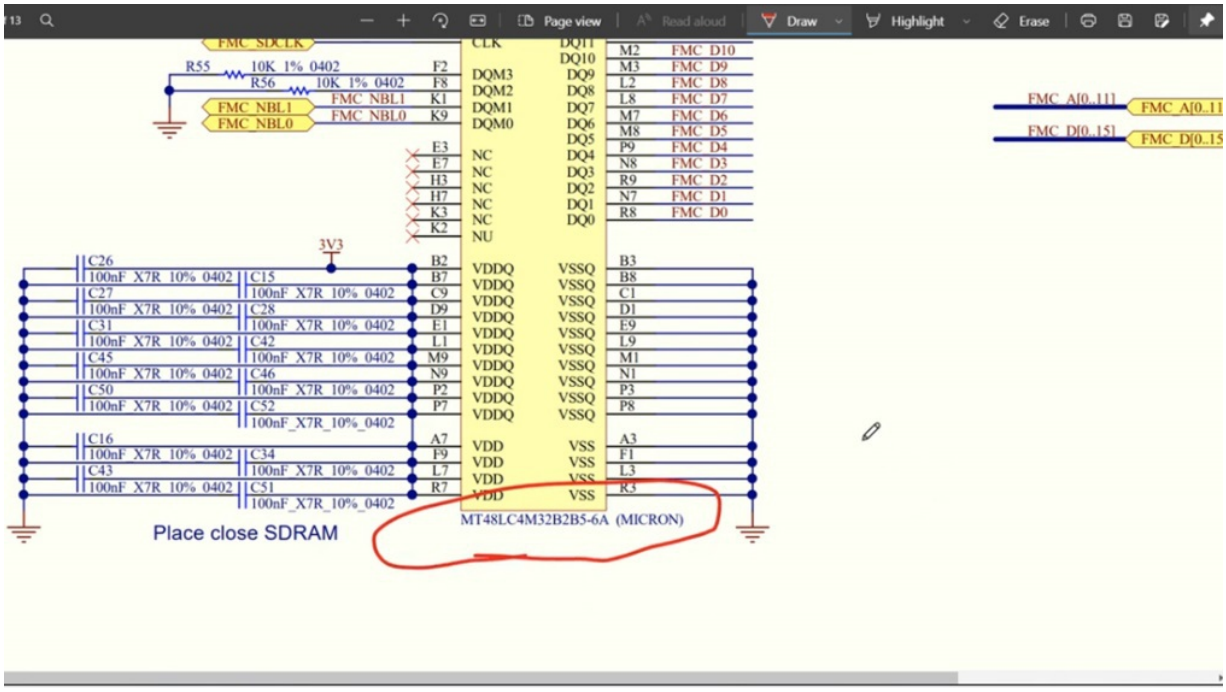
going to take a look at the F write and f read functions which do the same things. Here I am creating another file that is file two dot txt and I am going to use F right to write some data into it and f read to read the data from it. Also, buffer is clear to make sure that it is empty before we read the data. F write function takes the data from the buffer and write it into the file. Remember that it will start writing from the starting point in the file and any data will be overwritten, the bw is the pointer to the counter for the number of bytes written f read function will read the data from the file and save it in the buffer filled your tiff size is the size of the data in the file. b r is the pointer to the count variable for the number of bytes to read from the file. It's time to test the code now. As you can see, along with the results from the previous section, it gives us some new information. Also, it shows that the file two dot txt was created. And you can see the string that I wrote to the file to. Let's now see how to update an existing file. Here I want to update the second file. To do so I will use the function FL seek this function takes two parameters. The first is the pointer to the file. And the second is the offset from where you want to start the update process. Here I am using the file size as the offset. This means that I wanted to start from the end and continue writing. The rest of the reading process is same. Open the file, read the data, close the file. Let's test our code and check the results. As you can see, this data is up to the last section and this is the updated data. This string is first printed from the file too.



And after updating the file, it again gets printed along with the new string Let's see how to delete the files from the card we are going to use F fun Link function which takes only one parameter and that is the path to the file itself. If the result is okay, it will be printed in the output and at last we will unmount the SD card it's time to test it now as you can see both the files are removed and the SD card is unmounted successfully let me just show you what happens if we try to remove files which are not present in my SD card note that the success message is not shown now. And those repetitions you see on the top is due to the fact that the string is keep appending in the file.

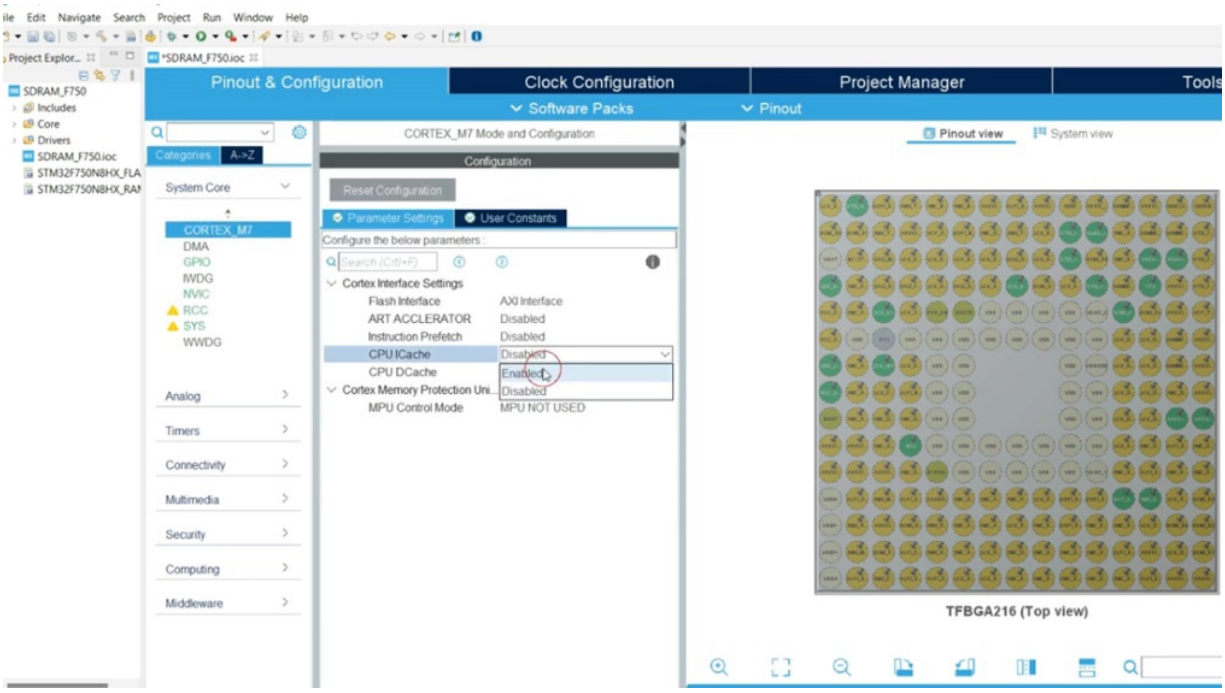
SDRAM IN STM32 MT48LC4

We will interface the external RAM with the help of the FMC peripheral. In today's project, we will just do the connection part for the external SD RAM.



We will also use mem copy functions which will prove that the RAM is being treated as the internal RAM later in the next project, I will show a very important use of it where we will use this SD RAM to store the LCD frame buffer into it. So let's start the today's project. During the research, I have found out that basically all the STM 32 microcontrollers which come preloaded with the SD RAM use this m t 48 l from the micron so this tutorial will be focused on this SD RAM only. You can see the schematic of the board I am using and it has the same memory let's start the cube ID and set up our project I am

using STM 32 F 750 Discovery Board give some name to this project and click Finish First things first, let's set up the clock.



I am using external crystal for the clock I am running the system at maximum possible 216 megahertz clock if you are using Cortex M seven enable the cache let's take a look at the datasheet for the SDRAM here you can see there are three variants available and these are their respective speeds. I have the variant six A and this one runs at 167 megahertz clock this is why I have configured the clock to the maximum so that we achieve this speed this is the connection for the SD RAM and you can see it uses the FMC peripheral and we will need this connection diagram during the setup let's see SD RAM peripheral here you can see we have SD RAM one and SD RAM two this depends on what bank is being used for the SD ram in the controller.

- Multibank ping-pong access
- Programmable timing parameters
- Automatic Refresh operation with programmable Refresh rate
- Self-refresh mode
- Power-down mode
- SDRAM power-up initialization by software
- CAS latency of 1,2,3
- Cacheable Read FIFO with depth of 6 lines x32-bit (6 x14-bit address tag)

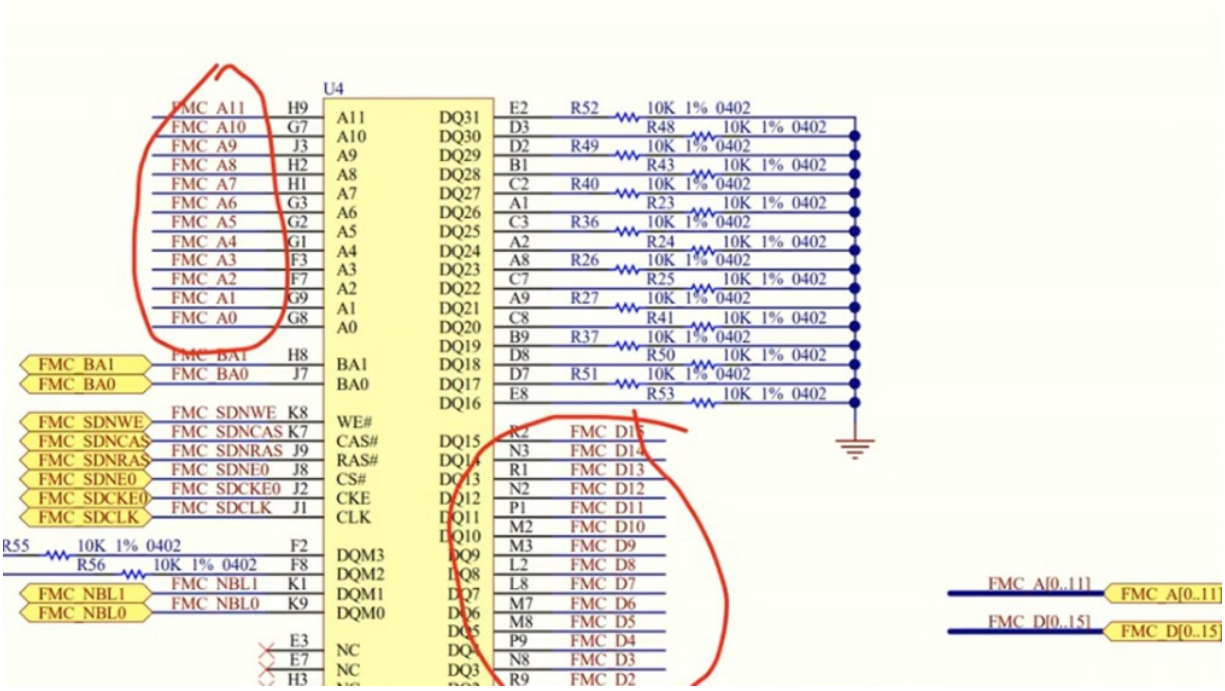
13.7.2 SDRAM External memory interface signals

At startup, the SDRAM I/O pins used to interface the FMC SDRAM controller with the external SDRAM devices must configured by the user application. The SDRAM controller I/O pins which are not used by the application, can be used for other purposes.

Table 88. SDRAM signals

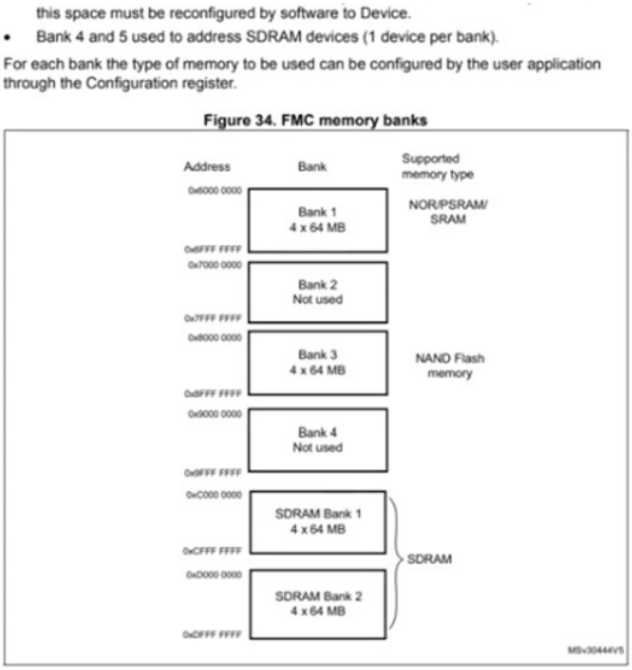
SDRAM signal	I/O type	Description	Alternate function
SDCLK	O	SDRAM clock	-
SDCKE[1:0]	O	SDCKE0: SDRAM Bank 1 Clock Enable *SDCKE1: SDRAM Bank 2 Clock Enable*	-
SDNE[1:0]	O	SDNE0: SDRAM Bank 1 Chip Enable *SDNE1: SDRAM Bank 2 Chip Enable*	-
A[12:0]	O	Address	FMC_A[12:0]
D[31:0]	I/O	Bidirectional data bus	FMC_D[31:0]
BA[1.0]	O	Bank Address	FMC_A[15.14]
NRAS	O	Row Address Strobe	-
NCAS	O	Column Address Strobe	-
SDNWE	O	Write Enable	-
NBL[3:0]	O	Output Byte Mask for write accesses	FMC_NBL[3:0]

For this particular board, the bank one is being used for the SD RAM so I am configuring the SD RAM one let's enable the clock and the chip. You can see here I don't have option to select the another one and this is because the SD RAM is connected to the bank one if you check the reference manual the SDRAM controller here you can see the option zero is for enabling the clock and chip for the SD RAM bank one. In the internal bank number we will use the four banks. This SDRAM have four banks of 32 megabits each and we will use all of them the address will be 12 bits and the data will be 16 bits. This part you can confirm in the schematics. Here we have the 16 data pins and here are the 12 address pins. Next all the pins make sure the pins are correct. As I have mentioned this cube MX sometimes configures the wrong pins So do cross check them with the schematics.



Also the maximum speed should be set to very high for all the pins. Now we will configure the parameters number of column address bits must be eight you can see it here the column addressing is eight bits from zero to seven. The rest of the setup should be same for all the SD rams keep the c a s latency to three clock cycles. Common clock to two cycles, enable the burst read and leave the read pipe delay to zero cycles configure the rest of the options as shown 2747322 This is it for the setup. Click Save to generate the project. Now the first thing we will do is copy these SD RAM library files into the project include the file we just copied into the project. Now go to the FMC initialization function. And here in the User Code section, we will initialize the SD RAM module separately, you can get this code from the link in the description. Here we need to make few changes. For example, what bank are we initializing? For me, it's bank one. So I am leaving it to default then the c a s latency. If you remember I have used the latency of three so I am changing this. All right now we will test the SD RAM let's start by creating the write and read buffers. Let's include the string dot h for the mem copy functions. We also need to define the address for the SD RAM. This address can vary depending on the microcontroller we can find it in the reference manual let's check the FMC memory distribution

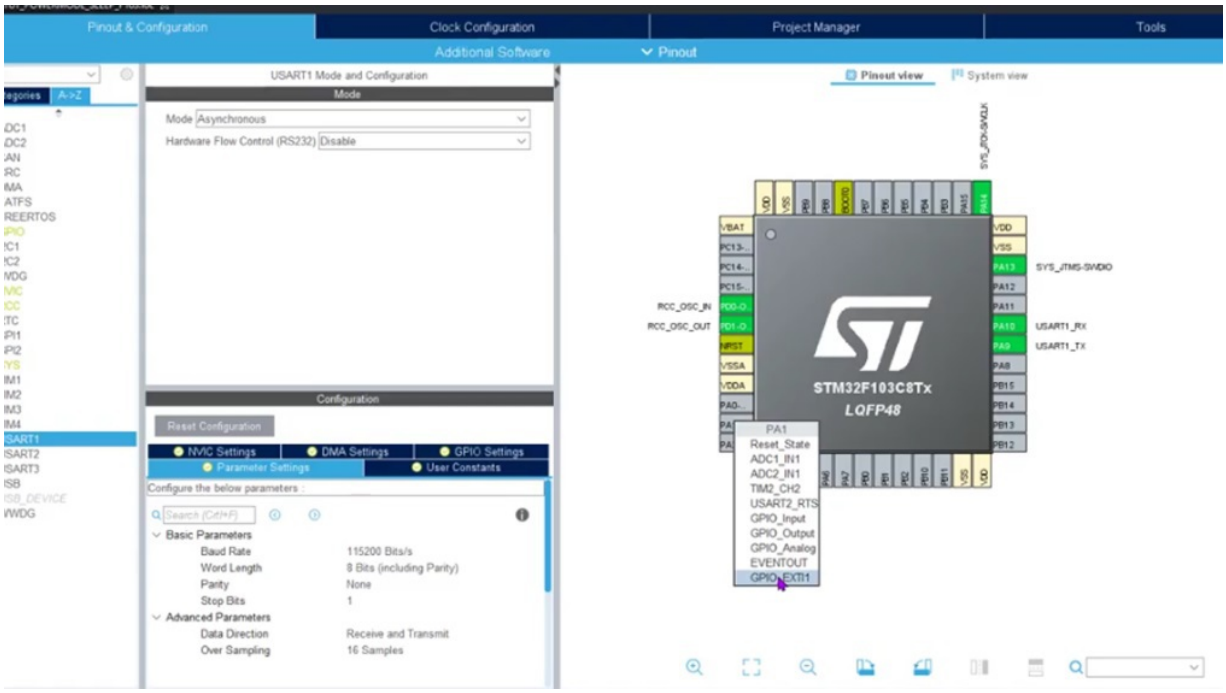
depending on if we are using the bank one or bank two, we have the different base address for the SDRAM since I am using bank one, the start address will be zero cross c million, we will write the main function now. First we will copy the data into the SD RAM and then we will read the data from this SD RAM let's build and debug this add the SD RAM address to the memory viewer. I am putting a breakpoint here in the initialization function. As soon as we step over this function, the data in the SD RAM starts showing in the memory viewer.



And now we will copy the data into the memory here you see the data gets copied into the SD RAM and now if we copy the data from this location into our buffer, we get the same that we copied. So we were able to use the mem copy function to read and write the data to the SD RAM. This means that the external SD Ram was seen as the internal memory to the microcontroller in the next project when I will cover the LT DC peripheral. I will make use of this SD RAM to store the frame buffer for the display.

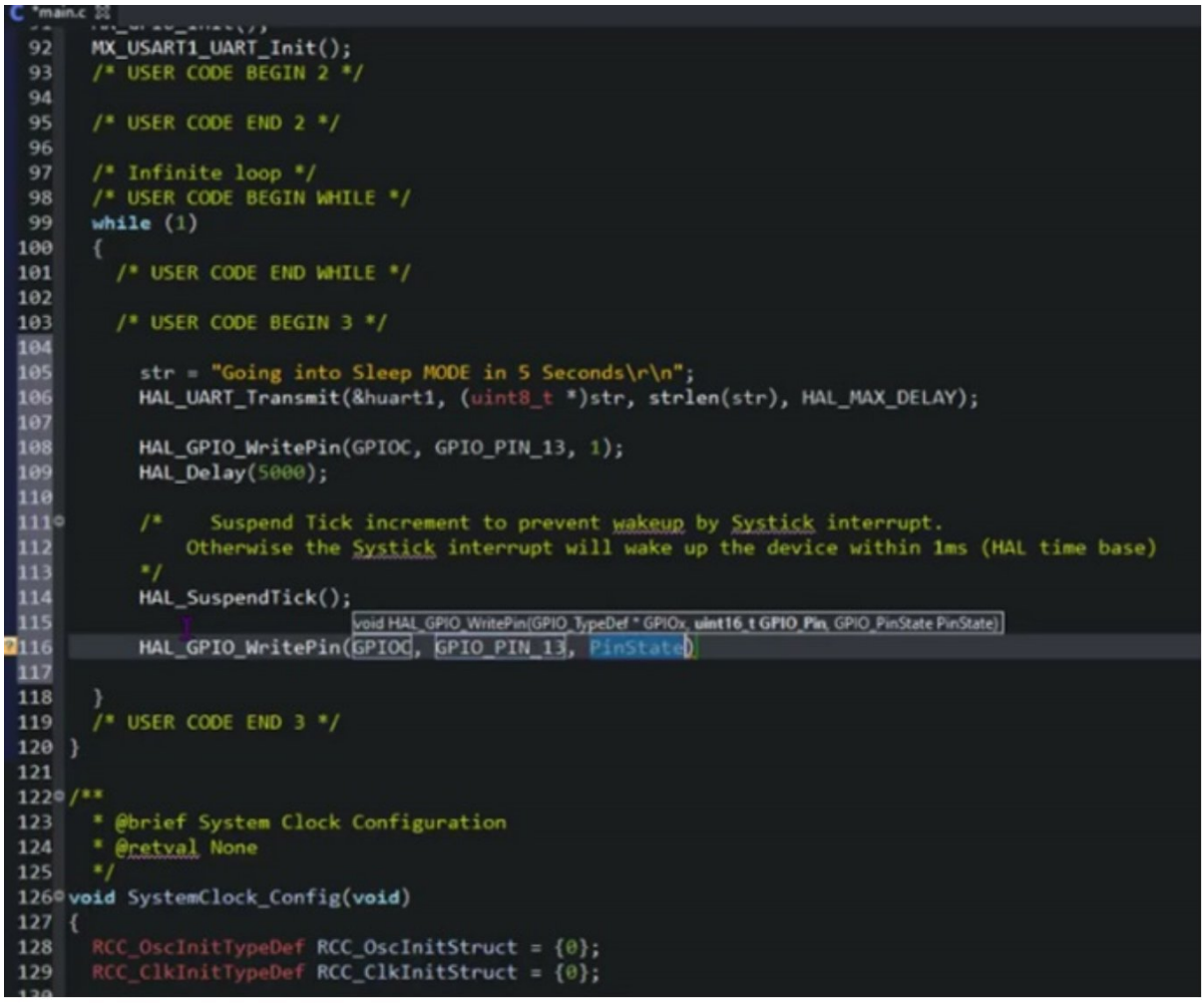
SLEEP MODE IN STM32F103 CUBEIDE LOW POWER MODE CURRENT CONSUMPTION

We will see the low power modes in STM 32. Let's take a look at the reference manual first. STM 32 F 103 have three low power modes in sleep mode CPU clock is off and all other clocks kept running in Stop Mode pretty much all clocks can be turned off and the last is standby mode which is the deep sleep mode.



In this tutorial we will cover the sleep mode let's start with the STM 32 Cube Id first I am using version one dot 3.0 And with this version we got a Run button right here create a new project I am using STM 32 F 103 ch controller give some name to the project and click Finish in the cube MX I am first enabling the external crystal for the clock Next select the SIS debug serial wire select the UART and enable the interrupt select the external interrupt on pin P A one basically we are going to wake up the device using UART interrupt and external interrupt on pin PA one select GPIONVIC Tab and turn on the external interrupt let's see the clock setup I am selecting maximum clock that is 72 mega hertz this is connected to the onboard LED click Save to generate the project I am creating a variable to store the string this string will be transmitted before the controller goes into the sleep mode. LED will be on for five seconds before going into the sleep mode the how tics must be suspended otherwise the SysTick interrupt will wake up the device within one millisecond turn off the LED just to indicate that the sleep mode is activated finally enter the sleep mode. main regulator will be on and the wakeup will be triggered by the interrupt. When the control reaches here, the CPU will go into the sleep. After the wakeup is triggered, next statement will be executed. So it is necessary that we resume the hell ticks first. This string will be printed next. And the LED will toggle few times to indicate the wakeup I am also going to write a UART interrupt callback function. RX data will store the received data from the UART. Stop the reception in interrupt mode to receive only one byte of data. As I mentioned earlier, the wakeup will be triggered by the UART interrupt and the external interrupts on pa one. To continue this reception, we must put the same thing inside the callback function also, a little correction guys, in f1 03 c eight, the LED pin is active low pin. So to turn the LED on, we must pull it low. Let's build the code we will directly run it as the debugging is not possible with the sleep mode. As you can see, the debugger shuts down automatically. I will reset the controller sleep mode will activate in five seconds. Take a look at the LED it goes off means sleep mode is activated. When the button is pressed, the external interrupt triggers and the device wakes up from sleep. You can see the LED blinking The while loop will run again and the controller will go back

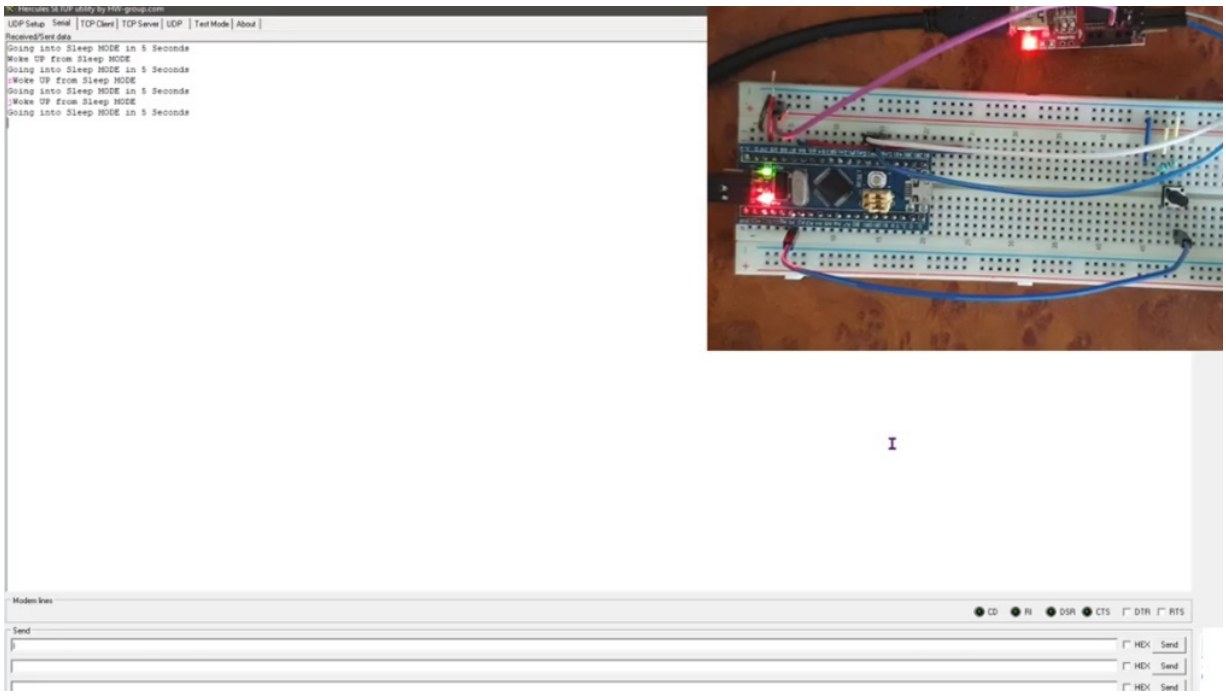
to sleep on sending the data from the UART the interrupt will be triggered and the CPU will wake up again.



```
92   MX_USART1_UART_Init();
93   /* USER CODE BEGIN 2 */
94
95   /* USER CODE END 2 */
96
97   /* Infinite loop */
98   /* USER CODE BEGIN WHILE */
99   while (1)
100  {
101    /* USER CODE END WHILE */
102
103    /* USER CODE BEGIN 3 */
104
105      str = "Going into Sleep MODE in 5 Seconds\r\n";
106      HAL_UART_Transmit(&huart1, (uint8_t *)str, strlen(str), HAL_MAX_DELAY);
107
108      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, 1);
109      HAL_Delay(5000);
110
111      /*    Suspend Tick increment to prevent wakeup by Systick interrupt.
112          Otherwise the Systick interrupt will wake up the device within 1ms (HAL time base)
113      */
114      HAL_SuspendTick();
115
116      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, PinState);
117
118  }
119  /* USER CODE END 3 */
120 }
121
122 /**
123   * @brief System Clock Configuration
124   * @retval None
125   */
126 void SystemClock_Config(void)
127 {
128   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
129   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

And the same thing will continue again. I also want to show the current consumption during sleep mode. For this purpose, I am using a 446 r e as it's easy to measure current in nucleo boards. As you can see, during the normal run the current consumption is 6.7 million pair. When the CPU enters the sleep mode, the current consumption reduces to 1.8 milli ampair. When the external interrupt is triggered, the CPU wakes up again and current increases. Now let's take a look at the sleep on exit function. When the control exits from the interrupt service routine, the CPU will go to sleep I will write the callback function for the External Interrupt line. These strings will print based on which interrupt is responsible for waking the controller now we will enable the sleep on exit function. But before

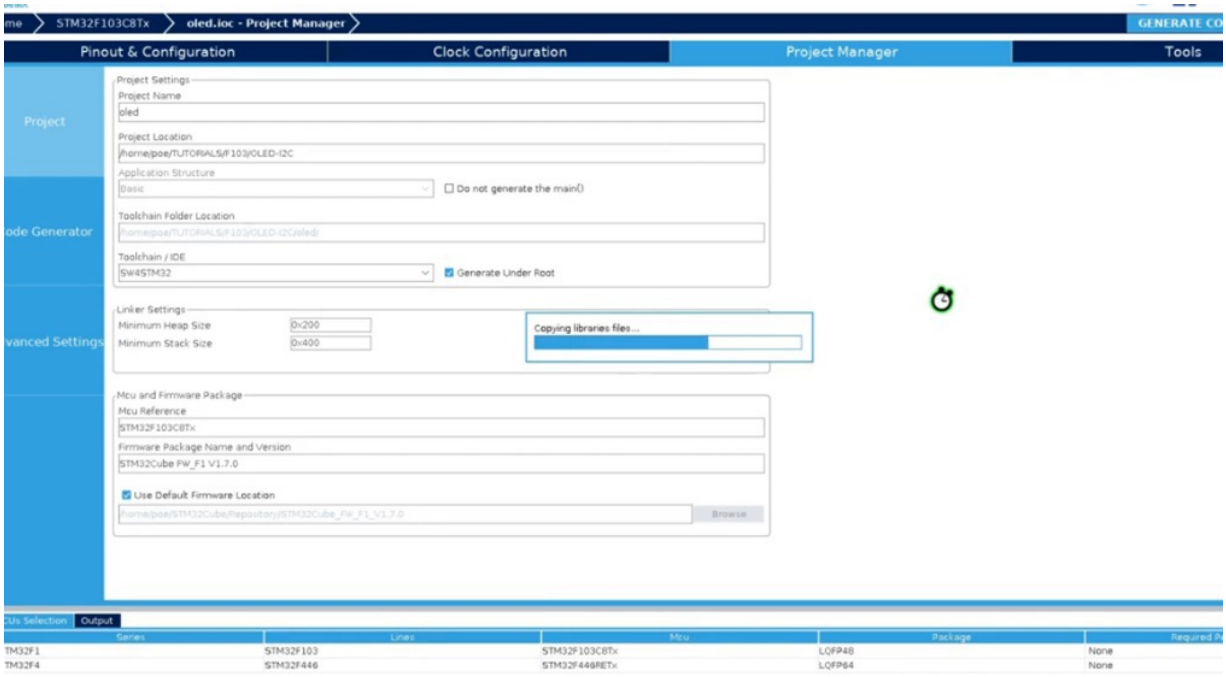
that I am disabling the sleep now function. also disable the cystic resume or else it will wake the CPU every one millisecond. Let's flash the code currently, the CPU is running. As soon as I send some data via the UART the interrupt is triggered, the processor will process the interrupt and the CPU will go to sleep. As you can see, no other instruction is being executed other than the callback functions. Basically, the CPU wakes up processes the interrupt handler and goes back to sleep.



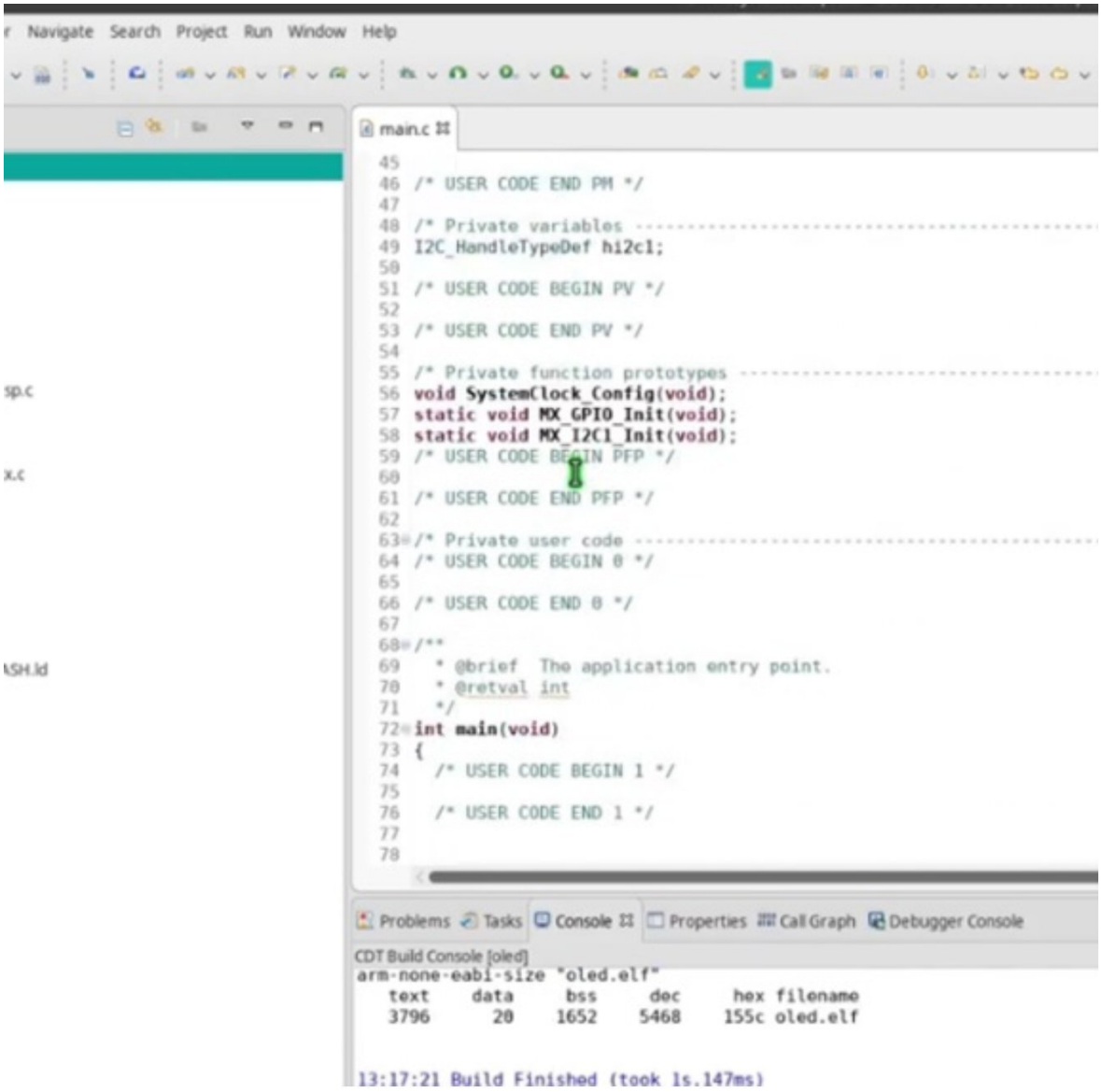
This is an interrupt driven control Let's disable the exit on sleep. But make sure that you enable the cystic before that or else the delay functions will not work. I I'm enabling the sleep now function. Let's build this code and run it as you can see, when the data is received by UART, the processor goes back to sleep. But when the external interrupt is triggered, sleep on exit is disabled and the control resumes from the very next statement in the while loop, which means this statement gets printed and the LED will toggle few times. After that the control will start from the beginning of the while loop. How TIG will be suspended and sleep on exit will be enabled again. As you can see, whenever the interrupt is triggered from the external pin, the entire while loop will be executed.

SSD1306 OLED AND STM32 128X64 SW4STM CUBEMX

We will use some advanced display functions in SSD 1306 I have modified the previous library and now we can display lines shapes, bitmaps and also animations. You can also scroll text and bitmaps on the screen in different directions.

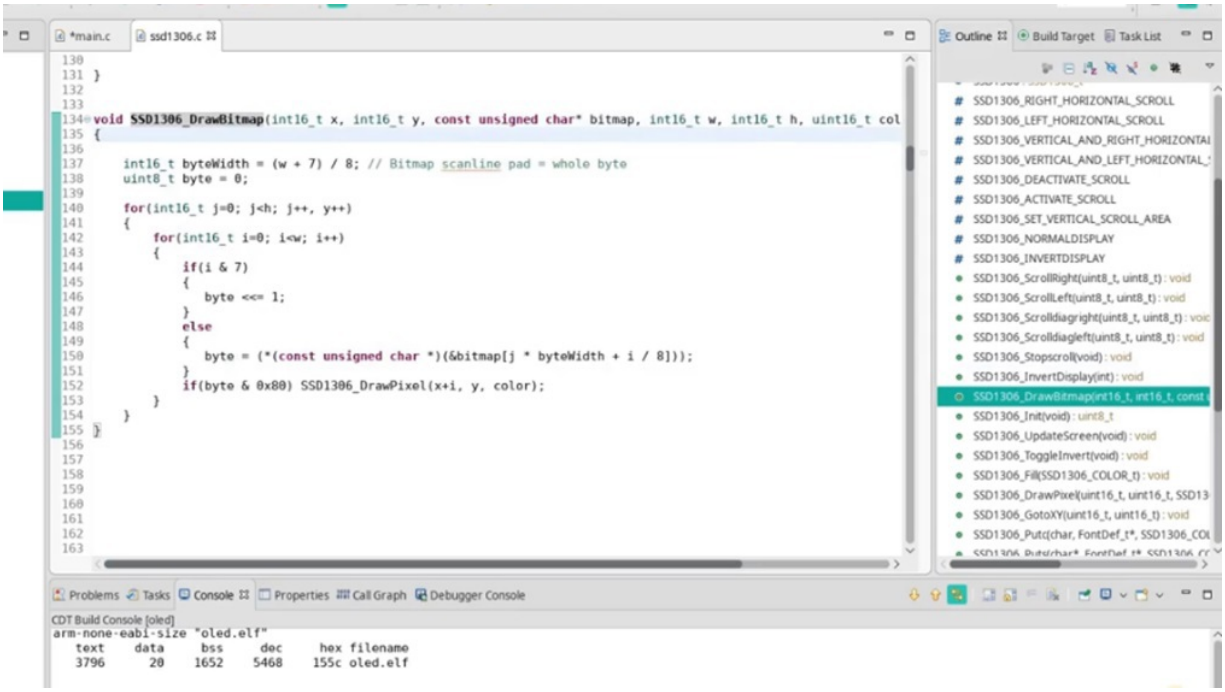


I am starting my setup with cube MX I am using my usual F 103 controller make sure you use the fast mode at 400 kilo hertz.

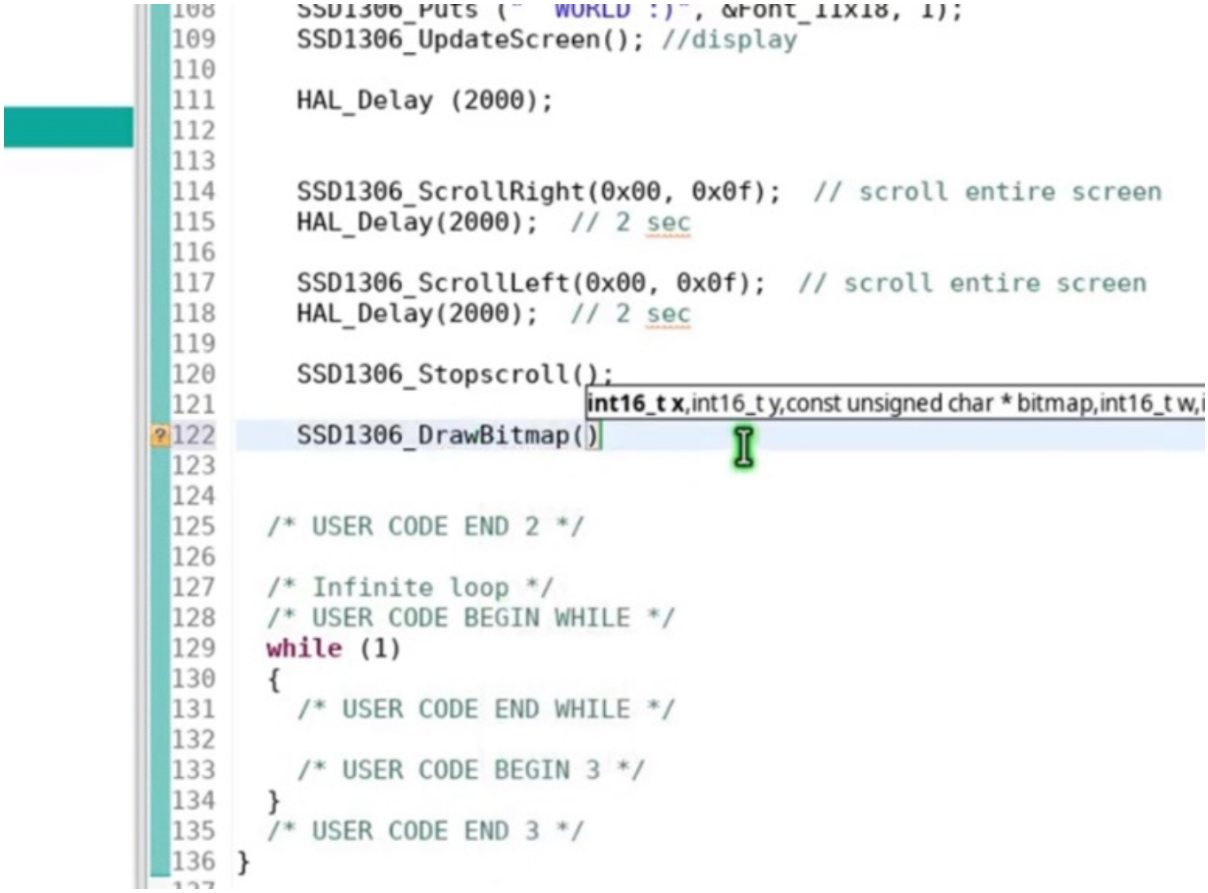


Let's copy some library related files in the project now we need to include all those header files that we copied let's just quickly build it once Okay, let's see the list of functions available for the programming. We will start by initializing the display let me just print some string on the display. And it is the usual one hello world. You have to write update screen function every time you want something to display on the screen. I want to keep this text for two seconds, and that explains the delay in the end. Let's check the list of functions again. We have scroll functions to right and left along with diagonal scrolling and then there is stopped scroll, we can also invert the display or display bitmap. Let's start with scrolling the screen, I

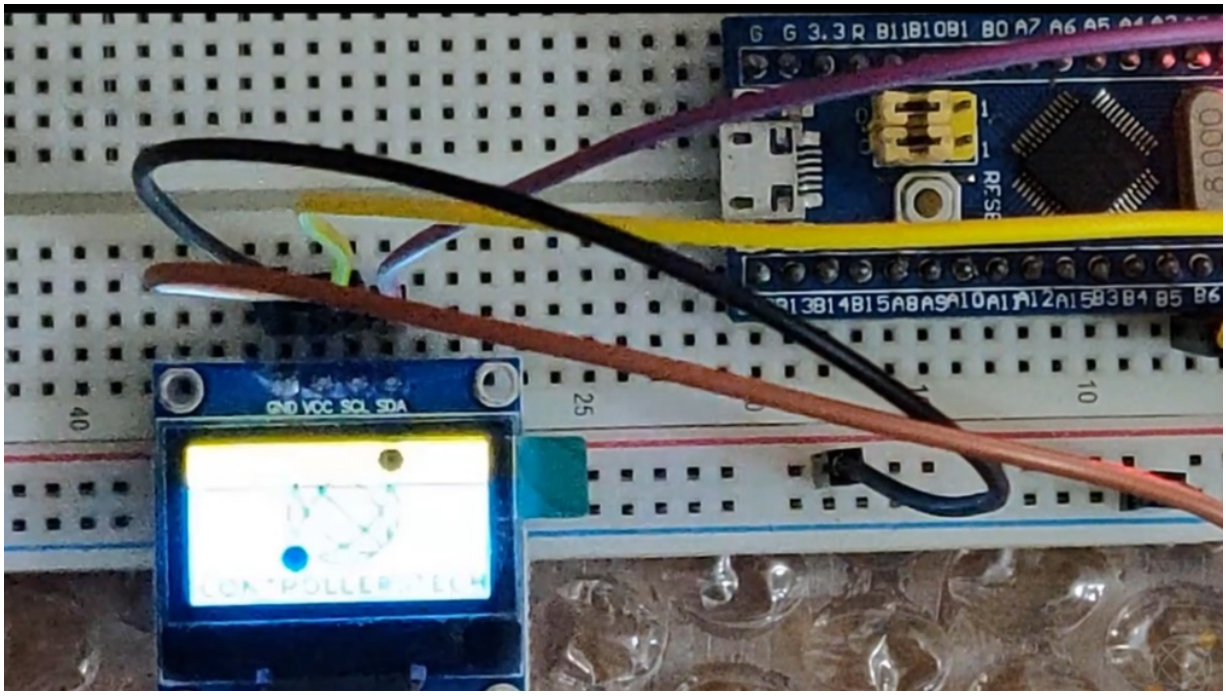
am going to scroll the text above. As you can see, this function takes two parameters. The first one is the row where you want the scroll to start. The second one is the end row. Well basically these are not the rows.



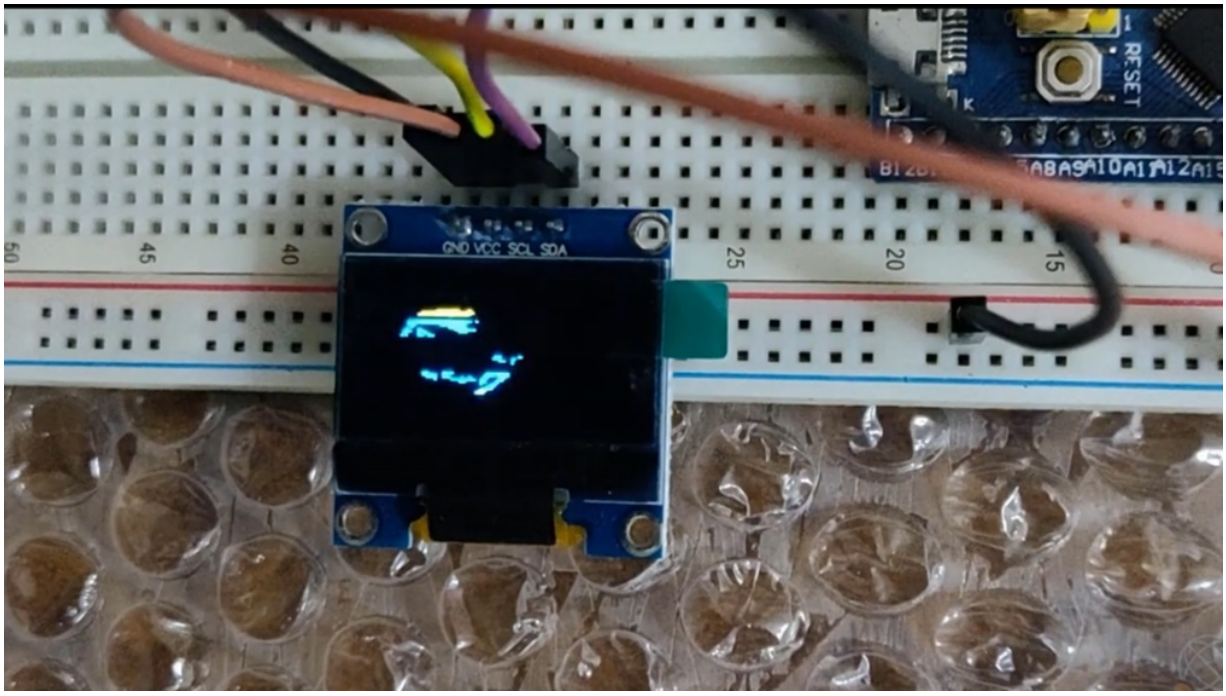
These are the pages. This display have seven pages. You can scroll however you want. As I want to scroll the entire screen start page will be zero and the end page will be seven providing delay we'll keep the scrolling for that amount of time, you should stop the scroll or else the screen will keep scrolling. Let's draw some bitmap now. I have this image right here which I want to display on the screen. I am using GIMP to modify the image. It's free and open source and it's good first thing we need to do is resize the image to 128 by 64. Next, we need to change the color index to one bit black and white only And now we can export this image as a bitmap image with BMP extension net Next I am using LCD assistance to convert it to the hex format you can Google it, it's easily available for download make sure that the byte orientation is horizontal and the size endianness is beak. After this just save the file and you will get your hex code I am creating another file in the project where I can store my bitmaps and I will name it bitmap itself just include it in the project and we are good to go.



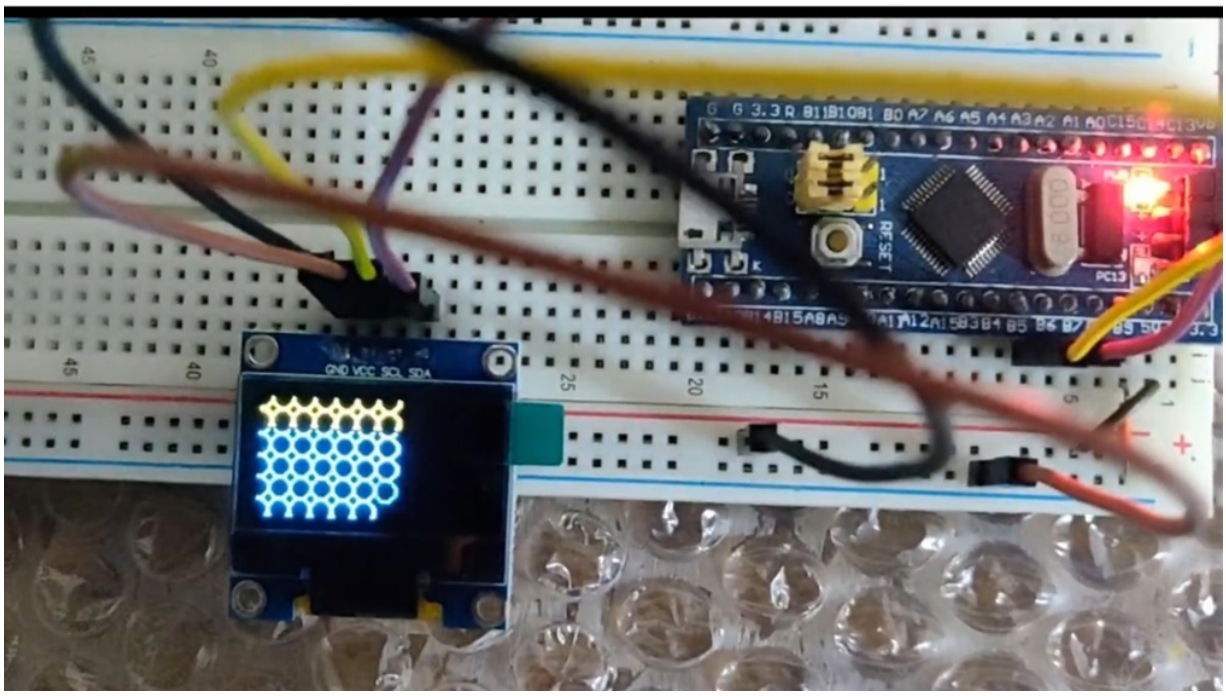
Draw bitmap takes six parameters, the first two other x and y coordinates where you want to start the draw from next is the pointer to the array than the size of the image and at last the color it will remain one in our case, we still need to update the display to print the image on the screen. I am going to keep the image for some time and then scroll the image in different directions. Note that there is some delay after every scroll.



Like I mentioned before, we need to give some delay or else we won't be able to see it scrolling. And at last I am going to invert the display and then normalize it again. Let's compile the code and download it to the controller to see the results. As you can see the text scrolling, the bitmap scrolling and the inverted display functions are working properly. Now let's see how to display any animation on the screen. I have this horse running GIF here you can just Google them they are available to download for free. Open it with GIMP look at the right side we have 10 different frames for this animation. First thing we need to do is scale our image to 128 by 64. And then we need to change the color index to one bit black and white. Look at the picture. If I show each frame one by one, you can actually see the animation. Next we need to hide all other frames and keep the first one then export it to the BMP extension just like we did in the bitmap. Then open the LCD assistance and convert it to the hex code I will create another file where I can keep this animation and I am naming it as porcelain name dot h go back to GIMP now and delete the first frame unhide the second frame and repeat the same steps you have to do this for all 10 frames now it's time to display this animation on our screen.



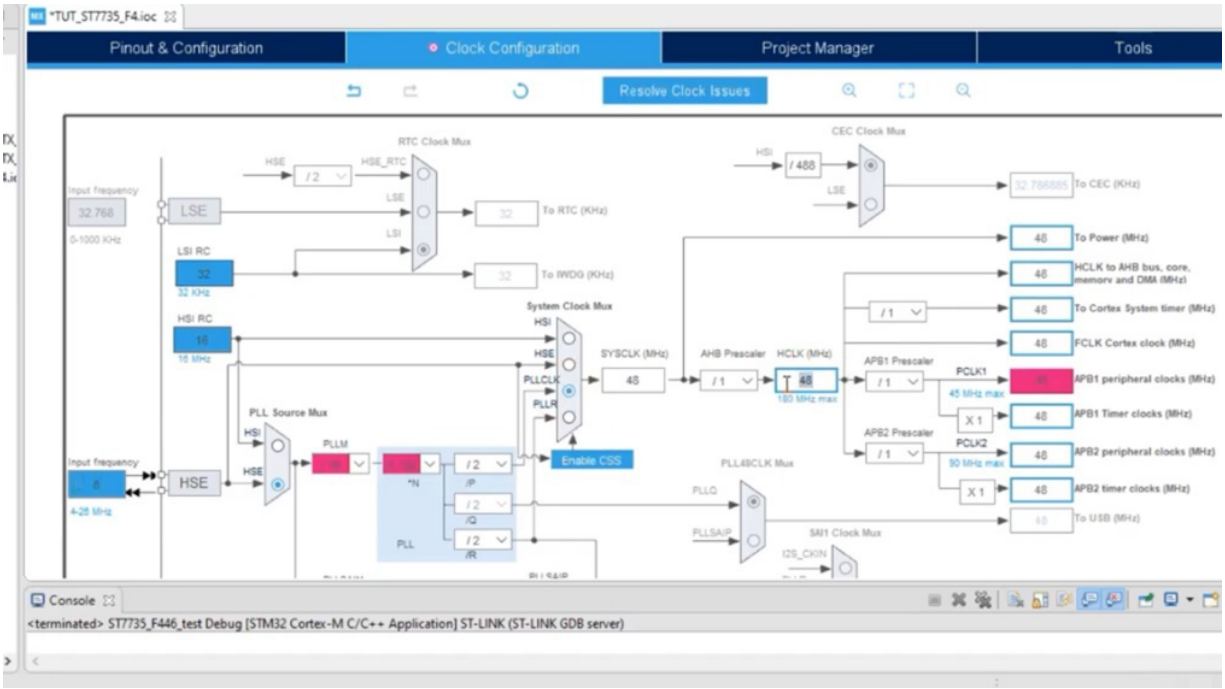
First we need to include the file we created in our project. Next in the while loop, clear the screen draw the bitmap of the first frame and update the screen. Repeat these steps for all other frames to. Let's Compile, download and run Yeah, so the horse is running good. I am only displaying the animation part here. Rest is usual. I also imported some tests for the screen. If you open the test dot c file, you will see the list of tests we can perform these includes lines, rectangles, circles and triangles.



Let's write them all they all take color as the only parameter except circles. It takes two parameters. One is radius and other is color. Let's test them now.

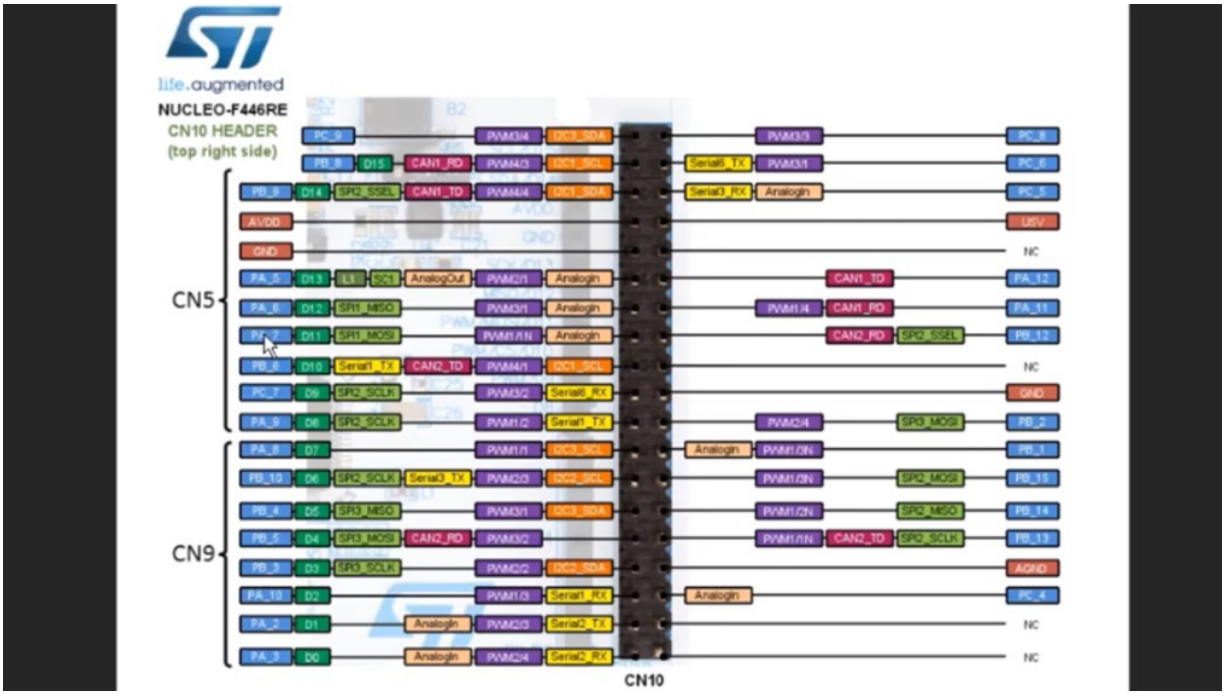
ST7735 TFT DISPLAY AND STM32 HAL

We will see how to interface S T 7735 TFT display with STM 32. I am using a 1.8 inches TFT display and I will use SPI to interface it. The libraries used in this project can be downloaded from the link in the description. Let's start by creating a project in cube ID I am using NUCLIO F 446 R E for this project here is our cube MX first of all I am going to set up the clock as the SP I will need it to set the baud rate I have eight megahertz Crystal and I want the system to run at maximum possible clock I am using SPI one and half duplex mode will be enough since we will only be sending data to the TFT as you can see the clock and the mossy pins got selected.

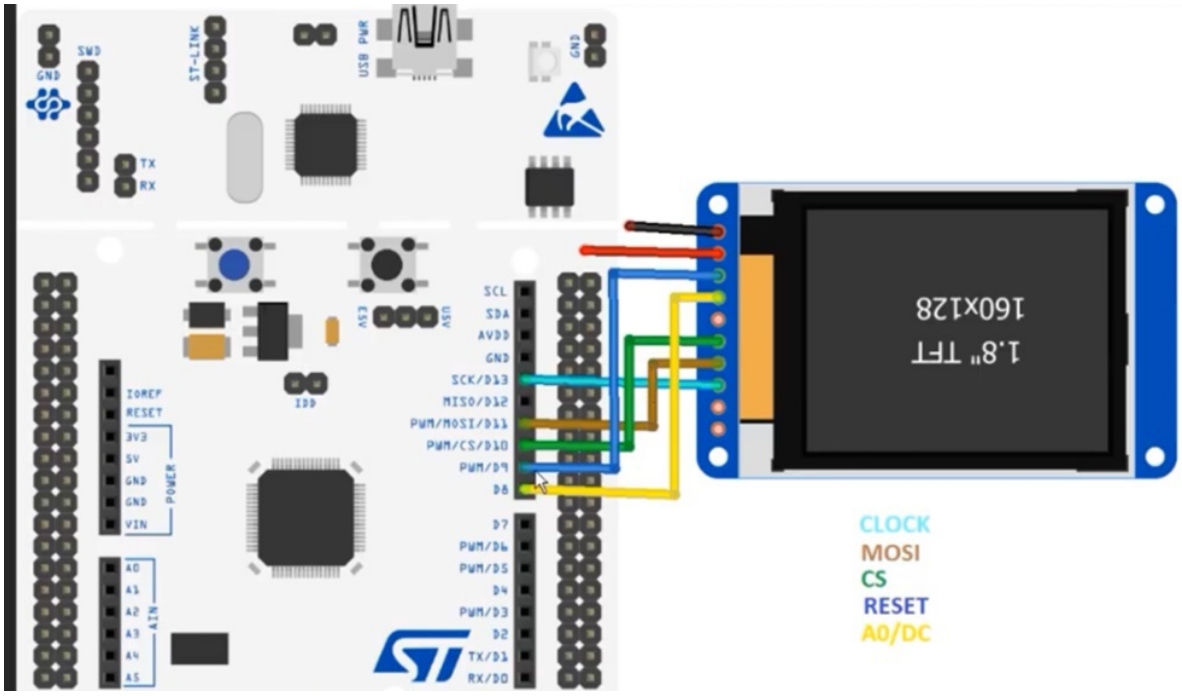


I am selecting the prescaler of 32 here so the baud rate is around 2.8 megabits per second. I am keeping this baud rate intentional for

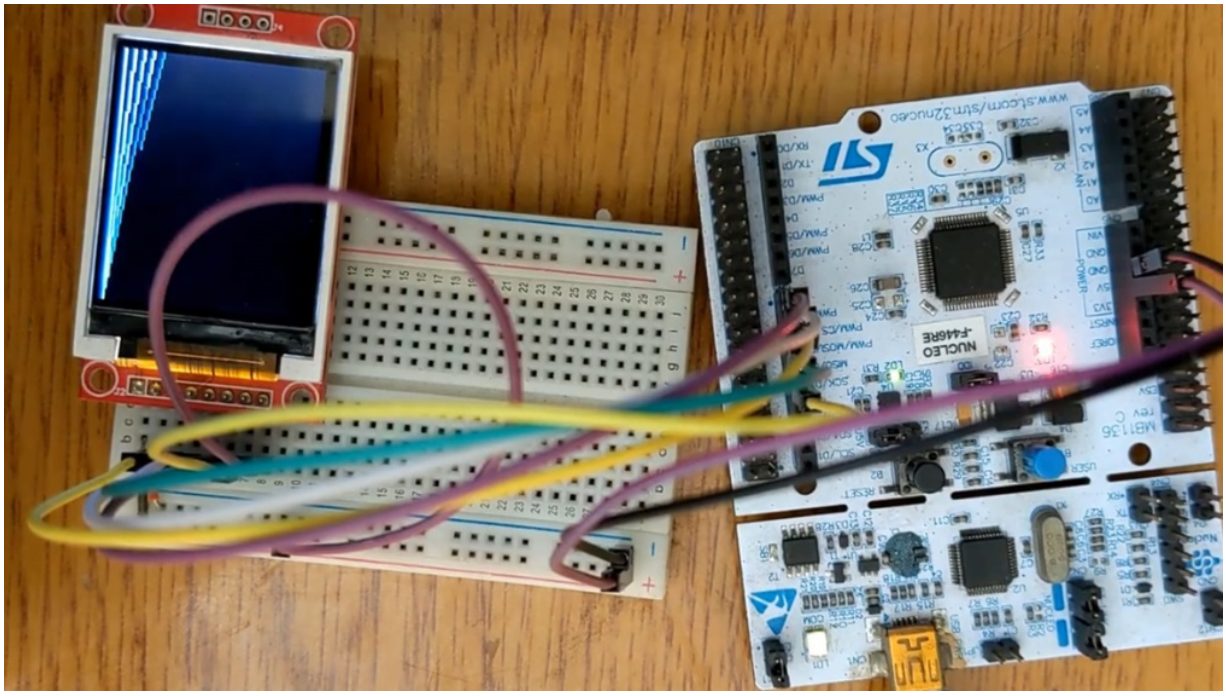
now. I will show you what happens when we increase the baud rate. This here is the pin out of my MCU pin PA five is the SPI clock pin PA seven is the mossy pin and I am going to use PB six PC seven and Pa nine also. Let's take a look at the connection.



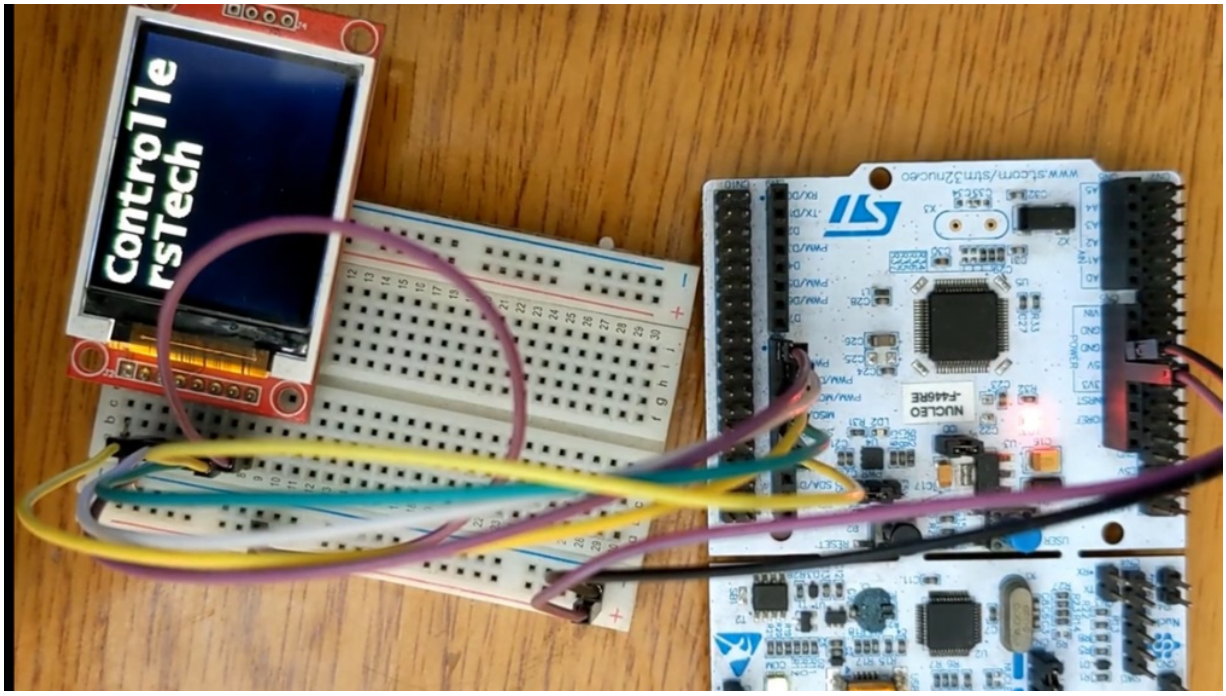
Here you can see other than the clock and the mossy pins, the CS reset and the DC pins are also connected so I am going to select the PV six pin for the chip select C seven for the reset and the a nine for the data or command pin. That's all for the setup. Click Save to generate the project here is our main file. Let's include the libraries first, copy the C files in the source directory and header files in the include directory. Let's take a look at the ES T 7735 header file first. Change the SPI handler here if you are using any other define the pins that you're using for CS reset and DC now define the type of the display that you are using and also define the new width and the new height.



That's all the changes you need to make. Here are some functions available specific for the SP 7735. And GFX functions have all the common functions that are used in the TFT. I have included a test all function which will perform some tests by drawing different shapes on the display. Let's go back to our main file now. Include the SD 7735 dot h and the GFX functions dot h files. Initialize the display first. This function takes the rotation as the parameter I will keep the default orientation at first so rotation will be zero.



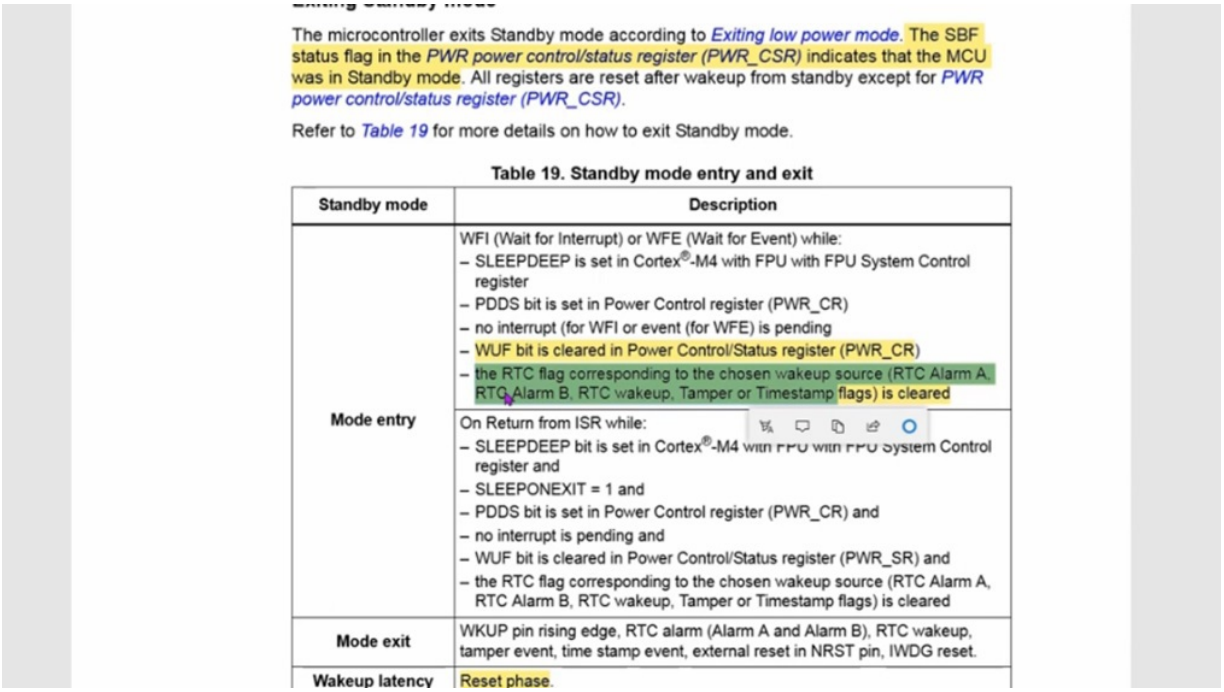
Now fill the screen with black color we have more colors available. As you can see the color list is right here and now I am going to call the test or function to perform the tests. Let's build this now. We don't have any errors let's flush it you can see all the tests being performed on the display now I talked about the SPI baud rate in the beginning let's try changing it I am going to increase it to 11 megabits per second let's build them flash the new code you can see the display processing is faster now.



So choose the baud rate according to your requirement. Now let's print some strings on this display. I am setting the rotation to zero and I am going to print hello there are some more fonts available in the fonts dot c file this here is the color of the text and then the background color then I am rotating the display and printing world rotate again and print from and again let's build and flash it you can see the display rotation and text printing. Both are working all right this is it for this project. I hope the process was clear.

STANDBY MODE IN STM32 LOW POWER MODES CUBEIDE

I will cover the standby mode in STM 32. Let's take a look at the datasheet. These are the common steps in other modes to other than these, the wakeup flag must be clear before going to standby mode.



The microcontroller exits Standby mode according to *Exiting low power mode*. The SBF status flag in the *PWR power control/status register (PWR_CSR)* indicates that the MCU was in Standby mode. All registers are reset after wakeup from standby except for *PWR power control/status register (PWR_CSR)*.

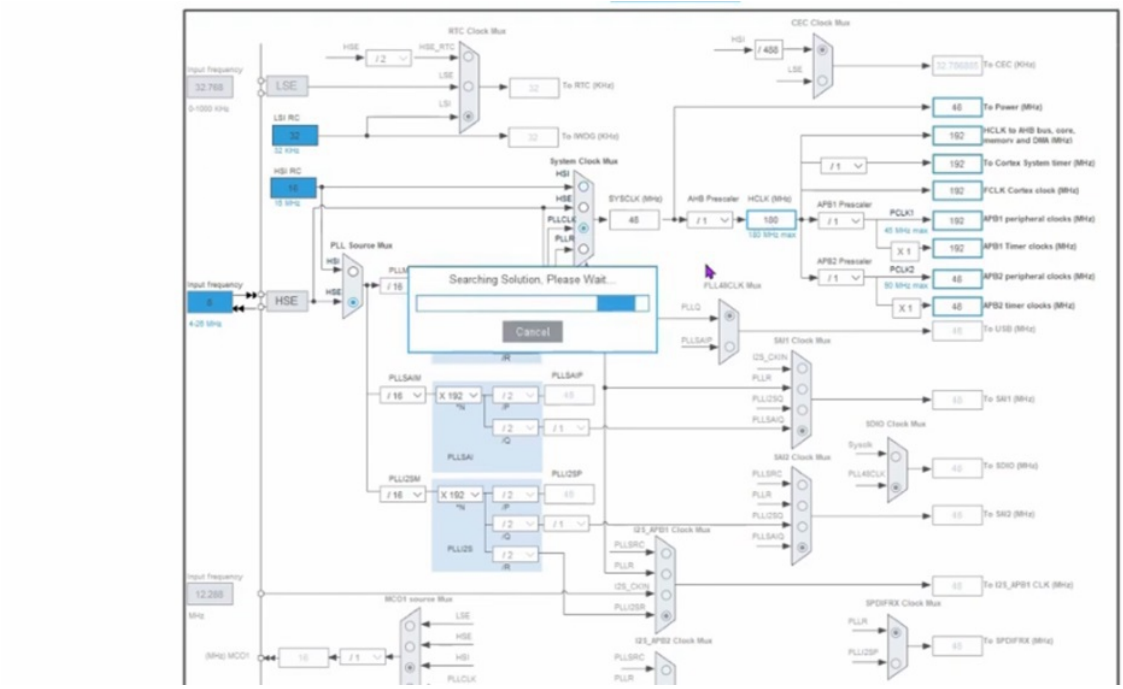
Refer to *Table 19* for more details on how to exit Standby mode.

Table 19. Standby mode entry and exit

Standby mode	Description
Mode entry	WFI (Wait for Interrupt) or WFE (Wait for Event) while: – SLEEPDEEP is set in Cortex®-M4 with FPU with FPU System Control register – PDDS bit is set in Power Control register (PWR_CR) – no interrupt (for WFI or event (for WFE) is pending – WUF bit is cleared in Power Control/Status register (PWR_CR) – the RTC flag corresponding to the chosen wakeup source (RTC Alarm A, RTC Alarm B, RTC wakeup, Tamper or Timestamp flags) is cleared
	On Return from ISR while: – SLEEPDEEP bit is set in Cortex®-M4 with FPU with FPU System Control register and – SLEEPONEXIT = 1 and – PDDS bit is set in Power Control register (PWR_CR) and – no interrupt is pending and – WUF bit is cleared in Power Control/Status register (PWR_SR) and – the RTC flag corresponding to the chosen wakeup source (RTC Alarm A, RTC Alarm B, RTC wakeup, Tamper or Timestamp flags) is cleared
Mode exit	WKUP pin rising edge, RTC alarm (Alarm A and Alarm B), RTC wakeup, tamper event, time stamp event, external reset in NRST pin, IWDG reset.
Wakeup latency	Reset phase.

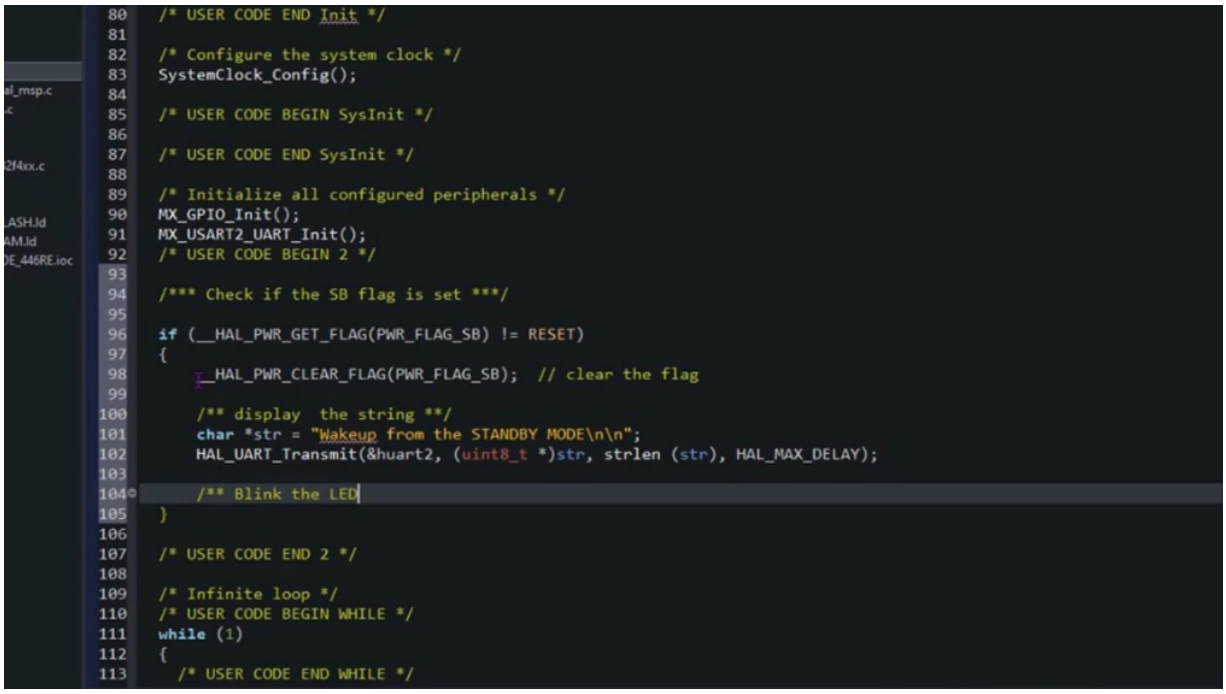
If the RTC is chosen than RTC wakeup flag must also be cleared we can exit the standby mode using wakeup pin or RTC interrupt or watchdog timer the latency of standby mode is same as reset. When the mcu wakes up from the standby mode, the SPF flag in the power control register is set we will use this fact to perform some operations after waking up from the standby let's start by creating a

project in the cube ID I am using STM 32 F 446 r e controller give some name to this project and click Finish. Here is the cube MX first of all I am selecting the external crystal for the clock enable the UART for communicating with the computer P A five is connected to the LED enable the wakeup pin let's go to the clock setup.



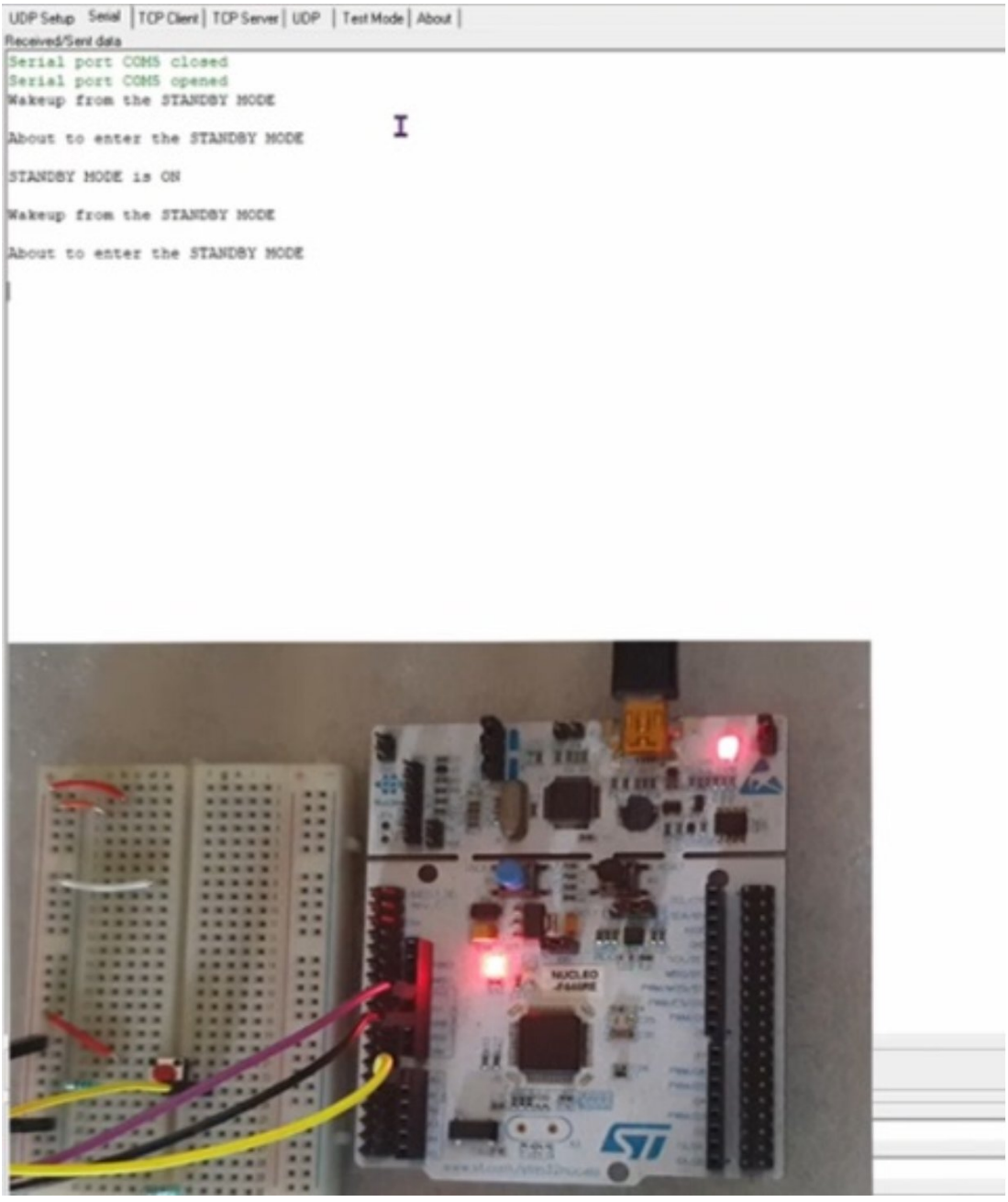
I have eight megahertz external Crystal and I want the system clock of 180 megahertz click Save to generate the project when the mcu wakes up all these will be executed again to confirm if the system was resumed from standby mode we will check this SP flag if the flight is set means the MCU woke up from the standby mode first thing we will do is clear the flag next we will display this string on the console and then toggle the LED few time with some small delay. Last disable the wake up pin wake up pin one is common in all MCU wake up pin two and three are supported by these microcontrollers only. I am using wakeup pin one As it's present in all the MCU and it is the pin PA zero now this is the procedure to enter the standby mode as mentioned in the datasheet we must clear the wakeup flag before entering the standby I am going to display this string blink the LED for the indication this blinking rate is different from the initial one we must enable the wake up PIN before entering standby just to make sure now, finally enter the standby mode include the string dot

h file for string related operations let's build this code I am going to use the debugger. As you can see when I press the reset, the string got printed on the console LED blinks at 750 milliseconds delay and the MCU goes into the standby mode. When the button is pressed, the mcu wakes up and string got printed and LED blinks at 200 milliseconds to indicate the wake up. Same thing happens if I press the button again. We can also use the RTC to wake the device up. I have already covered the RTC setup in my Stop Mode project. In RTC wakeup, select internal wakeup make sure you turn on the interrupt.



Leave everything as it is we don't need to set the time. This is a periodic wake up and setting time doesn't affect it now go to the RTC initialization function and copy this function also commented out here just like we enable the wakeup pin. We are also going to enable the RTC wake up inside the main function I have already explained this calculation in my sleep mode project. You can check it out on the top right. Anyway this calculation is self explanatory. I am using the periodic delay of five seconds here. As mentioned in the datasheet if we are using RTC to wake the MCU up, we need to clear the RTC wakeup flag also after wake up, just like disabling the wakeup pin we must also deactivate the RTC wake up Let's build

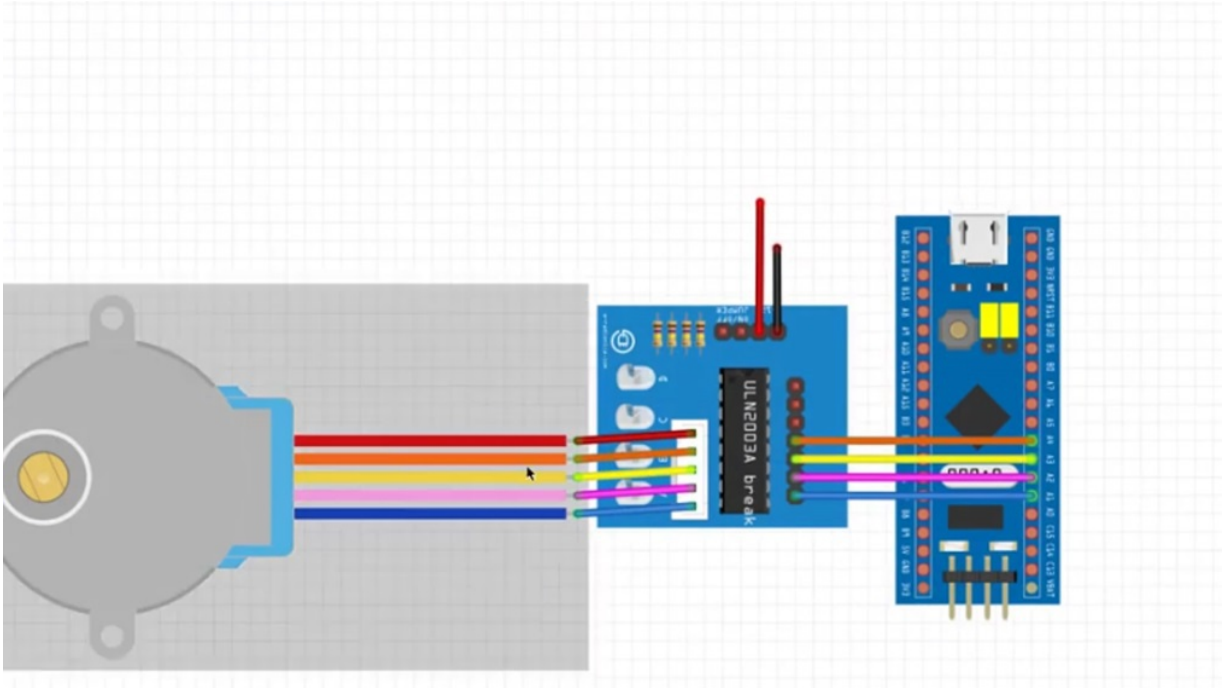
this code and run it you can see the current consumption on the ammeter also. Right now the range is set at 200 milliamps. When the device is running normally, the consumption is around 36 milliamps. But when it goes into the standby mode, it's showing zero. Let me reduce the range to two milliamps. Now you won't be able to see the higher range but you can see the current during standby mode.



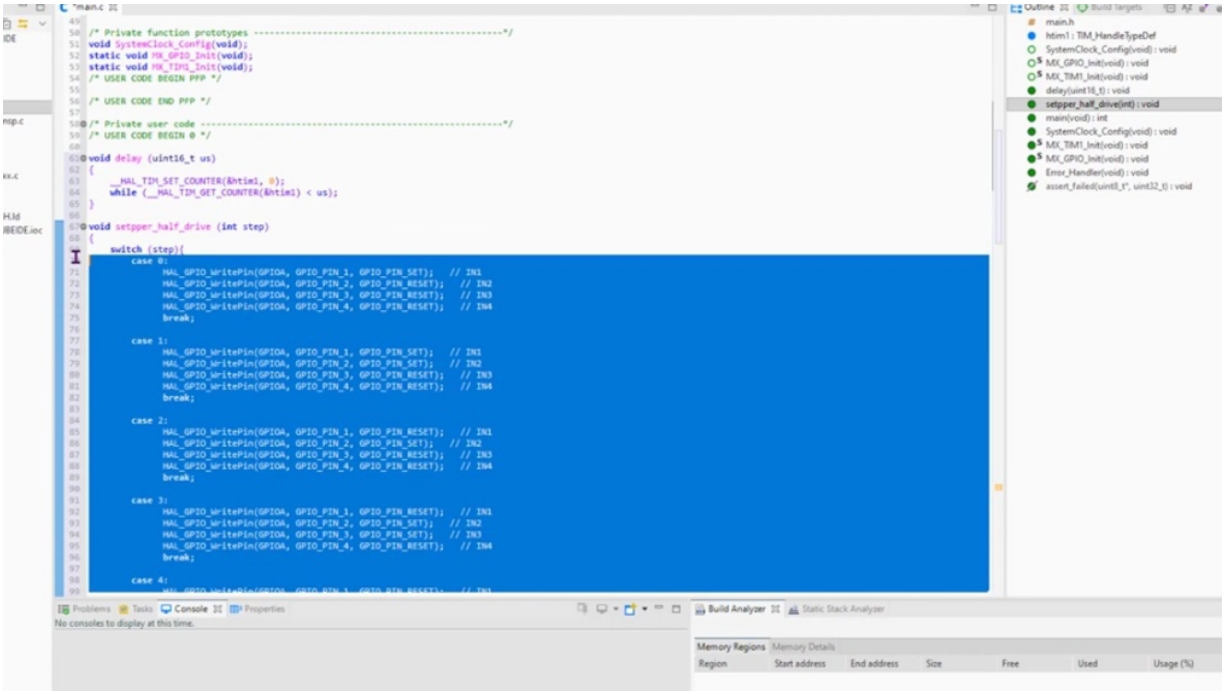
Yes, it's around point 002 milliamps, which is two micro ampair. This is extremely low consumption. The MCU is waking up every five seconds because of the periodic wake up of RTC. We can also use the wakeup pin to wake the controller. You can see the SPF flag and Wu F flag described in the datasheet. Now all three low power modes are covered. in stop mode, there is one feature called under drive mode. It's not present in Cortex M three and maybe in other series also. But it's present in F 446 R E and I will cover it in few days. You can use these low power modes according to your application.

STEPPER MOTOR AND STM32 ANGLE RPM AND DIRECTION CONTROL CUBEIDE

I am going to make it a bit more interesting. This is another project on stepper motor and this time we will vary the RPM change the direction and also step the motor through some angles. Let's start by creating project in cube ID I am using F 103 C eight controller let's give a name to this project let's set up the cube MX part. First of all I am enabling the external crystal for the clock. Next select the serial wire debug. Now I am setting pa one to PA four as output. These pins are connected to the stepper I will show you the connection in a while. Let's set the clock now. I want the controller to run at maximum clock possible here in this case 72 megahertz. Note that APB to Timer Clock is also at 72 megahertz setting the timer for microseconds delay if you don't know what I am doing check the project on the top right setup is complete click Save to generate the project this is the connection stepper motor is connected via UL n 2003 motor driver to the F 103 controller pin a one goes to I n one or two to I n two and so on motor driver is needed because the current consumption is stepper is high let's write the code now.



First of all I am writing a function to create delay in microseconds set the counter to zero wait for the counter to reach the entered value now I am going to write a function for the half drive for the motor as shown in my websites, the motor takes eight steps in the half drive. First only A is high than A and B both are high, the only B is high, then B and C are high and so on. Here A B C and D represents the pins connected to the motor. Hear only first is high here first and second both are high, the only second than second and third and so on. Now the motor takes 4096 steps in half drive to complete one revolution. eight steps in half drive is called a sequence.

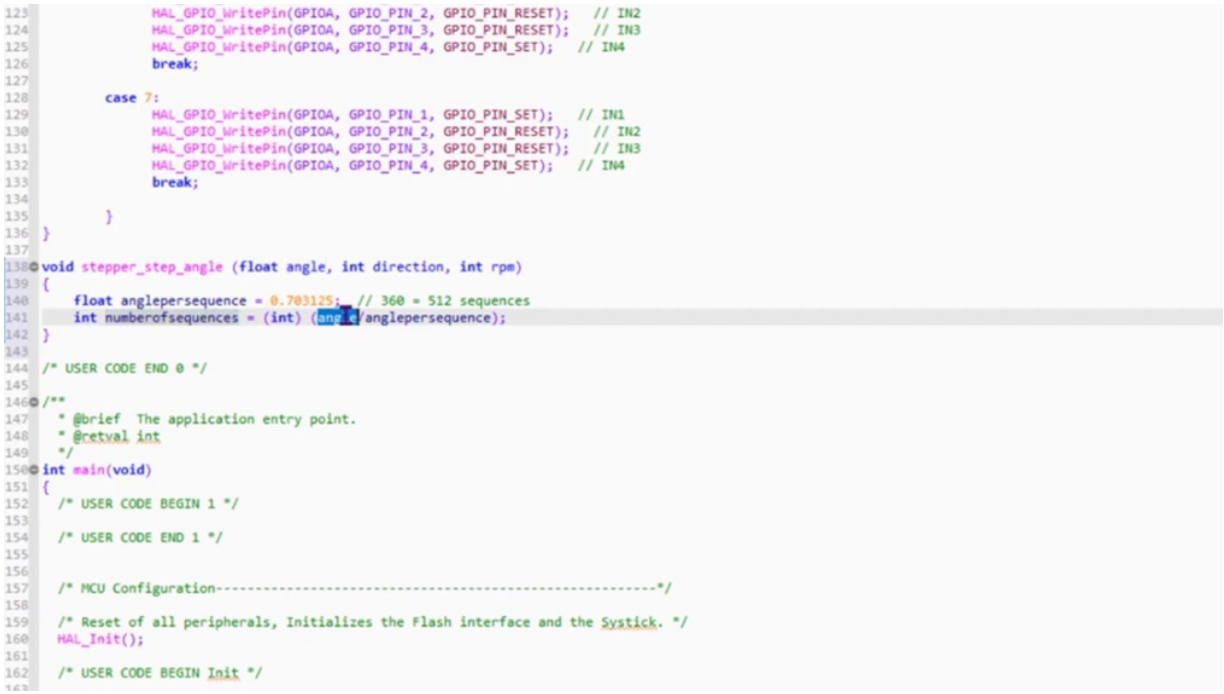


```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void delay (uint16_t us)
{
    __HAL_TIM_SET_COUNTER(&htim1, 0);
    while (__HAL_TIM_GET_COUNTER(&htim1) < us);
}

void setpper_half_drive (int step)
{
    switch (step){
        case 0:
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);   // IN1
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);   // IN2
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);   // IN3
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);   // IN4
              break;

        case 1:
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);   // IN1
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);   // IN2
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);   // IN3
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);   // IN4
              break;

        case 2:
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);   // IN1
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);   // IN2
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);   // IN3
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);   // IN4
              break;

        case 3:
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_RESET);   // IN1
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);   // IN2
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_SET);   // IN3
              HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);   // IN4
              break;

        case 4:
```

This way the motor completes 512 sequences for one Evolution I am defining here the number of steps this motor takes for one revolution if you were using Wave drive or full drive then steps per revolution would be 2048 but the sequences would still the 512 as the motor now takes four steps to complete a sequence this here is the function to set the RPM for the motor I have tested it and found out that maximum RPM for this stepper is around 14 Although it misbehaves at 14 So I wrote here the Max can be 13 you can use this function as a delay function now inside the main function first we have to start the timer let's see the calculation if I want to run the motor at rpm of 14 the delay would be 1046 microseconds to run it one revolution per minute the delay would be 14,648 microseconds as I mentioned 512 sequences are needed for one complete revolution I am going to write a for loop for these 512 sequences for each sequence the motor is going to take eight steps making it 4096 Steps let's set the RPM of five at first so, there is no error in the code let's debug it then select STM 32 application leave everything default and click OK run the code now as you can see the motor is rotating as expected you can time this project and count the RPM this part is exact one minute long so it is easier to count it. The

stepper made five revolutions as we asked it to do let's increase the RPM now.

I am making it 13 flash the code again notes that the speed has increased I am only running this part for 30 seconds if you want to count the motor completed around six and a half revolutions in 30 seconds exactly what we needed now it's time to write the function for the angle the motor will rotate by the angle in the parameter we can also control the direction and RPM like I mentioned there are 512 sequences for one revolution which means 0.7031 to five degrees per sequence we can find the angle the number of sequences motor needs to complete we can also control the direction by reversing sequence of the steps in the half drive let's test it for the different angles here I am writing 45 degrees first and clockwise direction with an RPM of 10 and it will keep repeating every one second let's build it and test I am steeping through this code to show you the angle calculation. Now when we enter the function, the angle is 45 degrees number of sequences come up to be 64 This is 1/8 of the 512 as 45 is of 360 Let's run it now you can see the results yourself. The motor rotates for 45 degrees every one second.



```
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);   // IN2
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);   // IN3
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);   // IN4
            break;

      case 7:
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_1, GPIO_PIN_SET);   // IN1
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_RESET);   // IN2
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_3, GPIO_PIN_RESET);   // IN3
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);   // IN4
            break;

      }
}

void stepper_step_angle (float angle, int direction, int rpm)
{
    float anglepersequence = 0.703125;  // 360 = 512 sequences
    int numberofsequences = (int) (angle/anglepersequence);
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */


  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
```

Let's change the angle to 22.5 degrees and flash the code again I will change the RPM to 12 As expected, the motor now takes 22.5 degrees in each second this is it guys? I hope you will make some better use of it.

STOP MODE IN STM32 CUBEIDE LOW POWER MODE

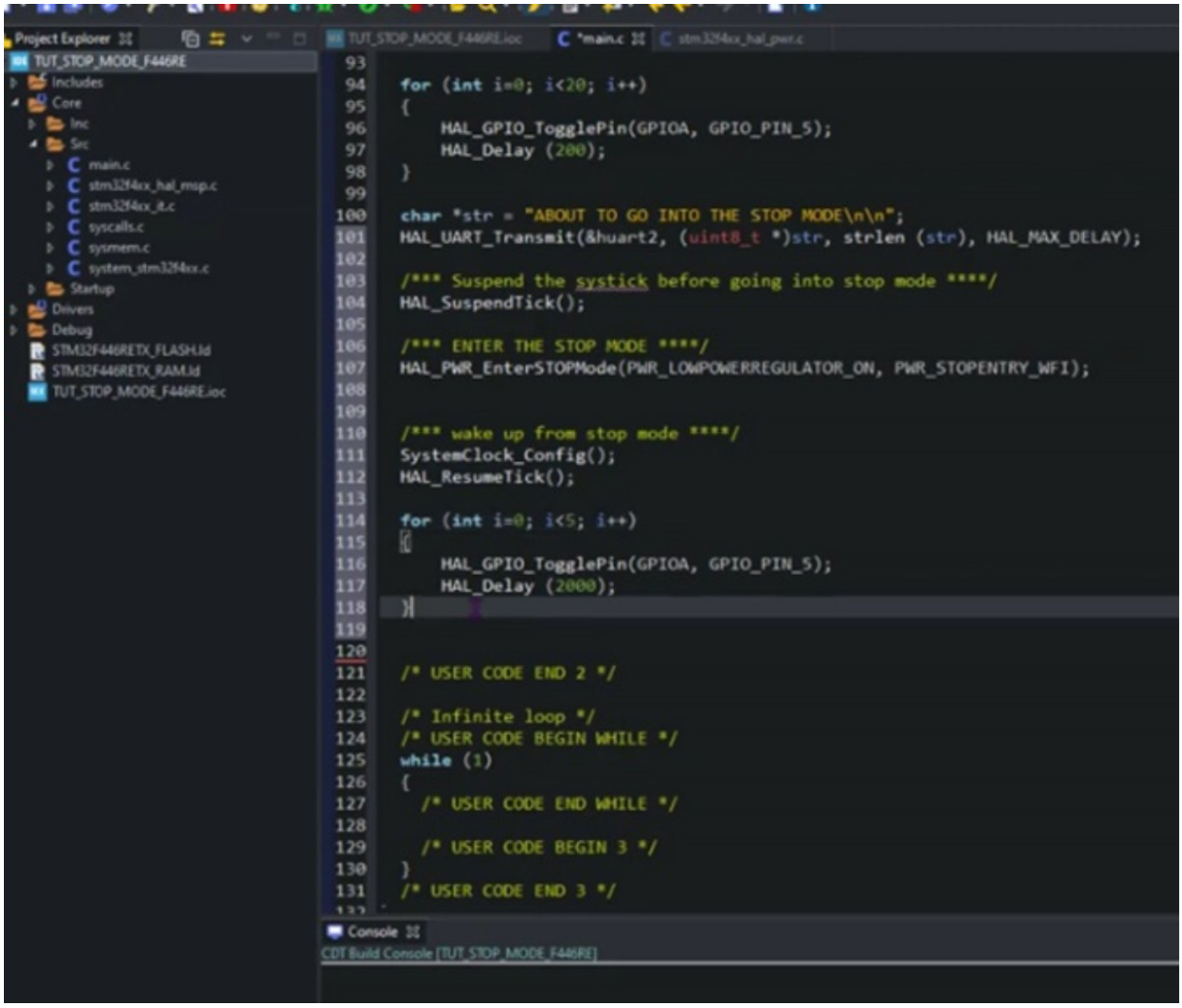
And today we will use the stock mode STOP mode is based on deep sleep mode and it is more aggressive than the sleep mode.

Table 17. Stop operating modes

Voltage Regulator Mode		UDEN[1:0] bits	MRUDS bit	LPUDS bit	LPDS bit	FPDS bit	Wakeup latency
Normal mode	STOP MR (Main Regulator)	-	0	-	0	0	HSI RC startup time
	STOP MR- FPD	-	0	-	0	1	HSI RC startup time + Flash wakeup time from power-down mode
	STOP LP	-	0	0	1	0	HSI RC startup time + regulator wakeup time from LP mode
	STOP LP-FPD	-	-	0	1	1	HSI RC startup time + Flash wakeup time from power-down mode + regulator wakeup time from LP mode
Under-drive Mode	STOP UMR-FPD	3	1	-	0	-	HSI RC startup time + Flash wakeup time from power-down mode + Main regulator wakeup time from under-drive mode + Core logic to nominal mode
	STOP ULP-FPD	3	-	1	1	-	HSI RC startup time + Flash wakeup time from power-down mode + regulator wakeup time from LP under-drive mode + Core logic to nominal mode

Of course the MCU will consume less current in stock mode but wakeup latency also increases in it let's take a look at the datasheet of F 446 r e controller as you can see we have many configurations in stop mode we can stop the main regulator or power down the flash also or stop the low power regulator also I am only going to cover the first one where we are only going to stop the main regulator let's start by creating the project in cube Id first I am using STM 32 F 446 Our eboard give some name to the project and click

Finish in the cube MX I am enabling the external crystal for the input clock pin PA five is connected to the onboard LED I am enabling you to communicate to the computer PC 13 is selected as external interrupt pin now go to GPIO and Vi C tab and turn on the External Interrupt line. Now we will go to the clock setup. I am using external crystal which is eight megahertz H S E means high speed external oscillator I want the MCU to run at 180 megahertz this completes the setup and let's just save it so that the code can be generated this is our main dot c file. Let's start writing our program. First of all I am blinking the LED 20 times at a delay of 200 milliseconds next we will display this string on the serial console before going into the STOP mode. Now just like in the sleep mode here also we need to suspend the tics before going into STOP mode or else the SysTick interrupt will wake the device up and now finally we will enter the STOP mode. How power enter STOP mode takes two parameters let's take a look at this function. The regulator can be configured as if we want to keep the main regulator on or keep the low power regulator on I am choosing the later one for low current consumption. I am choosing the entry mode as Wi Fi so as to wake from sleep using the interrupt After wakeup from the stock mode, first of all, we must reconfigure the system clocks. This is because the main regulator was turned off and we must initialize all the clocks again. Next we will resume the cystic so that we can work with Hol delay.



I am blinking the LED here every two seconds to indicate the wakeup and finally this string will be sent to the console let's build this code now I am using Hercules for the serial console, I will directly run the code as we can't debug in the STOP mode once the board is reset the LED will toggle few times before going into the STOP mode. The string is printed on the console and the controller is in the STOP mode. Now once I press the button, external interrupt gets triggered and MCU will wake up from sleep. You can see the LED blinking every two seconds and finally the string gets printed on the console. If I reset the MCU same thing will happen again. Now I am going to write the external interrupts callback function I am going to resume both of these functions here and this string is to confirm that the MCU is waking up from the external interrupt looks like I

got the warning about the callback function okay, we are good to go now.

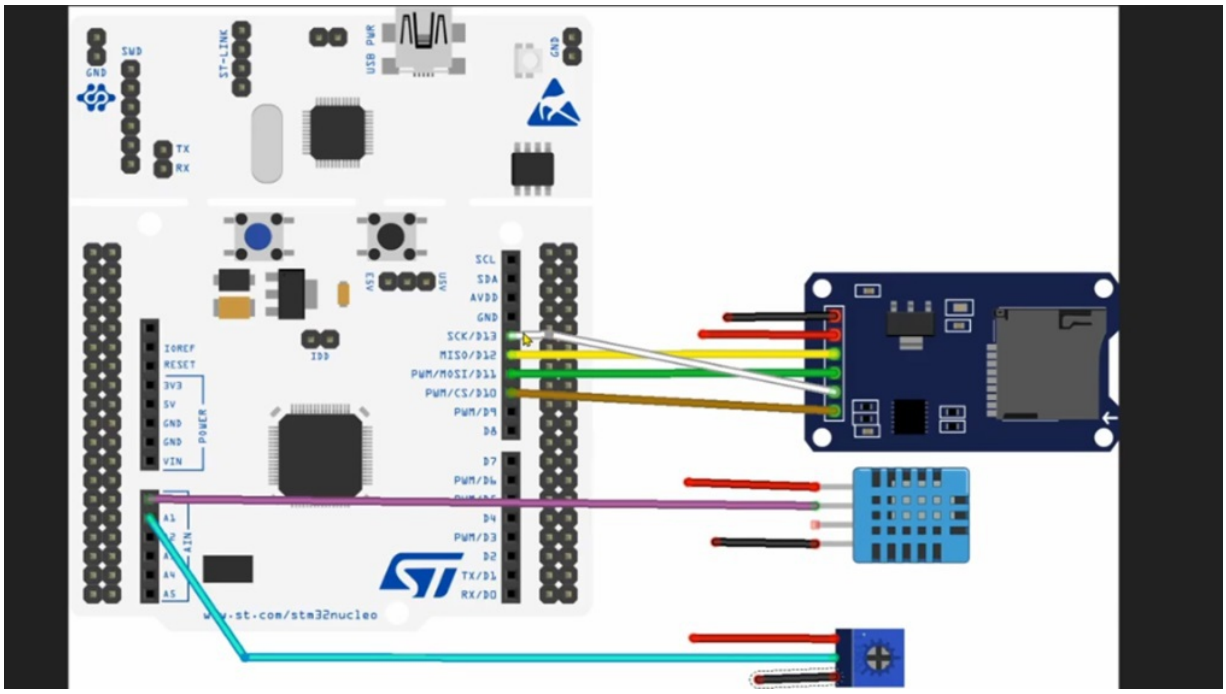
We can also wake up the MCU using the RTC interrupt. This is the vital part of this tutorial as we are going to use it in the standby mode also. First set up the values of a synchronous prescaler divider and synchronous prescaler divider. Let's take a look at the datasheet again. This is the formula to calculate the Espree clock to obtain the one hertz of SP clock the values are 128 and 256 for 32.768 kilohertz clock which are already set by default. So we will leave it as it is. In the wake up section. Select the internal wakeup and make

sure that you turn on the interrupt Here I am not changing anything. We don't care about the current time as it's going to be a periodic wakeup leave the wakeup counter to zero we are going to set it in the code itself in the n vi C tab, do the cross check if all the interrupts are active let's save this and we will go back to our main file here is our RTC initialization function first copy this wakeup function and comment it out from here we will paste it in our main function. Now we need to do the calculation for the counter value I took this calculation part from the examples in the SD folder. This part is self explanatory and I will just show you the calculation I am going to set the wakeup period of 10 seconds this value is 16 and we need to divide it by LSI which in my case is 32 kilohertz. Now we need to divide the 10 seconds by this result and then convert the value to the hex format and finally, feed this value to the counter rest of the functions will remain same I will just modify this string for better indication just like the external interrupt. We are going to write the RTC wakeup callback function. Copy this entire code and paste it here. Modify this string after the mcu wakes up, we must deactivate the periodic wake up or else the callback function will keep executing every 10 seconds. Let's build this now. No errors so we are good to go. Run the program. As you can see the LED blinks few time the string got printed the MCU have entered the STOP mode now. You can time the project and wait for 10 seconds. After 10 seconds the string from RTC callback is printed MCU is awake and LED is blinking every one second. And finally the string in the main function is printed. Let's reset the MCU and this time I will wake it by using the external interrupt pin now even after 10 seconds, the periodic task is not going to run because it got disabled in the main loop after the mcu wakes up Now let's take a look at the last part of this tutorial. That is the sleep on exit function. I did covered it the sleep mode also before entering the STOP mode, just enable asleep on exit these callback functions are the only functions that are going to execute as this is the interrupt operation only. As soon as the control comes out of the interrupt MCU will enter the STOP mode the MCU have entered the STOP mode. We will wait for 10 seconds for the RTC interrupt. It wakes up after executing the interrupt the MCU will go back to the STOP mode. We will wait for another 10 seconds to

see this. We can wake the MCU using the external interrupt also. The RTC wake up his periodic because the control never comes back to the main function and that's why the RTC wake up is never deactivated. I will modify this operation a little if the wakeup is from external interrupt Let's disable the sleep on exit and I am changing this time to five seconds as 10 seconds is too long. Remember, if the control reaches main function, the RTC wakeup will be disabled and rest of this code will work. Let's run this program now. The LED blinks few times string gets printed and the MCU enters the stock mode. We will wait for the RTC periodic wake up now. Five seconds later. The mcu wakes up processes the interrupt and go back to sleep. This will keep happening until we press the button. Now the sleep on exit is disabled so the rest of the functions in the main will execute now. Also note that the RTC periodic wakeup is deactivated as that was one of the instructions in the main function. And if we wait for another five seconds or so nothing happens. This is it guys. I hope you understood the STOP mode. I will cover standby mode in the upcoming project. By the way, the current consumption during STOP mode was around 0.18 milli ampair compared to 1.8 milliamp pair in sleep mode and 36 milli ampair during normal run, you can download the code from the link in the description. Have a nice day.

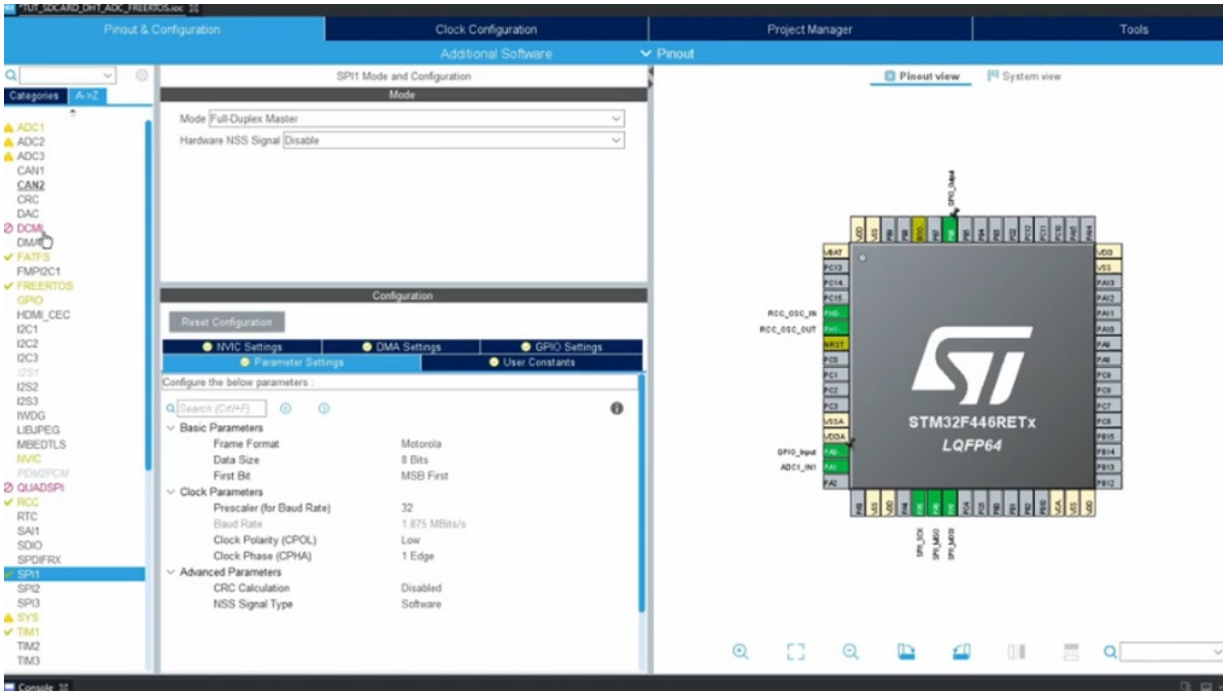
STORE DATA INTO SD CARD FREERTOS STM32 ADC DHT

I will combine them all. This project will cover how to save data in the SD card which will communicate through SPI.



This data will be collected from the potentiometer which is connected via ADC and the DHT 11 temperature sensor and all this will work with free RT O S. So let's start by creating the project in cube Id first I am using STM 32 F 446 R E but you can use for other microcontrollers too. I will also upload the code for STM 32 F 103 controller. Let's see the connection first SD card me so is connected to me so mossy Tomasi clocked a clock and CS pin is PB six in my case d h t 11 data pin is connected to PA zero and potentiometer

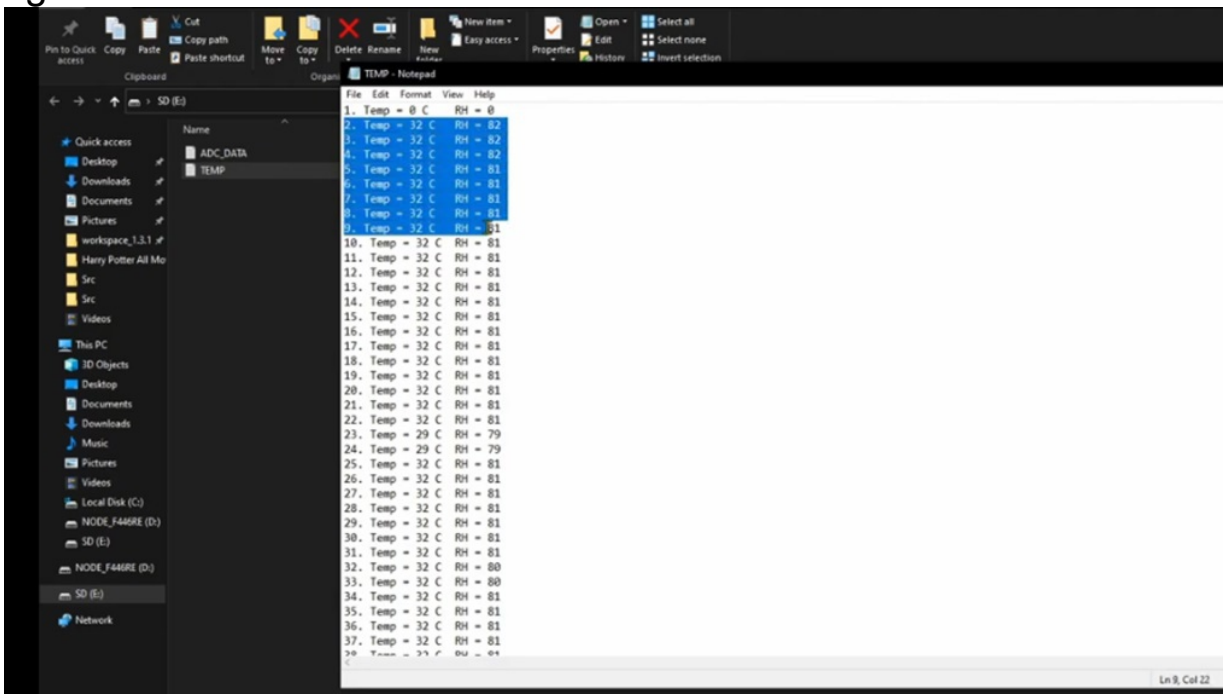
middle pin is connected to PA one all of them are connected to same five volts let's first turn on the ADC channel one for potentiometer. Enable the continuous conversion mode three cycle time is okay for us. Now enable the fat F s for SD card. Leave everything to default. Enable the free RT O S and again leave everything as it is. Here I am selecting the external crystal for the clock enable SPI for connecting the SD card also select the CST pin as output. In my case I am using PB six as the CST pin select time base other than cystic this is because of free R T O 's timer one will be used for the periodic delay. I will set it up later. After setting the clock. Timer seven will be used for microsecond delay for the D H T 11 sensor. Select the UART so that we can see the output on the serial console. Let's set the clock now. I am using external crystal which is of eight megahertz and I want the controller to run at 60 megahertz. Note that both the APB clocks are also at 60 megahertz. Now let's set the timer seven first, keeping prescaler as 60 will divide the APB clock to one megahertz. That means each count in auto reload register will take one microseconds. Set the auto reload register at max value. This is basically the maximum microseconds that we can count up to. Now let's go to the timer one. This time I am setting prescaler as 60,000. This will divide the APB clock to 1000 Hertz. Here, each count in auto reload register will take one millisecond. Setting auto reload register to 2000 means that it will count for 2000 milliseconds, enable the update event and turn on the interrupt. Guys, I hope you understood the difference between the setting of timer one and timer seven. In timer seven. Each count of auto reload register takes one microsecond and inside the code we will not count up to the full AR value. We will just count up to however long a microsecond delay we need. But in timer one, each count takes one millisecond and we are allowing it to count up to the full auto reload register value and that's why it will create a periodic delay of two seconds Let's go to the SP I now divide the clock such that the speed remains somewhat near two megabits per second.



Also set the PA zero as input for the D H T 11 sensor. This completes our setup now just save it to generate the code here is our main dot c file. First of all I will remove the CM sis related functions. So just copy the RT o s functions into the main code and comment out the CM sis I am going to remove all the default task related functions let's include the library files into our project now you can get these After downloading the code from the link in the description include the DHT 11 and file handling dot h files you need to make modifications here if you're using other instances or pins also define your CS pin here you can change the SPI if you are using any other instance of it as instructed here we need to copy this and paste it in the interrupt dot c file and this one should go inside the SysTick handlers interrupt function I am using timer six for the cystic also changed the UART if you are using any other now we need to make modifications in the use of this guy O dot c file. Just watch carefully you can replace this entire file with the one from my code also. Let's write the code now. I am creating three task handlers first. They are for d h t task, SD card task and the ADC task. Create a handler for the semaphore which will be used in the DHT task. Now the task functions where we will write the task codes also define the variables to hold the ADC value and temperature and

humidity values. Now inside the main function, first of all, we will create the files in the SD card. So mount the SD card formatted create the ADC file, create temp file and unmount the card. Now we will We'll create the binary semaphore remember that this semaphore must be given first and we will not give it here. Now, create the tasks, I am creating three tasks here d h t task will be of lowest priority, we will use the semaphore to execute this task SD card will be of highest priority because nothing should preempt it or else the volume could fail and at last stop the scheduler before starting the scheduler, we need to start the timers also start timer seven in normal mode and timer one in the interrupt mode. Now, let's write the task functions in ADC task, we will simply start the ADC poll for the conversion get the value and stop the ADC this task will run every 500 milliseconds. Now, the DHT task here we will acquire the semaphore first as the semaphore will be given every two seconds, we will wait for two and a half seconds for the semaphore. If it doesn't acquire, we will print this string or else the DHT 11 Read function will read the data and save it in the temperature and humidity. Note that we will not release the semaphore here. And finally, we will write the ADC and d h t data to the SD card index variable is used for indexing. Before using s printf function use the malloc for allocating the memory. We will mount the SD card, update the files and unmount the SD card this task will run every second. Now we need to release the semaphore periodically. And you already know that when we give semaphore from an ISR, we need to use the high priority task switching also. This completes everything let's build this code and debug it I have put those variables in the live expression so that we can watch them I will first show you my SD card. There are two files in it. I will delete these files and create new files just to show you that format SD function also works. For now the format function only removes the files and any directory will still be there. I am putting the card back in the module let's run this now. You can see the values here.

There seems to be some error in opening the temp file. Let's take a look at the code again. It's all right here. Okay, I have created directory by mistake. Let's change this to file. Build and debug again.

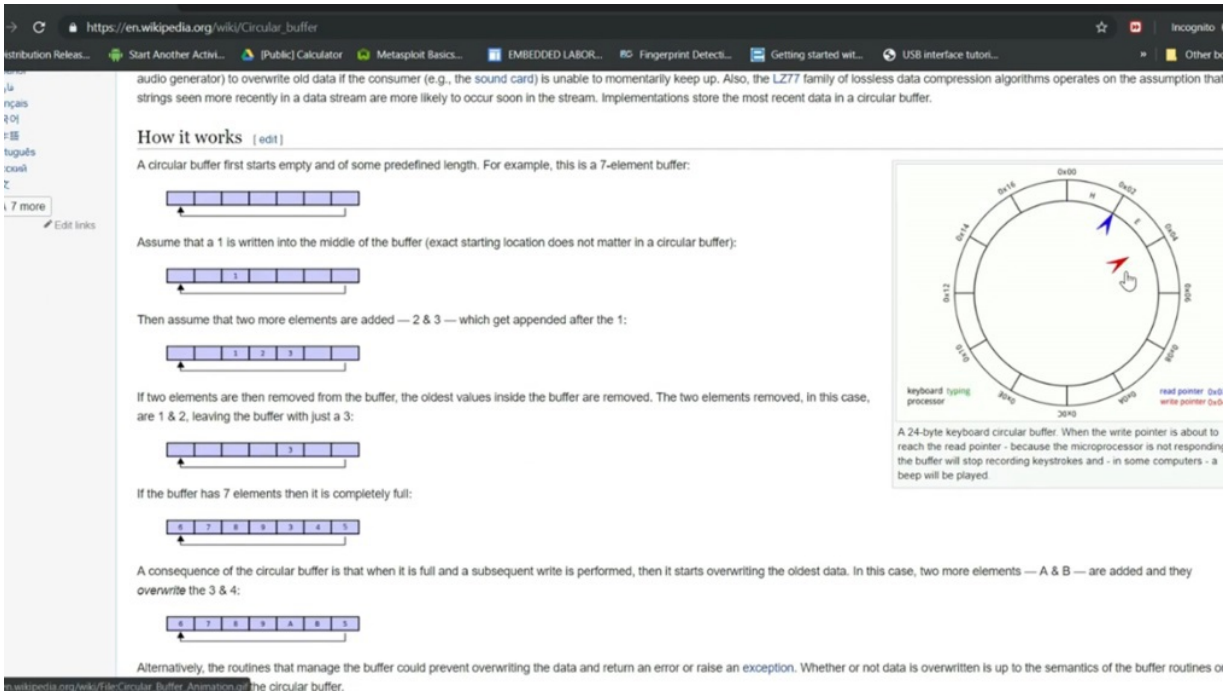


Now the output is perfect the files are updating as expected this whole updating data works every one second take a look at the ADC

value I will reduce it and now I will increase it again I am touching the temperature sensor now you can see the rise in the value okay, let's pause it and it's now the time to check our SD card you can see the two files here. See how the ADC values decreased and they increased again just as I changed it. Same thing with the temperature also. It's pretty much consistent but then increases a bit so the data is saved properly let's see what we get in the output now. Obviously error in everything here. This is because the card is not inserted into the module.

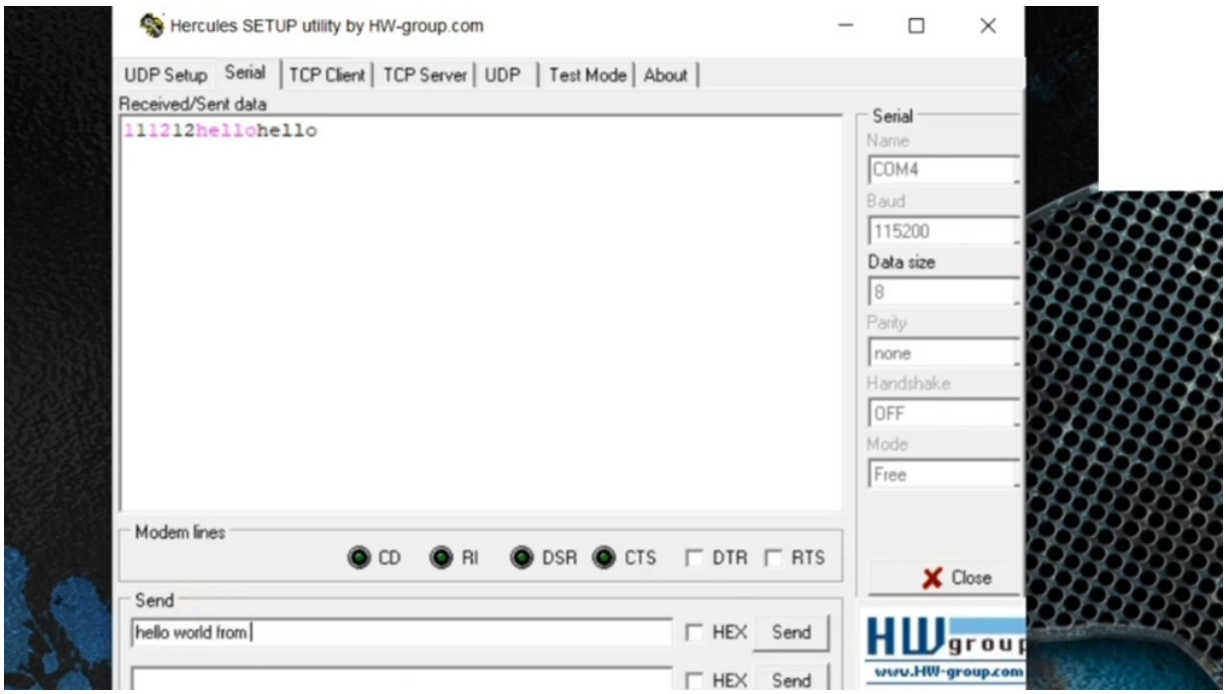
UART RING BUFFER USING HEAD AND TAIL IN STM32 CUBEIDE

The latest one of them was circular buffer using DMA interrupt and idle line detection. Circular buffer was quite good, but it is very complex. And sometimes we might miss the data. Today in this project, I will use yet another but more effective method of data transfer using UART. I have implemented the Arduino like head and tail methods to receive and transmit data. I am using STM 32 Cube IDE and I will recommend that you use it too. So let's start by creating our project in the IDE. I am keeping the baud rate at 11 5200 and make sure that you enable the interrupt I am keeping the clock at max frequency. That's it for the setup. Now let's build our code before starting, we need to copy some library files into the code. include the header file, and let's check if there is any error let me just change it for f1 series and done let's take a look at some functions.

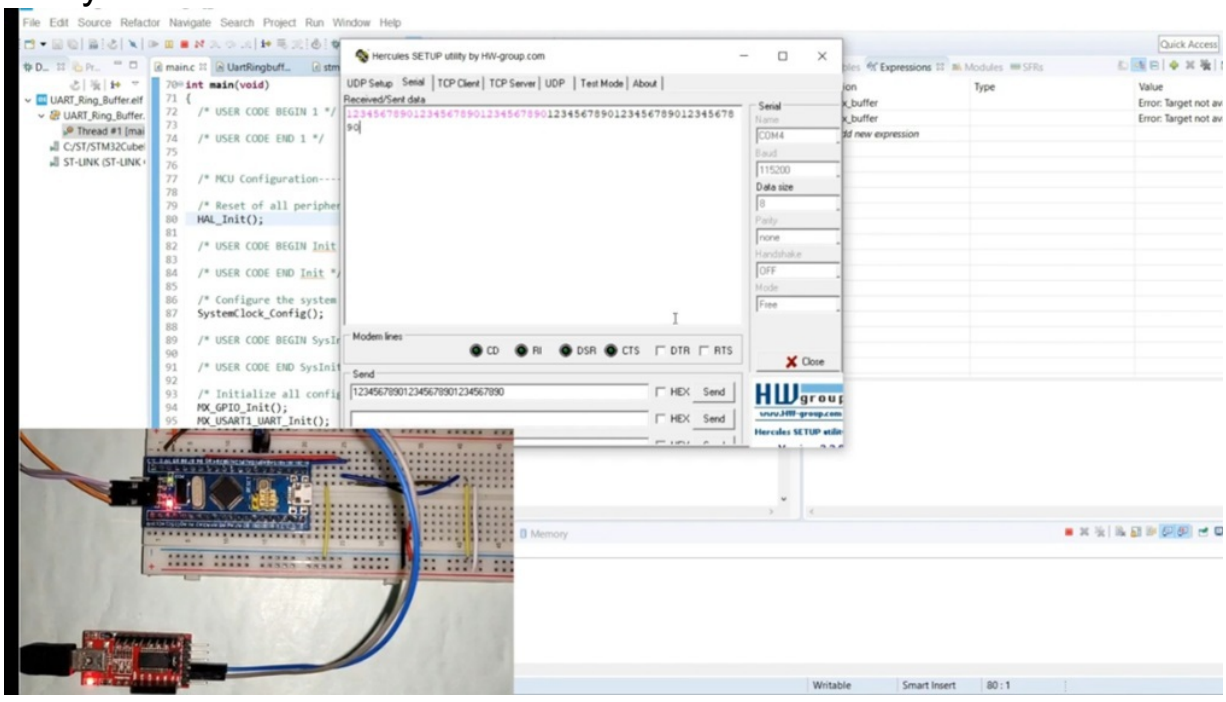


Now you can change the size of the buffer here. If you are going to receive large data at once, make it big for small continuous data transfer, keep it low, make sure you define your UART handle type here and we need to put this one in the interrupt file. This is the function definition for the ISR that we are going to use instead of the default one. And next we need to route the ISR to the one written by us and comment out the default one note here that every time you regenerate the code you have to comment it out again Okay guys, let's talk about the functions now you are to read is going to read the data that is in the RX buffer. It reads one character at a time. After successfully reading it will increment the tail by one and returns the character in the decimal format. The head counts in the RX buffer will increment each time there is a new character received in the buffer. This process is interrupt driven. To make sure that the tail is always catching up with the head we have to read data as soon as it arrives. Or else there will be no more space for the new data and any incoming data will be lost. If you just Google the ring buffer you can check out the Wikipedia page as you can see here head increments when we type in some data the tailing promotes when we read the data if the first two bytes are already read, the head can overwrite the data. But think of a situation that we haven't read these

first two bytes then the process will stop there and we will lose the last two bytes of the data. That's why I said in the beginning that if you are expecting large data at once, make sure the buffer size is big or else the head will reach the end and any further data will be lost. You are tried function I will write a character to the PX buffer and increment the head in the TX buffer sending data to the UART is again an interrupt driven process and the tail will be incremented after the data is sent, you are sent string will send the entire string instead of single character, you offprint base prints the number in different formats such as hex decimal, et cetera. His data available checks whether there is any data available to be read in the RX buffer, it returns one on success wait for checks for the provided string within the incoming data it returns the position of the string within the buffer initialize the ring buffer first inside the while loop first we should check the data and if it is available, we will read it and then write it to the URL again let's build the code and run it. As you can see that the baud rate is set to 11 Five to double zero sending single character and received it to pink color is what we are sending and black color is the one that we received sending two characters Let's send the string hello send even bigger string make sure this string size is less than the size of the RX buffer yup received it too.



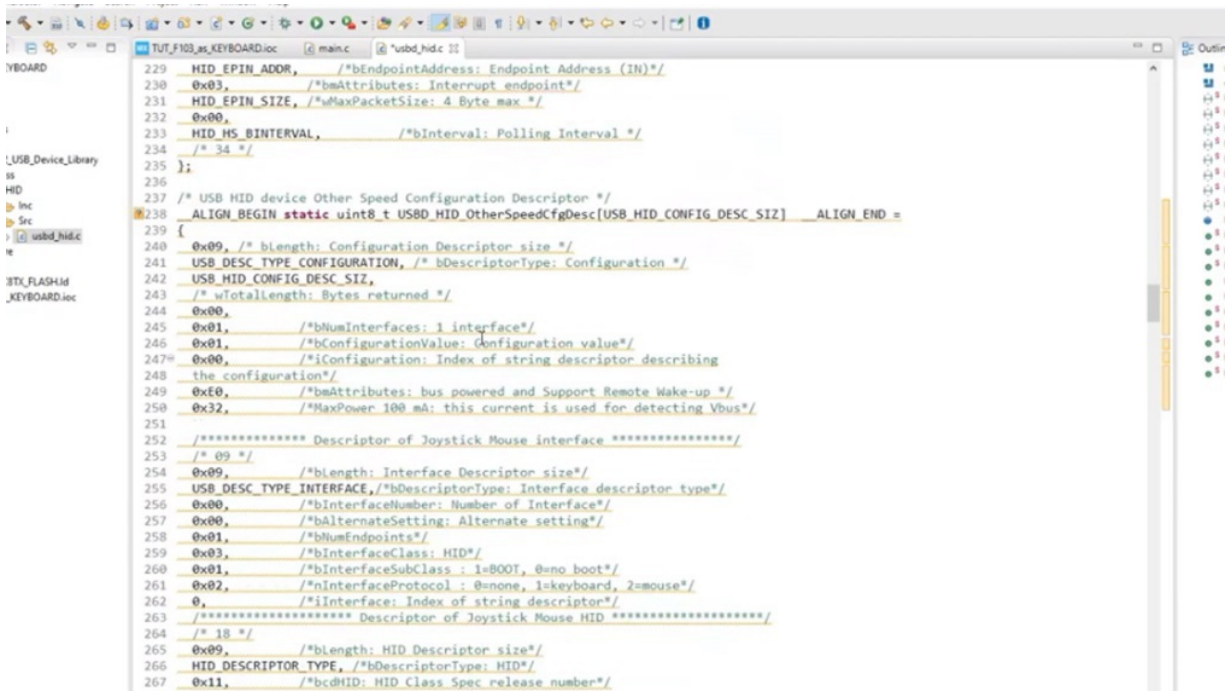
So the UAV is working irrespective of the length of the incoming data. I will come back to this in a while let's check the wait for function now. Here I want to know the position of the Hello string in the buffer and if the position is valid, it will send it to the UART. Build it and run if I send some random string nothing get printed now I will send hello and see that some number is printed on the terminal. If you take a look at the RX buffer, you will Notice that this is the position where the string hello ended let's try with some other string this time I will send Hal along with other characters nothing on the terminal as expected now same string but complete hello and you see the position is printed let's take a look at the buffer again and yes it is the position where Hello ended next I want to demonstrate that this is not a blocking mode function so for that I have to use LED blinking. So I will receive and print data and also the LED will blink every 100 milliseconds. Luck Let's send some data now.



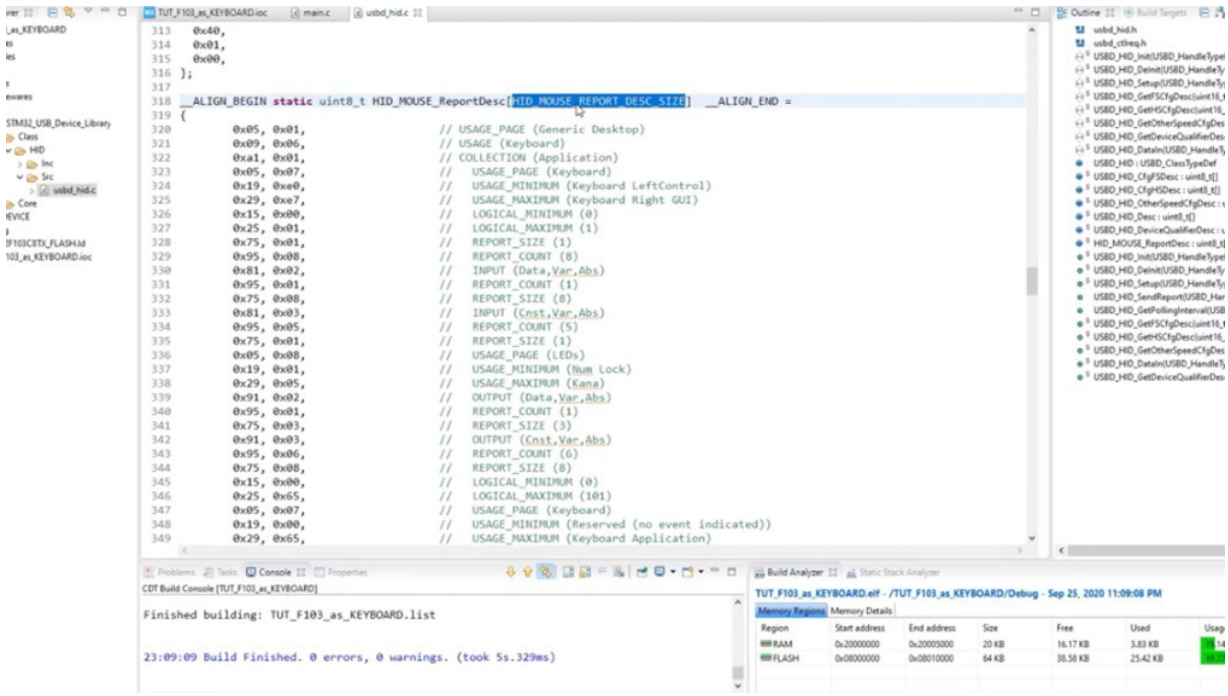
As you can see that le V is blinking as it should also the data is being written every 100 milliseconds let's change this code a little and try to print all the data at once. Now, the data is being printed all at once.

USE STM32 AS A KEYBOARD F103C8 USB DEVICE HID

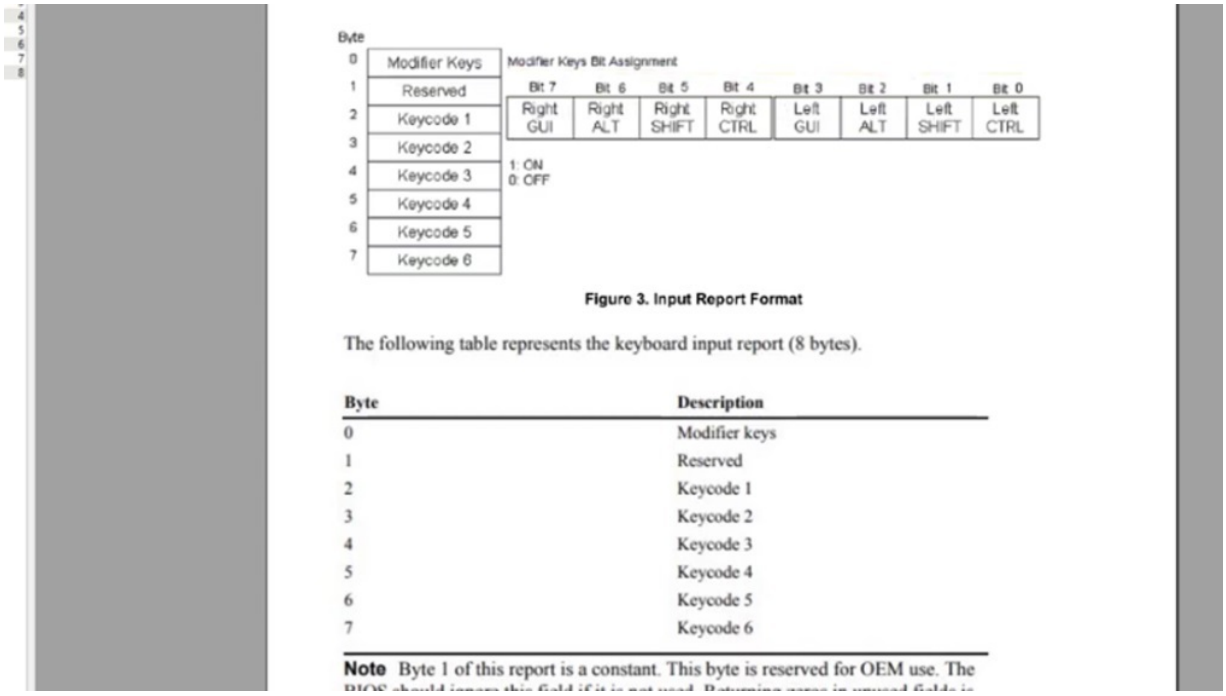
I will show you guys how can we use it as a keyboard. This project is divided into two halves. First half will show some simple usage of keyboard and in the second half, I will show some application using an actual keypad. So let's start by creating the project in cube Id first. I am fast forwarding this part as it is a very common thing at this point. Now the basic setup is complete select the USB and choose device F s leave everything to default here. Now in the USB device, select the classes H ID class. You can leave everything default here too, but I am going to make this small change. This way computer will recognize the device's STM 32 keyboard. That's all here. Let's go to the clock setup now. It have sorted out everything, but I will run the MCU at 72 megahertz. Let's click Save to generate the project. Here is our main file. This theme looks a bit ugly. So I am going to change it. It looks better now. First of all, open the USB d h ID dot c file we need to make some changes there. Scroll down to H AI D configuration descriptor here is the line we need to change. As you can see two is for the mouse one for the keyboard. By default, the code will always generate the setup for the mouse.



So we need to change this to one to use it as a keyboard. Make sure that you make the changes in fs configuration only and not in any other configurations. Now scroll down to mouse report descriptor this here is the descriptor for the mouse and we need to change it with the keyboard descriptor. I have the keyboard descriptor downloaded right here. You can Google it or I will leave the link to it in the description. Copy these descriptors and replace them with the mouse descriptors. We will leave the function name to mouse itself because other functions might be linked with it and we don't want to mess things up.



Also note that the size here is 63. Looks like I need to build it first Okay, now I can access it. Change the size to 63. That's all for this setup. Remember that whenever you generate the project from cube MX, you need to do this all over because cube MX will generate for the mouse again. Let's go back to our main file. First of all include the USB d h ID dot h file We need to declare the USB device handling our main file make sure you declare it as an external type now before we create a structure for the keyboard data, let's see the keyboard protocol. Here in case of keyboard, we need to send eight bytes of data.



Byte	
0	Modifier Keys
1	Reserved
2	Keycode 1
3	Keycode 2
4	Keycode 3
5	Keycode 4
6	Keycode 5
7	Keycode 6

Modifier Keys Bit Assignment

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Right GUI	Right ALT	Right SHIFT	Right CTRL	Left GUI	Left ALT	Left SHIFT	Left CTRL

1: ON
0: OFF

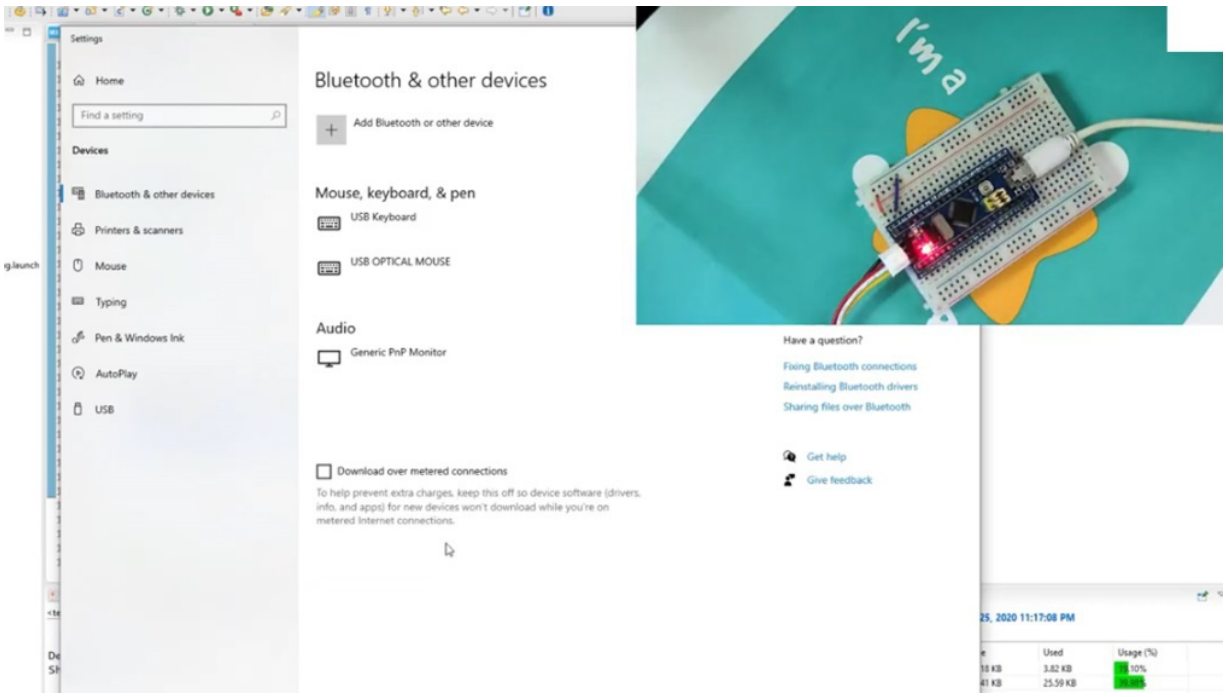
Figure 3. Input Report Format

The following table represents the keyboard input report (8 bytes).

Byte	Description
0	Modifier keys
1	Reserved
2	Keycode 1
3	Keycode 2
4	Keycode 3
5	Keycode 4
6	Keycode 5
7	Keycode 6

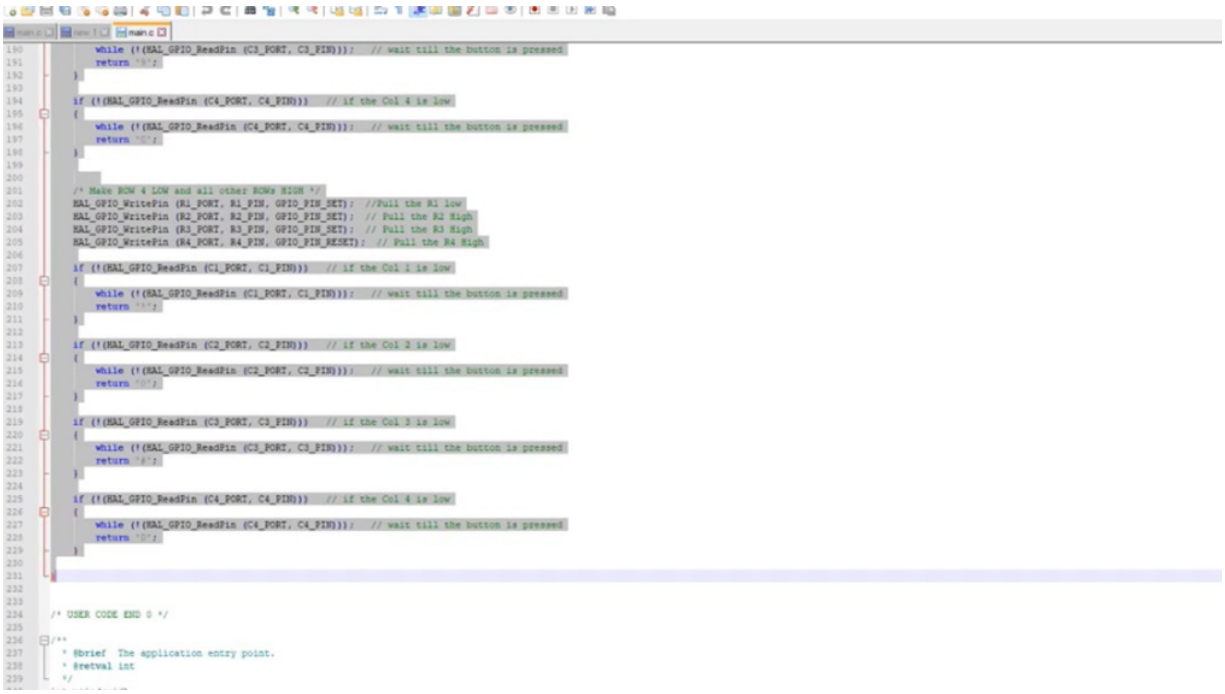
Note Byte 1 of this report is a constant. This byte is reserved for OEM use. The

This data contains the modifier keys, second byte is reserved, then key code 12345, and six. The modifier keys can be all these keys listed here let's create a structure to store these eight bytes. I will call it keyboard H ID let's initialize the structure with all zeros looks like one zero is missing. Now let's see the key codes here is the list of all the key codes that you can use like zero cross 04, for a five for B, six for C etc it's a huge list and I will leave the link for this PDF in the description I will just send an A for now let's build it once before going forward looks like I spelled keyboard wrong here now everything is fine. I will send the value to the key code one I am sending an A here. This basically means that the A is pressed. And now send the report wait for 50 milliseconds and now send the key code one zero to indicate that the key was released and I want this process to repeat every one second. Let's build this we have some warning about the structure but that's okay. Let's flush it to the MCU now let me open the device list also.



Now I am connecting the USB seems like some descriptor issue. I will try again. You can see here the device has been recognized as the STM 32 keyboard and it's sending an A every one second. You can see wherever I click it's printing every second just like as if it is a keyboard. Things are working all right. Now let's try some more combinations. There This time I am going to use the left shift modifier. This will print the upper case letters. To use the left shift the bid one must be one, so the modifier value will be zero cross 02. Let's write the zero cross zero to to the modifier key I will leave the key code one to print a but this time it will print up a case A. As we press the Left Shift, we need to release it also let's use one more key code we can use six key codes at the same time so the keyboard can print six keys at the same time I will just print two keys together I am sending B also let's build now and flash the keyboard is connected and it's printing uppercase A and B at the same time. So everything is working perfect. Like I said, we can send six key codes of the same time so it can print six letters or numbers at the same time. This is it for the basic tutorial. Next part of this project will cover an actual application of keyboard and here I am going to use a four by four keypad as a keyboard. I have already covered how to interface the keypad with STM 32 You can check out the project on

the top right corner. I am going to use the same code for this tutorial also. First of all I need to add pins for the keypad select four input pins for the columns and four output pins for the rows use the pull up resistance for the input pins. I have already covered all this in the keypad project. Watch that if you don't understand the Setup, click Save to generate the new project again we need to modify the H ID dot c file again I will fast forward this part of the project so this is the main file from that keypad code I am going to use these functions as it is. In the while loop, we will read the key which was pressed on the keypad. If the key pressed is a one we need to send the key code for one if we look in the PDF, the hex value for one is zero cross one E.

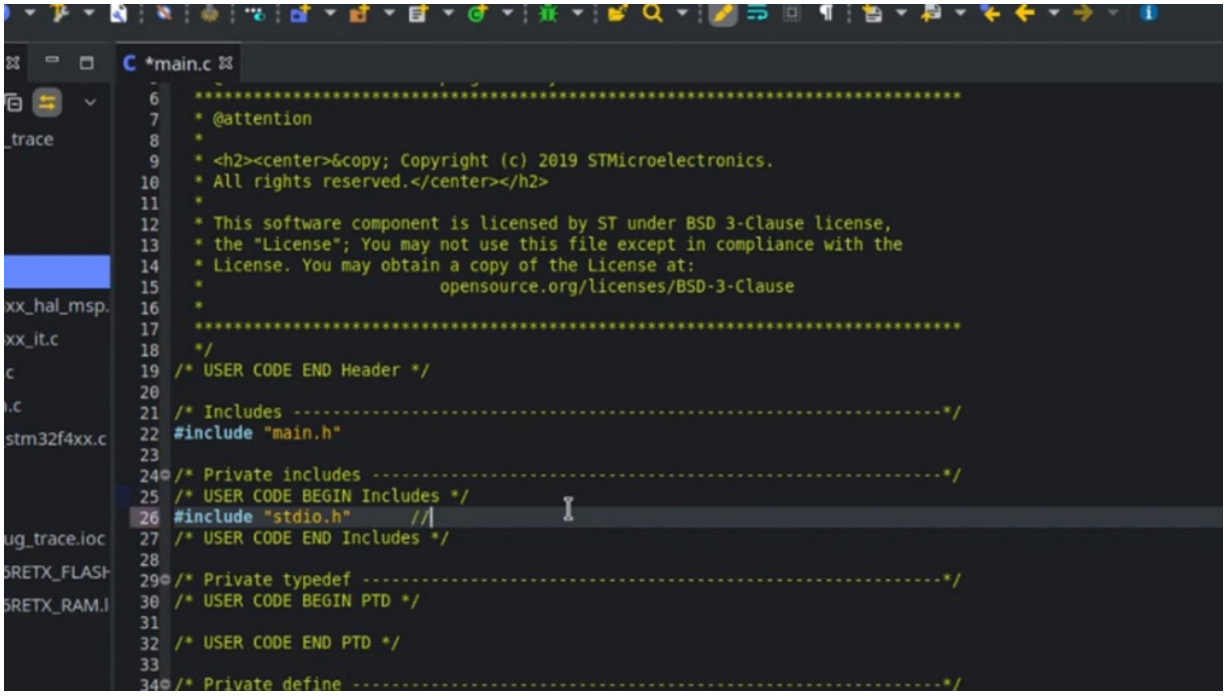


So we will put that in the key code one. Similarly write the values for other numbers to here is the final code for numbers. Now, if the key pressed is capital A we need to find Press the shift and then press a similarly, we will write for other letters also because this keypad have A, B, C and D, so we need to write for them too we also need to write for the star and for the hash star is basically shift pressed with the number eight and hashes shift with the number three. That's all now we need to send this information wait for 50 milliseconds send the key released report also and I am going to comment out this

code let's build it we don't have any errors. During testing, I found out that the key value is always zero cross 01 So I am adding this part that the keyboard part will only work if the value is not zero crosses 01 This is not that necessary as we do have conditions for the keys pressed let's flash it to the controller now I have attached the keypad this time the keyboard gets detected on connecting let's try pressing these keys now you can see the values are printing just all right and we can type them anywhere we want.

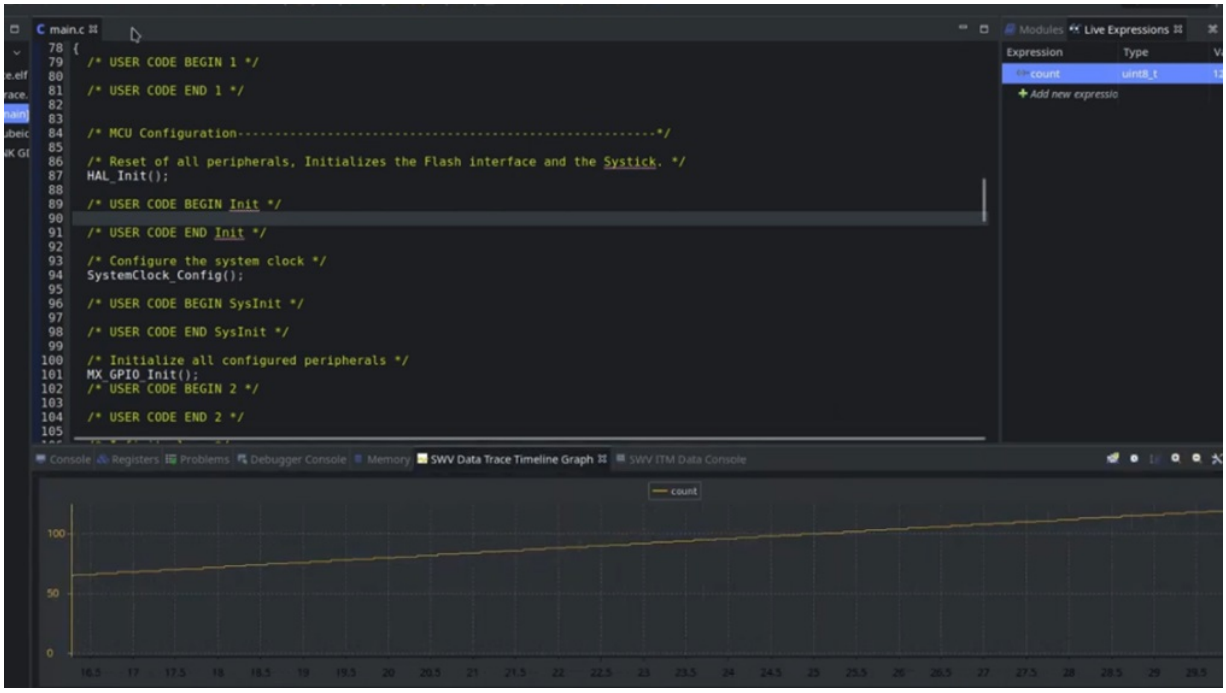
USING PRINTF DEBUGGING SWV TRACE IN CUBEIDE ITM SWV

I am going to show you guys how to use live expressions SW V trace and printf debugging in cube ID using of course, STM 32. I am using STM 32 F 446 r e controller although it should work with others too if you follow the procedure properly. So, let's start the ID and create our project in cube MX select the micro controller here I am going to name it as printf debug trace click finish and wait for the project to open. First thing I am going to do is set the clock to the external crystal This is it pretty much we don't need any additional setup here I am going to set the onboard LED as output you don't have to do this. Now, let's move to the clock section I have external crystal have eight mega hertz and I am going to configure it to give me a frequency of 180 mega hertz you can configure to any clock you want so everything looks ok go to File and click Save this will generate the project Next, open the main dot c file here first of all we need to include stdio dot h header file for the printf to work.



Next, we need to overwrite the write function with the following code print F rely on write function and we are modifying it to send the data to the I T m. So all the printf commands will forward to the ITM. I am declaring a variable called count which I will increment and then trace it on s WV trace let's write the main function now I will toggle the onboard LED first then increments the count variable then print the value of the count variable on the console. And give a small delay for all these functions. compile the code and hit the debug button choose STM 32 application in the debugger tab, make sure the debug mode is gdb server, scroll down and enable serial wire viewer in the core clock, typing your H CL K frequency. Mine is set to 180 megahertz that I set up in the clock configuration. Also make sure that the live expressions is enabled. Hit OK and wait for the debugger to launch. In the debugger, we need to enable the monitoring First, go to the Window Show View and enable the SW V trace and SW V ITM console click on any of these view go to setting for the trace we need to input the variable name comparator one is for the ITM to print the printf output make sure you select the ITM simulus port zero press the red button to enable the record type the variable in the live expression that you want to monitor and finally start to debug you can see all the features working properly the

delay I declared was very less let me just increase that to one second.



Let's launch the debug session again. This time I will not select the trace. So we will only monitor the print f output Sorry, I forgot to turn on the record button we can see the printf output now and there is no trace as expected. We can go to Settings enable the trace, reset the controller and start and it is back on similarly, if you only want to use the trace, just remove the other from the setting option and you can see that only trace is enabled now.