QUICK REFERENCE · TK104

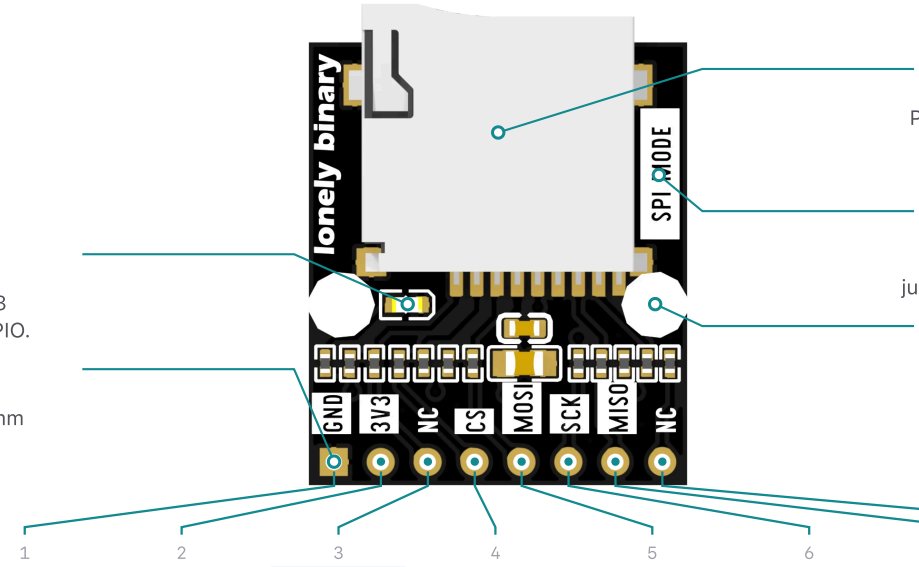
microSD Card Reader

SPI, SD 1-bit and SD 4-bit from one 8-hole header.

3.3 V only The 3V3 hole goes straight to the card, which is rated 2.7 to 3.6 V. There is no regulator and no level shifter. **Not for a 5 V Arduino Uno** on its own: a 5 V board needs a level converter in every signal wire and a separate 3.3 V supply.

One header, two sets of names

The front is printed with SPI names, the back with SD bus names for the same holes. Count holes from the square GND pad.



Power LED
Red, lit whenever 3V3 is powered. On no GPIO.

No header fitted.
Solder a 1 × 8, 2.54 mm pin header first.

Push-push microSD socket
Push until it clicks; push again to release. A card that is only half in reads as no card.

SPI MODE is a label
for the front names, not a jumper. Wiring picks the mode.

2 mounting holes
Ø 2.5 mm, 17.0 mm apart

	1	2	3	4	5	6	7	8
Front	GND	3V3	NC	CS	MOSI	SCK	MISO	NC
Back	GND	3V3	DATA2	DATA3	CMD	CLK	DATA0	DATA1
SPI	GND	3V3	·	CS	MOSI	SCK	MISO	·
SD 1-bit	GND	3V3	·	·	CMD	CLK	DATA0	·
SD 4-bit	GND	3V3	DATA2	DATA3	CMD	CLK	DATA0	DATA1

NC ≠ not connected Holes 3 and 8 are **DATA2** and **DATA1**, wired through 22 Ω and pulled up like every data line. NC only means SPI does not use them. 4-bit mode needs both.

Three ways to talk to the card

SPI

4 signal wires + 3V3 and GND

`SD.h`

Any 3.3 V microcontroller with SPI

Read	1.56
Write	1.39

SD 1-bit

3 signal wires + 3V3 and GND

`SD_MMC.h`

ESP32 or ESP32-S3

Read	2.67
Write	2.16

SD 4-bit

6 signal wires + 3V3 and GND

`SD_MMC.h`

ESP32-S3, any six free pins

Read	3.80
Write	2.63

More data lines speed up reads more than writes, and never by the ×1 / ×2 / ×4 on the back of the board: that counts wires. The card's own flash and the file system do not get faster. SPI is plenty for a data logger; use 4-bit to read large files.

Read and write in MB/s, sustained, measured on this board: ESP32-S3, jumper wires on a breadboard, one card, five rounds of a large file with every round checksum-verified. Bus clock settled at 24 MHz (SPI), 40 MHz (1-bit), 40 MHz (4-bit). Only the ESP32-S3 was tested.

Your first file, over SPI

Arduino IDE · Board: ESP32S3 Dev Module · USB CDC On Boot: Enabled

```
#include <SPI.h>
#include <SD.h>

#define SD_CS    10
#define SD_MOSI  11
#define SD_SCK   12
#define SD_MISO  13

SPIClass sdSPI(FSPI);

void setup() {
  Serial.begin(115200);
  delay(500);

  sdSPI.begin(SD_SCK, SD_MISO, SD_MOSI, SD_CS);
  if (!SD.begin(SD_CS, sdSPI, 24000000)) {    // Hz
    Serial.println("no card: check 3V3, GND, then the signals");
    return;
  }
  Serial.printf("card mounted, %llu MB\n", SD.cardSize() >> 20);

  File f = SD.open("/hello.txt", FILE_WRITE);
  f.println("written by an ESP32");
  f.close();

  f = SD.open("/hello.txt");
  Serial.print("read back: ");
  while (f.available()) Serial.write(f.read());
  f.close();
}

void loop() {}
```

```
card mounted, 30436 MB
read back: written by an ESP32
```

What Serial Monitor at 115200 should print. Line one proves the bus, line two the file system.

Wiring

	FRONT	BACK	ESP32-S3	SPI	4-BIT
1	**GND**	GND	**GND**	✓	✓
2	**3V3**	3V3	**3V3**	✓	✓
3	**NC**	DATA2	**GPIO 9**	–	✓
4	**CS**	DATA3	**GPIO 10**	✓	✓
5	**MOSI**	CMD	**GPIO 11**	✓	✓
6	**SCK**	CLK	**GPIO 12**	✓	✓
7	**MISO**	DATA0	**GPIO 13**	✓	✓
8	**NC**	DATA1	**GPIO 14**	–	✓

SPI works on any four free GPIOs; these are the pins the bench used. On an ESP32-S3 avoid GPIO 19 and 20 (USB) and the strapping pins 0, 3, 45 and 46. No library to install: SD and SD_MMC ship with the ESP32 core.

The card

FAT32 or exFAT. A card that has been in a Raspberry Pi is neither, and needs formatting on a computer first. An ordinary 4–32 GB class 10 card is enough; a faster one gains nothing here. Push it in until it clicks.

Switching to 4-bit: two more wires

```
#include <SD_MMC.h>
// in setup(): CLK, CMD, D0, D1, D2, D3
SD_MMC.setPins(12, 11, 13, 14, 9, 10);
// mount point, 1-bit = false, format = false, clock in kHz
if (!SD_MMC.begin("/sdcard", false, false, 40000)) return;
// then SD_MMC.open(...) exactly as SD.open(...)
```

Add holes 3 and 8 (both printed NC) and swap `SD.h` for `SD_MMC.h`. There is no chip select in SD mode. `SD_MMC.begin()` takes **kHz**, not Hz. On jumper wires 4-bit settled at 40 MHz on this board: if 40000 fails, try 20000. On a classic ESP32 the pins are fixed, so remove `setPins()` and read “stopped booting” below.

When it does not work

It prints “no card” every time

Supply first: the red LED lit, 3.3 V between GND and 3V3. Then the card: FAT32 or exFAT. Then count the holes again from GND.

SD.h worked, SD_MMC.begin() fails

4-bit needs DATA1, DATA2 and DATA3 all wired to the microcontroller; the pull-ups are no substitute. DATA2 is hole 3, DATA1 is hole 8.

A classic ESP32 stopped booting

Its SD pins are fixed and D2 lands on GPIO 12, a strapping pin. Use 1-bit: CLK 14, CMD 15, D0 2, `begin("/sdcard", true)`.

It mounts at 4 MHz but not at 24

The wiring is marginal, not wrong. Shorten the jumper wires, the ground wire most of all, and press the pins fully into the breadboard.

It mounted once, then never again

A card that has answered in SPI stays in SPI until its power is removed. The reset button leaves the card powered: unplug the board instead.

A faster card did not help

The fastest thing measured on this board was 5.04 MB/s, and class 10 is rated for 10. A fake card is the one to avoid: it loses data where its real capacity ends.

Specifications

Supply	3.3 V only. No regulator, no level shifter	**Header**	8 holes, 2.54 mm pitch. Header not fitted
Bus parts	22 Ω in series × 6; 10 kΩ pull-ups × 5, none on CLK	**Decoupling**	10 µF + 100 nF beside the socket
Socket	TF PUSH push-push, rated for 5000 insertions	**Size**	20.5 × 26.1 mm with the socket, 3.65 mm tall
Mounting	2 × Ø 2.5 mm plated holes, 17.0 mm apart	**Cards**	FAT32 or exFAT. Ordinary 4–32 GB, class 10