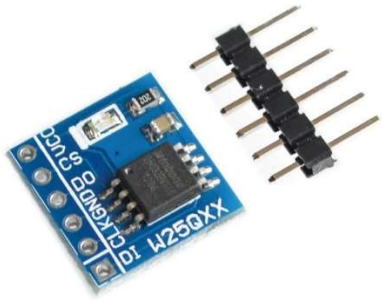
Flash memory W25Q64

Content



Product characteristics

- Using serial nor flash external expansion memory chip W25Q32
- Support SPI-interface
- Provide STM32 test code
- Main parameters:
- Capacity: 32M-bit/4M-byte
- The clock frequency is less than 104MHz
- Operating voltage: 2.7~3.6V
- Size: 14mm * 15mm

Product Features

- Using serial nor flash external expansion memory chip w25q64
- Support SPI Interface
- Provide STM32 test code

Main parameters

- Capacity: 64m-bit/8m-byte
- Clock frequency: ≤104mhz
- Operating voltage: 2.7~3.6V
- Size: 14mmx15mm

Wiring the Winbond W25Q80BV / W25QXX SPI Serial Flash

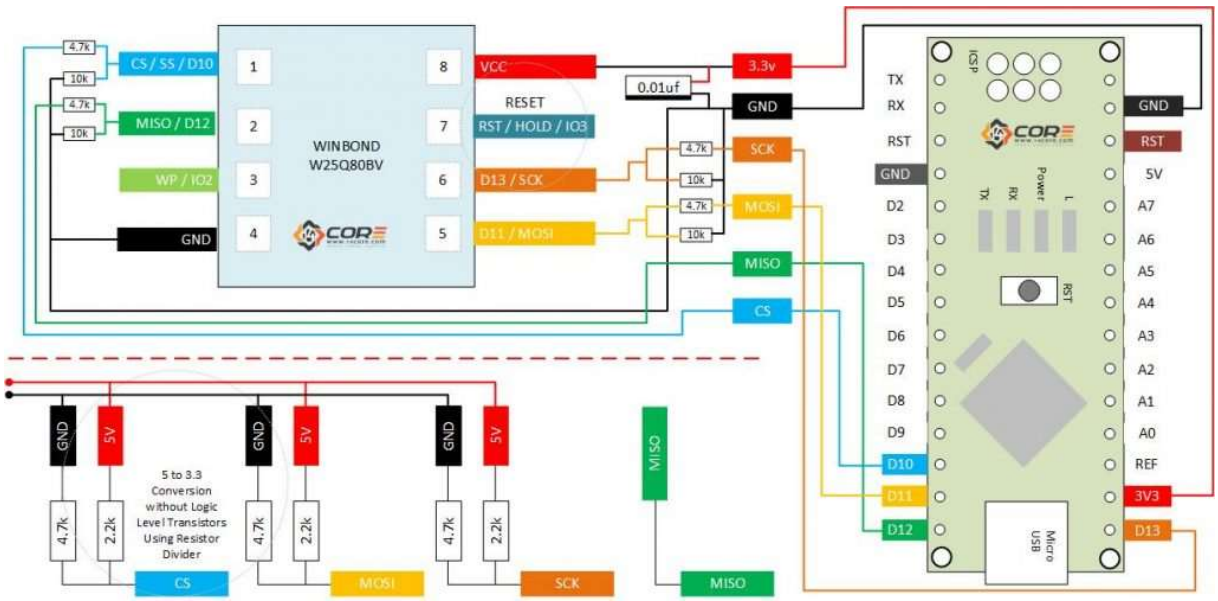
The Winbond Flash Memory provides extended megabits and bytes serial flash, used for storage on chip solution system with limited space, pins and power. The W25Q series provides flexibility and enhance performance beyond ordinary Serial Flash Devices. This device is ideal for code shadowing to RAM, executing code directly from the Dual Quad SPI storing voice, text, and data. This device operates on a single 2.7v ~ 3.6v power supply with current consumption as low as 4mA active & 1uA for power down. For this demonstration we will going to wire the W25Q80BV an 8Mbit serial flash memory which is available in modular type suited for any microcontroller like Arduino, Tenssy, and other popular microcontrollers.

Required Components

Arduino Microcontroller. Note: The Diagram below is using NANO. Please refer to the respective pin-outs.

- WindBond SPI Flash Memory Chip / Windbond SPI Flash Memory Module
- 10k Resistor
- 4.7k Resistor
- 0.01 uF Capacitor
- Jumper Wire / DuPont Wire
- Solder Less Bread Board

Wiring Guide



Source Code

```
#include <SPI.h>

#define writeEnable       0x06 // Address Write Enable
#define writeDisable      0x04 // Address Write Disable
#define chipErase         0xc7 // Address Chip Erase
#define readStatusReg1    0x05 // Address Read Status
#define readData          0x03 // Address Read Data
#define pageProgramStat   0x02 // Address Status Page Program
#define chipCommandId     0x9f // Address Status Read Id


boolean g_command_ready(false);
String g_command;

/*
print_page_bytes() is a simple helper function that formats 256 bytes
*/
void print_page_bytes(byte *page_buffer) {
  char buf[10];
  for (int i = 0; i < 16; ++i) {
    for (int j = 0; j < 16; ++j) {
      sprintf(buf, "%02x", page_buffer[i * 16 + j]);
      Serial.print(buf);
    }
    Serial.println();
  }
}

/*

This functions map to user commands. wrap the low-level calls with
print/debug statements to read
*/

/*
The chip command id  is fairly generic, just to verify function setup
*/
void chipCmdIda(void) {
  Serial.println("Set Command: chipCmdIda");
  byte b1, b2, b3;
  chipCmdId(&b1, &b2, &b3);
  char buf[128];
  sprintf(buf, "ID: %02xh\nMemory Type: %02xh\nCapacity: %02xh",
    b1, b2, b3);
  Serial.println(buf);
  Serial.println("Ready");
}

void chip_erase(void) {
  Serial.println("command: chip_erase");
```

```
51          _chip_erase();
52          Serial.println("Ready");
53        }
54
55        void read_page(unsigned int page_number) {
56          char buf[80];
57          sprintf(buf, "command: read_page(%04xh)", page_number);
58          Serial.println(buf);
59          byte page_buffer[256];
60          _read_page(page_number, page_buffer);
61          print_page_bytes(page_buffer);
62          Serial.println("Ready");
63        }
64
65        void read_all_pages(void) {
66          Serial.println("command: read_all_pages");
67          byte page_buffer[256];
68          for (int i = 0; i < 4096; ++i) {
69            _read_page(i, page_buffer);
70            print_page_bytes(page_buffer);
71          }
72          Serial.println("Ready");
73        }
74
75        void write_byte(word page, byte offset, byte databyte) {
76          char buf[80];
77          sprintf(buf, "command: write_byte(%04xh, %04xh, %02xh)", page, offset, databyte);
78          Serial.println(buf);
79          byte page_data[256];
80          _read_page(page, page_data);
81          page_data[offset] = databyte;
82          _write_page(page, page_data);
83          Serial.println("Ready");
84        }
85
86
87        void chipCmdId(byte *b1, byte *b2, byte *b3) {
88          digitalWrite(SS, HIGH);
89          digitalWrite(SS, LOW);
90          SPI.transfer(chipCommandId);
91          *b1 = SPI.transfer(0); // manufacturer id
92          *b2 = SPI.transfer(0); // memory type
93          *b3 = SPI.transfer(0); // capacity
94          digitalWrite(SS, HIGH);
95          not_busy();
96        }
97
98        /*
99        See the timing diagram in section 9.2.26 of the data sheet, "Chip Erase (C7h / 06h)". (Note:
100       */
101       void _chip_erase(void) {
```

```
102         digitalWrite(SS, HIGH);
103         digitalWrite(SS, LOW);
104         SPI.transfer(writeEnable);
105         digitalWrite(SS, HIGH);
106         digitalWrite(SS, LOW);
107         SPI.transfer(chipErase);
108         digitalWrite(SS, HIGH);
109
110         /* See notes on rev 2
111         digitalWrite(SS, LOW);
112         SPI.transfer(writeDisable);
113         digitalWrite(SS, HIGH);
114         */
115         not_busy();
116       }
117
118       /*
119       * See the timing diagram in section 9.2.10 of the
120       * data sheet located below, "Read Data (03h)".
121       */
122       void _read_page(word page_number, byte *page_buffer) {
123         digitalWrite(SS, HIGH);
124         digitalWrite(SS, LOW);
125         SPI.transfer(readData);
126
127         // Construct the 24-bit address from the 16-bit page
128         // number and 0x00, since we will read 256 bytes (one
129         // page).
130         SPI.transfer((page_number >> 8) & 0xFF);
131         SPI.transfer((page_number >> 0) & 0xFF);
132         SPI.transfer(0);
133         for (int i = 0; i < 256; ++i) {
134           page_buffer[i] = SPI.transfer(0);
135         }
136         digitalWrite(SS, HIGH);
137         not_busy();
138       }
139       /*
140       * See the timing diagram in section 9.2.21 of the
141       * data sheet, "Page Program (02h)".
142       */
143       void _write_page(word page_number, byte *page_buffer) {
144         digitalWrite(SS, HIGH);
145         digitalWrite(SS, LOW);
146         SPI.transfer(writeEnable);
147         digitalWrite(SS, HIGH);
148         digitalWrite(SS, LOW);
149         SPI.transfer(pageProgramStat);
150         SPI.transfer((page_number >>  8) & 0xFF);
151         SPI.transfer((page_number >>  0) & 0xFF);
152         SPI.transfer(0);
```

```
153          for (int i = 0; i < 256; ++i) {
154            SPI.transfer(page_buffer[i]);
155          }
156          digitalWrite(SS, HIGH);
157          /* See notes on rev 2
158          digitalWrite(SS, LOW);
159          SPI.transfer(writeDisable);
160          digitalWrite(SS, HIGH);
161          */
162          not_busy();
163        }
164
165        /*
166        * See section 9.2.8 of the datasheet
167        */
168        void not_busy(void) {
169          digitalWrite(SS, HIGH);
170          digitalWrite(SS, LOW);
171          SPI.transfer(WB_READ_STATUS_REG_1);
172          while (SPI.transfer(0) & 1) {};
173          digitalWrite(SS, HIGH);
174        }
175
176        /*
177        * string, setting a boolean used by the loop() routine
178        * as a dispatch trigger.
179        */
180        void serialEvent() {
181          char c;
182          while (Serial.available()) {
183            c = (char)Serial.read();
184            if (c == ';') {
185              g_command_ready = true;
186            }
187            else {
188              g_command += c;
189            }
190          }
191        }
192
193        void setup(void) {
194          SPI.begin();
195          SPI.setDataMode(0);
196          SPI.setBitOrder(MSBFIRST);
197          Serial.begin(9600);
198          Serial.println("");
199          Serial.println("Ready");
200        }
201
202        /*
203        */
```

```
204         void loop(void) {
205           if (g_command_ready) {
206             if (g_command == "chipCmdIda") {
207               chipCmdIda();
208             }
209             else if (g_command == "chip_erase") {
210               chip_erase();
211             }
212             else if (g_command == "read_all_pages") {
213               read_all_pages();
214             }
215             // A one-parameter command...
216             else if (g_command.startsWith("read_page")) {
217               int pos = g_command.indexOf(" ");
218               if (pos == -1) {
219                 Serial.println("Error: Command 'read_page' expects an int operand");
220               } else {
221                 word page = (word)g_command.substring(pos).toInt();
222                 read_page(page);
223               }
224             }
225             // A three-parameter command..
226             else if (g_command.startsWith("write_byte")) {
227               word pageno;
228               byte offset;
229               byte data;
230
231               String args[3];
232               for (int i = 0; i < 3; ++i) {
233                 int pos = g_command.indexOf(" ");
234                 if (pos == -1) {
235                   Serial.println("Syntax error in write_byte");
236                   goto done;
237                 }
238                 args[i] = g_command.substring(pos + 1);
239                 g_command = args[i];
240               }
241               pageno = (word)args[0].toInt();
242               offset = (byte)args[1].toInt();
243               data = (byte)args[2].toInt();
244               write_byte(pageno, offset, data);
245             }
246             else {
247               Serial.print("Invalid command sent: ");
248               Serial.println(g_command);
249             }
250         done:
251             g_command = "";
252             g_command_ready = false;
253           }
254         }
```

```
255
```

Downloads

Download the W25Q80BV Datasheet | PDF
Download the W25Q80BV Code library using ATTINY85 | Zip

Source

https://www.14core.com/wiring-the-winbond-w25qxx-spi-serial-flash-memory-with-microcontroller/