Reading and writing SRAM with an Arduino Uno

Electronics II / Digital Electronics presentation
LMU Physics - SS2020

David Katheder

06.07.2020

Contents

Reasons to use this SRAM chip

- memory on microcontrollers is scarce
- e.g. Arduino Uno 32 kB FLASH, 2 kB SRAM, 1 kB EEPROM
- store larger amounts data at runtime (volatile)
- easy to setup with Arduino (SPI bus, SPI libary)
- detailed datasheet and information for recommended usage (see References)

Ratings and features of 23LC1024 Chip Layout

- 128 kB storage in 128K x 8-bit organisation
  - 17 bit address space
  - 4096 x 32-byte pages
- available as DIP, 2.80€ e.g. at Conrad
- controllable vie SPI
- 2.5-5.5 V power supply
- 20 MHz maximum clock frequency
- 8-bit mode register
- three R/W operating modes: byte, sequential, page

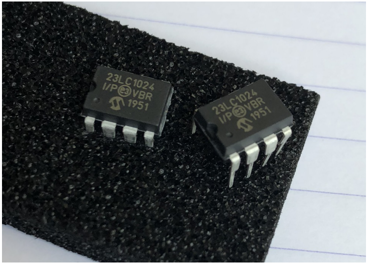

Figure 1: Microchip Technologies 23LC1024 SRAM chip

Pin layout, circuit and used parts

Pin	Name	Function
1	$\overline{CS}$	Chip Select Input
2	MISO	Serial Output Pin
3	NU	Not used
4	Vss	Ground
5	MOSI	Serial Input Pin
6	SCK	Serial Clock
7	$\overline{HOLD}$	HOLD Pin
8	Vcc	Power Supply

Table 1: Overview of pins and with the corresponding function.

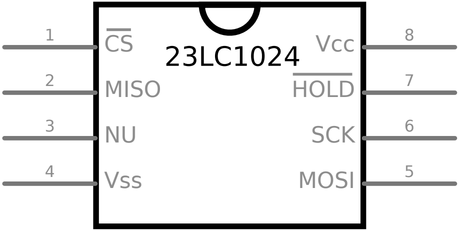

Figure 2: Pin layout in SPI mode.

Pin layout, circuit and used parts

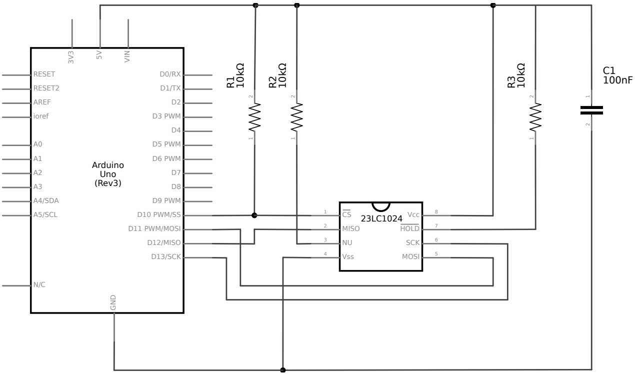

Figure 3: Circuit for driving 23LC1024 with Arduino through SPI, with HOLD functionality disabled, circuit as suggested in [1].

Parts:

- Arduino Uno
- Breadboard
- 23LC1024 chip
- 100 nF capacitor
- three 10 kΩ resistors
- wires

Circuit on a breadboard

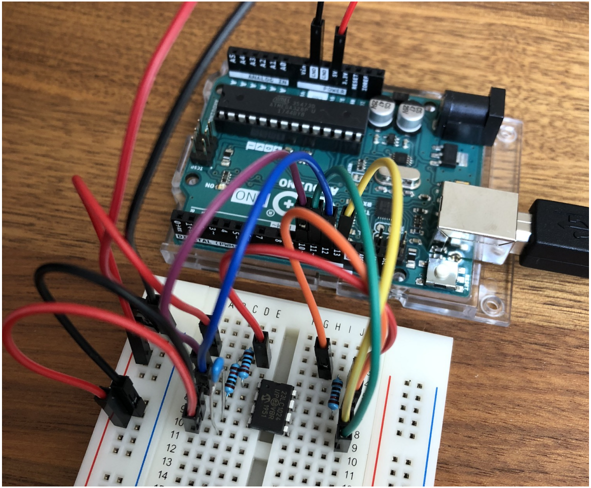

Figure 4: Circuit realization on a breadboard.

Communication with the chip

How does the communication work?

- Every interaction starts by setting chip select low, and transferring a instruction byte.
- Depending on the instruction, the chip expects to recieve data or returns data with the following clock cycles.
- Four relevant instructions in SPI mode: RDMR, WRMR, READ, WRITE

Instruction Name	Instruction Format	Hex Code	Description
READ	0000 0011	0x03	Read data from memory array beginning at selected address
WRITE	0000 0010	0x02	Write data to memory array beginning at selected address
EDIO	0011 1011	0x3B	Enter Dual I/O access (enter SDI bus mode)
EQIO	0011 1000	0x38	Enter Quad I/O access (enter SQI bus mode)
RSTIO	1111 1111	0xFF	Reset Dual and Quad I/O access (revert to SPI bus mode)
RDMR	0000 0101	0x05	Read Mode Register
WRMR	0000 0001	0x01	Write Mode Register

Figure 5: Instuction set for 23LC1024 chip [2].

Writing to the mode register

setting the operation mode of the device

Mode register entry determines the way we read and write data from the device.
Before a R/W operation we have to set the register to the desired mode:

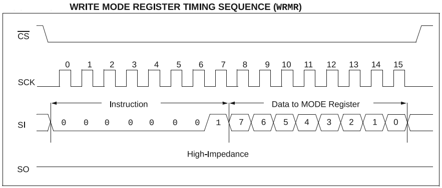

Figure 6: Timing sequence for writing to the mode register [2]

1. pull $\overline{CS}$ LOW
2. clock in WRMR instruction byte
3. clock in mode byte
4. set $\overline{CS}$ to HIGH

mode	mode byte
Byte	0000 0000
Sequential	0100 0000
Page	1000 0000

Operating modes

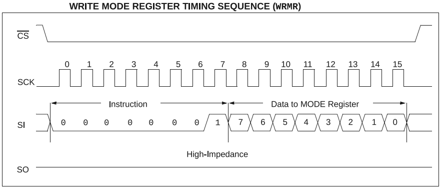

Figure 7: Timing sequence for writing to the mode register [2]

mode	mode byte
Byte	0000 0000
Sequential	0100 0000
Page	1000 0000

- Byte: R/W a single byte per operation.
- Sequential:
  - R/W an arbitrary number of bytes
  - address pointer is automatically incremented
  - rolls over to 0 if highest address is reached
- Page: like sequential mode, but address pointer rolls over to page start if the end of 32-byte page is reached.

Writing instructions

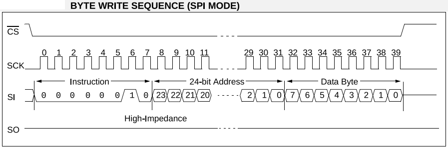

Figure 8: Timing sequence for writing in byte mode [2].

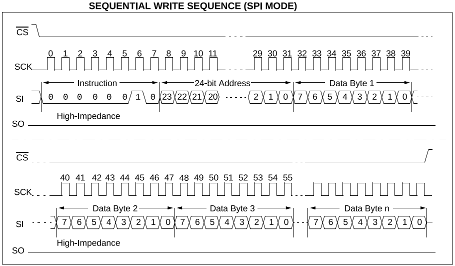

Figure 9: Timing diagram for writing data in sequential mode [2].

- set $\overline{CS}$ LOW
- clock in WRITE instruction byte
- transfer 24 bit-address, bits 23 to 17 are ignored (17 bit address space)
- transfer data byte
- in sequential/page mode subsequent bytes can be written by transfering data with following clock cycles (Fig. 9).
- set $\overline{CS}$ HIGH

Reading instructions

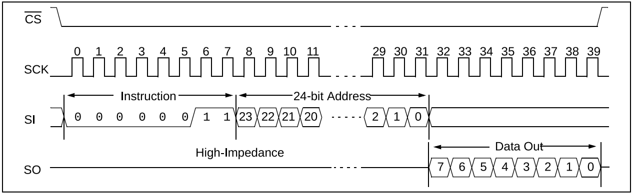

Figure 10: Reading a word in Byte Mode [2].

- set $\overline{CS}$ LOW
- clock in READ instruction byte
- transfer 24 bit-address, bits 23 to 17 are ignored (17 bit address space)
- data is shifted out with the next eight clock cycles MSB first
- in sequential/page mode subsequent bytes can be read, by providing additional clock cycles.
- set $\overline{CS}$ HIGH

Setting up the Arduino SPI libary and the SPI bus

```
#include <SPI.h> // Import the SPI libary

SPISettings settings = SPISettings(20000000, MSBFIRST, SPI_MODE0)
/**
In between here alot of other constants are defined.
**/
void setup() {
  Serial.begin(9600); // this is for displaying response of arduino on the serial monitor

  while (!Serial) {
    ; // wait for serial port to connect
  }

  SPI.begin(); // Initializes the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.

  Serial.println("Setup complete");
}
```

- import SPI libary
- call SPI.begin() in setup() to initialize the Arduino SPI bus
- define SPISettings object for your device (maximum clock rate, bit order, clock polarity and phase)

Code example: Writing to the mode register

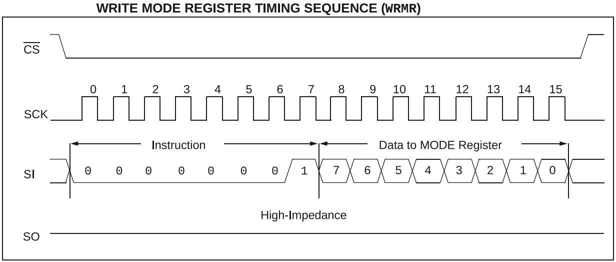

Figure 11: Timing sequence for writing to the mode register [2].

```
const int CS = 10;        // pin number for chip select
const byte WRMR = 0x01;    // write mode register instruction byte as HEX

void WriteModeRegister(byte mode) {
  SPI.beginTransaction(settings);    // assert SPISettings, gain control of SPI bus
  digitalWrite(CS, LOW);             // assert chip select

  // recievedVal = SPI.transfer(Val);
  SPI.transfer(WRMR);               // transfer WRMR instruction
  SPI.transfer(mode);                // transfer mode byte

  digitalWrite(CS, HIGH);            // deassert chip select
  SPI.endTransaction();              // release SPI bus
}

```

Code example: Writing a single byte

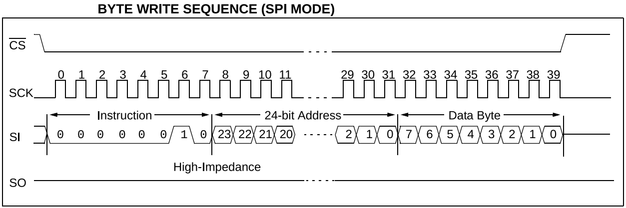

Figure 12: Timing sequence for writing a single byte [2]

```
void WriteByte(uint32_t address, byte data) {
  WriteModeRegister(ByteMode);          // write register to ByteMode
  SPI.beginTransaction(settings);       // assert SPISettings, gain control of SPI bus
  digitalWrite(CS, LOW);                    // assert chip select

  SPI.transfer(WRITE);                      // transfer WRITE instruction byte
  // example of shifting operation
  // uint16_t test = 0x0100; // 0000 0001 0000 0000 // 256 DEC
  // (byte)(test >> 8); // 0000 0001
  // (byte) test; // 0000 0000

  SPI.transfer((byte) (address >> 16));     // transfer highest byte
  SPI.transfer((byte) (address >> 8));      // transfer middle byte
  SPI.transfer((byte) address);             // transfer lowest byte
  SPI.transfer(data);                       // write single data byte
  digitalWrite(CS, HIGH);                   // deassert chip select
  SPI.endTransaction();                     // release SPI bus
}
```

Code example: Reading a single byte

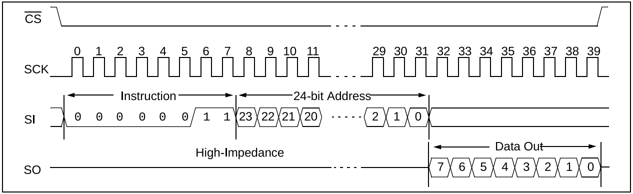

Figure 13: Timing sequence for writing a single byte [2]

```
byte ReadByte(uint32_t address) {
  WriteModeRegister(ByteMode);              // write register to ByteMode
  SPI.beginTransaction(settings);           // assert SPISettings, gain control of SPI bus
  digitalWrite(CS, LOW);                    // assert chip select

  SPI.transfer(READ);                       // transfer READ instruction byte

  SPI.transfer((byte) (address >> 16));     // transfer highest byte first
  SPI.transfer((byte) (address >> 8));      // transfer middle byte
  SPI.transfer((byte) address);             // transfer lowest byte

  byte data = SPI.transfer(0x00);           // read data

  digitalWrite(CS, HIGH);                   // deassert chip select
  SPI.endTransaction();                     // release SPI bus

  return data;                              // return data
}
```

Demo time!

1. Read and write basic data types to SRAM
2. Weather station - RTC module and DTH22 sensor

Sample code is available is this repository:
`https://github.com/davidkatheder/supreme-potato`

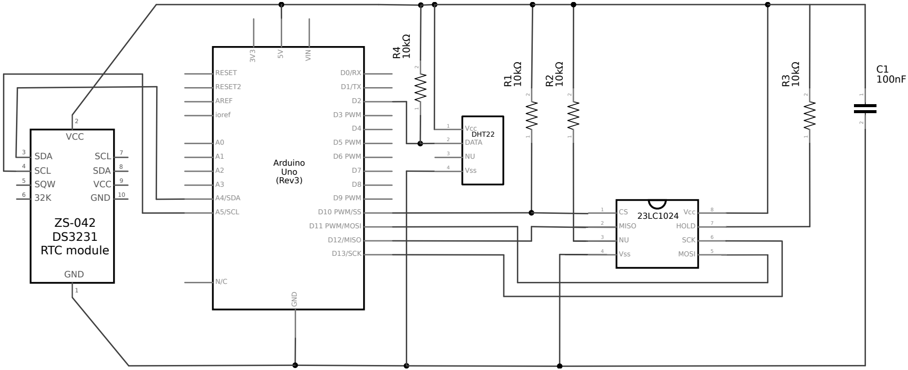

Figure 14: Circuit for the "weather station"

Additional parts

- 1x 10kΩ resistor
- DHT22 sensor
- DS3231 RTC module

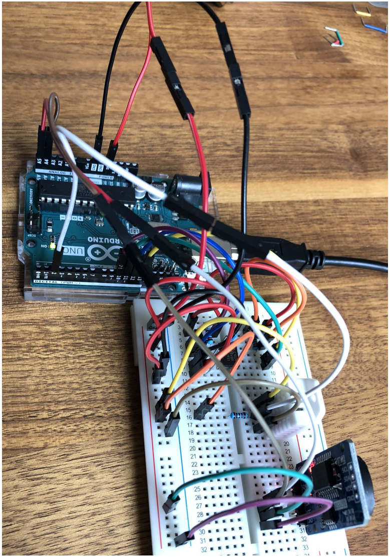

Figure 15: Picture of the "weather station" circuit. Looks messy but works :D.

References

[1] Microchip Technology, Recommended usage of microchip 23XX512/23XX1024 serial SRAM devices, Oct 2012, `http://ww1.microchip.com/downloads/en/Appnotes/01484A.pdf`, Acessed on 26.06.2020.

[2] Microchip Technology, 1Mbit SPI Serial SRAM with SDI and SQI Interface (23LC1024 Datasheet), Mar 2014, `http://ww1.microchip.com/downloads/en/DeviceDoc/20005142C.pdf`, Acessed on 26.06.2020.