

MINISTRY SCIENCES AND HIGHER EDUCATION OF THE RUSSIAN
FEDERATION

FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER
EDUCATION
"NORTH CAUCASIAN FEDERAL UNIVERSITY"

METHODOLOGICAL INSTRUCTIONS
By completion of practical exercises
by discipline

**« High frequency electronic circuits /High Frequency Electronic
Circuit"**

For students directions preparation

11.03.02 Infocommunication technologies And systems connections Direction (profile)
"Intellectual processing data V infocommunication systems And networks"

Stavropol, 2026

CONTENT

INTRODUCTION	3
Practical lesson No. 1. Study features programming microprocessors on language assembler And V machine codes	4
Practical lesson No. 2. Study debug boards Arduino Uno And Wednesday developments Arduino IDE	18
Practical lesson No. 3. Programming procedures input-output bi- tovoy information	34
..... Practical lesson No. 4. Study features programming timers/counters	41
Practical lesson No. 5. Study features processing And form- worlding analog signals With using microcontrollers	49
Practical lesson #6. Development programs measurements temperatures using analog And digital sensors	61
Practical lesson #7. Study features programming liquid crystal indicators	79
Practical lesson #8. Programming PLC V environment CoDeSys	102

INTRODUCTION

Microprocessor technology in information and communication systems can be considered a special case of high-frequency electronic circuits designed to process input signals and generate output signals. Input and output signals can include analog signals, single digital signals, digital codes, and sequences of digital codes. The system may store and accumulate signals (or information), but the essence of the system remains the same. If the system is digital (and microprocessor systems are classified as digital), then the input analog signals are converted into sequences of sampling codes using an ADC, and the output analog signals are formed from the sequence of sampling codes using a DAC. Information processing and storage are performed digitally.

This course of work proposes to become familiar with the internal structure of embedded microprocessor devices used in information and communication systems.

Main affected areas:

structure of microcontrollers, structure of the microcontroller processor core, program and data memory of the microcontroller, input /output ports, hardware for ensuring reliable operation of the microcontroller.

Practical work is carried out with the aim of consolidating the acquired information and acquiring skills for its application in solving practical problems in the relevant subject area.

PRACTICAL LESSON No. 1

STUDY FEATURES PROGRAMMING MICROPROCESSORS IN ASSEMBLY LANGUAGE AND IN MACHINE CODES

Objective of the work: To study the structure of an eight-bit microprocessor and to acquire skills in programming typical control tasks in assembly language and machine codes

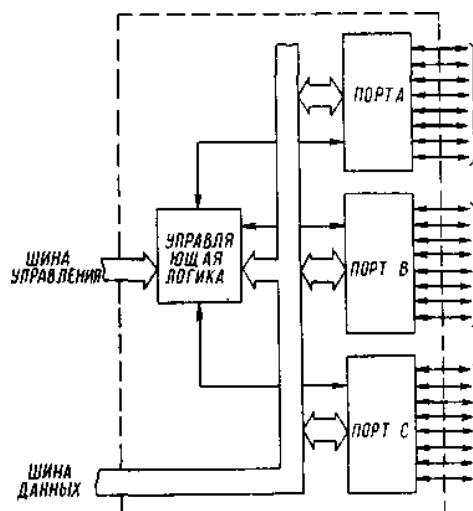
Educational questions:

1. Input one-time teams And their indication
2. Implementation conclusion information on seven-segment indicator
3. Input data With keyboards
4. Conclusion data on LED indicator V dynamic cut -

me

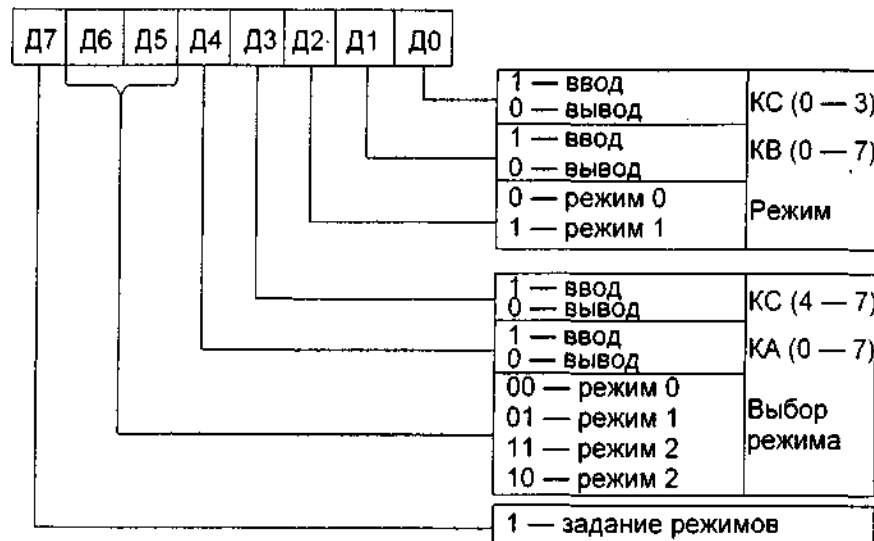
1. Input one-time teams And their indication

Eight-bit general-purpose microprocessors such as Intel 8080 or KR580VM80A served as the architectural basis for many subsequent generations of microprocessors. A feature of low-level programming is the need to know the composition and structure of the microprocessor at the level of programmable registers. The KR580VM80A microprocessor can address up to 256 input ports and the same number of output ports. The microprocessor laboratory "Microlab" has input /output ports that are implemented on a programmable peripheral interface (KR580BB55 microcircuit, Figure 1). The ports of this interface can be either output ports or input ports, depending on how the interface is programmed.



Drawing 1- Programmable parallel interface

The port registers have the following addresses in hexadecimal format : Port A - F8, Port B - F9, Port C - FA and the control register - FB. Three bits of port C of this interface are connected to toggle switches. This port will be programmed as an input port. LED indicators are connected to eight bits of port B. This port will be programmed as an output port, i.e., the microprocessor (MP) will be able to read data from the toggle switches and output it to the LED indicators. The first control word format is used for programming the microcircuit ; the assignment of the bits is shown in Figure 2.



Drawing 2- Format 1 manager words

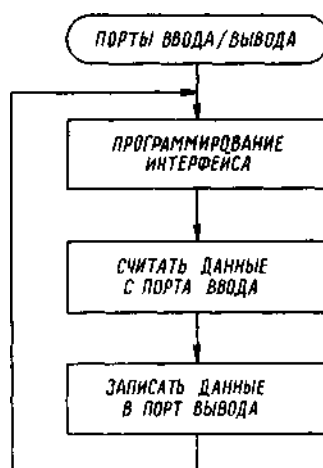
Figure 3 shows algorithm diagram, demonstrating the use of these ports. In this example data is read from input port C and written to output port B. Listing of the program corresponding to the algorithm presented in Figure 3, shown in Table 1.

To program the interface as intended (C is the input port, B is the output port), it is necessary to supply a control word with the value 81 to the address of the control register FB. The first command MVI A, 81 means loading the accumulator with the code 81. The next command OUT FB writes down content battery V port at the FB address, T. e. The port (interface) is programmed. Next comes the IN FA command, which writes data from the port with the address FA, assigned to port C, to the accumulator. The OUT F9 command outputs data from the accumulator to the output port at address F9. Since this address is assigned to port B in our case, the data is output from the accumulator to port B (to the LED indicators). Thus, the program transfers data from the input port to the output port. The last unconditional jump command, JMP, transfers control to the beginning of the loop. The loop executes continuously until the RESET button is pressed.

For execution programs complete next actions:

1. Enter program, indicated V table 1.
2. Get Started execution programs With addresses 8000.

3. Set the input port toggle switch to any position. The upper position corresponds to 1, the lower one to 0.
4. Take a look on LEDs 2-4, counting on the right. They show those the same data that is set on the input port switches.
5. Change the position of the input port switches. The output port data should change accordingly.
6. Click button RESET. Program stop, control will move to the monitor .
7. Change the position of the input port switches. The output port data will not change, since the entered program is no longer running.



Drawing 3- Scheme algorithm For forwarding data from port input V output port

Table 1- Listing programs For censuses data from port input V port conclusion

Address	Content	Label	Team	Comments
8000	ZE	START:	MVI A, 81	Recording V battery data For
8001	81			programming interface
8002	D3		OUT FB	Programming interface
8003	FB			
8004	DB		IN FA	Reading data from port input
8005	FA			
8006	D3		OUT F9	Recording data V output port
8007	F9			
8008	SZ		JMP START	Cycle
8009	04			
800A	80			

It should be noted that, if desired, before writing data to the output port , you can perform any actions on them provided by the MP command system . For example, can be considered data from port input, invert them and transfer them to the output port. To do this, a command providing data inversion (SMA) must be added to the program (see Table 1) .

The microlab's ROM contains a monitor program that reads data from the keyboard, executes the selected operation, and controls the display. The microlab runs the monitor program at all times, except when it is executing a user program. When the START button is pressed, the monitor program causes the processor to jump to the address indicated on the " Microlab " display. When the SBR button is pressed, the microlab returns to the monitor program. The monitor program allows you to check the contents of the registers in a step-by-step mode after executing each command (i.e., after each step). On the third and fourth indicators, counting from the right, after executing each command (step-by-step mode), the contents of the accumulator are displayed. In addition, after each step of the command, the monitor program writes the contents of the registers to special RAM cells. Therefore, you can check the contents of the registers at each step by viewing the corresponding RAM cells (table 2).

Table 2 - Addresses registers MP

Address	Register
83EV	Battery
83EA	Register signs
83E9	To the register
83E8	With register
83E7	D register
83E6	E register
83E5	N register
83E4	L register
83EZ	Pointer stack (high byte)
83E2	Pointer stack (low byte)
83E1	Program counter (senior byte)
83EO	Program counter (younger byte)

2 Implementation conclusion information on seven-segment indicator

The microlaboratory uses eight seven - segment digital indicators for display. Each indicator corresponds to a memory cell, which stores a seven-segment code that controls the illumination of the indicator segments. Information from these cells is sent to indicators using a special circuit or program that provides a dynamic indication mode.

The task of the software is to prepare seven-segment codes and send them to eight memory cells with addresses starting from 83F8 to 83FF. Cell 83F8 corresponds to the left indicator, 83FF to the right

The microlab monitor contains the SEGCG program (address = 01C0), which converts hexadecimal codes 0, 1, 2... E, F into seven- segment codes and stores them in memory cells 83F8...83FF. The initial data for this program must be in cells 83F4...83F7. Each byte from these cells corresponds couple indicators: content cells 83F4 By-

After conversion, it will be displayed on the two left indicators, and the contents of cell 83F7 will be displayed on the two right indicators.

Table 3 shows the information display program using the SEGCG subroutine. Enter the program and check its operation. The indicators should sequentially display the symbols 1, 2, 3, 4, A, B, C, D.

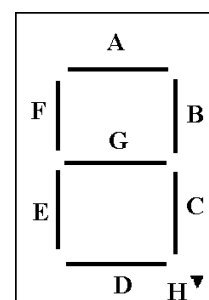
Table 3- Program conclusion information on simisegmental indicators

Address	Contents	Tags	Team	Comments
8000	ZE		MVI A, 12	12 → [A]
8001	12			
8002	32		STA 83F4	A → [83F4]
8003	F4			
8004	83			
8005	ZE		MVI Ah, 34	34 → A
8006	34			
8007	32		STA 83F	A → [83F5]
8008	F5			
8009	83			
800A	ZE		MVI Oh, AB	AB → A
800W	AV			
800C	32		WHAT 83F6	A → [83F6]
800D	F6			
800E	83			
800F	ZE		MVI Oh, CD	CD → A
8010	CD			
8011	32		WHAT 83F7	A → [83F7]
8012	F7			
8013	83			
8014	CD		CALL SEGCG	Challenge SEGCG
8015	C0			
8016	01			
8017	76		HLT	Stop

The previous program used the SEGCG subroutine. But this subroutine operates only With hexadecimal symbols

— 0 ... F, i.e. does not fully utilize the capabilities of the seven-segment indicators . It is possible to form a seven-segment code and receive on the indicators any symbols that are possible to obtain using the indicators used in the microlaboratory . As already mentioned, seven-segment codes must be sent to specific memory cells. Each bit in these cells corresponds to a specific segment (Figure 4). If a bit is 1, then the corresponding segment will be "lit", and vice versa.

Бит	7	6	5	4	3	2	1	0
Сегмент	H	G	F	E	D	C	B	A



Drawing 4- Correspondence discharges And segments indicator

Table 4 shows a listing of the program that displays the inscription “ ScFU ” on the indicators . By changing the seven-segment codes, display the inscription corresponding to the number of the study group, for example Ibt191.

Table 4- Program conclusion symbols on indicators without subroutines SEGCG

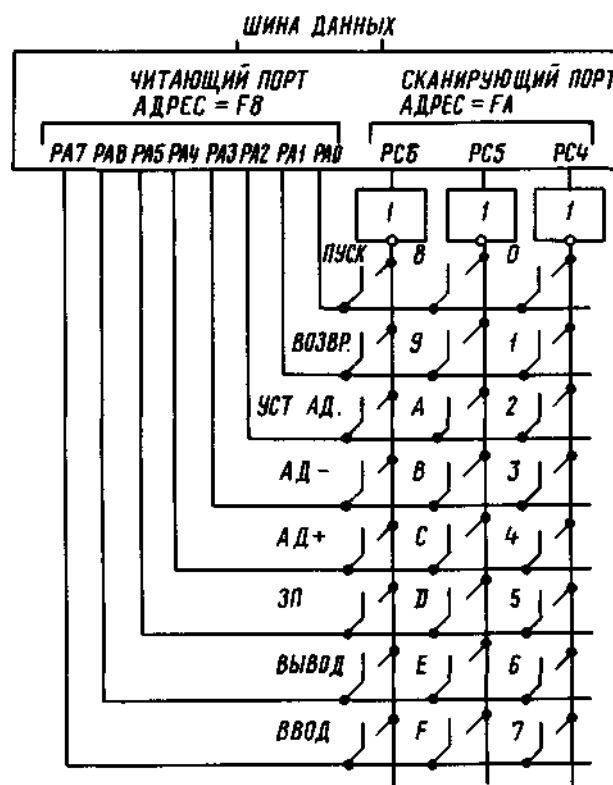
Address	Content	Label	Team	Comments
8000	ZE		MVI A, 6d	Seven-segment code letters S → A
8001	6d			
8002	32		STA 83F8	A → [83F8]
8003	F8			
8004	83			
8005	ZE		MVI A, 58	Seven-segment code letters c → A
8006	78			
8007	32		STA 83F9	A → [83F9]
8008	F9			
8009	83			
800A	ZE		MVI A, 71	Seven-segment code letters F → A
800B	30			
800C	32		STA 83FA	A → [83FA]
800D	FA			
800E	83			
800F	ZE		MVI A, 3E	Seven-segment code letters U → A
8010	6d			
8011	32		STA 83FB	A → [83FB]
8012	FB			
8013	83			
8014	76		HLT	Stop

3 Input data With keyboards

The microcomputer has 25 keys and mode switches on the front panel, providing direct and convenient communication with the microprocessor system without any additional hardware. The keyboard has a matrix layout, except for the RESET button. Keyboard query and the processing of its codes is performed by the monitor program using a programmable parallel interface (PPI). The PPI microcircuit contains three programmable 8-bit input/output buses. To poll the keyboard, the 4 most significant bits of the port C bus are programmed to output , and port A to input. The monitor program outputs data to the C outputs in turn PPI microcircuits low level pulses, reading after each output of information from port A (Figure 5). Each digit of port C is supplied to 8 buttons that make up a row. Each input of port A is an assembly of three buttons from different rows. If none of the buttons in a row are which filed short level, Not pressed, V port A will arrive code FF₁₆, es -

If at least one button is pressed, then a log. 0 will appear in the corresponding bit of port A.

Considering number rows And discharge port A, V which at input was log. 0, the monitor generates the code of the pressed button and operates accordingly to the required subroutine that performs the operation corresponding to this button .



Drawing 5- Simplified scheme keyboards

The read data is converted into a code corresponding to the pressed button using the KEYIN monitor program (address 0216h). This program is called using the CALL KEYIN command. After the KEYIN program returns , the accumulator will contain the code of the pressed button (Table 5). This monitor program can be used to read the keyboard .

Table 6 shows the listing of the program that causes a sound signal if button 7 is pressed. The keyboard is read using the above -described KEYIN program, which stores the code of the pressed key in the battery .

The CPI 07 command sets the processor zero flag if the accumulator contents are 07. The JNZ READ command causes the program to return to the beginning if the zero flag is not set. If the zero flag is set, the BEEP beeper subroutine (address 0350) will be called. The process then repeats. Thus, the beeper will be generated when button 7 is pressed.

Enter listing programs V microlaboratory And check right-

error . Run the program. Press button 7 - the speaker hums. Press any other button - there is no sound.

Table 5- Codes microlaboratory buttons

Button	Code (Hex)
0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
A	0A
In	0V
S	0C
D	0D
E	0E
F	0F
PUSK	10
VOZVR	11
MOUTH AD	12
AD -	13
AD +	14
ZP	15
CONCLUSION	16
INTRODUCTION	17

Create a program that generates a beep when three specific buttons are pressed in a certain sequence, for example, 7, 3, 9. This program can be used as a basis for creating an electronic lock. Start creating the program with a block diagram. Enter the program into the microlab and test its execution.

The keyboard scanning is performed cyclically by the READ subroutine . In one cycle , data is read from one row of eight buttons (see Fig. 5). For simplicity, we will consider only one row, i.e. buttons 0, 1, 2...

7. Reading a row of buttons is a two-step operation: the first step is to write data to the scan port to select the buttons , and the second step is to read a byte of data from the read port .

To select a specific row of buttons, the corresponding bit of the scanning port (PC4, PC5 or PC6) is set to logic 1, the other bits remain equal to logic 0. Thus, to select row of buttons 0...7, you need to send code 1001 1111 (9Fh) to the FA port to select the buttons 8 ... F - code 1010 1111 (AFh) and to select the START . . . ENTER buttons, - code 1100 1111 (CFh). Bits PC0...PC3, PC7 are not used when scanning the keyboard -

are And are being installed equal log.1. So, to count data With rows 0 ... 7, is output to the port byte 9F. Then the input port reads to collect information about the button states.

Table 6 - Program call sound signal

Address	Content	Label	Team	Comments
8000	CD	READ	CALL KEYIN	Call programs, readings data from the keyboard
8001	16			
8002	02			
8003	FE		CPI 07	Comparison code key With 07
8004	07			
8005	C2		JNZ READ	Return, If button 7
8006	00			Not pressed
8007	80			
8008	CD		CALL BEEP	Sound signal if button 7 is pressed
8009	50			
800A	03			
800V	SZ		JMP READ	Repetition programs
800C	00			
800D	80			

When pressed certain buttons, the corresponding bit of the reading port will be set to 0 (Table 7). Please note that the button codes given in Tables 5 and 7 do not match. The button from other rows when reading the selected row does not affect the reading results.

Table 7- Codes, readable from the keyboard

Button	Readable code (binary)	Readable code (HEX)
No pressed buttons	11111111	FF
0	11111110	FE
1	11111101	FD
2	11111011	FB
3	11110111	F7
4	11101111	EF
5	11011111	DF
6	10111111	BF
7	01111111	7F

Listing, presented V table 8, shows program, implementing the reading process just described .

Table 8- Program definitions states buttons

Team	Comments
MVIA,9F OUT FA IN F8	Choice rows buttons 0 ... 7 Reading button states

The KEYIN monitor program not only scans the keyboard, but also converts it codes, readable With columns (V dependencies from selected row), in the codes given in Table 5. Table 9 shows the program for checking the state of button 2, without using the KEYIN subroutine. If button 2 pressed, is generated sound signal. Dial program on Microlab And check her work.

Table 9- Program checks states buttons 2

Address	Content	Label	Team	Comments
8000	3E	READ:	MVI A, 90	Programming port A for data input and port C for output
8001	90			
8002	D3		OUT FB	
8003	FB			Choice rows buttons 0 ... 7
8004	ZE		MVI A, 9F	
8005	9F			
8006	D3		OUT FA	Reading column states
8007	FA			
8008	DB		IN F8	
8009	F8			Comparison battery With ko-house (button 2)
800A	FE		CPI FB	
800V	FB			
800C	C2		JNZ READ	Read again if button 2 is not pressed
800D	04			
800E	80			
800F	CD		CALL BEEP	If pressed button 2, appear - etsya sound signal
8010	50			
8011	03			
8012	SZ		JMP READ	Repetition programs
8013	04			
8014	80			

4 Conclusion data on LED indicator V dynamic mode

Output port indicators (LEDs) can be used to display one-time (bit) commands. A "running lights" simulator is built on this principle . The running lights controller must regulate the following LED lighting sequence:

1. They're burning LEDs 1, 4, 7, rest repaid.
2. Delay time combustion established LEDs.
3. They're burning LEDs 2, 5, 8; rest repaid.
4. Delay time combustion established LEDs.
5. They're burning LEDs 3, 6; rest repaid.
6. Delay time combustion established LEDs . .
7. Repetition process (transition To step 1).

On drawing 6 given algorithm work controller.

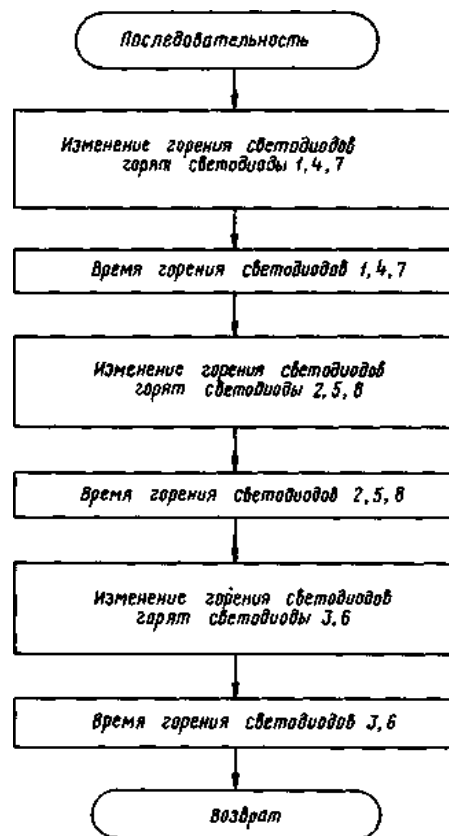


Figure 6 - Algorithm of the LED burning sequence program Output indicators

(LEDs YD1...YD8) connect To tire data using a programmable interface. Table 10 shows the combinations of signals used to simulate "running lights" on LEDs . A one in the binary code of the signal means that the indicator is on .

Table 10- Meaning one-time teams For implementation "running lights"

VD1	VD2	VD3	VD4	VD5	VD6	VD7	VD8	Code
1	0	0	1	0	0	1	0	92 ₁₆
0	1	0	0	1	0	0	1	49 ₁₆
0	0	1	0	0	1	0	0	24 ₁₆

If a program were to implement a program that would issue one-time commands sequentially without a delay, the LED frequency would be very high, and the running light effect would not be observed. Therefore, it is necessary to develop a subroutine that would cause a delay of a specified duration. The simplest way to generate a delay is to execute commands cyclically. To increase the delay duration, register pairs and nested loops are used.

is set up in register D to obtain a delay. The number initially written to register D is determines, How many once is being carried out delay duration V 0.786 With.

This delay is generated by the inner loop using the BC register pair . The maximum delay obtained by two cycles is thus $256 \times 0.786 = 201 \text{ s}$.

Table 11 shows the delay program used for the running lights controller . The LXI B, 0 command places 0 in the register pair B, C. A delay of 0.786 s is then generated. Register D is used to count the main delay.

Table 11-. Delay program

Tags	Teams	Comments
DELAY: LOOP:	LXI B, 0 DCX IN MOV A, B ORA C JNZ LOOP DCR D RET	Start accounts internal loops Loop for- holders 0.786 s Decrease main counter And work on the loop, if $\neq 0$

The basic sequence program is quite simple (table 12). At first program sets V register A number 92 — code output indicators, in which LEDs 1, 4, 7 are lit, and LEDs 2, 3, 5, 6 are off. Then it calls the delay subroutine. To do this, the sequence program sets the delay value in register D, after which the delay subroutine itself is called . Then this sequence is repeated for other LED output signal codes.

For checks programs:

1. Enter program from tables 12 V RAM.
2. Make sure, What she Right recorded V memory.
3. Do it the main one program, beginning By address 8000.
4. The time delay values can be replaced with other values to reduce the speed of the “running lights” at the addresses 8009, 8012, 801B, 8023, 8024.

By changing the code values, achieve the opposite direction of movement of the "running lights".

Make adjustments to the program text so that the speed of movement of the "running lights" changes upwards and downwards .

Exercise For independent work

Run all example programs. A report containing the program text in assembly language and machine code with the necessary comments is required for all lab assignments . Conclusions must be drawn regarding programming technology using assembly language and machine code.

Table 12- Program controller running lights

Addresses	Codes	Tags	Team	Comments
8000	ZE	FIRE:	MVI A, 81	Programming interface
8001	81			
8002	D3		OUT Facebook	
8003	Facebook			
8004	ZE	SEQ:	MVI A, 92	Installation code combustion 1.4, 7 light- emitting diodes
8005	92			
8006	D3		OUT F9	
8007	F9			
8008	16		MVI D,01	Combustion this sequences
8009	01			
800A	CD		CALL DELAY	Call subroutines delays
800V	22			
800C	80			
800D	ZE		MOV A, 49	Installation code combustion 2, 5, 8 light- emitting diodes
800E	49			
800F	D3		OUT F9	
8010	F9			
8011	16		MVI D, 01	Combustion this sequences
8012	01			
8013	CD		CALL DELAY	
8014	22			
8015	80			
8016	ZE		MOV A, 24	Setting the combustion code for 3, 6 LEDs
8017	24			
8018	D3		OUT F9	
8019	F9			
801A	16		MVI D, 01	Combustion this sequences
801B	01			
801C	CD		CALL DELAY	
801D	22			
801E	80			
801F	SZ		JMP SEQ	Return V start basic programs „running lights"
8020	04			
8021	80			
8022	01	DELAY:	LXI B,0000	
8023	00			
8024	00			
8025	OB	LOOP:	DCX B	Internal loop, generating for-support 0.786 With
8026	78		MOV A, B	
8027	B1		ORA WITH	
8028	C2		JNZ LOOP	
8029	25			
802A	80			
802B	15		OCR D	Main loop delays
802C	C2		JNZ DELAY	
802D	22			
802E	80			

802F	C9		RET	Return
------	----	--	-----	--------

PRACTICAL LESSON No. 2

STUDY DEBUGGING BOARDS ARDUINO UNO AND DEVELOPMENT ENVIRONMENTS ARDUINO IDE

Goals laboratory works:

- познакомиться With platform Arduino Uno And integrated programming environment Arduino IDE;
- study compound And possibilities integrated Wednesday programming Atmel Studio 7.

Educational questions:

1. Technology programming microcontrollers With using a debug board Arduino Uno and the Arduino IDE
2. Compound And possibilities integrated Wednesday Atmel programming Studio 7

1. Technology programming microcontrollers With using a debug board Arduino Uno and programming environments Arduino IDE

To effectively learn the basics of programming in the Arduino environment , it is necessary to study the implementation of basic algorithms in the Arduino environment and provide examples of software writing.

The programming language used in Arduino projects is based on C and C++, the most widely used programming languages , supporting both working with low-level commands and building complex objects.

Arduino controllers is conveniently carried out in a special Arduino IDE environment , since it includes the basic functionality for working with them.

Installing the Arduino IDE. You can learn the basics of programming on the Arduino platform using both a range of physical devices and the Electronics Lab online virtual environment provided by Autodesk . on the website <https://circuits.io>. The Arduino IDE can be downloaded from the Internet at the project's main website: www.arduino.cc/en/Main/Software. The software is freely distributed and should be downloaded in the Download section. the Arduino IDE Windows option Installer (Figure 1), and then JUST DOWNLOAD (Figure 2).



Drawing 1 — Window downloads programs Arduino IDE

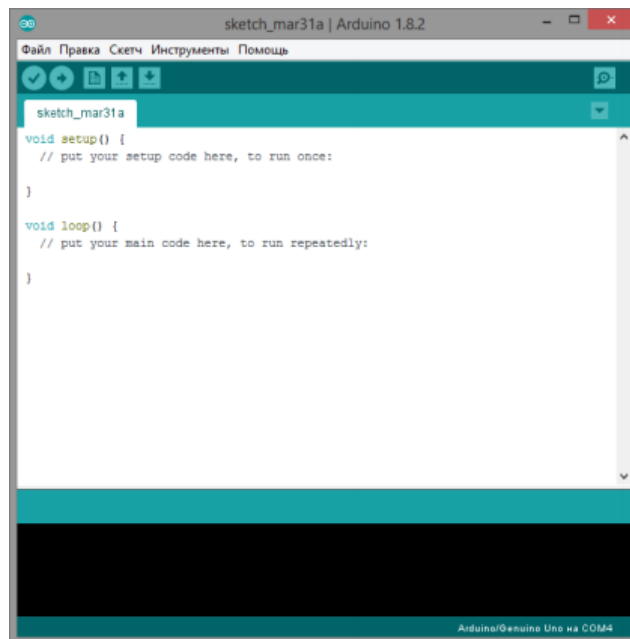


Drawing 2 — Button downloads installation file Arduino IDE

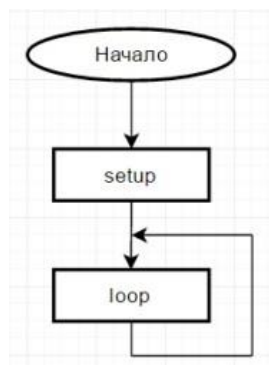
Arduino IDE program on your computer - Currently, this is the file arduino-1.8.13-windows.exe. You must run it with administrative privileges, accept the terms of the GNU Lesser General Public License, and agree to the proposed installation option . You should answer affirmatively to all Windows warnings during the installation (continue installation). When the installer offers to install port drivers, also answer affirmatively .

Getting started with the Arduino IDE. The development environment has an intuitive Russian-language interface. When you launch the installed Arduino IDE, a window will open (Figure 3), which already contains a program template . It consists of two functions: setup and loop . The setup function contains commands that are executed once when the Arduino is turned on . These include setting the I/O port numbers for controlling monitors and setting the data transfer rate between the Arduino and the computer. The loop function fulfilled -

is repeated an infinite number of times - until we turn off the power, in fact, it is in a loop, algorithmically this is depicted in Figure 4. The interface allows you to implement the previously studied basic algorithms.



Drawing 3 — Window Arduino IDE



Drawing 4 — Typical algorithm programs Arduino IDE

Connecting the Arduino controller to a PC . The capabilities of the environment allow the use of different USB cables depending on the Arduino controller (Figure 5):

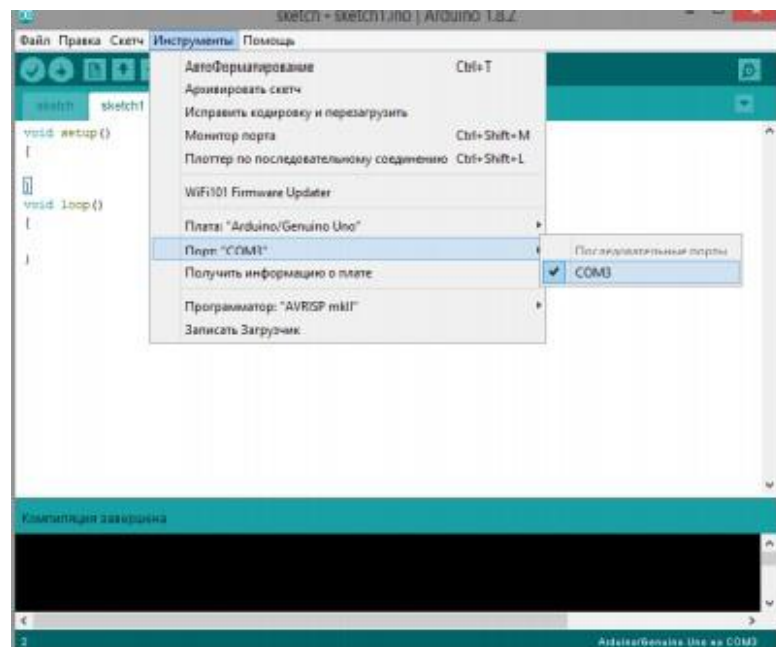
- for Arduino UNO R3 and Mega - this is a cable with a USB printer connector (Type B) on one end and a standard USB connector (Type A) on the other;
- For controller Nano required cable With connector mini - USB;
- for Arduino Micro is micro USB, and for programming the controller Arduino Mini and Pro Mini will require a programmer as they do not have a standard interface to connect to a computer.



Drawing 5 — Options USB cables: A) USB type B For Arduino UNO and Mega; б) cable for Arduino Nano ; в) micro -USB for Arduino Mini

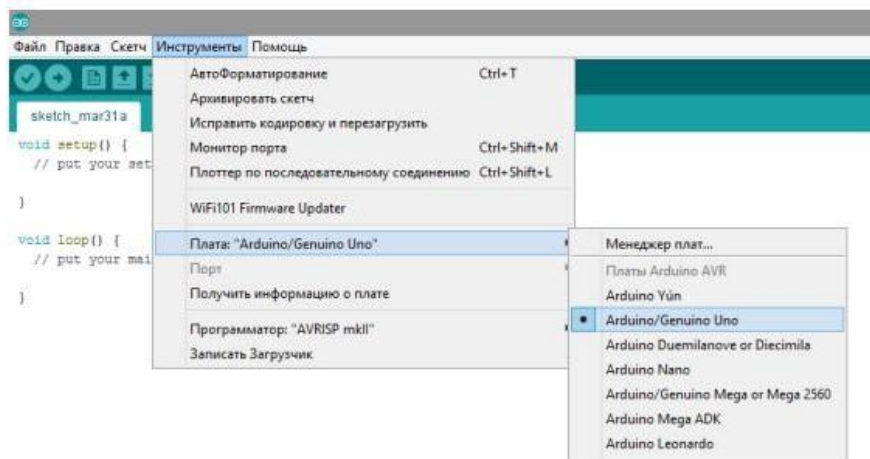
Some controllers require a non-standard driver to work. They require the installation of a separate driver that is not included in the kit . Arduino IDE—its name is ch341ser.exe. It's not hard to find, and installing the driver will solve the problem.

The controller connects to the PC via a USB interface and establishes connections between it and the Arduino IDE. To do this, you need to specify the port number to which the controller is connected (Figure 6). If there are many ports and it's difficult to find the one you need, it's recommended to memorize all the available ones, then physically disconnect the Arduino from the cable and re-examine the list of ports. The one that disappeared is the one you need. After connecting , you need to select the desired port. Check the corresponding box . Sometimes it takes some time for a port to appear in the list while the computer's operating system analyzes and checks the connected device, so you should wait a little while for this process to complete .



Drawing 6 — Choice needed port

the Arduino controller type . In Figure 7, you can see that Arduino / Genuino is selected. Uno .



Drawing 7 — Choice types controller Arduino

If the system automatically selects the controller, you need to verify that it's selected correctly. For some controllers, you also need to select the microcontroller type on which the Arduino board is implemented . The microcontroller name can be found on its chip.

- this is, as a rule, the largest microcircuit on the board and it is located in its center.

C++ syntax . Before you begin learning C / C++, you need an integrated development environment (IDE). The C/C++ language is a set of commands. This set of commands is also called code or source code.

Uploading a program . The program written in the Arduino IDE is uploaded to the controller by pressing the right arrow button (see Figure 3). The shell will check the program for errors, then translate it into binary code for data and commands of the selected microcontroller and write it to the Arduino . If after the word Upload appears in the bottom line of the Arduino IDE window , the TX and RX LEDs on the Arduino board blink , then the program is uploaded . The download to the board started successfully. If the message "Download complete" does not appear after this , then the Arduino board type is most likely incorrect . If the TX and RX LEDs do not blink at all, then the problem lies in the selected Arduino board port . If everything goes smoothly, the message "Download complete " will appear , and the program will automatically begin executing. If an empty program is downloaded, nothing will happen—the empty code will repeat indefinitely as long as the board is powered.

Let's look at Arduino programming using the example of a control program built-in LED, attached to the 13th pin of the board connector (Figure 8).

Digital ports can only receive and transmit information as a sequence of ones and zeros. In the example shown, this sequence is transmitted to port 13—visually indicated by the blinking LED on the controller board. A feature of the scenario Arduino IDE is usage built-in subroutines And

functions, A Also opportunity use logical names microcontroller registers .

```
void setup() // настройка
{
    pinMode (13, OUTPUT); // перевод 13-го порта в состояние вывода информации
}

void loop() // основная программа
{
    digitalWrite (13, 1); // включение светодиода на плате
    delay (1000);         // задержка в 1 секунду
    digitalWrite (13, 0); // выключение светодиода на плате
    delay (1000);         // задержка в 1 секунду
}
```

Drawing 8 — Program blinking LEDs

When LED is on fire, on port high electric voltage 5V, and when off - low (0V).

First built-in subroutine defines appointment 13 output of the connector located on the debug board (Figure 9).

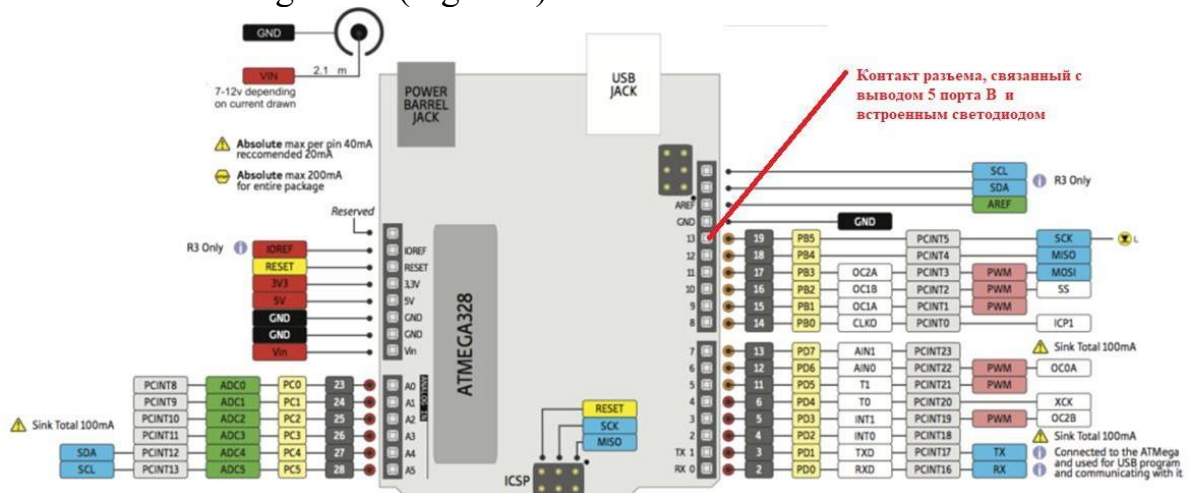


Figure 9 – Pin assignment of the debug board Arduino Uno Syntactic

analyzer, built-in in

Arduino IDE, highlights go

known logical variable names and function words are shown in blue , and in red are the names of built-in subroutines and functions.

Assigning pin 5 of port B to output a bit of information (OUTPUT) can also be done in the following way:

$\text{DDRB} |= 1 < 5;$

In the program, written with using Atmel Studio conclusions ports intended for receiving information are assigned by default. Arduino IDE is required execute the pinMode subroutine with argument - volume INPUT, for example for PORTD7:

pinMode (7,INPUT).

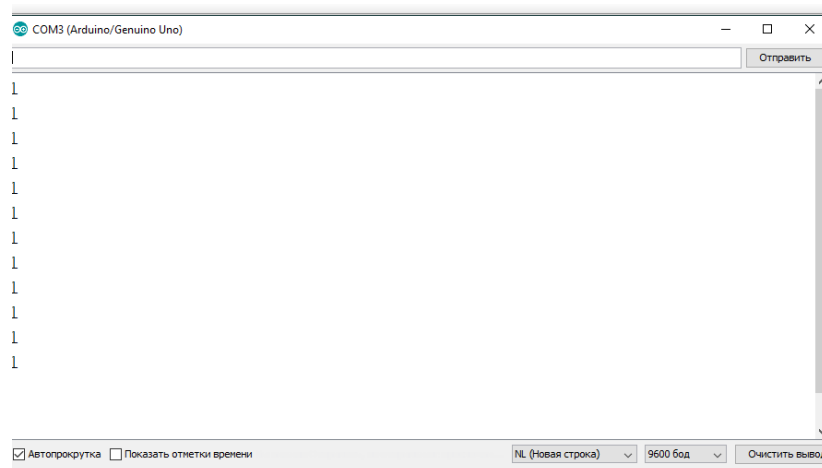
Subroutines are used to output and receive bit information . digitalWrite And

digitalRead With arguments V view numbers previously configured -

output and the value to output. The output state can be set using reserved variables HIGH to set the value to one and LOW to set the value to zero.

Built-in subroutine delay (1) carries out delay on 1 ms.

Program operation monitoring. Program monitoring allows you to debug the device. Serial library routines are used to exchange information between the personal computer and the microcontroller during program execution . The serial communication protocol is configured by setting the following parameters in the section setup subroutines Serial.begin (9600), which specifies the speed at which information is exchanged with the PC. Information exchange requires that the receiver and transmitter communicate at the same speed, and as a controller Arduino is quite slow, so the most suitable speed for exchanging information with it is 9600 baud. Exchanging information at higher speeds is also possible, but it uses more controller resources , and interrupts of the current program occur more often, and the main code Maybe be carried out slower. Team Serial.println () transmits from the controller to the PC, the value indicated in brackets. The result of the work is shown in Figure 10. To obtain the obtained results on the computer screen after starting the program, you will need to open the Tools | Port Monitor tab and adjust the port speed (9600 baud) - in the Port Monitor window, the speed is set in the lower right corner by selecting from the pop-up list .



Drawing 10 — Conclusion results on screen computer With using a serial protocol

Variables. Variables are an important component of programming language syntax . Variables are memory areas that have a name, which is also called an identifier. Before working with variables their need to announce, How And V any another environment programming, such as Turbo Pascal . In C/C++ all variables must be declared . This means, What, Firstly, V beginning each programs or func -

tions must be given list everyone used variables, A secondly, the type of each of them is indicated (Figure 11).

```
// переменные видны в программе везде
int a;
float b;

// переменная видна только внутри функции setup
void setup() // настройка
{
    int c;
    c=10;
    Serial.begin(9600); // скорость порта связи Arduino - ПК
    a=c*5; //значение переменной a изменится на 10*5
}

void loop() // основная программа
{
    // следующая строка содержит ошибку
    c=a+5; // переменная c не может быть видна за пределами функции setup
    delay(a*100); // задержка 10*5*100 миллисекунд
    Serial.println(millis()); // передача на ПК время работы в миллисекундах
    delay(1000); // задержка 1 секунду
}
```

Drawing 11 — Definition variables. Zones visibility. Message about the error

Figure 11 contains a deliberately added variable declaration error for educational purposes, and when you try to compile it, you will get the error message: 'c' was not declared in this scope . Variables can differ in data type, i.e. in the size of memory cells (in bytes) reserved for these variables (Table 1).

Table 1 — Data types

Designation in program	Name	Accepted values	Size memory
boolean	Logical	True / False 1/0	1 byte
char	Symbolic	Code ASCII	1 byte
byte	Short unknown - kovy	0- 255	1 byte
int	Whole	- 32768...32768	2 byte
long	Long whole	- 2147483648... 2147483648	4 йайта
float	With floating point	-3, 4028235 x 10 ³⁸ - 3,4028235 x10 ³⁸	4 byte

, it is important to understand that they are not a perfect storage of information - for example, integer variables can overflow. This happens when the value,

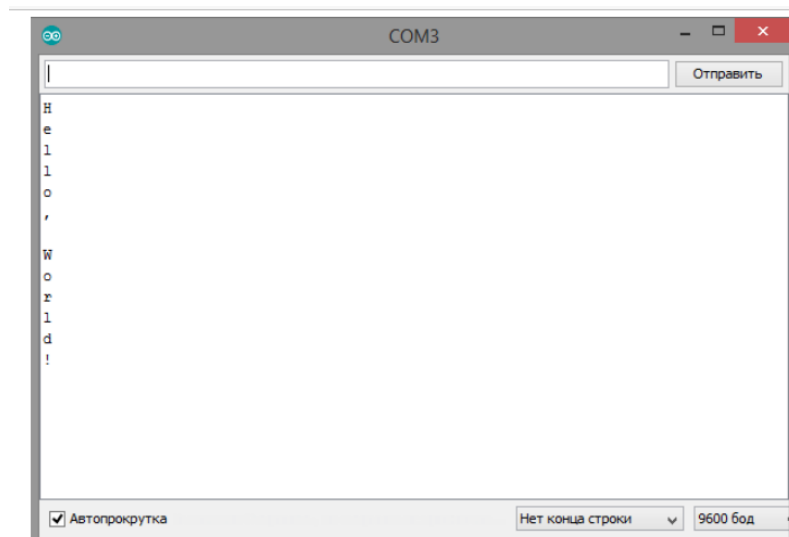
which should be written into the variable is greater than the maximum possible for this type data. Variables With floating point are subject to Another problem is that they round their values when adding a large number to a small one. Also, the Arduino IDE doesn't always handle data type conversion correctly. The following example demonstrates a small program that receives data from a computer via an I/O port (Figure 12).

```
void setup() // настройка
{
  Serial.begin(9600); //скорость порта связи Arduino - ПК
}

void loop() // основная программа
{
  char symbol; // переменная символьного типа данных
  if (Serial.available() > 0)
  {
    symbol=Serial.read(); // считывание символов из переменной
    Serial.println(symbol); // вывод на ПК то, что было считано
  }
}
```

Drawing 12 — Receipt data from computer through port data input/output

Once you've compiled and loaded this program, you can type "Hello, World!" into the top portion of the Port Monitor window and send it to the port. The program will send the message back to your PC character by character (Figure 13).



Drawing 13 — Conclusion messages on PC

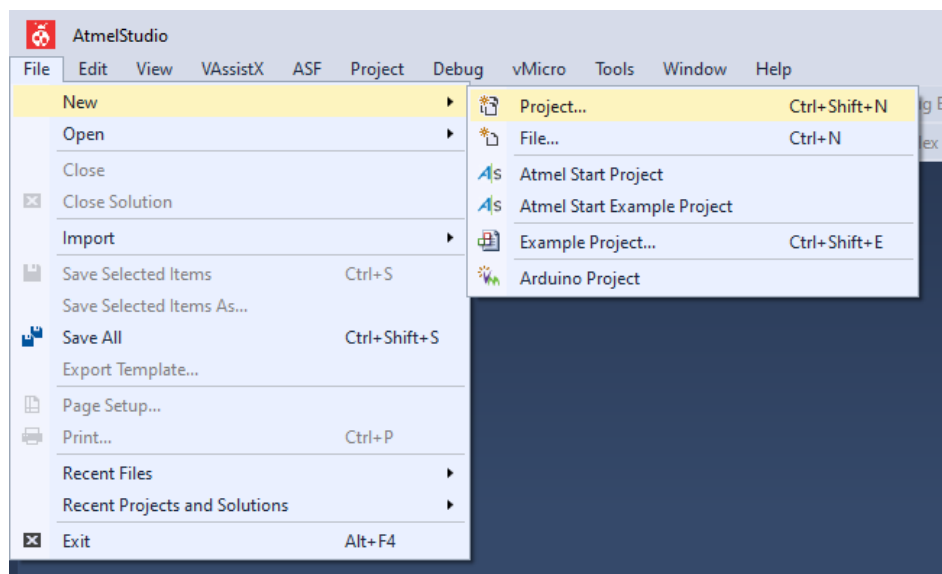
2. Compound And possibilities integrated Wednesday Atmel programming Studio 7

Atmel Integrated Development Environment Studio boots like normal Windows application. Atmel splash screen appearance The current seventh version of the studio is shown in Figure 14.



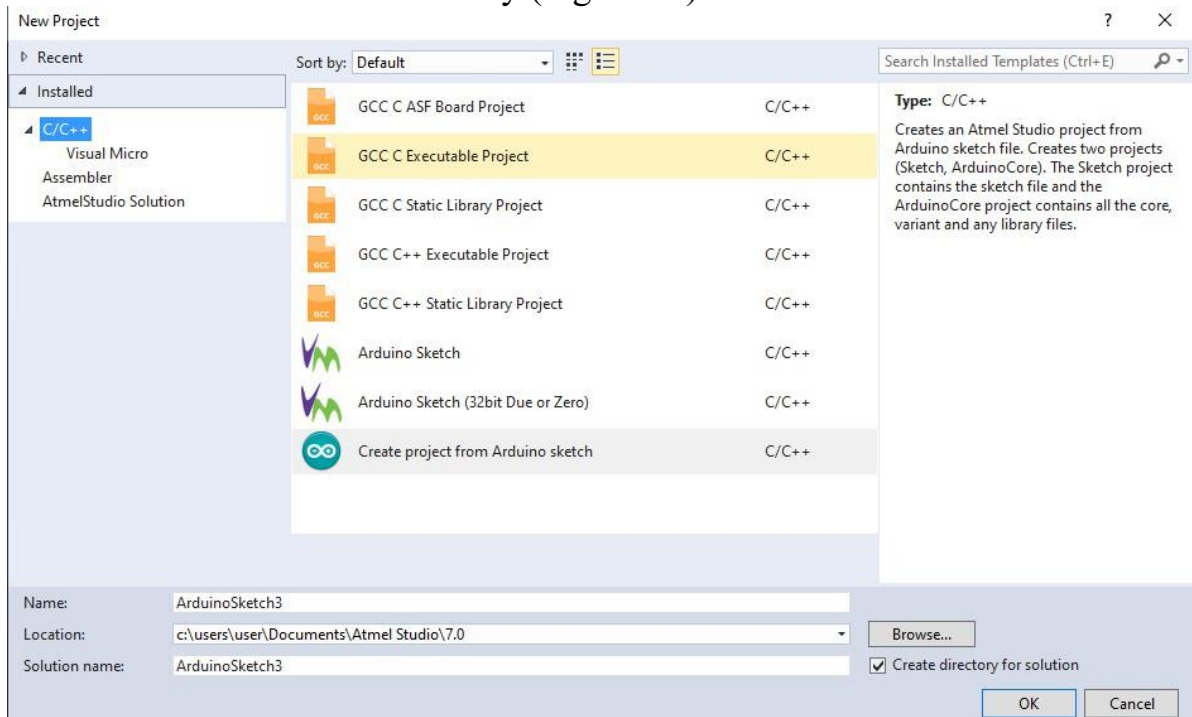
Figure 14 – Appearance of the Atmel Studio7 program during loading After

downloads opens working window programs V which guesses - acquaintances are coming for everyone Windows applications means work with files File – New-Project (Figure 14).



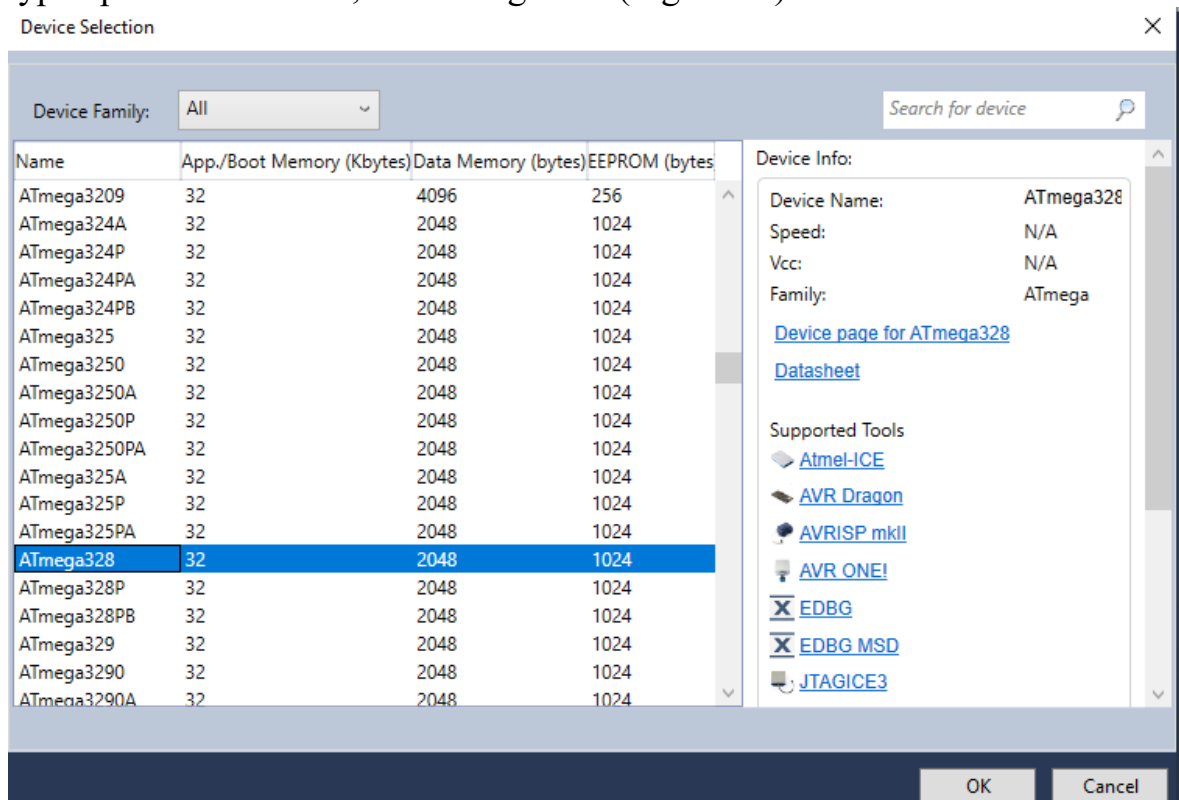
Drawing 14 - Creation new project

In the window that opens, you can select different types of projects. In the form of a script (Arduino Sketch), an executable program in C or C++ or a subroutine for a subroutine library (Figure 15).



Drawing 15 - Choice options project

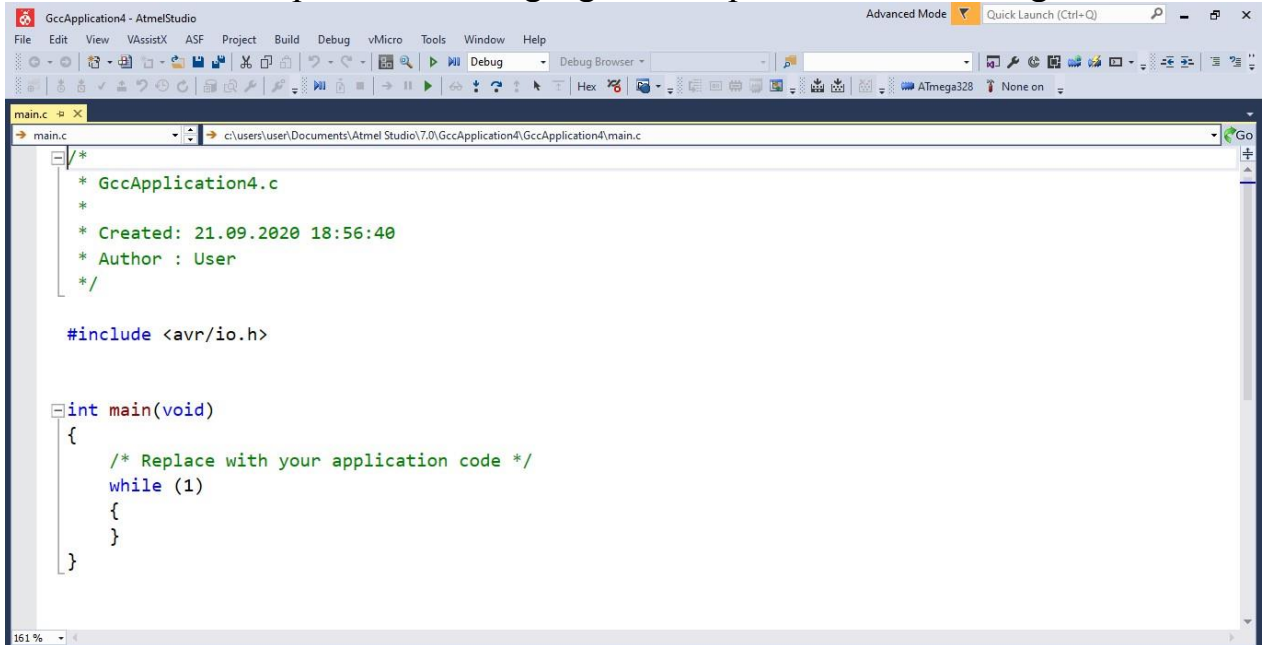
When performing laboratory work, you must select an executable file project in the C language . After selecting it, a window for selecting the microcontroller type opens. In our case, the Atmega 328 (Figure 16).



Drawing 16 - Choice microcontroller type

After selecting a microcontroller, the necessary libraries are added to the new project , in which the physical addresses of the register are replaced with their logical names, which are more understandable to humans.

Window template file on language WITH presented on drawing 17.



```

main.c
main.c
c:\users\User\Documents\Atmel Studio\7.0\GccApplication4\GccApplication4\main.c
/*
 * GccApplication4.c
 *
 * Created: 21.09.2020 18:56:40
 * Author : User
 */

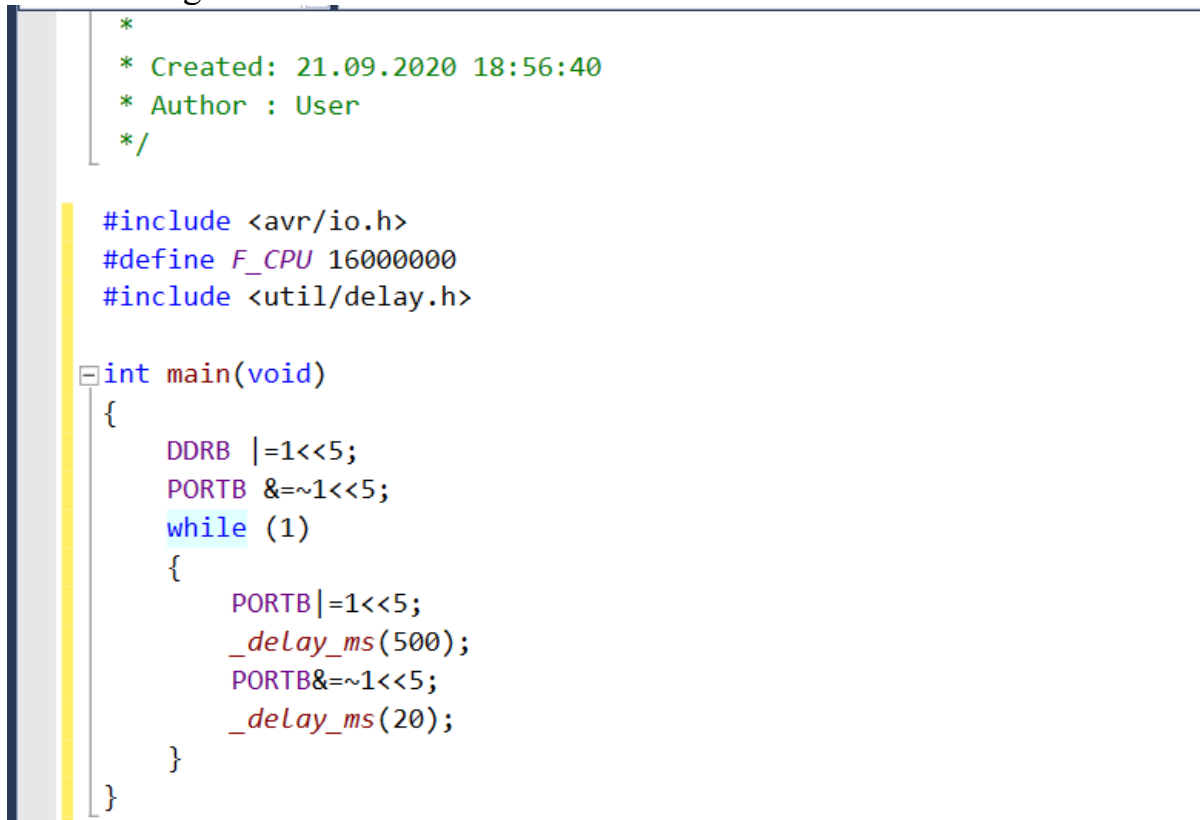
#include <avr/io.h>

int main(void)
{
    /* Replace with your application code */
    while (1)
    {
    }
}

```

Drawing 17 - Sample programs on WITH

Figure 6 shows the program for controlling the LED of the Arduino board. Uno , assigned to the fifth pin of Port B (PB5). Complete the program text as shown in Figure 18.



```

/*
 * Created: 21.09.2020 18:56:40
 * Author : User
 */

#include <avr/io.h>
#define F_CPU 16000000
#include <util/delay.h>

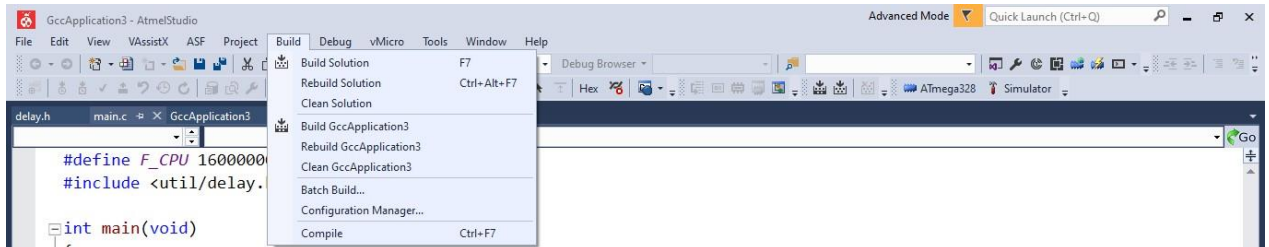
int main(void)
{
    DDRB |= 1<<5;
    PORTB &=~1<<5;
    while (1)
    {
        PORTB |= 1<<5;
        _delay_ms(500);
        PORTB &=~1<<5;
        _delay_ms(20);
    }
}

```

Drawing 18 - Program management blinking LED

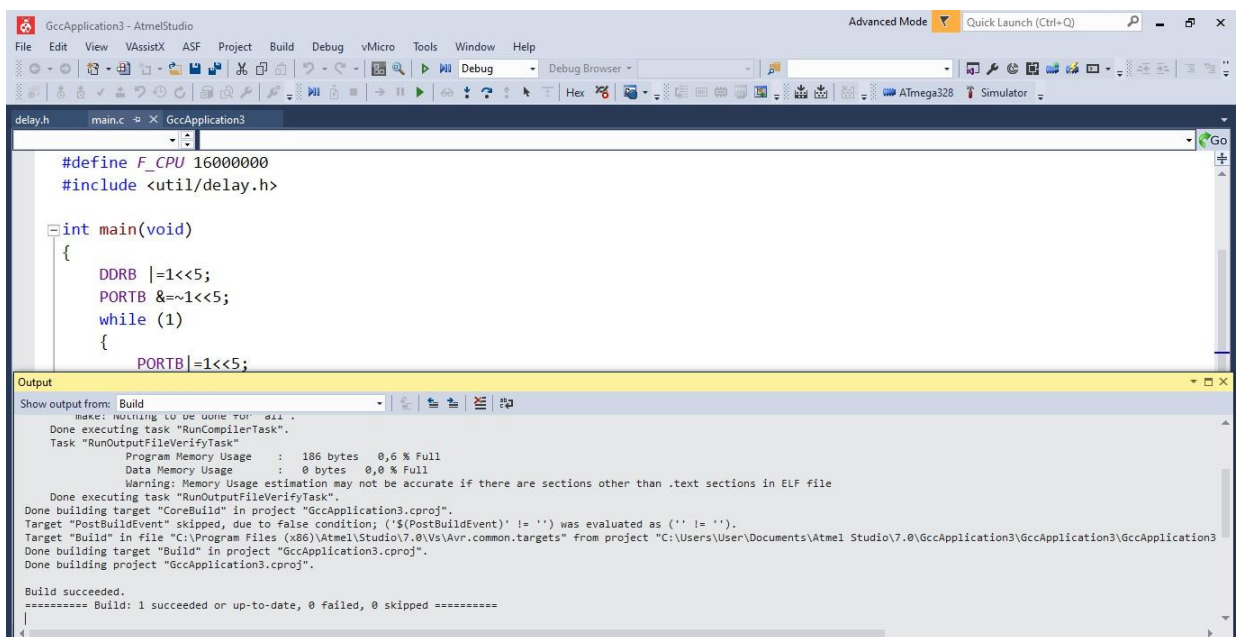
Find it in the folder where the integrated environment is located. Atmel Studio files `io.h` and `iom328p.h` and read their contents. Possible path to file placement `C:\Program Files (x86)\Atmel\AVR Tools\avr8-gnu-toolchain-win32_x86\avr\include\avr`

Compile the program (convert the program text to hexadecimal code). To do this, use the toolbar item Build (F7 button) in the Project menu (Figure 19).



Drawing 19 - Panel compiler

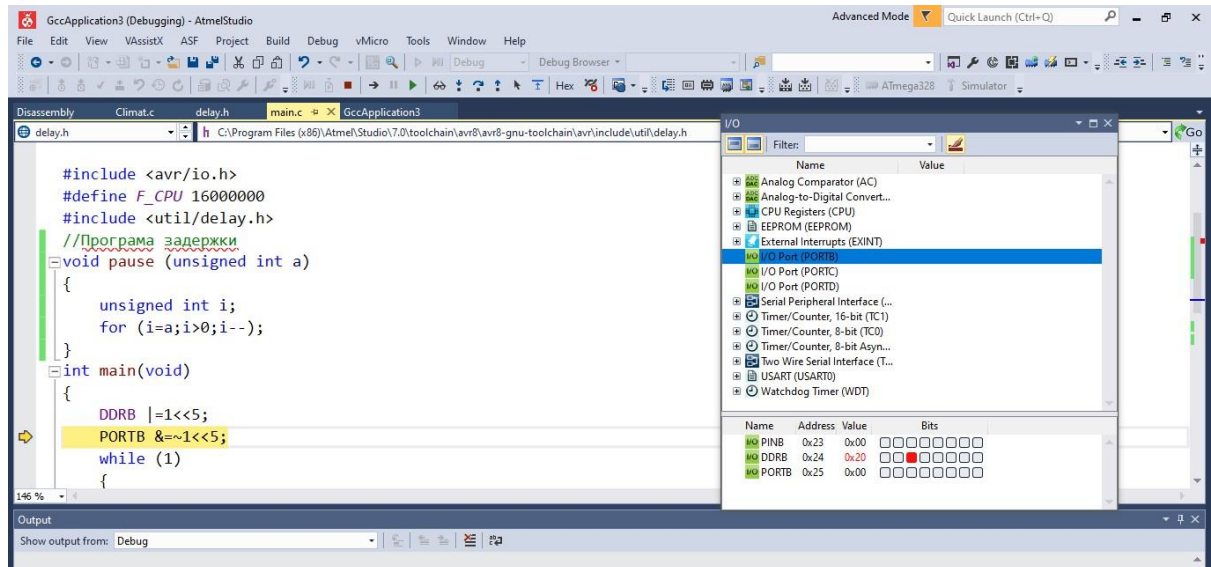
View window Output contains compiler messages. This window displays information about the number of code and data words, the presence of errors, and other information (Figure 20). Information is displayed on the percentage of program memory (the main program code) and data memory used .



Drawing 20 - Window messages compiler

To localize translation errors, if any, you can In the message box, place the mouse cursor on the error message and double - click click left button mice. When this V window editing initially-

The cursor will be positioned on the line that generated the error message , and this line will be highlighted. The translation results in an output file in the specified format. To launch the debugger, you must execute the Debug procedure , which is called by pressing the corresponding button (or F5+Ctrl) on the control panel. Atmel screen Studio in debug mode is shown in Figure 21.



Drawing 21 - Window programs V mode debugging

When executing the debugging procedure (or when loading an object file), the source code window of the program executed by the microcontroller is automatically opened . After executing the procedure, a yellow arrow appears indicating the position of the microcontroller's program counter (see Fig. 21). This pointer is always located on the line that will be executed in the next cycle. The user can execute the program completely in step-by-step mode, tracing blocks of functions, or executing the program up to the point where the cursor is located. In addition , an unlimited number of breakpoints can be defined, each of which can be enabled or turned off. Points stop are saved between sessions work. At Atmel Studio provides two step-by- step commands for debugging a program. mode: Step Over And Step Into . Difference between them V volume, What team Step Over does not work in subroutines. Using step mode commands, you can trace changes in values in I/O device registers, memory, and the register file. Step mode commands also include Auto Step and Multi Step. In addition to step mode, you can debug your program using breakpoints . The Go command The program execution starts . The program will run until stopped by the user or until a breakpoint is encountered.

Exercise For independent work

Develop and debug programs in accordance with individual assignments in accordance with Table 2.

Table 2- Options individual tasks

Parameters tasks	Number options				
	1	2	3	4	5
Input port	PD2	PD3	PD4	PD5	PD6
Output port	PB1	PB2	PB3	PB4	PB5
Installation condition port conclusion V 1	PD2=0	PD3=1	PD4=0	PD5=1	PD6=0
Parameters tasks	Number options				
	6	7	8	9	10
Input port	PB1	PB2	PB3	PB4	PB5
Output port	PC0	PC1	PC2	PC3	PC4
Installation condition port conclusion V 1	PB1=0	PB2=1	PB3=0	PB4=1	PB5=0

Order execution tasks

An example of project implementation in the environment Atmel Studio is considered for the option original data V which V quality port input is used PC1, as the output port of PB1, and the condition for setting the output port to 1 is that PC1 is equal to zero.

In accordance with the task, the assignment of the corresponding bits of ports PC1 to data input and port PB1 to output is determined. For this purpose, in the program, the first bits of the DDR direction registers of ports C and B are set to 0 and 1, respectively:

```
DDRC &=~1<<1;
```

```
DDRB |=1<<1;
```

We are installing initial meaning PC1=0:

```
PORTB &=~1<<1;
```

In the cyclic part of the algorithm we place the condition for setting the output port in accordance with the task.

```
while (1)
```

```
{
```

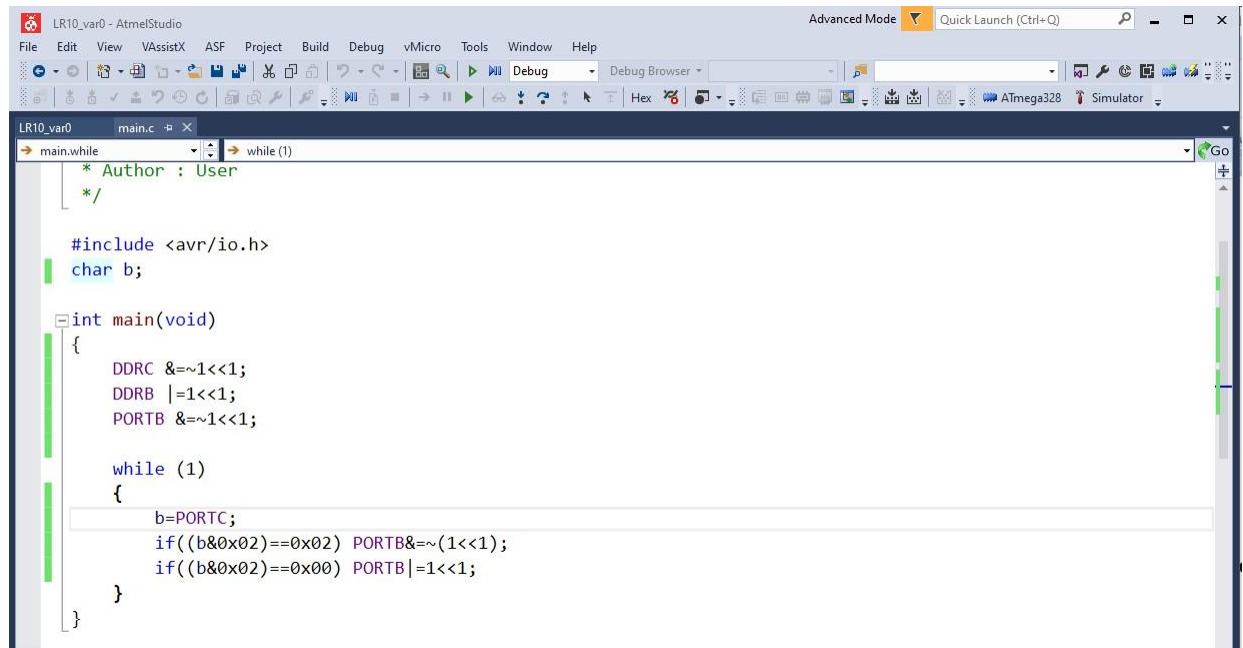
```
    b = PORTC ;
```

```
    if (( PORTC &0x 02)= =0x02) PORTB &=~(1<<1);
```

```
    if (( b &0x02)==0x00) PORTB |=1<<1;
```

```
}
```

Where variable type char b provides temporary storage contents of port C. Figure 22 shows the full text of the program.



```

LR10_var0 - AtmelStudio
File Edit View VAssistX ASF Project Build Debug vMicro Tools Window Help
Debug Debug Browser
LR10_var0 main.c
main.c while (1)
/* Author : User
*/

#include <avr/io.h>
char b;

int main(void)
{
    DDRC  &=~1<<1;
    DDRB  |=1<<1;
    PORTB &=~1<<1;

    while (1)
    {
        b=PORTC;
        if((b&0x02)==0x02) PORTB&=~(1<<1);
        if((b&0x02)==0x00) PORTB|=1<<1;
    }
}

```

Drawing 22 - Program management condition port

Requirements To design report

Report By laboratory work must contain:

- title sheet;
- theme, target And educational questions;
- original data By given option;
- results debugging programs V mode simulator;
- screenshots with software code And results of work present in the report.

Do conclusions O technologies programming With using the integrated programming environments Arduino IDE and Atmel Studio.

PRACTICAL LESSON No. 3

PROGRAMMING PROCEDURES BIT INPUT/OUTPUT

Target laboratory works:

Explore the capabilities of microcontrollers for processing bit information. Acquire skills in programming tasks with binary input/output signals in an environment Arduino IDE.

1. Theoretical intelligence And examples programs

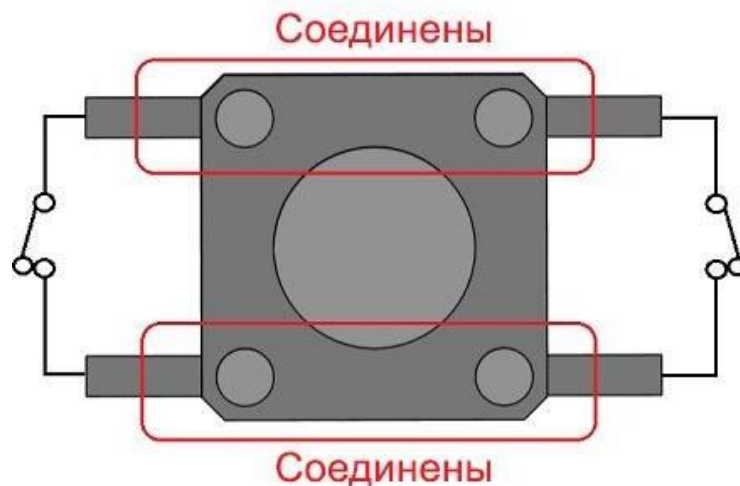
The minimum amount of binary information is one bit. Many control devices work with bit variables. For example, a simple security system can poll the status of window and door sensors, and if they are open, generate a signal to activate a sound and light alert device. To implement a control device such alarm in program microcontroller It is necessary to organize polling of individual port bits associated with binary data sources, and upon receipt of a trigger signal, a high-level signal must be sent to the port bits associated with the visual and audible alarm. To test the functionality of the software code, buttons can be used to simulate alarm sensor contacts, and LEDs can be used to simulate an alarm indicator.

A pushbutton is a simple device that can be used to control the flow of a program on a microcontroller by closing and opening a contact connected to a power source or common terminal (GND). Pushbuttons come in several types:

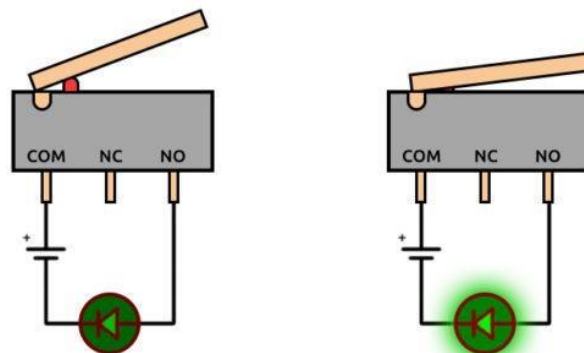
With locking - the button remains pressed after being released, without locking - it switches back off.

Normally open (Normal Open, NO) - at pressing closes contacts. Normally closed (Normal Closed (NC) – when pressed, opens the contacts.

Tact switches – close or open a contact. Conventional tact switches have their terminals connected lengthwise through the housing (Figure 1). Switches (Figure 2) – typically have three contacts, a common COM, and a normally open NO and normally closed NC. When the button is released, the COM-NC circuit is closed; when pressed, the COM-NO circuit is closed.



Drawing 1 - Contacts buttons from set laboratory equipment -
nia

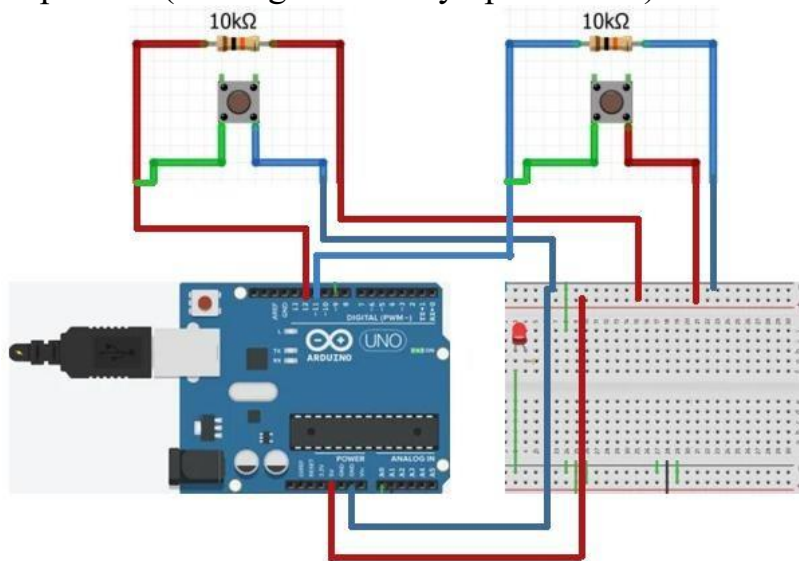


Drawing 2 - Usage switch For management LED -
hou
se

The level generated by the button must be binary: either a logical one (voltage close to 5 volts) or a logical zero (GND). To ensure unambiguous states at the input of the port bits, a pull-up is performed . up) pin to the power source or to ground (pull) down) using a resistor. The pull-up is performed opposite to the received signal , i.e. if it is necessary to receive a high signal level, the pull-up is performed to ground, if the active state signal logical zero - pull-up is performed to the power source.

On Figure 3 shows two options connecting the button, With pull-up to VCC and GND, respectively. The pull-up resistor value is typically selected in the range of 5-50 kOhm. If it's set higher, the pull-up may not provide a stable signal level on the port pin, and if it's set lower, energy will be lost due to resistor heating. For example, with a resistance of 1 kOhm, a current of $5 \text{ V} / 1000 \text{ Ohms} = 5 \text{ mA}$ will flow through it. For comparison, the board An Arduino with a microcontroller (MC) in active mode consumes 20-22 mA. A 10 kOhm resistor is most often used for pull-up. AVR microcontrollers (MC) have built-in resistors for all digital ports . input-output (GPIO), these resistors connected To nutrition (To VCC).

The connection of these resistors is carried out programmatically . Microcontrollers with other architectures may have a pull-up to GND, or may not have an internal pull-up at all . When using a pull-up to power, we get an inverted signal – the function `digitalRead ( )` will return 1 when the button is released, and 0 when it is pressed (if using a normally open button).



Drawing 3 - Connection buttons With using external resisto - ditch

Below is a program listing that uses an internal pull-up resistor to digital pin #3 (port D3). The results of the button request are displayed on the monitor.

```
void setup( ) { Serial.begin
(9600); pinMode (3,
INPUT_PULLUP);
}
void loop ( ) {
// will lead out 0, If button pressed
// and 1 if No Serial.println (
digitalRead ( 3)); delay(10);
}
```

In most real-world applications, work with the current state of the button Very it's inconvenient, For example, When action should be completed - but Once the button is pressed, the button will be triggered. Let's complicate the design a bit by adding a single flag that will remember the button's state. This design allows us to track the button's press and release and respond to them once:

```
void setup( ) { Serial.begin
(9600); pinMode (3,
INPUT_PULLUP);
```

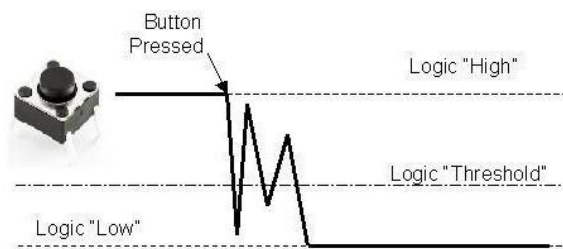
```

}
bool flag = false;
void loop( ) {
// we read inverted meaning For bool amenities
tnState = ! digitalRead (3);
if ( btnState & & !flag ) { // handler clicks flag =
true;
Serial.println ("press");
}
if (! btnState && flag) { // handler release flag =
false;
// Serial.println (" release ");
}
}
}

```

The button isn't perfect, and the contact doesn't close immediately; it "bounces" for a while. In the program presented above, the button is polled for approximately 6 μ s , meaning the button is polled 166,666 times per second . This is enough to generate several thousand false alarms (Figure 4). The minimum contact bounce duration is 50 ms .

Button "Bounce"



Drawing 4 - Rattling contact at pressing buttons

Contact bounce can be eliminated using both hardware and software : the hardware solution is an RC circuit, that is, a resistor (1-10k Ω) and a capacitor (approximately 100nF). The hardware bounce protection circuit is shown in Figure 5.

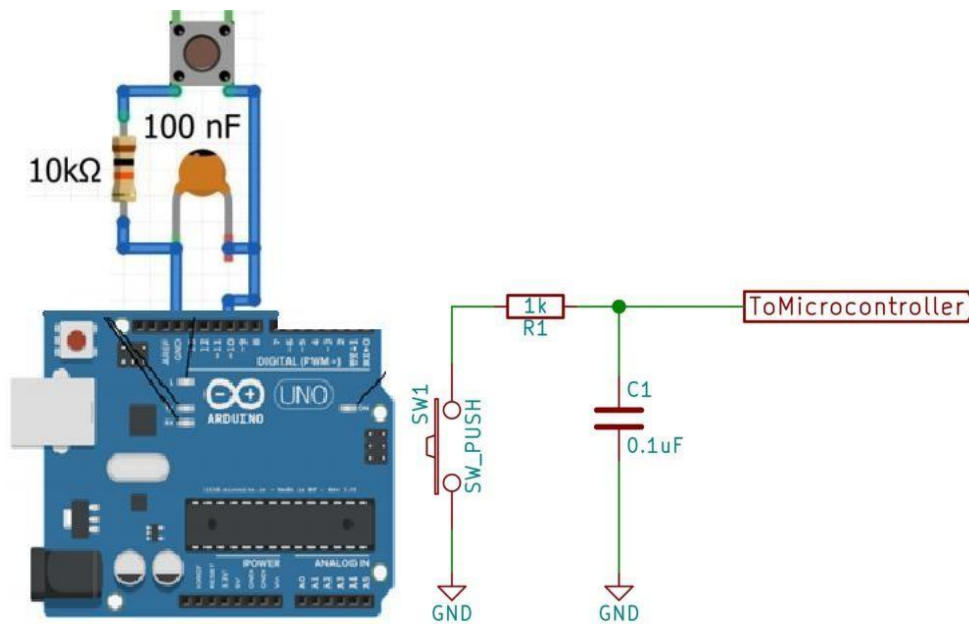


Figure 5 - Anti-bounce protection using RC filter Software Can enter

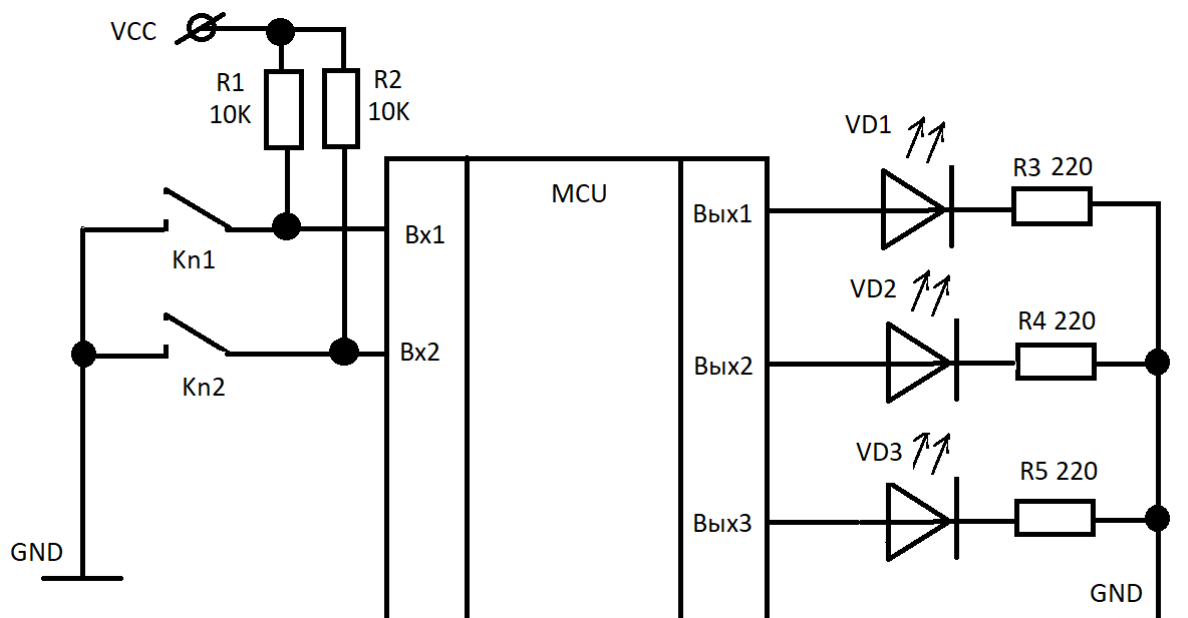
the simplest timer pressing, founded on Using the built-in millis () function , we'll set the debounce time to 100 milliseconds . An example program listing with software debounce protection is presented below:

```
void setup( ) { Serial.begin
(9600); pinMode (3,
INPUT_PULLUP);
}
bool flag = false;
uint32_t btnTimer = 0;
void loop( ) {
// we read inverted meaning For bool amenities
tnState = ! digitalRead (3);
if (btnState && !flag && millis() - btnTimer > 100) {
flag = true;
btnTimer = millis();
Serial.println("press");
}
if (!btnState && flag && millis() - btnTimer > 100) {
flag = false;
btnTimer = millis();
//Serial.println("release");
}
}
```

Application hardware method preferable, So How Not loads the processor with solving additional problems.

2. Tasks For independent work

Using the LED blinking sketch example and the program listings provided above, develop a program to poll two input signals and three output (control) signals. The electrical circuit diagram of the device is shown in Figure 6.



Drawing 6- Electric scheme devices

options for the initial data, specifying the ports used . Perform the electrical circuit wiring using a circuit board. The choice of debounce method is up to you.

Table 1 - Options original data For independent work

Number options	Bx1	Bx2	Exit 1	Exit 2	Exit 3	Conditions form-worlding
1	PD2	PB5	PD7	PD3	PB4	Out1=In1; Out2=In2; Out3= Bx2.
2	PD3	PB4	PB0	PD4	PB3	
3	PD4	PB3	PC5	PD5	PB2	
4	PD5	PB2	PC4	PD6	PB1	
5	PD6	PB1	PC3	PD7	PD2	
6	PD7	PD2	PB5	PB0	PD3	
7	PB0	PD3	PB4	PC5	PD4	
8	PC5	PD4	PB3	PC4	PD5	
9	PC4	PD5	PB2	PC3	PD6	
10	PC3	PD6	PB1	PB5	PD7	

3. Design And protection report

Report must contain:

title sheet, topic And target laboratory work;
screenshots or listings programs, presented V quality note-

ditc

h; electrical principled diagram With specification input And
weekend ports;

listing of the program that provides the specified operating algorithm; screen
copies And photos, confirming performance
programs.

PRACTICAL LESSON #4

STUDY FEATURES PROGRAMMING TIMERS/COUNTERS

Goals works:

- To study the programming of delay functions, measuring time intervals , generating a sequence of rectangular pulses of a given frequency and duty cycle using Arduino platforms Uno and the Arduino IDE
- Learn programming of built-in timers/counters of a microcontroller ATmega 328 in the Atmel tool environment Studio .
- Improve your skills in the Arduino IDE and Atmel development environments Studio .

Educational questions:

- 1 Creating programs using built-in timers in Atmel environment Studio delays using the library Arduino
- 3 Measurement temporary intervals
- 4 Generating a pulse sequence with a given period and duty cycle

1 Creating programs using the built-in timers in the Atmel environment Studio

ATmega microcontrollers have two 8-pin and one 16-bit timer/counter. As its name suggests, it counts either time intervals (timer) or input events (counter). The timer /counter has a TCNT register from which you can read how much time has passed since the timer was started. The value in this register is not in minutes or seconds , but in timer 'ticks'. The value of one tick depends on the clock frequency at which the microcontroller operates and the timer settings .

The timer/counter also has a configurable frequency divider, which determines how much the microcontroller's clock frequency will be divided before it is fed to the timer/counter. The duration of the timer tick is the reciprocal of the frequency obtained as a result of division.

For example :

Clockwise frequency = 16,000,000 Hz (16 MHz)

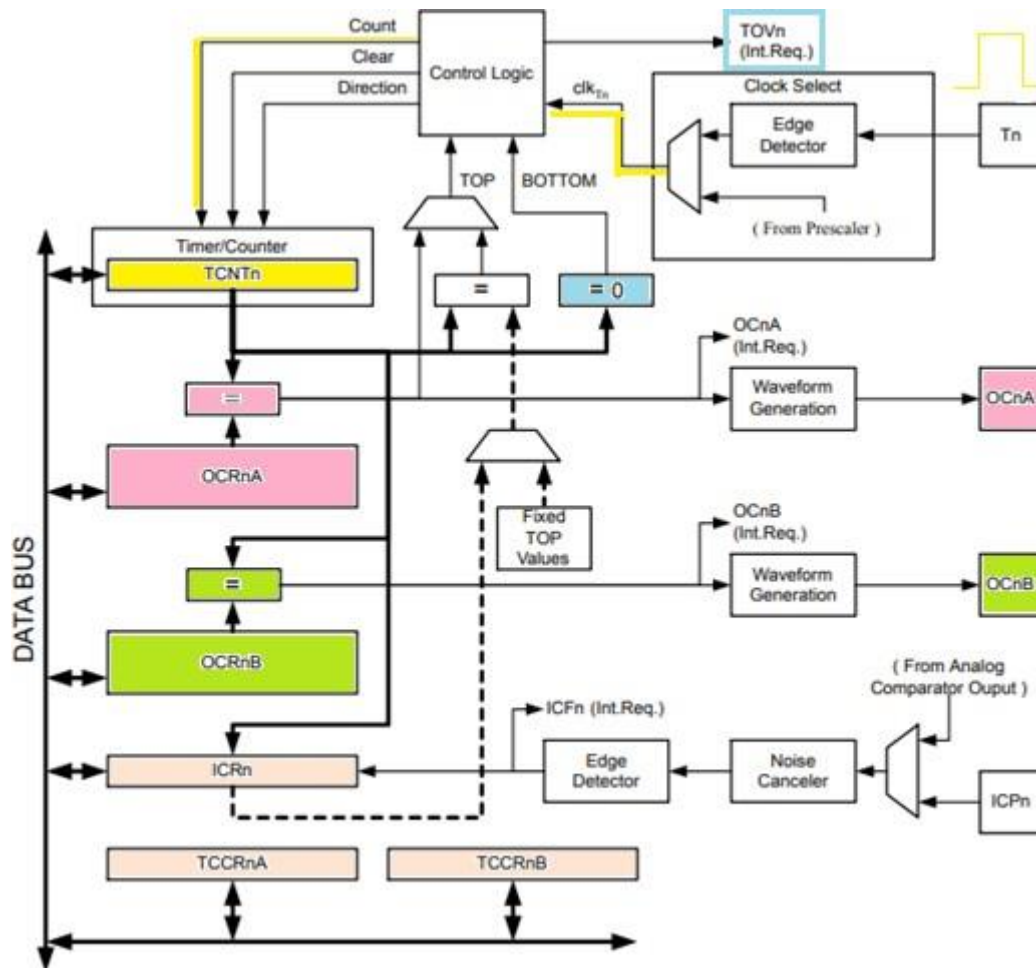
Divisor = 1024

Timer/Counter

Frequency = Clock

Frequency/Divide = 16000000/1024 = 15625 Hz

Duration of the period (tick) = $1/15625 = 64 \cdot 10^{-6}$ seconds = 64 μ s
 Programming timers/counters let's consider on example 16
 discharge timer T1, the structural diagram of which is shown in Figure 1.



Drawing 1

For settings divider are used CS bits register TCCR.

For example, for timer 1 in ATmega328, the TCCR1B register and the CS10, CS11, CS12 bits are used.

Bits CS12 (2), CS11 (1), CS10 (0) **TCCR1B** register set the clocking mode and clock prescaler of the timer/counter T1 :

- 000 - timer/counter T1 stopped
- 001 - clock generator CLK
- 010 - CLK/8
- 011 - CLK/64
- 100 - CLK/256
- 101 - CLK/1024
- 110 - external source on conclusion T1 (11 leg) By decline signal

- 111 - external source on conclusion T1 (11 leg) By increase signal

Example code settings divider For timer 1 ATmega328:

```
TCCR1B = (1<<CS12)|(0<<CS11)|(1<<CS10); //clk/1024
```

The ATmega328 timers can be 8- or 16-bit. The bit depth determines the maximum number of ticks the timer can count. For an 8-bit timer, this is 256 ticks; for a 16-bit timer, this is 65,536.

In our case, one tick is equal to 64 μ s, so the maximum value is, which Maybe measure 16 bit timer, This $65536 * 64 * 10^{-6} = 4.194$ seconds, 8 bit — 0.01632 seconds.

After the timer reaches its maximum value, it overflows, i.e., starts counting from zero. This situation can be handled using interrupts. To do this, enable the timer overflow interrupt and set the general interrupt enable bit.

Example code For permissions interruptions timer 1 ATmega328:

```
TIMSK |= (1<<TOIE1); //разрешить прерывание по переполнению таймера счетчика
sei(); // выставить бит общего разрешения прерываний
```

So same necessary define handler this interruptions — a function that will be called when the timer overflows.

Let's add V this one function code changing state legs PB0, to which the LED is connected.

Example code functions handler interruptions For timer 1 ATmega328:

```
ISR( TIMER1_OVF_vect )
{
    if( PINB & ( 1<<PB0 ) ) {
        PORTB &= ~( 1<<PB0 );
    }
    else {
        PORTB |= ( 1<<PB0 );
    }
}
```

The LED will change its state every 4 seconds (frequency 0.25 Hz).

a period of 0.5 s), the timer needs to count down less than 65,536 ticks. Let's calculate the required number of ticks:

$$0.5 \text{ s} / 64 * 10^{-6} = 7812.$$

not from zero, but from a certain value, during initialization and at each interrupt triggering In the TCNT1 register we will write the number $(65536 - 7812) = 57724$.

Listing final programs For timer 1 ATmega328 :

```
#include <avr/io.h>;
#include <avr/interrupt.h>;

ISR( TIMER1_OVF_vect )
{
    TCNT1 = 57724; //выставляем начальное значение TCNT1
    if( PINB & ( 1 << PB0 ) ) {
        PORTB &= ~( 1 << PB0 );
    }
    else {
        PORTB |= ( 1 << PB0 );
    }
}

int main()
{
    DDRB = ( 1 << PB0 ); // настраиваем PBO на выход
    TCCR1B = (1<<CS12)|(0<<CS11)|(1<<CS10); // настраиваем делитель
    TIMSK1 |= (1<<TOIE1); // разрешаем прерывание по переполнению таймера
    TCNT1 = 57724; // выставляем начальное значение TCNT1
    sei(); // выставляем бит общего разрешения прерываний
    while(1); // вечный цикл
    return 0;
}
```

Change the LED blinking frequency according to the initial data option (Table

1)

Number options	1	2	3	4	5	6	7	8	9	10
Frequency , Hz	1	2	3	4	5	6	7	8	9	10

Do it firmware programs With help programmer USBasp .

2 Programming temporary delays With using the library Arduino

IN library Arduino.h two built-in functions delays.

delay (time) function "pauses" code execution for time milliseconds. The argument type is unsigned. long allows you to pause the execution of the main program for a period from 1 ms to ~50 days (4,294,967,295 milliseconds) with a resolution of 1 millisecond.

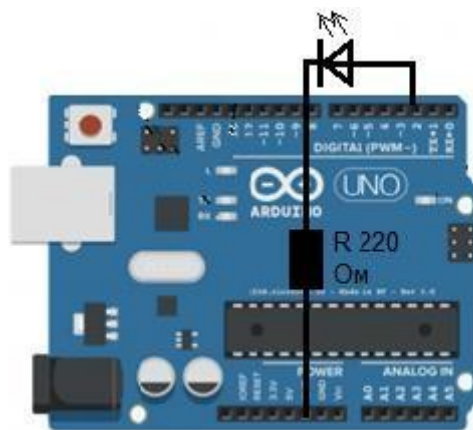
delayMicroseconds (time) function is similar to the delay () function, but pauses code execution for time , specified in microseconds . The time argument has unsigned data type int and can delay the execution of the main program for a period of 4 to 16383 μ s in 4 μ s increments.

Let's consider programming time delays in the Arduino IDE using the example of a control program LED attached to pin 2 of the board connector.

Listing 1.

```
const int ledPin = 2;
void setup() { pinMode(ledPin, OUTPUT); }
void loop() {
  digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(500);                // wait for a 0,5 second
  digitalWrite(ledPin, LOW);  // turn the LED off by making the voltage LOW
  delay(500);                // wait for a 0,5 second
}
```

Connect the LED as shown in Figure 2. Type the program text in the Arduino IDE text editor . Check the program for errors using the Sketch - Test/Compile tab. If there are no errors, connect the Arduino board. Uno to a personal computer using a USB A-USBB cable. Load the program by pressing the button in the form of a horizontal arrow or by selecting Sketch-Load (key combination) Ctrl /U). When loading the program, blinking LEDs are observed. Rx , Tx . After loading, the program will start and the LED connected to pin 2 of the board as shown in Figure 2 will begin to change its state with a frequency of 0.5 s.



Drawing 2 - Connection LEDs

3 Measurement temporary intervals

The tool for measuring time intervals uses functions that count the time from the start of the microcontroller (power on). There are no such functions in the library Arduino.h two:

millis ()

– Returns the number of milliseconds elapsed since startup. Returns a variable of type unsigned long , 1 to 4,294,967,295 milliseconds (~50 days), has Resolution 1 millisecond, reset to 0 after overflow . Works on the system timer Timer 0

micros ()

– Returns the number of microseconds elapsed since startup. Returns variable of unsigned type long , from 4 to 4,294,967,295 microseconds (~70 minutes), has Resolution is 4 microseconds, after overflow it is reset to 0. Works on the system timer Timer 0

To measure time intervals using the built-in functions millis () and micros (), it is necessary to provide in the program the storage of the time accumulated from the moment the microcontroller is turned on, in a specially allocated variable at the moment the measured interval begins. And at the end of the measured interval, subtract the value of the saved start time from the current time value millis () and micros ().

In the example below, **the millis () function** causes the LED to blink as in the previous example.

Listing 2.

```
const int ledPin = 2;
unsigned long next_time ; // time another one switching
                        // the LED
int timeout = 500; // half the period blinks
int led_state = 0; // initial state LEDs - void is disabled setup ( ) {
    pinMode ( ledPin , OUTPUT);
    digitalWrite ( ledPin , led_state ); // we extinguish LED
    next_time = millis ( ) + timeout ; // we calculate time next
                                // switching
}
void loop( ) {
    int now_time = millis ( ); // current time
    if ( now_time >= next_time ){ // If current time exceeded
                                // planned time, That
        next_time = now_time + timeout; // we calculate time next
```

```
// switching
led_state = ! led_state ; // change state on opposite digitalWrite ( ledPin ,
led_state ); } } // let's light it up or we extinguish LED
```

Do it input programs, her debugging And compilation. At If there are no errors, download the program V memory microcontroller. After When the download is complete, watch the LED blink.

4 Generating a pulse sequence with a given period and duty cycle

To control digital devices, it is necessary to be able to generate a sequence of rectangular pulses of a given duration and repetition rate. Figure 3 shows an example of such a sequence . Amplitude impulses is determined level logical units and is for the ATmega 328 microcontroller 5B.

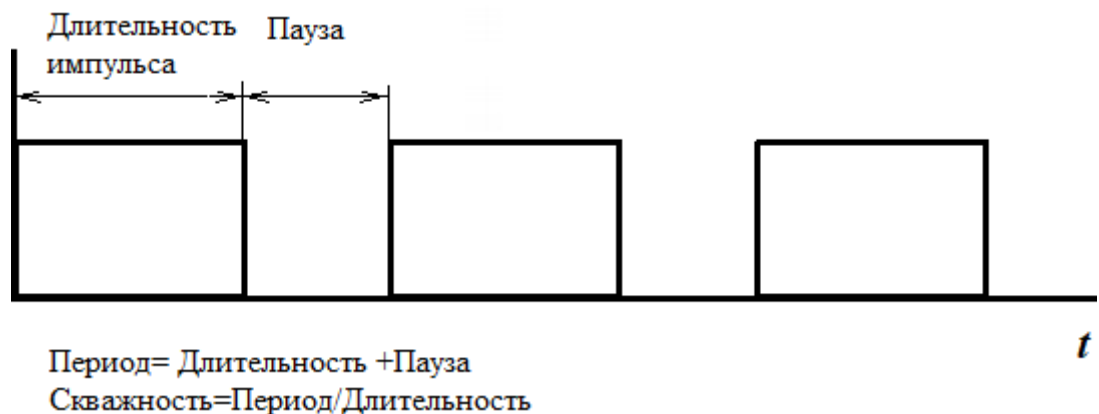


Figure 3 – Sequence of rectangular pulses Implementation this sequences

Can execute on base listing 1 by setting the delay for the output state set to one equal to *the pulse duration*, and the delay for the pause is formed by the delay states exit at zero meaning (LOW) equal The Pause value . The disadvantage of this approach is that the program will not be able to perform any other actions, otherwise the duration and pause times will change randomly.

A more preferable option is the one based on the `millis ( )` function, at his use V interval between operators checks conditions `if ( now_time >= next_time_1 )` Can fulfill any other actions .

Listing of the program in which the duration the pulse duration is 500 milliseconds, and the pause duration is — 150 ms is presented below.

Listing 3

```

const int Pin_Out = 3;
unsigned long next_time_1; // current time duration pulse unsigned long
next_time_2; // current time pauses
int timeout_1 = 500; // duration pulse int
timeout_2 = 150; // pause duration
int out_state = 1; // initial state output: void is enabled setup ( ) {
    pinMode ( Pin_Out , OUTPUT);
    digitalWrite ( Pin_Out , out_state ); // we put exit V zero state next_time_1 = millis
    ( ) + timeout_1; // calculate the time of the next
                                // set the output to 1 next_time_2 =
    millis ( ) + timeout_2; // time shutdown pauses
}
void loop( ) {
    int now_time = millis ( ); // current time if(
out_state ==1) {
    if( now_time >= next_time_1 ){ // If current time exceeded
                                // planned time , That
        out_state = 0; // change state on off digitalWrite (
        Pin_Out , out_state );
        next_time_2= now_time + timeout_2; // we calculate time end pauses
    }
}
if( out_state ==0) {
    if( now_time >= next_time_2 ){ // If current time exceeded
                                // planned time completion pauses , That
        out_state = 1; // change state on off digitalWrite ( Pin_Out , out_state );
        next_time_1= now_time + timeout_1; // we calculate time end impulse
    }
}
// Here can be any other operators
}

```

Loading program, turn on nutrition. Now LED blinking with different on and off intervals

Exercise For independent work

Run all example programs. Make changes to the program listings in accordance with the instructor's instructions. Provide screenshots of the program code and work results in the report. Draw conclusions about programming technology using the Arduino IDE.

PRACTICAL LESSON No. 5

STUDY FEATURES PROCESSING AND FORMATION OF ANALOG SIGNALS USING MICROCONTROLLER

Goals laboratory works:

1. To study the features of programming analog-to-digital converters and analog comparators using **Arduino** platforms **Uno** and the **Arduino IDE** tool environment
2. Acquire skills in working with analog signals in the **Arduino IDE** and **Atmel** instrumentation environments **Studio** .
3. programming PWM signals using **Arduino** platforms **Uno** and the **Arduino development environment IDE**
4. Acquire skills in working with PWM signals in the **Arduino IDE** and **Atmel** instrumentation environments **Studio** .

Educational questions:

1. Development programs With using analog-to-digital converter
2. Setting up the Atmega328 microcontroller's analog comparator

1 Development programs With using analog-to-digital preobraz - educator

ADC programming in scenario mode is described in detail on the website <http://arduino.ru/Reference/Analogreference>

Main library functions:

`analogRead ( pin )` -- The function reads the value from the specified analog input . Returns: int (0 to 1023).

Reading the value from the analog input takes approximately 100 microseconds (0.0001 sec), i.e. the maximum frequency readings approximately 10,000 times per second.

`analogReference ( type )` -- defines the reference voltage relative to which analog measurements are made. Type takes one of the following values:

DEFAULT: standard reference voltage is 5V (on platforms with 5V supply voltage) or 3.3V (on platforms with 3.3V supply voltage).

INTERNAL: Built-in reference voltage of 1.1 V on ATmega168 and ATmega328 microcontrollers, and 2.56 V on ATmega8.

EXTERNAL: external reference voltage source connected to the AREF pin (it is recommended to connect to the AREF pin via a 5 kOhm resistor).

The reference voltage can be set using the `analogReference` function or using the `REFT[1:0]` bits in the `ADMUX` register.

Use the internal 1.1V reference voltage for accurate measurements of external voltages:

```
analogReference ( INTERNAL1V1); // select the internal reference voltage of 1.1V
```

The 1.1 V reference voltage is more stable and unaffected by changes in supply voltage or temperature. This allows for absolute value measurements.

Modes work

1) A single sample is actually what the Arduino does when calling the `analogRead ( )` function.

```
void setup( ) {  
    Serial.begin (9600);  
}  
void loop( ) {  
    int val = analogRead (A0);  
    Serial.println ( val ); delay(  
        1000);  
}
```

Because of noises 2 younger but usually discard: `int val = analogRead (A0)>>2;`

V in the end we get values from 0 to 255, those. accuracy $5/255=19.6\text{mV}$.

2) Continuous sample:

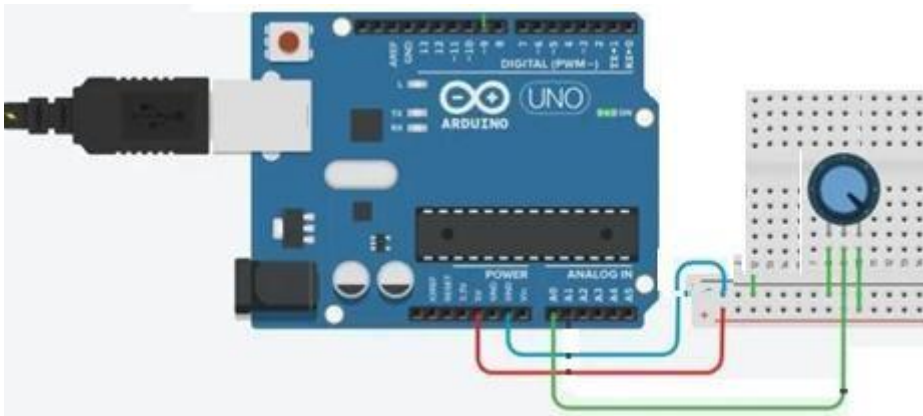
A good idea when continuously sampling a signal is to use interrupts .

Microcontrollers ATmega328 and ATmega2560 can be transferred to continuous sampling mode (free running mode). In this mode, the ADC starts automatically after the previous processing is completed.

To enable continuous sampling mode, three registers must be set: `ADMUX`, `ADCSRA` and `ADCSRB`.

Connecting a potentiometer to the ADC inputs. Potentiometers act as adjustable voltage dividers. They come in various sizes and shapes, but always have three terminals.

Connect one end of the potentiometer to ground and the other to the 5V rail. Potentiometers are symmetrical, so it doesn't matter which side you connect to the power rail and which to ground. Connect the middle end to one of the analog pins on the Arduino board, as shown in Figure 1 . When you turn the potentiometer knob, the analog input signal will smoothly change from 0 to 5 V.



Drawing 1 - Connection potentiometer to the ADC

To read the signal from the sensor into the program, we need **the analogRead () function** . It takes the port number as an argument . The pins that can be used for analog input are labeled ANALOG IN on the **Arduino** board .

The voltage applied to the analog input, typically 0 to 5 volts, will be converted to a value between 0 and 1023, with a quantization step (resolution) of 0.0049 volts. The voltage spread and step can be changed using the **analogReference ()** function .

Reading a value from an analog input takes about 100 microseconds (0.0001 sec), which means the maximum reading frequency is approximately 10,000 times per second.

the potentiometer output into binary code can be implemented based on those already available in the library. Arduino IDE examples. For example, let's open the **AnalogInOutSerial program**. From the **File menu** , **select Examples and select Analog** . Add a constant and variable to connect the potentiometer and the value it receives .

```
const int analogInPin1 = A0;  
int sensorValue1 = 0;  
float outputValue = 0;
```

IN functions **setup ()** let's open consistent port. A V functions **loop ()** let's add reading signal With connected potentiometer.

```
void loop( ) {
  // read the analog in value:
  sensorValue1 = analogRead (analogInPin1);
```

When we use data from analog sensors and output voltage to a pin depending on these sensors, we need to take care of converting numbers from one range to another.

The data from the potentiometer will come in the form of decimal values from 0 to 1023, to convert them into the range from 0 to 255 you can use **map ()** function .

It proportionally transfers the value from the current range of values to the new range. For example ,

```
outputValue = map ( ( sensorValue1, 0, 1023, 0, 255);
```

In this case, the value of c connected potentiometer is generated by the ADC in the range from 0 to 1023. The function converts the result into new range from 0 to 255.

You can also use the function **constrain ()** . It takes three parameters and outputs the bounds of a range if the value being tested is outside the bounds of that range. For example,

```
sensVal = constrain(0, 5, 150); // 5
sensVal = constrain(200, 5, 150); // 150
sensVal = constrain(95, 5, 150); // 95
// we limit values sensVal range from 5 up to 150
```

Full text programs

```
int analogInPin1 = A0; // Analog input pin
int sensorValue1 = 0;      // value read from the pot
float outputValue = 0;
void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}
void loop( ) { // read the analog in value:
  sensorValue1 = analogRead (analogInPin1);
  //transformation result V voltage coefficient 0.0048
  //получен делением 5/1024
  outputValue=0.0048*sensorValue1;
  // print the results to the Serial Monitor:
  Serial.print("sensor1 = ");
  Serial.print(sensorValue1);
  Serial.print("\t output = ");
  Serial.println(outputValue);
  // wait 0,5 seconds before the next loop for the analog-to-digital
```

```

        // converter to settle after the last reading:
delay(500);
}

```

When programming in the environment Atmel Studio The registers are configured directly in the user program. Below is an example of a program for measuring voltage and outputting the result to an LED indicator in the form of a binary code.

```

/** Usage ADC. LED scale */

#include < avr / io.h >
#include < util / delay.h >

int main (void)
{
    DDRD = 0xFF;
    PORTD = 0x00;

    /** Setting up ADC */
    ADCSRA |= (1 << ADEN ) // Turning on ADC
               |(1 << ADPS1)|(1 << ADPS0); // prescaler converter on 8
    REFS1)|(0 << REFS0) // external ION
               |(0 << MUX0)|(0 << MUX1)|(0 << MUX2)|(0 << MUX3); // вход PC0

    while(1)
    {
        unsigned int u;
        ADCSRA |= (1 << ADSC ); // Let's begin transformation
        while ((ADCSRA & (1 << ADIF)) == 0); // Wait for the end flag
        // transformations
        u = (ADCL | ADCH << 8); // Let's read ADC

        if (u > 128) // 0.625V
            PORTD = 0b00000001;
        else
            PORTD = 0b00000000;

        if (u > 256) // 1.25V
            PORTD = 0b00000011;
        if (u > 384) // 1.875V

```

```

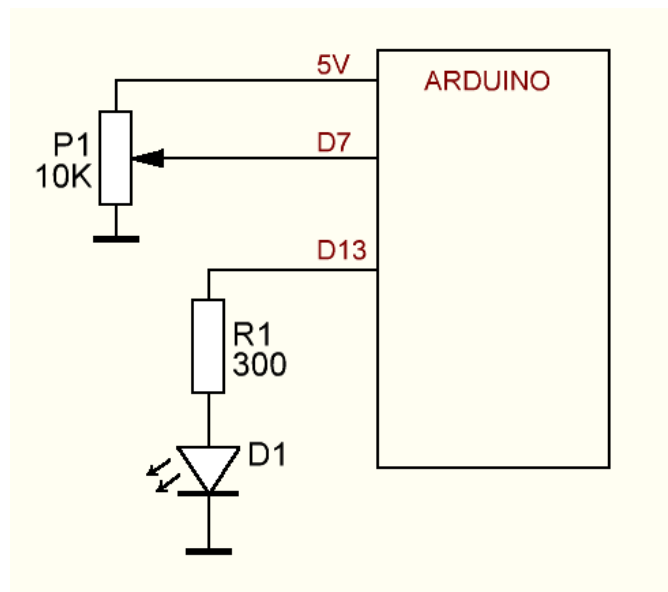
    PORTD = 0b00000111;
    if (u > 512) // 2.5V
        PORTD = 0b00001111;
    if (u > 640) // 3.125V
        PORTD = 0b00011111;
    if (u > 768) // 3.75V
        PORTD = 0b00111111;
    if (u > 896) // 4.375V
        PORTD = 0b01111111;
    if (u > 1020) // 5V
        PORTD = 0b11111111;
    _delay_ms(30);
}
}

```

2 Setting up analog comparator microcontroller Atmega328

An analog comparator is designed to compare two voltages supplied to the corresponding inputs. On the debug board Arduino The Uno uses inputs AIN0 and AIN1 (D6 and D7) as inputs of the analog comparator . An external or internal reference voltage source is connected to input AIN0 , and a voltage is applied to input AIN1 (inverse input) for comparison with the voltage AIN0 (direct input).

Work comparator let's consider on next example (drawing 2).



Drawing 2 - Example connections external signal

```

void setup ( ) {
    ACSR |= (1 << ACBG); // let's connect interior ION ko entrance AIN0 D6)
    DDRB |= (1 << 5);    // output PB5 as output (D13 }
}

```

```

}
void loop ( ) {
  while ((ACSR & 0x20) == 0x20) { // If bit ACO register ACSR equal 1, then
    execute the code in a loop
    PORTB |= (1 << 5); // PB5 HIGH
  }
  PORTB &= ~(1 << 5); // PB5 LOW
}

```

Another example software implementation comparator

// So How comparator program need to ask some accuracy limit

```
int delta = 5;
```

```

void setup( ){
  pinMode ( 1, OUTPUT);
  pinMode(2, OUTPUT);
}
void loop(){
  int u1 = analogRead(A1);
  int u2 = analogRead(A0);
  if (u1-delta > u2)
    {digitalWrite(1, HIGH);
     digitalWrite(2, LOW);
    }
  else if(u2-delta > u1)
    { digitalWrite ( 1, HIGH);
      digitalWrite (2, LOW);
    }else
    { digitalWrite ( 1, LOW);
      digitalWrite (2, LOW);
    }
}

```

Assemble the circuit taking into account the connection of two potentiometers to analog inputs A0 and A1, as well as two LEDs to digital outputs 1 and 2.

3. Development programs formations PWM signals

Pulse-width modulation (PWM) or in English Pulse - Width Modulation (PWM) is the control of the effective voltage on the load by changing the duty cycle (duration) of pulses at a constant frequency and signal amplitude.

To implement PWM, built-in timers/counters T0, T1, T2 are used. Their outputs, which can be used to generate a PWM signal, are marked on the Arduino board with a special symbol ~. For each timer/counter, working in PWM mode provided two output - OCnA for generating a non-inverse PWM signal and OCnB for generating an inverse PWM signal (see lecture 9). On the Arduino board uno, these are contacts 6 and 5 for T0; 9 and 10 for T1; 11 and 3 for T2.

Do it example work With PWM from libraries Arduino IDE: File -

Examples - 01.Basic - Fade

Analyze appointment variables And order work program -

We.

Before using necessary define corresponding contact How day off. IN environment Arduino IDE For this is used function pinMode (Num ,OUTPUT); where Num output 6,5,9,10,11,3.

Actually conclusion signal With current voltage V interval 0-5 V is carried out by command

analogWrite (Num, Aout); Where Out =[0- 255]

Example 1. Program for cyclic linear increase and decrease of the effective voltage at the PWM output.

```
int led = 10;      // the PWM pin the LED is attached to
int Aout = 0;      // Analog signal
void setup() {
  // declare pin 10 to be an output:
  pinMode(led, OUTPUT);
}
void loop() {
  int i;
  for(i=0;i<255;i++){
    Aout=i;
    analogWrite(led, Aout);
    delay(30);
  }
  for( i = 255;i >0;i--) {
    Aout = i ; analogWrite (
    led,Aout ); delay(30);
  }
}
```

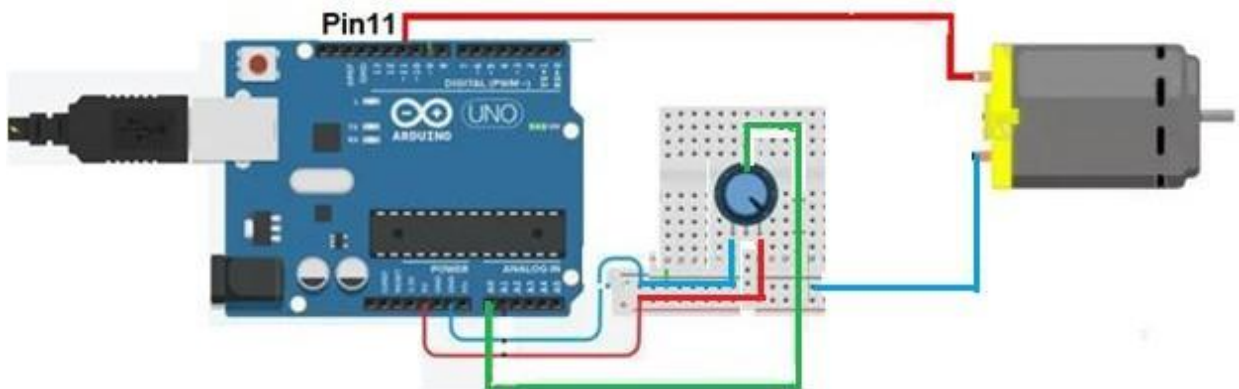
Run the example. To check the program's operation, output ~10 Connect the LED anode. The LED cathode should be connected to the common terminal (-5 V).

4. Control speed rotation electric motor With using pulse - width modulation

Collector An electric motor can be designed for different supply voltages. If the motor operates at 3-5 volts, its windings can be connected directly to the Arduino board . Motors for RC toy cars, which come complete with gearboxes and wheels, are designed for 6 volts or more, so they should be controlled via a field-effect transistor (bipolar transistor) or an L298N driver.

On drawing 3 shown connection electric motor directly.

To set the required rotation speed, we will use a potentiometer and an ADC. We will connect the potentiometer to A0.



Drawing 3 - Connection electric motor permanent current, operating at 3-5 V

```
int analogInPin = A 0; // Analog input pin
int analogOutPin = 11; // Analog output pin with PWM
int sensorValue = 0; // value read from the pot
int Aout = 0; // value output to the PWM (analog out)
void setup() {
  Serial.begin(9600);
  pinMode(analogOutPin, OUTPUT); // declare pin 11 to be an output:
  pinMode(analogInPin, INPUT); // declare pin A0 to be an input:
}
void loop() {
  sensorValue = analogRead ( analogInPin );
  Aout = sensorValue >>2; Serial.println (
  Aout ); analogWrite ( analogOutPin,Aout );
  delay(30);
}
```

5. Programming PWM V environment Atmel Studio

IN Arduino on base ATmega168/328 For PWM are used three half-
ra :

Timer 0 — conclusions 5 And 6

Timer 1 — conclusions 9 And 10

Timer 2 — conclusions 3 And 11

Each timer in the Arduino Uno is based on a microcontroller.

ATmega168/328 has two channels connected to the outputs. Timers in PWM mode are controlled by settings of control registers .

Examples subroutines settings regime PWM

1. Implementation PWM using Timer/Counter 2 and the division coefficient = 8

Control register TCCR2A

Bit	7	6	5	4	3	2	1	0
	COM2A1	COM2A0	COM2B1	COM2B0			WGM21	WGM20
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

1, COM 2A0 bits determine the PWM operating mode at the OC2A output.

Options values

COM2A1	COM2A0	Mode work exit OC2A
0	0	Ordinary digital port (Not PWM)
0	1	Alternately switching
1	0	Not inverse exit
1	1	Inverse exit

1 and COM 2B0 bits determine the PWM mode for the OC2B output. The value options are similar to those for COM2A1 and COM2A0.

Control register TCCR2B

Bit	7	6	5	4	3	2	1	0
	FOC2A	FOC2B			WGM22		CS2[2:0]	
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

Purpose discharges WGM22-WGM20 reflected V table

Mode	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCR0x at	TOV Flag Set on ⁽¹⁾
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Note:

1. MAX = 0xFF
2. BOTTOM = 0x00

Bits 7.6 - (FOC2 A,FOC 2B): Control comparators A And IN.

Bits active only V volume case, if bits WGM ask mode other than PWM.

Bits choice coefficient divisions

CA22	CA21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0		1	clk _{I/O} /1 (No prescaling)
0	1	0	clk _{I/O} /8 (From prescaler)
0	1	1	clk _{I/O} /32 (From prescaler)
1	0	0	clk _{I/O} /64 (From prescaler)
1	0	1	clk _{I/O} /128 (From prescaler)
1	1	0	clk _{I/O} /256 (From prescaler)
1	1	1	clk _{I/O} /1024 (From prescaler)

// this code sets up counter2 for an 8kHz Fast PWM wave @ 16Mhz Clock

```
#include <avr/io.h>
```

```
int main(void)
```

```
{
```

```
    DDRB |= (1 << 3);
```

```
    // PB3 is now an output
```

```
    OCR2A = 128; // set PWM for 50% duty cycle
```

```
    TCCR2A |= (1 << COM2A1); // set none-inverting mode
```

```
    TCCR2A |= (1 << WGM21) | (1 << WGM20); // set fast PWM Mode
```

```
    TCCR2B |= (1 << CS21); // set prescaler to 8 and starts PWM
```

```
    while (1);
```

```
    {
```

```
        // we have a working Fast PWM
```

```

    }
}

```

2. PWM With using timer 1. Installed maximum PWM frequency (divider disabled)

```

void pwm_initial ( ) {

    DDRB |= 1 << 1 ; // PB1 How exit, channel 1
    DDRB |= 1 << 2; // PB2 How exit, channel 2

    TCCR1A = 0; // Clear register data TCCR1B =
    0; // Clear register data TCNT1 = 0; //
    Installation lower limit

    TCCR1B |= (1 << CS 10); // Job without divider

    TCCR1A |= (1 << COM 1A1) | (1 << COM1B1); // Not inverse mode ICR1
    operation = 639; // 16 MHz / 1 * (1 + 639) = 25 kHz

                                // Installation top limit
                                // meaning V beat ICR1
    TCCR1A |= (1 << WGM11);
    TCCR1B |= (1 << WGM12) | (1 << WGM13);

    OCR1A = 416; // Initial meaning level 30% OCR1B = 416;
    // Initial meaning level 30%

}

```

Exercise For independent work

Execute All examples programs. Screenshots screen With Present the program code and results in a report. Draw conclusions about the feasibility of different approaches to generating analog signals.

PRACTICAL LESSON No. 6

DEVELOPMENT PROGRAMS MEASUREMENTS TEMPERATURES WITH THE USE OF ANALOGUE AND DIGITAL SENSORS

Goals laboratory works:

Learn the specifics of developing programs using analog and digital temperature sensors. Acquire skills in working with 1-Wire serial protocols . and I2C in the Arduino IDE and Atmel Studio tool environments .

Educational questions:

1. Measurement temperatures With using thermistor NTC 10K
2. Measurement temperatures With using sensor DS1820 And protocol 1-Wire
3. Measurement temperatures With using sensor Mirex And I2C protocol

1. Measurement temperatures With using thermistor NTC 10K

A thermistor (or thermistor) is a resistor that changes its electrical value. resistance V dependencies from temperature. There are two types of thermistors: PTC, with a positive temperature coefficient , and NTC, with a negative temperature coefficient . A positive coefficient means that the thermistor's resistance increases with increasing temperature. An NTC thermistor behaves in the opposite way.

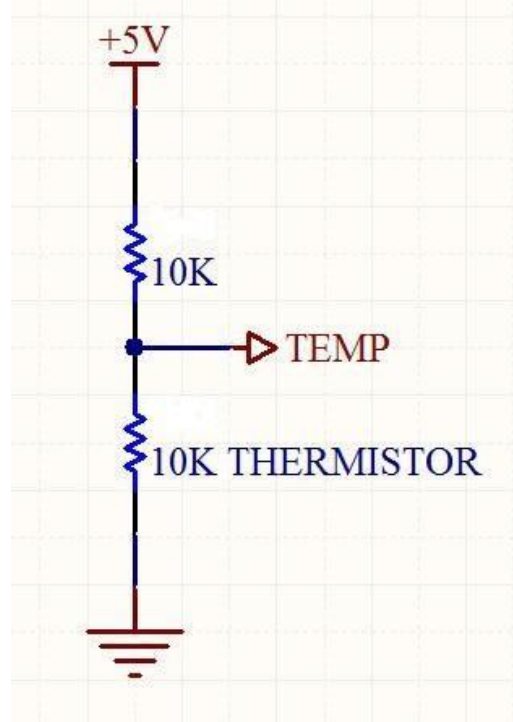
corresponds to room temperature—25°C. For example, thermistors with a nominal resistance of 100 kOhm and 10 kOhm are popular . These thermistors are often used in 3D printers.

This lab will use a B57164-K 102-J NTC thermistor with a nominal resistance of 10K at a temperature of 25 C° (Figure 1).



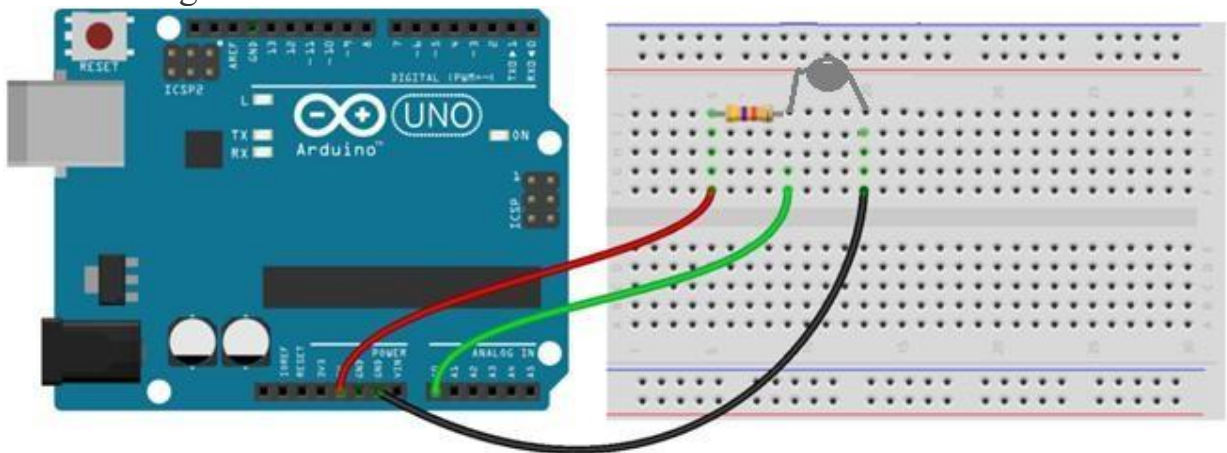
Drawing 1 - Thermistor B57164-K 102-J, 10 kOhm, 5%, NTC

Connection thermistor To Arduino . To measure Let's connect the thermistor's resistance as the lower arm of a voltage divider (Figure 2). The middle point of the divider is connected to the Arduino's analog input , A0.



Drawing 2 - Scheme connections thermistor

External view layout With connected sensor And resistor The divider is shown in Figure 3.



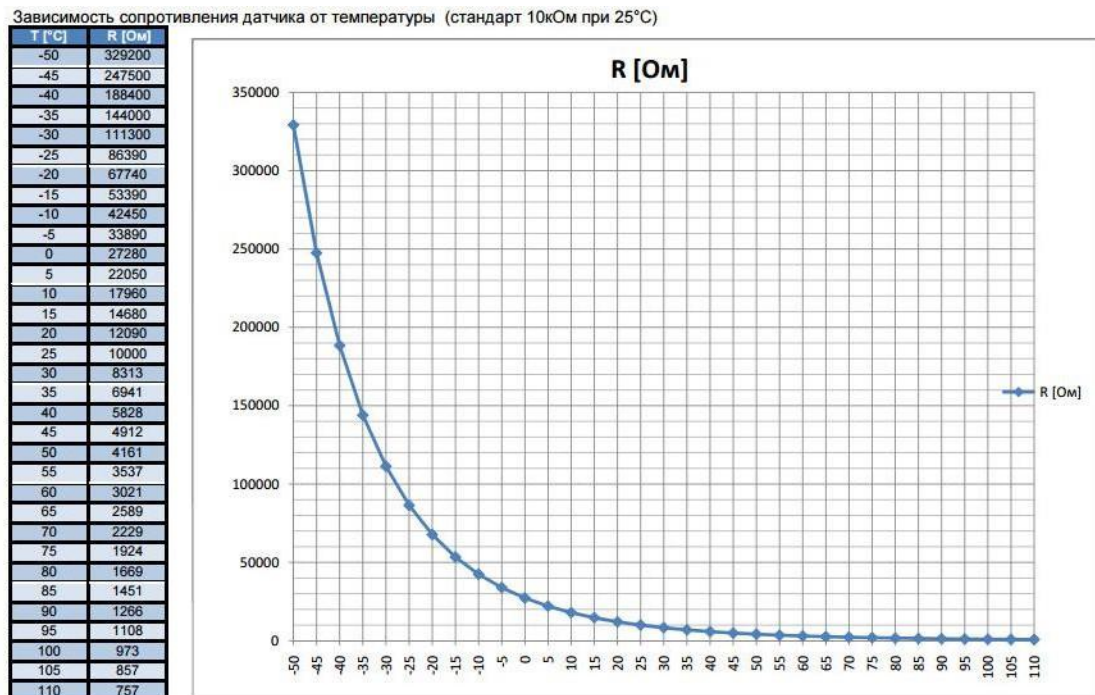
Drawing 3 - External view layout

To calculate To determine the temperature value using a thermistor, use the Steinhart -Hart formula:

$$\frac{1}{T} = A + B \ln(R) + C(\ln(R))^3$$

The equation has parameters A, B and C, which must be taken from the sensor specification.

Below (drawing 4) presented dependencies resistance thermistor from temperature in the form of a graph and in a tabular form.



Drawing 4 - Addition resistance sensor NTC type from temperatures

The graph shown in Figure 4 can be approximated by an exponential relationship (B-equation):

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \ln \left(\frac{R}{R_0} \right)$$

IN this equation unknown remains only parameter B, which for NTC thermistor is 3977. Other parameters:

- T₀ — indoor temperature V Kelvin, For which the thermistor rating is indicated; T₀ = 25 + 273.15;
- T — sought after temperature, V Kelvin;
- R — measured resistance thermistor V Ohms;
- R₀ — nominal resistance thermistor V Ohms.

Listing programs:

```
//Values from datasheet #
define RT0 10000 // Ohm
#define B 3977 // K
//-----
#define VCC 5.0 //Voltage power supply
#define R 10000 //R=10 kOhm

//variables
float RT, VR, ln, TX, T0, VRT;
```

```

void setup() {
  Serial.begin(9600);
  T0 = 25 + 273.15; //Temperature T0 from datasheet , we transform from
  Celsius to Kelvin
}

```

```

void loop ( ) {
  VRT = analogRead (A0);          //Read analog meaning VRT VRT =
  (VCC / 1023.00) * VRT;          //Convert to voltage
  VR = VCC - VRT;
  RT = VRT / (VR / R);            //Resistance RT

```

```

  ln = log ( RT / RT0);
  TX = (1 / (( ln / B) + (1 / T0))); //Temperature with

```

```

  thermistor TX = TX - 273.15;    //Convert in Celsius

```

```

  Serial.print ("Temperature:");
  Serial.print ("\t"); Serial.print
  (TX); Serial.print ("C\t\t");
  Serial.print (TX+273.15);      // Transform V Kelvin
  Serial.print ("K\t\t");
  Serial.print ((TX * 1.8) + 32); // Transform V Fahrenheit
  Serial.println ("F");
  delay ( 1000);

```

```

}

```

This sketch will measure temperature And output her once a second into the monitor port.

2. Measurement temperatures With using sensor DS1820 and the 1-Wire exchange protocol

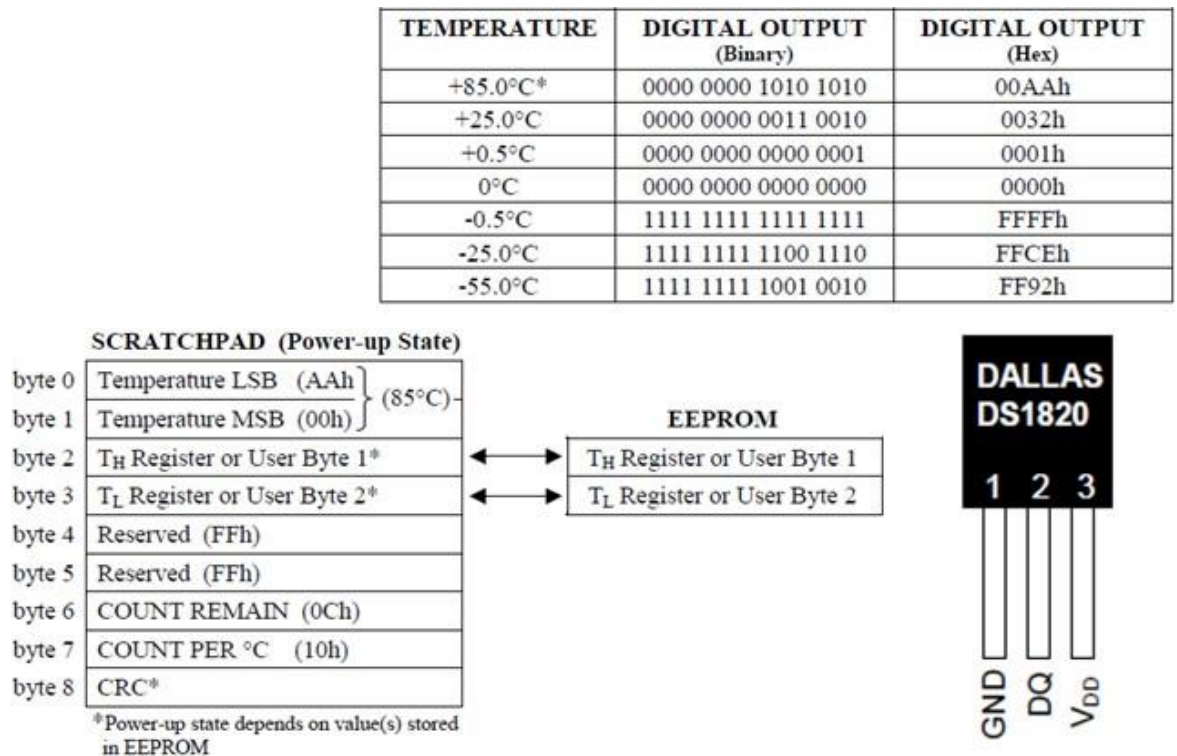
The DS18B20 is a digital thermometer with programmable resolution from 9 to 12 bits , which can be stored in the device's EEPROM memory. The DS18B20 communicates via a 1-Wire bus and can operate as a single device on the line or as part of a group. All bus processes are controlled by a central microprocessor.

The measurement range is from -55°C to $+125^{\circ}\text{C}$ and the accuracy is 0.5°C in the range from -10°C to $+85^{\circ}\text{C}$. IN addition, DS18B20 Maybe eat voltage

data lines (“ parasite power ”), in the absence of an external voltage source .

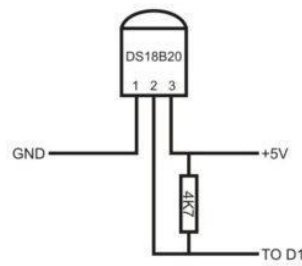
Each DS18B20 has a unique 64-bit serial code that allows it to communicate with a group of DS18B20 sensors installed on a single bus. This principle allows a single microprocessor to control multiple DS18B20 sensors distributed over a large area. Applications that can benefit from this feature include temperature control systems in buildings, equipment , or machines, as well as monitoring and control of thermal processes .

Drawing 5 - Digital temperature DS18B20 sensor



DS18B20 is a complete microprocessor device that can store measurement values, signal when the temperature goes beyond the set limits (we can set and change the limits themselves), change accuracy measurements, way interactions With controller And and much more. All this in a very compact package, which is also available in a waterproof version.

The chip has three outputs, only one of which is used for data; the other two are ground and power. The number of wires can be reduced to two by using a parasitic power supply circuit and connecting a single wire. Vdd with ground. K You can connect several DS18B20 sensors at once to a single data wire, and only one pin will be used on the Arduino board.



Drawing 6 - Connection sensor

- Error measurements Not more 0.5 WITH (For temperatures from -10C to +85C), What allows exactly define meaning temperature. No additional calibration required.
- Temperature range measurements lies V within from -55 WITH to +125 C.
- Sensor feeds on voltage from 3.3V up to 5V.
- Can programmatically ask maximum resolving ability up to 0.0625C, highest resolution 12 bits.
- Present function disturbing signal.
- Each device possesses your own unique serial code.
- Not required additional external elements.
- Can connect straightaway to 127 sensors To one lines connections.
- Information transmitted By protocol 1- Wire.
- For accession To microcontroller needed only 3 wires.
- There is a so-called parasitic power supply mode – in it there is power supply directly from the communication line. In this case, only 2 wires are required for connection. It is important that in this mode, there is no guarantee correct Job at temperatures higher 100C. Mode parasitic power supply is conveniently usually used for applications with a remote temperature sensor.

1.1 Programming process measurements temperatures V Arduino environment Uno

Connecting DS18B20 to Arduino . DS18B20 is a digital sensor . Digital sensors transmit the value of the measured temperature in the form of a specific binary code, which is sent to digital or analog pins Arduino and then decoded. The codes can be very different , ds18b20 works according to the protocol 1-Wire data. We will not go into the details of this digital protocol, but will only indicate the minimum necessary for understanding the principles of interaction.

Exchange information V 1-Wire is happening thanks to next operations:

- Initialization is the definition of the sequence of signals from which it begins measurement And other operations. Leading device

serves pulse reset, after this sensor must submit a presence impulse indicating readiness to perform an operation.

- Recording data - is happening broadcast byte data V sensor.
- Reading data - is happening reception byte from sensor.

For work With sensor us will be needed software security:

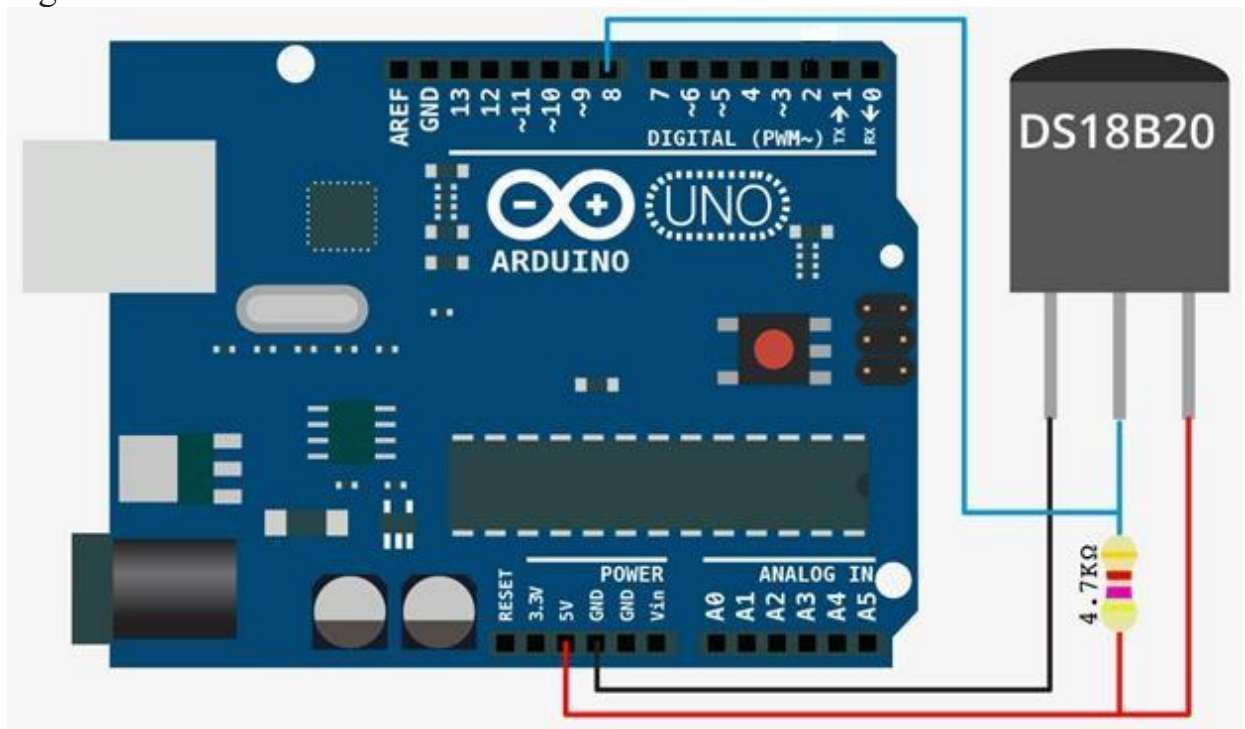
- Arduino IDE;
- Library OneWire , If is used some sensors on tire, you can use library DallasTemperature . She will work on top of OneWire .

From equipment you will need:

- One or some sensors DS18B20;
- Debugging pay Arduino uno ;
- Connectors;
- Resistor on 4.7 kOhm (V case connections one sensor will do resistor with a value of 4 to 10K);
- Assembly pay;
- USB cable For connections To computer.

the Arduino UNO board simply: GND from the temperature sensor is connected to GND of the Arduino , Vdd is connected to 5V, Data – to any digital pin .

The simplest connection diagram for the DS18B20 digital sensor is shown in Figure 7.



Drawing 7 - Connection sensor temperatures To board Arduino Uno

In parasitic power supply mode, the Vdd contact from the sensor is connected to GND on Arduino - V this case will come in handy only two wires. Work V

parasitic mode better not to use without necessary, as performance and stability may be affected.

The algorithm for obtaining temperature information in a sketch consists of the following steps:

- Definition addresses sensor, examination his connections.
- A command is sent to the sensor requesting it to read the temperature and store the measured value in the register. This procedure takes longer than the others , requiring approximately 750 ms.
- A command is given to read information from the register and send the received value to the “port monitor”,
- If required, conversion to degrees Celsius / Fahrenheit is performed.

The simplest sketch for working with a digital sensor looks like this . (In the sketch, we use the OneWire library , which we'll talk about in more detail later.)

```
#include < OneWire.h >
/*
 * Description interactions With digital sensor ds18b20
 * Connection ds18b20 To Arduino through pin 8
 */
OneWire ds ( 8); // We create object OneWire For tires 1-Wire, With with
the help of which the work with the sensor will be carried out

void setup( ){
  Serial.begin (9600);
}

void loop ( ){
  // We define temperature from sensor DS18b20 byte data
  [ 2]; // Place for temperature value

  ds.reset (); // Let's begin interaction with reset everyone previous commands
and parameters
  ds.write (0xCC); // We give sensor DS18b20 team skip Search by
address. In our case, only one device.
  ds.write (0x44); // We give the DS18b20 sensor a command to measure
the temperature . Itself meaning temperatures We more Not we get - the
sensor will put it in the internal memory

  delay ( 1000); // Chip measures temperature, A We we're waiting.

  ds.reset (); // Now we're getting ready get meaning measured temperature
  ds.write (0xCC);
```

```

    ds.write(0xBE); // Просим передать нам значение регистров со значе-
нием температуры

    // Получаем и считываем ответ
    data[0] = ds.read(); // Читаем младший байт значения температуры
    data[1] = ds.read(); // А теперь старший

    // Формируем итоговое значение:
    // - сперва "склеиваем" значение,
    // - затем умножаем его на коэффициент, соответствующий раз-
решающей способности (для 12 бит по умолчанию - это 0,0625)
    float temperature = ((data[1] << 8) | data[0]) * 0.0625;

    // Выводим полученное значение температуры в монитор порта
    Serial.println(temperature);
}

```

Library OneWire For work With DS18B20

Main teams libraries OneWire :

- search (addressArray) - is looking for temperature sensor, at If it is located , its code is written to the addressArray array ; otherwise, it is false .
- reset_search () - is produced search on first device.
- reset () - execution reset tires before those, How contact with the device.
- select (addressArray) - is selected device after reset operation, its ROM code is written.
- write (byte) - is produced recording byte information on device-state .
- write (byte , 1) - similarly write (byte), But V mode parasitic nutrition.
- read () - reading byte information With devices.
- crc8(dataArray , length) - calculation CRC code. dataArray - selected array, length – code length.

It's important to configure the power mode correctly in the sketch. For parasitic power, line 65 should contain ds.write (0x44, 1); For external power, line 65 should contain ds.write (0x44).

Write allows you to send a command to the temperature sensor. The main commands sent in the form of bits are :

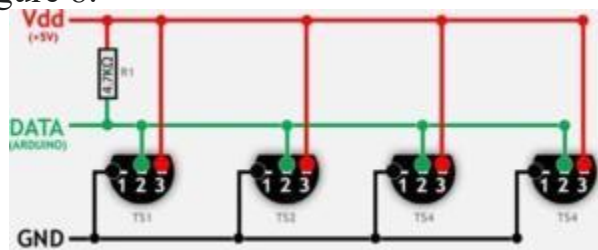
- 0x44 - measure temperature, write down received meaning V SRAM.
- 0x4E - recording 3 byte V third, fourth And fifth SRAM bytes .
- 0xBE - consistent reading 9 byte SRAM.
- 0x48 - copying third And fourth bytes SRAM V EEPROM.

- 0xB8 - copying information from EEPROM V third And fourth byte of SRAM.

- 0xB4 – returns the power supply type (0 – parasitic, 1 – external).

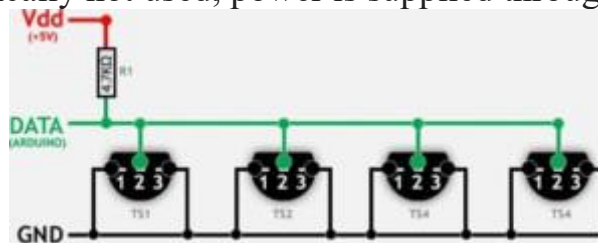
Connecting multiple DS18B20 temperature sensors to an Arduino All sensors DS18B20 connect parallel, For them everyone enough - exactly one resistor. Using the OneWire library , you can simultaneously read all data from all sensors. If you connect more than 10 sensors, you need to select a resistor with a resistance of no more than 1.6 kOhm. Also, for more accurate temperature measurements, you need to place an additional 100-120 Ohm resistor between the data output on the Arduino board and the data on each sensor. Find out, which sensor received a particular value can be determined using a unique 64-bit serial code that will be issued as a result of executing the program.

To connect temperature sensors in normal mode, you need to use the diagram shown in Figure 8.



Drawing 8

In parasitic mode power supply circuit looks like otherwise (Figure 9). Contact Vdd is practically not used, power is supplied through the data output .



Drawing 9

1.2 Programming process measurements temperatures in Atmel environment Studio

Program, the listing of which is presented below, contains temperature measurement and temperature display on a four-character seven - segment indicator. Periodicity survey The sensor is set by a timer. The dynamic indication mode is selected for the indicator.

Implementation of exchange with a temperature sensor via an interface One-Ware using PC0 for input and PC1 for output.

Control The selection of a digit on the seven-segment indicator is carried out through port B, terminals PB1-PB4. Control segments through port D.

/*

```

* LessonTerm_LCD.c
*/

#define F_CPU 16000000L
#include <avr/io.h>
#include <math.h>
#include <util/delay.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <stdlib.h>

volatile int display = 0;
int Tcount;
int Time;
uint8_t simv;
uint8_t count=1;
//-----0---1---2---3---4---5---6---7---8---9
unsigned char SEGMENTE[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
0x7D, 0x07, 0x7F, 0x6F};
unsigned char num_position [ ] = {0x10, 0x08, 0x04,
0x02 }; unsigned char segcounter = 0;
unsigned char data[ 4 ];
unsigned char that ;

ISR(ISR) TIMER 1_OVF_vect )
{
    TCNT1 = 65500;; //set the initial value of TCNT1 for period 0.05 s

    Tcount ++;

    dat =0;
    dat = display % 10000 / 1000;
    if( dat == 0)data [0]=~SEGMENTE[0]; else data[0]=~SEGMENTE[ dat ];
    dat =display % 1000 / 100; if( dat ==0)data[1]=~SEGMENTE[0]; else
data[ 1]= ~SEGMENTE[ dat ];
    dat =display % 100 / 10; if( dat == 0)data [2]=~SEGMENTE[0]; else data
[2]=~SEGMENTE[ dat ];
    dat = display % 10; if( dat == 0)data [3]=~SEGMENTE[0];
else da- ta[3]=~SEGMENTE[ dat ];

    PORTB = num_positions [ segcounter ]; // Select indicator PORTD
    = data[ segcounter ];
    segcounter ++;

```

```

    if( segcounter > 3) segcounter = 0;
}

// Sensor temperatures

/* We define Where connected sensor */ #define
THERM_PORT PORTC
#define THERM_DDR DDRC
#define THERM_PIN PINC
#define THERM_DQ PC1
#define THERM_DQ_IN PC0
/* Macros For changes regime input / output */ #define
THERM_LOW() THERM_PORT&=~(1<<THERM_DQ)
#define THERM_HIGH() THERM_PORT|=(1<<THERM_DQ)

// reset sensor
uint8_t therm_reset (void
){ uint8_t i;
  // we omit leg down on 480uS
    THERM_LOW();
  _delay_us ( 500);
    THERM_HIGH();
  // we raise line on 60uS
  _delay_us ( 60);
  // we get meaning on lines V period 480uS i =(
    THERM_PIN & (1<<THERM_DQ_IN));
  _delay_us ( 420);
  // we return meaning (0=OK, 1=sensor Not found) return i;
}

// function dispatches void bit
therm_write_bit ( uint8_t bit){
  // we omit on 1uS THERM_LOW(
  );
  _delay_us ( 5);
  // If we want to send 1, we raise the line (if not,
  leave it as is)
  if(bit) THERM_HIGH();
  // we're waiting 38uS And we raise line
  _delay_us ( 38 );
  THERM_HIGH();
}

```

```

}

// reading bat
uint8_t therm_read_bit (void ){
    uint 8_t bit=0;
    // we omit on 1uS THERM_ LOW(
    );
    _delay_us ( 2 ) ;
    THERM_HIGH();
    // we raise on 11uS
    _delay_us ( 11);
    // read state if( THERM_PIN&(1<<THERM_DQ_IN))
    bit=1;
    // we're waiting 45 ISS And we return meaning
    _delay_us ( 45);
    return bit ;
}

// we read byte
uint8_t therm_read_byte (void ){ uint
    8_t i =8, n=0;
    while (i -- ){
        // we shift to the right on 1 And save following value n >= 1;
        n|=( therm_read_bit ( ) <<7);
    }
    return n;
}

// we are sending byte
void therm_write_byte ( uint8_t byte){
    uint8_t i =8;
    while( i -- ){
        // we are sending bit And we shift to the right on 1 therm_write_bit
        (byte&1);
        byte>>=1;
    }
}

void start_temperature(void){
    therm_reset();
    therm_write_byte(0xcc);
    therm_write_byte(0x44);
}

```

```

}

uint16_t read_temperature(void){
    uint8_t temperature[2];
    int16_t digit;
    therm_reset();
    therm_write_byte(0xcc);
    therm_write_byte(0xbe);
    temperature[0]=therm_read_byte();
    temperature[1]=therm_read_byte();
    therm_reset();
    digit=temperature[0]>>4;
    digit|=(temperature[1]&0xF)<<4;
    _delay_ms(50);
    return(digit);
}

int main()
{
    uint8_t n=0;
    uint8_t Termnew,Term;
    DDRD=0xFF;
    DDRB |= 0x1E; // configure PB 1..PB4 to output (indicator control )
    DDRB|=( 1<<5);

    TCCR1B = (1<<CS 12)|(0<<CS11)|(1<<CS10); // we're setting up
timer divider 1
    |= (1 << TOIE1); // enable timer overflow interrupt
    TCNT1 = 65500;      // we exhibit initial meaning TCNT1

    sei ( );           // we exhibit bit general permissions interruptions

    while( 1)          // eternal cycle
    {

        // if( Tcount>300 ) {n=n+1;display=n;Tcount=0;}
        if(          Tcount>300          )      {Termnew=read_temperature();
if(abs(Termnew-Term)<5)Term=Termnew;Tcount=0;}
        if(Tcount==0){ start_temperature();}
        _delay_ms(500);
    }
    return 0;
}

```

Execute the program input in the Atmel text editor Studio . Debug the program and compile it. Disassemble the program into modules for working with a temperature sensor, a module for working with an indicator, and a module for working with a timer.

2 Dimension temperatures With using sensor Mlex And I2C protocol

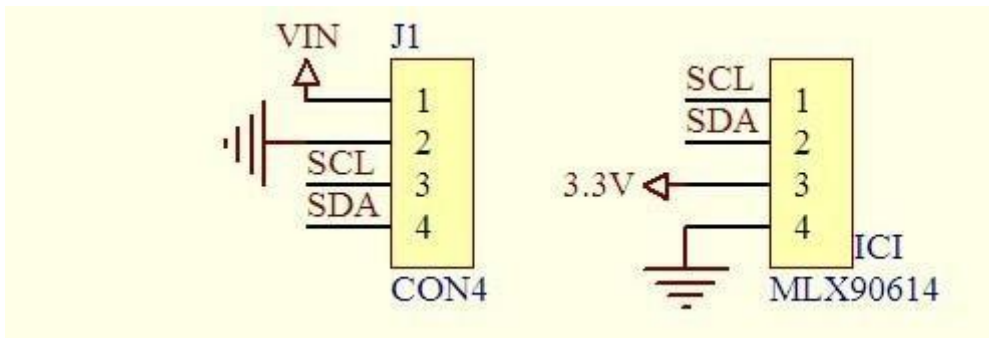
Infrared thermometers The MLX90614 operates on a phenomenon called blackbody radiation. Anything above absolute zero has moving molecules inside. The higher the temperature, the faster the molecules move. As they move, the molecules emit infrared radiation—a type of electromagnetic radiation below the visible spectrum. As they heat up, they emit more infrared radiation and even begin to emit visible light. This is why heated metal can glow red or even white . Infrared thermometers detect and measure this radiation.

Main characteristics infrared sensor MLX90614:

- Range measurable temperature: -40°C ... + 125°C For ambient temperature ; 70°C ... + 380°C for object temperature;
- Accuracy measurements - 0.5°C V wide range temperatures (0°C ... + 50° C for both options);
- Maximum permission measurements temperatures 0.02°C;
- Protocol informational exchange - I2 C ;
- Opportunity calibration;
- Distance to object - 1 cm.

The sensor can be supplied on the board or separately; the contacts for connecting the board and the sensor separately are shown in the figure below.

Arduino pin:	MLX90614 pin:
GND	GND
+3.3v	VCC (+2.8-3.6v)
A5	SCL (I ² C Clock)
A4	SDA (I ² C Data)



Drawing 9- Connection sensor V view boards And actually IR sensor Ready same module With strapping looks like Here How shown on drawing 10.



Drawing 10

Separate sensor Can connect By next scheme Write it down in

Arduino IDE next sketch:

```
#include <i2cmaster.h>
void setup(){
  Serial.begin(9600);
  Serial.println("Setup...");

  i2c_init(); //Инициализация шины i2c
  PORTC = (1 << PORTC4) | (1 << PORTC5); //устанавливаем PC4 и PC5 ( SDA SCL) в 1
}

void loop(){
  int dev = 0x5A << 1; //0xB4 – адрес датчика
  int data_low = 0;
```

```

int data_high = 0 ;
int pec = 0;
i2c_start_wait(dev+I2C_WRITE);

i2c_write(0x07);

// read
i2c_rep_start(dev+I2C_READ);
data_low = i2c_readAck ( ) ; //Reading ml byte And formation of ack
data_high = i2c_readAck ( ) ; // Reading Art. byte And formation ack
dog = i2c_readNak ( ) ;
i2c_stop ( ) ;

//Recalculation temperatures V degrees
double tempFactor = 0.02 ; // 0.02 degrees per LSB (measurement resolution of the MLX90614 )
double tempData = 0x0000; // zero out the data
int frac; // data past the decimal point

// Removal elders bits, containing information about mistakes
tempData = ( double )( (( data_high & 0x007F ) << 8 ) + data_low );
tempData = (tempData * tempFactor)-0.01;

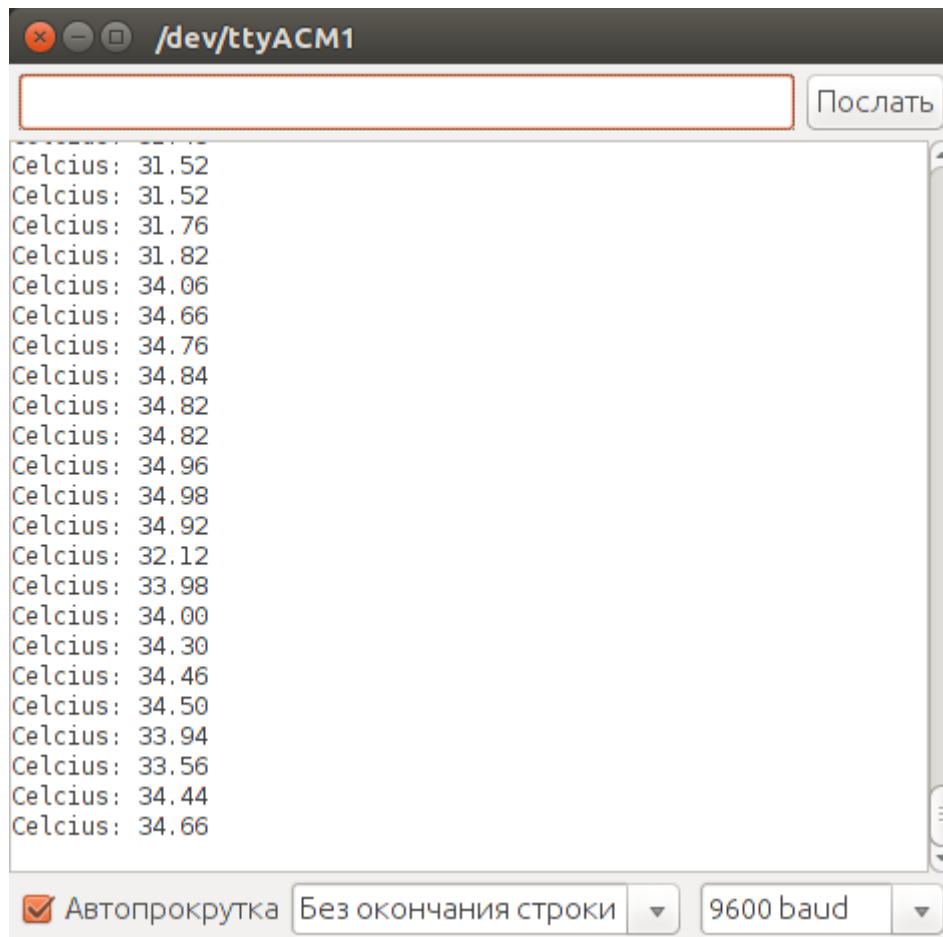
float celcius = tempData - 273.15;

Serial.print("Celcius: ");
Serial.println(celcius);

delay ( 1000 ); // expectation 1 With And transition on beginning of the cycle
}

```

By loading the sketch into the Arduino and enabling the serial monitor, we can see how the sensor readings change depending on its direction. The figure shows the temperature measurement of a palm held 10 cm from the sensor.



IN sketch is used library `<i2cmaster.h>`
library can be found on the website [http://arduino-
ru.blogspot.com/2008/12/i2cmaster-arduino.html](http://arduino.ru.blogspot.com/2008/12/i2cmaster-arduino.html).

download The

IN composition standard Arduino libraries IDE For work with interface I2C is used by `<Wire.h>`.

Example programs With usage m `<Wire.h>`:

```
#include <Wire.h> // подключаем библиотеку "Wire"
byte val = 0; // значение для передачи потенциометру

void setup() {
    Wire.begin(); // подключаемся к шине I2C как мастер
}

void loop() {
    Wire.beginTransmission(44); // начинаем обмен с устройством с I2C адресом
    "44" (0x2C)
    Wire.write(byte(0x00)); // посылаем инструкцию записи в регистр RDAC
    Wire.write(val); // задаём положение 64-позиционного потенциометра
    Wire.endTransmission(); // завершаем I2C передачу

    val++; // инкрементируем val на 1
    if (val == 63) { // по достижении максимума потенциометра
        val = 0; // сбрасываем val
    }
    delay(500);
}
```

PRACTICAL LESSON No. 7

STUDY FEATURES PROGRAMMING LIQUID CRYSTAL INDICATORS

Goals works:

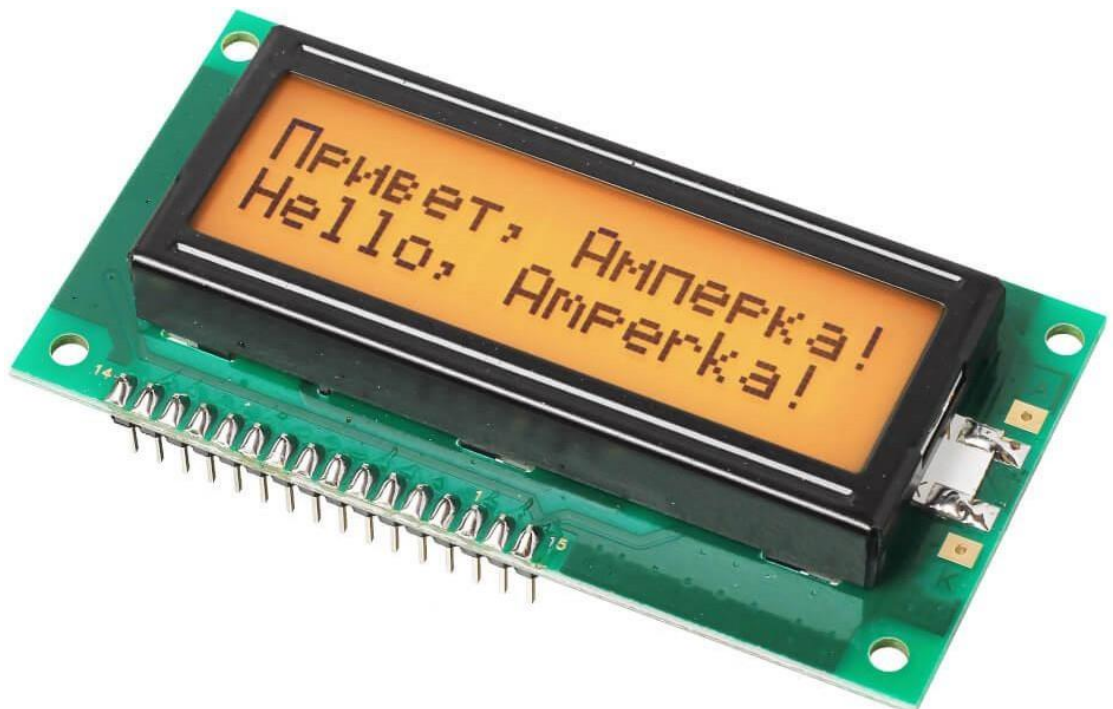
- to study the features of programming text liquid crystal display (LCD) screens using Arduino platforms Uno and the Arduino IDE tool environment ;
- learn programming of the USART serial interface to use a personal computer as a monitor;
- acquire skills in developing programs using information display tools.

Educational questions:

1. Liquid crystal two-line display 16x2 MT- 16S2
2. Programming various tasks with displaying the results of work on the LCD
- 3 Programming USART to organize communication between the microcontroller and a personal computer

1 Liquid crystal two-line display MT- 16S2

Text screen 16×2 MT-16S2 is used to display sensor readings, simple menus, prompts and greetings in the form of two lines of 16 characters each (Figure 1).



Drawing 1 - External view display

Table 1 shows the purpose of the indicator board pins and possible pin assignments. Arduino Uno when using the LCD in the mode of transmitting control codes of 4 digits.

Table 1 - Purpose conclusions indicator

Conclusion	Designation		Conclusion boards Arduino Uno
1	GND	General conclusion	GND
2	VCC	Voltage nutrition (5 IN)	5V
3	VO	Control contrast	GND
4	RS	Choice register (command or data)	11
5	R/W	Choice regime records or readings	GND
6	E	Permission appeals To indicator (as well as data strobe)	12
7	DB0	Data bus (8-bit mode) (least significant bit in 8- bit mode)	—
8	DB1	Tire data (8 bit mode)	—
9	DB2	Tire data (8 bit mode)	—
10	DB3	Tire data (8 bit mode)	—
11	DB4	Data bus (8-bit and 4-bit modes) (least significant bit in 4-bit mode)	5
12	DB5	Data bus (8-bit and 4- bit modes)	4
13	DB6	Data bus (8-bit and 4- bit modes)	3
14	DB7	Data bus (8-bit and 4- bit modes)	2
15	VCC	Nutrition backlighting (+)	5V
16	GND	Nutrition backlighting (—)	GND

On drawing 2 shown connection indicator MT-16S2 To Arduino debug board Uno using a breadboard.

Characteristics display MT- 16S2:

Type display: Text Color:

Monochrome

Technology: LCD (Liquid Crystal Display)

Indication: 2 lines of 16 characters Matrix

drivers: KB1013VG6 Interface: parallel 4/8 bit

Type backlight: LED

Backlight color (depending on model): red / green / blue / black - amber

Color symbols (depends from models): green / black / white Supply voltage:
5 V

Maximum current consumption: 1 mA

Backlight current consumption: 100 mA

Voltage logical levels: 3.3–5

Dimensions:
84×44×13 mm

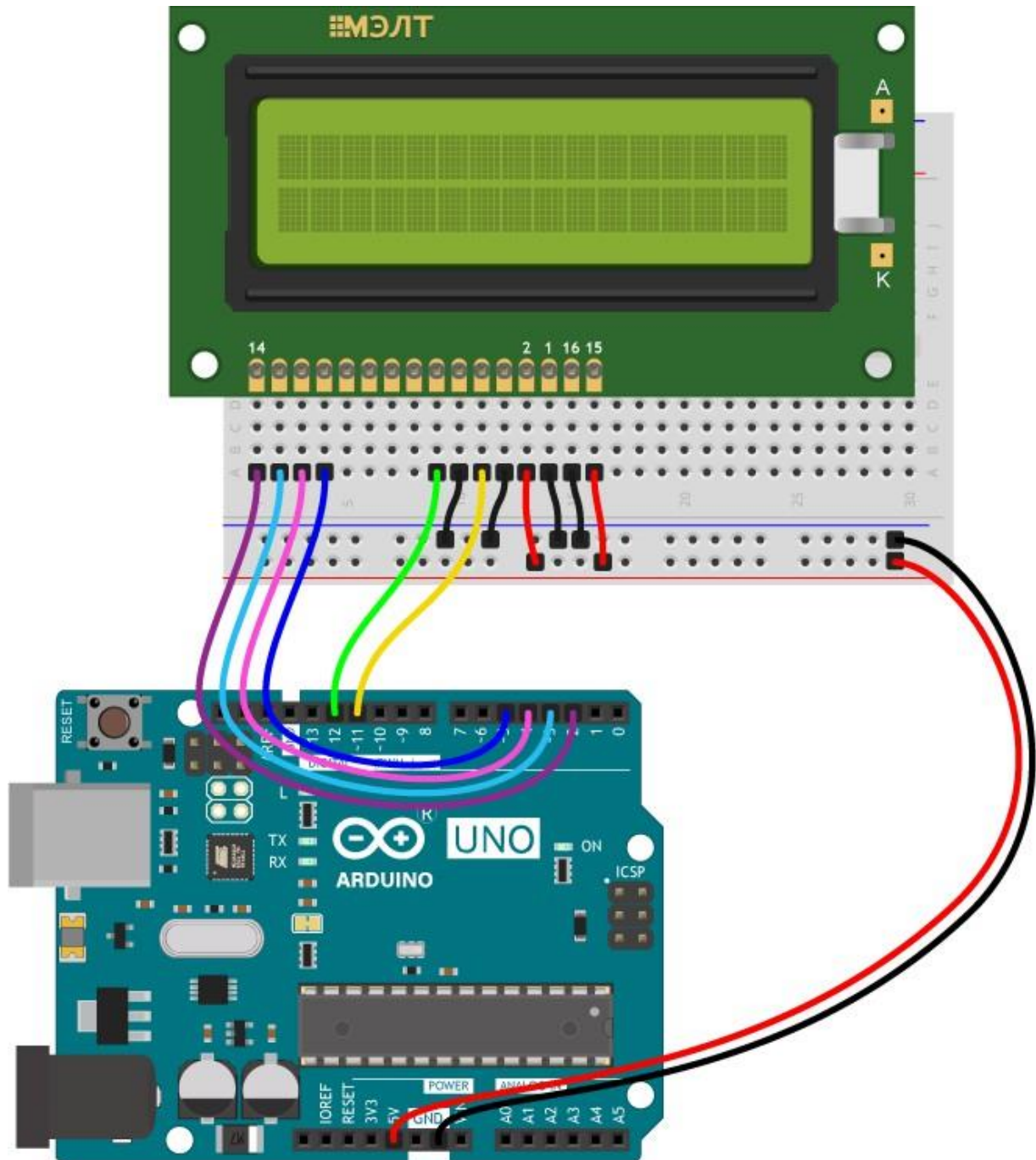


Figure 2 – Connecting the indicator to the debug board Arduino Uno For

simplifications work With LCD display use built-in lib -
lioteku Liquid Crystal . In it you will find code examples with detailed com-
ments .

2. Programming various tasks With display results of work on LCD

To acquire skills in working with LCD, we will perform several typical tasks with the output of text that does not change during operation (for example, a splash screen at the beginning of a program or a greeting), and then we will consider how the output of numerical data is carried out.

2.1 Conclusion text

To display the greeting text, use the following program listing 1:

Listing 1

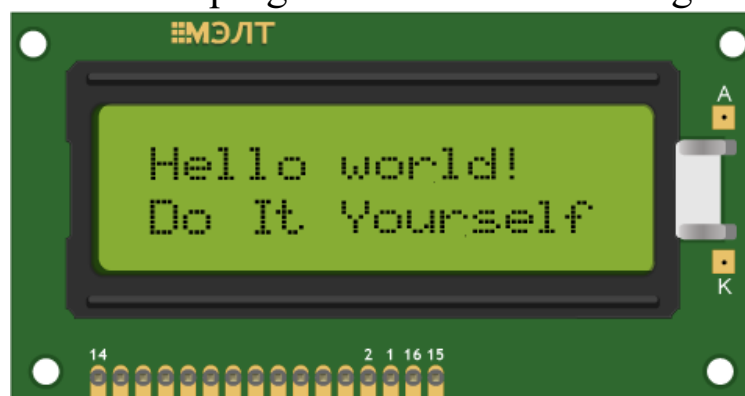
```
// let's connect standard library LiquidCrystal # include <
LiquidCrystal.h >

// initialize screen object, we transmit used
// For connections contacts on Arduino V in order:
// RS, E, DB4, DB5, DB6, DB7
Liquid Crystal lcd ( 11, 12, 5, 4, 3, 2);

void setup ( ) {
  // we are installing size (quantity columns And lines) lcd.begin (16, 2) ;
  // we are printing first lcd line .
  print ( " Hello world ");
  // we are installing cursor V column 0, line 1
  // on itself case This second line, because numbering it begins With zero lcd.setCursor (0,
  1);
  // we are printing the second line
  lcd. print ( "Do It Yourself");
}

void loop ( ) {
}
```

Results work programs shown on drawing 3.



Drawing 3 - Conclusion greetings on screen LCD

To output text using Cyrillic characters, you can use two methods:

- With help built-in tables character generator;
- With help libraries LiquidCrystalRus .

Character generator table. The display module stores two character generator pages in memory (Figures 4, 5), which consist of various symbols and letters.

Старшая цифра кода символа (в шестнадцатеричном виде)

Младшая цифра кода символа (в шестнадцатеричном виде)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	x	...		Ø	Q	P	`	P	...	±	Б	Ю	Ч	.	Д	¼
1	x	!!	!	1	A	Q	а	я	...	≡	Г	Я	ш	і	Ц	¼
2	x	÷	"	2	B	R	б	р	...	•	Ё	б	ъ	и	Щ	¼
3	x	→	#	3	C	S	c	s	...	...	Ж	В	ы	!!	д	¼
4	x	←	\$	4	D	T	d	t	...	✓	З	Г	ь	Ъ	Ф	¼
5	x	\	%	5	E	U	e	u	...	і	И	ё	э	х	ц	¼
6	x	г	&	6	F	V	f	v	...	1	И	ж	ю	ъ	щ	¼
7	x	н	'	7	G	W	g	w	...	2	Л	з	я	І	'	¼
8	б	Ø	<	8	H	X	h	x	...	3	П	и	«	И	"	¼
9	µ	Ø	>	9	I	Y	i	y	...	т	о	У	й	»	↑	¼
A	Û	≤	*	:	J	Z	j	z	...	-	€	Ф	к	«	↓	¼
B	l	≥	+	:	K	[	k	]	...	<	■	Ч	л	"	Н	¼
C	ï	√	,	<	L	φ	l	φ	...	>	■	Ш	М	Н	и	¼
D	ï	¥	-	=	M	]	m	]	...	т	б	Н	і	Н	+	¼
E	€	≠	.	>	N	^	n	^	...	б	Ы	п	f	т	÷	¼
F	€	×	/	?	O	_	o	_	...	б	Э	Т	£	.	о	¼

Drawing 4 - Code table "0"

Старшая цифра кода символа (в шестнадцатеричном виде)

Младшая цифра кода символа (в шестнадцатеричном виде)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	x	¼		Ø	P	°	p	i	†	■	°	А	Р	а	р	
1	x	½	!	1	A	Q	a	q	1	4	ÿ	±	Б	С	б	с
2	x	¾	"	2	B	R	b	r	2	▼	ÿ	♦	В	Т	в	т
3	x	¾	#	3	C	S	c	s	3	▲	£	◇	Г	У	г	у
4	x	÷	\$	4	D	T	d	t	!!	...	l0	«	Д	Ф	д	ф
5	x	≡	%	5	E	U	e	u	...	l-	¥	”	Е	Х	е	х
6	x	г	&	6	F	V	f	v	↑	ll	o	¶	Ж	Ц	ж	ц
7	x	✓	'	7	G	W	g	w	↓	ll	ſ	f	З	Ч	з	ч
8	P	★	<	8	H	X	h	x	€	у	ё	ё	И	Ш	и	ш
9	T	†	>	9	I	Y	i	y	к	н	ø	µ	Й	Щ	й	щ
A	¶	≤	*	:	J	Z	j	z	ø	ə	Є	е	К	Ъ	к	ъ
B	■	≥	+	;	K	[	k	[	f	f	«	»	Л	Ы	л	ы
C	■	®	,	<	L	\	l		К	к	€	♪	М	Ь	м	ь
D	■	Р	-	=	M	]	m	>	Н	н	-	♪	Н	Э	н	э
E	■	≠	.	>	N	^	n	~	У	у	ø	ø	О	Ю	о	ю
F	■	≈	/	?	O	_	o	ø	ø	ø	ï	ï	П	Я	п	я

Drawing 5- Code page "1"

To display a symbol on the display, it is necessary to transmit its number in the hexadecimal system from the character generator table.

So letter I corresponds code 0xB1 V hexadecimal system. To display the string " Yandex " on the screen, you must explicitly embed the character code into the string using the \x ## sequence :

```
lcd.print("\xB1ndex");
```

You can mix regular characters and explicit codes in a single string as much as you like. The only caveat is that after the compiler sees the `\x` sequence in a string, it reads all characters that follow it, which could be hexadecimal digits, even if there are more than two. Because of this, you can't use characters in the range 0-9. And A-F followed by a two-digit character code, otherwise the display will show an incorrect information. To go around this moment, Can use the fact that two lines written next to each other are glued together.

Compare two lines code For conclusion inscriptions " Yeee ":

```
lcd.print ("xB1eee"); // error
lcd.print ("\xB1" "eee"); // Right
```

Using received information we will deduce on display message
"Hello, "Amperka !":

Listing 2:

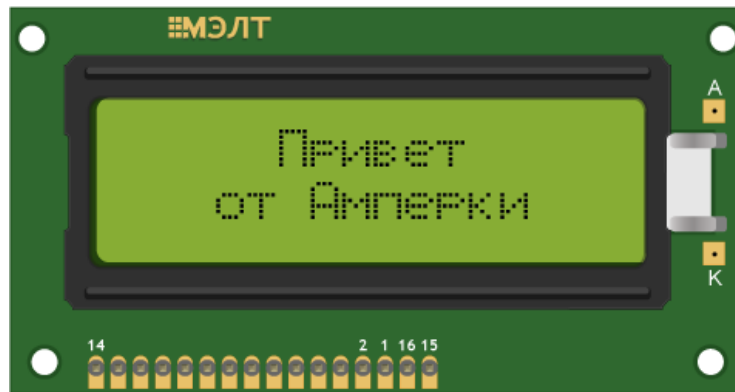
```
// let's connect standard library LiquidCrystal # include <  
LiquidCrystal.h >
```

```
// initialize screen object, we transmit used  
// For connections contacts on Arduino V in order:  
// RS, E, DB4, DB5, DB6, DB7  
Liquid Crystal lcd ( 11, 12, 5, 4, 3, 2);
```

```
void setup ( ) {  
  // we are installing size (quantity columns And lines) lcd.begin (16, 2) ;  
  // we are installing cursor V column 5, line 0  
  // on itself case This first line, because numbering it begins With zero lcd.setCursor (5, 0);  
  // print the first line lcd.print  
  ("xA8""p""xB8\xB3""e\xBF ");  
  // we are installing cursor V column 3, line 1  
  // on itself case This second line, because numbering it begins With zero lcd.setCursor (3,  
  1);  
  // print the second line lcd. print ( "o\xBF A\  
  xBC \xBE "" ep \xBA \xB8");  
}
```

```
void loop ( ) {  
}
```

Results work programs shown on drawing 6.



Drawing 6

Switching pages character generator.

The display module stores two pages of the character generator in its memory. By default, page zero is set. To switch between pages, use the following methods:

```
// switching With null pages on first command
(0x101010);
// switching With first pages on null command
(0x101000);
```

The display cannot display characters from different pages at the same time. Let's look at an example (Listing 3) in which the same string will be displayed differently — V dependencies from selected pages (drawing 7).

Listing 3:

```
// Let's connect standard library LiquidCrystal # include <
LiquidCrystal.h >
```

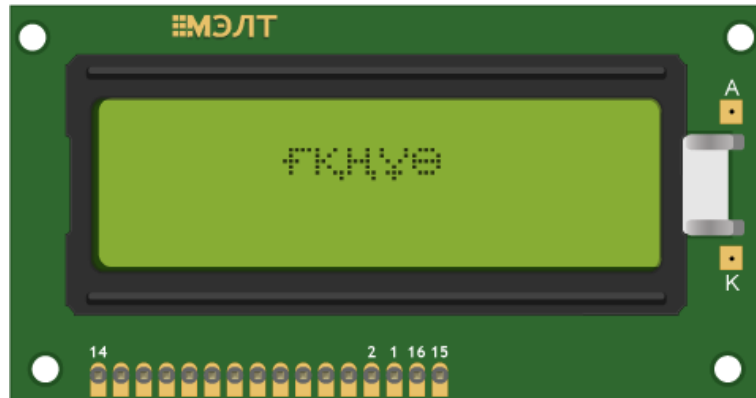
```
// Initialize screen object, we transmit used
// For connections contacts on Arduino V in order:
// RS, E, DB4, DB5, DB6, DB7
LiquidCrystal lcd ( 11, 12, 5, 4, 3, 2);
```

```
void setup ( ) {
  // we are installing size (quantity columns And lines) lcd.begin (16, 2) ;
  // we are installing cursor V column 5, line 0
  // on itself case This first line, because numbering it begins With zero lcd.setCursor (5, 0);
  // print line lcd.print ("x9b\x9c\x9d\x9e\x9f");
}
void loop ( ) {
  // we are installing 0 village character generator (costs By default) lcd.command
  (0b101000);
  // we're waiting 1
  second delay ( 1000);
  // we are installing 1 village character generator
```

```

lcd.command(0b101010);
// we're waiting 1 a
second delay( 1000);
}

```



Drawing 7

Usage libraries LiquidCrystalRus

You don't have to struggle with a character generator to get Russian symbol. For solutions problems download And install library - ku [LiquidCrystalRus](#) .

LiquidCrystal library . with the addition of the Russian language. The code added to the library transforms Russian UTF8 characters into the correct codes for the text screen.

IN quality example we will deduce phrase "Hello from Amperki » on the display.

```

// let's connect library LiquidCrystalRus #
include < LiquidCrystalRus.h >

// initialize screen object, we transmit used
// For connections contacts on Arduino V in order:
// RS, E, DB4, DB5, DB6, DB7
LiquidCrystalRus lcd(11, 12, 5, 4, 3, 2);

void setup() {
  // we are installing size ( quantity columns And lines ) screen lcd.begin (16, 2);
  // we are installing cursor V column 5, line 0
  // on itself case This first line, because numbering it begins With zero lcd.setCursor
  (5, 0);
  // we are printing first line lcd.print
  ("Hello");
  // we are installing cursor V column 3, line 1
  // on itself case This second line, because numbering it begins With zero
  lcd.setCursor (3, 1);
  // we are printing the second lcd line .
  print ( "from Amperka ");
}

void loop ( ) {
}

```

To output digital data, it is necessary to convert the binary code symbol by symbol into codes ASCII. An example listing of the conversion routine is given below:

```
void ConvertNum_ASCII ( uint32_t Num)
{
    int ost,has ,q ;
    uint8_t mData[10]; int c=0;
    has=Num;
        for (q=0;q<11;q++) {mDATA[q]=0x30;};
    while(has)
    {
        ost=has %10;
        has=has/10;
        switch(ost)
        { case 0: mData[c]=0x30; break;
          case 1: mData[c]=0x31; break;
          case 2: mData[c]=0x32; break;
          case 3: mData[c]=0x33; break;
          case 4: mData[c]=0x34; break;
          case 5: mData[c]=0x35; break;
          case 6: mData[c]=0x36; break;
          case 7: mData[c]=0x37; break;
          case 8: mData[c]=0x38; break;
          case 9: mData[c]=0x39; break;
        }
        c++;
    }
    // UART_putString0 (" = ");
    lcd.print(" = ");
    for(q=c;q>=0;q--)
    {
        //          UART_putchar0 (mDATA[q]);
        lcd.print(mDATA[q]);
    }
    //          UART_putString0 (" ");
    lcd.print ( " ");
};
```

3 Programming USART to organize communication with the microcontroller with a personal computer

The USART interface is a serial universal synchronous-asynchronous receiver-transmitter. Data transmission in the USART is carried out at regular intervals. This time interval is determined by the specified USART speed and is specified in bauds (For characters that can take values equal to only zero or one, baud is equivalent to bits V second). Exists common row standard

speeds: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600 baud.

In addition to data bits, USART automatically inserts synchronization bits into the stream . tags, So called starting And stop bits. At During reception, these extra bits are removed. Typically, the start and stop bits separate one byte of information (8 bits), but there are USART implementations that allow the transmission of 5, 6, 7, 8, or 9 bits. The bits separated by the start and stop signals are the minimum transmission. USART allows the insertion of two stop bits during transmission to reduce the likelihood of receiver and transmitter desynchronization under heavy traffic conditions. The receiver ignores the second stop bit, perceiving it as a short pause on the line .

USART (Universal Synchronous-Asynchronous Receiver-Transmitter) is a serial A bus with a full-duplex mode of information exchange . Typically used for data exchange between digital devices, including between a microcontroller and a personal computer (PC) .

Also USART they call UART (Universal Asynchronous Receiver - Transmitter (universal asynchronous receiver-transmitter), as it was called until recently. The second abbreviation simply lacks the synchronization element . Before the advent of the synchronous UART, data exchange took place only over two wires, but then a synchronization bus was added to this interface.

Direct PC signals USART are not used, therefore, to connect the microcontroller and the PC, adapter chips are used, such as How RS 232-USART, RS485-USART, USB-USART, CAN-USART or others.

Figure 8 shows the functional diagram of the USART, organized in the Atmega328 microcontroller.

As can be seen from Figure 8, There are three main outputs used in USART - yes RxD , TxD and XCK.

RxD - This contact For reception information.

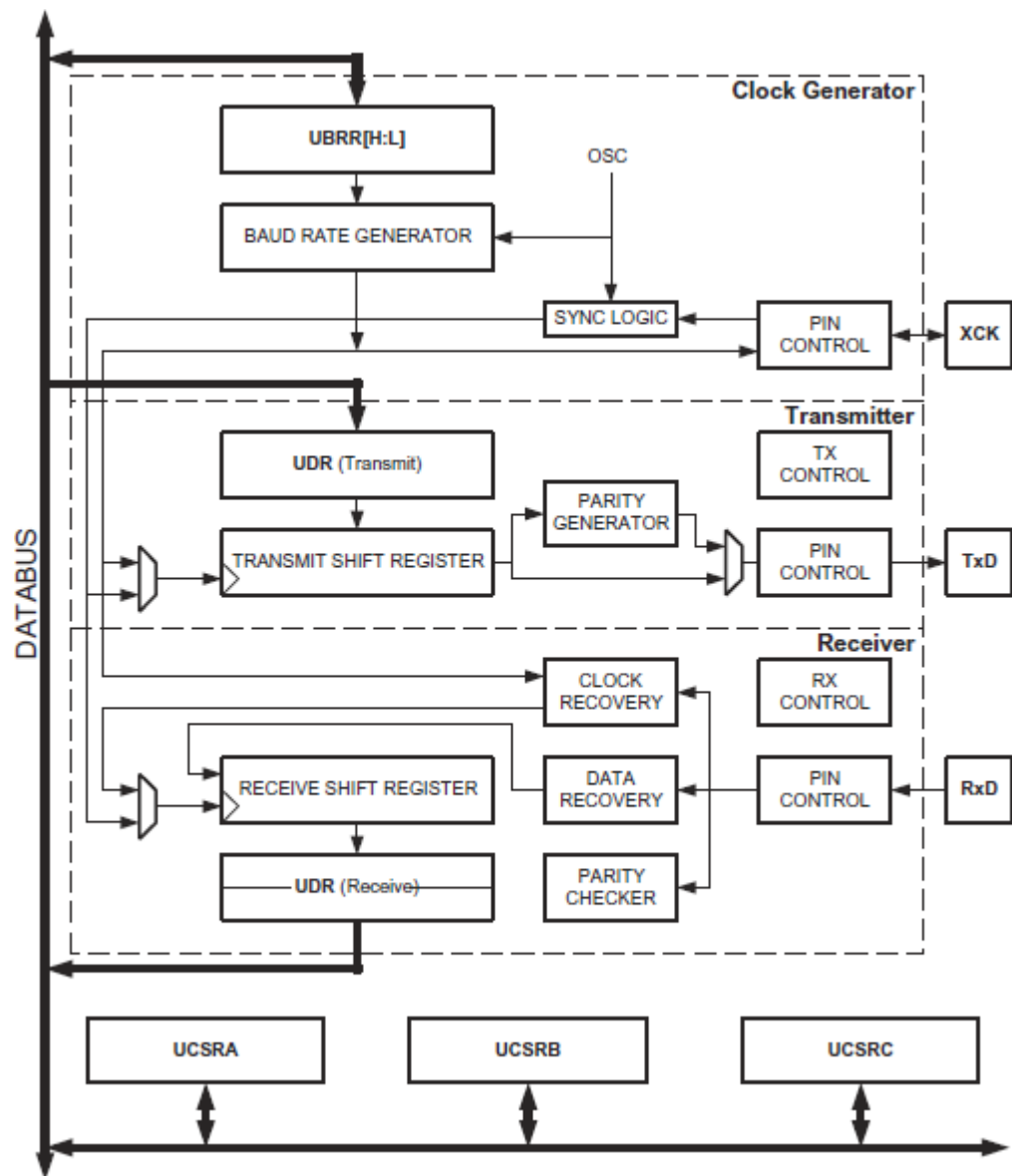
TxD – contact for transmitting information.

XCK – synchronization contact.

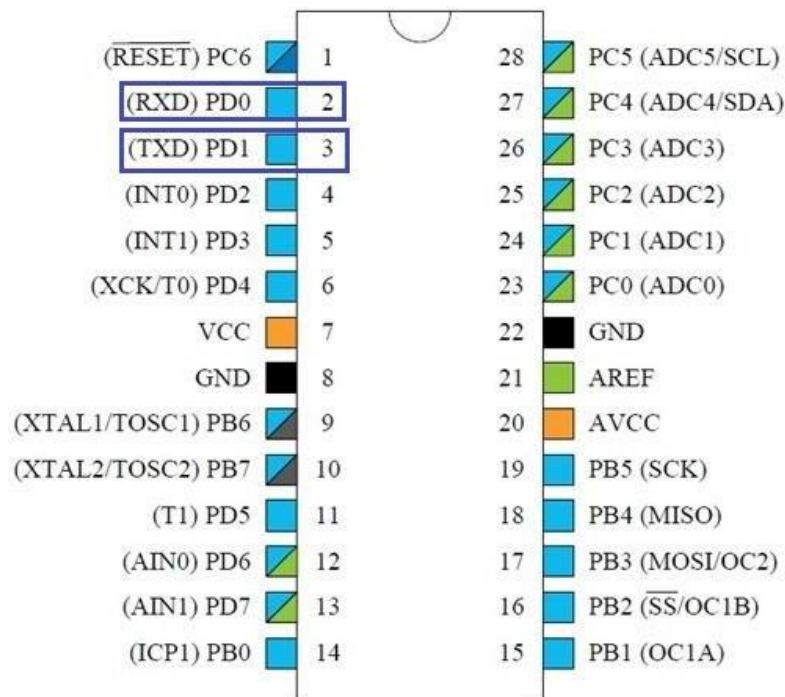
IN asynchronous mode conclusion HSC Can Not use.

There are also several modules here, each corresponding to its own output, as well as a number of control registers that are used to program (configure) the USART.

To implement the USART interface in a microcontroller Atmega 328 uses multi-function port pins D (PD0 – RXD, PD1 – TXD) (Figure 9).

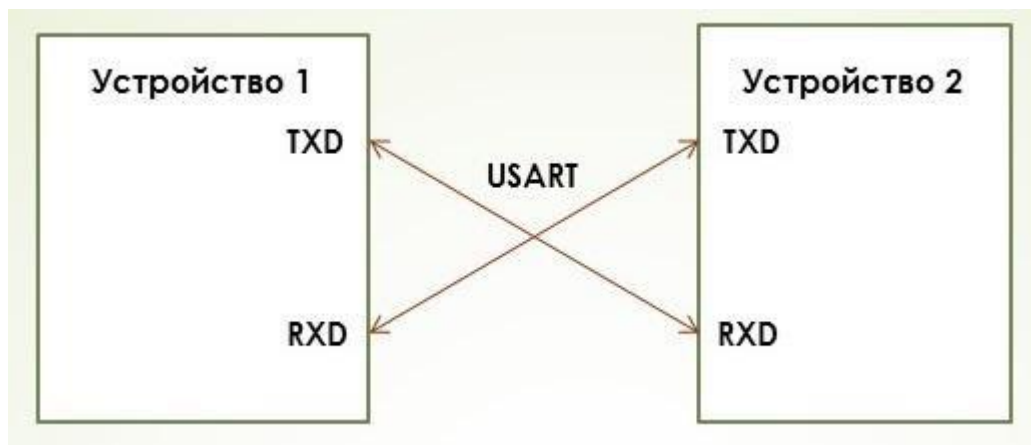


Drawing 8 - Functional scheme USART



Drawing 9 - Purpose conclusions integral microcircuits microcontroller Atmeg 328

Two devices communicate with each other via the USART interface by directly connecting the TxD pin of one device to the RxD pin of the other, and vice versa: connecting the RxD pin of one device to the TxD pin of the other. This means that data from the output of one device is fed to the input of the other, and vice versa (Figure 10).



Drawing 10 - Compound two devices With USART

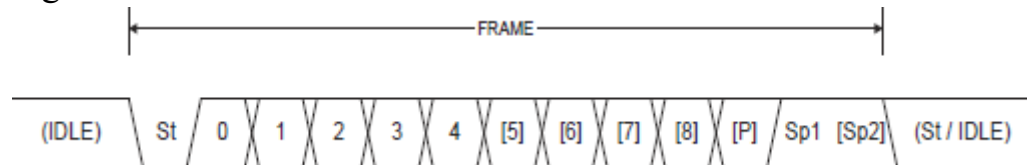
When working with the USART bus, interrupts can be used for three events:
 reception external data V buffer completed;
 USART data register is empty;
 broadcast data from buffer register Tx completed.

In the table 2 presented vectors And conditional designations interrupt data .

Table 2 - Vectors interruptions from USART

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete

The structure (format) of the data frame transmitted via the USART bus is shown in Figure 11.



Drawing 11 - Format frame USART

In idle mode, when the bus is not in use and no data is being transmitted or received on a particular pin, the line is high. A mandatory start bit appears at the beginning of the frame. (St). As we can see, it consists of a transition from a high state to a low one. The duration of one bit is determined by dividing one second by the speed set for transmitting and receiving data, which is measured in bits per second, or, as they are also called, in bauds. After the stop bit time has expired, the controller will consider the following bits as information bits. There can be from five to nine of them, depending on the mode. Generally, 8 is most often used, since it is most convenient to transmit data in bytes. The state of the information pin (TXD) during the transmission of a specific information bit will be dictated by the information bit itself. That is, if a one needs to be transmitted, then high; if a zero, then low. It is also important to note that the transmission of information bits begins with the least significant, and then the more significant.

The next bit, P, is the parity bit. If odd parity is not used, this bit is absent from the frame.

Then come the stop bits, indicating that the transmission of the packet is complete. There can be one or two of these bits, also depending on the mode. These bits are transmitted by organizing a high logic state on the pin. And then the process repeats from the beginning, that is, we transmit the next packet.

The procedure for calculating the parameters for setting the interface speed is presented in Table 3. The result obtained is entered into the UBRR register.

For example, for the Arduino board Uno c Atmega 328 microcontroller, the quartz resonator frequency is 16 MHz, the most common baud rate is 9600, accordingly, the register value UBRR=103.

Table 3 - Calculation parameter, defining speed work interface

Operating Mode	Equation for Calculating Baud Rate ⁽¹⁾	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{osc}}{16(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{osc}}{8(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{osc}}{2(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

Since we are not considering the synchronous mode, we are interested in the top two lines. Second column – this is the calculation of the speed depending on the clock frequency and the value of the UBRR register, and the third is the same equation, only transformed to calculate the required UBRR value depending on the clock frequency and the desired bus speed. Accordingly, the USART bus speed must be universal and adhere to certain standards. But since as a result of division, we cannot obtain a strictly defined speed, and we cannot write fractional values into the register, there is a certain percentage of errors. But, as a rule, these errors are not occurs due to the stop and start bits, at this time the correction takes place.

Now difference V table first And second lines. There is also bit U2X in a certain register, which, at zero, has no effect on the exchange rate, but at 1, it doubles the rate. Because of this, the denominator in the formula will no longer be 16, but 8.

Despite the correction due to stop and start bits, one should try to select the value of the UBRR register and the U2X bit so that the percentage is smaller.

Exchange data By consistent interface

The receiver and transmitter buffer registers are located one after the other. address spaces input/output And are designated How register Dan- nykh UDR (Universal Data Register). This register stores the lower 8 bits of received and transmitted data. When reading, an access to the buffer register is performed. UDR receiver, when writing - to the transmitter's buffer register. In USART modules The receiver buffer is a two-level (FIFO buffer), the state of which changes with any access to the register UDR. In this regard, the register should not be used. UDR as operands of read/modify/write (SBI and CBI) commands. In addition, care should be taken when using SBIC check commands. and SBIS, since They also change the state of the receiver buffer.

For management modules USART are used three register: UCSRA, UCSRB and UCSRC.

The description of the bits of the UCSRA, UCSRB and UCSRC registers is presented in Tables 4-6, respectively.

Table 4- Description discharges UCSRA register

Discharge	Name	Description
7	RXC	Receive completion flag. The flag is set to "1" if there are unread messages. data V buffer receiver (register data UDR). The flag is cleared by hardware after the buffer is empty (in UART) — after readings register data). If discharge RXCIE UCSRB registry installed, That at installation flag is generated request to the interrupt "reception complete"
6	TXC	Transmission completion flag. The flag is set to "1" after the transmission of all bits of the message from the transmitter's shift register, condition, What V register data UDR no new value was loaded. If the TXCIE bit of the UCSRB (UCSRnB) register is set, then when the flag is set is generated interrupt "broadcast "completed." Flag reset by hardware when executing an interrupt handling routine or by software by writing log 1 to it
5	UDRE	Data Register Empty Flag. This flag is set to "1" when the transmitter buffer is empty (after transferring a byte from the UDR data register to the transmitter shift register). When set , this flag means that a new value can be loaded into the data register. If the UDRIE bit of the UCR (UCSRB) register is set , a "data register empty" interrupt request is generated. The flag is cleared by hardware when writing to the data register.
4	FE	Framing Error Flag. The flag is set to "1" when a framing error is detected , i.e., if the first stop bit of the received message is "0". Flag resets upon reception stopbit , equal to "1"
3	DOR	Overflow flag. In USART, the flag is set to "1" if, when a new startbit is detected, the receiver shift register contains the last received word and the receiver buffer is full (two values). IN The UART flag is set to "1" if a new frame is placed in the receiver shift register before the previous word is read from the data register. The flag is cleared when received data is transferred from the receiver shift register to the buffer.
2	PE	Parity error flag. The flag is set to "1" if A parity error was detected in the data in the receiver buffer . When parity is disabled, this bit is permanently cleared to "0"

1	U2X	Doubling speed exchange. If this discharge installed V "1", the division factor of the baud rate controller prescaler is reduced from 16 to 8, thereby doubling the asynchronous communication speed over the serial channel. In the USART, the U2X bit is used only at asynchronous mode work. IN son -
		chronome mode He must be reset
0	MPCM	Multiprocessor exchange mode. The MPCM bit is used in multiprocessor exchange mode. If it is set to "1", the slave microcontroller waits for a frame containing an address . Frames that do not contain a device address are ignored.

Table 5 - Description discharges UCSRB registry

Discharge	Name	Description
7	RXCIE	Enable interrupt upon completion of reception. If this bit installed in "1", That at installation flag RXC register UCSRA generates a "reception complete" interrupt (if the I flag of the SREG register is set to "1")
6	TXCIE	Enables the transmission complete interrupt. If this bit is set to "1," setting the TXC flag in the UCSRA register generates a "transmission complete" interrupt. (if the I flag of the SREG register is set to "1")
5	UDRIE	Enable interrupt when clearing the UART data register. If this bit is set to "1", then when the UDRE flag is set in the UCSRA register a "data register empty" interrupt is generated (if the I flag of the SREG register is set to "1")
4	RXEN	Permission reception. When installing this category in "1" The operation of the USART/UART receiver is enabled and the operation of the RXD (RXDn) pin is redefined . When the RXEN bit is cleared, the operation of the receiver is prohibited, A his buffer is reset. Values the TXC, DOR/OR and FE flags become invalid
3	TXEN	Transmit Enable. Setting this bit to "1" enables the UART transmitter and overrides the TXD pin . If this bit is cleared to "0" during a transmission, the transmitter will not turn off until the data stored in the shift register and transmitter buffer has been transferred .
2	UCSZ2	Parcel format. This bit is used to specify the size of data words transmitted over the serial channel. In USART modules, it is used in conjunction with the UCSZ1:0 bits of the register . UCSRC. IN modules UART, If discharge CHR9 installed V "1" - 9-bit data is transmitted and received, if reset - 8-bit data

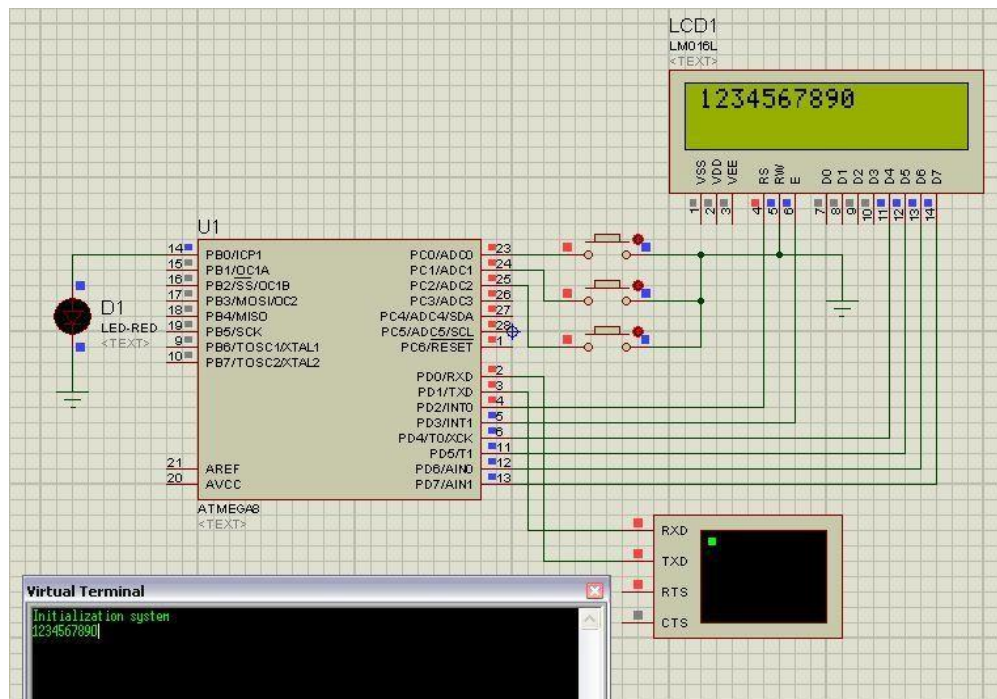
1	RXB8	8th bit of the received data. When using 9-bit data words, this bit contains the value of the most significant bit of the received word . In the case of USART, the contents of this bit must be read before reading the UDR data register.
0	TXB8	8th digit of the transmitted data. When using 9-bit data words, the contents of this category is the eldest bit of the word being transmitted. The desired value must be written to this bit before loading the data byte into the UDR register.

Table 6 - Description discharges UCSRC registry

Discharge	Name	Description
7	URSEL	Register Select. This bit determines which of the module's registers is written to. If the bit is set to "1" , the access is produced To register UCSRC. If same discharge reset V "0", appeal is produced To register UBRRH.
6	UMSEL	USART operating mode. If the bit is cleared to "0," the USART module operates in asynchronous mode. If the bit is set to "1," the USART module operates in synchronous mode.
5	UPM1	Operating mode of the parity check and generation circuit. These bits determine the functioning of the parity check and generation circuits .
4	UPM0	
3	USBS	Number of stop bits . This bit determines the number of stop bits sent by the transmitter. If the bit is cleared to "0", the transmitter sends 1 stop bit ; if it is set to "1", it sends 2 stop bits . The contents of this bit are irrelevant to the receiver.
2	UCSZ1	Parcel format. Together with the UCSZ2 digit, these bits determine the number of data bits in the messages (word size)
1	UCSZ0	
0	UCPOL	Clock signal polarity. The value of this bit determines the moment of data output and reading at the module's pins. This bit is used only when operating in synchronous mode. in asynchronous mode it should be reset to "0"

Below is an example of a test program for studying the USART protocol , where the Atmega 328 microcontroller exchanges information with the terminal, For clarity to controller LCD is connected 16X02 display. Pressing buttons 1-3 displays the corresponding lines in the terminal. Also, if you enter the characters "a" or "b" in the terminal, the LED connected to the controller's PB0 port will light or go out. Any characters entered in the terminal will also be displayed on the LCD display. For hardware testing, the microcontroller is connected to the computer via a MAX232 level converter chip. The standard Hyper Terminal program from Microsoft.

Connection diagram microcontroller and external devices shown in Figure 12.



Drawing 12 - Connection conclusions microcontroller

```
/* Example work With USART interface microcontrollers AVR */ #include
```

```
<avr / io. h>#include <avr / interrupt.h>
```

```
#define BAUDRATE 9600 // Data exchange rate #define F_CPU  
16000000UL // Working frequency controller
```

```
unsigned char NUM = 0;  
unsigned char count = 0;  
unsigned char byte_receive = 0;  
unsigned char i = 1;
```

```
// Function delays V ISS  
void _delay_us(unsigned char time_us)  
{ register unsigned char i;
```

```
for(i = 0; i < time_us; i++)  
{  
asm volatile( " PUSH R0 ");  
asm volatile(" POP R0 ");  
}  
}
```

```
// Function delays V ms  
void _delay_ms ( unsigned int time_ms )  
{ register unsigned int i;
```

```
for(i = 0; i < time_ms; i++)  
{  
_delay_us(250);  
_delay_us(250);
```

```

    _delay_us(250);
    _delay_us ( 250);
}
}

#define RS PD2
#define EN PD3

// Function transmissions void
lcd_com commands ( unsigned
char p)
{
    PORTD &= ~( 1 << RS); // RS = 0 (recording commands)
    PORTD |= (1 << EN ); // EN = 1 (start records teams V LCD) PORTD
    &= 0x0F; PORTD |= (p & 0xF0); // senior nibl
    _delay_us ( 100);
    PORTD &= ~( 1 << EN); // EN = 0 (end records teams in LCD)
    _delay_us ( 100);
    PORTD |= (1 << EN ); // EN = 1 (start records teams V LCD) PORTD
    &= 0x0F; PORTD |= (p << 4); // low nibble
    _delay_us ( 100);
    PORTD &= ~( 1 << EN); // EN = 0 (end records teams in LCD)
    _delay_us ( 100);
}

// Function transmissions void
data lcd_data ( unsigned char
p)
{
    PORTD |= (1 << RS )(1 << EN); // RS = 1 (data write), EN - 1 (start writing command to
    LCD)
    PORTD &= 0x0F; PORTD |= (p & 0xF0); // senior nibble
    _delay_us ( 100);
    PORTD &= ~( 1 << EN); // EN = 0 (end records teams in LCD)
    _delay_us ( 100);
    PORTD |= (1 << EN ); // EN = 1 (start records teams V LCD) PORTD
    &= 0x0F; PORTD |= (p << 4); // low nibble
    _delay_us ( 100);
    PORTD &= ~( 1 << EN); // EN = 0 (end records teams in LCD)
    _delay_us ( 100);
}

// Function initialization LCD void
lcd_init ( void )
{
    _delay_ms ( 50); // Expectation readiness LCD module

// Configuration four-digit mode PORTD |= (1 << PD 5);
PORTD &= ~( 1 << PD4);

// Activation four-digit PORTD mode |= (1 << EN );
PORTD &= ~( 1 << EN);

```

```

    _delay_ms ( 5);

    lcd_com (0x28); // 4-bit bus, LCD - 2 lines lcd_com
    (0x08); // complete shutdown display lcd_com
    (0x01); // clear the display
    _delay_us ( 100);
    lcd_com (0x06); // shift cursor to the right
    lcd_com (0x0C); // inclusion display, the cursor is not we see
}

// Function conclusion lines on LCD
void lcd_string ( unsigned char command, char *string)
{
    lcd_com (0x0C);
    lcd_com (command);
    while( *string != '\0')
    {
        lcd_data (*string);
        string++;
    }
}

// Function transmissions data By USART
void uart_send ( char data)
{
    while ( !(UCSRA & (1 << UDRE))); // We are waiting When will be cleansed buffer
    transmission UDR = data ; // Place data in the buffer, start transmission
}

// Function transmissions lines By USART
void str_uart_send ( char *string)
{
    while( *string != '\0')
    {
        uart_send (*string);
        string++;
    }
}

// Function reception data By USART
int uart_receive ( void )
{
    while ( !(UCSRA & (1 << RXC))); // We are waiting, When data will be received
    return UDR; // Let's read data from buffers And we return their at exit from subroutines
}

// Function initialization USART void
uart_init ( void )
{
    // Parameters connections: 8 bit data, 1 stop bit, No parity check
    // USART Receiver: Included
    // USART Transmitter: On

```

```

// USART Mode: Asynchronous
// USART Speed exchange: 9600

UBRRH = (F_CPU/BAUDRATE/16-1); // We calculate speed exchange data UBRRL =
(F_CPU/BAUDRATE/16-1) >> 8;

UCSRB |= (1 << RXCIE) | // We allow interrupt By completion reception data (1 <<
RXEN) | (1 << TXEN); // Turn on the receiver and transmitter
UCSRC |= (1 << URSEL) | // For access To register UCSRC we exhibit bit URSEL (1 <<
UCSZ1) | (1 << UCSZ0); // The size of the message in the frame is 8 bits
}

// Interrupt By ending reception data By USART ISR( USART_RXC_vect )
{
NUM = UDR; // We accept symbol By USART
byte_receive = 1;
uart_send ( NUM); // We are sending symbol via USART

if ( NUM == 'a') // If accepted symbol "a", turn on LED PORTB |= (1
<< PB0);
if ( NUM == 'b') // If accepted symbol "b", turn off LED PORTB &=
~(1 << PB0);
}

// Home int function
main ( void )
{
DDRB |= (1 << PB 0); // LED
PORTB = 0x00;

DDRC =
0x00; PORTC =
0xFF;

DDRD =
0b11111110; PORTD =
0x00;

lcd_init ( ) ; // Initialize LCD uart_init (); //
Initialization USART

sei(); // Globally we allow interruptions

str_uart_send ( " Initialization system\r\n"); // We are transmitting line By
USART lcd_string (0x80, " AVR USART TEST "); // Output line on LCD
_delay_ms ( 2500); lcd_com
(0x01); // We clean LCD

while( 1)
{

if ( ( PINC & (1 << PC 0)) == 0) // If pressed button 1

```

{

```

while( (PINC & (1 << PC0)) == 0){} // Wait release buttons 1 str_uart_send
("Button 1 TEST\r\n"); // We are transmitting line By USART lcd_string
(0x80, "Button 1 TEST"); // Output line on LCD
_delay_ms ( 1000); lcd_com
(0x01); // We clean LCD
}

if ( (PINC & (1 << PC1)) == 0) // If pressed button 2
{
while ((PINC & (1 << PC1)) == 0){} // Wait for button 2 to be released
str_uart_send (" Button 2 TEST\r\n"); // We are transmitting line By USART
lcd_string (0x80, " Button 2 TEST"); // Display the string on the LCD
_delay_ms ( 1000); lcd_com
(0x01); // We clean LCD
}

if ( (PINC & (1 << PC2)) == 0) // If pressed button 3
{
while ((PINC & (1 << PC2)) == 0){} // Wait for button 3 to be released
str_uart_send (" Button 3 TEST\r\n"); // We are transmitting line By USART
lcd_string (0x80, " Button 3 TEST"); // Display the string on the LCD
_delay_ms(1000);
lcd_com(0x01); // Очищаем LCD
}

if(byte_receive)
{
byte_receive = 0;
count++;
lcd_data ( NUM) ; // We are deducing symbol on
LCD if ( count > 16) // If the line is full
{
count = 0;
lcd_com (0x01); // We clean LCD
}
}
}
}
}
}

```

Exercise For independent work

Run all example programs. Make changes to the program listings in accordance with the teacher's instructions. Provide screenshots of the screen with the program code and the results of the work in the report. Draw conclusions about the technology of programming the means of displaying the results of the program using LCD and USART.

PRACTICAL LESSON No. 8

PROGRAMMING PLC IN WEDNESDAY CODESYS

Objective of the work : To acquire programming skills in the environment CoDeSys programmable logic controllers

Educational questions

1. Development programs management traffic light
2. Debugging programs V mode emulations
3. Visualization feasible programs

1 Development programs management traffic light

This lab work assumes creating a program for A simple traffic control unit at an intersection with traffic lights for two intersecting directions of traffic. It's clear that the traffic lights should have two opposing states—red and green. To avoid accidents, we'll add the standard transition stages: yellow and yellow-red. The latter stage should be longer than the previous one.

This lab will demonstrate how time - controlled processes can be represented using IEC 61131-3 languages, how different languages can be combined in CoDeSys , and how to use modeling tools. CoDeSys .

Order execution work.

Launch CoDeSys And select 'File' ' *Create*' ("*File*" "*New*"). In In the dialog box , we define the first **software component (POU – Program Organization Unit)** . By default, it is named PLC_PRG. Do not change it. The type of this POU must definitely be program. **Each** project must have a program with this name. We will select Continuous as the programming language for this POU . Function Chart (CFC). Now create three more objects. Use the '*Project*' '*Object - Add*' *command* . *Add* ") in the system or context (right-click in the Object Organizer) menu. Create: a program in the Sequential language Function Chart (SFC) named SEQUENCE, a function block in the Function language Block Diagram (FBD) named TRAFFICSIGNAL and another similar block - WAIT, which we will describe in the Instruction List (IL) language.

IN POU TRAFFICSIGNAL we will compare certain stages process -
sa appropriate flowers. That There is We let's make sure What red light

lit in the red stage and in the yellow-red stage, yellow light in the yellow and yellow-red stage, etc.

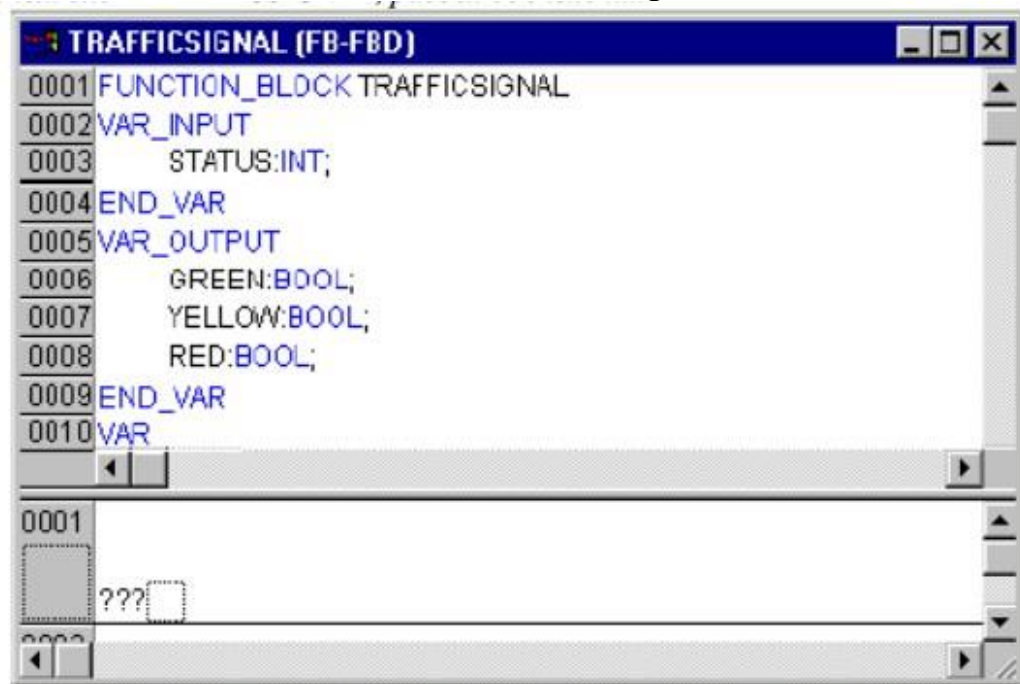
In WAIT We let's create simple timer, which on entrance receives the stage length in milliseconds and returns the TRUE state at the output after the specified period of time.

In SEQUENCE, everything will be combined so that the right lights come on at the right time and for the right period of time.

An input signal is entered into PLC_PRG to enable the traffic light to start working , and the 'color commands' of each lamp are connected to the corresponding outputs of the equipment.

"TRAFFICSIGNAL" ads. In the ad editor, define an input variable (between the VAR_INPUT and END_VAR keywords) named STATUS of type INT. STATUS will have four possible states , defining the corresponding stages - green, yellow , yellow-red, and red.

Since our TRAFFICSIGNAL block has three outputs, we need to define three more variables: RED, YELLOW, and GREEN. The declaration section of the TRAFFICSIGNAL block should now look like Figure 1.

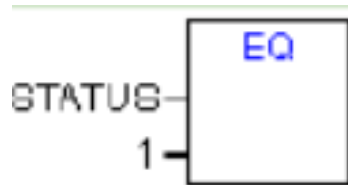


Drawing 1 - Functional block TRAFFICSIGNAL, chapter announcements

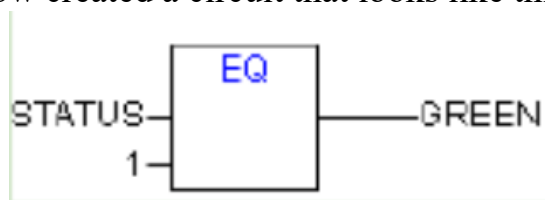
Programming "TRAFFICSIGNAL". You need to describe what connects the STATUS input to the output variables. To do this, go to the POU code section (body). Click the box to the left of the first chain (the gray area numbered 1). You've now selected the first chain. Now, select the 'Insert' menu item, then 'Element' (" Insert " " Box "). A rectangle with an AND operator and two inputs will be inserted into the first chain:



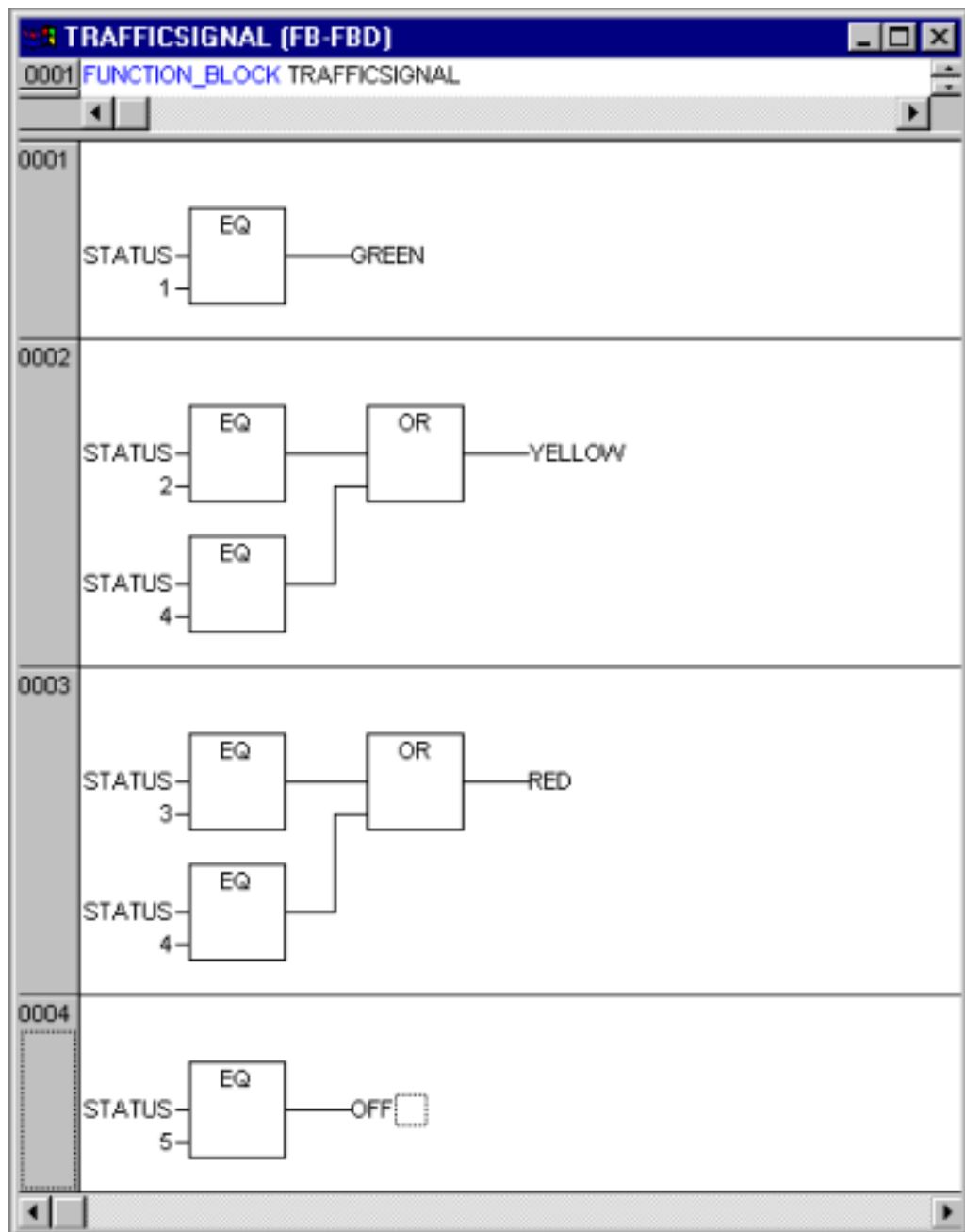
Click the text "AND" and replace it with "EQ." Replace the three question marks next to the top of the two inputs with the variable name "STATUS." For the bottom input, replace the three question marks with "1." The result should look like this:



Now click behind the EQ rectangle. The EQ output is now selected. Go to the menu and select "Insert" and "Assign ". Change the three ??? questions to GREEN. You have now created a circuit that looks like this:



STATUS is compared to 1, and the result is assigned to GREEN. Thus , GREEN will be on when STATUS is 1. For the other TRAFFICSIGNAL colors, we need two more networks. Create them with the 'Insert' 'Network (after)' command . *The completed POU should look like Figure 2 .*



Drawing 2- Functional block TRAFFICSIGNAL

select the input itself, not the text (highlighted by a rectangle). Next, use the 'Insert' 'Element' command ('Insert' 'Box'). Now our first POU is complete. As planned, TRAFFICSIGNAL will control the outputs based on the value of the STATUS variable.

Including standard.lib To create a timer in the WAIT POU, we need a POU from the standard library. To do this, open the library manager using the 'Window' 'Library Manager' command. Manager "). Select 'Insert' 'Add library' ("Insert" "Additional library"). A file selection dialog box should open. Select standard.lib from the list of libraries.

"WAIT" Declarations. Now let's return to the WAIT POU. This POU was supposed to function as a timer, setting duration of the TRAFFICSIGNAL stages. Our POU must have an input variable of type TIME and generate a binary (Boolean) output a variable we'll call OK. This variable should be set to TRUE when the desired time period is over. We first set this one variable in meaning FALSE V at the end lines declarations (but before the semicolon) ":= FALSE".

Now we need the POU TP time generator. It has two inputs (IN, PT) and two outputs (Q, ET). TP does the following : While IN is FALSE, ET will be 0 and Q will be FALSE. As soon as IN switches to TRUE, the ET output will start counting time in milliseconds. When ET reaches the specified PT value, counting will stop. Meanwhile, the Q output is TRUE as long as ET is less than PT. Once ET reaches the PT value, the Q output will switch back to FALSE.

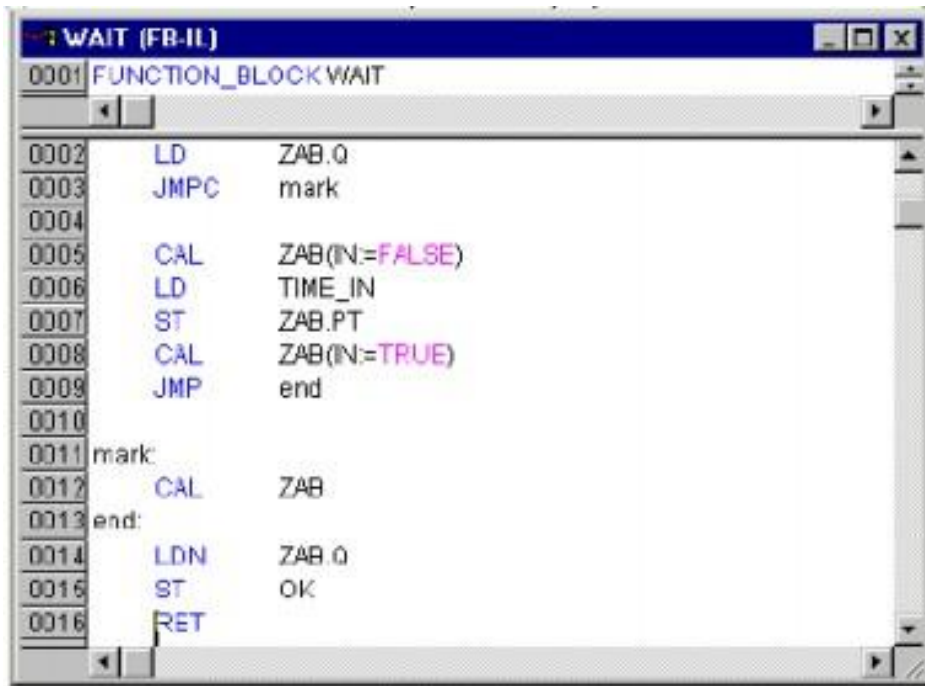
A description of all POUs from the standard library is given in the recommendations for working in the CoDeSys environment .

To use TP in the WAIT POU, we must create a local instance of it . To do this , we declare a local variable ZAB (counted time) of type TP (between the VAR, END_VAR keywords). The WAIT declaration section should now look like Figure 3.



Drawing 3 - Programming "WAIT"

To create the desired timer, the program text should be as shown in Figure 4



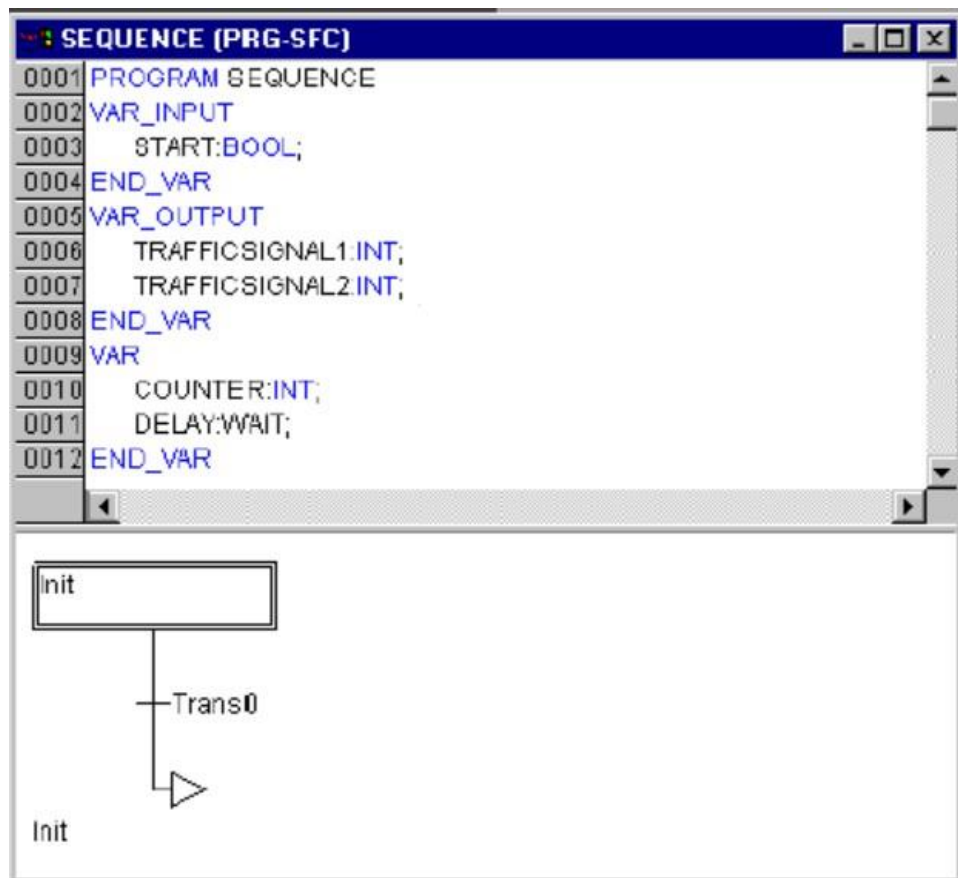
Drawing 4

First we check if Q is set to TRUE (maybe the counting has already started), in this case we do not touch the ZAB settings, but call the function block ZAB without input variables - to check, finished whether the time period is specified. Otherwise, we set the IN ZAB variable to FALSE and simultaneously set ET to 0 and Q to FALSE. Thus, all variables are set to their initial state.

Now we set the desired time TIME to the variable PT and call ZAB with IN:= TRUE. The ZAB function block will now run until it reaches the value TIME and sets Q to FALSE. The inverted value of Q will be stored in the variable OK after each execution of WAIT. As soon as Q becomes FALSE, OK will become TRUE.

The timer is complete. Now it's time to combine our two blocks, WAIT and TRAFFICSIGNAL, in the main program, PLC_PRG.

SEQUENCE version 1 . At first we will announce necessary variables. These are the input variable START of the BOOL type, two output variables TRAFFICSIGNAL1 and TRAFFICSIGNAL2 of the INT type, and one variable DELAY of the WAIT type. The SEQUENCE program now looks like this (Figure 5).



Drawing 5- Program SEQUENCE version 1, chapter announcements

Let's create an SFC diagram. Initially, the SFC graph always consists of the " Init " stage, the "Trans0" transition, and a return back to Init . Naturally, we'll have to supplement it a little.

Before programming specific stages and transitions, let's build the graph structure. First, we need stages for each stage of TRAFFICSIGNAL. Insert them, marking Trans0 and choosing commands 'Insert' 'Step-transition (bottom) ' (" *Insert " " Step transition (after)"* . Repeat this procedure three more times.

To edit the name of a transition or stage, simply click the mouse on the desired text. Name the first transition after Init "START", and all other transitions "DELAY.OK". The first transition is enabled when START is set to TRUE, and all others are enabled when DELAY in OK becomes TRUE, i.e., when the specified period ends. Stages (from top) down) are named Switch1, Green2, Switch2, Green1, and Init , of course, will retain its name. " Switch " should include the yellow phase, in Green1 TRAFFICSIGNAL1 will be green, in Green2 TRAFFICSIGNAL2 will be green . Finally, change the return address of Init to Switch1. If you did everything correctly, the diagram should look like the one shown in Figure 6.

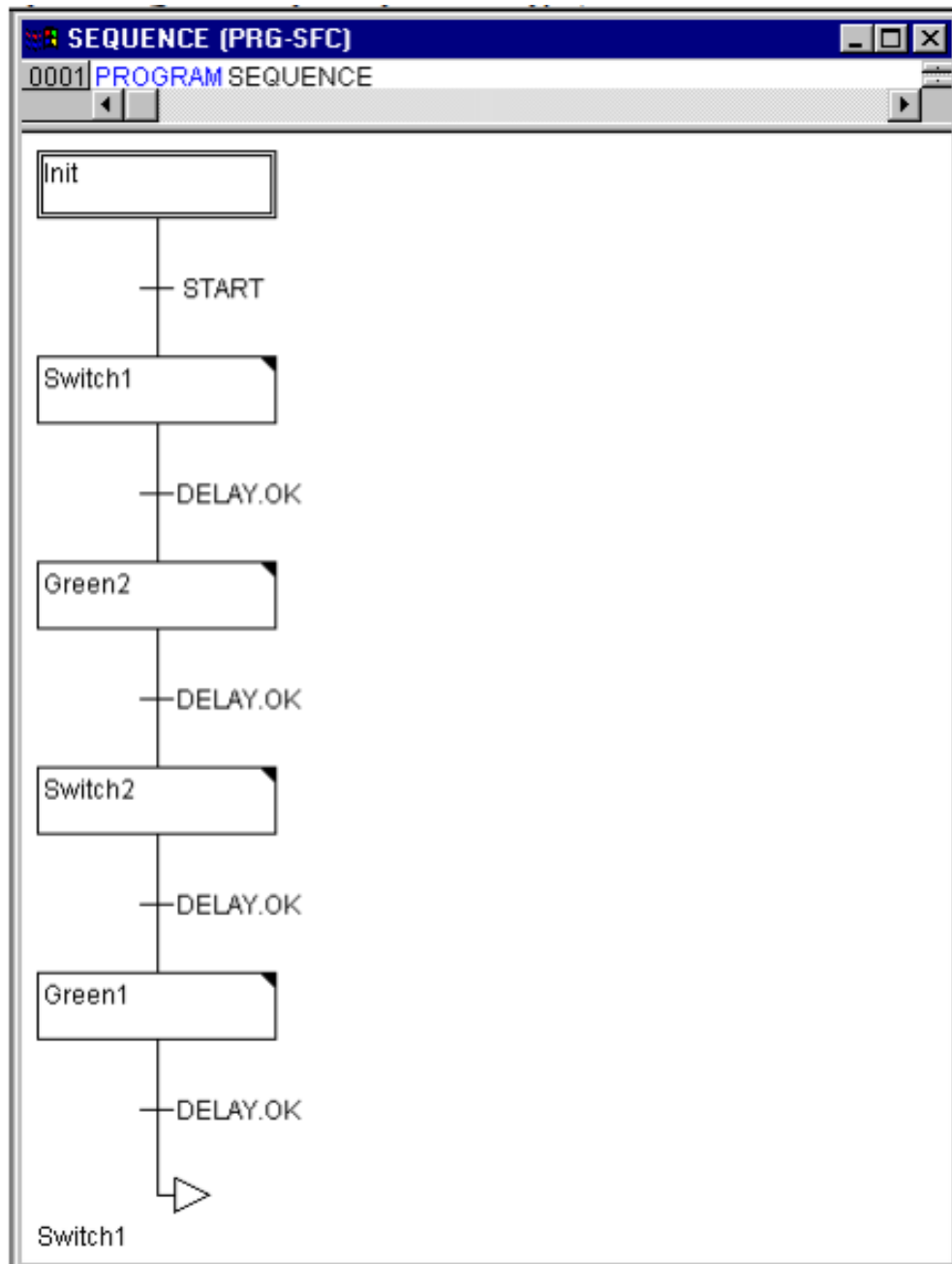


Figure 6 - SEQUENCE program version 1, instructions section Now We

should program stages. If You you will do double -

If you click the mouse on the step image, a dialog for defining a new action should open. In our case, we'll use IL (Instruction List) language.

Programming stages and transitions . During the Init stage , we check whether the START signal is active or not. If the signal is inactive , the traffic light is turned off. This can be achieved by assigning the number 5 to the TRAFFICSIGNAL1 and TRAFFICSIGNAL2 variables (Figure 7).

Action Init (IL)		
0001	LD	1
0002	ST	TRAFFICSIGNAL1
0003	LD	3
0004	ST	TRAFFICSIGNAL2

Drawing 7- Stage Init

In Green1, TRAFFICSIGNAL1 will be green (STATUS:= 1), TRAFFICSIGNAL2 will be red (STATUS:=3), and the delay will be 5000 milliseconds . Stage Green1:

Action Green1 (IL)		
0001	LD	1
0002	ST	TRAFFICSIGNAL1
0003	LD	3
0004	ST	TRAFFICSIGNAL2
0005	CAL	DELAY(TIME_IN:=t#5s)

Switch1 changes the state of TRAFFICSIGNAL1 to 2 (yellow) and, accordingly , TRAFFICSIGNAL2 to 4 (yellow-red). In addition, a delay of 2000 milliseconds is now set. The Switch1 stage looks like this:

Action Switch1		
0001	LD	2
0002	ST	TRAFFICSIGNAL1
0003	LD	4
0004	ST	TRAFFICSIGNAL2
0005	CAL	DELAY(TIME_IN:=t#2s)

Green2 includes red in TRAFFICSIGNAL1 (STATUS:= 3) and green in TRAFFICSIGNAL2 (STATUS:=1). The delay is set to 5000 milliseconds. Green2 stage:

Action Green2		
0001	LD	3
0002	ST	TRAFFICSIGNAL1
0003	LD	1
0004	ST	TRAFFICSIGNAL2
0005	CAL	DELAY(TIME_IN:=t#5s)

In Switch2, the STATUS of TRAFFICSIGNAL1 changes to 4 (yellow-red), and TRAFFICSIGNAL2 will now be 2 (yellow). The delay should now be 2000 milliseconds.

Stage Switch2:



First version our programs finished.

2 Debugging programs V mode emulations

To test the program in emulation mode, do the following : Open the PLC_PRG POU. Every project starts with PLC_PRG. Insert a component into it and replace "AND" with "SEQUENCE." Leave the inputs and outputs free for now.

Compile the project ('Project' 'Compile' - 'Project' 'Build') and check for absence errors. IN window messages you should see the text: "0 Errors , 0 Warnings ". Now enable the 'Online' 'Simulation mode' checkbox . mode') and give the command 'Online' 'Connect' ('Online' 'Login').

Launch the program 'Online' 'Start' ('Online' 'Run'). Open the program SEQUENCE.

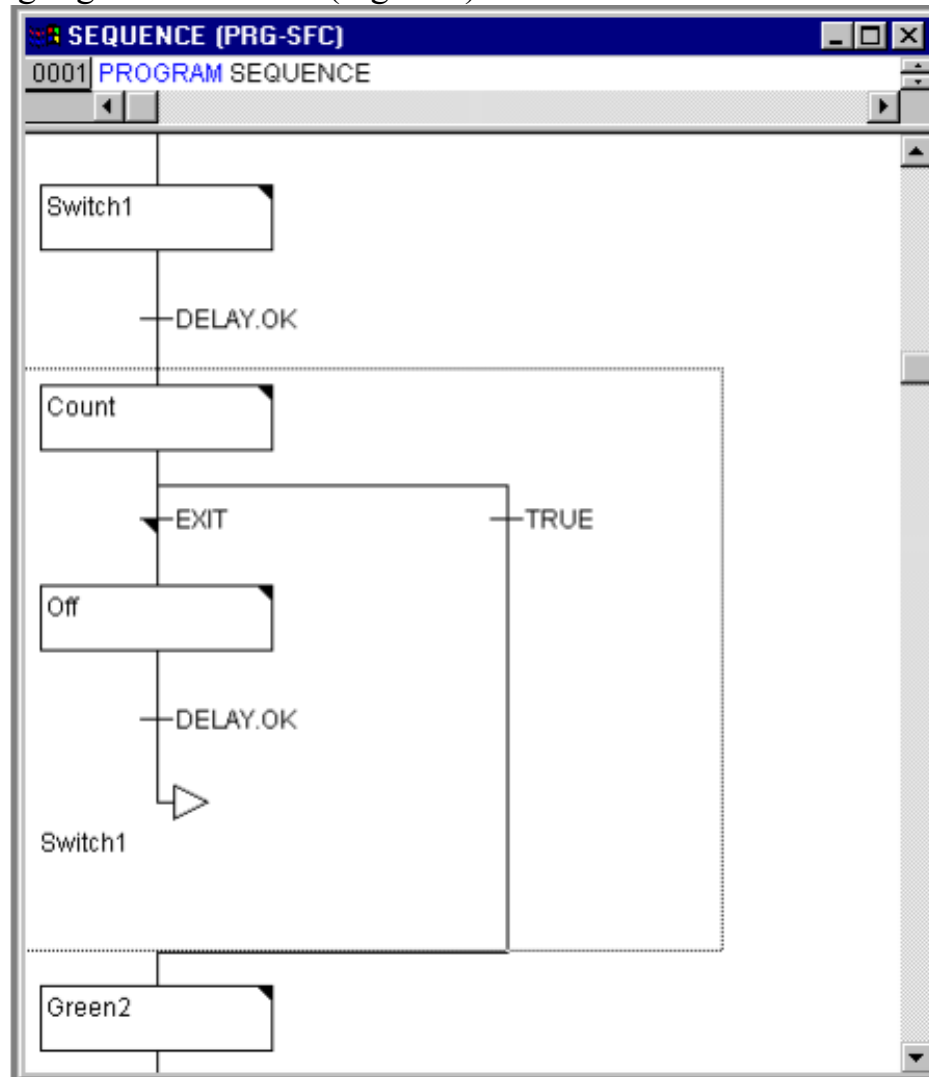
The program is running, but it doesn't work because the START variable must be set to TRUE. PLC_PRG will do this later, but for now you can change it manually. To do this, double-click on the variable declaration. Its value is now highlighted and equals TRUE. Issue the command to write the variable values ('Online' 'Write values' - 'Online' 'Write' values').

Now you can watch program operation. Active steps in the diagram are highlighted in blue.

To continue editing the program, close the online mode using the 'Online' 'Logout' command .

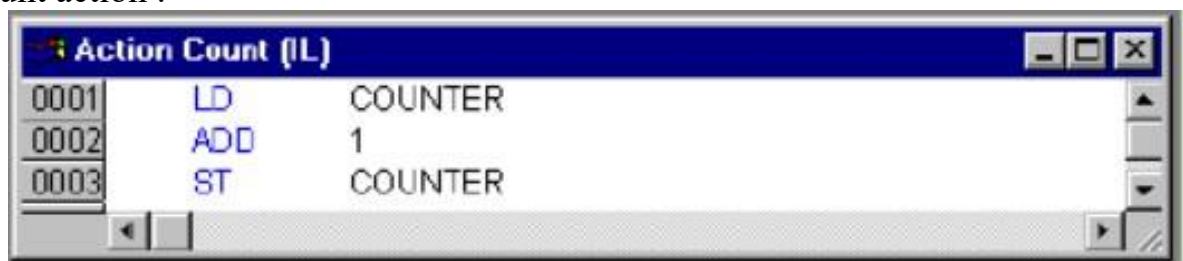
"SEQUENCE" second version . Now let's complicate our program a little . It would be reasonable to turn off our traffic lights at night. To do this, we will create a counter in the program that will turn off the device after a certain number of TRAFFICSIGNAL cycles. First, we need a new variable COUNTER of type INT. Declare it as usual in the SEQUENCE declaration section. Now select the transition after Switch1 and insert another stage and transition. Select the resulting transition and insert an alternative branch to the right. After the left transition, insert an additional

stage And transition. After new resulting transition insert jump to Init . Name the new parts as follows: the upper of the two new stages should be called " Count " and the lower " Off ". The transitions will be named (from top to bottom , left to right) EXIT, TRUE, and DELAY.OK. The new parts should now look like the fragment highlighted in the box (Figure 7).



Drawing 7- Program "SEQUENCE", chapter instructions

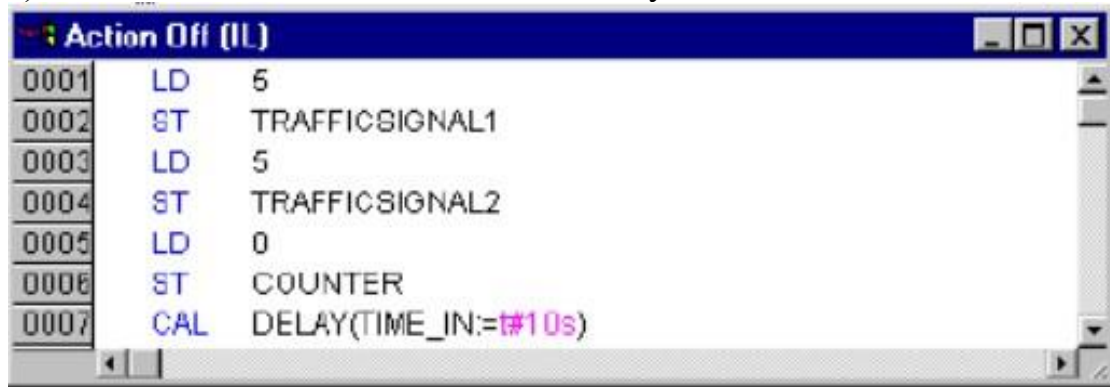
Now the two new stages and transitions need to be filled with content . At the Count stage , only one action is performed - COUNTER is incremented by 1: Count action :



At the EXIT1 transition, the counter is checked to see if it has reached a specified value , for example 7: EXIT transition:



In the Off stage , the state of both traffic lights is set to 5 (traffic light is off), COUNTER is reset to 0 and a time delay of 10 seconds is set. Action Off :



In our hypothetical situation, night falls after seven TRAFFICSIGNAL cycles. The traffic lights turn off completely until dawn, and the process repeats . If you wish, you can test the program's operation in the emulator again before continuing to improve it.

PLC_PRG . We have defined two strictly time-correlated traffic lights in the SEQUENCE block. Now we will complete the program. To do this, we need to distribute the input and output variables in the PLC_PRG block. We want to be able to start the system with the IN switch and we want to ensure that all six lamps (2 traffic lights) are switched by transmitting " switching commands " at each step of the SEQUENCE. Now we declare corresponding Boolean variables For everyone six outputs And one input, then we will create a program and map the variables to the corresponding IEC addresses.

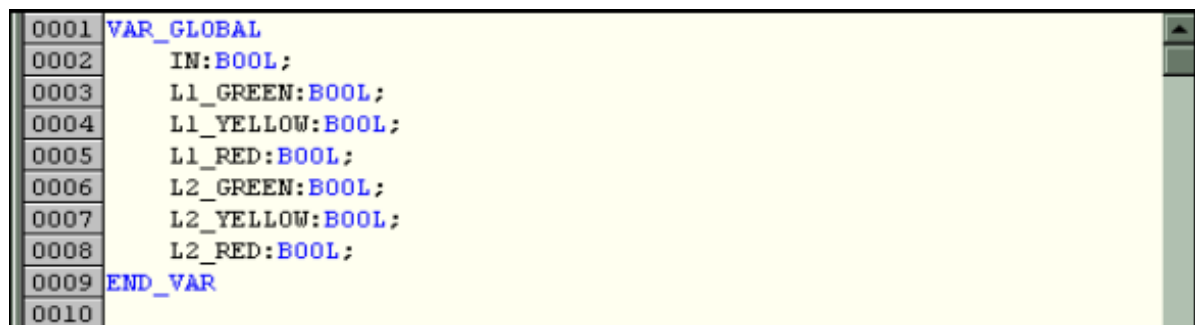
The next step is to declare the variables LIGHT1 and LIGHT2 of type TRAFFICSIGNAL in the declaration editor.

Announcement LIGHT1 And LIGHT2:



Boolean variables . However, we won't declare them in the declaration section of the PLC_PRG block; instead, we'll use global variables (Global Variables) from the Resources (Resources). The binary input variable IN, needed to set the START variable of the SEQUENCE block to TRUE, will be defined in the same way. Select the 'Resources' tab and open the 'Global Variables' list (Global Variables). Declaring global variables :

Let's finish PLC_PRG. To do this, we'll go to the editor window. We've

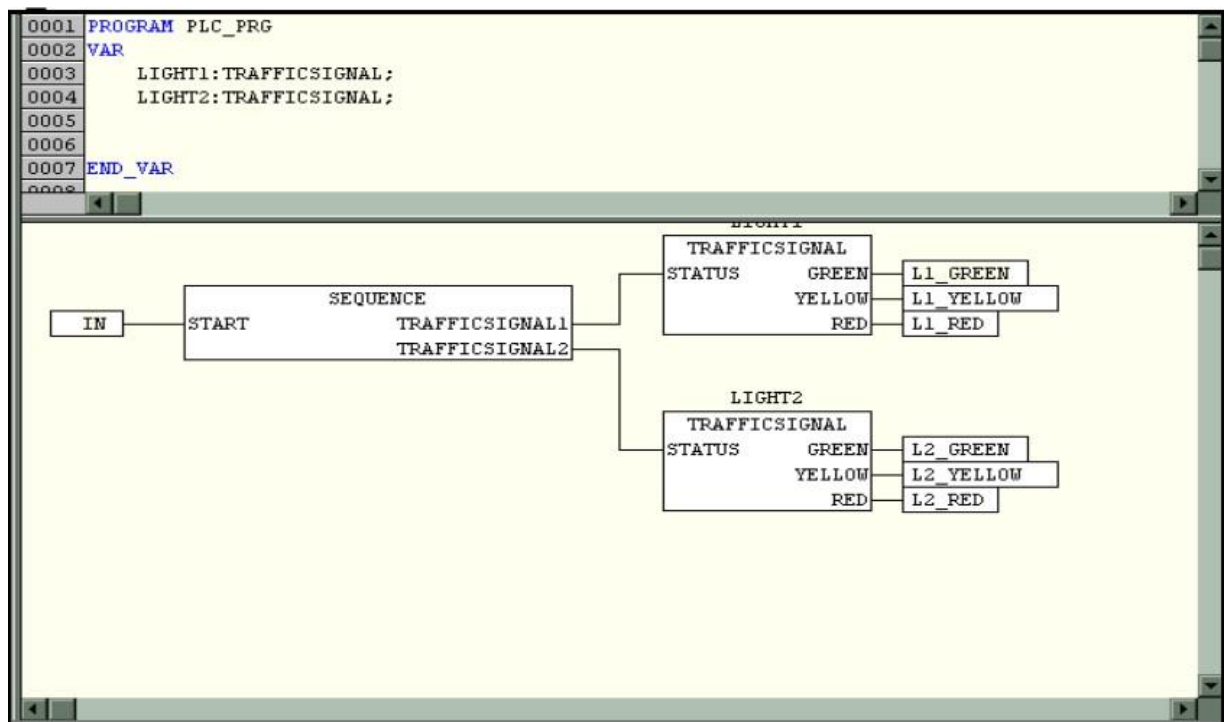


selected the Continuous Function Chart (CFC) editor, and therefore the corresponding toolbar is available to us.

Right-click in the editor window and select the 'Box' *element* . Click on the AND text and type 'SEQUENCE ' . The element will automatically be converted to a SEQUENCE with the input and output variables already defined. Next, insert two elements and name them TRAFFICSIGNAL .

TRAFFICSIGNAL is a function block, and as usual, you will get three red question marks, which need to be replaced with the already declared local variables LIGHT1 and LIGHT2. Now create an element of type Input , which will be called IN and six elements of type Output , which should be given the following names: L1_green, L1_yellow, L1_red, L2_green, L2_yellow, L2_red. All program elements are now in place, and you can connect the inputs and outputs. To do this, click on the short line. input/output And pull her (Not letting go key mice) To input/output of the desired element. Finally, your program should look like the one shown below.

PLC_PRG:

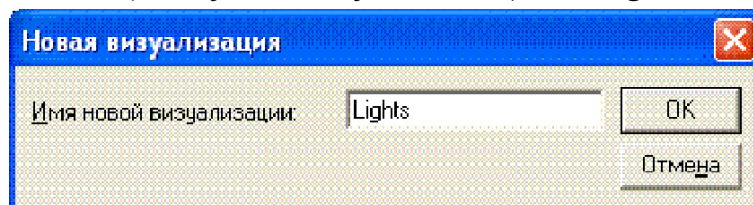


Now our program fully ready.

TRAFFICSIGNAL emulation . Now test your program in emulation mode. Ensure it works correctly by monitoring variable values and execution sequence in the CoDeSys editor windows .

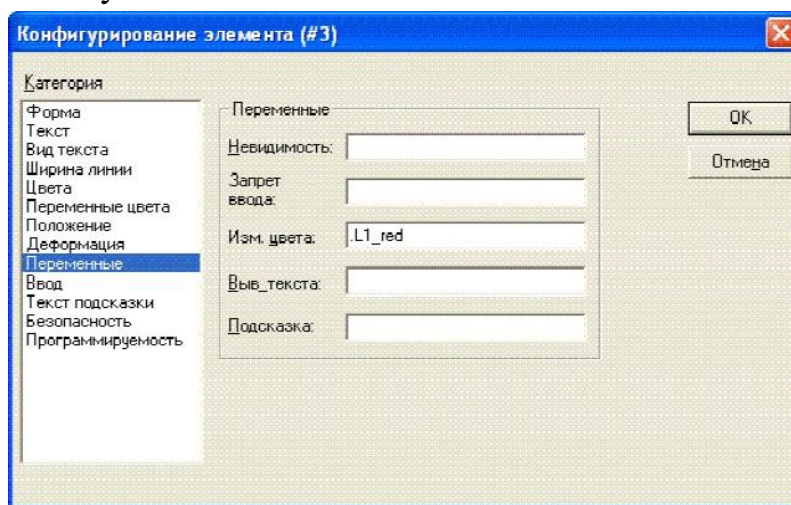
3 Visualization feasible programs

Visualization allows you to quickly and easily bring project variables to life. Complete description visualizations You you will find in [1]. Now We Let's draw two traffic lights and their switch, which will allow us to turn the traffic light control unit on and off. Creating a new visualization To create a visualization, select the 'Visualizations' *tab in the* organizer objects. Now do it command 'Project' 'Object - Add' (' *Project* ' ' *Object Add* '). Dialog for creating a new visualization :



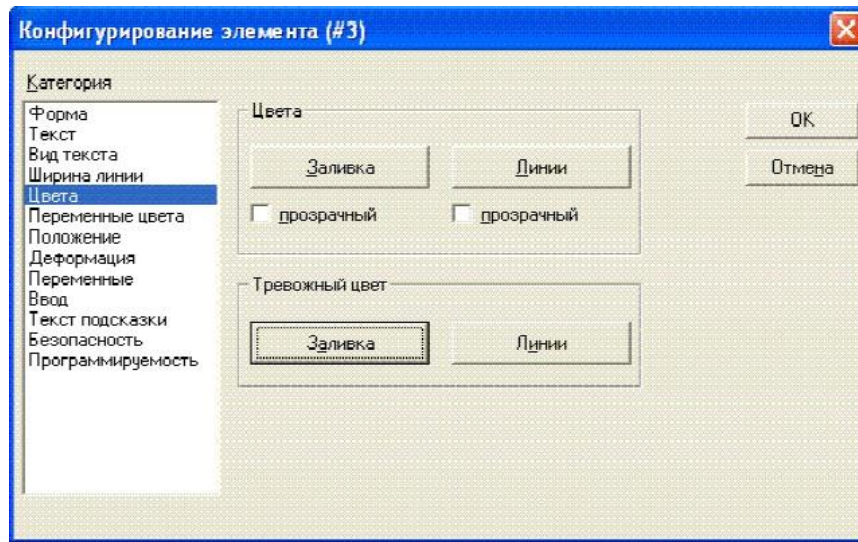
Enter any name for the visualization, for example, Lights . When you click the Ok button, a window will open in which you will create the visualization .

Inserting an element into a visualization. To create a traffic light visualization , follow these steps: Select the 'Insert' 'Ellipse' command *and draw a* circle with a diameter of about 2 centimeters . To do this , click the mouse on the working area and, holding down the left mouse button, stretch the resulting circle to the desired size. Double-click the circle. A dialog box for configuring the visualization element will appear. Select the 'Variables' category . And V field 'Change colors' (Change color) enter Name variable .L 1_red. It's convenient to enter the variable name using the Input Assistant. (Input Assistant) (<F1> key) (Figure 8). The global variable L1_red will control the color of the circle you draw.



Drawing 8

Select the 'Colors' category . In the 'Color' area, click the 'Inside ' button and in the window that appears, select any neutral color, such as black. Click the 'Fill' button (Inside) in the 'Alarm color' area (Alarm Color) and select red color (Figure 9).



Drawing 9

The resulting circle will be black when the variable's value is false and red when the variable is true. Thus, we have created the first light of the first traffic light.

The rest of the traffic light colors. Now call the 'Edit' 'Copy' command (< Ctrl > + < C >) and execute the 'Edit' 'Paste' command (< Ctrl > + < V >) twice . You will get **two new circles** . You can move these circles with the mouse. Arrange them so that they form a vertical row on the left side of the editor window. Double-clicking on a circle opens a window for setting the properties of the visualization element. In the 'Change colors' field Color) of the 'Variables' dialog box of the properties of the corresponding circles , enter the following variables:

For average circles : .L 1_yellow for
the bottom circle: .L1_green

In the 'Colors' category, in the ' Alarm color' area, color) set the colors of the circles (yellow and green).

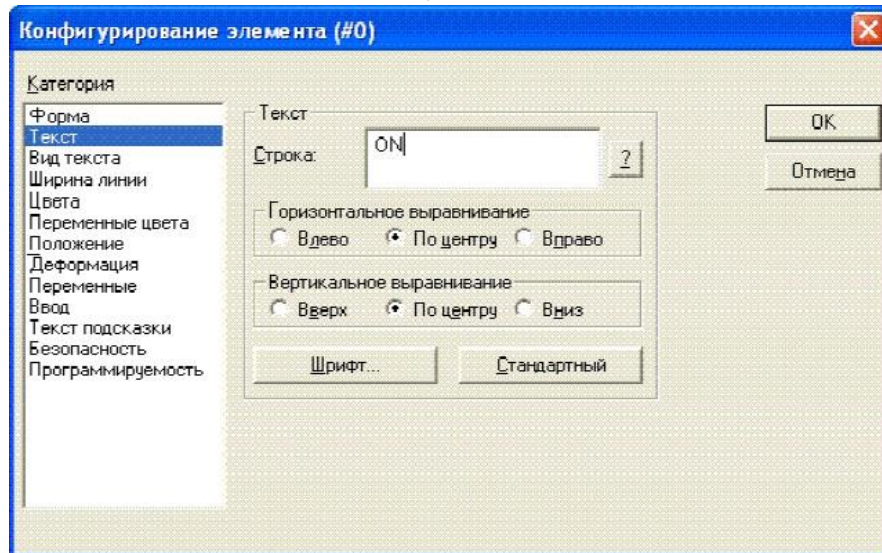
Traffic light body. Now call the ' Insert ' ' Rectangle - nickname' command and insert a rectangle so that the previously entered circles are inside it. Select the color of the rectangle and then execute the 'Extras' 'Send to back ' command . to back ”), which will move it to the back. After this, the circles will be visible again . Activate the emulation mode by executing the command 'Online' 'Emulation mode' – ' Online ' “ Simulation mode ” (the emulation mode is active if there is a check mark in front of the 'Emulation mode' item). Launch the program by executing the commands 'Online' 'Connect' (' Online ' ' Login ') and 'Online' 'Start' (' Online ' ' Run ') and you will see how the colors of the traffic light will change.

Second traffic light. The easiest way to create a second traffic light is to copy all the elements of the first. Select the elements of the first traffic light and

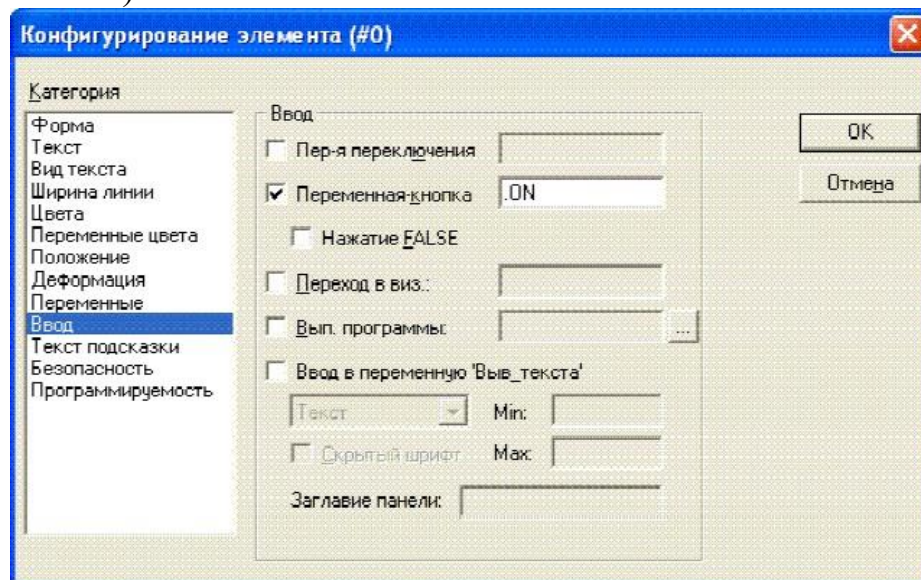
Copy them by using the commands 'Edit' 'Copy' (“ Edit ” “ Copy ”) and 'Edit' 'Paste' (“ Edit ” “ Paste ”). Replace the names of the variables that control the colors (for example , .L 1_red to .L2_red), and the second traffic light will be ready

ON switch. As described above, insert a rectangle, set its color, and enter the .ON variable in the 'Change Color' field of the 'Variables' category . In the 'Content' field of the ' Text ' category , enter the name "ON."

To switch the ON variable when you click the mouse on this element, in the

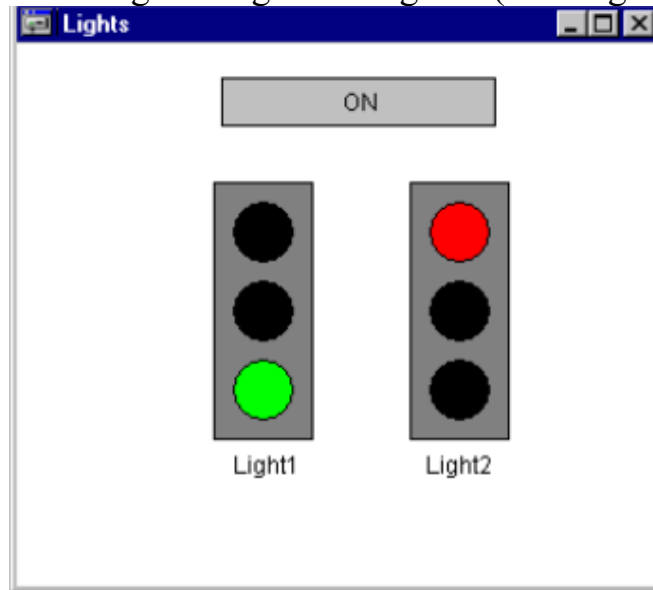


'Toggle variable' field (Toggle variable) in the 'Input' category , enter the variable .ON . The switch we created will turn the traffic lights on/off. You can display the on state with a color, just like for the traffic light. Enter the variable in the ' Change color ' field Color).



Visualization labels. We'll insert two rectangles under the traffic lights . IN properties element V categories Colors (Colors) color lines (frame)

rectangle ask white. IN field 'Line' - Contents (category 'Text' - Text) enter names traffic lights “Light1” " Light2” (drawing 10).



Drawing 10- Visualization For project Traffic Signal

Exercise For independent work

Execute All examples programs. Screenshots screen With Present the program code and results of the work in the report. Draw conclusions about the feasibility of different approaches to developing controller software .

LIST USED SOURCES

1. Bazhenov, A. V. Programming of embedded microprocessors systems: textbook / A. V. Bazhenov, N. Yu. Bratchenko, N. V. Grivennaya. - Stavropol: SKFU Publishing House, 2022. - 302 p., unlimited copies.
2. Vodovozov, AM Microcontrollers for automation systems [Electronic resource]: textbook / AM Vodovozov. - Electronic text data . - M.: Infra-Engineering, 2016. - 164 p. - 978-5-9729-0138-8.
— Mode access: <http://www.iprbookshop.ru/51727.html>
3. Fundamentals of programming microprocessor controllers in digital process control systems [Electronic resource] : textbook / V. S. Kudryashov, A. V. Ivanov, M. V. Alekseev [et al.]. — Electronic text data. — Voronezh: Voronezh State University of Engineering Technologies, 2014. — 144 p. — 978-5-00032-054-9. — Access mode: <http://www.iprbookshop.ru/47437.html>
4. Shegal , A. A. Application of the Multisim software package for designing devices on microcontrollers [Electronic resource]: laboratory practical training / A. A. Shegal ; edited by V. I. Ievlev. - Electronic text data. - Ekaterinburg: Ural Federal University, Electronic Library System ASV, 2014. — 116 c. — 978-5-7996-1117-0. — Mode access :
<http://www.iprbookshop.ru/65968.html>
5. Aliyev, M. T. ; ; ; Microprocessors and microprocessor systems management. 8-bit processors families AVR : laboratory practice / M.T. Aliyev, T.S. Bukanova ; Povolzhsky gosudarstvennyy technological university. - Yoshkar- Ola : PSTU, 2016. - 64 p. : schema., tab., ill. - <http://biblioclub.ru/>. - Bibliography in the book. - ISBN 978-5-8158-1775-3, unlimited copies
6. Bezuglov, D. A. Digital devices and microprocessors: a textbook / D. A. Bezuglov, I. V. Kalienko . - 2nd ed. - Rostov n / D: Phoenix, 2008. - 468 p.: ill. - (Higher education). - Bibliography : pp. 464-465. - ISBN 978-5-222-13917-2
7. Hartov , IN. I. Microcontrollers AVR : workshop For beginners : textbook allowance / IN. I. Hartov . - M. : MSTU, 2007. - 240 sec. : ill. - Adj.: With. 231-238. - Bibliography : With. 239. - ISBN 978-5-7038-3051-2
8. Management user By programming PLC V CoDeSys 2.3//https://owen.ru/product/codesys_v2/documentation
9. - temporal signal processing: Tutorial / A.V. Bazhenov, N.V. Grivennaya , M.F. Goryainov, S.V. Malygin - Stavropol: YEW, 2016. - 219 With., ill.

METHODOLOGICAL INSTRUCTIONS

For students By organizations And conducting independent work
By discipline « High-frequency electronic circuits/ High Frequency Electronic Circuit »
For students areas of training
11.03.02 Infocommunication technologies and communication systems Profile
Intellectual processing data V infocommunication -
tional systems and networks

Stavropol, 2026

Content

[Introduction](#)

Methodological recommendations By study theoretical Methodological
material recommendations By note-taking primary sources
Methodological guidelines for preparation for credit with a grade
List recommended literature

INTRODUCTION

Modern student training standards require a significant amount of extracurricular work. Completing the course program requires both theoretical knowledge of social psychology and the development of practical skills. Therefore, Independent work within the course is focused on both theoretical and practical aspects.

Independent work of students is the completion of theoretical and practical assignments by students to master the studied discipline "Microprocessor technology in infocommunications"

The purpose of independent work is to consolidate and deepen the knowledge acquired by students in lectures, preparation for current practical classes, intermediate forms of knowledge assessment and final assessment.

Didactic goals independent classes :

- formation professional skills;
- formation skills And skills independent mental labor;
- motivation regular purposeful work By development specialties;
- development independence thinking;
- formation of conviction, strong-willed character traits, and the ability to self - organize ;
- mastery technological educational tool.

Independent work includes those sections of the course that were not sufficiently covered in lectures due to limited lecture time and the large volume of material being studied.

Methodological security independent work By discipline consists of from:

- 1) Definitions of learning questions that students should study independently ;
- 2) Selection of the necessary educational literature, mandatory for study and research;
- 3) Search for additional scientific literature that students can refer to at their own discretion if they become interested in a given topic;
- 4) Definitions of control questions that allow students to independently check the quality of the knowledge they have acquired;
- 5) Organizing consultations between teachers and students to clarify questions that have caused difficulties for students in independently mastering the educational material.

a specially organized activity that includes in its structure such components as:

- clarification goals And delivered educational tasks;
- clear And systemic planning independent work;
- search necessary educational And scientific information;
- development own information And her logical processing;
- use of research methods, research work to solve the assigned tasks;
- production own positions By about received task;
- performance, justification And protection received solutions;
- conducting introspection and self-control

Types independent works By educational discipline:

- Job With conceptual apparatus;
- note-taking primary sources;
- design digital devices on microcontrollers;
- analysis scientific research;

- Preparation scientific project By topic.
ongoing and final assessments. Ongoing assessment is carried out during the course and includes assessment of students' work during practical classes (round tables, interviews, group research projects , testing), as well as homework preparation.

METHODOLOGICAL RECOMMENDATIONS

BY STUDY THEORETICAL MATERIAL

A comprehensive study of the academic discipline offered to students “High-frequency electronic circuits/ High Frequency Electronic " Circuit " involves mastering the materials of lectures, textbooks, programs, creative work of students during practical classes, as well as systematic completion of assignments for independent work.

Lectures explore key issues within the topics covered, emphasizing the most complex and interesting aspects of the material being studied, which students should consider. Lecture materials form the basis for preparing students for practical classes.

Job With lecture notes

Review your notes immediately after class. Mark any lecture material that you find difficult to understand. Try to find answers to difficult questions using the provided literature. If you can't understand the material on your own, formulate questions and ask your instructor for help during the current tutorial or at the next lecture.

Each week, set aside time to review the material covered, testing your knowledge, skills, and abilities with quizzes and tests.

Performing laboratory work works on discipline "High-frequency electronic circuits/ High Frequency Electronic Circuit »

At the first lesson, you will receive course assignments and preparation plans from the teacher. for laboratory work. Obtain all necessary methodological support.

Tasks laboratory research:

- development instrumental funds developments built-in microprocessor systems,
- studying languages descriptions of the schemes,
- familiarization With modern devices on base microcontrollers And PLC,
- development And implementation devices on PLC,
- organization interface between basic computer And device on base micro-processor devices.

METHODOLOGICAL RECOMMENDATIONS BY NOTE- TAKING SOURCERS

The study and analysis of literary sources is a mandatory form of independent student work. The study of literature on a chosen topic aims to trace the way various authors have posed and solved a particular problem, the arguments for their conclusions and generalizations, and to analyze and systematize the obtained material based on one's own understanding in order to clarify the current state of the issue.

The processing of the selected material must be accompanied by the simultaneous recording of what has been read and your comments. The recording can take the form of summaries, extracts, as well as a card index of provisions, theses, ideas, and methods, which will subsequently facilitate the classification and systematization of the obtained material. This kind of recording is the best way to accumulate and initially process material, one from the obligatory forms of organization of mental work.

The plan is convenient to use when preparing for an oral presentation on the chosen topic. Each point of the plan should cover one from the sides of the chosen topic, and the entire plan should cover it in its entirety.

Theses are concise statements of the main points of a text, either affirmatively or negatively. They are a more advanced form of writing and provide a basis for discussion. They are also easy to remember.

Note-taking is the process of mentally processing and recording information in writing, in the form of a brief summary of the main content and meaning of any text.

The result of note-taking is a record that allows the note-taker to immediately or later fully recall the information obtained. The word "summary" translates from Latin as "overview." Essentially, it should be compiled as an overview, containing the main ideas of the text without details or secondary quirks. A summary is personalized: it is intended for the author and therefore may be difficult for others to understand.

In order to carry out this type of work, in each specific case it is necessary to competently solve the following tasks:

1. Get your bearings V general compositions text (be able to define introduction, main part, conclusion).
2. To see the logical and semantic outline of the message, to understand the author's system of presenting information as a whole, as well as the course of development of each individual thought.
3. Identify the "key" thoughts, i.e. the main ones semantic milestones, on which "strung" the entire content of the text.
4. Define detailing information.
5. Formulate the main information concisely, without transferring everything verbatim into writing.

Every scientific text contains two types of information: primary and secondary. Primary information is the information that has the most significant meaning. To reveal the content of a topic or question. This includes: definitions of scientific concepts, formulations of laws, theoretical principles, etc. The purpose of auxiliary information is to help the reader better understand the material presented. This type of information includes various kinds of commentary. We record the main information as fully as possible, while auxiliary information is generally omitted. Note-taking consists of processing the main information to summarize and condense it. To summarize means to present it in a more general, schematic form, in the form of abstracts, conclusions, separate headings, a presentation of the main results, etc. When reading, we intuitively use certain words and phrases as references. Such reference words and phrases are called keywords. Keyword and phrases carry the main semantic and emotional load of the text.

The selection of keywords is the first stage of semantic condensation, semantic compression of the material.

Important requirements for notes are clarity and visibility of notes and their arrangement in such a way as to make it possible to understand logical connections and the hierarchy of concepts.

By form notes are subdivided on formalized and graphic.

1. **Formalized** (All records are introduced V in advance prepared tables).

This is convenient, firstly, when taking notes on materials when the list of characteristics of the objects or phenomena being described is more or less constant, and secondly, when preparing a single summary based on several sources, especially if there is a need to compare individual data. A variation of a formalized summary is a record compiled in the form of answers to pre-prepared questions, providing exhaustive characteristics of similar objects or phenomena.

2. Graphic (elements of the work being outlined are arranged in such a way that the hierarchy of concepts and the relationship between them is visible).

there may be not one, but several graphic summaries reflecting the book as a whole and its individual parts. Taking graphic notes is the most perfect way to depict the internal structure of a book, and this process itself helps in assimilating its content.

The following main types of notes can be distinguished: planned, textual, summary, and thematic.

Planned – easy to obtain with the help of a pre-made plan of the work; each question of the plan is answered by a specific part of the notes:

a) question-and-answer (the outline provides precise answers to the plan points expressed in interrogative form);

b) schematic planned abstract (reflects logical structure and the relationship between individual provisions).

Textual - This abstract, created basically from quotes.

A summary combines excerpts, quotations, and sometimes theses; part of the text may be provided with an outline.

Thematic – provides a more or less comprehensive answer (depending on the number of sources involved) and other material, for example, your own notes) on the question posed - topic: overview; chronological.

The role of notes is purely educational: it helps to capture the main concepts and provisions of the primary text and reproduce them at the right moment, for example, when writing an essay or preparing for an exam.

Methods note-taking.

- **Theses** are brief statements of the main ideas and propositions of the material being studied. Theses succinctly express the essence of what is being read and provide an opportunity to expand on the content. When beginning to master the notation of theses, it is useful to mark the passages in the text itself that most clearly formulate the main idea the author is proving (unless, of course, it is a library book). This selection is often facilitated by highlighting the text itself.

- **Linear-sequential text recording.** When taking notes using the linear-sequential method, it is advisable to use poster-style design tools, which include the following:

- shift text notes by horizontals, vertically;
- selection bold (or to others) font especially significant words;
- usage various flowers;
- underline;
- conclusion V frame main information.

- With **the "question and answer" method.** It consists of dividing the page in half by a vertical line, the person taking notes on the left side of the page independently formulates questions or problems raised in the text, and on the right side gives answers to them.

One modification of the question-and-answer method is a table, where the question is replaced by the problem statement raised by the author (lecturer), and the answer is replaced by the solution to that problem. Sometimes the table may also include additional columns, such as "my opinion," etc.

- A **diagram with fragments** is a method of taking notes that allows one to more clearly reveal the structure of the text; in this case, fragments of the text (key words, phrases, explanations of all kinds) in combination with graphics help to create a rational and concise summary.

- **Simple scheme** - note-taking method, close to scheme With fragments, The note - taker doesn't write explanations for them, but must be able to give them orally. This method requires a highly qualified note-taker. Otherwise, such notes cannot be used.

- **Parallel** note-taking. Notes are written on two sheets of paper, either side by side, or one sheet is divided in half by a vertical line, and notes are written on both the right and left sides of the sheet.

However, **it** is better to use different note-taking methods to record one and the same material.

Combined notes are the pinnacle of mastering rational note - taking . All the above methods are skillfully used, combining them in one note- taking (one type of note-taking freely flows into another depending on the text being noted and the desire and skill of the note-taker). It is with combined notes that more notes the level of preparation and student's individuality.

Principles compilation notes read.

1. , and place of publication. If the text is taken from a periodical (newspaper or magazine), record its title , year, month, issue, date, and place of publication.

2. You can highlight the margins on the left or right, or on both sides. The left margins are used to mark the original pages, structural sections of the article or book (paragraph titles, subheadings, etc.), and formulate the main points. The right margins are used to indicate the ways in which the information you've read will be recorded.

One type of reading – in-depth reading – involves deep assimilation of what has been read and often the retention of information for subsequent reference. The effectiveness of such reading increases if what has been read is fixed not only in memory, but also on paper. Psychologists say that written down better and is absorbed more fully, more firmly stored in memory. It has been established that if you read 1000 words and then write down 50 summarizing what you have read, the absorption coefficient will be higher than if you read 10,000 words without having recorded none. Except Moreover, writing down what you've read develops the skill of compiling information. Finally, alternating reading and writing reduces fatigue and increases work efficiency and mental productivity.

1. Summary .

've read, containing an evaluation of it. A summary characterizes the book's main findings and conclusions.

The choice of linguistic means for constructing summary conclusions is subordinated to the main task of information condensation: a minimum of linguistic means – a maximum of information. This is usually one to three clear, concise, expressive sentences that, in the author's opinion , reveal the very essence of the object being described.

2. Annotation.

An abstract is a brief, generalized description of a printed work (book, article), sometimes including its evaluation. This is the shortest possible summary of the contents of the primary document , providing a general idea of the topic.

Its main purpose is to give some idea about the book (article, scientific work) in order to recommend it to a certain circle of readers or to use their notes when performance of work research, abstract nature. Therefore, the annotation does not need to outline the content of the work; it only lists the issues covered in the primary source (the content of these issues is not disclosed) . The annotation answers the question: “What does the primary text say?”, provides

an idea only of the main topic and the list of issues raised in the original text .

Abstract text is not standardized. Various requirements for abstract composition can be found in the scientific literature. For example, a reference abstract may include the following information:

- type and title of the annotated document (monograph, dissertation, collection, article, etc.)
- tasks, delivered by the author annotated document
- the method used by the author (experiment, comparative analysis, compilation of other sources)
- affiliation author To certain scientific school or direction
- structure annotated document
- item And topic works, main provisions And conclusions author
- characteristics auxiliary illustrative materials, additions, appendices, reference apparatus, including indexes and bibliography.

METHODOLOGICAL INSTRUCTIONS BY PREPARATION FOR CREDIT WITH ASSESSMENT

Monitoring and assessing students' knowledge is an integral part of the educational process at the university. The exam session is the most crucial time in a student's life: it represents the culmination of students' extensive work throughout the semester. With assessment - is a method of testing students' knowledge in part or in full course of the academic discipline, produced by staging oral or written questions. It provides an objective, officially recorded assessment of students' performance over a specified period of time.

The assessment with a grade has a multifaceted purpose: firstly, it is a test of the student's knowledge, secondly, They ourselves By to myself are important link in mastering science, thirdly, it is a continuation of the educational process; finally, it is of great importance as a factor in stimulating in-depth study of the subject.

Preparing for the test The assessment process consists of two interconnected stages. The first is systematic work throughout the semester and academic year, encompassing all forms of the educational process: lectures, study, and note-taking. recommended literature , active participation in seminars (practical) and laboratory classes.

The second is preparation immediately before the test. With assessment. It allows students to grasp the full perspective of what they have learned in a relatively short period of time . and better understand the basic patterns and phenomena.

For direct preparation To credit With assessment additional time does not stand out.

You need to prepare according to a strictly thought-out schedule, consistently moving from topic to topic, without skipping any of them.

Complex questions that were not sufficiently clarified during the preparation for the test must be written down and clarified by teachers during pre - exam consultations.

It's worth noting that students who skip tutorials because they perceive a lack of challenging questions are acting recklessly. During tutorials, the lecturer often not only answers questions but also, on their own initiative , explains the most difficult sections of the course .

So, the main thing the task of preparing students for the test With Assessment involves systematizing knowledge of the educational material and creatively interpreting it. The most important teaching aid at this stage of the student's work is their own notes from lectures and independently researched course topics.

In accordance with the regulations on examinations, they are usually taken by teachers who teach the course of lectures.

After receiving the question, the student must carefully consider the content of the questions posed . A significant number of unsuccessful answers are explained by a vague understanding of the problem posed . A correct understanding of the question will ensure success in answering it . When preparing to answer the question, it is necessary to make a detailed plan for each question .

When answering the questions on the exam, don't rush. Present the material calmly, deliberately, maintain self-control, and pay attention to your sentence structure. Avoid broad strokes and generalities. It's recommended to construct your answers clearly, consistently, concretely , and, if possible , exhaustively. At the same time, it's highly desirable to quickly and accurately illustrate your answer with examples.

From examinee required: definition of concepts, justification put forward provisions, free manipulation of factual material, provide alternative approaches to individual problems. Logic, consistency, Literary literacy and literacy of presentation are essential features of a complete answer. Neither excessive brevity, which borders on schematism, nor verbosity should be tolerated. Both are unjustified. Brevity prevents the instructor from understanding the student 's mastery of the material, while verbosity may indicate that the student is unable to emphasize the main points and is too vague.

LIST RECOMMENDED LITERATURE

1. Scroll basic literature:

1. Bazhenov, A. V. Programming of embedded microprocessor systems: textbook / A. V. Bazhenov, N. Yu. Bratchenko, N. V. Grivennaya. - Stavropol: SKFU Publishing House, 2022. – 302 p., unlimited copies.

2.. Novikov, Yu. V. Fundamentals microprocessor technology: a tutorial / Yu. V. Novikov, P. K. Skorobogatov. - Fundamentals of microprocessor technology, 2022-07-28. - Electronic data. (1 file). - Moscow: Internet University of Information Technologies (INTUIT), IPR Media, 2020. - 405 p. - electronic. - The book is in the premium version of the IPR BOOKS electronic library system. - ISBN 978-5-4497-0677-5, unlimited copies

3. Vodovozov, AM Microcontrollers for automation systems [Electronic resource] : educational allowance / A. M. Vodovozov. - Electron. text data. — M. : Infra-Engineering, 2016. — 164 p. — 978-5-9729-0138-8. — Access mode: <http://www.iprbookshop.ru/51727.html>

4. Fundamentals of programming microprocessor controllers in digital process control systems [Electronic resource]: tutorial / V. S. Kudryashov, A. V. Ivanov, M. V. Alekseev [et al.]. - Electronic text data. - Voronezh: Voronezh State University of Engineering Technologies, 2014. — 144 c. — 978-5-00032-054-9. — Mode access: <http://www.iprbookshop.ru/47437.html>

5. Shegal, A. A. Application of the Multisim software package for designing devices on microcontrollers [Electronic resource]: laboratory practical training / A. A. Shegal; edited by V. I. Ievlev. - Electronic text data. - Ekaterinburg: Ural Federal University, EBS ASV, 2014. — 116 c. — 978-5-7996-1117-0. — Access mode: <http://www.iprbookshop.ru/65968.html>

2. Scroll additional literature:

1. Aliev, M. T. Microprocessors and microprocessor control systems. 8-bit processors of the AVR family: laboratory practical training / M.T. Aliyev, T.S. Bukanova; Volga State Technological University. - Yoshkar-Ola: PSTU, 2016. - 64 p. : diagrams, tables, take. - <http://biblioclub.ru/>. - Bibliogr. in the book. - ISBN 978-5-8158-1775-3, copies unlimited

2. Bezuglov, D. A. Digital devices and microprocessors: a textbook / D. A. Bezuglov, I. V. Kalienko. - 2nd ed. - Rostov n / D: Phoenix, 2008. - 468 p.: ill. - (Higher education). - Bibliography: pp. 464-465. - ISBN 978-5-222-13917-2

3. Khartov, V. Ya. AVR microcontrollers: a practical guide for beginners: a tutorial / IN. I. Hart's. - M.: MSTU, 2007. - 240 With.: ill. - Adj.: With. 231-238. - Bibliography: With. 239. - ISBN 978-5-7038-3051-2

4. Melnikov, E. V. Basics microprocessor techniques : laboratory practical training / E. IN. Melnikov. - Basics microprocessor technology, 2026-09-20. - Electronic data (1 file). - Samara: Samara State Technical University, EBS ASV, 2020. - 47 p. - electronic. - The book is in the premium version of the EBS IPR BOOKS. - ISBN 2227-8397, unlimited copies

3. List of educational and methodological support for independent work of students in a discipline (module)

1. Bazhenov, A.V. Methodological recommendations for the implementation of laboratory work on the discipline "Microprocessor technology in infocommunications", Stavropol: SKFU, 2022. - 125 p.