TH244A001_UserGuide

TH244A001_UserGuide

Version: 0.40

2024/08 version 0.40

1. Revision

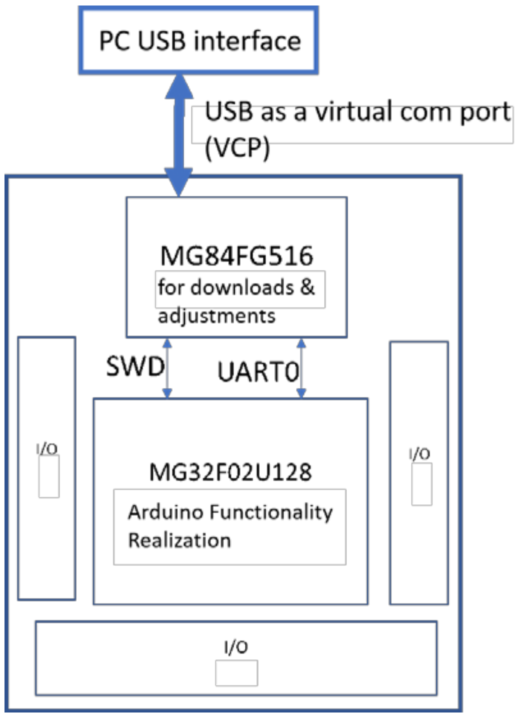
1. 20230712 Initial release V0.20
2. 20231217 Development kit 2.0.0 released, modified to support 36MHz mains frequency, 18MHz SPI clock and other descriptions.
3. 20240229 Modification of some expressions and addition of more graphic material
4. 20240514 update to V0.30

   Due to the release of 2.2.0, User Guide has added the use of SLEEP/IWDT, see SLEEP/STOP Mode and IWDT for details.
5. 20240618 Released version 2.3.0 of the development kit. Instruction manual Updated to V0.40 with several textual representations and image changes; Add library files that support timer interrupts Timer One and Mstimer2; Add library files supporting EEPROM (built-in FLASH to simulate maximum 8K EEPROM operation); Add the library file which supports digital tube (TM1637Display/ShiftRegister74HC595); Add PWM can set the frequency range from 300Hz~5000Hz, add support to low frequency 1Hz; Add serial port disable function; Modify the description text of SLEEP/IWDT section;
6. 20240830 Update TH244A to TH244A001.

2. Development Board Overview

Development board TH244A001, compatible with UNO R3 interface and provides more available GPIO and communication interfaces. Combine the hardware interfaces of Arduino UNO R3 and Arduino MEGA2560. The MCU is upgraded from 8 bit MCU of UNO/2560 to 32 bit Cortex-M0, and the MCU has USB2.0 interface, which can be acted as a keyboard or mouse.

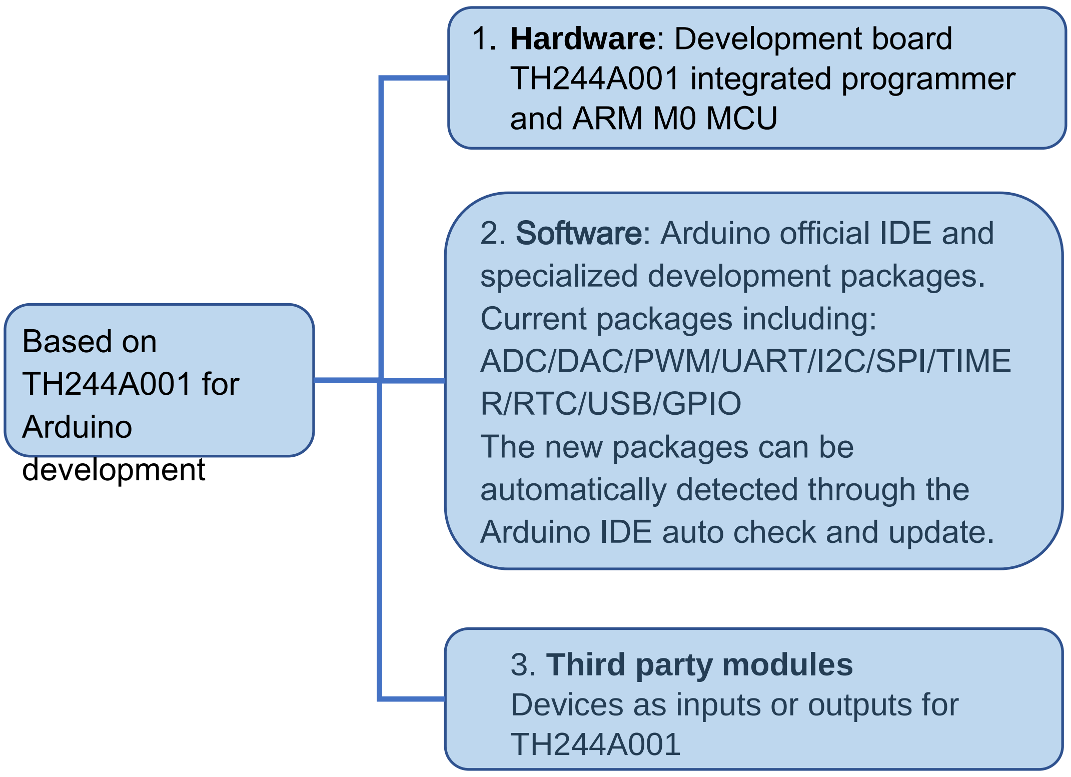
Development board TH244A001 is based on Megawin ARM Cortex-M0 chip MG32F02U128AD64. TH244A001 built-in downloader MLink, users do not need to buy other downloader.


PC USB interface
USB as a virtual com port (VCP)
MG84FG516
for downloads & adjustments
SWD
UART0
I/O
MG32F02U128
Arduino Functionality Realization
I/O
I/O

TH244A001 hardware architecture

In addition to support common MCU development environments, such as Keil. In order to support the open source Arduino IDE development environment, a development kit is specially designed.

Only by installing the TH244A001 corresponding support package, you can use the Arduino IDE to develop the programs, compile and download program.


Based on TH244A001 for Arduino development
1. **Hardware**: Development board TH244A001 integrated programmer and ARM M0 MCU
2. **Software**: Arduino official IDE and specialized development packages. Current packages including: ADC/DAC/PWM/UART/I2C/SPI/TIMER/RTC/USB/GPIO The new packages can be automatically detected through the Arduino IDE auto check and update.
3. **Third party modules** Devices as inputs or outputs for TH244A001

Hardware and software composition based on TH244A001 development

The download and update method of the development kit please refer to **TH244A001_UserManual**- **the installation and update of the development kit of the third-party development board**.

3. Specification of development board

Board name	Operate voltage	Frequency	Digital IO	Analog ADC	Analog output DAC
Megawin TH244A001	5V /3.3V optional	The clock source is 12MHz MCU frequency 36MHz	**47**	**16** 8bit 10bit 12bit (Default)	**1** 12bit

PWM	UART	SPI	IIC	USB DEVICE	Programming interface
x7 8bit default to 1KHz, adjustable from 1Hz~5KHz	**x7** Transmit/receive buffer 64Bytes Default 9600bps Macro definitions can be used to shut down specified ports to conserve SRAM	**x1(Master)** Supports up to 18MHz clock	**x2(Master/Slave)** Supports setting the clock to 100KHz; 400KHz; 1MHz	**x1** USB2 as USB-HID device: Mouse, Keyboard	USB1 uploads programs via MLink MG84FG516

- MG32F02U128AD64 (ARM 32 bit Cortex™-M0 CPU);
- The clock source is built-in IHRCO (built-in high frequency RC oscillator) 12MHz;
- IWDT Clock Source: Embedded ILRCO (Built-in high frequency RC oscillator) 32KHz;
- MCU max frequency up to 36MHz.
- Development board has 47 digital I/O pins (0~46);

- Pin 46 is connected to the USER KEY on the development board through a pull-up resistor, which is convenient for the design and testing of the keypad
- 7 channels PWM output;
  - ◆ 8 bit Duty resolution which can be set from 0 to 255 corresponding to 0 to 100%;
  - ◆ Default frequency is 1000Hz;
  - ◆ Provides a function to quickly modify the PWM frequency: analogWriteFrequency(PWM_FRQ_XX，Fre),
    Frequency **Fre** setting range 1Hz~5000Hz;
  - ◆ 7 channels PWM can be set independently in 3 groups (see PWM section for details);
  - ◆ 7 channels PWM duty can be set independently;
- 2-way can be set up for PWM output by using Timer One, a timer library, which supports PWM output on pin 42/43, see Timer One for details.
- 16 analog input channels ADC. Using analogRead() to get ADC values for analog voltages.
  - ■ Default ADC resolution is 12bit, ADC conversion range 0~4095;

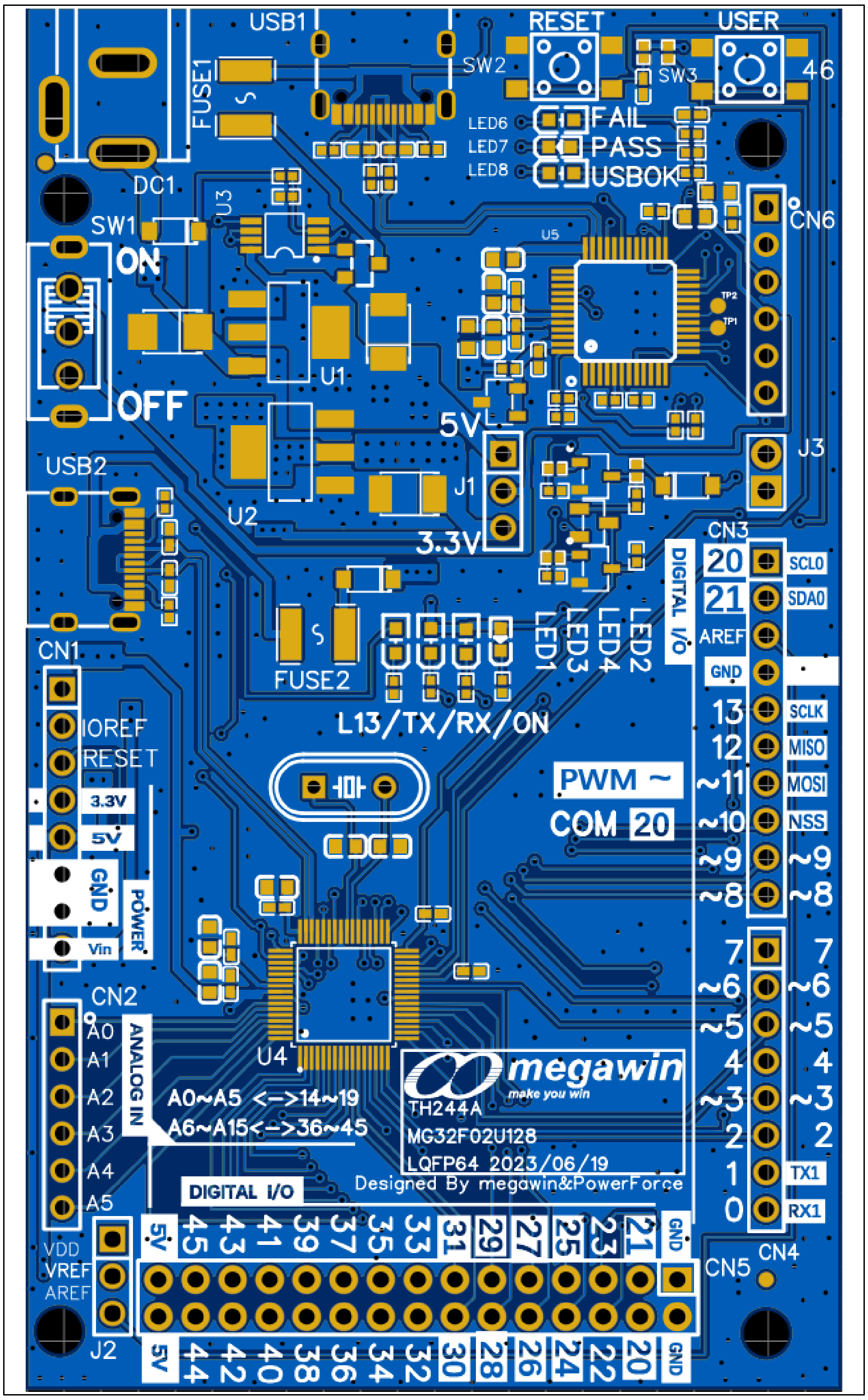
  - ADC resolution can be set to 8（0-255）,10（0-1023）,12（0-4095）by using analogReadResolution(ref)

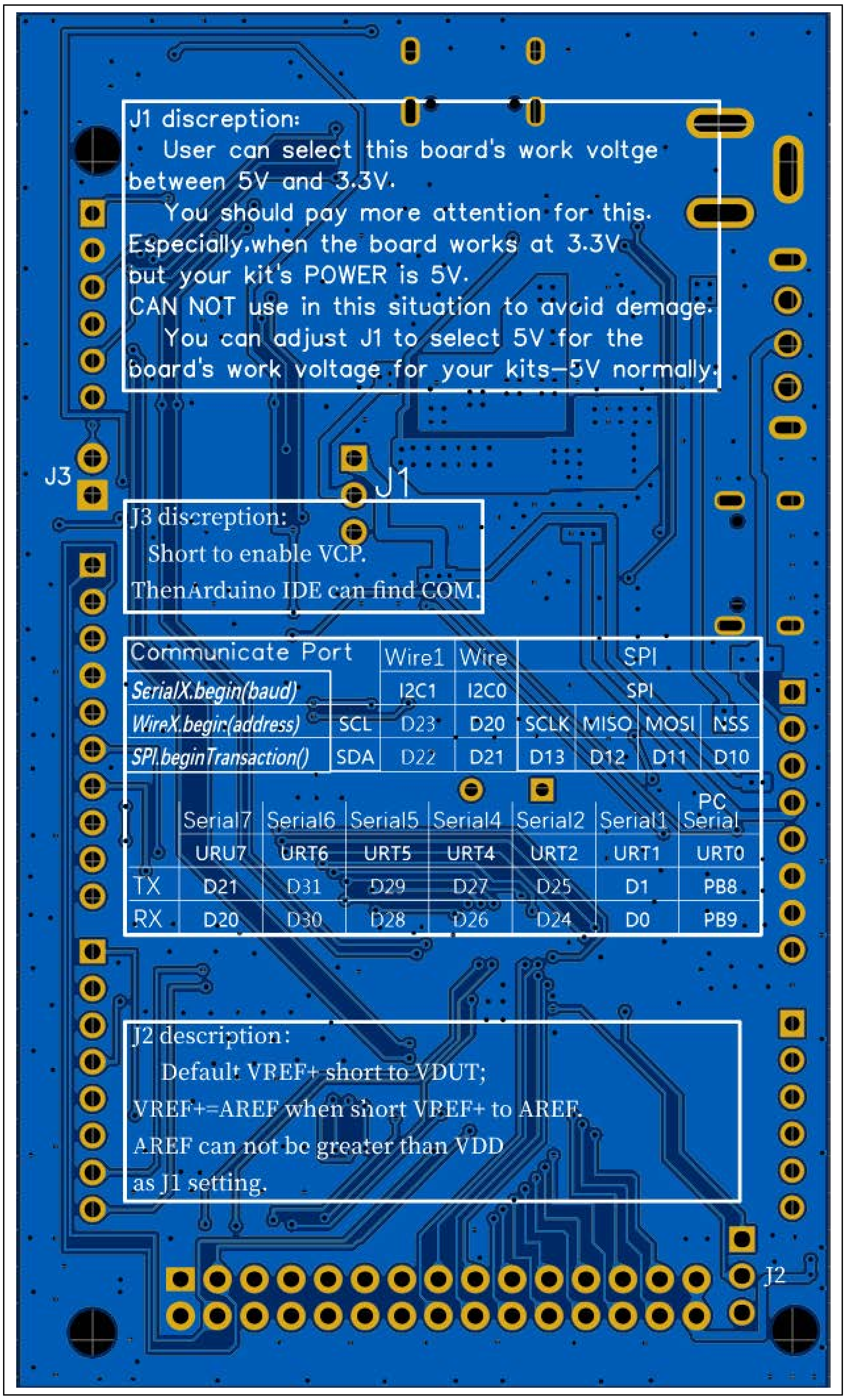
  Notice: ADC Value needs conversion formula to get actual voltage, need re-calibration.

- 1 12bit DAC channel, output from development board 21 pin，using analogWrite(21，Value) can output from 0~VDD linear voltage; Value range 0~4095；
- 7 channels UART maximum buadrate 115200bps
  - UART0 is designed for downloading programs or serial port monitors;
  - UART1/2/4/5/6/7 for debugging or communicating with other devices (UART7 is multiplexed with IIC0)
  - The development kit initializes all serial ports Serial/Serial1/2/4/5/6/7 by default, and the default speed is 9600bps.
  - In order to save the SRAM occupied by the initialized serial ports, the development kit supports disabling the specified serial ports through macro definitions, see the UART
- 2 channels IIC (IIC0 is multiplexed with UART7); SCL is 100 KHz by default. Can be set to 400KHz,1MHz
- 1 channel SPI, maximum speed is 18Mbps;

- Any I/O pin or RST pin to GND voltage: -0.5V~VDD+0.5V;
- 128KB Flash for code and data; The maximum 8K bytes can be simulated as EEPROM through the use of EEPROM library. in order to save SRAM, the development kit initializes 512 bytes EEPROM capacity by default, and the EEPROM size can be modified by macro definition, see EEPROM for details;
- 16KB bytes SRAM
- USB2.0 Full speed, ,Support HID, can acted as USB-Mouse or USB-Keyboard, can using Joystick for USB-Mouse;
- The operation voltage of the development board can be selected to 5V or 3.3V through the 2.54mm 2pin Jumper J1;
- Power supply can using DC JACK，VIN or USB1/2;
    - When using DC JACK supply, Vin port can be used as output;
- 3.3V output current limit
    - 50mA MAX for DC JACK/VIN input;
    - 300mA MAX for USB input;
- 5V output current limit
    - The DC input JACK/VIN, 50 ma MAX.
    - 300mA MAX for USB input;

TH244A001 PCB placement are as follows:





J1 discreption:

User can select this board's work voltge between 5V and 3.3V.

You should pay more attention for this. Especially,when the board works at 3.3V but your kit's POWER is 5V. CAN NOT use in this situation to avoid demage.

You can adjust J1 to select 5V for the board's work voltage for your kits—5V normally.

J3 discreption:

Short to enable VCP.

ThenArduino IDE can find COM.

Communicate Port		Wire1	Wire	SPI			
SerialX.begin(baud)		I2C1	I2C0	SPI			
WireX.begin(address)	SCL	D23	D20	SCLK	MISO	MOSI	NSS
SPI.beginTransaction()	SDA	D22	D21	D13	D12	D11	D10

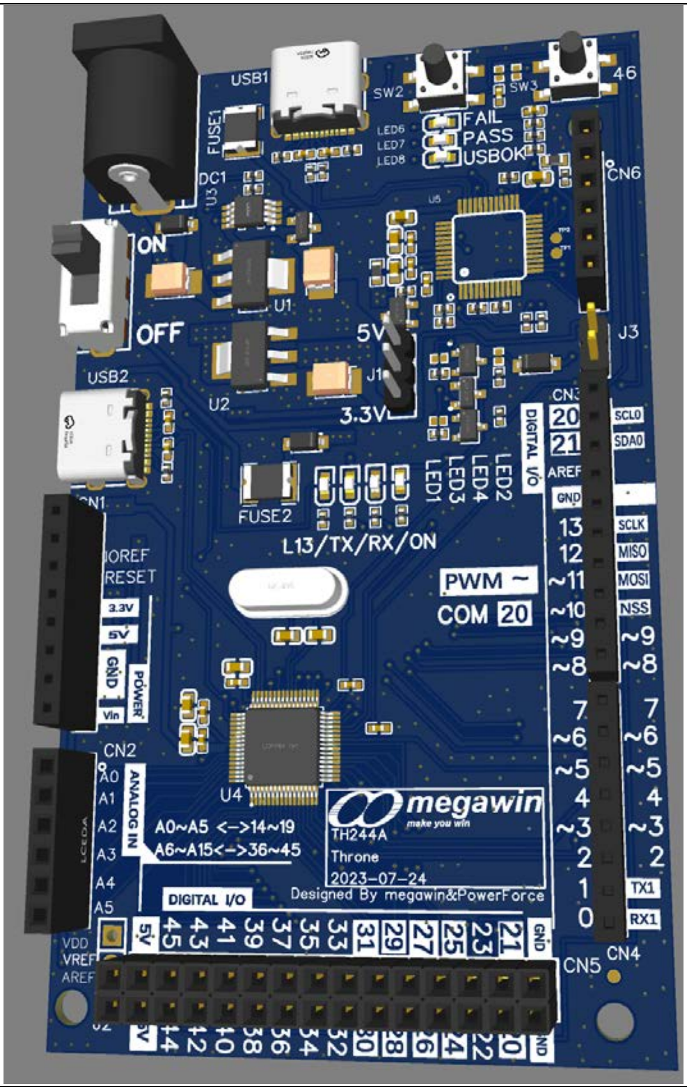
	Serial7	Serial6	Serial5	Serial4	Serial2	Serial1	PC Serial
	URU7	URT6	URT5	URT4	URT2	URT1	URT0
TX	D21	D31	D29	D27	D25	D1	PB8
RX	D20	D30	D28	D26	D24	D0	PB9

J2 description:

Default VREF+ short to VDUT;

VREF+=AREF when short VREF+ to AREF.

AREF can not be greater than VDD as J1 setting.



Power supply and J1, J2, J3

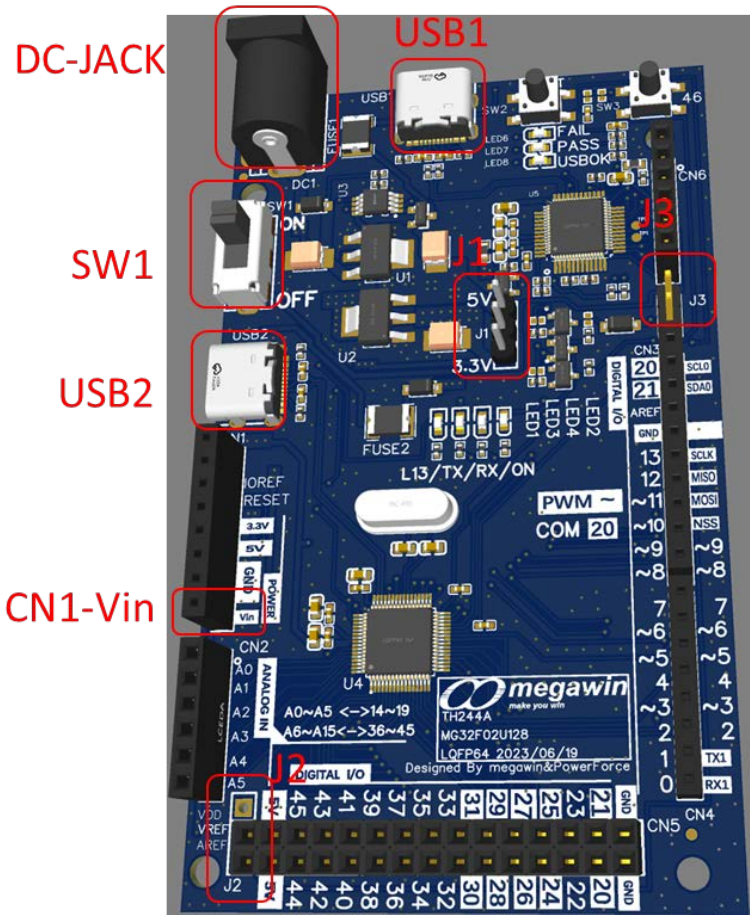
TH244A001 power can be supplied from USB1, USB2, DC JACK (2mm DC JACK input) or CN1 VIN input. TH244A001 can operate under any power supply input mode.

USB1 must be connected to PC when download program. (USB 2.0 Type A connector to Type C cable or USB 3.1 Type C TO C cable)

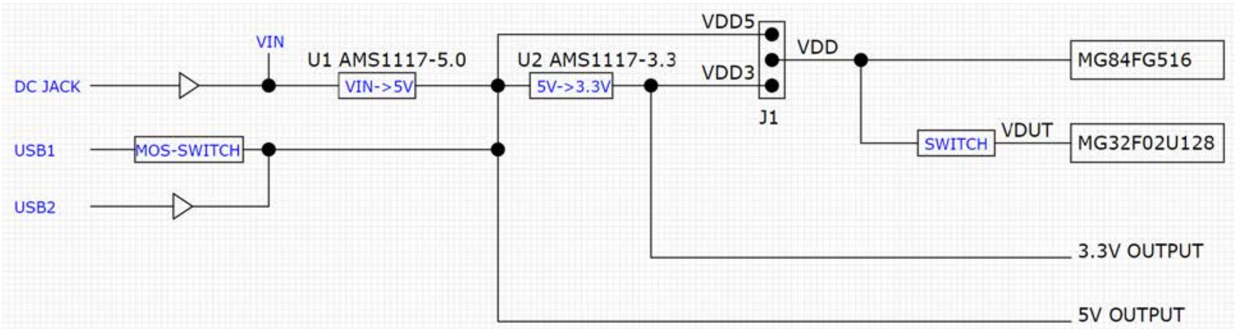
Operate voltage	Development board operate voltage 3.3 V or 5 V, use J1 to switch (see note J1)

DC IN/VIN input	7V~12V, 7V~9V is recommended.
MCU MG32F02U	VDD maximum current 200mA
MCU MG32F02U I/O	I/O current limit is 38.5mA(5V), 13mA(3.3V)
5V/3.3V output	Limit to 50mA; **When a larger current is needed for the external module power supply, the ground of module and development board should be reliably;**
USB1/USB2	500mA

Main interfaces

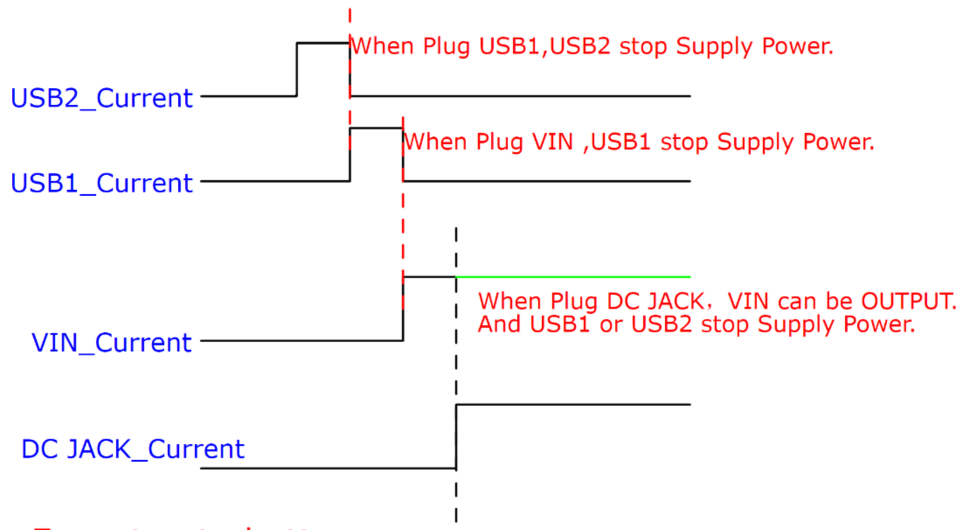


3.1. Power supply block diagram



3.2. Power supply priority

DC JACK/VIN >USB1 (Debug port)>USB2 (Native port)

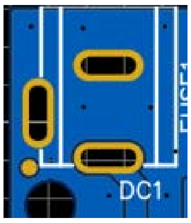
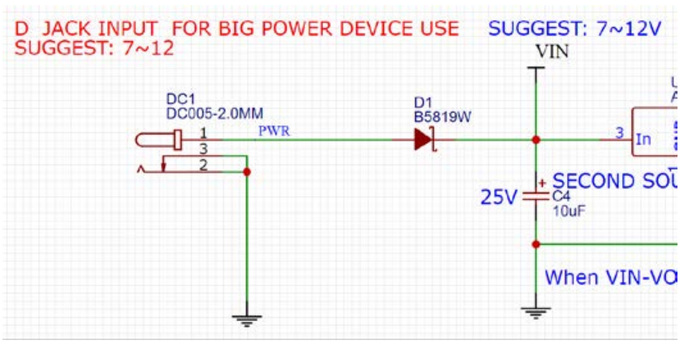

When Plug USB1,USB2 stop Supply Power.
USB2_Current
When Plug VIN ,USB1 stop Supply Power.
USB1_Current
When Plug DC JACK，VIN can be OUTPUT.
And USB1 or USB2 stop Supply Power.
VIN_Current
DC JACK_Current

From top to bottom,
different power supply is connected,
and the circuit power supply is selected Auto.

3.3. External power DC-JACK（Location DC1）：

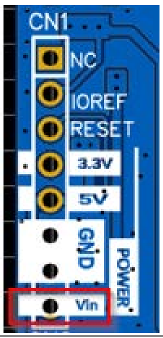
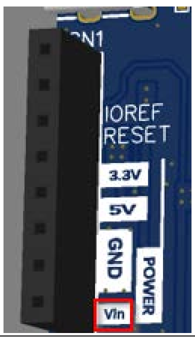
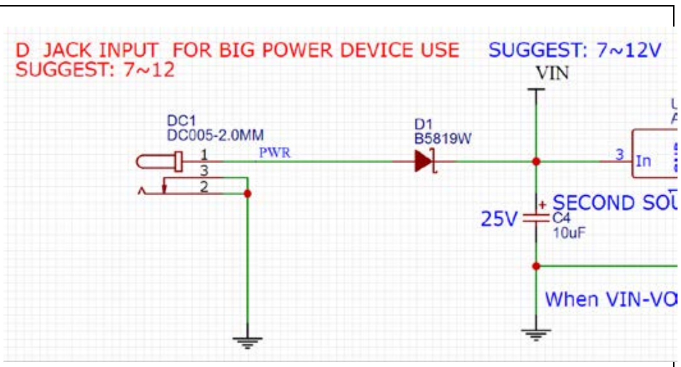
AC-DC adapter can be used (7V~9V DC output is recommended). The DC power adapter can be connected to DC-Jack via a 2.0 mm plug. The DC input range of the on board voltage regulator is from 7V to 12V. If the input voltage is lower than 7V, the main 5V DC supply may lower than 5V. If the input voltage is higher than 12V, it will damage the on board voltage regulator. The recommended input voltage range is 7V to 9V.

When an external power supply DC-jack is used, it can also be used for the external devices. The VIN pin can be used as the power supply of the external devices (please make sure that the external devices operating voltage is within the voltage of the DC JACK). For example, the motor power can be connected to VIN, which can provide enough power for the motor.

2D **3D** 	 D JACK INPUT FOR BIG POWER DEVICE USE SUGGEST: 7~12 SUGGEST: 7~12V VIN DC1 DC005-2.0MM PWR D1 B5819W 25V SECOND SOU C4 10uF In When VIN-VO

3.4. External power VIN(Location CN1):

- The system can use VIN PIN as an external power source;
- When power source from the DC JACK, VIN PIN can also supply power to external devices. Pay attention to whether the voltage suits the requirements of the modules.

2D **3D**  CN1 NC IOREF RESET 3.3V 5V GND POWER Vin 	 D JACK INPUT FOR BIG POWER DEVICE USE SUGGEST: 7~12 SUGGEST: 7~12V VIN DC1 DC005-2.0MM PWR D1 B5819W 25V SECOND SOU C4 10uF In When VIN-VO

3.5. USB1/USB2 power supply(Location USB1，USB2)

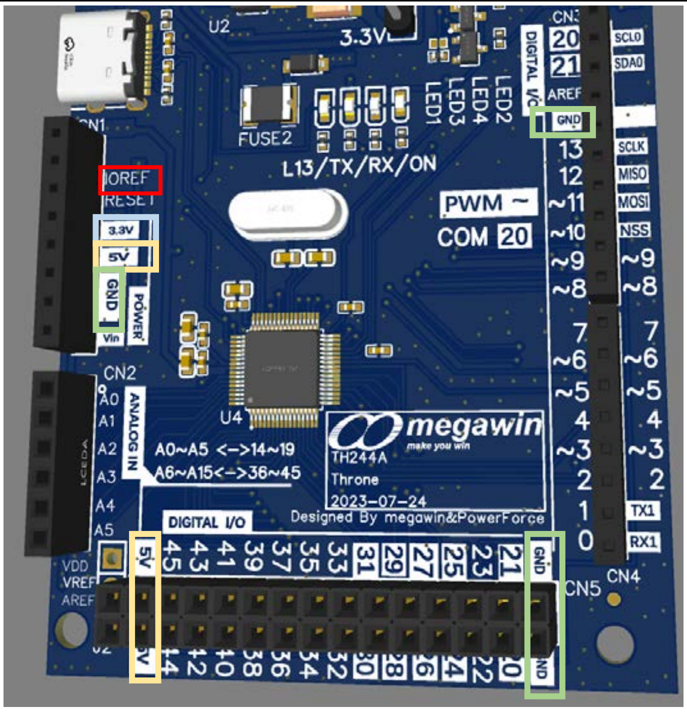
USB1 and USB2 are USB Type C interface, which can provides 5V/500mA power supply to the development board and accessory board. USB1 is the main channel of power supply.

3.6. 5V/3.3V output(Location CN1 CN5):

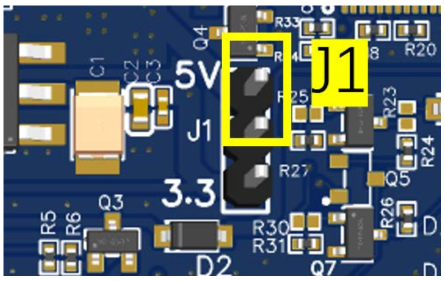
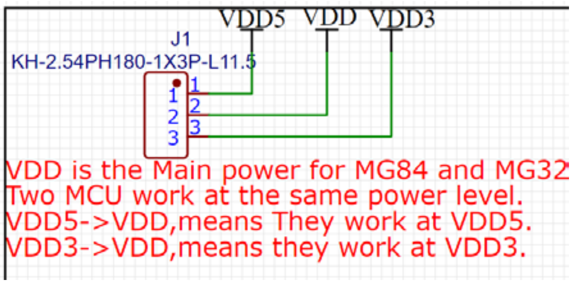
5V: Through LDO output or USB1/USB2. TH244A001 provides 3 pins for 5V output, located at CN1 and CN5. Users can supply power to the TH244A001 from DC-JACK, USB1, USB2 and VIN. The 5V and 3.3V are used as output, and input is not allowed. When use DC-JACK as power source, +5V output maximum current 50 mA. When USB power supply, +5V output maximum current is 300mA.

The 3.3V voltage pin current output limit by the LDO, when the DC JACK power supply, the maximum current of **+3.3V** is 50mA; When the USB is supplied, the maximum current of **+3.3V** is 300mA. This voltage is the operating voltage of MG32F02U128 and MG84FG516. See the **power supply block diagram** for details.

5V or 3.3V is the operating voltage of MG32F02U128 and MG84FG516. The working voltage of the development board is switched by switch J1. Select 5 V or 3.3 V power supply as needed.

 IOREF
RESET
3.3V
5V
GND
POWER
Vin
CN2
ANALOG IN
A0~A5 <->14~19
A6~A15<->36~45
DIGITAL I/O
PWM ~
COM 20
L13/TX/RX/ON
FUSE2
U2
U4
megawin
TH244A
Throne
2023-07-24
Designed By megawin&PowerForce
AREF
GND
SCL0
SDA0
SCLK
MISO
MOSI
NSS
TX1
RX1
CN4
CN5
	TH244A001 provides:
	1 3.3V output pin;
	3 5V output pin;
	5 GND pin;
	1 IOREF voltage which the same as MG32F02U J1 setting voltage;
	It is convenient for users to access more modules

3.7. J1 development board operating voltage selection

	 VDD5 VDD VDD3 J1 KH-2.54PH180-1X3P-L11.5 1 2 3 VDD is the Main power for MG84 and MG32 Two MCU work at the same power level. VDD5->VDD,means They work at VDD5. VDD3->VDD,means they work at VDD3.

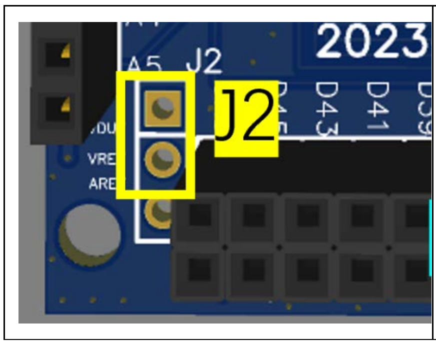
Users choose the appropriate operating voltage according to their needs.

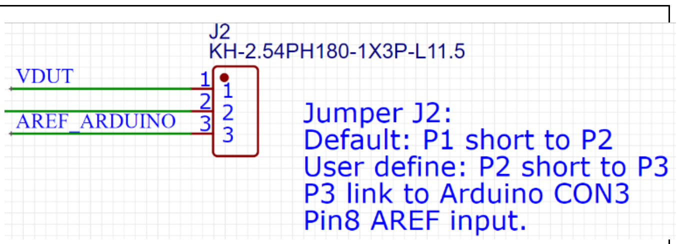
Development board to work for 5V voltage by default.

3.8. J2 ADC external reference voltage switching

MG32F02U128 in TH244A001 can set the ADC reference voltage to external voltage (VERF+) or internal IVR24. VREF+ can be fixed as the MCU working power supply VDUT (which is the more commonly used way), or the reference voltage AREF can be provided by the external device. The development board uses J2 to switch the two connections, and the default is VREF+ directly connected to VDUT. However, when the analog signal input is weak, such as 1.8V or 3.3V or other ranges, use external voltage as a reference, which can improve the sampling accuracy.

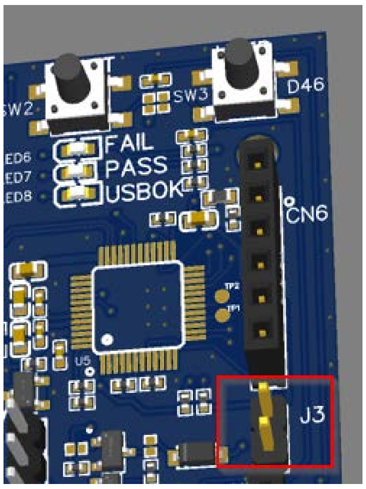
The ADC reference voltage is set to external VREF+ as the reference voltage by default, and the VREF+ voltage is the same as the selection of J1.





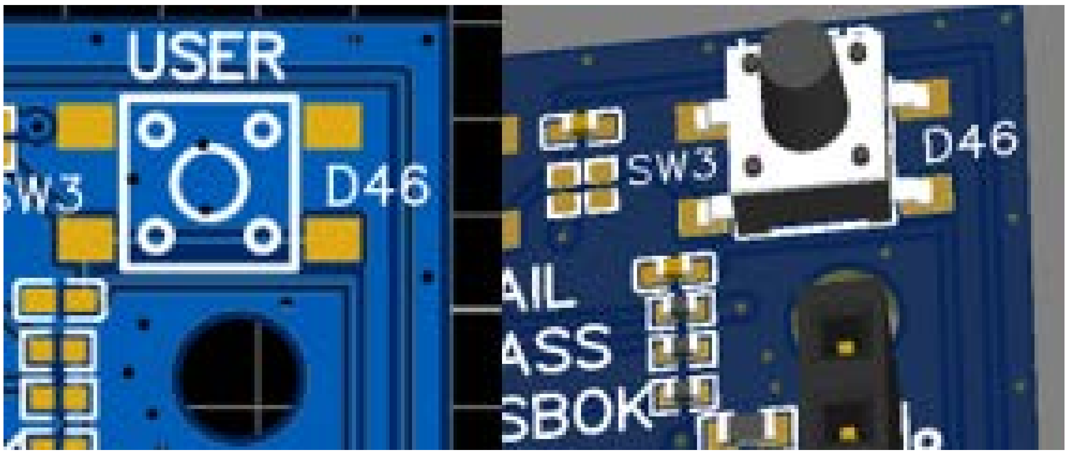
3.9. J3 Virtual com port enabled

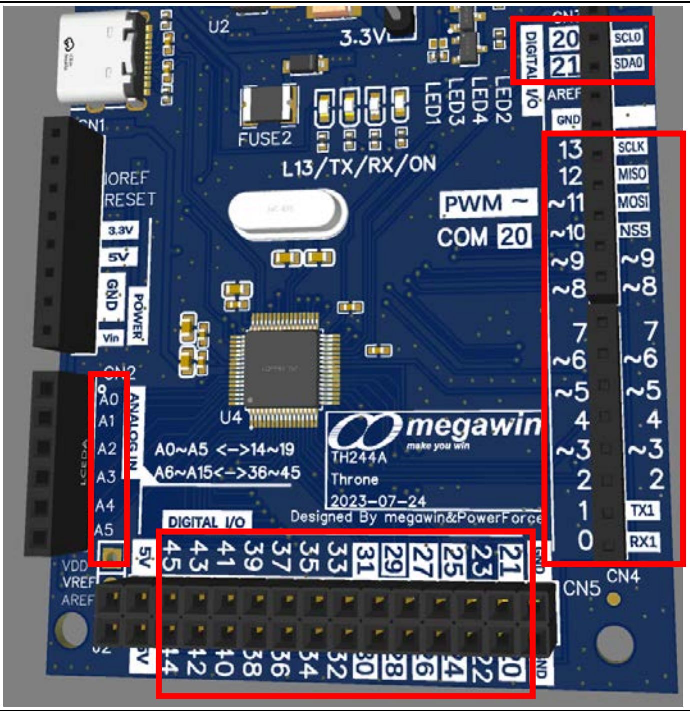
Short J3 circuit, is to enable USB1 virtual com port (VCP), used for downloading programs and debugging.

 SW2 SW3 D46 FAIL PASS USBOK CN6 U5 J3	Development board default remain short circuit, Place 2.54mm 2pin Jumper to enable virtual com port

4. Digital I/O

4.1. Brief Introduction

Digital I/O: Arduino number 0~46, 47 in total (pin 46 is specifically defined as USER KEY input, see **USER KEY SW3** for details)
 Pin 46 default button SW3 status: not pressed for HIGH, pressed for LOW;
Pin 0~45 Connect to the connector, in addition to GPIO function, multiplexing ADC/DAC/ serial I/O(UART/IIC/SPI), etc.;



TH244A001 defines 47 GPIOs that can be used by direct programming according to the use of Arduino. User can directly use the official Arduino functions, such as pinMode (), digitalWrite (), digitalRead () and other functions. The board works at 3.3V or 5V (depending on J1 on the board, the default is 5V). The working voltage is 5V or 3.3V. The current of the I/O port is related to the output power setting, as shown in the table below. The default setting is full power output, and the pull-up resistor is disabled.

When driving the load with GPIO, should be considered that the driving current should not exceed the limit. The suggestions are as follows:

- **Operate voltage 5.0V: 38.5mA Max.**
- **Operate voltage 3.3V: 13mA Max.**

4.2. Drive load capacity reference

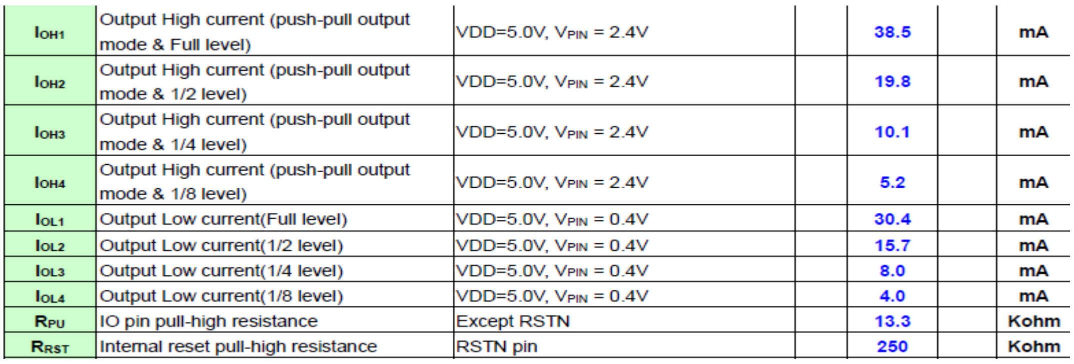
Default GPIO is full power, low speed mode, internal pull up resistor is not enabled. When low power or high speed mode is required, special Settings are required. For example, SWD/IIC/UART/SPI/USB and other modes with IO configured for communication are configured to high speed mode by default.

VDD=5V Condition，

I/O full power output HIGH, maximum current (push pull) 38.5mA

I/O full power output LOW, maximum current 30.4mA

The internal reset pin pull-up resistance is 250Kohm. All other I/Os have 13.3Kohm internal pull-up resistors;



I_{OH1}	Output High current (push-pull output mode & Full level)	VDD=5.0V, V_{PIN} = 2.4V		**38.5**		**mA**
I_{OH2}	Output High current (push-pull output mode & 1/2 level)	VDD=5.0V, V_{PIN} = 2.4V		**19.8**		**mA**
I_{OH3}	Output High current (push-pull output mode & 1/4 level)	VDD=5.0V, V_{PIN} = 2.4V		**10.1**		**mA**
I_{OH4}	Output High current (push-pull output mode & 1/8 level)	VDD=5.0V, V_{PIN} = 2.4V		**5.2**		**mA**
I_{OL1}	Output Low current(Full level)	VDD=5.0V, V_{PIN} = 0.4V		**30.4**		**mA**
I_{OL2}	Output Low current(1/2 level)	VDD=5.0V, V_{PIN} = 0.4V		**15.7**		**mA**
I_{OL3}	Output Low current(1/4 level)	VDD=5.0V, V_{PIN} = 0.4V		**8.0**		**mA**
I_{OL4}	Output Low current(1/8 level)	VDD=5.0V, V_{PIN} = 0.4V		**4.0**		**mA**
R_{PU}	IO pin pull-high resistance	Except RSTN		**13.3**		**Kohm**
R_{RST}	Internal reset pull-high resistance	RSTN pin		**250**		**Kohm**

VDD=3.3V Condition，

I/O full power output HIGH, maximum current (push pull) 13mA

I/O full power output LOW, maximum current 11.3mA

The internal reset pin pull-up resistance is 426Kohm. All other I/ Os have 19.5Kohm internal pull-up resistors;

I_{OH1}	Output High current (push-pull output mode & Full level)	VDD=3.3V, V_{PIN} = 2.4V		13		mA
I_{OH2}	Output High current (push-pull output mode & 1/2 level)	VDD=3.3V, V_{PIN} = 2.4V		6.5		mA
I_{OH3}	Output High current (push-pull output mode & 1/4 level)	VDD=3.3V, V_{PIN} = 2.4V		3.5		mA
I_{OH4}	Output High current (push-pull output mode & 1/8 level)	VDD=3.3V, V_{PIN} = 2.4V		1.7		mA
I_{OL1}	Output Low current(Full level)	VDD=3.3V, V_{PIN} = 0.4V		22		mA
I_{OL2}	Output Low current(1/2 level)	VDD=3.3V, V_{PIN} = 0.4V		11.3		mA
I_{OL3}	Output Low current(1/4 level)	VDD=3.3V, V_{PIN} = 0.4V		5.6		mA
I_{OL4}	Output Low current(1/8 level)	VDD=3.3V, V_{PIN} = 0.4V		2.8		mA
R_{PU}	IO pin pull-high resistance	Except RSTN		19.5		Kohm
R_{RST}	Internal reset pull-high resistance	RSTN pin		426		Kohm

4.3. Digital I/O list

No.	MG32F02U128AD64 LQFP64 pin No.	MG32F02U128AD64 Pin name	**Arduino Pin name***
1	35	PC9/RXD1	**0**
2	34	PC8/TXD1	**1**
3	14	SD3/PB4	**2**
4	20	PB10	**3**
5	33	PC7	**4**
6	21	PB11	**5**
7	26	ICKO/PC0	**6**
8	15	SD2/PB5	**7**
9	38	PC12	**8**
10	42	PD0	**9**
11	51	NSS/PD9	**10**

12	44	MOSI/PD2	**11**
13	49	MISO/PD7	**12**
14	50	SPI0_CLK/PD8	**13**
15	58	PA0/ADC0	**14/A0****
16	59	PA1/ADC1	**15/A1**
17	60	PA2/ADC2	**16/A2**
18	61	PA3/ADC3	**17/A3**
19	62	PA4/ADC4	**18/A4**
20	63	PA5/ADC5	**19/A5**
21	13	MOSI/PB3	**20**
22	12	SCK/PB2	**21 & DAC*****
23	37	PC11/SDA1	**22**
24	36	PC10/SCL1	**23**
25	17	PB7	**24**
26	16	PB6	**25**
27	24	PB14	**27**
28	23	PB13	**26**
29	25	PB15	**28**
30	22	PB12	**29**
31	10	NSS/PB0	**31**
32	11	MISO/PB1	**30**

33	43	PD1	**32**
34	45	PD3	**33**
35	52	PD10	**34**
36	53	PD11	**35**
37	64	PA6/ADC6	**36/A6****
38	1	ADC7/PA7	**37/A7****
39	2	ADC8/PA8	**38/A8****
40	3	ADC9/PA9	**39/A9****
41	4	ADC10/PA10	**40/A10****
42	5	ADC11/PA11	**41/A11****
43	6	ADC12/PA12	**42/A12****
44	7	ADC13/PA13	**43/A13****
45	8	ADC14/PA14	**44/A14****
46	9	ADC15/PA15	**45/A15****
47	27	PC1	**46 for USER KEY******

Digital I/O special notes

*** Arduino Pin name** The number of this marker can be referenced directly in Arduino. Integers also make it easy to use loops, as follows:

- digitalWrite(40，HIGH), write 40 pin output HIGH
- digitalRead(15), read 15 pin input state

- analogWrite(3，200), pin 3 output PWM which duty is 200/255=78.4%

  The default frequency is 1000Hz;
- analogRead(A0) reads the simulated analog data value from the AO

  port
- for(int i = 0; i<20;i++){

  pinMode(i，OUTPUT);

  }　//Using loop, setting pin 0~19 as output

****Means digital port**, and can be reused as an analog input port;

A0~A15 mean analog input channels

*** **21** is DAC output, analogWrite(21，value)　value: 0~4095, output

voltage would be 0~VDD without external loads.

**** **USER KEY**, used for key function, the hardware has been designed for pull-

up resistor, can only be used as INPUT;

Common use functions:

PinMode ()

DigitalRead ()

DigitalWrite ()

See GPIO read and write control for common functions and more details.

5. Time function TIMER

The time function is very important in timing control. Arduino provides functions about time control. Arduino provides four main time functions as follows.

- Block functions：delay(ms) and delayMicroseconds(us)
  - delay(ms): Pause execution of the device for milliseconds (ms)
  - delayMicroseconds(us): Pause execution of the device for microseconds (us)

In delay time, MCU can only make time to wait, until the delay time to complete. If the waiting phase needs to be delayed and other requests need to be responded to at any time, a blocking function such as delay cannot be used. Use this with caution in general programming.

For example, use delay () to delay by 1s:

Delay (1000);　//delay for 1s, MCU can't do anything else at this stage;

- Non-blocking：Use systick. When using millis or micros ((),
- the MCU can handle something else while it it's timing
  - millis()　Returns the number of milliseconds since the Arduino board started
  - micros() Returns the number of milliseconds since the Arduino board started

For example, using millis () to achieve an interval of 1s, the system timing

phase can perform other tasks

```
unsigned long previousMillis = 0; // define variable to store current systick
const long interval = 1000;       // time interval to wait;
unsigned long currentMillis = millis (); // Return the number of ms, this time will
start at 0 when the system has run, repower or RESET;
if (currentMillis - previousMillis >= interval)
{
    previousMillis = currentMillis;
    // Place tasks after meet the needs of the time waiting
}
```

Common use functions:

delay (ms)

delayMicroseconds (us)

millis ()

micros ()

See Delay function, Timer Using for details.

6. Alternate function selection

Many Digital I/O pins can be configured to use special functions, which are described in the chip manual AFS section.

Specific functions here mainly refer to PWM, Analog Output (DAC), Analog input (ADC), UART, IIC, SPI, etc.

Generally, users should complete the initialization of these multiplexing functions in setup (). After a pin initializes function completed, it cannot use another function in the definition to avoid conflicts. For example, pin 21 is set to an analog voltage output DAC, then IIC0 (20->SCL, 21->SDA) cannot be configured. Such as the initialization IIC0 (20 - > SCL, 21 - > SDA), cannot use UART7 (20 - > 21 - > TX and RX).

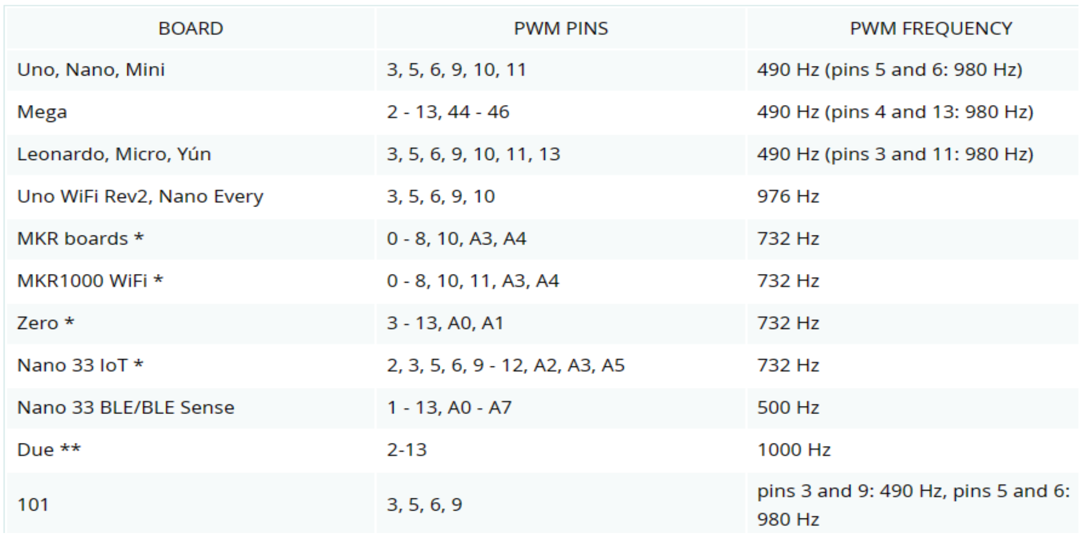
7. PWM

TH244A001 provides 7 PWM channels at 3, 6, 5, 8, 9, 10 and 11. The default frequency is 1000Hz. The PWM channels can control the duty independently, ranging from 0 to 100%.

Different from other official development boards, the development kit of TH244A001 supports frequency setting. Users can set the frequency range from 1Hz to 5000Hz. The frequency setting is divided into three groups according to the following figure: 3/6 (TM20), 5/8/9(TM36) and 10/11(TM26).

Group	MG32F02U128AD64 LQFP64 pin No.	MG32F02U128AD64 Pin name	**Arduino Pin name**	TMxx	Frequency
TM20	20	PB10	3	TM20_OC11	Default 1KHz
	26	PC0	6	TM20_OC00	Default 1KHz
TM36	21	PB11	5	TM36_0C12	Default 1KHz
	38	PC12	8	TM36_OC3	Default 1KHz
	42	PD0	9	TM36_OC2	Default 1KHz
TM26	44	PD2	11	TM26_OC00	Default 1KHz
	51	PD9	10	TM26_OC11	Default 1KHz

Refer to the PWM fixed frequency of the official Arduino board:



BOARD	PWM PINS	PWM FREQUENCY
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)
Mega	2 - 13, 44 - 46	490 Hz (pins 4 and 13: 980 Hz)
Leonardo, Micro, Yún	3, 5, 6, 9, 10, 11, 13	490 Hz (pins 3 and 11: 980 Hz)
Uno WiFi Rev2, Nano Every	3, 5, 6, 9, 10	976 Hz
MKR boards *	0 - 8, 10, A3, A4	732 Hz
MKR1000 WiFi *	0 - 8, 10, 11, A3, A4	732 Hz
Zero *	3 - 13, A0, A1	732 Hz
Nano 33 IoT *	2, 3, 5, 6, 9 - 12, A2, A3, A5	732 Hz
Nano 33 BLE/BLE Sense	1 - 13, A0 - A7	500 Hz
Due **	2-13	1000 Hz
101	3, 5, 6, 9	pins 3 and 9: 490 Hz, pins 5 and 6: 980 Hz

TH244A001 only 3/6, 5/8/9, 10/11 pin that support PWM function can use the function to set the frequency.

Special Note 1: 3/6(TM20), 5/8/9(TM36) and 10/11(TM26) **can be set independently. Default frequency is 1000Hz;**

- analogWriteFrequency(PWM_FRQ_D3_D6，Frq) : set the 3/6 output frequency
- analogWriteFrequency(PWM_FRQ_D5_D8_D9，Frq) : set the 3/6 output frequency
- analogWriteFrequency(PWM_FRQ_D10_D11，Frq) : set the 3/6 output frequency
- analogWriteFrequency(PWM_FRQ_ALL，Frq) : set the 3/6, 5/8/9, 10/11 output frequency
- Frq is the set frequency, can be set in the range of 1Hz~5000Hz

Special Use Note 2:

When using the Timer One library function, the function of PWM 10/11 cannot be used.

When using MsTimer2 library function, PWM D3/D6 function cannot be used.

When using the tone () function, all PWMs cannot be used.

Common use functions:

Function: analogWrite(pin， ulValue) // ulValue=0~255

Function: analogWriteFrequency(PWM_FRQ_XX，Fre); Set PWM frequency.

See for PWM details.

8. DAC

TH244A001 provides one 12 bit **DAC_P0** (0 ~ 4095) of the DAC output channel. The full scale output analog voltage range is 0V~VDD, and the output is linear.

TH244A001 PIN 21 is DAC output interface. User can connect PIN 21 to external modules. The hardware notes are as follows for improve output voltage stability.

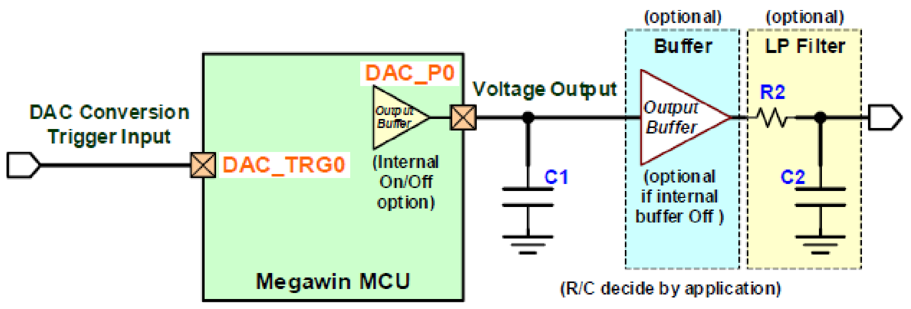
DAC_P0 is MG32F02U 12 bit analog output channel

5. DAC Application Circuit

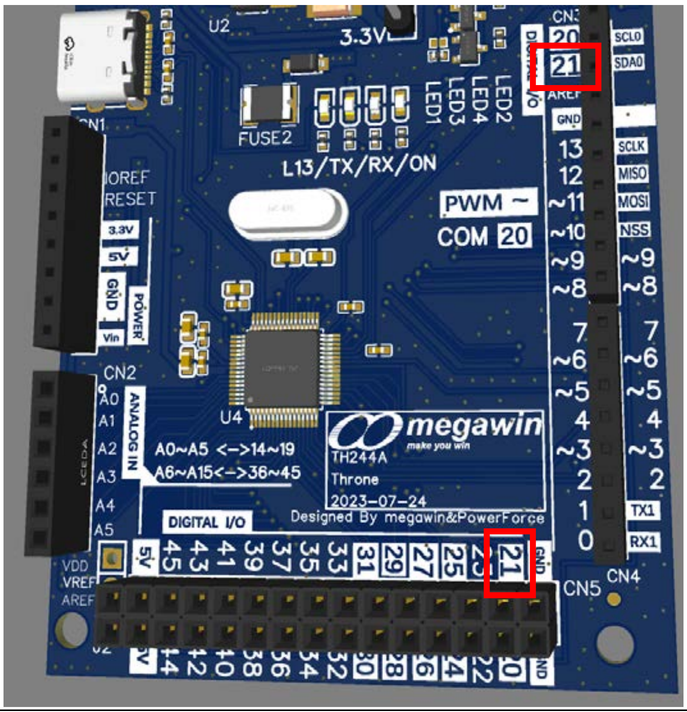
The DAC is built-in an internal output buffer which output resistive load is minimum 7.5Kohm. A capacitor (*C1*) is suggested and used to keep more stable output voltage. When the internal output buffer is enabled, the DAC output range is 0.2 volt to VDD-0.2 volt.

The external output buffer is suggested necessary if the output resistive load is over the 7.5Kohm. An external low-pass filter (*R2*/*C2*) is suggestion to be implemented in the application circuit for more better outputting performance. An optional **DAC_TRG0** pin is able to input the trigger signal for DAC output conversion.

Figure 7-5. DAC Application Circuit



Any PIN 21 can be used



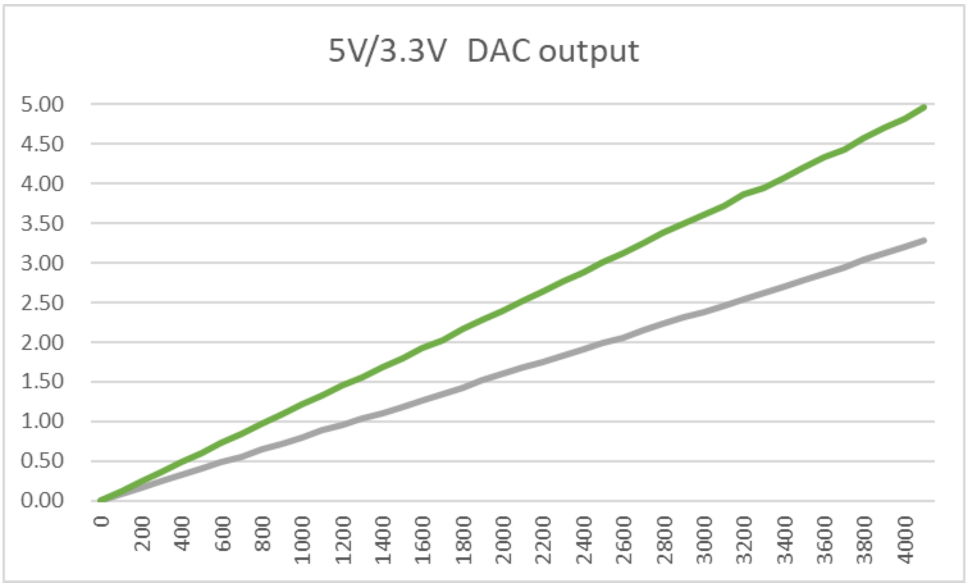
Function: analogWrite(21， value)

The value range is 0 to 4095. The DAC uses the same function as the PWM control, corresponding to the actual output voltage of 0~VDD;

Demo code:

```
void setup () {
  analogWrite(21,500);   // Modify different values, the simulation output voltage
is 500/4095*VDD
}
void loop () {
}
```

DAC output (without load)


5V/3.3V DAC output
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
2400
2600
2800
3000
3200
3400
3600
3800
4000

Reference data

analogWrite(21， Value)	VDD3.3V		VDD5V	
Value	**mV**	**V**	**mV**	**V**
0	0	0	0	0
100	80	0.08	122	0.122
200	160	0.16	243	0.243
300	240	0.24	363	0.363
400	320	0.32	484	0.484
500	400	0.4	606	0.606
600	481	0.481	727	0.727
700	560	0.56	840	0.84
800	643	0.643	968	0.968
900	718	0.718	1080	1.08
1000	800	0.8	1210	1.21
1100	884	0.884	1330	1.33
1200	960	0.96	1450	1.45
1300	1030	1.03	1560	1.56
1400	1110	1.11	1680	1.68
1500	1190	1.19	1800	1.8
1600	1270	1.27	1920	1.92
1700	1340	1.34	2030	2.03

1800	1430	1.43	2170	2.17
1900	1520	1.52	2280	2.28
2000	1600	1.6	2400	2.4
2100	1680	1.68	2520	2.52
2200	1750	1.75	2640	2.64
2300	1830	1.83	2760	2.76
2400	1910	1.91	2880	2.88
2500	1990	1.99	3010	3.01
2600	2060	2.06	3120	3.12
2700	2150	2.15	3260	3.26
2800	2240	2.24	3380	3.38
2900	2320	2.32	3490	3.49
3000	2380	2.38	3600	3.6
3100	2460	2.46	3720	3.72
3200	2540	2.54	3860	3.86
3300	2620	2.62	3940	3.94
3400	2700	2.7	4080	4.08
3500	2780	2.78	4200	4.2
3600	2860	2.86	4340	4.34
3700	2940	2.94	4430	4.43
3800	3040	3.04	4580	4.58

3900	3120	3.12	4700	4.7
4000	3210	3.21	4820	4.82
4050	3240	3.24	4860	4.86
4095	3280	3.28	4960	4.96

Common use functions:

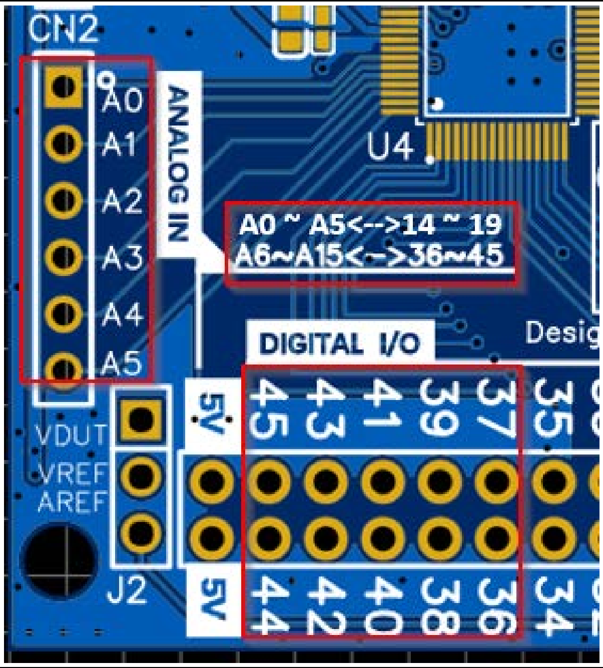
Function: analogWrite(21，Value)

See DAC for details.

Note: when 21 is set to analog output mode, I2C0 (SCL0->20; SDA0->21) is not available;

9. ADC

9.1. ADC brief introduction

There are 16 analog inputs in total, that A0-A5 corresponds to 14-19; A6-A15 corresponds to 36-45
 CN2 A0 A1 A2 A3 A4 A5 ANALOG IN U4 A0 ~ A5<-->14 ~ 19 A6~A15<-->36~45 DIGITAL I/O 5V 45 43 41 39 37 35 44 42 40 38 36 34 VDUT VREF AREF J2

12 bit 1.5Msps SAR ADC

− Configurable resolution: 12/10/8 bit. TH244A001 is set to 12bit by default.

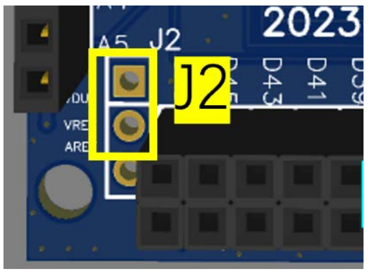
− TH244A001 provides 16 channels input.

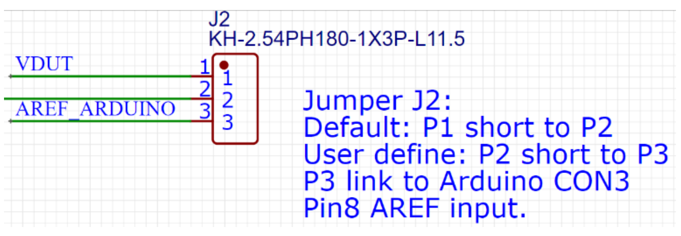
-ADC reference voltage can set to external VREF+ or internal IVR24.

- analogRead (): read A0~A15 ADC Value, return value is 12 bit, 0~4095 by default. analogRead (A0) acted the same as analogRead (14). Generally use analogRead(Ax), x:0~15;

- analogReadResolution (ref): set ADC resolution to 8 bit, 10 bit or 12 bit. The default is 12 bit;
- analogReference (type): set ADC reference voltage to external VREF+ or internal IVR24. It is recommended to use the default external voltage source VREF+ as the reference voltage;
- The parameter type: *AR_INTERNAL, AR_EXTERNAL, AR_DEFAULT* (AR_EXTERNAL)

TH244A001 can be set ADC reference voltage to external VREF+ or internal IVR24. VREF+ fixed as MCU operate voltage VDD (this is the more common way), can also be provided by external reference voltage AREF. TH244A001 use J2 to switch the external reference and the default is VREF+. When external analog input is weak, such as 1.8V, 3.3V or other voltage, using external voltage can improve ADC performance.





The sampling accuracy of the ADC is as follows:

When ADC is set to 10 bit resolution,

1. For example, VREF+ is fixed to VDD 5V

- Analog input 0~5V will be mapped to 0~1023, and precision is 5/1024= 4.882mV

  Input voltage between 0~4.882mV will be mapped to 0,

  Input voltage between 4.883mV~9.764mV will be mapped to 1……

  Input voltage between 9.765mV~14.646mV will be mapped to 2……
- Analog input 0~3.3v will be mapped to 0~676 (676*4.882mV=3.3v)

2. If VREF+ is set to AREF, and AREF input is 3.3V, then analog input 0~3.3v will be mapped to 0~1023, the precision is 3.3/1024= 3.223mV. And the actual precision is increased from 4.882mV to 3.223mV.

3 If the resolution is set from 10 bit to12 bit,

For example, VDD= 5V, VREF+ is connected to VDD,

10bit resolution, 5V/1024=4.882mV, TH244A001 is 10 bit by default.

12bit resolution, 5V/4096=1.221mV,

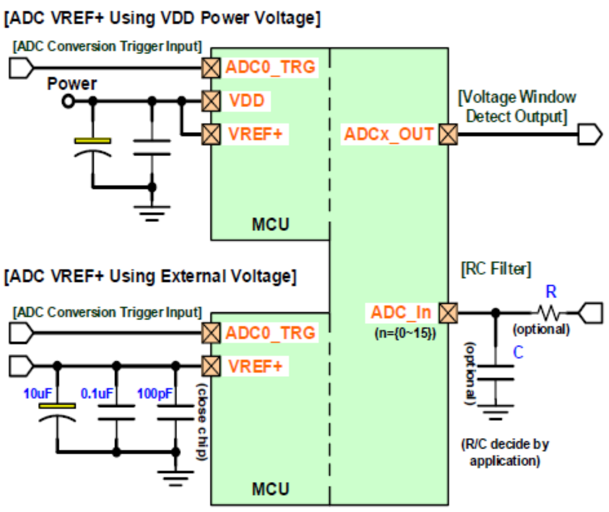
ADC reference voltage source note:

The ADC reference voltage source can be from (1) VDD power by connecting **+VREF** pin to **VDD** pin directly (2) external quiet reference voltage source.

When uses the VDD power as the ADC reference voltage, it must connect **+VREF** pin trace to the point which is at current flow behind the power capacitor(s). When uses the external reference voltage source as the ADC reference voltage, it must add some decoupling and bypass capacitors, as shown in following figure.

An optional **ADCx_TRG** pin is able to input the trigger signal for ADC input conversion and an optional **ADCx_OUT** pin is used to output the internal ADC window detection status.

Figure 7-4. ADC Application Circuit


[ADC VREF+ Using VDD Power Voltage]
[ADC Conversion Trigger Input]
Power
ADC0_TRG
VDD
VREF+
MCU
ADCx_OUT
[Voltage Window Detect Output]
[ADC VREF+ Using External Voltage]
[ADC Conversion Trigger Input]
ADC0_TRG
VREF+
10uF
0.1uF
100pF
(close chip)
MCU
[RC Filter]
ADC_In
(n=(0~15))
R
(optional)
C
(optional)
(R/C decide by application)

9.2. ANALOG IN list

No.	MG32F02U128AD64 LQFP64 pin No.	MG32F02U128AD64 Pin name	**Arduino Pin name**	Arduino name2
1	58	PA0/ADC0	14	A0
2	59	PA1/ADC1	15	A1
3	60	PA2/ADC2	16	A2
4	61	PA3/ADC3	17	A3
5	62	PA4/ADC4	18	A4
6	63	PA5/ADC5	19	A5
7	64	PA6/ADC6	36	A6
8	1	ADC7/PA7	37	A7

9	2	ADC8/PA8	38	A8
10	3	ADC9/PA9	39	A9
11	4	ADC10/PA10	40	A10
12	5	ADC11/PA11	41	A11
13	6	ADC12/PA12	42	A12
14	7	ADC13/PA13	43	A13
15	8	ADC14/PA14	44	A14
16	9	ADC15/PA15	45	A15

Common use functions:

Function: analogRead ()

Function: analogReadResolution (ref)

Function: analogReference (type)

Function: map ()

See for ADC details.

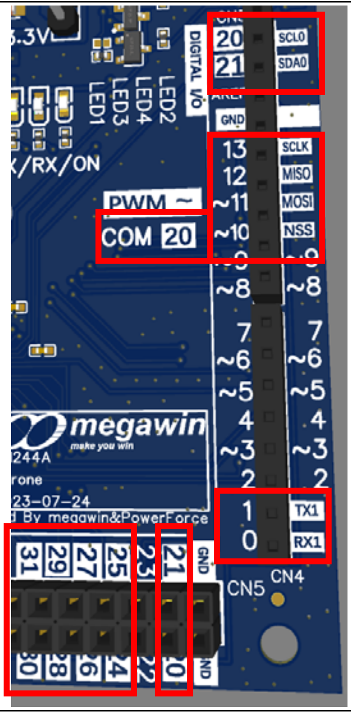
10. Serial I/O

TH244A001 provides many interfaces to communicate with other PC, Arduino board or modules. TH244A001 provides 7 channels UART, 2channels IIC, 1 channel SPI. Using these interfaces can communicate with other PC, external devices or modules.

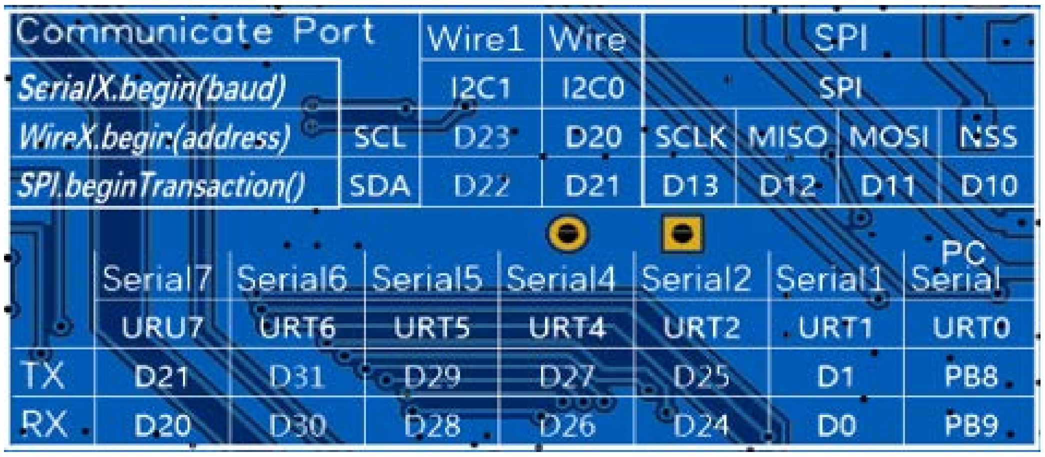
Usually using library function when using Arduino. Commonly used three kinds of interfaces that UART, IIC and SPI are using Arduino library function. Users need to learn to use the functions. When users need to write their own function, then need to write the functions according to the protocols.

UART -> Serial.h; IIC-> Wire.h; SPI->SPI.h

The front side of TH244A001 has a special white square marking the UART/IIC/SPI pin position



The back side of TH244A001 has specifically list communication UART/IIC/SPI available pin position



Communicate Port		Wire1	Wire	SPI			
SerialX.begin(baud)		I2C1	I2C0	SPI			
WireX.begin(address)	SCL	D23	D20	SCLK	MISO	MOSI	NSS
SPI.beginTransaction()	SDA	D22	D21	D13	D12	D11	D10

	Serial7	Serial6	Serial5	Serial4	Serial2	Serial1	PC Serial
	URU7	URT6	URT5	URT4	URT2	URT1	URT0
TX	D21	D31	D29	D27	D25	D1	PB8
RX	D20	D30	D28	D26	D24	D0	PB9

10.1. UART

TH244A001 provides 7 channels UART: UART0 (download code only)/ UART1/ UART2/ UART4/ UART5/ UART6/ UART7. Arduino library Serial.h. The UART receive and transmit buffer is 64 bytes.

Support baud rate: 1200bps, 2400bps, 4800bps, 9600bps, 19200bps, 38400bps, 57600bps, and 115200bps. Default 9600bps

MG32F02U128AD64 LQFP64 pin No.	MG32F02U128AD64 Pin name	Arduino Pin name	Initialization UART function	UART No.	remark
18	PB8	NA	Serial.begin(baud)	UART0	UART0_TX download code only
19	PB9	NA			UART0_RX download code only
35	PC9/RXD1	0	Serial1.begin(baud)	UART1	UART1_RX
34	PC8/TXD1	1			UART1_TX
17	PB7	24	Serial2.begin(baud)	UART2	UART2_RX
16	PB6	25			UART2_TX
23	PB13	26	Serial4.begin(baud)	UART4	UART4_RX
24	PB14	27			UART4_TX
25	PB15	28	Serial5.begin(baud)	UART5	UART5_RX
22	PB12	29			UART5_TX
11	PB1	30	Serial6.begin(baud)	UART6	UART6_RX
10	PB0	31			UART6_TX
13	MOSI/PB3	20	Serial7.begin(baud)	UART7	UART7_RX multiplexed with IIC0_SCL
12	SCK/PB2	21			UART7_TX multiplexed with IIC0_SDA

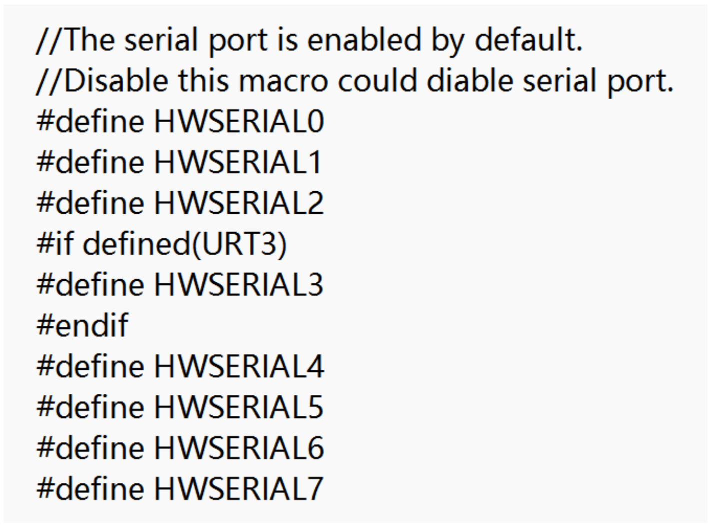
Enabling and Disabling Serial Ports

The board supports 7 serial ports, all of which are enabled by default, and the speed is 9600bps by default.

In order to save the initialization of the serial port transceiver buffer occupies SRAM, when you do not need the serial port, you can modify the macro definition in pins_arduino.h to disable some of the serial ports. The method of disabling is as follows:

Under the path of Arduino (decided according to the local computer, x.x.x represents the actual version number used)

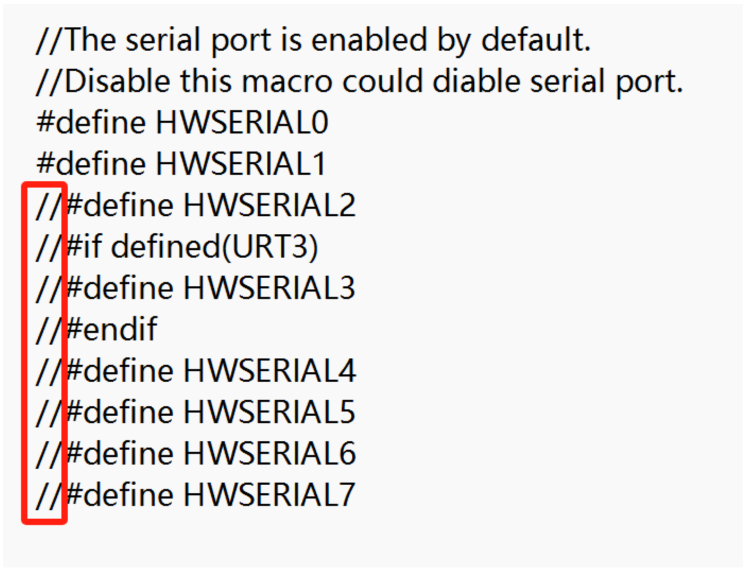
......\Arduino15\packages\megawin\hardware\MG32x02z\x.x.x\variants\TH244A001, open the pins_arduino.h file in the following section:



```
//The serial port is enabled by default.
//Disable this macro could diable serial port.
#define HWSERIAL0
#define HWSERIAL1
#define HWSERIAL2
#if defined(URT3)
#define HWSERIAL3
#endif
#define HWSERIAL4
#define HWSERIAL5
#define HWSERIAL6
#define HWSERIAL7
```

By default UART0, UART1/2/4/5/6/7 are enabled.

If you need to disable a serial port, just comment out this line. For example, if you only want to keep UART0 and UART1, but don't need the other ports for development, you can set the following settings to disable all the ports except UART0 and UART1, and save the file. After that, when you use Arduino IDE to compile the project, you will no longer initialize the disabled serial ports.



```
//The serial port is enabled by default.
//Disable this macro could diable serial port.
#define HWSERIAL0
#define HWSERIAL1
//#define HWSERIAL2
//#if defined(URT3)
//#define HWSERIAL3
//#endif
//#define HWSERIAL4
//#define HWSERIAL5
//#define HWSERIAL6
//#define HWSERIAL7
```

Common use functions

Function: Serial.begin (baud);   enable UART ()

Serial**x**.begin(baud) x:1,2,4,5,6,7

Baud means transmit binary bits per second. For example, Baud rate 9600 means 9600 means that 9,600 binary bits can be transmitted per second, which is 9,600/8 = 1,200 bytes

Function: Serial.println (); Serial.print (); prints information as a string

Function: Serial.available (); gets the number of bytes read on the Serial port. In particular, the data that received and is stored in the receive buffer. Generally used with Serial.read ();

Function: Serial.read (); read UART receive buffer, returning 1 byte at a time. Need to use with Serial.available () to read all data from receive buffer;

Function: Serial.Write (); Write data to UART

Note: Difference between Serial.print () and Serial.write ()

Serial.print () sends characters; Serial.println () sends Enter key after finish sending characters;

Serial.write () sends byte;

Serial.println (int 97) will output 97 while Serial.Write (int 97) output ASCII 97, which is a letter 'a'

See UART for details.

10.2. IIC

TH244A001 provides 2 channels IIC. TH244A001 supports IIC master and slave mode. Arduino library Wire.h.

TH244A001 acted as master and using Wire.h to communicate with other devices. The IIC speed on TH244A001 is 100 KHz by default. It can be set to 400 KHz or 1MHz.

There is necessary to add pull-up resistors on external IIC devices to ensure stable communication.

Users can use Arduino library function in Wire.h to read/write from/to devices. Users can also design their own functions according to their specific needs to achieve more flexible communication requirements.

MG32F02U128AD64 LQFP64 pin No.	MG32F02U128AD64 Pin name	**Arduino Pin name**	Initialization functions	IIC index	IIC No.
13	MOSI/PB3	20	Wire.begin()	IIC0	IIC0_SCL multiplexed with UART7_RX
12	SCK/PB2	21			IIC0_SDA multiplexed with UART7_TX
36	PC10/SCL1	23	Wire1.begin()	IIC1	IIC1_SCL
37	PC11/SDA1	22			IIC1_SDA

Wire common use functions	Comment
Wire.begin() ; Wire.begin(address)	Initial IIC
Wire.onReceive(receiveEvent)	IIC event. Slave request master to send data Master writes and sends, slave reads and receives
Wire.onRequest(requestEvent)	IIC event. Master request slave to send data Master reads and receives, slave writes and sends
Wire.requestFrom(address,quantity)	Used with Wire.onRequest(requestEvent). Master request specific slave to send specific length data

Wire.available()	After master used Wire.requestFrom(), master can use Wire.available() to receive length of the receive buffer; Using Wire.read() to read out data, the returning value of Wire.available () will be decreased; Slave uses Wire.onReceive(receiveEvent) to send data, slave can use Wire.available() to receive length of the receive buffer; Using Wire.read() to read out data, the returning value of Wire.available () will be decreased;
Wire.read()	Master or slave can use Wire.read() to read out data byte-by-byte after received data; Generally used with Wire.available（）；
Wire.onRequest(requestEvent)， Wire.requestFrom(address,quantity) ， Wire.available()，Wire.read() Generally, master will use these functions to request slave data;	
Wire.beginTransmission(address)	Send a START character to send buffer, and specifies the address of the slave to receive data. Usually used with Wire.write()

Wire.write(value)
Wire.write(string)
Wire.write(data, length)
	Send data to the send buffer, can be a string, number, or specify the data length.

Wire.endTransmission()
Wire.endTransmission(stop)
	Complete the transfer process from master to slave.
	Ending the process started by Wire.endTransmission(), and Sent by Wire.write().

Master send is generally performed using the following functions in sequence: Wire.beginTransmission(address)，Wire.write()，Wire.endTransmission()	

Refer to the Arduino official for more instructions on Wire:

Wire - Arduino Reference

https://www.arduino.cc/reference/en/language/functions/communication/wire/

See IIC for details.

10.3. SPI

TH244A001 provides 1 channel SPI master by using SPI.h.

SPI usually used in SPI Flash and SPI LCD. Users can use Arduino library functions to control. SPI maximum speed is 18 Mbps.

MG32F02U128AD64 LQFP64 pin No.	MG32F02U128AD64 Pin name	**Arduino Pin name**	Initialization function
51	NSS/PD9	10	

44	MOSI/PD2	11	SPI.begin()
49	MISO/PD7	12	
50	SPI0_CLK/PD8	13	

There are 2 transfer orders on SPI: MSB first and LSB first:

- LSBFIRST: least significant bit first means that the least significant bit of binary data is transmitted or received first
- MSBFIRST : most significant bit first means that the most significant bit of binary data is transmitted or received first

Transmit and receive mode must be consistent. For example, master transmit data of MSBFIRST, slave should also receive data of MSBFIRST.

SPI provides transfer mode: MODE0/1/2/3, there are differences in SCLK and sampling timing among these 4 modes.

SCLK idle state is determined by CPOL (Clock Polarity):

- When CPOL is 0, the level when the clock is idle is low;
- When CPOL is 1, the level when the clock is idle is high;

Sampling time is determined by CPHA (clock phase):

- When CPHA is 0, data sampling on leading edge;
- When CPHA is 1, data sampling on trailing edge;

CPOL	CPHA	Mode

0	0	MODE0
0	1	MODE1
1	0	MODE2
1	1	MODE3

Users need to ensure the specifications of the slave, and use the mode that master and slave supported.

Common use functions:

SPI.begin()	Enable/Disable Bus function
SPI.end()	
SPI.beginTransaction()	Enable/Disable Bus function with SPI configuration
SPI. endTransaction（）	
SPISettings 作为 SPI.beginTransaction()的参数使用	Setting SPI configuration, such as: speed and MSB/LSB and transfer mode.
SPI.transfer()	Data transfer function
SPI.setClockDivider(SPI_CLOCK_DIVx) ● SPI_CLOCK_DIV2 ● SPI_CLOCK_DIV4 ● SPI_CLOCK_DIV8	Setting SPI clock, divided by CPU clock (36MHz). Usually it is no need to be configured that the clock is 18 MHz

● SPI_CLOCK_DIV16 ● SPI_CLOCK_DIV32 ● SPI_CLOCK_DIV64 ● SPI_CLOCK_DIV128	

Seed SPI for details.

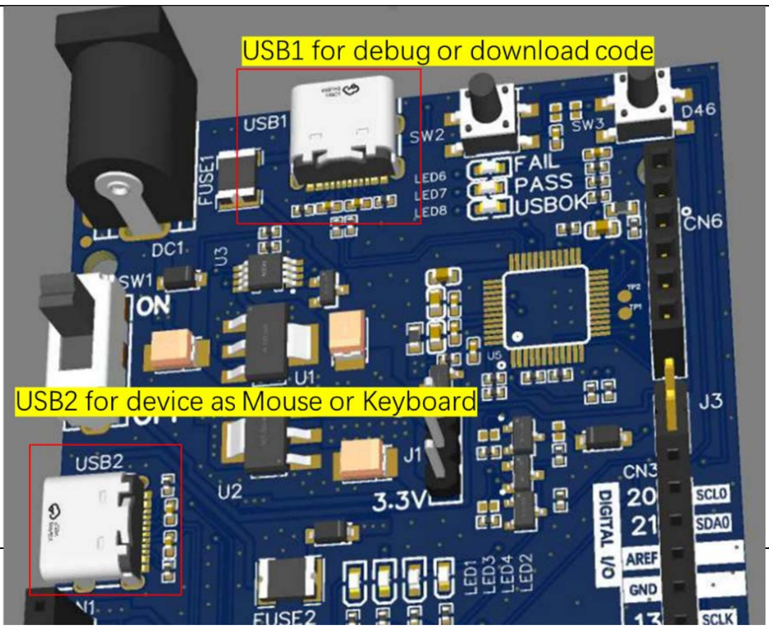
Refer to the Arduino official for more instructions on SPI.

SPI - Arduino Reference

https://www.arduino.cc/reference/en/language/functions/communication/spi/

11. USB

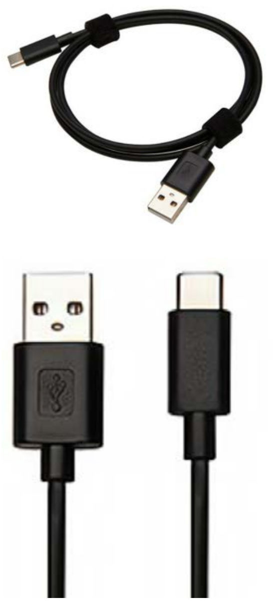
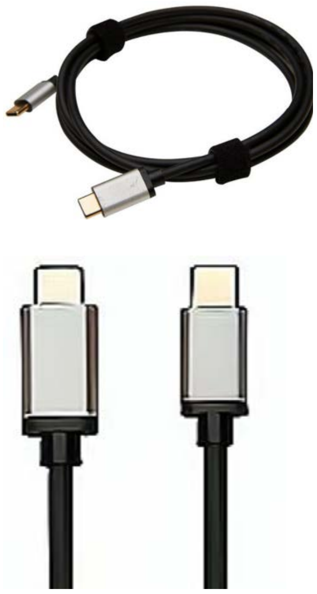
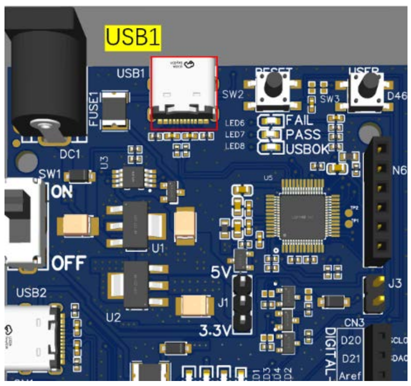
TH244A001 provides two USB interface (Type-C). USB1 is specially used for debugging and downloading programs; USB2 can be used for device, connected to the PC as a Mouse or Keyboard


USB1 for debug or download code
USB1
USB2 for device as Mouse or Keyboard
USB2

11.1. USB1

USB1 is used as a debug and download port. Use a USB A TO C cable, or USB 3.1 C TO C cable connected to the PC. USB1 is specially used for debugging and downloading programs. PC will identify the virtual com port, Arduino IDE will download programs or transfer data through com port.

USB A TO C cable	**USB 3.1 C TO C cable**	USB1 interface for download programs

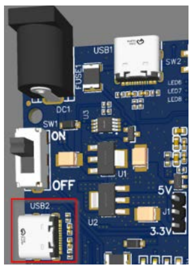
Must include USB2.0 signal (DP/DM)	Not actually use USB3.1 signal, only use USB2.0 signal (DP/DM)	
		 USB1 USB1 RESET USER SW2 SW3 D46 FUSE1 LED6 LED7 LED8 FAIL PASS USBOK DC1 U3 SW1 ON OFF U1 5V J1 J3 USB2 U2 3.3V CN3 DIGITAL D20 D21 Aref

11.2. USB2

TH244A001 USB2 acted as USB device port. Users can initialize TH244A001 as a native USB HID Keyboard or native USB HID Mouse by using Keyboard.begin () or Mouse.begin ().

Through Arduino Keyboard.h to process keyboard input.

Through Arduino Mouse.h, users can easily process mouse by functions(), for example, defining five keys to represent the movement and click, then users can simulate the application of mouse movement and click and long press. Mainly using digitalRead () to read keys state. Users can also use joystick to process the mouse movement and click. (ADC detect joystick voltage to sense the position).

USB2 as device port, acted as a mouse or keyboard	

MG32F02U128 USB as a device

- USB2.0 full speed
  - Compliant with USB specification v1.1/2.0
- Supports USB suspend/resume and remote wake-up
- Support 8 configurable endpoints
  - Each endpoint support flexible In, Out, simultaneous In and Out operations
  - Support 7 configurable endpoints with relocated address value except endpoint-0
  - Support independently receive and transmit buffer with separated start address for each endpoint
  - Support double buffer mode for each endpoint independently
- Support control transfer for endpoint 0
- Support interrupt, bulk and isochronous transfer for all endpoints except endpoint-0
- Supports USB SRAM size 512 bytes shared for all endpoints
- Supports USB 2.0 power management

- Received and transmitted data are buffered with DMA capability for EP3,4

TH244A001 USB2 common use functions

Native USB HID Mouse usually has three buttons ● MOUSE_LEFT (default) left key ● MOUSE_RIGHT right key ● MOUSE_MIDDLE middle key	Mouse.begin ()	Enable mouse
	Mouse.end ()	Disable mouse
	Mouse.click (); Mouse.click (button)	mouse click event
	Mouse.press (); Mouse.press (button)	long press mouse button
	Mouse.release (), Mouse.release (button)	Release mouse button
	Mouse.isPressed (), Mouse.isPressed (button)	Mouse button state detection
	Mouse.move(x，y，wheel)	Mouse movement event
Native USB HID Keyboard	Keyboard.begin ()	Enable keyboard
	Keyboard.end ()	Disable keyboard
	Keyboard.press ()	Long press keyboard button
	Keyboard.release () Keyboard.releaseAll ()	Release button Release all buttons

	Keyboard.print () Keyboard.println ()	Print event Print event with ENTER
	Keyboard.write ()	Send a single tap instruction

See USB for details.

12. RTC

Arduino does not provide standard function about RTC. TH244A001 development kit provides a **class MG32x02z_RTC**, users can use the RTC as follows. Because RTC depends on the TH244A001 to run, when the power is off, RTC will stop. After the power is turned back on, the RTC needs to be reinitialized.

TH244A001 development kit provides a **class MG32x02z_RTC** to use the RTC functions. Generally enable the RTC, and then set the time and date. Read out the time information for needs.

When using RTC, should be done in two parts: initializing RTC and read time information.

Common use functions：

Initializing RTC:

```
/* This part of the initialization usually goes in setup */
Rtc.begin ();                               //Enable RTC
Rtc.setDate (day，  month，  year);          //Set date
Rtc.setTime ( hour，  minute，  sec);   //Set time
```

Read time information：

```
/*used as requirement*/
Rtc.getHour ();                             //get the current hour
Rtc.getMinute ();                           // get the current minute
```

```
        Rtc.getSecond ();                    // get the current second

        Rtc.getDay ();                       // get the current date

        Rtc.getMonth ();                     // get the current month

        Rtc.getYear ();                      // get the current year
```

See RTC for details.

13. SLEEP/STOP Mode

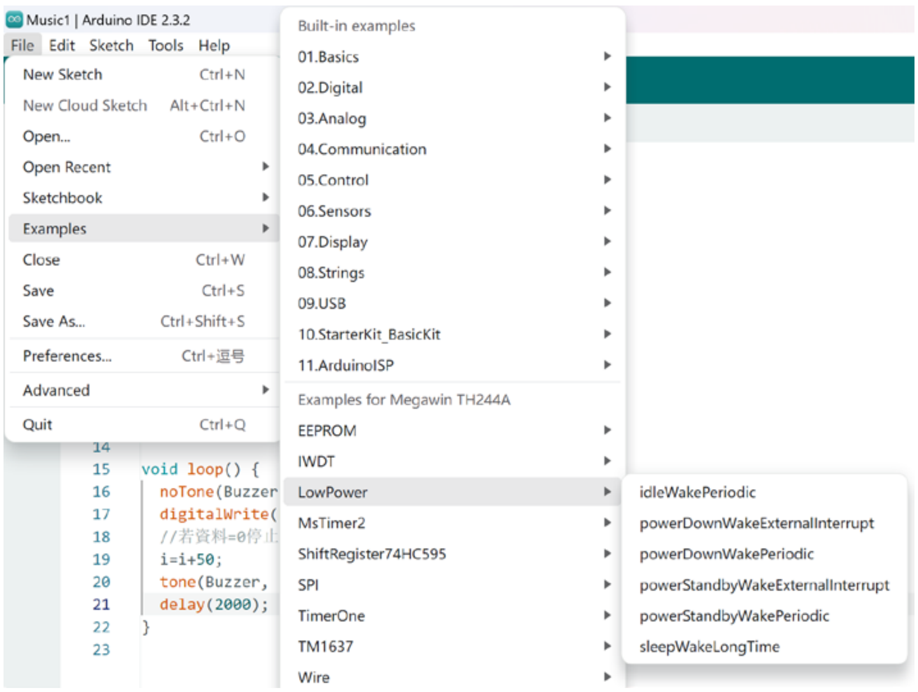
The Megawin M0 MCU MG32F02U128's power controller supports a total of three power operation modes: ON, SLEEP, and STOP. Two of the power-down modes are SLEEP mode and STOP mode. The power-down modes can reduce the power consumption of the chip and provide a variety of power-saving solutions for the chip application. Arduino development kits since version 2.2.0 support the power-down modes for use in the Arduino IDE.

- ON mode：In ON mode, the CPU can run at full speed, and all peripherals can work normally at full power. At the same time, these modules can be enabled and disabled independently to reduce power consumption.
- SLEEP mode: In SLEEP mode, only the CPU will be frozen, and all peripherals can be set to continue working or sleep. In this mode, the chip can be woken up by an associated interrupt or event.
- STOP Mode: STOP mode provides the lowest power consumption, and differs from SLEEP mode in that the CPU enters deep sleep mode and all peripherals are disabled except for some special modules or devices. These special modules or devices can be set to continue to work or not in STOP mode, these special modules are IWDT, RTC, CMP, LVR, BOD0, BOD1, BOD2, and the internal voltage regulator will also run in low

power mode. In this mode, the chip can be woken up by some external input lines (GPIO) and some event detection.

Note: The development kit example program is designed for STOP mode and SLEEP mode wake-up, currently only external pin interrupt wake-up and IWDT (open door dog wake-up, using ILRCO 32KHz as a clock) examples are designed.

Example program description



```
void loop() {
  noTone(Buzzer
  digitalWrite(
  //若資料=0停止
  i=i+50;
  tone(Buzzer,
  delay(2000);
}
```

example program	mode	Basic instructions
idleWakePeriodic.ino	**STOP**	**Enter STOP mode for a sustained period of time and wake up automatically**

powerDownWakeExternalInterrupt.ino	**STOP**	**Enter STOP mode and wake up via external interrupt**
powerDownWakePeriodic.ino	**STOP**	**Enter STOP mode, wake up via IWDT watchdog**
sleepWakeLongTime.ino	**STOP**	**Enter STOP mode, wake up via IWDT watchdog**
powerStandbyWakeExternalInterrupt.ino	**SLEEP**	**Enter SLEEP mode and wake up via external interrupt**
powerStandbyWakePeriodic.ino	**SLEEP**	**Enter SLEEP mode, wake up via IWDT watchdog**

example program	mode	Default	optional	instruction
idleWakePeriodic.ino	**STOP**	**ADC_OFF BOD_OFF USB_OFF**		**Actual operation is only IWDT, RTC, plus IO interrupt for wakeups**
powerDownWakeExternalInterrupt.ino	**STOP**	**ADC_OFF BOD_OFF USB_OFF**	**ADC_OFF BOD_OFF USB_OFF**	**In addition to the optional modules, the**

				only practical operation is IWDT, RTC, plus IO interrupts for wake-up.
powerDownWakePeriodic.ino	STOP	ADC_OFF BOD_OFF USB_OFF	ADC_OFF BOD_OFF USB_OFF	In addition to the optional modules, the only practical operation is IWDT, RTC, plus IO interrupts for wake-up.
sleepWakeLongTime.ino	STOP	ADC_OFF BOD_OFF USB_OFF	ADC_OFF BOD_OFF USB_OFF	In addition to the optional modules, the only practical operation is IWDT, RTC, plus IO interrupts for wake-up.

powerStandbyWakeExternalInterrupt.ino	**SLEEP**	**ADC_OFF USB_OFF SYS_TICK**	**ADC_OFF USB_OFF**	**In SLEEP mode, practically all modules are in working state and theoretically all modules can be woken up**
powerStandbyWakePeriodic.ino	**SLEEP**	**ADC_OFF USB_OFF SYS_TICK**	**ADC_OFF USB_OFF**	**In SLEEP mode, practically all modules are in working state and theoretically all modules can be woken up**

To use these features, you must add the LowPower.h

#include "LowPower.h"

See Common Functions and Basic Usage SLEEP/STOP mode for details.

14. IWDT

14.1. Independent Watchdog Timer

When the watchdog timer is enabled, software needs to always reset the timer before the timer times out. When the watchdog timer is reset, the timer will reload the 0xFF value and start timing again. If the chip is out of control due to tampering, the firmware may fail to reset the timer and cause the timer to time out, which will cause the IWDT to generate a reset event and send it to the Reset Source Controller (RST) to be reset as a hot reset or cold reset.

It is also possible to apply wdt_disableRST () to set the watchdog interrupt not to perform a system reset, but to execute the dedicated interrupt function ISR (WDT_vect) {}.

The development kit supports waking up or resetting the system via the IWDT in SLEEP or STOP mode.

IWDT Timing is based on ILRCO 32KHz。Support the following time constants.

```
 SLEEP_15MS = WDTO_15MS,        // watchdog timer 15ms timeout
SLEEP_30MS = WDTO_30MS,        // watchdog timer 30ms timeout
SLEEP_60MS = WDTO_60MS,        // watchdog timer 60ms timeout
SLEEP_120MS = WDTO_120MS,      // watchdog timer 120ms timeout
SLEEP_250MS = WDTO_250MS,      // watchdog timer 250ms timeout
SLEEP_500MS = WDTO_500MS,      // watchdog timer 500ms timeout
```

```
  SLEEP_1S    = WDTO_1S,         // watchdog timer 1s timeout
  SLEEP_2S    = WDTO_2S,         // watchdog timer 2s timeout
  SLEEP_4S    = WDTO_4S,         // watchdog timer 4s timeout
  SLEEP_8S    = WDTO_8S,         // watchdog timer 8s timeout
```

14.2. IWDT dedicated interrupt

```
ISR (WDT_vect)
{
  // do anything
   wdt_reset ();     //Feed the dog. Reset the timer.
}
```

14.3. IWDT library main usage

Main usage of IWDT library based on Megawin MG32F02U128 design:

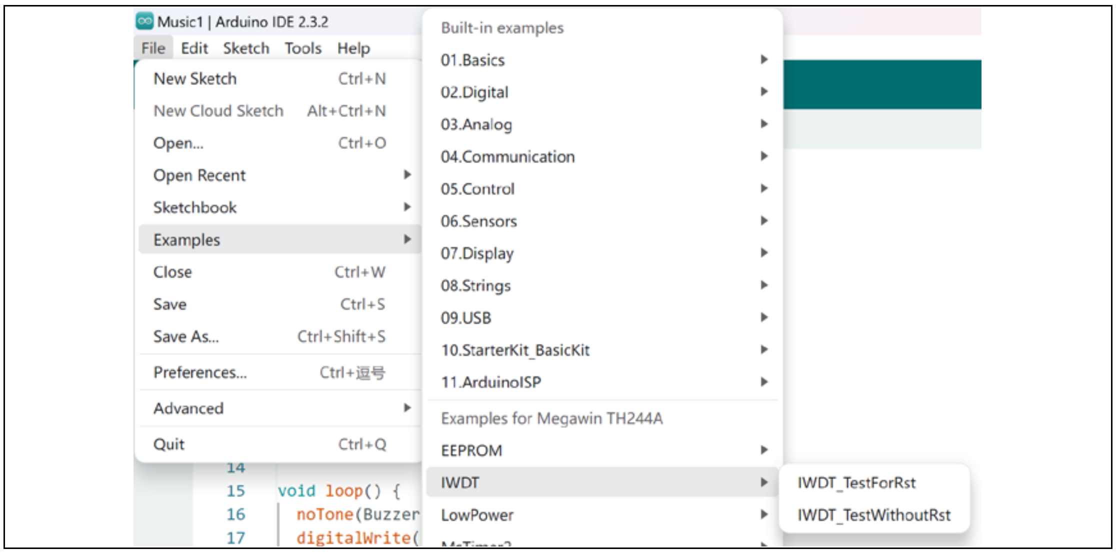
☐ wdt_disableRST();//disable IWDT interrupt reset system function, generally used in combination with ISR(WDT_vect){}.

☐ wdt_enableRST(); //Enable IWDT interrupt reset system function, feed the dog on time, when the MCU system program running or dead, can not feed the dog, to achieve the use of reset system.

☐ wdt_enable(WDTO_4S);//start the watchdog function, timer timeout for WDT0_4S, that is, 4s

☐ wdt_disable() //disable watchdog function

☐ wdt_reset(); // feed the dog, beyond the timeout, no feed the dog, it will enter the IWDT exclusive interrupt ISR(WDT_vect){} or RESET MCU



```
void loop() {
  noTone(Buzzer
  digitalWrite(
```

See Common Functions and Basic Usage for details IWDT.

15. Timer interrupt-TimerOne

15.1. TimerOne(introduce)

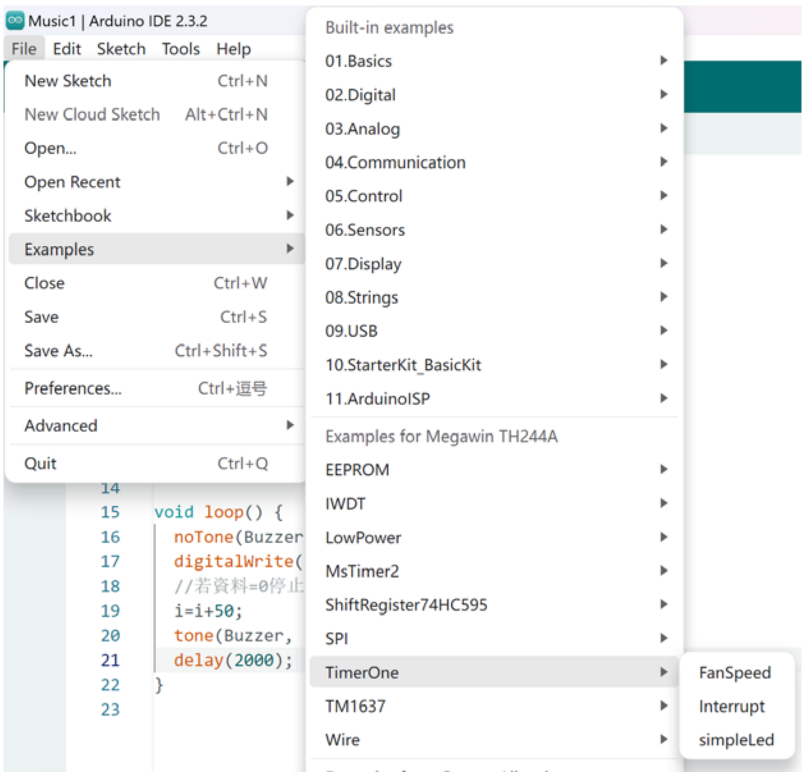
TimerOne library is to use MCU internal timer to realize flexible time interval control. Timer interrupt TimerOne library is supported from development kit 2.3.0. The basic functions and usage are the same way as Arduino's current platform. The library file is only for MG32F02U128 platform-

TH244A001.

The TimerOne library supports the setting of PWM signal output from D42/D43. When TimerOne is used, the PWM function of D10/D11 cannot be used.

TimerOne Library support for interrupt setup interrupt functions。

sample program:



```
void loop() {
  noTone(Buzzer
  digitalWrite(
  //若資料=0停止
  i=i+50;
  tone(Buzzer,
  delay(2000);
}
```

15.2. Timer One library control output PWM

const int pin_PWM = 43; // Setting definition needs to set the output pin of PWM, only 42 and 43 can be used; no need to output PWM, can not be set.

Timer1.initialize (timeout); //unit us, timeout=40 us, it means PWM frequency is 25 KHz;

Timer1.pwm (pin_PWM, duty); //this time the duty is 0~1023. Actually corresponds to the positive duty cycle of 0~100%.

15.3. TimerOne library timing interrupt

Timer1.initialize(timeout); // unit us, timeout=1000000us , initialize the timer for 1s; here the time can support a maximum of 10s; generally recommended for use as an interrupt, here the time is set to a maximum of 1s to ensure time accuracy, the need for a larger time can be carried out using cycle counting;

Timer1.attachInterrupt (timer1_ISR); //Set the interrupt callback function

void timer1_ISR () {} //Timer interrupt handling function

To use these functions, you must load the TimerOne.h library

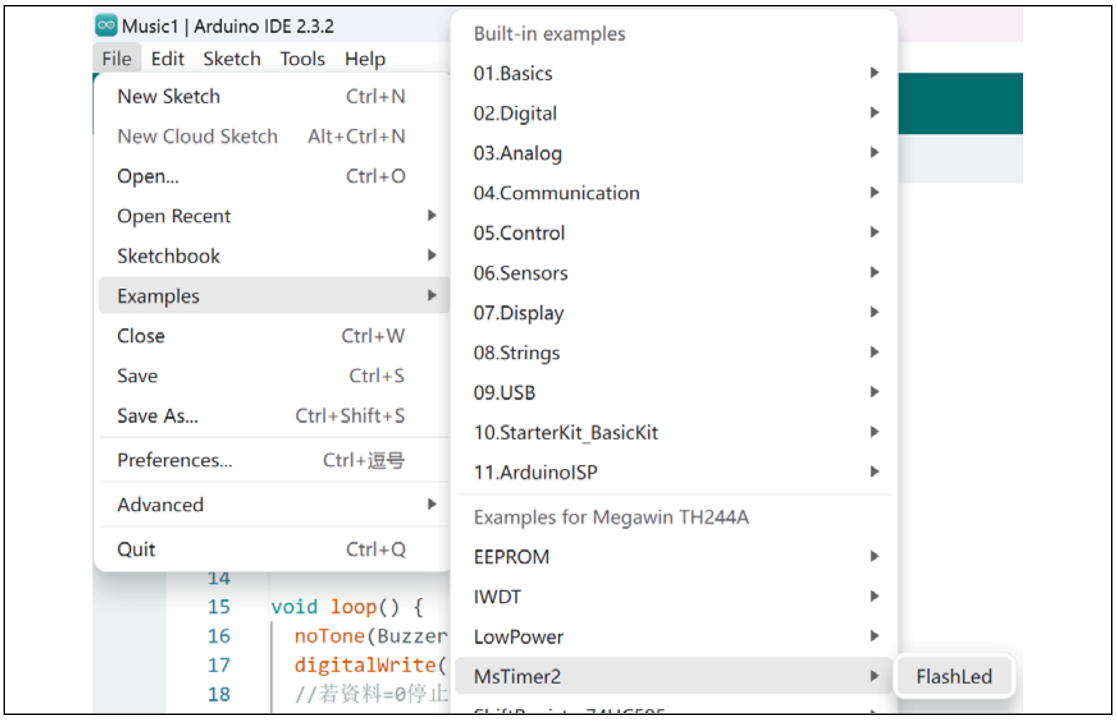
#include <TimerOne.h>

See TimerOne for more details on common functions and basic usage.

16. Timer interrupt Mstimer2

MsTimer2 library is to use MCU internal timer to realize flexible time interval control. Timer interrupt Mstimer2 library is supported from development kit 2.3.0. The basic functions and usage are the same way as Arduino's current platform. The library file only applies to MG32F02U128 platform-TH244A001.

When using MsTimer2 library timing interrupt, the PWM function of D3/D6 can not be used.



```
void loop() {
  noTone(Buzzer
  digitalWrite(
  //若資料=0停止
```

Library Function

MsTimer2:set (unsigned long ms, void (*f) ());

//Set the interval of timing interrupt and the interrupt service program to be called.

//ms indicates the length of the interval of the timing time in ms; //void (*f).

//void (*f) () indicates the name of the called interrupt service program function.

MsTimer2::start (); //start timer interrupt

MsTimer2::stop (); //stop the timer interrupt, you can restart the interrupt at any time with the start method.

void (*f) () {} //Timer interrupt handler function

To use these functions, the MsTimer2.h library must be loaded.

#include <MsTimer2.h

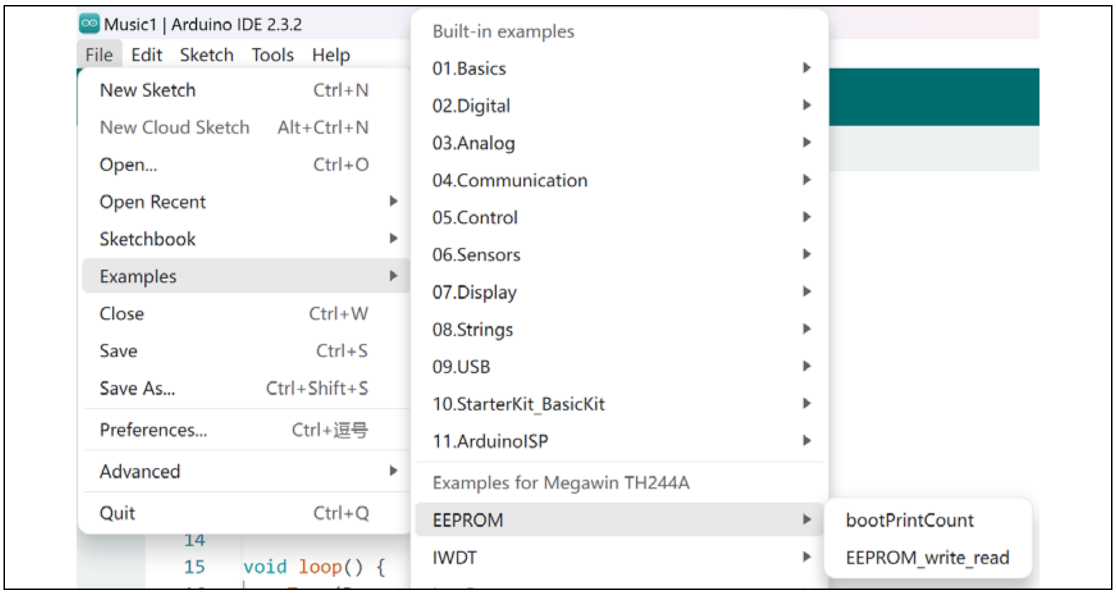
See MsTimer2 for more details on common functions and basic usage.

17. EEPROM

Megawin-MG32x02z-TH244A001-2.3.0 has an EEPROM that can be powered down without losing the stored data. The TH244A001 development kit has a design that divides the flash memory to emulate the Arduino's EEPROM because the emulated EEPROM will open up the same amount of space in the SRAM after initialization. The simulated EEPROM will open a space of the same size in SRAM after initialization. In order to utilize the limited SRAM memory efficiently, the TH244A001 uses the default 512 bytes Flash emulation as the EEPROM, and developers can manually modify the EEPROM.h file under the installation path of the development kit to realize it if they need more space. The EEPROM.h in the development kit only applies to the Megawin MG32F02U128 TH244A001 platform.

In particular, the flash memory and EEPROM support unlimited reading of their data, but limit the number of data write operations.

The EEPROM has a limit on the number of write operations, so be careful not to place write operations in loops or other cyclic execution functions.



```
void loop() {
```

An EEPROMClass object named EEPROM has been defined by default in the EEPROM.h library. The basic way to use it is as follows:

- Load #include <EEPROM.h> to use EEPROM;

The EEPROM library already has an object named EEPROM defined by default, with a default size of 512 bytes. You can just use this object directly. If you need to define an object with another name, you can use EEPROMClass to define it.

The default size of the defined EEPROM object is 512 bytes, and the user operable address is 0~511.

By default, when using EEPROM:

- first load EEPROM.begin (size), size is the maximum address of the data byte to be read or written + 1, taking value 1~512;
- Use EEPROM.write (addr, data) to write data, parameters are address &

single byte data respectively;

- After all write data, need to commit by EEPROM.commit () to save data from staging area to flash to realize power-down protection.
- Use EEPROM.read (addr) to read single-byte data at the specified address;

- Modify EEPROM capacity

If there is a need for larger capacity, users can modify the macros in the library EEPROM.h to realize it. Note that this macro can no longer be defined in the Arduino IDE ino project file.

Under the path of Arduino (depending on the local computer, x.x.x represents the actual version number used)

......\Arduino15\packages\megawin\hardware\MG32x02z\x.x.x\libraris\EEPROM\src to modify the EEPROM_SIZE macro parameter in EEPROM.h.

Support selectable from 512 bytes to 8K bytes, default setting is 512 bytes.

```
//===========Support EEPROM_SIZE list begin============
#define EEPROM_SIZE 512     //default set EEPROM 0.5K        512 bytes
//#define EEPROM_SIZE 1024  //set EEPROM 1K                  1024 bytes
//#define EEPROM_SIZE 1536  //set EEPROM 1.5K                1536 bytes
//#define EEPROM_SIZE 2048  //set EEPROM 2K                2048 bytes
//#define EEPROM_SIZE 2560  //set EEPROM 2.5K                2560 bytes
//#define EEPROM_SIZE 3072  //set EEPROM 3K                3072 bytes
```

```
//#define EEPROM_SIZE 3584   //set EEPROM 3.5K        3584 bytes
//#define EEPROM_SIZE 4096   //set EEPROM 4K          4096 bytes
//#define EEPROM_SIZE 4608   //set EEPROM 4.5K        4608 bytes
//#define EEPROM_SIZE 5120   //set EEPROM 5K          5120 bytes
//#define EEPROM_SIZE 5632   //set EEPROM 5.5K        5632 bytes
//#define EEPROM_SIZE 6144   //set EEPROM 6K          6144 bytes
//#define EEPROM_SIZE 6656   //set EEPROM 6.5K        6656 bytes
//#define EEPROM_SIZE 7168   //set EEPROM 7K          7168 bytes
//#define EEPROM_SIZE 7680   //set EEPROM 7.5K        7680 bytes
//#define EEPROM_SIZE 8192   //set EEPROM 8K          8192 bytes
//===========Support EEPROM_SIZE list end============
```

Commonly used functions:

EEPROM.begin (Size); //apply the size of the read/write operation, Size must be smaller than the size defined by EEPROM_SIZE.

Size must be smaller than the size defined by EEPROM_SIZE. For example, the default EEPROM_SIZE=512, then here Size is 1~512.

When you need to write or read a single byte:

EEPROM.write (address, data) //default EEPROM_SIZE=512, the available address is 0~511. Write method is to store bytes, data can only be 0x00~0xFF, used to store the read data;

EEPROM.commit (); //Enable to save data from SRAM to flash to realize power-down protection.

EEPROM.read (address) //read the specified address of the data, read () method is to read byte units

When you need to write or read multiple bytes, such as floating-point data or shaping data or other data structures (need to store data in multiple bytes), you can use the following methods to read and write:

EEPROM.put (address, data) //write data to the specified address, data is the data to be stored

EEPROM.get (address, data) //read the data at the specified address, data is the data to be stored.

To use these functions, the EEPROM.h library must be loaded.

#include <EEPROM.h

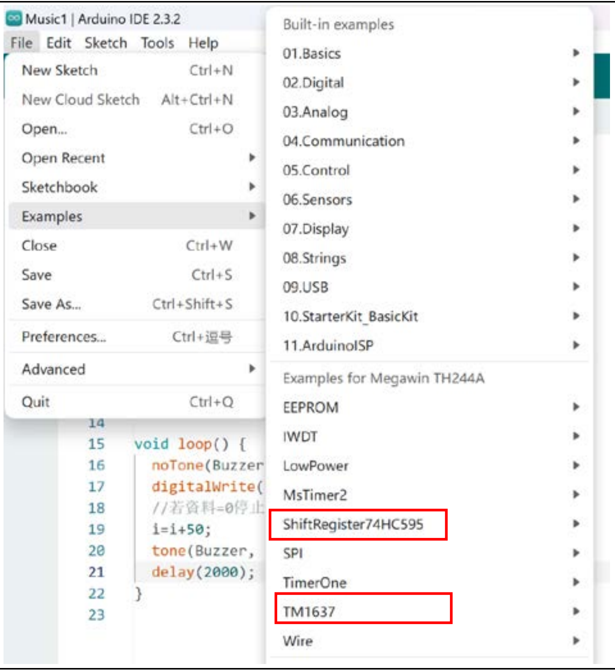
For more details, see Common Functions and Basic Usage of EEPROM.

18. Multi-segment digital tube library display

TM1637Display.h or ShiftRegister74HC595.h must be loaded to realize the display control of digital tube.

#include <TM1637Display.h> or #include <ShiftRegister74HC595.h>.

The basic operation can be referred to the example.

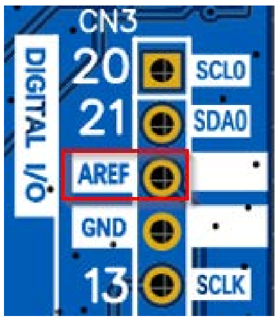


```
void loop() {
  noTone(Buzzer
  digitalWrite(
  //若资料=0停止
  i=i+50;
  tone(Buzzer,
  delay(2000);
}
```

19. Other Important Notes

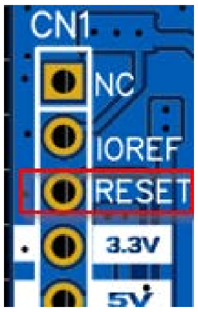
19.1. AREF (CN3)

Analog input reference voltage. Using J2 to switch VDD or AREF as VREF + input. Users can refer to the ADC part specifications. analogReference() is used to select whether the external VREF+ or IVR24 is the reference voltage. See the **ADC** section for details.

AREF is another input as VREF+. Users need to install 2.54 mm 2 pin jumper **J2** to use AREF
 CN3 DIGITAL I/O 20 SCL0 21 SDA0 AREF GND 13 SCLK

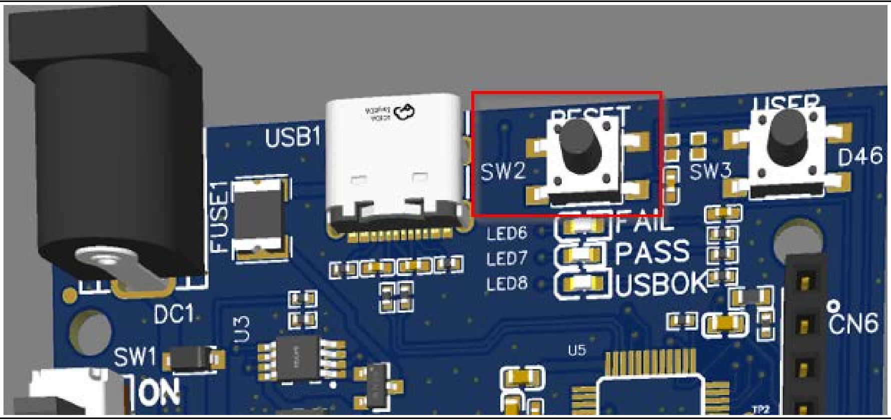
19.2. RESET PIN (CN1-RESET)

Connect to external modules' RESET signal to process external module reset MG32F02U128. It is the same as press RESET KEY on TH244A001.

RESET located at CN1, used to be connected to external signal to reset MG32F02U128.	

19.3. RESET KEY: SW2

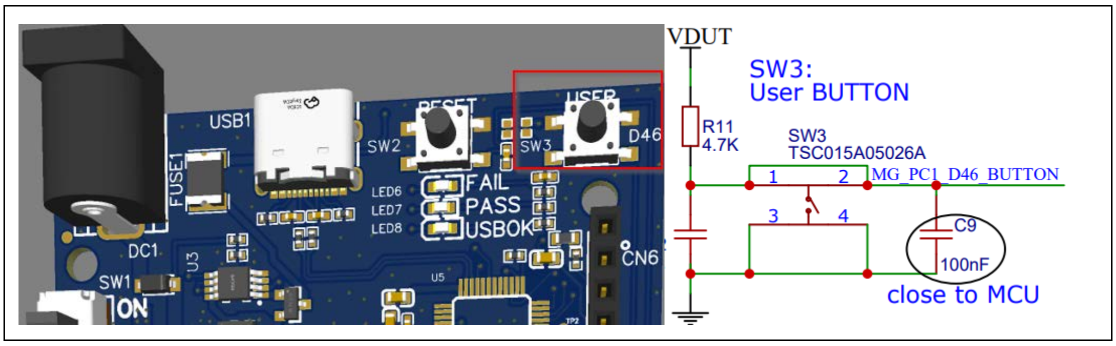
Press RESET KEY SW2 to reset MG32F02U. SW2 is a reset signal which connected to PC6 RSTN pin. When SW2 is pressed, MCU will be reset.

Press the SW2, RESET signal is pulled low and generate a reset signal. Pressed: LOW level; No pressed:HIGH level
 RESET USER USB1 SW2 SW3 D46 FUSE1 LED6 LED7 LED8 FAIL PASS USBOK DC1 U3 SW1 ON CN6 U5

19.4. USER KEY SW3

Connected to 46, users can use this button to do a button function. Hardware has been designed a pull-up resistor, users can using 46 to read the button state.

Users can program the button SW3, using Arduino function pinMode(46， INPUT) to set the pin to digital input, and using digitalRead(46) to read the state. Pressed: LOW level; No pressed:HIGH level



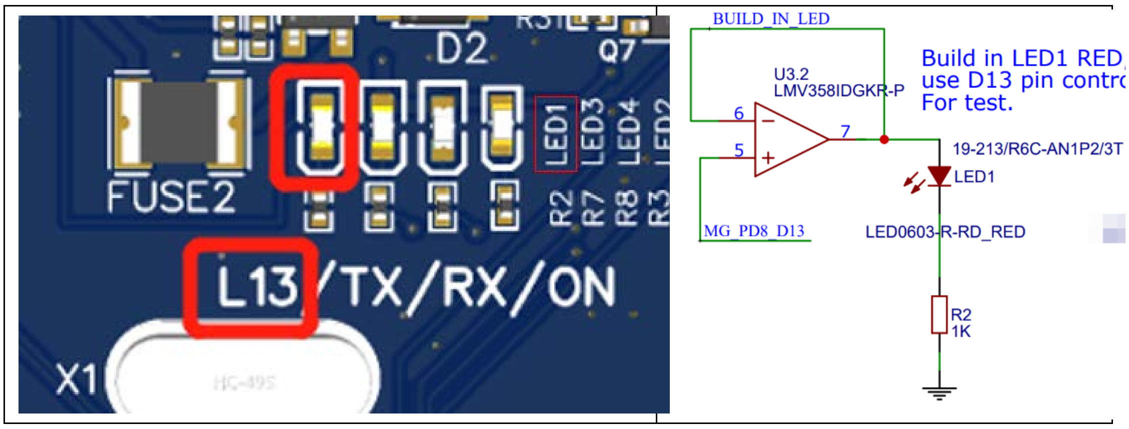
19.5. L13:　LED1

Pin 13, macro defined as LED_BUILTIN. Users can use 13 or LED_BUILTIN in code; it is connected to a red LED named LED1 through a comparator.

Pin 13 can also be configured as SCLK for SPI. op-amps are utilized in the line to minimize the impact of the LED1 load on the SPI function of pin 13.

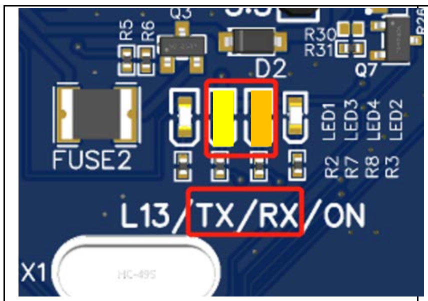
When pin 13 is high level, LED1 turn on while low level to turn off.

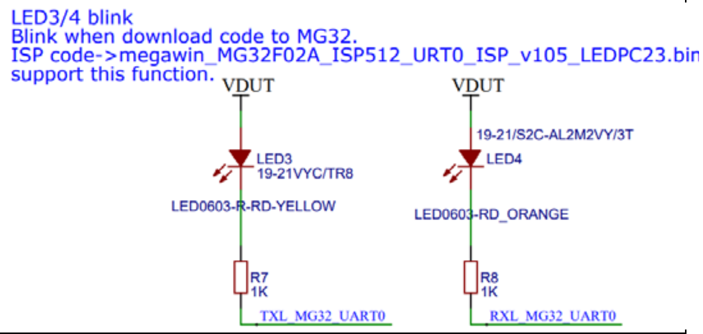
Factory default program: Red LED1 flashes with ON 1s, OFF 1s



19.6. TXLED/RXLED

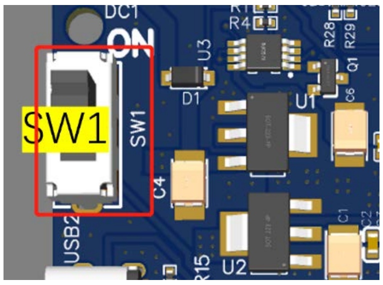
When upgrading MG32F02U128 AP code through PC, YELLOW LED3 /ORANGE LED4 keep flashing, upgrade is complete, LED3/4 will be turn off.

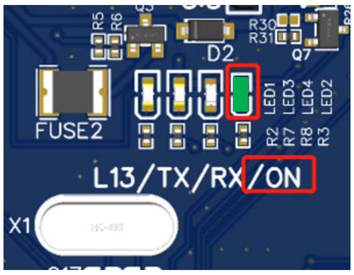


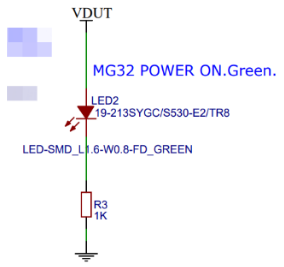


19.7. MCU power switch SW1 / Power ON LED2

When users need to power off or repower MCU, can switch SW1. Green LED2 indicated whether MG32F02U is power on or off.

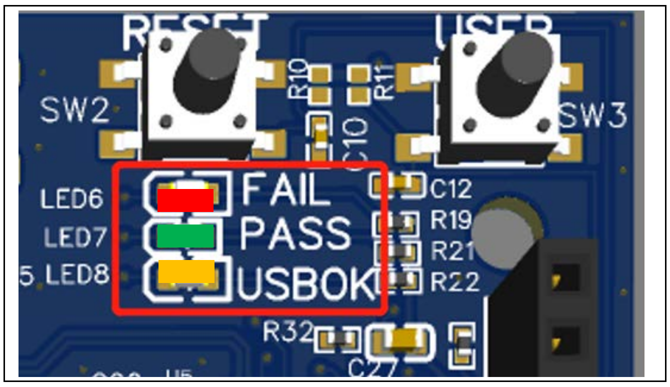






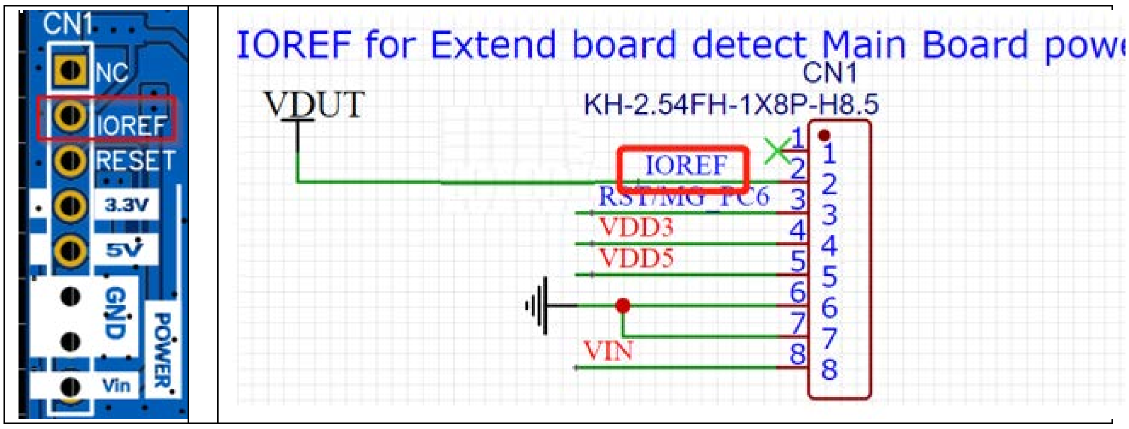
19.8. LED6/7/8 indicated MLink connection state between USB1 and PC

GREEN LED7/ORANGE LED8 is turned on if the connection of USB1 is normal while RED LED6 is turned on if connection abnormal.



19.9. IOREF

The pin connected to Arduino MCU MG32F02U reference voltage. Usually used to read by Arduino extension board and choose appropriate operate voltage (which is the function of extension board), or provides level shift of 3.3V or 5V. IOREF voltage is the same as J1 selected voltage 5V or 3.3V.



19.10. Development kit path management

Third party development kit should be placed in the specified path to make sure Arduino IDE can identify and load correctly.

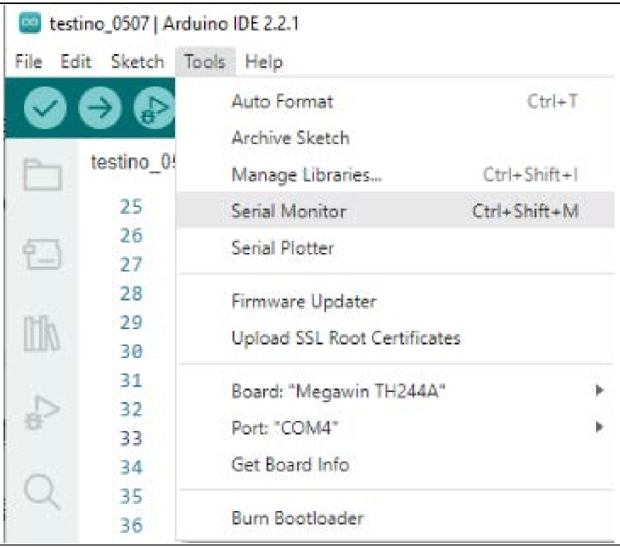
Commonly used Arduino AVR development boards, such as UNO , MEGA , NANO and other official AVR development boards, will not automatically download the official development kit when installing Arduino IDE2.x.x , and need to manually download and install (Arduino IDE1.8.x version will be downloaded and installed by default). Refer to **TH244A001_UserManual**- Official AVR development kit installation for details.

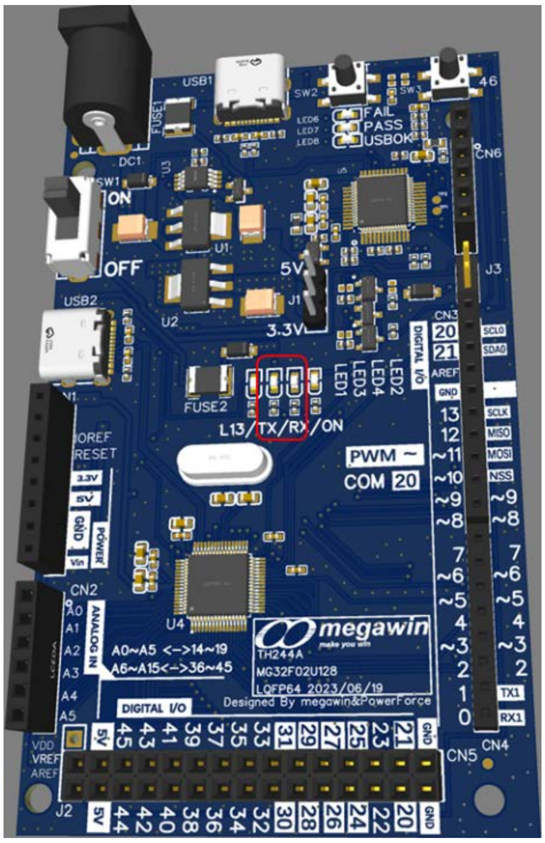
The development of the third party development board, such as STM32 Nucleo Nu_edu_Arduino, Megawin TH244A001, can not direct identified by IDE, users need to install special development kit. The unofficial development board is called the development board contributed by the third party, and the development kit needs to be setup and installed manually. Refer to **TH244A001_UserManual**- third party development board’ s development boards installation and update for details.

19.11. Virtual com setting and program download

TH244A001 use MG84FG516 to download program via USB1. Configure MG84FG516 VCP mode to enable virtual com port (Megawin provides SHA-1 and SHA-2 drivers). Short J3, is to enable virtual com VCP.

MG32F02U128 UART0 is connected to MG84FG516 to do program download and Arduino IDE com debug. Arduino IDE provides a serial monitor, which can transfer and receive data through UART to be the main mean of debugging. TH244A001 RX TX LED will keep flashing when the program is downloading through ARDUINO IDE.

Serial monitor' s main function is to use print functions like Serial.printIn (). Print special definition of information on PC through UART0 to achieve debugging.




TX LED/RX LED blinks during burning.

MG84FG516 MCU through SWD to communicate with MG32F02U128, can achieve real-time debugging (Arduino IDE2.X is only support the official development board). TH244A001 as a third party development board, cannot directly use Arduino IDE2.X to do SWD debugging. But, TH244A001 still can use other IDE to achieve SWD debugging.

20. TH244A001 reset

1. When Arduino IDE is uploading program, TH244A001 will be reset;
2. Press SW2（RESET） button, TH244A001 will be reset. Refer to RESET.
3. Repower TH244A001, TH244A001 will be reset;
4. Send Low level signal to CN1 RESET pin, TH244A001 will be reset, it is the same as 2;

21. Common use function and basic usage

21.1. GPIO read and write control

Digital ports read/write handler.

Common use functions, refer to Arduino official instructions.

Function: pinMode(pin，MODE)

Comment: Set pin MODE to OUTPUT (output mode), INPUT (input mode), INPUT_PULLUP (input mode with pull-up resistor)

Reference: https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/

Function: digitalWrite(pin， value)

Comment: Write level to pin; value: HIGH (voltage: VDD), LOW (voltage: 0)

Reference: https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

Function: digitalRead (pin)

Comment: Read pin level state; return value is HIGH or LOW

Reference: https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/

21.2. Delay function

Function: delay (ms)

Comment: Specified delay in milliseconds

delay (ms)

Function: delayMicroseconds (us)

Comment: Specified delay in microseconds

delay (), delayMicroseconds (us) will occupy MCU resources and block MCU execution.

Example 1: Using delay to do LED flash

```
//LED flashing control by delay
void setup () {
  // Initialize 13 pin as output, 13 macro define is LED_BUILTIN
  pinMode (LED_BUILTIN, OUTPUT);
}
void loop () {
  digitalWrite (LED_BUILTIN, HIGH); // Output high, turn on LED
  delay (1000);                     //Keep high for 1s
  digitalWrite (LED_BUILTIN, LOW);  // Output low, turn off LED
  delay (1000);                    //Keep low for 1s
}
```

21.3. Timer

Data of millis () or micros () is large and the variable is defined as unsigned long. In most cases, using the ms level is enough.

Function: millis ()

Comment: Return since TH244A001 to run the current program the number of milliseconds. Non-blocking programming must. This is the best way to replace delay ().

Function: micros ()

Comment: Return since TH244A001 to run the current program the number of microseconds. If users need more resolution, micros () can be used.

21.4. PWM

Function: analogWrite(pin， Value)

Comment: Set PWM duty of specified pin. (TH244A001 pin 3, 5, 6, 8, 9, 10, 11 support PWMfunction), default resolution is 8 bit, value can be 0~255 for duty is 0%~100%;

Example: analogWrite (3, 25)   pin 3 output duty 25/255=10%

AnalogWrite (10,230)   pin 10 output duty 230/255=90%

Reference:

https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/

Function: analogWriteFrequency(PWM_FRQ_XX，Fre)

Comment: Configure specified pin PWM frequency. Fre can be 300Hz~5000Hz, default frequency is 1 KHz.

Beyond the range of values, will be constrained for 300 Hz (less than 300 Hz) or 5000 Hz (above 5000 Hz).

Note：

TH244A001 only pin 3/6, 5/8/9, 10/11 can use this function to set frequency.

3/6(TM20), 5/8/9(TM36) and 10/11(TM26) can configure to different frequency.

Default frequency is 1000Hz;

For example:

Method one

- analogWriteFrequency(PWM_FRQ_D3_D6，Frq)   Set 3/6 frequency
- analogWriteFrequency(PWM_FRQ_D5_D8_D9，Frq)   Set 5/8/9 frequency

- analogWriteFrequency(PWM_FRQ_D10_D11，Frq)   Set 10/11 frequency
- analogWriteFrequency(PWM_FRQ_ALL，Frq)   Set 3/6, 5/8/9, 10/11 frequency

Example: Set pin 3 output PWM 3 KHz, duty 90%; pin 6 output PWM 3 KHz, duty 10%

```
void setup () {
  analogWriteFrequency (PWM_FRQ_D3_D6, 3000);   //Set PWM_FRQ_D3_D6
output frequency 3000Hz
}
void loop () {
  analogWrite (3, 230);   //Set pin 3 duty 230/255=90%
  analogWrite (6, 25);    // Set pin 6 duty 25/255=10%
}
```

Method two

- analogWriteFrequency(3，Fre)   Set pin 3 frequency while pin 6 is also set to this frequency
- analogWriteFrequency(5，Frq)   Set pin 5 frequency while pin 8, 9 is also set to this frequency
- analogWriteFrequency(10，Frq)   Set pin 10 frequency while pin 11 is also set to this frequency
- analogWriteFrequency(PWM_FRQ_ALL，Frq)   Set pin 3/6，5/8/9，10/11 frequency

Example: Set pin 3 output PWM 3 KHz, duty 90%; pin 6 output PWM 3 KHz, duty 10%

```
void setup () {
  analogWriteFrequency (3, 3000);   // Set pin 3 frequency while pin 6 is also set to
this frequency
}
void loop () {
  analogWrite (3, 230);   // Set pin 3 duty 230/255=90%
  analogWrite (6, 25);    // Set pin 6 duty 25/255=10%
}
```

21.5. DAC

Function: analogWrite(21， Value)

Comment: Write analog value to pin 21. Using this function, pin 21 will be set to analog output mode. (TH244A001 only pin 21 support DAC output).

DAC resolution is 12 bit, so Value can be set to 0~4095, actual output voltage is 0~VDD

Example 1:

```
void setup () {
}

void loop () {

    analogWrite (21,3000);   //modify different values, output voltage is

3000/4095*VDD

                              //When VDD=5V, output voltage 3.6V; when VDD=3.3V, output

voltage 2.38V }
```

21.6. ADC

Function: analogRead (pin)

Comment: return value read from analog pin. “pin” is the specified analog

input pin. analogRead (A0). TH244A001 analog pin are A0~A15

A0~A5 corresponding 14~19

A6~A15 corresponding 36~45

Reference:

https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

Example 1: Default resolution is 12 bit; VDD=5V

```
void setup () {
  Serial.begin (9600);   // initialize UART
}
void loop () {
  int sensorValue = analogRead (A0);                 //read A0 analog input, return
value is ADC value, value range is 0~4096 (12 bit resolution)
  float voltage = sensorValue * (5.0 / 4096.0);   //VDD=5V, default 12 bit
resolution, count actual voltage
  Serial.println (voltage);                             //print data for debug
}
```

Function: analogReadResolution (res);

Comment: Set ADC sampling resolution;

Parameter can be: 8, 10, and 12. Default is 12 bit for compatible with AVR official board. User can set to others for actual requirement to get higher precision. Pay special attention to the mapping range when calculating the actual voltage or using map ().

8 bit 0~255   255 corresponds to the ADC reference voltage, default is external VDD=5v

10bit 0~1023 1023 corresponds to the ADC reference voltage, default is external VDD=5v

12bit 0~4095 4095 corresponds to the ADC reference voltage, default is external VDD=5v

Reference:

https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogreadresolution/

Example 2: Set resolution from 12 bit to 10 bit, VDD=5V

```
void setup () {
  analogReadResolution (12);   //Set resolution to 12 bit
  Serial.begin (9600);         // initialize UART
}
```

```
void loop () {
  int sensorValue = analogRead(A0);              // read A0 analog input, return
value is ADC value, value range is 0~1023  (10 bit resolution)
  float voltage = sensorValue * (5.0 / 1024.0);  //VDD=5V, 10 bit resolution, count
 actual voltage.
  Serial.println (voltage);                      // print data for debug
}
```

Function: analogReference(res); Default is external VREF+, the parameter is *AR_DEFAULT (*AR_EXTERNAL*)*

Comment: Set ADC reference voltage;

Optional parameters are: *AR_INTERNAL, AR_EXTERNAL, AR_DEFAULT (AR_EXTERNAL)*

Note: Recommend to use the default external voltage source VREF+ as the reference voltage without modification.

Function: map (value, fromLow, fromHigh, toLow, toHigh)

Comment: map () usually used with ADC for linear mapping.

Reference:

https://www.arduino.cc/reference/en/language/functions/math/map/

Example 3: Using map () to change ADC data range from 0~5000 of 1000 to 0~255 of 51.

```
// change ADC data range from 0~5000 of 1000 to 0~255 of 51

int fromLow = 0;

int fromHigh = 5000;

int toLow = 0;

int toHigh = 255;

int value = 1000;

void setup () {

  Serial.begin (9600);   // initialize UART

}

void loop () {

  int x = map(value, fromLow, fromHigh, toLow, toHigh);   //Linear mapping：x =

a/(fromHigh-fromLow)*(toHigh-fromLow)

  Serial.println(value);              //print raw data range 0~5000 of value = 1000；

  Serial.println(x);                  //print new data range 0~255 of value x=

1000/(5000-0)*(255-0) =51

  delay (1000);                                       //delay 1s for watching data

}
```

21.7. UART

UART0 (download programs only)/ UART1/ UART2/ UART4/ UART5/ UART6/ UART7. Using Arduino **Serial library** to use UART.

Common use functions:

Function: Serial.begin (baud)

Comment: Initialize UART, set baud rate to 9600, 19200, 115200 and so on. Default 9600 bps.

Note: The serial port tool need to be set to the same baud rate to be able to communicate with MCU;

Example 1: Initialize UART

```
void setup () {
  Serial.begin (9600);   //Enable UART, baud rate 9600bps
}
void loop () {}
```

Function: Serial.println ()   print with ENTER; Serial.print () print without ENTER

Comment: UART print data for user requirement.

Function: Serial.available ()

Comment: Returning the number of bytes of data received by UART receive buffer

Function: Serial.read ()

Comment: Read data received from UART, read one byte at a time. Return -1 means no received data. Usually used with Serial.available (), using loop function to read buffer until finish read process.

Example 2: print when UART received data.

```
// print when UART received data;
void setup () {
  Serial.begin (115200);   //Set UART baud rate to 115200
}
void loop () {
  while (Serial.available ()) {           // while UART received data, Serial.available ()
return 1 for condition judgment
    char serialData = Serial.read ();    // use read function to read data, one byte at
 a time
    Serial.println ((char) serialData);   // print data which read from UART by read
function
  }
}
```

Example 3: UART0 received data, send to UART1 print out; UART1 received data, send to UART0 print out;

```
//TH244A001 UART0 and UART1
//Serial.begin () initialize UART0, connect to PC;
```

```
//Serial1.begin () initialize UART1, using serial port tool to connect to PC or other
development boards' UART;
//Judge from Serial.available () whether UART0 receive buffer is not empty;
// Judge from Serial1.available () whether UART1 receive buffer is not empty;
// When there is data in the receive buffer, perform the read operation and print it
void setup () {
Serial.begin (9600);    // initialize UART0, set baud rate to 9600bps
Serial1.begin (9600);   // initialize UART1, set baud rate to 9600bps
}
void loop () {
// read data from UART0; send data to UART1 byte by byte;
  if (Serial.available ()) {
    int inByte = Serial.read ();
    Serial1.println (inByte, DEC);

    }
// read data from UART1; send data to UART0 byte by byte;
  if (Serial1.available ()) {
    int inByte = Serial1.read ();
    Serial.println (inByte, DEC);
    }
```

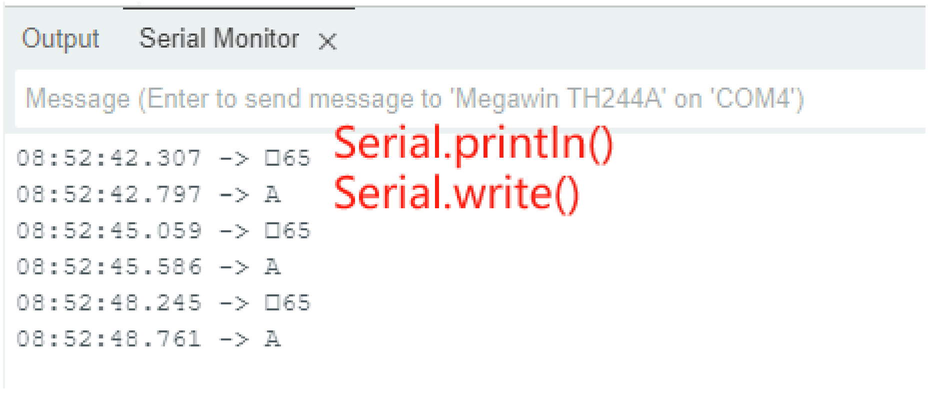
}

Function: Serial.write ()

Comment: Print one byte through UART, actually print ASCII code

Example 4: Compare Serial.println () with Serial.write ()

```
// Compare Serial.println () with Serial.write ()

int num = 65;   // define a decimal number

void setup () {

  Serial.begin (115200);

  Serial.println (num); //print character 65

  delay (500);

  Serial.write (num);   // Print character A with ASCII 65

  Serial.println ();    //Output ENTER

}

void loop () {

}
```


Output
Serial Monitor ×
Message (Enter to send message to 'Megawin TH244A' on 'COM4')
Serial.println()
Serial.write()

```
08:52:42.307 -> □65
08:52:42.797 -> A
08:52:45.059 -> □65
08:52:45.586 -> A
08:52:48.245 -> □65
08:52:48.761 -> A
```

Note:

- Serial means UART0
- Serialx means UARTx X : 1, 2, 4, 5, 6, 7

See Arduino official for details:

Serial - Arduino Reference

https://www.arduino.cc/reference/en/language/functions/communication/serial/

21.8. IIC

Generally, the development board is used as the master and the module is used as the slave.

When there are other development boards or devices as the master, TH244A001 can be added to the bus as a slave.

Wire means IIC0 while Wire1 means IIC1, IIC default clock is 100 KHz.

Function: Wire.begin () and Wire.begin (address)

Comment: Initialize the Wire library and add to the IIC bus. Wire.begin () initialize to be master while Wire.begin (address) to be slave. These functions can only be called once which usually placed in setup ();

Parameter:

Address: 7 bit device address(optional), if the parameter is empty, TH244A001 is set to master in IIC bus, while not empty to be slave in IIC bus;

Function: Wire.onReceive (receiveEvent), Wire is slave;

Comment: Slave responds master data. Register receiveEvent, when slave received master transfer, this function will be called. receiveEvent () should be used with int parameter (store slave received number of bytes), this function is not return anything.

Function: Wire.onRequest (requestEvent), Wire is slave;

Comment: master requests slave to send data

When master requests slave to send data, **slave** set function through onRequest.

Asserted a requestEvent, this function will be called when slave received master data request. Function requestEvent ()

Function: Wire.requrstFrom (address,quantity)   master requests slave data

Wire.requrstFrom (address,quantity,stop) master requests slave data with STOP character

Comment: master requests slave data, data can be received by master Wire.read () or Wire.available ().

Parameter:

address: 7 bit device address

Quantity: number of data (bytes)

stop: bool data, 1: Send STOP and release bus after request finish; 0: continue sending request

Return value:

Returning the number of data bytes from slave received;

Function: Wire.beginTransmission (address)

Comment: Master transfer a START to send buffer, and specifies the slave address to received data.

Parameter:

Address: slave 7 bit address

Return: no return value

Function: Wire.endTransmission () and Wire.endTransmission(stop)

Comment: STOP the slave transfer process which start from Wire.beginTransmission () and arrange by Wire.write (). Wire.endTransmission () can received a bool type variable stop. Value 1: endTransmission () transfer a STOP; Value 0: transfer a START

Return value

0 success

1 data overflow

2 slave received NACK when transfer address

3 received NACK when transfer data

4 other errors

Function: Wire.write (value)

Wire.write (string)

Wire.write (data, length)

Comment: Send data to slave, can be string, number or any other specified length data

Parameter:

value: value to send

string: pointer of string

data: data array

length: transfer length

Function: Wire.available ()

Comment: After master or slave received data, can use available function to receive the number of receive buffer data bytes

Function: Wire.read ()

Comment: After master or slave received data, can use Wire.read () to received data, usually used with Wire.available ();

Example 1: master transfer data, slave received and print

```
//TH244A001 test IIC0  IIC1 communications

//TH244A001 IIC0  SCL->20; SDA ->21; IIC1  SCL ->23; SDA ->22

//TH244A001 connect 20 to 23; 21 to 22; SDA install a 4.7KOhm pull-up resistor to
VDD (5V or 3.3V)
```

```
//Set TH244A001 IIC0 master, IIC1 slave; master sends specific length data to
slave;
#include <Wire.h>
void setup () {
  Wire.begin ();   // initialize IIC0 as master
  Wire1.begin (100);                  // initialize IIC1 slave ,address is 100; address
 must be 7 bit: 0~127.
  Wire1.onReceive (receiveEvent);   // regist receive event, receiveEvent () is the
function need to handled by users define;
  Serial.begin (9600);   // start UART for output
}
byte x = 0;   //range: 0~255
void loop () {
  Wire.beginTransmission (100);   // start to transfer data to slave which address is
 100;
  Wire.write ("x is ");            // transfer 5 bytes character type letters: x, space, i, s,
space.
  Wire.write(x);                   // transfer one byte, character type
  Wire.endTransmission ();   // perform transfer, transfer 6 bytes in total
  x++;
  delay (1000);   //send 6 bytes every 1s delay time
}
//define receiveEvent ()function, IIC1 received master data, then print out.
void receiveEvent (int howMany)   // int howmany muset be defined, howmany
here is actually counted received the number of bytes
{
  Serial.println ("howMany");
  Serial.println (howMany);         //print slave received number of bytes, here is 6
bytes.
  while (1 < Wire1.available())   // Wire1.available () check received number of
bytes, which is decremented by 1 each time Wire.read () is executed
                                  //set Wire1.available ()>1 to print the last byte as a
integer type;
                                  //or character type number will be recognized as ASCII
letters, not numbers via Serial.println ();
  {
    char c = Wire1.read ();   // read one byte of received data
    Serial.print (c);          // print one byte of received data
  }
  int x = Wire1.read ();   // print the last byte as an integer type
  Serial.println (x);       // print integer type
}
```

Example 2: master requests specific data from the slave, master receives the data, and the slave sends the data

```
// TH244A001 test IIC0   IIC1 communications

//TH244A001 IIC0   SCL->20;SDA ->21;IIC1   SCL ->23;SDA ->22

// TH244A001 connect 20 to 23; 21 to 22; SDA install a 4.7KOhm pull-up resistor
to VDD (5V or 3.3V)

// Set TH244A001   IIC0 master, IIC1 slave; master sends specific length data to
slave;

#include <Wire.h>

void setup () {

  Wire.begin ();                    // initialize IIC0 as master

  Wire1.begin (2);                  // initialize IIC2 as slave, address is 2

  Wire1.onRequest (requestEvent);   // regist master request slave data event

  Serial.begin (9600);              // initialize UART, for print information

}

void loop () {

  Wire.requestFrom (2, 8);   // used with master request event in the setup(),
transfer 8 bytes data to address 2 slave.

  while (Wire.available ())   // less than requested, will cause a problem.
// Wire.available () returns the number of bytes to receive buffer data and
decreased by reading by Wire.read () until return 0. Here using Wire.available () to
judge whether read operation is finished.
  {
    char c = Wire.read ();  // read one byte, and change to character type.
    Serial.print (c);       // print character
  }
  Serial.println ();
  Delay (500);
}
//master requests data event function. Slave prepares specific length data for
 master to read.
void requestEvent ()  //
{
  Wire1.write ("hello,TH244A001 ");  //Slave transfers data which master
requested.
}
```

21.9. SPI Master

TH244A001 is only supports SPI master mode. Generally slave are SPI Flash, SPI sensors or SPI display.

Function: SPI.begin ()

Comment: Enable and initialize SPI bus, set SCK, MOSI and SS as output, SCK and MOSI pull low; SS pull high; no return data;

Function: SPI.end ()

Comment: Disable SPI bus, but the pin mode set by initialization will be preserved so that the bus can be started again at any time. At this point, all devices on the SPI bus can no longer use the bus. No data is returned;

Function: SPI.beginTransaction （SPISettings(6000000, MSBFIRST, SPI_MODE0))

Comment: Configure transmission speed, data order and mode, no return data;

Define a **SPISettings** object, SPISettings object is used to configure SPI port for the SPI slave device transmission parameters. A single SPISettings object has speedMaximum dataOrder, dataMode of three parameters.

Example:

SPISettings SlaveA (6000000, MSBFIRST, SPI_MODE0) //Slave A transmission configuration;

SPI.beginTransaction（SlaveA） // Configure SPI to slave A 's required Settings;

Function: SPI. endTransaction ()

Comment: Disable SPI bus, specifically stop a slave device from using the bus, generally use this function after disable (pull high a slave device's CS pin) the slave. This operation does not affect other slave devices communicate with master. No data returned;

Function: SPI.transfer (val); SPI.transfer (val16); SPI.transfer (buffer, size)

Comment: Master transfer data to slave, and received slave data at the same time.

This function returns data.

SPI transmission are sent and received at the same time: received data returned with receivedVal (or receivedVal16). In the case of a buffer transfer, the received data is also stored in the buffer.

Parameter:

val: one byte length variable sent by bus

val16: two bytes length variable sent by bus

buffer: data array sent by bus

size: data length to be sent

Function: SPI.setClockDivider (SPI_CLOCK_DIV**8**)

Comment: Set SPI clock, no return data; SPI_CLOCK_DIV8 means SPI clock set to CPU clock divided by 8; Generally use default clock does not need to be modified.

21.10. USB

TH244A001 USB2 as device connected to PC, which can be used as a keyboard or mouse.

When the USB module is enabled in the program design, the TH244A001 will be connected to the computer as a keyboard or mouse input.

21.11. USB-mouse

TH244A001 USB2, which can be initialized for use as a mouse. Users can use the keyboard to simulate the mouse keys, users can also use the joystick to achieve the mouse movement function.

Requires #include <Mouse.h>

Function: Mouse.begin ()

Comment: Enable USB port as mouse

Function: Mouse.end ()

Comment: Disable mouse

Function: Mouse.click (); Mouse.click (button)

Comment: Mouse click event and release, equivalent to moments of the mouse click event, such as generally used left-click and right-click event.

Note: Button has three states, which represent the left button, right button and middle button of the mouse:

- Mouse.click (MOUSE_LEFT) -(default) left-click
- Mouse.click (MOUSE_RIGHT) right-click

- Mouse.click (MOUSE_MIDDLE)　middle-click

Function: Mouse.press (); Mouse.press (button)

Comment: Long press mouse button (without release). Usually used in time of long press left button to select or drag. It is usually used with Mouse.move (). Use Mouse.release () to release.

Note: Button has three states, which represent the left button, right button and middle button of the mouse:

- Mouse.press (MOUSE_LEFT) -default left button pressed
- Mouse.press (MOUSE_RIGHT)　right button pressed
- Mouse.press (MOUSE_MIDDLE)　middle button pressed
- Mouse.press (MOUSE_LEFT | MOUSE_RIGHT)　left and right button pressed at the same time

Function: Mouse.release (), Mouse.release (button)

Comment: Mouse button release. Release the button has been long pressed.Used with Mouse.press ().

Note: Button has three states, which represent the left button, right button and middle button of the mouse:

- MOUSE_LEFT (default)
- MOUSE_RIGHT
- MOUSE_MIDDLE

Function: Mouse.isPressed (), Mouse.isPressed (button)

Comment: Detect mouse button state. If a mouse button is pressed or Mouse.press() has sent a long press command, return ture.

Note: Button has three states, which represent the left button, right button and middle button of the mouse:

- MOUSE_LEFT (default)
- MOUSE_RIGHT
- MOUSE_MIDDLE

Function: Mouse.move(x，y，wheel)

Comment: Move mouse cursor a specified position, here relative. Cursor movement on the screen is always relative to the current position. Users should use Mouse.begin() before using Mouse.move(). Usually called in setup().

Parameter:

xVal: The amount of movement along x axis. Data type: signed char.

yVal: The amount of movement along y axis. Data type: signed char.

wheel: The amount of wheel movement. Data type: signed char.

Because all three parameters are signed char, the range is x (-127 to 127), y (-127 to 127) or wheel (-127 to 127).

Example 1: Use Mouse.h to perform Joystick function, with a switch (46). Control the mouse cursor movement by moving the joystick X axis and Y axis. And with a button function (default left mouse button).

//Use official file->sample->USB->Mouse->JoystickMouseControl to modify

```
//Use ADC detect joystick X/Y axis

//Joystick button acted as mouse left button

//Pin 13 connected to LED, represent mouse on/off state

//Pin 46 connected to USER KEY to switch USB mouse on/off state

//Joystick X/Y axis output connected to A0,A1; button of joystick connected to pin
  0;

#include "Mouse.h"

//Set joystick pins of axis, button and LED.

const int switchPin = 46;    // USER KEY to switch on/off mouse, different from
Arduino official.

const int mouseButton = 0;   // button of joystick connected to pin 0, set to mouse
  left button in this program, different from Arduino official.

const int xAxis = A0;         // Joystick X axis analog input

const int yAxis = A1;         // Joystick Y axis analog input

const int ledPin = 13;        //mouse control LED, LED in this program set to
  TH244A001 LED1. Enable mouse: LED turn on; Disable mouse: LED turn off;
  Different from Arduino official.

//define variables to read joystick data

int range = 21;               // X, Y axis the scope of the total mobile will be the
joystick here X or Y, the total amount of mobile level is defined as the 21; Different
from Arduino official.
```

```
int responseDelay = 5;       // mouse delay time, the parameter can be adjusted;
The smaller the mouse movement, the more sensitive it is;
int threshold = range / 7;  // Silence threshold is between positive and negative
threshold, do not do any reaction, as no mouse movement action occurs;
int center = range / 2;      // middle position
bool mouseIsActive = false;  // mouse state
int lastSwitchState = LOW;   // Default state for the switch
void setup() {
  pinMode(switchPin, INPUT);     // initialize the switch pin to input
  pinMode(ledPin, OUTPUT);        // initialize the mouse state LED control pin to
output
  pinMode(mouseButton, INPUT);  //
  Mouse.begin();                   //enable mouse function
}
void loop() {
  // Read the switch state, if there is a change and the latest state is HIGH, it
means the switch has pressed next. Change of state of HIGH->LOW->HIGH
  int switchState = digitalRead(switchPin);
  if (switchState != lastSwitchState) {      // switch state change is detected
    if (switchState == HIGH) {               // Determine HIGH after the state change
is complete
```

```
      mouseIsActive = !mouseIsActive;        // mouse state switch
      digitalWrite(ledPin, mouseIsActive);  // Mouse control LED state switch
    }
  }
  //keep new state of switch for next comparison
  lastSwitchState = switchState;
  // use ADC to detect joystick X, Y axis data
  int xReading = readAxis(A0);  //use readAxis() to handle X axis data; return
mouse X axis relative movement amount.
  int yReading = readAxis(A1);  // use readAxis() to handle Y axis data; return
mouse Y axis relative movement amount.
  //if mouse state is HIGH, then enable USB mouse movement operation.
  if (mouseIsActive) {
    Mouse.move(xReading, yReading, 0);  //0 means wheel is no change.
                                        //xReading positive: left movement; xReading
negative: right movement;
                                        //yReading positive: down movement; yReading
negative: up movement;
  }
  // judge whether joystick button is pressed (set to mouse left button in this
program), if not pressed the left button, then press the left button;
```

```
  if (digitalRead(mouseButton) == HIGH) {
    if (!Mouse.isPressed(MOUSE_LEFT)) {
      Mouse.press(MOUSE_LEFT);
    }
  }
  // else not pressed, send button
  else {
    // if the mouse is pressed, release it:
    if (Mouse.isPressed(MOUSE_LEFT)) {
      Mouse.release(MOUSE_LEFT);
    }
  }
  delay(responseDelay);
}
//define ADC sampling. Calculate X or Y axis relative movement;
  int readAxis (int thisAxis) {
  int reading = analogRead (thisAxis);   //read X or Y axis through ADC
  //use map () function to map ADC data from 0->1023 to 0->range
  reading = map (reading, 0, 1023, 0, range);
  // if the output reading is outside from the rest position threshold, use it:
  int distance = reading - center;
  if (abs(distance) < threshold) {
    distance = 0;
  }
  // return the distance for this axis:
  return distance;
}
```

21.12. USB-Keyboard

Function: Keyboard.begin ()

Comment: Enable keyboard. This function enable a virtual keyboard and connected to PC. Use Keyboard. End() to terminate the USB connection

Function: Keyboard.end ()

Comment: Disable keyboard

Function: Keyboard.press ()

Comment: Long press keyboard, to output button to PC. Use Keyboard.release () or Keyboard.releaseAll () to release. For example, long press UP button.

Function: Keyboard.release ()

Comment: Release a specific key, responds to Keyboard.press ()

Function: Keyboard.releaseAll ()

Comment: Release all keys

Function: Keyboard.print ()

Comment: Typing on the keyboard, simulating typing events, such as simulating text input, can be done using this command.

Function: Keyboard.println ()

Comment: Similar to Keyboard.print (), with ENTER typing event

Function: Keyboard.write ()

Comment: Send one keyboard keystroke command. Similar to Keyboard.print (), Keyboard.println (), but Keyboard.write () only sends one key command. Keyboard.println (), Keyboard.print () or Keyboard.write (), only send ASCII character, like: ABC, 123, space, ENTER.

Example 1: Enable keyboard, print numbers and characters

```
//Enable keyboard, use basic write (), print () and println ()
//write () can only print one byte, it's a number, it's the ASCII equivalent of a
character
//print (), println () can print numbers, letters, strings, etc
#include "Keyboard.h"
void setup() {
  Keyboard.begin ();   // Enable keyboard
}
void loop() {
  Keyboard.write (97);     //print ASCII 97:   a
  Keyboard.print ('\n');   //ENTER
  Keyboard.write ('a');    //print character   a
  Keyboard.print ('\n');   // ENTER

  Keyboard.print (97);     //print 97

  Keyboard.print ('\n');   // ENTER

  Keyboard.print ('a');    // print character a

  Keyboard.print ('\n');   //ENTER

  Keyboard.println (97);   // print 97

  Keyboard.print ('\n');   //ENTER

  Delay (5000);                  //delay 5s

}
```

Example 2:

```
//Detect whether USER KEY 46 is pressed, if pressed, then keep print Hello,
TH244A001! through USB2 keyboard function.
//note: if PC keyboard has already set to Caps Lock, will output different mode.
//note: preset input mode to English mode
//For example, Caps in small letter this program prints Hello, TH244A001!
// For example, Caps in capital letter this program prints hELLO, TH244A001!
#include <Keyboard.h>
int pinpin = 46;
void setup () {
  //Because pin 46 already has pull-up resistor, initialize with INPUT instead of
INPUT_PULLUP;
  pinMode (pinpin, INPUT);   // Use pinMode(pinpin, INPUT_PULLUP) if connecting
a GPIO without external pull-up resistor;
  Keyboard.begin ();
}
void loop() {
  //if the button is pressed
  if (digitalRead(pinpin) == LOW) {
    //Send the message
    Keyboard.print("Hello,TH244A001!");
  }
}
```

Example 3: Detect whether USER KEY is pressed, enable keyboard output. Output the number of pressed button time; Used Keyboard.begin(), Keyboard.print(), Keyboard.print();

```
// Detect whether USER KEY is pressed, enable keyboard output and output the
number of pressed button time;
#include "Keyboard.h"
const int buttonPin = 46;          // connected button pin, TH244A001 USER KRY is
 pin46
int previousButtonState = HIGH; // button state, TH244A001 USER KEY is already
install pull-up resistor. Press the button, level will change to low from high.
int counter = 0;                    // counter of keystroke time, for recording keystroke
time
void setup() {
  // initialize button pin, use INPUT_PULLUP if there is no pull-up resistor in the
circuit
  pinMode(buttonPin, INPUT);
  // initialize keyboard function
  Keyboard.begin();
}
```

```
void loop() {
  // read button state
  int buttonState = digitalRead(buttonPin);
  // if button state changed, and the current button state is HIGH level, the
process of pressed and released, pin 46 level should be changed
HIGH->LOW->HIGH;
  if ((buttonState != previousButtonState) && (buttonState == HIGH)) {
    // button counter increase
    counter++;
    // keyboard print: print the number of user pressed the button
    Keyboard.print("You pressed the button ");
    Keyboard.print(counter);
    Keyboard.println(" times.\n");   //print ENTER after output finish
  }
  // store the current button state for next comparison
  previousButtonState = buttonState;
}
```

21.13. RTC

Initialize RTC: /* This part of the initialization usually placed in setup */

Function: Rtc.begin();

Comment: Enable RTC

Function: Rtc.setDate(day，month， year);

Comment: Set date

Function: Rtc.setTime(hour， minute， sec);

Comment: Set time default is 24H system

Get time ingormation: /*Used in requirement*/

Function: Rtc.getHour();

Comment: Get the current hour

Function: Rtc.getMinute();

Comment: Get the current minute

Function: Rtc.getSecond();

Comment: Get the current second

Function: Rtc.getDay();

Comment: Get the current date

Function: Rtc.getMonth();

Comment: Get the current month

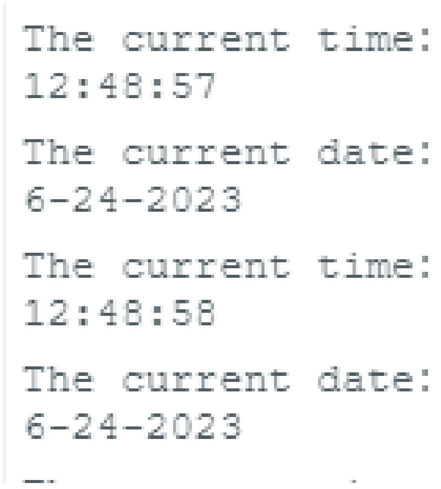
Function: Rtc.getYear();

Comment: Get the current year

Example 1: Test UART print and RTC function

```
// TH244A001 provides a class MG32x02z_RTC for basic RTC function;

//This program use UART to print time and date information;

//Test UART UART print and RTC function;

#include <Keyboard.h>

void setup() {

  /* This part of the initialization usually placed in setup */

  Rtc.begin();                  //enable RTC

  Rtc.setDate(24, 6, 2023);   //set date2023-6-24

  Rtc.setTime(12, 48, 30);    //set time 12：48：30

  Serial.begin(9600);   //enable UART for print time and date information

}

void loop() {



  //Get time, printed by UART0;

  Serial.println("The current time：");

  Serial.print(Rtc.getHour());

  Serial.print(":");

  Serial.print(Rtc.getMinute());

  Serial.print(":");

  Serial.print(Rtc.getSecond());
  Serial.println();

  //get date, printed by UART0;

  Serial.println("The current date： ");

  Serial.print(Rtc.getMonth());

  Serial.print("-");

  Serial.print(Rtc.getDay());

  Serial.print("-");

  Serial.print(Rtc.getYear());

  Serial.println();

  delay(1000);

}
```

The output after the RTC is set is as follows



```
The current time:
12:48:57
The current date:
6-24-2023
The current time:
12:48:58
The current date:
6-24-2023
```

21.14. SLEEP/STOP mode

time constant

Defines some LowPower time constants that are common to Arduino and can be used in sleep and IWDT.

IWDT timing is based on ILRCO 32KHz.

```
 SLEEP_15MS   = WDTO_15MS,       //15ms time
SLEEP_30MS   = WDTO_30MS,       //30ms time
SLEEP_60MS   = WDTO_60MS,       //60ms time
SLEEP_120MS = WDTO_120MS,      //120ms time
SLEEP_250MS = WDTO_250MS,      //250ms time
SLEEP_500MS = WDTO_500MS,      //500ms time
SLEEP_1S     = WDTO_1S,          //1s time
SLEEP_2S     = WDTO_2S,          //2s time
```

```
  SLEEP_4S     = WDTO_4S,          //4s time
  SLEEP_8S     = WDTO_8S,          //8s time
  SLEEP_FOREVER = -1               // FOREVER
```

LowPowerClass

LowPowerClass is defined to enable energy saving mode settings.

```
class LowPowerClass
{
public:
    void idle( period_t period=SLEEP_FOREVER );
    void standby( period_t period=SLEEP_FOREVER );
    void longPowerDown(uint32_t sleepTime);
    void sleep(uint32_t sleepTime);
    void powerDown(period_t period, adc_t adc=ADC_OFF, bod_t bod=BOD_OFF,
usb_t usb=USB_OFF);
    void powerStandby(period_t period, adc_t adc=ADC_OFF, usb_t usb=USB_OFF);
};
extern LowPowerClass LowPower;
```

Basic Use of Class LowPower and Methods：

```
LowPower.idle(SLEEP_FOREVER)          // Keeps STOP until a wakeup event occurs
```

LowPower.idle(SLEEP_500MS);   // Hold STOP for 500ms and it will wake up.

LowPower.standby(SLEEP_FOREVER)   // Keep SLEEP until a wakeup event occurs

LowPower.standby(SLEEP_500MS); // Hold SLEEP for 500ms and it will wake up.

LowPower. longPowerDown(uint32_t sleepTime) // Set STOP any time in ms

LowPower. **Sleep(uint32_t sleepTime)   // Set STOP any time in ms**

LowPower. **powerDown(period_t period,**

adc_t adc=ADC_OFF,

bod_t bod=BOD_OFF,

usb_t usb=USB_OFF);

// STOP setting with selective shutdown function, period_t period is the STOP state time, automatically wake up when the time is reached.

ADC_OFF、BOD_OFF、USB_OFF Select to close the project

LowPower. **powerStandby(period_t period,**

adc_t adc=ADC_OFF,

usb_t usb=USB_OFF);

// SLEEP setting with selective shutdown function, period_t period is the time of SLEEP state, wake up automatically when the time is reached.

ADC_OFF、USB_OFF Select to close the project

21.15. IWDT

IWDT timer is base onILRCO 32KHz。

```
 SLEEP_15MS   = WDTO_15MS,       // watchdog timer 15ms timeout
 SLEEP_30MS   = WDTO_30MS,       // watchdog timer 30ms timeout
 SLEEP_60MS   = WDTO_60MS,       // watchdog timer 60ms timeout
 SLEEP_120MS = WDTO_120MS,      // watchdog timer 120ms timeout
 SLEEP_250MS = WDTO_250MS,      // watchdog timer 250ms timeout
 SLEEP_500MS = WDTO_500MS,      // watchdog timer 500ms timeout
 SLEEP_1S     = WDTO_1S,         // watchdog timer 1s timeout
 SLEEP_2S     = WDTO_2S,         // watchdog timer 2s timeout
 SLEEP_4S     = WDTO_4S,         // watchdog timer 4s timeout
 SLEEP_8S     = WDTO_8S,         // watchdog timer 8s timeout
```

Examples of the use of combining interruptions: Instead of resetting the MCU system, execute the interrupt service program ISR(WDT_vect)

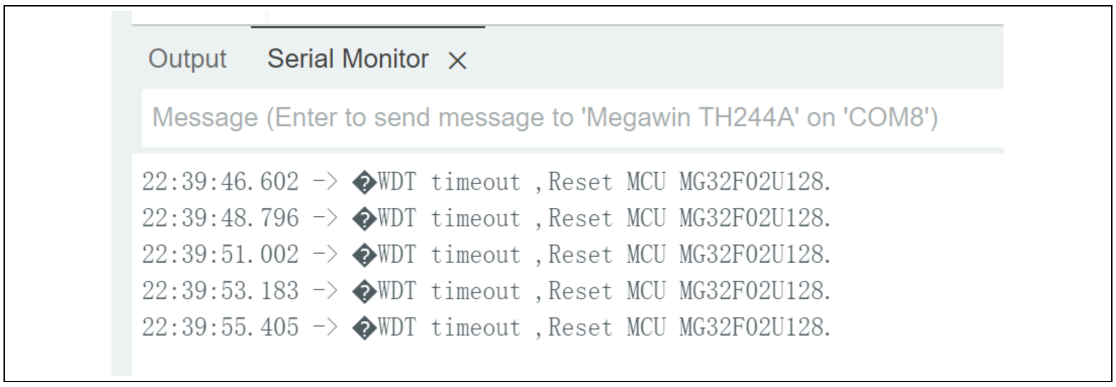
```
/* The watchdog setting achieves the effect of 4S, changing the lamp number once.
Because after entering the interrupt, the timer still continues to time, so you can not
perform too many things in the interrupt service program ISR. Generally is the
implementation of a flag bit setting or state of the flip-flop, we have to feed the
dog action.
*/
#include "IWDT.h" //must include this file
int LEDpin = 13;
volatile int state =HIGH;
```

```
ISR(WDT_vect)
{
     state = !state; //What is performed in an interrupt must be simple
   wdt_reset();        // clear WDT
}
void setup() {
    pinMode( LEDpin, OUTPUT);
    wdt_disableRST();             //disable WDT Reset MCU Function
    wdt_enable(WDTO_4S);       // enable WDT Timeout set to 4s
    wdt_reset();                    // clear
}
void loop() {
    digitalWrite( LEDpin,   state );
}
```

Combined with the use of reset example 2: open the door to the dog timeout for 2s, because the program is not designed to feed the dog, so it reaches 2s, it will reset the system. Program each time from Setup () to start executing, through the serial port to print information can be known every 2s, a RESET occurs.

```
#include "IWDT.h" //must include this file
void setup() {
  // put your setup code here, to run once:
```

```
  Serial.begin(9600);

  Serial.println("WDT timeout ,Reset MCU MG32F02U128.");

  wdt_enableRST();       // Enable watchdog interrupt reset MCU function

  wdt_enable(WDTO_2S);   // Start the watchdog and set the timeout to 2s.

  wdt_reset();           //Watchdog timer reset (feeding dog)

}



void loop() {

  // put your main code here, to run repeatedly:

}
```


Output
Serial Monitor ×
Message (Enter to send message to 'Megawin TH244A' on 'COM8')

```
22:39:46.602 -> �WDT timeout ,Reset MCU MG32F02U128.
22:39:48.796 -> �WDT timeout ,Reset MCU MG32F02U128.
22:39:51.002 -> �WDT timeout ,Reset MCU MG32F02U128.
22:39:53.183 -> �WDT timeout ,Reset MCU MG32F02U128.
22:39:55.405 -> �WDT timeout ,Reset MCU MG32F02U128.
```

Combined with the use of reset example 3: open the door dog timeout for 2s, because the program is designed to feed the dog, will not happen to reset the system. The program can consistently keep 0.5s light on and off.

```
#include "IWDT.h"        // must include this file



void setup() {
  // put your setup code here, to run once:
  pinMode(LED_BUILTIN, OUTPUT);     // LED_BUILTIN built-in LED, pin D13
  digitalWrite(LED_BUILTIN, LOW);
  wdt_enableRST();                    // Enable door open dog interrupt reset MCU
function
  wdt_enable(WDTO_2S);                // Start the door opener, set the timeout to 2s.
  wdt_reset();                        // Feed the dog (watchdog timer reset)
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(500);
  digitalWrite(LED_BUILTIN, LOW);
  delay(500);
  digitalWrite(LED_BUILTIN, HIGH);
  delay(500);
  digitalWrite(LED_BUILTIN, LOW);
  delay(500);
  wdt_reset();       // Flash time 2s, feed the dog (reset timer) before the watchdog
timer timeout 2s (the actual length of time is slightly more than 2s) to avoid
reboot  // put your main code here, to run repeatedly:
}
```

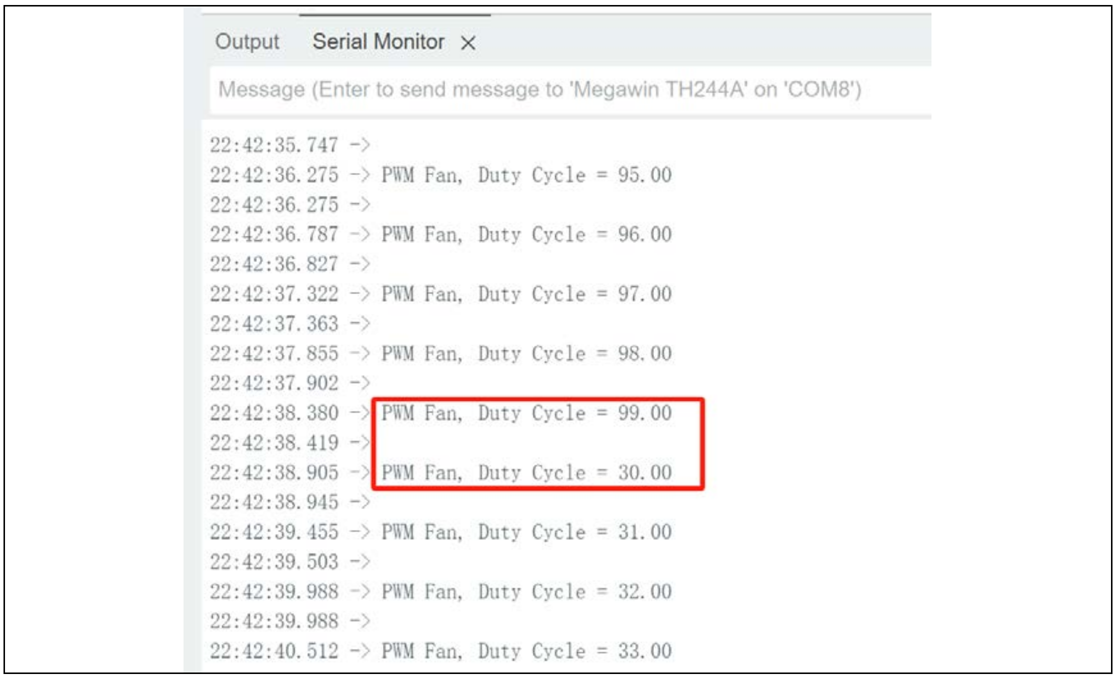
21.16. TimerOne

The output PWM function and the interrupt function are mainly utilized.

Example 1:

// Generate a PWM signal to control an external device via TimerOne; the PWM signal frequency is 25 KHz; the PWM // duty cycle increases from 30% to 100% in 1% increments; repeat the increase back to 30%. The running process will print out the duty cycle.

```
#include <TimerOne.h>

// This example creates a PWM signal with 25 kHz carrier.

//

const int fanPin = 43;


void setup(void)

{

  Timer1.initialize(40);  // 40 us = 25 kHz

  Serial.begin(9600);

}


void loop(void)

{
  // slowly increase the PWM fan speed

  //

  for (float dutyCycle = 30.0; dutyCycle < 100.0; dutyCycle++) {

    Serial.print("PWM Fan, Duty Cycle = ");

    Serial.println(dutyCycle);

     Timer1.pwm(fanPin, (dutyCycle / 100) * 1023);

     Serial.println();

    delay(500);

  }

}
```



Output
Serial Monitor
Message (Enter to send message to 'Megawin TH244A' on 'COM8')

```
22:42:35.747 ->
22:42:36.275 -> PWM Fan, Duty Cycle = 95.00
22:42:36.275 ->
22:42:36.787 -> PWM Fan, Duty Cycle = 96.00
22:42:36.827 ->
22:42:37.322 -> PWM Fan, Duty Cycle = 97.00
22:42:37.363 ->
22:42:37.855 -> PWM Fan, Duty Cycle = 98.00
22:42:37.902 ->
22:42:38.380 -> PWM Fan, Duty Cycle = 99.00
22:42:38.419 ->
22:42:38.905 -> PWM Fan, Duty Cycle = 30.00
22:42:38.945 ->
22:42:39.455 -> PWM Fan, Duty Cycle = 31.00
22:42:39.503 ->
22:42:39.988 -> PWM Fan, Duty Cycle = 32.00
22:42:39.988 ->
22:42:40.512 -> PWM Fan, Duty Cycle = 33.00
```

Example 2:

```
//Use TimerOne timer interrupt to realize 0.5s blinking light and print the number
of blinking light in real time.
// The program also uses volatile modifier variables to achieve variable sharing
between the interrupt function and the main program. The program also uses
volatile modifiers to share variables between the interrupt function and the main
program. The interrupt is flexibly controlled by disabling the interrupt // and
restarting the interrupt.

#include <TimerOne.h

//This example uses a timer interrupt to make the LED blink, and also
demonstrates how variables are shared between the interrupt function and the
main program using volatile modifiers.



const int led = LED_BUILTIN;  // the pin with a LED

void setup(void)
{
  pinMode(led, OUTPUT);
  Timer1.initialize(150000);
  //Timer1.setPeriod(150000);

  Timer1.attachInterrupt(blinkLED); // blinkLED to run every 0.15 seconds

  Serial.begin(9600);

}



// The interrupt will blink the LED, and keep

// track of how many times it has blinked.

int ledState = LOW;

volatile unsigned long blinkCount = 0; // use volatile for shared variables


void blinkLED(void)

{

  if (ledState == LOW) {

    ledState = HIGH;

    blinkCount = blinkCount + 1;  // increase when LED turns on

  } else {

    ledState = LOW;

  }

  digitalWrite(led, ledState);

}
```

```
// The main program will print the blink count

// to the Arduino Serial Monitor

void loop(void)

{

  unsigned long blinkCopy;  // holds a copy of the blinkCount

  // to read a variable which the interrupt code writes, we

  // must temporarily disable interrupts, to be sure it will

  // not change while we are reading.  To minimize the time

  // with interrupts off, just quickly make a copy, and then

  // use the copy while allowing the interrupt to keep working.

//Save a copy of blinkCount

//To read the variables written by the interrupt code, we

//must temporarily disable the interrupt to ensure that it

// won't be changed when we read it. To minimize interrupt shutdown time.

// just make a quick copy and then use the copy while re-allowing the interrupt to

continue.

   noInterrupts(); // temporarily disable interrupts

   blinkCopy = blinkCount.

   interrupts(); //start interrupts

  Serial.print("blinkCount = ");

  Serial.println(blinkCopy);
  delay(100);
```

21.17. MsTimer2

MsTimer2::set(unsigned long ms, void (*f)());//Set the interval of the timing interrupt and the interrupt service program called.

//ms indicates the length of the timing interval in ms;.

//void(*f)() indicates the name of the called interrupt service program function.

MsTimer2::start();//start timer interrupt

MsTimer2::stop(); //stop the timer interrupt, you can restart the interrupt at any time with the start method.

void(*f)(){} //Timer interrupt handler function.

Example:

```
// Blink three times (0.5s on, 0.5s off), pause for 3s (pause interrupted for 3s).
Repeat.
#include <MsTimer2.h>
const int led_pin = LED_BUILTIN;  //    LED_BUILTIN   = 13;
volatile int LED_times = 0;


void flash() { //define the interrupt function and record the number of times
   LED_times++;
   Serial.println(LED_times); //print the number of times the interrupt was
triggered, and pause the timer when it reaches 6 times
  static boolean output = HIGH;
  digitalWrite(led_pin, output);
  output = !output;
}


void setup() {
  pinMode(led_pin, OUTPUT);
  MsTimer2::set(500, flash); // set the timer period 500ms to trigger an interrupt,
the interrupt function is flash
  MsTimer2::start(); // start timer
}


void loop() {


  if (LED_times == 6) // interrupt triggered 6 times, pause timer interrupt for 3s,
start timer interrupt again
  {
    MsTimer2::stop(); // pause the timer
    delay(3000);
    LED_times = 0.
    MsTimer2::start(); //start timer
  }
}
```

21.18. EEPROM

EEPROM.begin(Size); //Apply the size of the read/write operation, Size must be smaller than the size defined by EEPROM_SIZE. For example, the default EEPROM_SIZE=512, then here Size is 1~512.

Single-byte data read/write operation:

EEPROM.write(address,data) //write data to the specified address, write method is to byte as the unit of storage, the

EEPROM.read(address) //read the specified address of the data, read() method is to read byte units

EEPROM.commit(); //so that the data from the temporary storage area SRAM saved to flash, to realize the power-down protection

When you need to write or read multiple bytes, such as floating point data or shaping data or other data structures (which require multiple bytes to store data), you can use the following method to read and write:

EEPROM.put(address, data) //write data to the specified address, data for the data to be stored

EEPROM.get(address, data) //read the data at the specified address, data is the data to be stored.

Byte count reference for common data in Arduino ARM 32-bit development board:

Data types	byte count	definition	Comment
void			Used only as a function declaration to indicate that there is no return value
boolean	1	true,false	

bool	1	true,false	
char	1	-128~127	A char in Arduino is signed and is equivalent to a signed char. The char is often used to store ASCII characters. If you want to store data, it is recommended to use the byte type.
signed char	1	-128~127	
unsigned char	1	0~255	unsigned char , byte and uint8_t type equivalents
byte	1	0~255	
uint8_t	1	0~255	
short	2	-32768~32767	
uint16_t	2	0~65535	
int	4	-2147483648~2147483647	
unsigned int	4	0 ~ 4,294,967,295	
word	4	0 ~ 4,294,967,295	
long	4	-2147483648~2147483647	
unsigned long	4	0 ~ 4,294,967,295	unsigned long equivalent to uint32_t type
uint32_t	4	0 ~ 4,294,967,295	

float	4	- 3.4028235E+38~3.4028235E+38	Only 6~7 decimal precision
double	4	- 3.4028235E+38~3.4028235E+38	Only 6~7 decimal precision

Example program:

Storing 4 different types of data, which occupy different storage spaces, involves single-byte reads and writes and multi-byte reads and writes.

After storing, it is printed out.

☐ char data ------------1 bytes

☐ int data -------------4 bytes

☐ float type data ---------- float type 4 bytes

☐ string (test character array) - 25 bytes. It is usually necessary to carry an end flag byte 0. So the actual fetch length is 26.

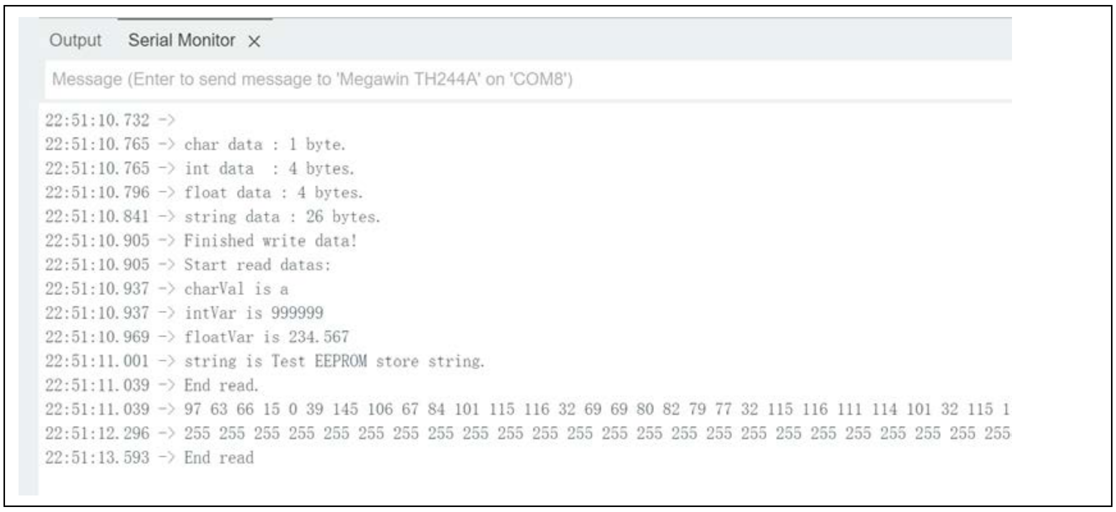
/*

Single byte read and write : EEPROM.read(address); EEPROM.write(address, byte)

Multi bytes read and write :

EEPROM.get(address,data); EEPROM.put(address,data)

```
EEPROM.commit();   Submit to save data from the temporary storage area to flash,
achieving power-off protection
Here, store char data(1 byte)& int data   (4 bytes)& float data(4 bytes) &
string(char array)   (25 bytes) data to EEPROM
*/

//Load Library Files
#include <EEPROM.h>

//Define four different types of variabless
char charVal = 'a';                          // 1   byte of char data to be stored in
EEPROM
int intVar = 999999;                          // 4   bytes of int data to be stored in
EEPROM
float floatVar = 234.567;                     // 4   bytes of floating-point data to be
stored in EEPROM
char string[] = "Test EEPROM store string.";   // 25 bytes of string to be stored in
EEPROM,include   a stop byte 0.

//Define three addresses to store four variables
int charValAddr = 0;
int intVarAddr = 1;

int fVarAddr = 5;

int stringAddr = 9;


void setup() {

  EEPROM.begin(512);

  Serial.begin(9600);


  //sizeof() is used to calculate the type length of the sizeof operator, in bytes

  Serial.println(" ");

  Serial.println("char data : " + String(sizeof(charVal)) + " byte.");      //1 byte

  Serial.println("int data  : " + String(sizeof(intVar)) + " bytes.");      //4 bytes

  Serial.println("float data : " + String(sizeof(floatVar)) + " bytes.");  //4ytes

  Serial.println("string data : " + String(sizeof(string)) + " bytes.");   //26 bytes of
string to be stored in EEPROM,include  a stop byte 0.



  EEPROM.write(charValAddr, charVal);  // Store charVal in EEPROM address 0

  delay(10);

  EEPROM.put(intVarAddr, intVar);  // Store intVar in EEPROM address 1

  delay(10);
  EEPROM.put(fVarAddr, floatVar);   // Store floatVar in EEPROM address 5
  delay(10);
  EEPROM.put(stringAddr, string);   // Store string in EEPROM address 9
  delay(10);
  EEPROM.commit();   //Submit to save data from the temporary storage area to
flash,
  Serial.println("Finished write data!");


  Serial.println("Start read datas:");
  charVal = EEPROM.read(charValAddr);
  EEPROM.get(intVarAddr, intVar);
  EEPROM.get(fVarAddr, floatVar);
  EEPROM.get(stringAddr, string);
  Serial.println("charVal is " + String(charVal));
  Serial.println("intVar is " + String(intVar));
  Serial.println("floatVar is " + String(floatVar, 3));   // three decimal places
  Serial.println("string is " + String(string));
  Serial.println("End read.");


  for (int addr = 0; addr < 512; addr++) {
    int data = EEPROM.read(addr);   //read one byte
    Serial.print(data);

    Serial.print(" ");

    delay(2);

    if ((addr + 1) % 256 == 0)   //

    {

      Serial.println("");

    }

  }

  Serial.println("End read");

}



void loop() {

  //   put your main code here, to run repeatedly:

}
```


Output
Serial Monitor ×
Message (Enter to send message to 'Megawin TH244A' on 'COM8')

```
22:51:10.732 ->
22:51:10.765 -> char data : 1 byte.
22:51:10.765 -> int data  : 4 bytes.
22:51:10.796 -> float data : 4 bytes.
22:51:10.841 -> string data : 26 bytes.
22:51:10.905 -> Finished write data!
22:51:10.905 -> Start read datas:
22:51:10.937 -> charVal is a
22:51:10.937 -> intVar is 999999
22:51:10.969 -> floatVar is 234.567
22:51:11.001 -> string is Test EEPROM store string.
22:51:11.039 -> End read.
22:51:11.039 -> 97 63 66 15 0 39 145 106 67 84 101 115 116 32 69 69 80 82 79 77 32 115 116 111 114 101 32 115 1
22:51:12.296 -> 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
22:51:13.593 -> End read
```