5.3 Use of coprocessor

Note: In the image operating system provided by Yahboom, the APP remote control process is enabled by default, in order to avoid multiple occupations of internal resources, causing some functions to fail to operate normally.

Before you running the code of this course, please follow the following to close the APP remote control process.

If you want to permanently close the function of the APP control process that starts automatically after booting, execute the following command:

```
sudo systemctl disable jetbotmini_start
```

If you want to permanently open the function of the APP control process that starts automatically after booting, execute the following command:

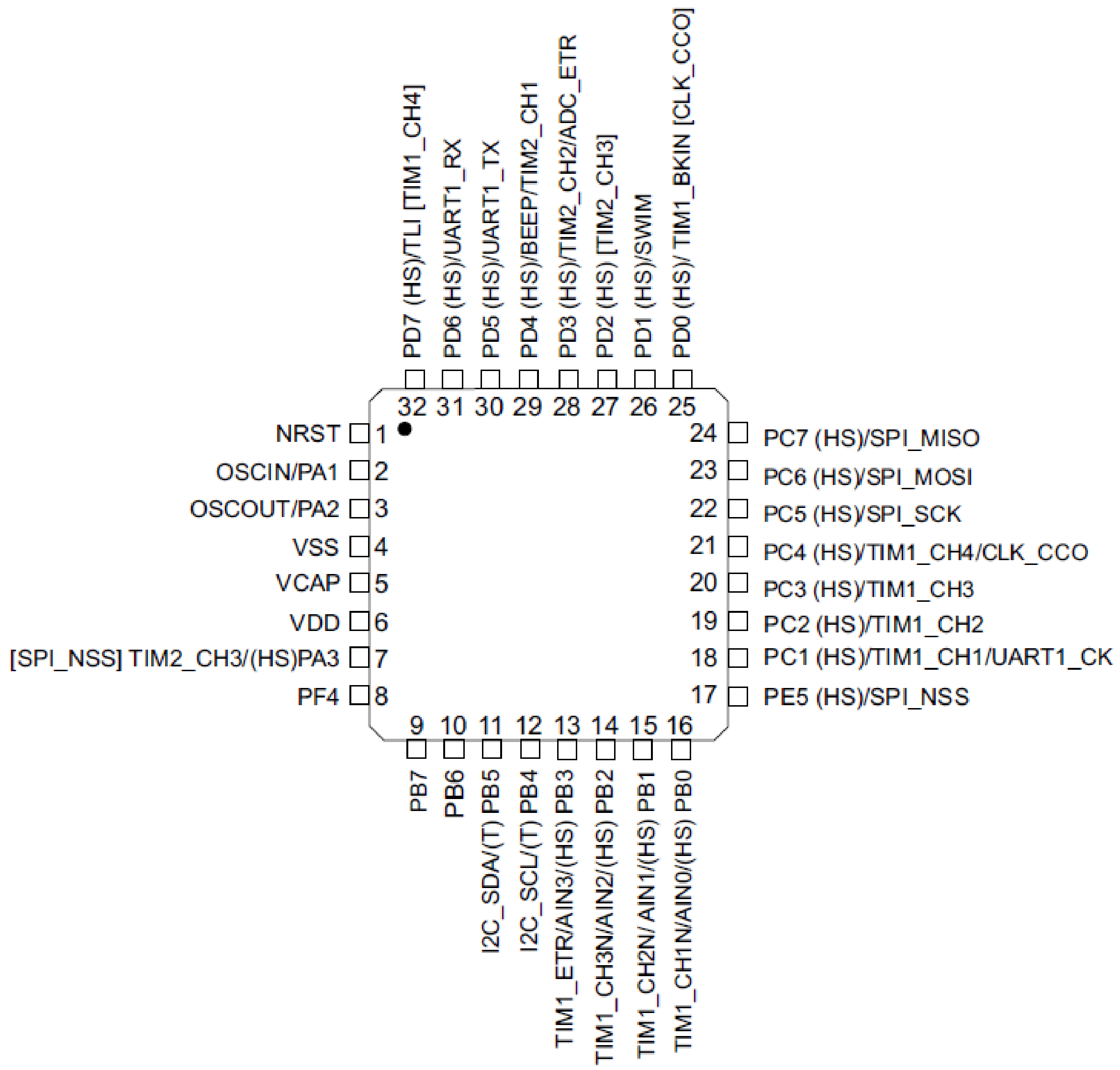
```
sudo systemctl enable jetbotmini_start
```

If you do not restart Jetbotmini to restart the APP control process function, execute the following command:

```
sudo systemctl stop jetbotmini_start
sudo systemctl start jetbotmini_start
```

5.3.1 Introduction to coprocessors

First, let me explain why we can’t use Jetson nano to generate PWM like MCU microcontrollers. From the Jetson.GPIO library file, there is no pin that supports PWM generation, which means Jetson nano does not have hardware to generate PWM. Therefore, we have to use other methods to generate PWM to complete the drive control. STM8 just solves this problem and saves its limited GPIO resources.

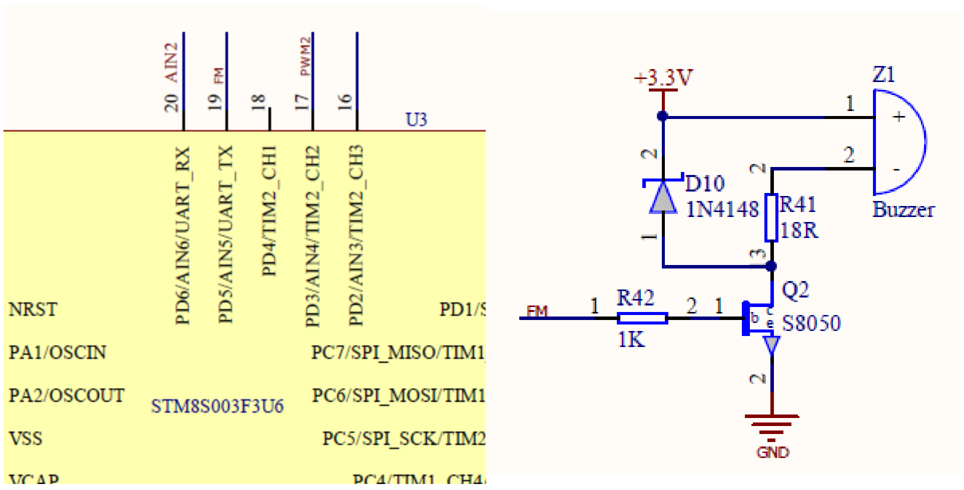


We are using the STM8 in the QFN20 package shown in the figure above, and its main parameters are as follows:

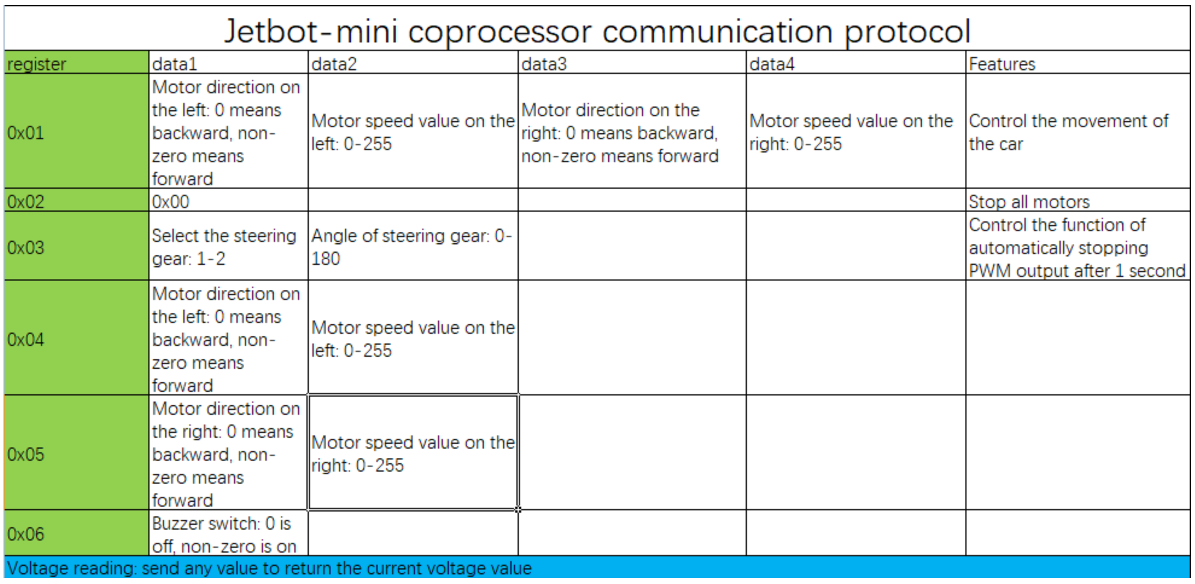
1. I2C interface, support multiple PWM output
2. Built-in 16MHz crystal oscillator, external crystal oscillator may not be connected, or external crystal oscillator can be connected
3. Support 2.95V-5.5V voltage, maximum withstand voltage 5.5V
4. With power-on reset, software reset and other functions

So how do we drive it? Next we will drive the buzzer through the coprocessor

5.3.2 Turn on the buzzer



The active buzzer circuit driven by the triode in the figure above, and the triode control pin is connected to the coprocessor, it can be seen that turning on the buzzer only needs to give the corresponding instruction to the coprocessor through the IIC. Below we see the communication protocol:



Jetbot-mini coprocessor communication protocol

register	data1	data2	data3	data4	Features
0x01	Motor direction on the left: 0 means backward, non-zero means forward	Motor speed value on the left: 0-255	Motor direction on the right: 0 means backward, non-zero means forward	Motor speed value on the right: 0-255	Control the movement of the car
0x02	0x00				Stop all motors
0x03	Select the steering gear: 1-2	Angle of steering gear: 0-180			Control the function of automatically stopping PWM output after 1 second
0x04	Motor direction on the left: 0 means backward, non-zero means forward	Motor speed value on the left: 0-255			
0x05	Motor direction on the right: 0 means backward, non-zero means forward	Motor speed value on the right: 0-255			
0x06	Buzzer switch: 0 is off, non-zero is on				
Voltage reading: send any value to return the current voltage value					

From the communication protocol, we can see that we send 1 to the register 0x06 of the coprocessor (address 0x1B) through the IIC to turn on the buzzer, and send 0 to turn off the buzzer. Below we see the source code:

```
import smbus
import time
bus = smbus.SMBus(1)
Buzzer_ADD = 0x1B
print("Starting demo now!")
curr_value = 1
while True:
    time.sleep(1)
    # Toggle the output every second
    print("Outputting {}".format(curr_value))
    bus.write_byte_data(Buzzer_ADD,0x06,curr_value)
    curr_value ^= 1
```

The phenomenon after the execution of the above code is that the buzzer is turned on or off every second. In the next code of the code:

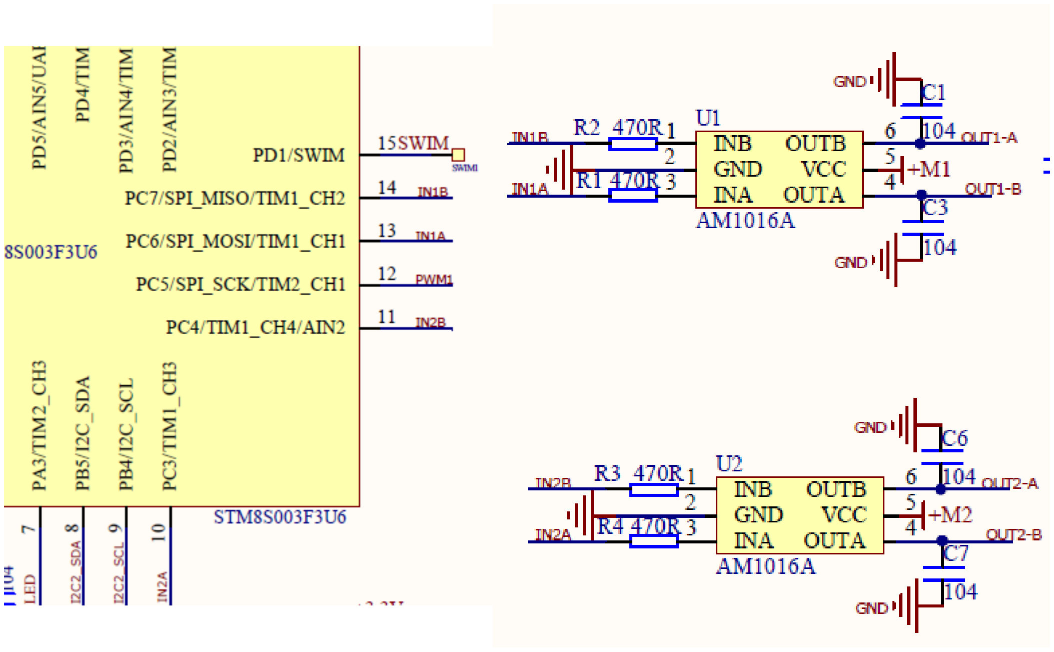
```
import smbus
import time
bus = smbus.SMBus(1)
Buzzer_ADD = 0x1B
bus.write_byte_data(Buzzer_ADD,0x06,1)
time.sleep(0.1)
bus.write_byte_data(Buzzer_ADD,0x06,0)
time.sleep(0.2)
bus.write_byte_data(Buzzer_ADD,0x06,1)
time.sleep(0.1)
bus.write_byte_data(Buzzer_ADD,0x06,0)
time.sleep(0.2)
bus.write_byte_data(Buzzer_ADD,0x06,1)
time.sleep(0.1)
bus.write_byte_data(Buzzer_ADD,0x06,0)
time.sleep(0.2)
```

The corresponding complete source code is located:

/home/jetson/Notebooks/English/3.Use_of_GPIO/3.Turn_on_the_buzzer/Turn_on_the_buzzer.ipynb

After execution, the phenomenon is three rapid beeps.

5.3.3 Two-way motors that drive and control the robot's travel



Regarding the low-level circuit that controls the two motors on the left and right of Jetbotmini, as shown above, the motor control part uses the Jetbotmini library. If you are interested, you can go to the bottom file to find out. Let's use the code to experience the operation.

First, create an instance of the control object to control the rotation of the left and right motors:

```
from jetbotmini import Robot
import time
robot = Robot()
#Control the speed of the left motor of Jetbotmini to 0.8 (speed range is 0-1.0)
robot.left_motor.value = 0.8
#Control the speed of the right motor of Jetbotmini to 0.8 (speed range is 0-
1.0)
robot.right_motor.value = 0.8
```

Indicates that the range of the value for controlling the motor speed is 0-1.0, which means that the duty cycle of the given PWM is 0-100%, and then the method of turning off the motor is:

```
robot.left_motor.value = 0
robot.right_motor.value = 0
```

That is, set the PWM duty cycle output to the motor to 0.

The corresponding complete source code is located:

/home/jetson/Notebooks/English/4.Use_of_motor/Drive_two_motors.ipynb