Laboratory work No. 8

**Goal of research:**

Study of working with external EEPROM memory using the I2C bus.

**Software:**

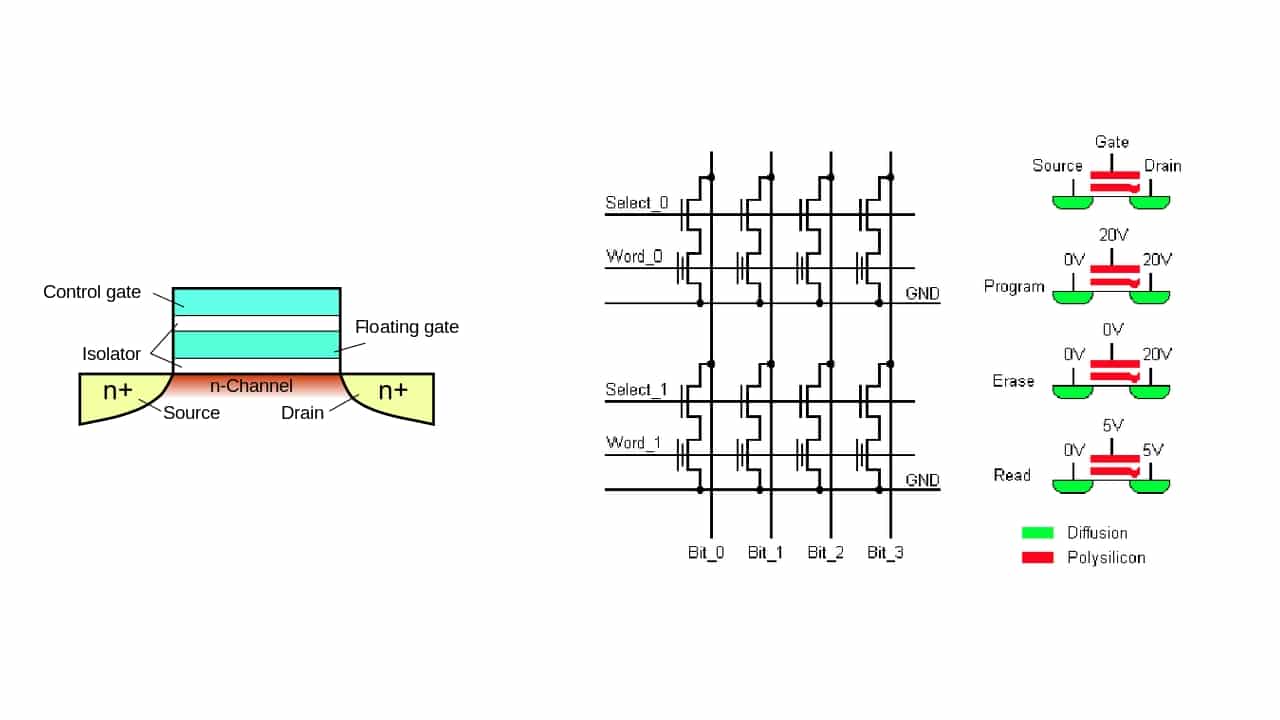
STM32CubeIDE, Matlab.

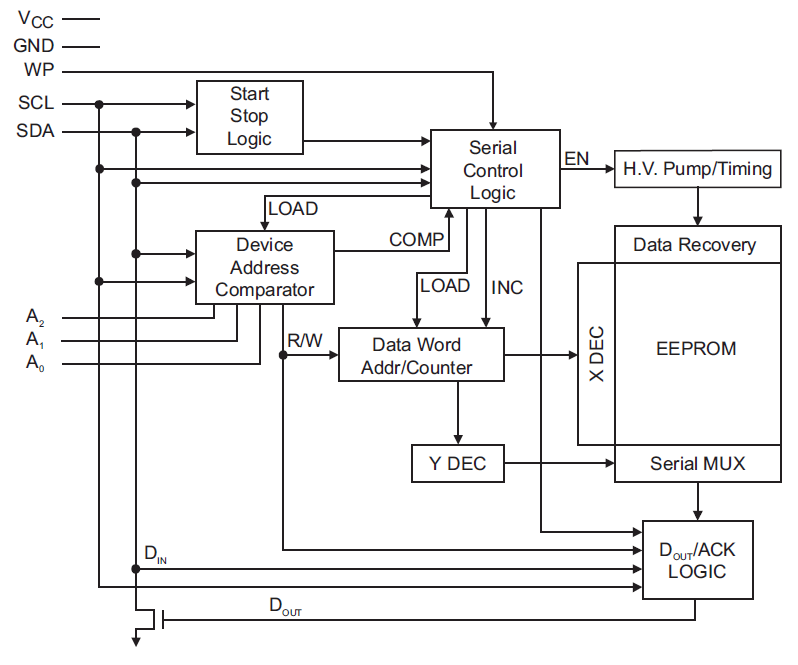
**General information:**

EEPROM (Electrically Erasable Programmable Read-Only Memory) - Electrically Erasable Programmable Read-Only ROM (EEPROM), a type of non-volatile memory (such as PROM and EPROM). This type of memory can be erased and filled with data up to one million times.

An EEPROM memory cell is a MOS transistor in which the gate is made of polycrystalline silicon. This gate is then oxidized during the chip fabrication process and as a result it will be surrounded by silicon oxide, which is a dielectric with excellent insulating properties. In a floating-gate transistor, when the ROM is completely erased, there is no charge in the "floating" gate, and therefore this transistor does not conduct current. During programming, a high voltage is applied to the second gate above the floating gate and charges are induced into it due to the tunnel effect. After the programming voltage is removed, the induced charge remains on the floating gate and hence the transistor remains in a conducting state. The charge on its floating gate can be stored for tens of years.

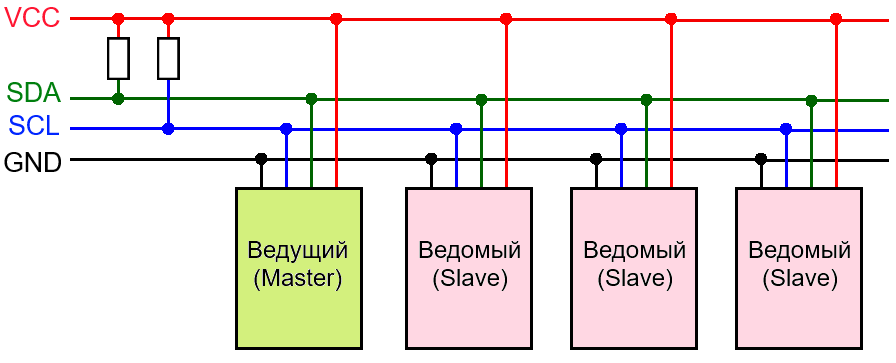
A similar memory cell was used in EPROMs with ultraviolet erasure (EPROM). In a memory cell with electrical erasure, not only writing but also erasing of information is possible. The erasure of information is made by applying to the programming gate voltage opposite to the write voltage. Unlike ROM with ultraviolet erasure, the time of erasing information in EEPROM memory is about 10 ms.

  
Fig. 1 - The design of the transistor in the EEPROM.

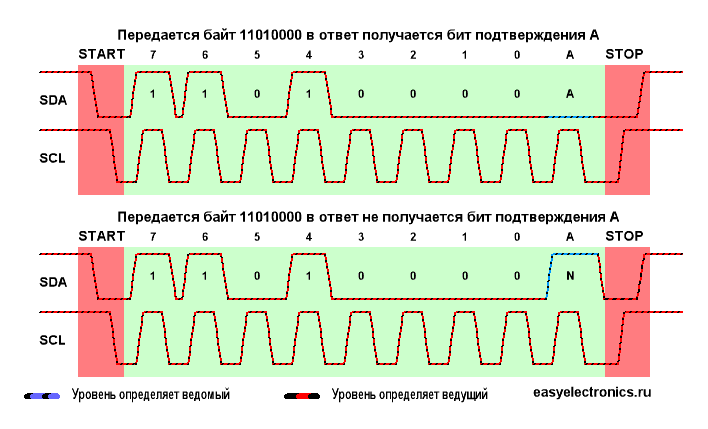
  
Fig. 2. - Structural diagram of the AT24C256 chip.

The I2C bus is used for memory operation. IIC serial communication protocol (also called I2C - Inter-Integrated Circuits). Developed by Philips Semiconductors in the early 1980s as a simple 8-bit intercom bus for developing control electronics.

wo bidirectional data lines are used for data transmission. SDA (Serial Data) serial data bus and SCL (Serial Clock) clock bus. Both buses are pulled up by resistors to the plus power bus. Transmit/Receive signals are transmitted by pushing the line to 0, it is set to one by pull-up resistors.

 Fig. 3 - I2C bus diagram.

The network has at least one master (Master), which initializes data transmission and generates synchronization signals, and slaves (Slave), which transmit data at the request of the master. Each Slave device has a unique address at which the Master accesses it. The master starts to push the SCL bus to zero at a certain frequency and the SDA bus is pushed or released for a certain number of clock cycles by transmitting a One or a Zero. Data transmission starts with the START signal, then 8 bits of data are transmitted and with the 9th bit the slave device confirms the reception of bytes by pressing the SDA bus to minus. The transmission ends with the STOP signal.

  
Fig. 4 - Data transmission via bus

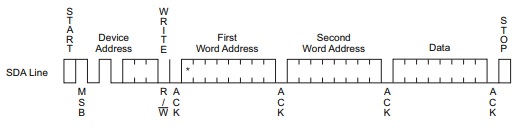
*Memory address*

It is set by three pins A0..A2. If the pin is pressed to Gnd, the bit value is 0, if it is pressed to Vcc, the bit value is 1. The chip uses an eight-bit address, the last bit is responsible for selecting the operation. If the bit value is high, the read operation is initialized, if it is low (zero), the write operation is initialized.



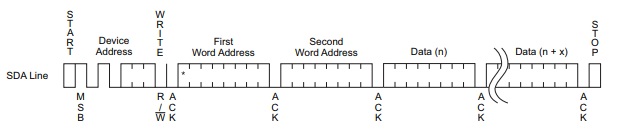
That is, if all three pins are pinned to GND and we want to write to memory, the device address will look like 0b10100000.

*Writing data to memory*



To write, we first access the memory with the Write bit in the address. Then we send two 8-bit addresses, followed by a data byte and a STOP signal. The EEPROM then enters an internally synchronized write cycle tWR (Write Cycle Time 5 ms) to non-volatile memory. All input signals are disabled during this write cycle and the EEPROM will not respond until the write is complete.

The chip memory is organized as 512 pages of 64 bytes each. That is, we can write up to 64 bytes of information in one command. To do this, we transfer all 64 bytes of information and only then send the STOP signal.

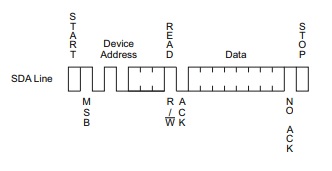


*Reading data:*

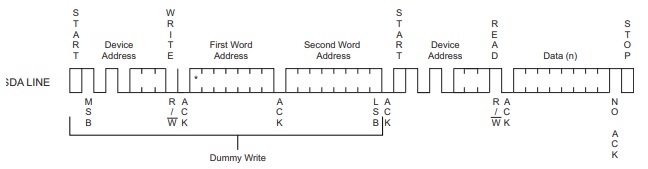
The memory supports three read options:

* Read the current address;
* Read random address;
* Serial read;

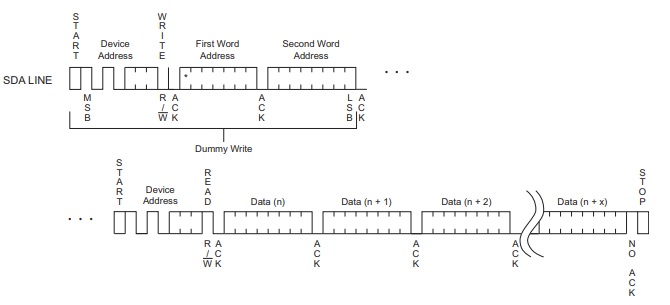
The memory remembers the last write address until the power is turned off, so we can read the last byte without specifying an address.



To read a random address it is necessary to first send a write command and pass the address we want to read. Then send a read command and get the desired byte, ending with a STOP command.



Serial reading can be performed either from the current address or from a random address and will continue until the microcontroller sends a STOP signal. In case of address overflow the memory will be reset and addressing will start from the beginning.



*Required HAL functions*

Writing data via I2C using DMA.

*HAL\_StatusTypeDef HAL\_I2C\_Mem\_Write\_DMA(I2C\_HandleTypeDef \*hi2c, uint16\_t DevAddress, uint16\_t MemAddress, uint16\_t MemAddSize, uint8\_t \*pData, uint16\_t Size)*

* *hi2с* – the pointer to the configuration structure of I2C\_HandleTypeDef type;
* *DevAddress* – the address of the memory chip;
* *MemAddress* – the address of the memory cell;
* *MemAddSize* – the size of I2C\_MEMADD\_SIZE\_8BIT or I2C\_MEMADD\_SIZE\_16BIT memory address;
* *\*pData* – the pointer to the data array of uint32\_t format;
* *Size –* the number of bytes to be transferred.

Reading data through I2C using DMA.

*HAL\_StatusTypeDef HAL\_I2C\_Mem\_Read\_DMA (I2C\_HandleTypeDef \*hi2c, uint16\_t DevAddress, uint16\_t MemAddress, uint16\_t MemAddSize, uint8\_t \*pData, uint16\_t Size)*

* *hi2с* – the pointer to the configuration structure of I2C\_HandleTypeDef type;
* *DevAddress* – the address of the memory chip;
* *MemAddress* – the address of the memory cell;
* *MemAddSize* – the size of I2C\_MEMADD\_SIZE\_8BIT or I2C\_MEMADD\_SIZE\_16BIT memory address;
* *\*pData* – the pointer to the data array of uint32\_t format;
* *Size –* the number of bytes to be transferred.

**The order of work:**

*Part I. Developing a program using a code generator.*

1. Start STM32CubeIDE, in the opened window select the path to your working folder. There should be no Russian letters in the path to the working folder and the project name. In this folder should be stored all laboratory works.

2. Take work #4 and #7 as a basis.

3. Based on the documentation, determine which pins are connected to the memory chip and which I2C block in the microcontroller is used to control it.

4. In the window of graphical initialization of the controller (name.IOC), it is necessary to configure the microcontroller pins connected to the memory chip and the necessary I2C block.

5. On the board with 7-segment indicator it is necessary to set the MEM\_Ax jumpers according to the variant.

6. In Simulink, make a circuit that allows you to set the address of a cell in memory and the value of that cell.

7. Implement the following algorithm for the program:

- On pressing one button on the stand, the microcontroller takes the address of the memory cell and the value from Simulink and writes this value into memory and then sends a flag to Simulink - that the writing has been done;

- By pressing another button on the stand, the microcontroller takes the value of the cell address from Simulink, reads the data from this cell from memory, sends the value to Simulink and displays it on the indicator.

**Tasks:**

1. Perform all of the steps in Part I.

Demonstrate all assignment items one by one to the instructor.

**Variants:**

|  |  |  |  |
| --- | --- | --- | --- |
| Variant no. | MEM\_A0 | MEM\_A1 | MEM\_A2 |
| 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 |
| 3 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 |
| 5 | 1 | 0 | 0 |
| 6 | 1 | 0 | 1 |
| 7 | 1 | 1 | 0 |
| 8 | 1 | 1 | 1 |