MATERIAŁY DYDAKTYCZNE DO PRZEDMIOTU

Introduction to Programming PICAXE Microcontrollers

Wykład 30 godzin

Wydział Elektrotechniki, Automatyki i Informatyki

Opracował
Krzysztof Ludwinek

Kielce, 2013

Contents

1. BASIC ARCHITECTURE OF THE PICAXE MICROCONTROLLER SYSTEM..... 3
1.1. PICAXE CHIP LABELS..... 4
1.2. CHARACTERISTICS OF THE PICAXE PROGRAM WRITTEN IN PICAXE THE PROGRAMMING EDITOR..... 5
1.2.1. Variables..... 5
1.2.2. Constants..... 7
1.2.3. Expressions..... 7
1.2.4. Logical Operations..... 9
1.2.5. Meaningful Names..... 10
1.2.6. Labels..... 10
1.2.7. Comments..... 11
1.2.8. Pausing and Stopping Program Execution 11
1.2.9. Commands..... 12
1.2.10. Selection..... 12
1.2.11. Iteration (Looping) 13
1.2.12. Subroutines..... 14
1.2.13. Interrupts..... 14
1.2.14. Sounds..... 15
1.2.15. Pulse Width Modulation..... 15
1.3. FIRST PROGRAM IN PICAXE PROGRAMMING EDYTOR..... 15

2. THE PICAXE MICROCONTROLLER FAMILY..... 16
2.1. PROGRAMMING THE PICAXE MICROPROCESSORS..... 17
2.2. DETAILED PINOUT DIAGRAMS FOR PICAXE MICROCONTROLLERS..... 18

3. PICAXE PROGRAMMING EDITOR..... 33

4. PRACTICAL UTILIZATION OF PICAXE IN CONTROLLING STEPPER AND DC MOTORS..... 40
4.1. THE STEPPER MOTOR..... 40
4.1.1. Switching of stepper motor windings 40
4.2. MODES OF CONTROLLING STEPPER MOTOR..... 44
4.2.1. The wave drive..... 45
4.2.2. The full step drive..... 46
4.2.3. Half step drive 47
4.3. EXAMPLE OF STEPPER MOTOR CONTROLLING 48
4.4. THE DC MOTOR..... 49
4.5. THE IR REMOTE CONTROLS..... 54
4.6. PICAXE COOPERATION WITH THE SEVEN-SEGMENT DISPLAY..... 57

REFERENCES..... 59

1. BASIC ARCHITECTURE OF PICAXE MICROCONTROLLER SYSTEM

Microcontroller can be described by a lot of definitions and it is hard to find one that will contain all of them. Generally, the microcontroller is an electronic device that consists of a combination of processor, memory and I/O devices. This device can be combined into two types: a single chip or separate chips. However the microcontroller is usually associated with a single chip solution. Currently, there are many manufacturers of microcontrollers on the market and over the years there will be more and more. Fig. 1 shows the basic architecture of an 8 bit microprocessor of Microchip or PICAXE Incorporation [1]. The only name that is engraved on the top of the PICAXE chip tells about the type of microchip unit [2]. PICAXE chips are pre-programmed and tested by Microchip PICmicro™ microcontrollers [3].

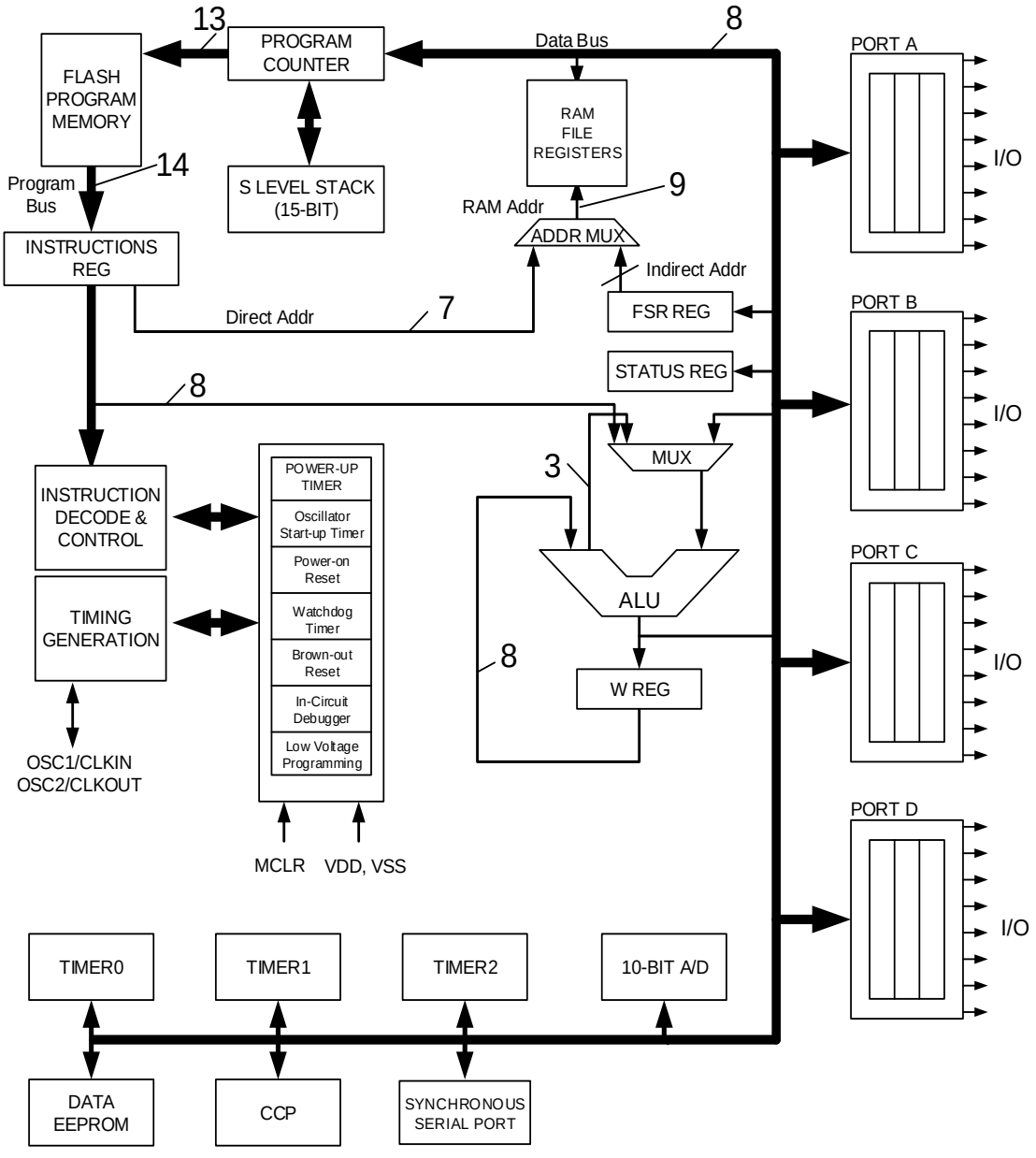


Fig. 1.1. Basic architecture of an 8 bit microprocessor by PICAXE (Microchip) Inc [1].

1.1. PICAXE CHIP LABELS

More recent M2 parts are custom parts 'factory engraved' with the full PICAXE name. Other parts are simply 'engraved' with the Microchip part name. Table 1 and table 2 present the PICAXE type and what is engraved on the top of the newer version chip (Table 1), which is the replacement type in the case of the older version chip (Table 2) [2] and about possibility programming in the Assembler code after converting into the Assembler Code.

Table 1. New PICAXE type and what is engraved on the top of the chip

PICAXE Type	***Engraving on top of chip***
PICAXE-08M2	PICAXE-08M2
• PICAXE-14M2	PICAXE-14M2
• PICAXE-18M2	PICAXE-18M2
• PICAXE-20M2	PICAXE-20M2
• PICAXE-20X2	PIC18F14K22
• PICAXE-28X1	PIC16F886
• PICAXE-28X2	PIC18F25K22
• PICAXE-40X1	PIC16F887
• PICAXE-40X2	PIC18F45K22

Table 2. Older PICAXE type, the name engraved on the top of the chip and the replacement type and possibility programming in the Assembler code

PICAXE Type	***Engraving***	***Replacement Type in the Assembler Code***	
PICAXE-08	PIC12F629	*Superseded by 08M2*	*yes*
PICAXE-08M	PIC12F683	*Superseded by 08M2*	*yes*
PICAXE-14M	PIC16F684	*Superseded by 14M2*	*yes*
PICAXE-18	PIC16F627(A)	*Superseded by 18M2*	*yes*
PICAXE-18A	PIC16F819	*Superseded by 18M2*	*yes*
PICAXE-18M	PIC16F819	*Superseded by 18M2*	*yes*
PICAXE-18X	PIC16F88	*Superseded by 18M2*	*yes*
PICAXE-20M	PIC16F677	*Superseded by 20M2*	
PICAXE-28A	PIC16F872	*Superseded by 28X1*	*yes*
PICAXE-28X	PIC16F873A	*Superseded by 28X1*	*yes*
PICAXE-28X2-5V	PIC18F2520	*Superseded by 28X2*	
PICAXE-28X2-3V	PIC18F25K20	*Superseded by 28X2*	
PICAXE-40X	PIC16F874A	*Superseded by 40X1*	
PICAXE-40X2-5V	PIC18F4520	*Superseded by 40X2*	
PICAXE-40X2-3V	PIC18F45K20	*Superseded by 40X2*	

1.2. CHARACTERISTICS OF THE PICAXE PROGRAM WRITTEN IN THE PICAXE PROGRAMMING EDITOR

The PICAXE family is programmed in Basic in the PICAXE Programming Editor in version 5 [2]. The PICAXE Programming Editor is the legacy freeware, specially developed software for programming, testing, and simulating BASIC programs for the PICAXE family [2]. All PICAXE code programs written in Basic are downloaded to the PICAXE chip. The PICAXE Program Code (in alphabetical order) consists of [3]:

- **Commands,**
- **Comments,**
- **Data,**
- **Labels,**
- **Reserved Words.**

The main component of the program Code is Data. Data may be the type of variables, constants, and expressions.

- Variables – items that can be changed in value using commands.
- Constants – items that can be used by commands, but the value cannot be changed.
- Expressions – connections between Variables, constants and all operators that can return values.

1.2.1. Variables

Various types of variables are used by the PICAXE chips [3]:
Word variables,
Byte variables,
Bit variables,
Special function variables,
Central processing unit (CPU) registers,
Electrically erasable programmable read-only memory (EEPROM).

Word Variables - this type of a variable (16th bits) hold ranges from 0 to 65535 unsigned numbers. Word variable consists of 16-binary digits (bits). PICAXE programmers have access to seven available byte variables which are named:
w0, w1, w2, w3, w4, w5, and w6.

Byte Variables - this type of a variable (8th bits) has ranges from 0 to 255 unsigned numbers or ASCII (*American Standard Code for Information Interchange*) characters. A byte variable consists of 8-binary digits (bits). PICAXE programmers have access to fourteen available word variables which are named:
b0, b1, b2, b3, b4, b5, t6, b7, b8, b9, b10, b11, b12, and b13.

Bit Variables - this type of variable has either the 0 or the 1 value. A bit variable consists of a single binary digit (bit). PICAXE programmers have access to sixteen available bit variables which are named:
bit0, bit1, bit2, bit3, bit4, bit5, bit6, bit 7, bit 8, bit 9, bit 10, bit 11, bit 12, bit 13, bit14, and bit15.

Relationship Between Variables

The relationship between byte and word variables is shown in Table 3. It can be seen, for example, that W0 is made up by B1 (MSB) and B0 (LSB) etc. All changes which the program makes to B1 or B0 will result in a change in W0. This also works the other way around.

Variable names in the Programming Editor can be written in the following way:

W4 or **w4**,
B2 or **b2**,
BIT0 or **Bit0** etc.

Table 3. The relationship between byte and word variables

WORD, AND BYTE VARIABLES		
WORD VARIABLE	MOST SIGNIFICANT BYTE (MSB)	LEAST SIGNIFICANT BYTE (LSB)
W0	B1	B0
W1	B3	B2
W2	B5	B4
W3	B7	B6
W4	B9	B8
W5	B11	B10
W6	B13	B12
etc.	etc.	etc.

Table 4 presents the relationship between W0, B0/B1, and bit variables. It shows that change to BIT4 that will affect B0 and W0, whereas BIT14 will affect B1 and W0

Table 4. The relationship between W0, B0, B1 and bit variables

W0, B0/B1, AND BIT VARIABLES		
WORD VARIABLE	BYTE VARIABLE	BIT VARIABLE
BIT0	B0, bit 0 (the least significant in byte B0)	W0, bit 0 (the least significant)
BIT1	B0, bit 1	W0, bit 1
BIT2	B0, bit 2	W0, bit 2
BIT3	B0, bit 3	W0, bit 3
BIT4	B0, bit 4	W0, bit 4
BIT5	B0, bit 5	W0, bit 5
BIT6	B0, bit 6	W0, bit 6
BIT7	B0, bit 7 (the most significant in byte B0)	W0, bit 7,
BIT8	B1, bit 0 (the least significant in byte B1)	W1, bit 8,
BIT9	B1, bit 1	W0, bit 9
BIT10	B1, bit 2	W0, bit 10
BIT11	B1, bit 3	W0, bit 11
BIT12	B1, bit 4	W0, bit 12
BIT13	B1, bit 5	W0, bit 13
BIT14	B1, bit 6	W0, bit 14
BIT15	B1, bit 7 (the most significant in byte B1)	W0, bit 15 (the most significant)

There are Special Function Variables that are used in the PICAXE Programming Editor are (in alphabetical order) [3]:

dirs: specifies the data direction. It is used for pins that can be I/O,
dirsc: specifies the data direction for port C pins,
infra: **Infrain** the command uses it as a predefined variable; for PICAXE-08M, it is a b13,
keyvalue: **keyin** the command uses it as a predefined variable and it is also another name for **infra**,
pins: used when reading – the input port and when writing – the output port,
pinsa: used when reading - the input port A,
pinsc: used when reading - the input port C, when writing – The output port C,
pin0, pin1, pin2, pin3, pin4, pin5, pin6, and pin7: The input port individual pins.

1.2.2. Constants

Data that does not change has the name "Constants". Constants cannot be placed to the left of the assignment operator (= assign). There are a few types of constants [3]:

Numeric Constants can be written as a decimal number, hexadecimal number or a binary number.
Binary constants preceded by a **(%)** symbol contain digits 0 or 1.
Hexadecimal Constants preceded by a **($)** symbol contain digits from 0 to 9 and from A to F.
Decimal Constants contain digits from 0 to 9.
Character or String this type of Constants are enclosed in double quote marks **(“)**.

Some examples of constants (Numeric, Character and String) are listed below:

%00000101	The binary constant 00000101 (5 in decimal)
$3C	The hexadecimal constant 3C
100	The decimal constant one hundred
“C”	The ASCII character C
“CONSTANT”	The ASCII string CONSTANT
“123”	The ASCII string 123
“$3C”	The ASCII string $3C
“%10”	The ASCII string %10

Any character without the **(“)** mark can be used in character or string constant.

1.2.3. Expressions

The simplest definition is that expressions are combinations of constants, operators (single values) and variables. We can use them in the **(let)** assignment command. There are more Programming Editor commands in which we use expressions. Below, the form of an expression is shown [3].

Variable operator {constant/variable} [operator {constant/variable} ...] where

Variable- PICAXE byte, bit, word or a special var.
Operator – Programming Editor operator. There can be more than one operator in an expression.
Constant – Number/Character.

The PICAXE Programming Editor uses the following operators:

*	Numeric multiplication, returns a lower portion of the result,
**	Numeric multiplication, returns a high portion of the result,
/	Numeric division, returns the quotient,
//	Numeric division, returns the remainder,
%	The same function as //,
+	Numeric addition,
-	Numeric subtraction,
=	Assigment operator,
MIN	Result of a numeric operation limited to a minimum value,
MAX	Result of a numeric operation limited to a maximum value,
I or OR	Logical OR
& or AND	Logical AND
^ or XOR	Logical XOR
&/	Logical AND NOT
I/	Logical OR NOT
^/	Logical XOR NOT

The mathematical expression is evaluated **from left to right** if there exists more than one operator. For example:

3+3+3*3 will be evaluated to 27 not 15. First, we have addition then we add multiplication unlike mathematical rules where (*) is before (+).

Arithmetical Operations mean Assigning Values to Variables. To assign value to a variable, the **let** command is required. Below, the Let command syntax is shown [3]:

[**LET**] variable = value

where

let – can be deleted (optional)
variable – PICAXE byte, bit, word or special variable.
Value – constant/variable that return value.

There are examples of the **Let** command using:

Let b0 = 1	‘value 1 assigned to b0
b0 = 1	‘the same meaning, but ‘**let**’ is deleted
b0 = b1	‘b1 value placed in b0
b0 = b0+1	‘b0 value is added to 1 (constant) and the result is assigned to ‘b0

1.2.4. Logical Operations

George Boole was a mathematician who first used Boolean algebra (because of his name). Usually we use Boolean algebra to set, clearing or shifting bits. Boolean algebra expresses logical operations. The values have only two different states and they could be [3]:

On/Off
True/False
1/0 (Basic)
Logic high/low

There are a lot of Boolean operations (called also logical operations) like

OR, AND, NOT, XOR (Basics) and **NAND, XNOR, NOR, or Logical shift.**

The logical operation and their functions are shown in Table 5 below.

Table 5. Logical functions

Input1	**Input2**	Input	NOT	OR	XOR	AND	NAND	NOR	XNOR
0	**0**	0	1	0	0	0	1	1	1
0	**1**	1	0	1	1	0	1	0	0
1	**0**	-	-	1	1	0	1	0	0
1	**1**	-	-	1	0	1	0	0	1

EXAMPLE

```
B1 = %00001111
B2 = %11110000
B0 = B1|B2 (B1 OR B2)        '(Logical OR, result is %11111111 in B0)

B1 = %00111100
B2 = %00011000
B0 = B1&B2                   '(Logical AND, result is %00011000 in B0)

B0 = %00011000
B0 = B0 * 2                  '(Logical shift left, result is %00110000 in B0)

B0 = %00011000
B0 = B0/2                    '(Logical shift right, result is % 00011000 in B0)
```

SHIFT BIT VALUES

2 One bit
4 Two bits
8 Three bits
16 Four bits
32 Five bits

Also **64, 128, 256, 512, 1024** and so on.

1.2.5. Meaningful Names

It is really difficult to get an indication as to what is in variables. It can be known what kind of variable it is (word, byte, bit). This situation does not facilitate reading program code. The PICAXE Programming Editor provides a **symbol command** that makes constants, variables or other items to give more meaningful names. Some examples of syntax of symbol commands are listed below [3]:

SYMBOL symbolname = value [operator constant]
where
symbolname – a name given to symbol,
value – a constant/variable must be valid in the symbol context,
operator – only one allowed the mathematical or Boolean operator,
constant – written in a binary, decimal, or hexadecimal from optional number or a predefined symbol.

To name a symbol, there are some rules which should be taken into account:
It must not consist of reserved words.
"SYMBOL" and "symbol" have the same meaning.
It may consist of letters or numbers.
It must begin with an underscore or letter.
It must be defined before use.
If the symbol is defined once, it can be used in a program if the context is valid.

There are some examples of symbols below:

```
Symbol dataout = 0,
Symbol switch = pin3,
Symbol outbyte = b4,
Symbol symbol_2 = %00000101 & $3C,
```

When **dataout** is used it will be interpreted as 0 and switch as pin3.

1.2.6. Labels

The label is a name that is used in the program as a reference point. There are some rules which should be taken into account [3]:
It must begin with a letter.
It may contain letters, numbers.
It must be unique.
It must not consist of reserved words.
It must end with a colon (:).
"LABEL" and "label" are the same.
If a label name used in some commands, the colon (:) is not used.
Labels are often written in column 1 by the editor. The labels can be written on the same line as a command, for example:

```
Label1:
        command1
Label2:
        command1
```

NOTE. If the label refers to commands, the colon is omitted.

```
goto label1
```

1.2.7. Comments

A programmer can insert some text (explanations) into a program. To begin a comment, the (‘) mark is required and the text after (‘) mark is not taken into account by the Programming Editor. Comments do not take any space of memory in PICAXE. Below, some examples of using comments are shown [3]:

```
                    ; This is a comment
Label1:; Comment about Label1

    ; pause _300    ; Comment about command pause_300
    ; Pause 300     ; In this line pause 300 is a comment and is not
                    ; taken into account by the Programming Editor
```

Inserting a blank is not required in the program, because the Programming Editor does not take blank into account.

1.2.8. Pausing and Stopping Program Execution

In order to Pause and Stop Program Execution in the PICAXE Programming Editor the following commands are used [3]:

Stop,
Pause,
Wait,
Sleep,
Nap,
End.

Stop command – until hardware will be reset by the restoring power, program execution stops. PICAXE is not in low power mode.

Pause command – usually used to make delay and “stop” program execution for some time. The period of time is as accurate as the PICAXE clock. The Pause command syntax is as follows [3]:

PAUSE time
where
time – **constant/variable** (from 0 to 65535), the time delay is specified in milliseconds.

Wait command – usually used to make delays and suspend program execution. Similar to the **Pause command**, but time must be a constant (Pause command can also be variable). It is specified in seconds (not in milliseconds). The **Wait command** syntax is as follows [3]:

WAIT time
where
time – constant (from 0 to 65), specified in seconds.

End command – stops program execution and changes PICAXE into a low power mode. PICAXE will be in low power mode until hardware reset or reset line. Very often the **End command** is the last command in the program code.

1.2.9. Commands

If PICAXE has to do some operations, some instructions are necessary. These instructions are precisely commands. A command may consist of single words such as **pause, let** or operate in pairs like **if … then** or **gosub … return.** Commands may need some parameters consisting of constants, variables or expressions specified by the programmer. Each command has its own syntax (rules). You can find the Programming Editor syntax by pressing the **<F1>** key in the PICAXE Programming Editor, where more than 70 commands are available.

1.2.10. Selection

Instructions choice **if...then\ elseif \ else \ endif** or **Branch** are used during the execution of one operation and if you want to change the operations during inputting pin or general variables (in program) for certain conditions. If these conditions meet the section **if...then\ elseif \ else \ endif** the program code is executed between the positions **if** ... **then** or **elseif** (or **else**) **... endif**. The **elseif** (or **else**) **... endif** section is executed if none of the **if** ... **then** or **elseif** conditions are true [3]

In PICAXE we can use commands like

if...then\ elseif \ else \ endif (Program executes in one of two possible places)
Branch (Program executes in one of many possible places)

If…then command syntax:

IF variable value (operator) [{**AND/OR**} variable value (operator)…] **THEN** label

where

variable -	PICAXE variable
operator-	Comparison operator
label -	After it the program continues execution
AND, OR -	Boolean operators.

Comparison operators (in the Programming Editor) [3]:

Is	equal to
=	equal to
!=	not equal to
<>	not equal to
<=	less than equal to
<	less than
>=	greater than or equal to
>	greater than

EXAMPLE

```
If w3 <= 1000 then label 1
If bit 3 = 10 and bit2 != 100 then label 2
If bit1 is 0 then label 3
```

The **Branch** command syntax given below is as follows [3]**:**

BRANCH variable, (label0, label1, label2, label3,…,labelN)
where
variable - PICAXE variable

1.2.11. Iteration (Looping)

If we want the program to execute some operations more than one time, we should (save time and place) use looping commands. There are 3 commands which PICAXE uses [3].

For…next (do operation n times in logical order)

If…then (do operation a specific number of times)

GOTO (command which ends the loop)

For…next command syntax:

FOR variable = startvalue **TO** endvalue [**STEP** (-) increment]
Command (n)
NEXT variable

Where:

variable - word/byte PICAXE variable
startvalue - from this constant/variable the loop will start
endvalue - at this constant/variable the loop will end

In this example, the loop will be executed twice.

```
b2 = 1
b3 = 2
for b0 = b2 to b3 step 1
        high c.5
        pause 500
        low c.5
        pause 500
next b0
```

If…then

EXAMPLE

```
Symbol endvalue = b2
        B0 = 2
        Endvalue = 5
Label1:
        Command (n)
        B0 = b0 + 1
        If b0 <= endvalue then label1
```

GOTO command syntax:

GOTO label

EXAMPLE

```
Label1:
        Command (n)
        goto label1
```

Here, command (n) is executed by the program until PICAXE will reset.

1.2.12. Subroutines

If you want to make the program easier, or reduce the program size, subroutines are the best option. To use the subroutines, the PICAXE programmer can use **gosub** and **return** commands. The **gosub** command in the program can be used 16 or 256 times (depending on the chip used). The **gosub** command is used to start a temporary user program. The temporary user program is executed in the separate section of a code between the **gosub** and **return** commands. When the **return** command is executed, the program returns to the main program, i.e. to the first command after the **gosub** Command [3].

Gosub command syntax:

GOSUB label

Return command syntax:

RETURN

```
        command1
        gosub sub1
        command2
        gosub sub1
        command3
        gosub sub2
        command4
        gosub sub1
        end
sub1: command6
        return
sub2: command7
        return
```

1.2.13. Interrupts

Sometimes our program has to react to external events (maybe one event) at the moment when the event occurs. To do such a thing we can use a special command which we place in our program. This works in a similar way as a subroutine. It starts with the **interrupt** label and ends with the **return** command. We place it at the end of the program. When we have **interrupt** - the program starts executing the interruption operation, and with the **return** command our program come back to the place when it stopped before the interruption. To enable interrupts, we use the **SETINT** command. It is shown below [3]:

SETINT condition, mask

where

Condition - variable/constant with the 1 or 0 pin state.
Mask - variable/constant which defines the pin.

1.2.14. Sounds

PICAXE chips allow to play the music with the '**tune**' command (not available on PICAXE- 8) [4]. **TUNE pin,speed**,(**note**, **note**, **note**...) [4]
where:

- **pin** is a variable/constant which specifies the number of pin to generate the sounds.
- **speed** is a variable/constant (1-15) which specifies the tempo of the tune, 1 - is very quick and 15 is very slow,
- **notes** are the tune sounds (data) that are generated by the '**tune**' command (see details [4]).

1.2.15. Pulse Width Modulation

PICAXE chips allow to change the average value of a voltage (and current) that fed to the load (e. g. DC motor). The technique of Pulse Width Modulation changes the width of the output pulse. Formally, the pulse duration is controlled by turning on and off the transistor output. **PWM pin,duty,cycles** [4]
where:

- **Pin** is a variable/constant which specifies the Input/Output pins,
- **Duty** is a variable/constant (0-255) which specifies the level of analog signals.
- **Cycles** is a variable/constant (0-255) which specifies the number of cycles (frequency). For 4MHz clock frequency, each cycle takes about 5ms [4].

Example.

```
pwm B.0,150,40        ; the (PICAXE-28X2 send series of 40 pwm pulses - out of ; pin B.0
```

1.3. FIRST PROGRAM IN PICAXE PROGRAMMING EDYTOR

```
Main:          ;in this way we define the program label
               ; label must not start with the digit and special names used in PICAXE
               ;Programming Editor. You can use Capital or small letters .
high 7         ; switch on the output (out)
HIGH 6         ; and 6 (the default port for port B)
pause 6000     ;wait for 6000 ms
Low 7          ; switch off the output 7
low 6          ;and 6
Pause 6000     ; pause in ms
goto main      ; jump to MAIN label
```

2. PICAXE MICROCONTROLLER FAMILY

At present, the PICAXE microcontrollers system can be used in the following physical sizes: 8, 14, 18, 20, 28 and 40 pins [2], [3]. The main difference between the size of these chips is the number of inputs and outputs. Of course, larger chips are more expensive because they have a larger number of inputs/outputs. The specified size of the chip may have different variations, such as 18-pin PICAXE occurs in 18, 18A and 18X variants. In general, the difference between the variants is in the amount of available memory (it means how much a program length can be uploaded into the chip). Each project can be used in a higher variety of the microcontroller (for example, when a program is too long, you can use a better version with more memory). Each higher variant of chips is compatible with lower microcontrollers. The power supply can be 3-5V (default - 5), 1.8 - 3.3 (standard - 3V) and 1.8 - 5 for universal X2 systems. To work properly, some terminals require additional 10 - 22 kΩ resistors. When programming, PICAXE chips do not pull the programming wire. RESET Pin must be included (tucked up) to Vcc via a resistor 4.7k. The reset occurs after shortening the pin to ground (0 V). Quartz resonators 4, 8, 10 and 16 MHz optional included (for systems with 28 and 40 pins) with built-in internal PLL feedback loop allows to increase system clock frequency by 4 times. The main differences between I/O and clock frequency of different options for PICAXE microcontrollers are presented in Table 6.

Table 6. The main differences between the PICAXE microcontrollers family

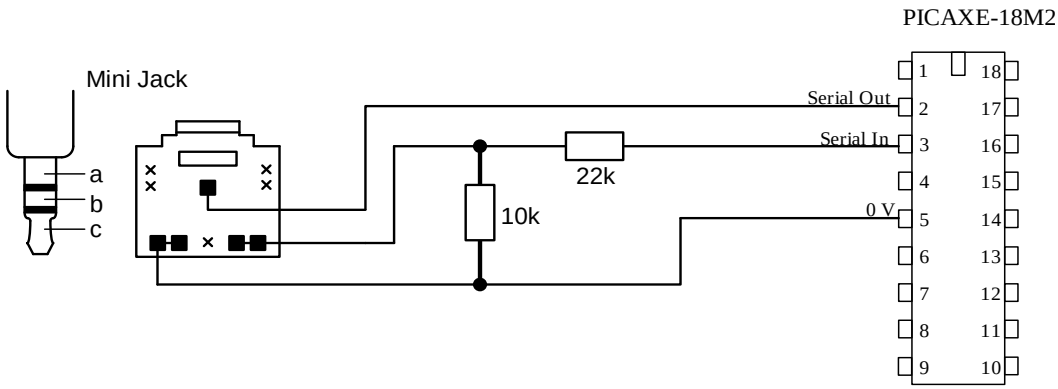
Level	Number of Pins	Digital I/O	ADC	Frequency MHz	Memory Length - Bytes
Educational	8A	5 inputs/outputs	1	4	128
	18A	5 inputs, 8 outputs	3	4	256
Initial/Budget	08M	5 inputs/outputs	3	8	256
	14M	5 inputs, 6 outputs	2	8	256
	18M	5 inputs, 8 outputs	3	8	256
Standard	18X	5 inputs, 8outputs	3	8	2048
	28X1	0-12 inputs, 9-17 outputs	0-4	20	4096
	40X1	8-20 inputs, 9-17 outputs	3-7	20	4096
Edvanced	28X2	22 configured in./out	0-19	up to 40 (4x10)	4096
	40X2	33 configured in./out	0-27	up to 64 (4x16)	4096

Particular differences between PICAXE chips presented in Table 6 are as follows [3, 4].
For Educational Level (08A, 18A) very low cost of purchase but only 128 - 256 bytes of program code.
For Budget level (08M, 14M, and 18M) program memory is twice the size (256 bytes). The chips have Support for digital temperature sensor and enable control via radio transmitter, they have Keyboard input and the ability to define user-music tone, Infrared transmission, 8/10 bit ADC, and the PWM motor control.

Standard Level „X": (18X) has all advantages of the properties for the "A" family, program memory (2048 bytes): 8 times larger than in the "A" family, it also has the interface for EEPROM and other devices, 8/10 bit ADC, 12-bit temperature reading option, the PWM motor control, and counts the pulses at the input.
Standard Level „X1": (28X1, or 40X1) has all advantages of the properties for the "X" family, program memory (4096 bytes) is 16 times greater than in the "A" family, it has an internal clock, increased mathematical opportunities, the real line used for simulation.
Advanced Level „X2": (28X2, or 40X2) has all advantages of the properties for the "X", program memory (4096 bytes) is 4 times larger than in "X1" family, it has an internal clock, increased number of variables of word type W (W0-W27), byte type B (B0 - B54), increased mathematical opportunities, and the real line used for simulation.

2.1. PROGRAMMING THE PICAXE MICROPROCESSORS

PICAXE microprocessors allow programming without using the programming device through the special Boot Loader program. Figure 2 shows a simple and more advanced serial download circuit with the mini Jack (with security against short circuit and anti-static - Shotky diode and an additional 180 Ω resistor) to programming PICAXE chips. The serial download circuit (in Fig. 2 connected to the PICAXE 18M2) is identical for all the PICAXE family [4]. In Fig. 2 the mini jack has the following indexes:
a - Serial Out,
b - Serial In,
c - 0 V.
a)



b)

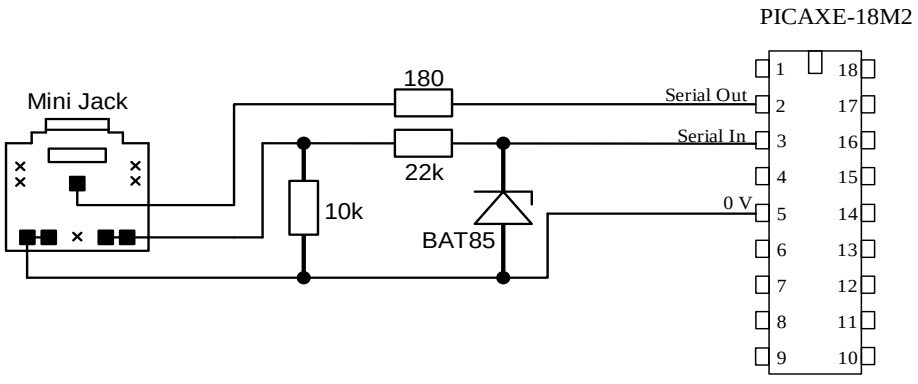


Fig. 2. The schema of a PICAXE chip programming a) simple way, b) more advanced

The most popular way of programming the PICAXE family is using the RS232 PC port. The way how to connect the wire (shown in Fig. 2) to the RS232 port of the computer is shown in Fig. 3 [4].

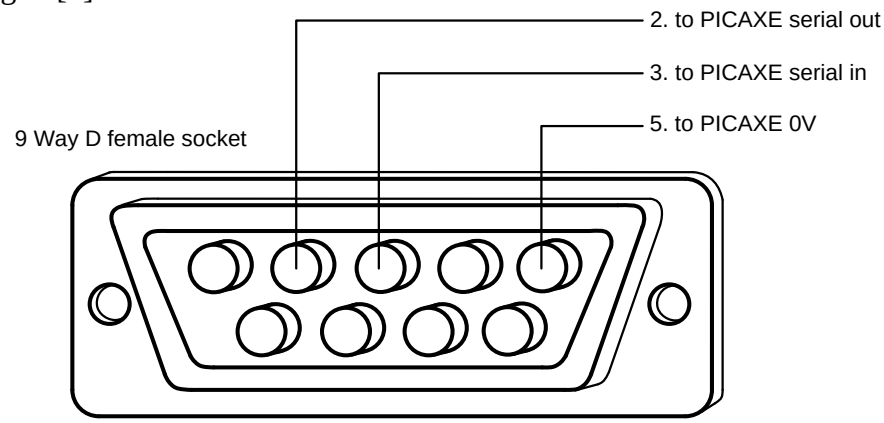


Fig. 3. How to connect the circuit (shown in Fig. 2) to the RS232 port of any PC computer

In the case of communication using the RS232 port, connection is possible with the purchase of additional wire (cable) communications, the same for the all PICAXE family. For programming from personal computers or notebooks using USB port you should buy PICAXE USB download cable [2].

2.2. DETAILED PINOUT DIAGRAMS FOR PICAXE MICROCONTROLLERS

In this section, detailed pinout diagrams for the 8-, 14-, 20-, 28- and 40-pins PICAXE devices are shown [4]. The detailed pinout diagrams for the PICAXE-08M chip with the original PICAXE pin names are shown in Figure 4 [4] and are also presented in Table 7.

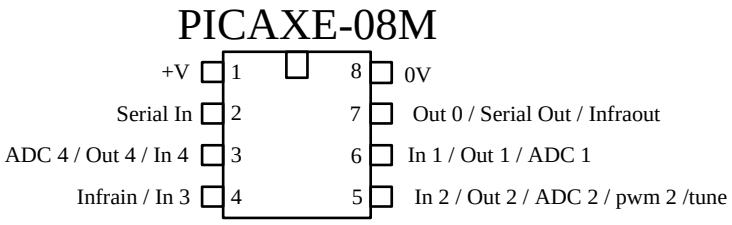


Fig. 4. Pinout diagram of the PICAXE-08M

Table 7. The detailed specification for pinout of the PICAXE-08M

No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1							+V
2						Serial In	
3	In 4	Out 4					
4	In 3						
5	In 2	Out 2	ADC 2		PWM 2	tune	
6	In 1	Out 1	ADC 1				
7		Out 0				Infraout /Serial Out	
8							0V

The detailed pinout diagrams for the PICAXE-014M chip with the original PICAXE pin names are shown both in Figure 5 [4] and in Table 8.

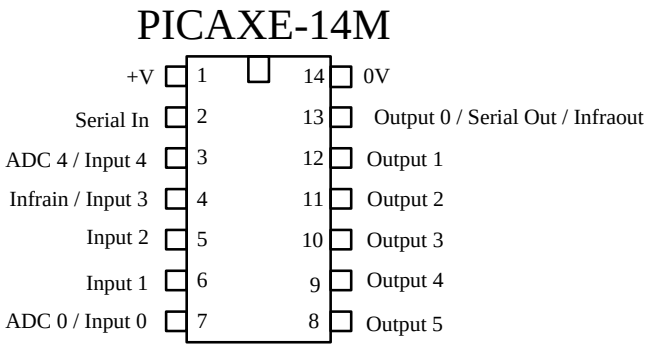


Fig. 5. Pinout diagram of the PICAXE-14M

Table 8. The detailed specification for pinout of the PICAXE-14M

No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1							+V
2						Serial In	
3	In 4		ADC 4				
4	In 3					Infrain	
5	In 2						
6	In 1						
7	In 0		ADC 0				
8		Out 5					
9		Out 4					
10		Out 3					
11		Out 2					
12		Out 1					
13		Out 0				Serial Out / Infraout	
14							0V

The detailed pinout diagrams for the PICAXE-14M2 chip with the original PICAXE pin names are shown both in Figure 6 [4] and in Table 9.

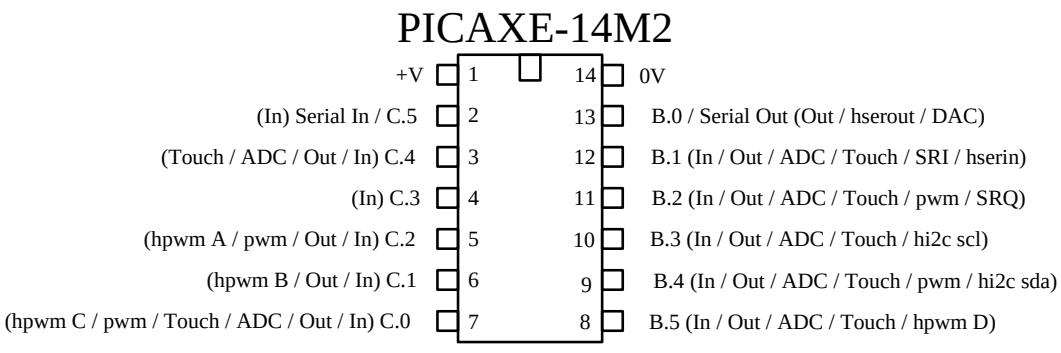


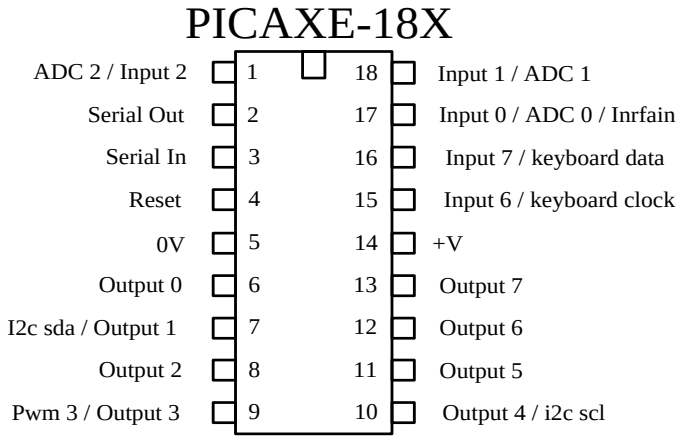
Fig. 6. Pinout diagram of the PICAXE-14M2

Table 9. The detailed specification for pinout of the PICAXE-14M2

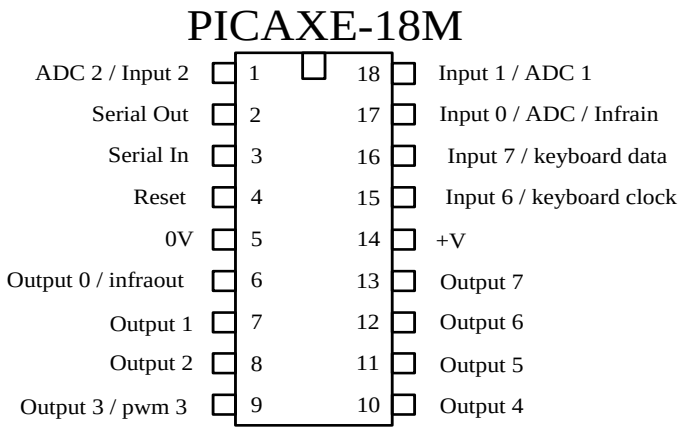
No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1							+V
2	In C.5					Serial In	
3	In C.4	Out C.4	ADC C.4			Touch	
4	In C.3						
5	In C.2	Out C.2			PWM C.2	HPWM A	
6	In C.1	Out C.1				HPWM B	
7	In C.0	Out C.0	ADC C.0		PWM C.0	HPWM C/Touch	
8	In B.5	Out B.5	ADC B.5			Touch	
9	In B.4	Out B.4	ADC B.4			Touch/hi2c sda	
10	In B.3	Out B.3	ADC B.3			Touch/hi2c scl	
11	In B.2	Out B.2	ADC B.2			Touch/SRQ	
12	In B.1	Out B.1	ADC B.1			Touch/hserin . SRI	
13		Out B.0		DAC B.0		Serial Out/hserout/	
14							0V

The detailed pinout diagrams for the PICAXE-18X, PICAXE-18M, PICAXE-18M2 chips with the original PICAXE pin names are shown in Figure 7 [4] and in Tables 10 and 11, respectively.

a)



b)



c)

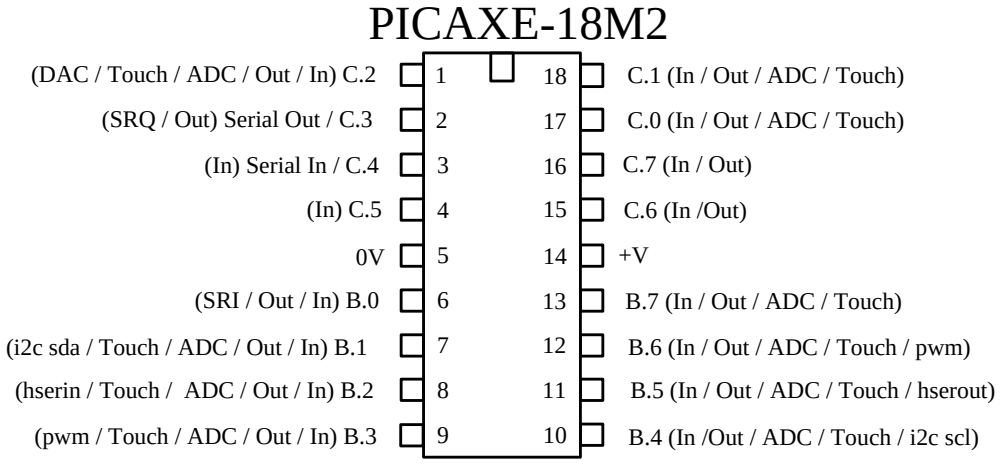


Fig. 7. Pinout diagram of a) PICAXE-18X, b) PICAXE-18M, c) PICAXE-18M2

Table 10. The detailed specification for pinout of the PICAXE-18X and PICAXE-18M

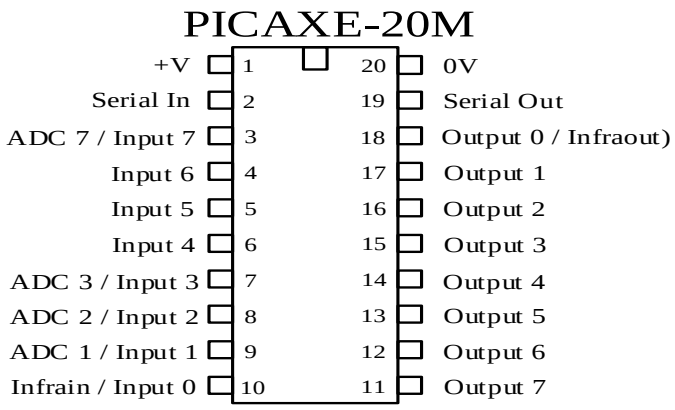
No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1	In 2		ADC 2				
2						Serial Out	
3						Serial In	
4						Reset	
5							0V
6		Out 0				Infraout	
7		Out 1					
8		Out 2/Sda (PICAXE 18X)					
9		Out 3			PWM 3		
10		Out 4/IC2 scl (PICAXE 18X)					
11		Out 5					
12		Out 6					
13		Out 7					
14							+V
15	In 6					Keyboard clock	
16	In 7					Keyboard data	
17	In 0		ADC			Infrain	
18	In 1		ADC 1				

Table 11. The detailed specification for pinout of the PICAXE-18M2

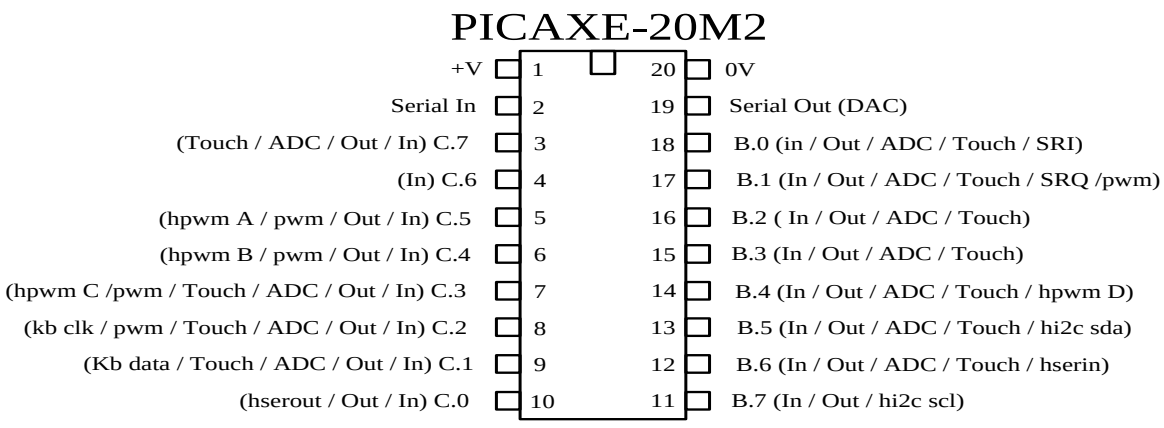
No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1	In C.2	Out C.2	ADC C.2	DAC C.2		Touch	
2		Out C.3				SRQ/Serial Out	
3	In C.4					Serial In	
4	In C.5						
5							0V
6	In B.0	Out B.0				SRI	
7	In B.1	Out B.2	ADC B.1			i2c sda/Touch	
8	In B.2	Out B.2	ADC B.2			Hserin/Touch	
9	In B.3	Out B.3	ADC B.3		B.3	Touch	
10	In B.4	Out B.4	ADC B.4			Touch/i2c scl	
11	In B.5	Out B.5	ADC B.5			Touch/hserout	
12	In B.6	Out B.6	ADC B.6		B.6	Touch	
13	In B.7	Out B.7	ADC B.7			Touch	
14							+V
15	In C.6	Out C.6					
16	In C.7	Out C.7					
17	In C.0	Out C.0	ADC C.0			Touch	
18	In C.1	Out C.1	ADC C.1			Touch	

The detailed pinout diagrams for the PICAXE-20M, PICAXE-20M2 and PICAXE-20X2 chips with the original PICAXE pin names are shown in Figure 9 [4] and in Tables 12, 13 and 14, respectively.

a)



b)



c)

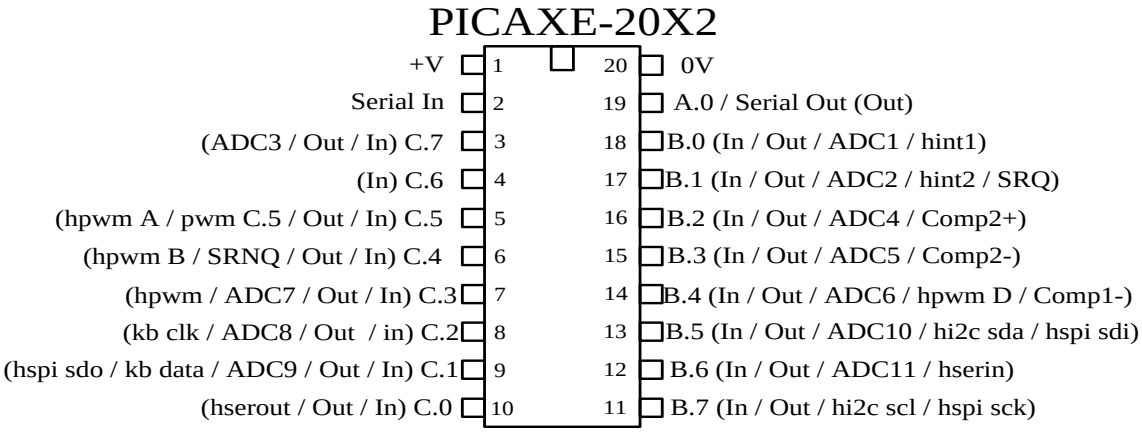


Fig. 8. Pinout diagram of a) the PICAXE-20X2, b) the PICAXE-20M2, c) the PICAXE-20M

Table 12. The detailed specification for pinout of the PICAXE-20M

No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1							+V
2						Serial In	
3	In 7		ADC 7				
4	In 6						
5	In 5						
6	In 4						
7	In 3		ADC 3				
8	In 2		ADC 2				
9	In 1		ADC 1				
10	In 0					Infrain	
11		Out 7					
12		Out 6					
13		Out 5					
14		Out 4					
15		Out 3					
16		Out 2					
17		Out 1					
18		Out 0				Infraout	
19						Serial Out	
20							0V

Table 13. The detailed specification for pinout of the PICAXE-20M2

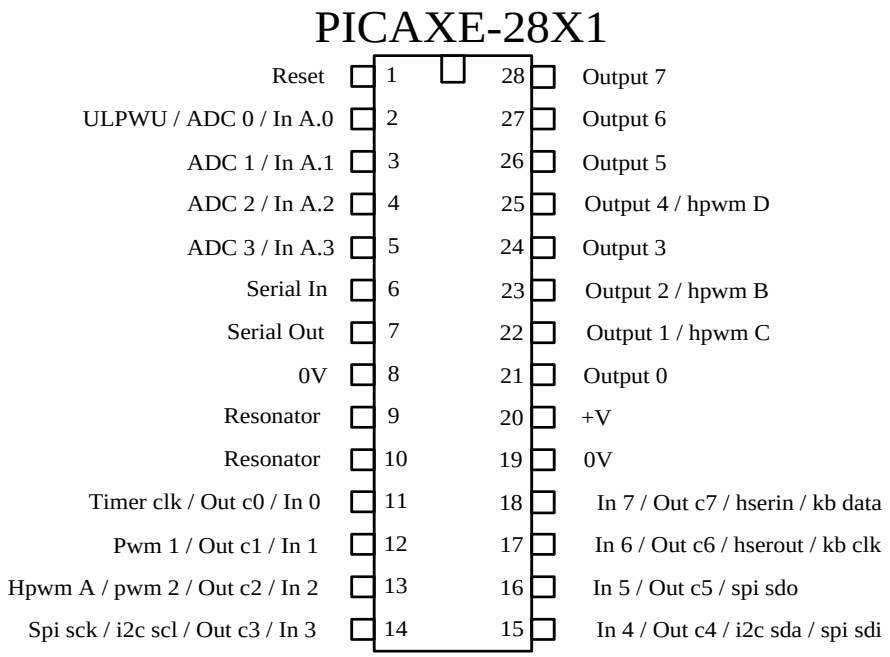
No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1							+V
2						Serial In	
3	In C.7	Out C.7	ADC C.7			Touch	
4	In C.6						
5	In C.5	Out C.5			PWM C.5	HPWM A	
6	In C.4	Out C.4			PWM C.4	HPWM B	
7	In C.3	Out C.3	ADC C.3		PWM C.3	HPWM C	
8	In C.2	Out C.2	ADC C.2		PWM C.2	Touch/Kb clk	
9	In C.1	Out C.1	ADC C.1			Touch/Kb data	
10	In C.0	Out C.0				hserout	
11	In B.7	Out B.7				Hi2c scl	
12	In B.6	Out B.6	ADC B.6			Touch/ hserin	
13	In B.5	Out B.5	ADC B.5			Touch/hi2c sda	
14	In B.4	Out B.4	ADC B.4			Touch/HPWM D	
15	In B.3	Out B.3	ADC B.3			Touch	
16	In B.2	Out B.2	ADC B.2			Touch	
17	In B.1	Out B.1	ADC B.1		PWM B.1	Touch/SRQ	
18	In B.0	Out B.1	ADC B.0			Touch/SRI	
19				DAC		Serial Out	
20							0V

Table 14. The detailed specification for pinout of the PICAXE-20X2

No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1							+V
2						Serial In	
3	In C.7	Out C.7	ADC 3				
4	In C.6						
5	In C.5	Out C.5			PWM C.5	HPWM A	
6	In C.4	Out C.4				SRNQ/HPWM B	
7	In C.3	Out C.3	ADC 7			HPWM	
8	In C.2	Out C.2	ADC 8			Kb clk	
9	In C.1	Out C.1	ADC 9			Kb data/hspi sdo	
10	In C.0	Out C.0				hserout	
11	In B.7	Out B.7				Hi2c scl/hspi sck	
12	In B.6	Out B.6	ADC 11			hserin	
13	In B.5	Out B.5	ADC 10			Hi2c sda/hspi sdi	
14	In B.4	Out B.4	ADC 6			HPWM D/Comp1-	
15	In B.3	Out B.3	ADC 5			Comp2-	
16	In B.2	Out B.2	ADC 4			Comp2+	
17	In B.1	Out B.1	ADC 2			Hint2/SRQ	
18	In B.0	Out B.0	ADC 1			Hint1	
19		Out A.0				Serial Out	
20							0V

The detailed pinout diagrams for the PICAXE-28X1 and PICAXE-28X2 chips with the original PICAXE pin names are shown in Figure 9 [4] and presented in Tables 15 and 16, respectively.

a)



b)

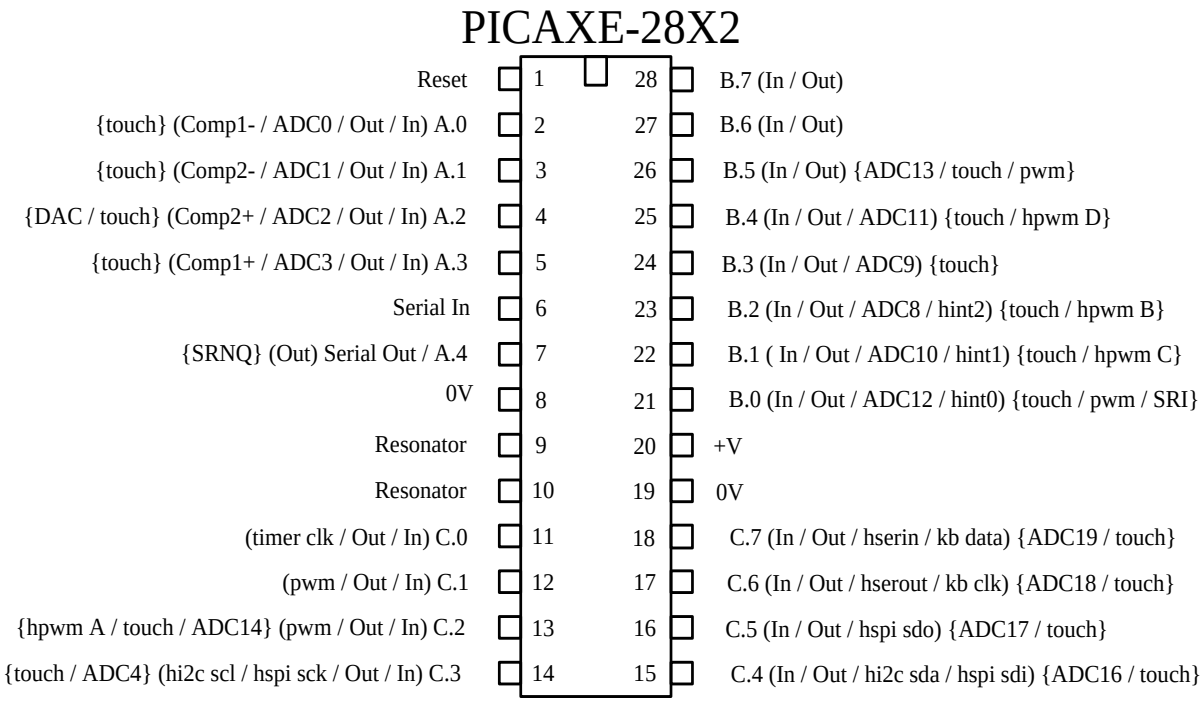


Fig. 9. Pinout diagram of a) PICAXE-28X1, b) PICAXE-28X2

Table 15. The detailed specification for pinout of the PICAXE-28X1

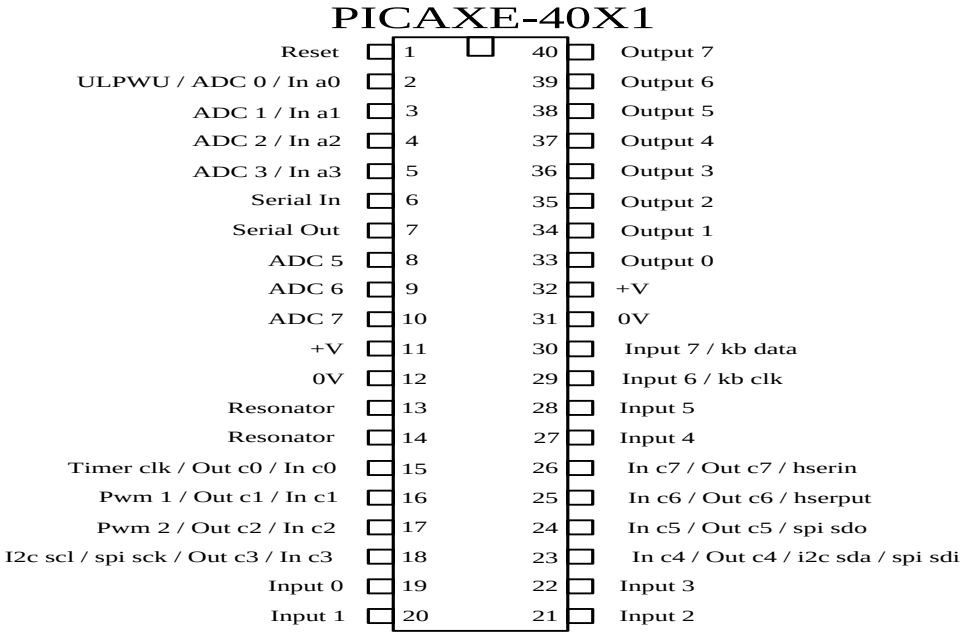
No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1						Reset	
2	In A.0		ADC 0			ULPWU	
3	In A.1		ADC 1				
4	In A.2		ADC 2				
5	In A.3		ADC 3				
6						Serial In	
7						Serial Out	
8							0V
9						Resonator	
10						Resonator	
11	In 0	Out C.0				Timer clk	
12	In 1	Out C.1			PWM 1		
13	In 2	Out C.2			PWM 2	HPWM A	
14	In 3	Out C.3				I2c scl/Spi sck	
15	In 4	Out C.4				I2c sda /s spi sdi	
16	In 5	Out C.5				Spi sdo	
17	In 6	Out C.6				Hserout/Kb clk	
18	In 7	Out C.7				Hserin/Kb data	
19							0V
20							+V
21		Out 0					
22		Out 1				HPWM C	
23		Out 2				HPWM B	
24		Out 3				HPWM D	
25		Out 4					
26		Out 5					
27		Out 6					
28		Out 7					

Table 16. The detailed specification for pinout of the PICAXE-28X2

No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1						Reset	
2	In A.0	Out A.0	ADC 0			Touch/Comp1-	
3	In A.1	Out A.1	ADC 1			Touch/Comp2-	
4	In A.2	Out A.2	ADC 2	DAC		Touch/Comp2+	
5	In A.3	Out A.3	ADC 3			Touch/Comp1+	
6						Serial In	
7		Out A.4				SRNQ/Serial Out	
8							0V
9						Resonator	
10						Resonator	
11	In C.0	Out C.0				Timer clk	
12	In C.1	Out C.1			C.1		
13	In C.2	Out C.2	ADC 14		C.2	Touch/HPWM A	
14	In C.3	Out C.3	ADC 4			Touch/hi2c scl . hspi sck	
15	In C.4	Out C.4	ADC16			Touch/hi2c sda/hspi sdi	
16	In C.5	Out C.5	ADC 17			Touch/hspi sdo	
17	In C.6	Out C.6	ADC 18			Touch/hserout/kb clk	
18	In C.7	Out C.7	ADC 19			Touch/hserin/kb data	
19							0V
20							+V
21	In B.0	Out B.0	ADC 12		B.0	Touch/SRI/hint0	
22	In B.1	Out B.1	ADC 10			Touch/HPWM C/hint1	
23	In B.2	Out B.2	ADC 8			Touch/HPWM B/hint2	
24	In B.3	Out B.3	ADC 9			Touch	
25	In B.4	Out B.4	ADC 11			Touch/HPWM D	
26	In B.5	Out B.5	ADC 13		B.5	Touch	
27	In B.6	Out B.6					
28	In B.7	Out B.7					

The detailed pinout diagrams for the PICAXE-40X1 and PICAXE-40X1 (44 Pin TQFP) chips with the original PICAXE pin names are shown in Figures 10 [4] and presented in Tables 17 and 18, respectively.

a)



b)

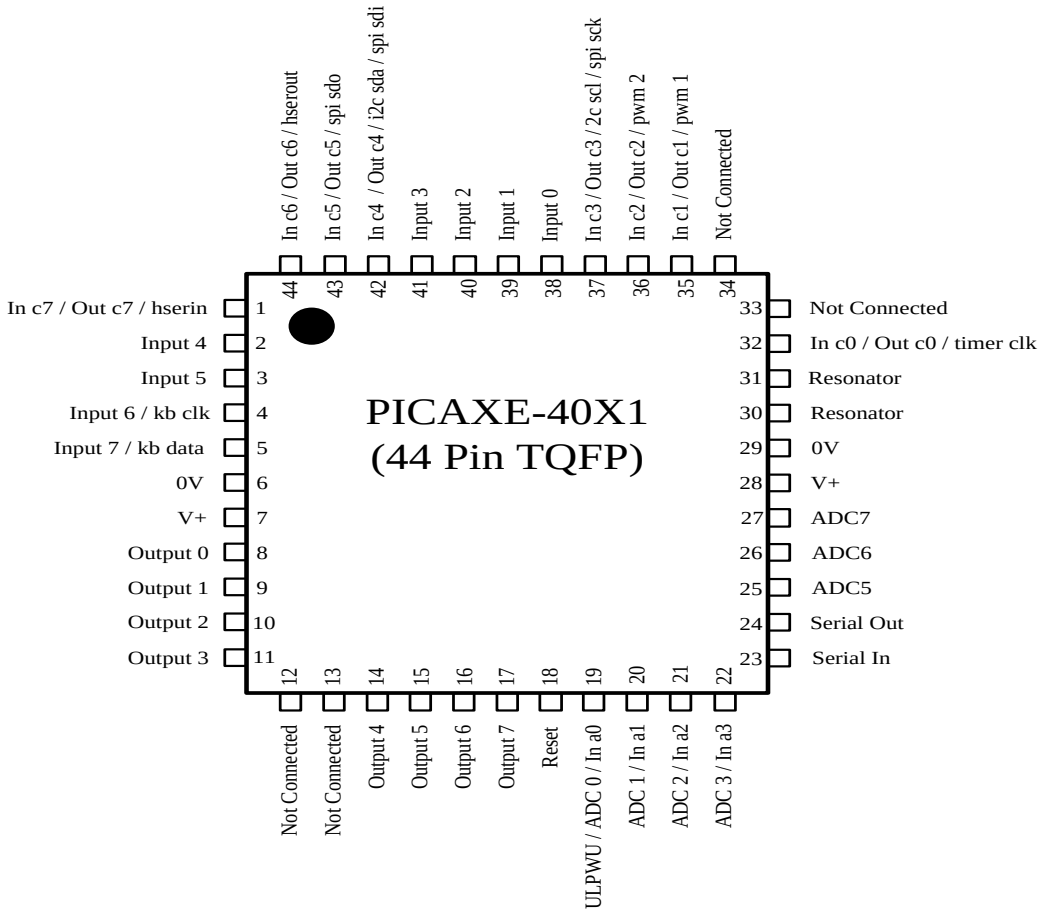


Fig. 10. Pinout diagram of a) the PICAXE-40X1, b) the PICAXE-40X1 (44 Pin TQFP)

Table 17. The detailed specification for pinout of the PICAXE-40X1

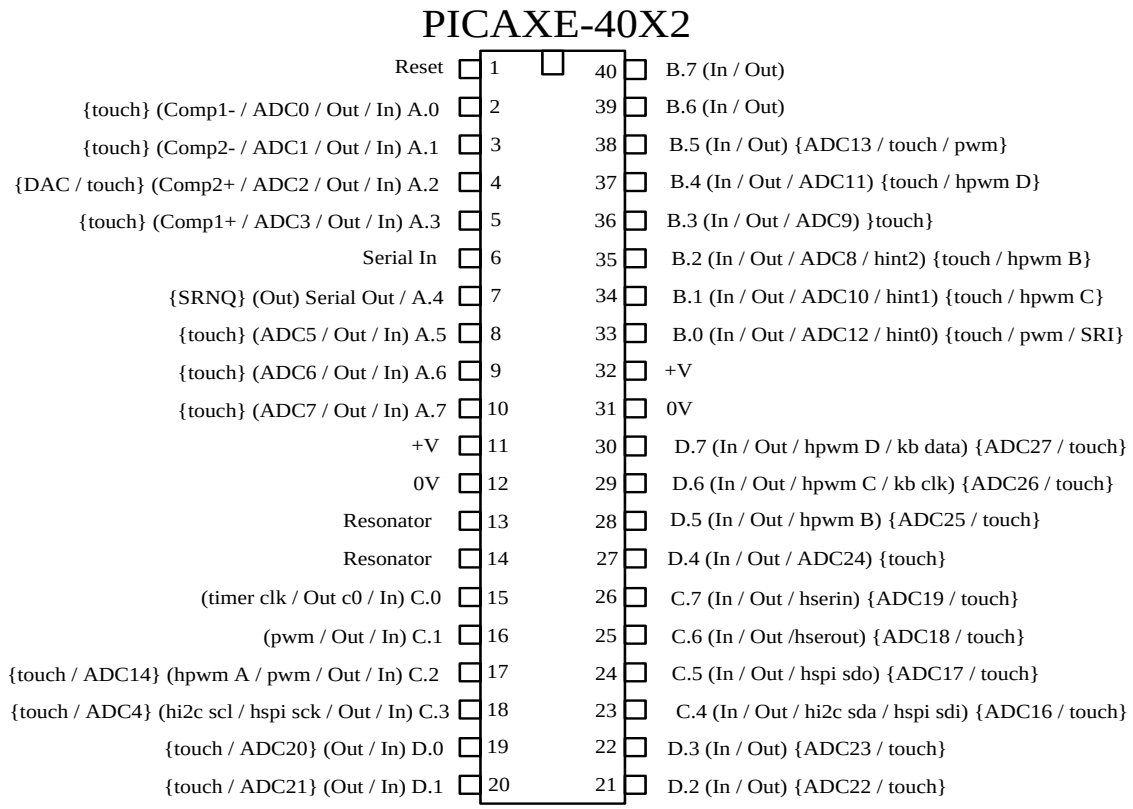
No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1						Reset	
2	In A.0		ADC 0			ULPWU	
3	In A.1		ADC 1				
4	In A.2		ADC 2				
5	In A.3		ADC 3				
6						Serial In	
7						Serial Out	
8			ADC 5				
9			ADC 6				
10			ADC 7				
11							+V
12							0V
13						Resonator	
14						Resonator	
15	In C.0	Out C.0				Timer clk	
16	In C.1	Out C.1			PWM 1		
17	In C.2	Out C.2			PWM 2		
18	In C.3	Out C.3				Spi sck/I2c scl	
19	In 0						
20	In 1						
21	In 2						
22	In 3						
23	In C.4	Out C.4				I2c sda/spi sdi	
24	In C.5	Out C.5				Spi sdo	
25	In C.6	Out C.6				Hserout	
26	In C.7	Out C.7				Hserin	
27	In 4						
28	In 5						
29	In 6					Kb clk	
30	In 7					Kb data	
31							0V
32							+V
33		Out 0					
34		Out 1					
35		Out 2					
36		Out 3					
37		Out 4					
38		Out 5					
39		Out 6					
40		Out 7					

Table 18. The detailed specification for pinout of the PICAXE-40X1 (44 Pin TQFP)

No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1	In C.7	Out C.7				Hserin	
2	In 4						
3	In 5						
4	In 6					Kb clk	
5	In 7					Kb data	
6							0V
7							+V
8		Out 0					
9		Out 1					
10		Out 2					
11		Out 3					
12	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-
14		Out 4					
15		Out 5					
16		Out 6					
17		Out 7					
18						Reset	
19	In A.0		ADC 0			ULPWU	
20	In A.1		ADC 1				
21	In A.2		ADC 2				
22	In A.3		ADC 3				
23						Serial In	
24						Serial Out	
25			ADC 5				
26			ADC 6				
27			ADC 7				
28							+V
29							0V
30						Resonator	
31						Resonator	
32	In C.0	Out C.0				Timer clk	
33	-	-	-	-	-	-	-
34	-	-	-	-	-	-	-
35	In C.1	Out C.1			PWM 1		
36	In C.2	Out C.2			PWM 2		
37	In C.3	Out C.3				2c scl/spi sck	
38	In 0						
39	In 1						
40	In 2						
41	In 3						
42	In C.4	Out C.4				I2c sda/spi sdi	
43	In C.5	Out C.5				Spi sdo	
44	In C.6	Out C.6				hserout	

The detailed pinout diagrams for the PICAXE-40X1 and PICAXE-40X1 (44 Pin TQFP) chips with the original PICAXE pin names are shown in Figures 11 [4] and presented in Tables 19and 20, respectively.

a)



b)

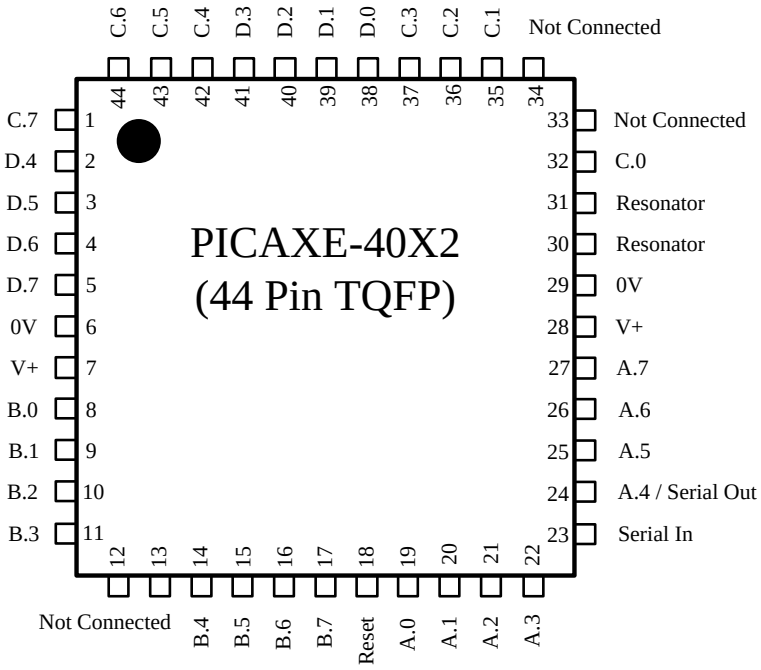


Fig. 11. Pinout diagram of a) the PICAXE-40X1, b) the PICAXE-40X1 (44 Pin TQFP)

Table 19. The detailed specification for pinout of the PICAXE-40X2

No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1						Reset	
2	In A.0	Out A.0	ADC 0			Touch/Comp1-	
3	In A.1	Out A.1	ADC 1			Touch Comp2-	
4	In A.2	Out A.2	ADC 2	DAC		Touch/Comp2+	
5	In A.3	Out A.3	ADC 3			Touch/Comp1+	
6						Serial In	
7		Out A.4				SRNQ/Serial Out	
8	In A.5	Out A.5	ADC 5			Touch	
9	In A.6	Out A.6	ADC 6			Touch	
10	In A.7	Out A.7	ADC 7			Touch	
11							+ V
12							0 V
13						Resonator	
14						Resonator	
15	In C.0	Out C.0				Timer clk	
16	In C.1	Out C.1			C.1		
17	In C.2	Out C.2	ADC 14		C.2	Touch/HPWM A	
18	In C.3	Out C.3	ADC 4			Touch/hi2c scl/hspi sck	
19	In D.0	Out D.0	ADC 20			Touch	
20	In D.1	Out D.1	ADC 21			Touch	
21	In D.2	Out D.2	ADC 22			Touch	
22	In D.3	Out D.3	ADC 23			Touch	
23	In C.4	Out C.4	ADC 16			Touch/hi2c sda/hspi sdi	
24	In C.5	Out C.5	ADC 17			Touch/hspi sdo	
25	In C.6	Out C.6	ADC 18			Touch/hserout	
26	In C.7	Out C.7	ADC 19			Touch/hserin	
27	In D.4	Out D.4	ADC 24			Touch	
28	In D.5	Out D.5	ADC 25			Touch/HPWM B	
29	In D.6	Out D.6	ADC 26			Touch/HPWM C/kb clk	
30	In D.7	Out D.7	ADC 27			Touch/HPWM D/kb data	
31							0 V
32							+ V
33	In B.0	Out B.0	ADC 12		B.0	Touch/SRI/hint0	
34	In B.1	Out B.1	ADC 10			Touch/HPWM C/hint1	
35	In B.2	Out B.2	ADC 8			Touch/HPWM B/hint2	
36	In B.3	Out B.3	ADC 9			Touch	
37	In B.4	Out B.4	ADC 11			Touch/HPWM D	
38	In B.5	Out B.5	ADC 13		B.5	Touch	
39	In B.6	Out B.6					
40	In B.7	Out B.7					

Table 20. The detailed specification for pinout of the PICAXE-40X2 (44 Pin TQFP)

No	Digital		Analog		PWM	Other	Vcc
	inputs	outputs	inputs	outputs			
1	In C.7	Out C.7				Hserin	
2	In 4						
3	In 5						
4	In 6					Kb clk	
5	In 7					Kb data	
6							0V
7							+V
8		Out 0					
9		Out 1					
10		Out 2					
11		Out 3					
12	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-
14		Out 4					
15		Out 5					
16		Out 6					
17		Out 7					
18						Reset	
19	In A.0		ADC 0			ULPWU	
20	In A.1		ADC 1				
21	In A.2		ADC 2				
22	In A.3		ADC 3				
23						Serial In	
24						Serial Out	
25			ADC 5				
26			ADC 6				
27			ADC 7				
28							+V
29							0V
30						Resonator	
31						Resonator	
32	In C.0	Out C.0				Timer clk	
33	-	-	-	-	-	-	-
34	-	-	-	-	-	-	-
35	In C.1	Out C.1			PWM 1		
36	In C.2	Out C.2			PWM 2		
37	In C.3	Out C.3				2c scl/spi sck	
38	In 0						
39	In 1						
40	In 2						
41	In 3						
42	In C.4	Out C.4				I2c sda/spi sdi	
43	In C.5	Out C.5				Spi sdo	
44	In C.6	Out C.6				hserout	

The 28 - 40 (X, X1 and X2) chips have internal oscillators (4 or 8MHz) and the external oscillator is optional. The external oscillator allows to increase work frequency to 40/64 MHz. Some 28 X2 - 40 X2 chips are supplied with 1.8 V to 3.3 V. They have **28X2-3V** and **40X2-3V** engraved on the top factory. Other chips of the PICAXE family are universal and are supplied with 3V to 5V.

3. THE PICAXE PROGRAMMING EDITOR

The PICAXE Programming Editor is the software application for developing and simulating PICAXE family microprocessors in BASIC language under Windows [2]. The PICAXE Programming Editor has a full suite of code development features such as [2]:

- a full simulation with animated all PICAXE family microprocessors under Windows screen,
- syntax check,
- program download without a programming device,
- line by line code highlighting,
- simulation breakpoints,
- testing tools for digital and analogue simulations.
- debug windows,
- serial terminal window.

After pressing the F1 key in the Check Syntax menu, you can check which of the PICAXE chip family can use any commands during programming the PICAXE microprocessors. In Figure 12 the example of using the Check Syntax menu is presented (the command *high portc* can be used by the PICAXE-14M, -28X1 and 40X1 chips).

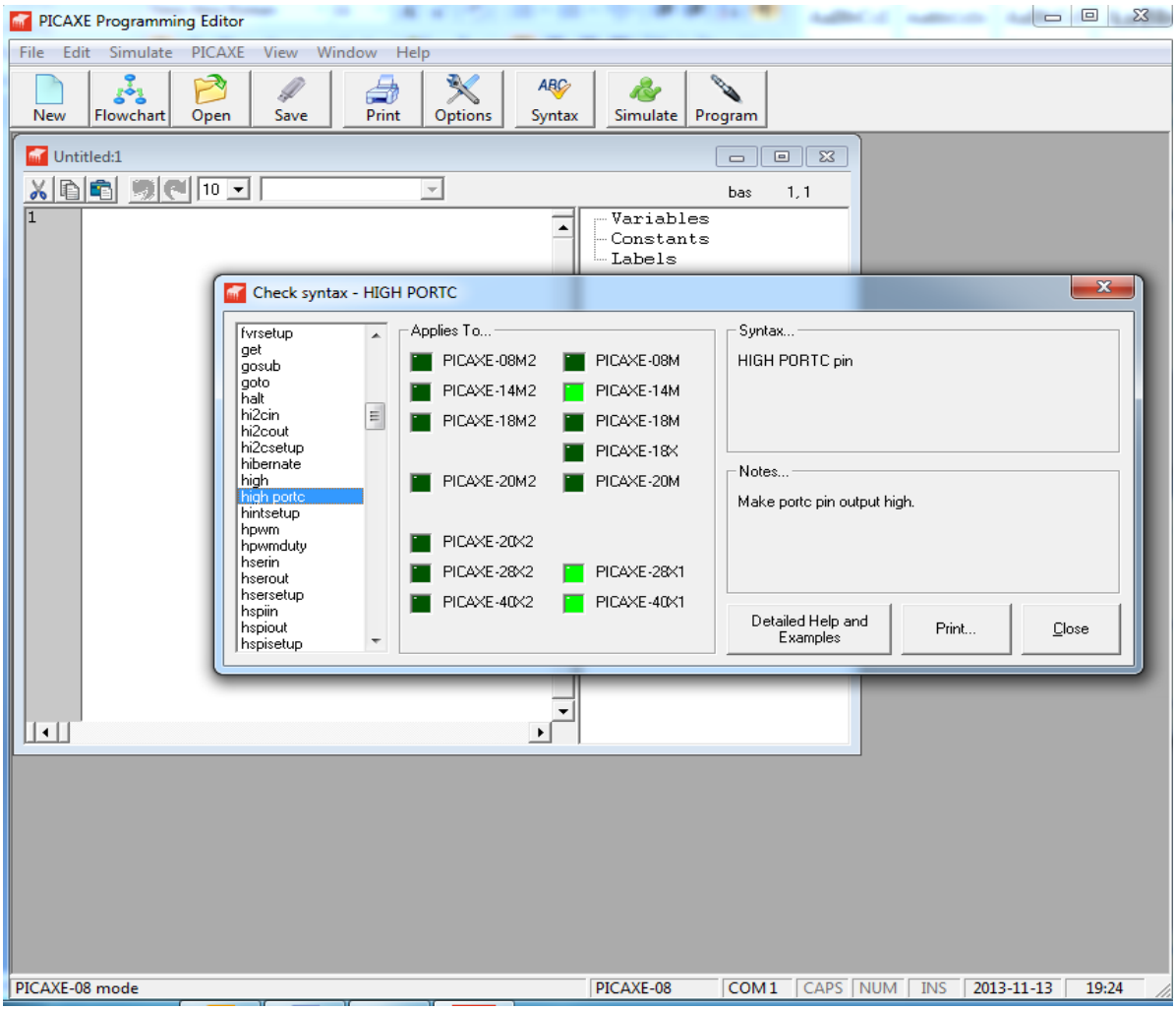
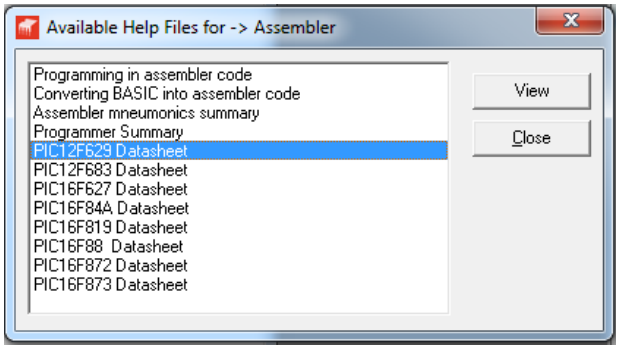


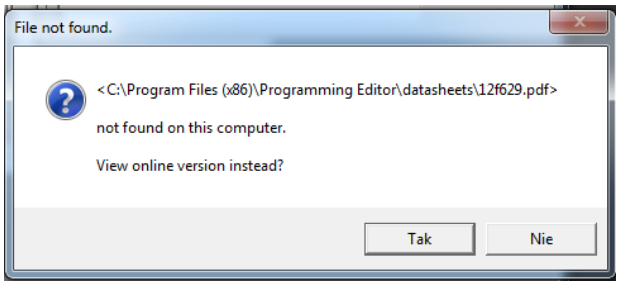
Fig. 12. The example of using the Check Syntax menu (F1 key)

Apart from Basic language, the PICAXE Programming Editor is the software application for programming all PICAXE family microprocessors in the Assembler language under Windows. The possibility of programming some chips from the PICAXE family in the Assembler code is presented in Table 2. If you want to download suitable datasheets for some chips of PICAXE family, open the Help menu and next open the Assembler datasheets.... and next choose a suitable PIC microcontroller. Press View and next press the "Tak" key. If the PICAXE Programming Editor could not find the detailed Datasheet (in PDF format) on your computer you can download it from the Internet. In Figure 13 it is shown how to download suitable datasheets for some chips of PIC microcontrollers.

a)



b)



c)

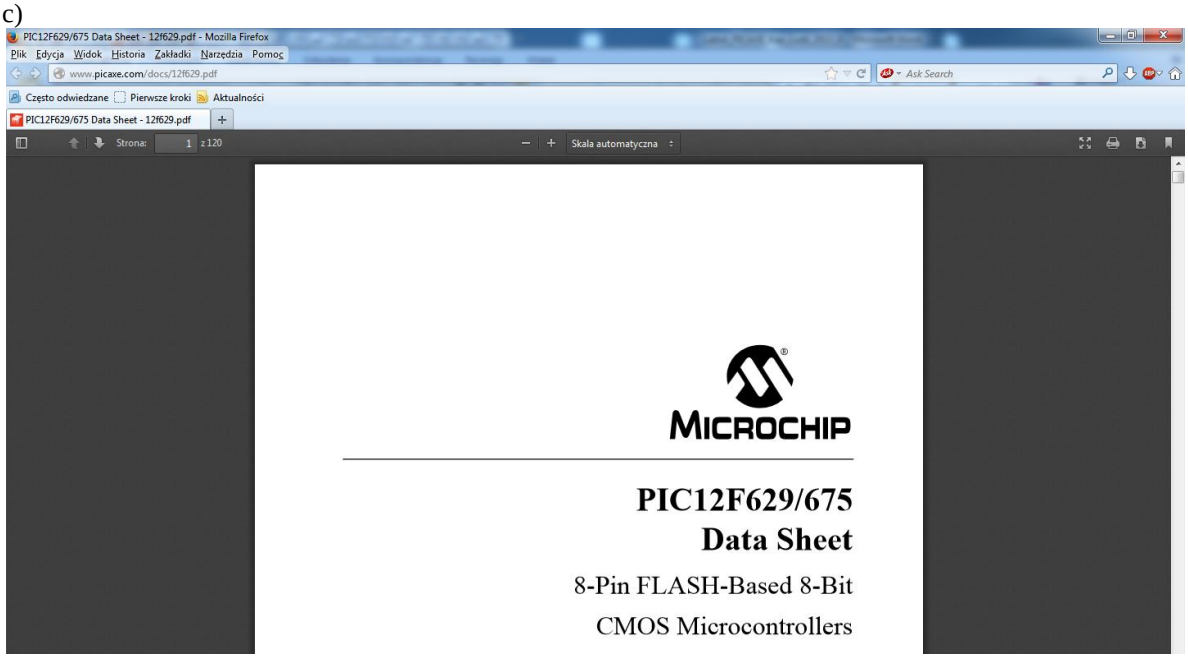


Fig.13. How to download suitable datasheets for some chips of PIC microcontrollers for programming in the Assembler language a) choosing the Assembler datasheets, b) downloding suitable datasheets from the Internet, c) view online - the datasheets for PIC12F629/675

Figure 14 shows nine tags of the Picaxe Programming Editor and their meanings.



New - allows to start a new project.



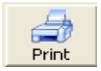
Flowchart - allows to build the program in a graphical way (Fig. 15).



Open - allows to open the existing program from a specified directory.



Save - allows to save a program.



Print - allows to print a program.



Options - selects the type of the PICAXE microprocessor (Fig. 16).



ABC-Syntax - checks the program validity (Fig. 18).



Simulate - can simulate using the program (Fig. 19).



Program - allows to download the project to the PICAXE microprocessor.

Figure 14. Tags of Picaxe Programming Editor

Fig. 15 shows the Tag Floechart – the graphical way of building a program in the PICAXE Programming Editor

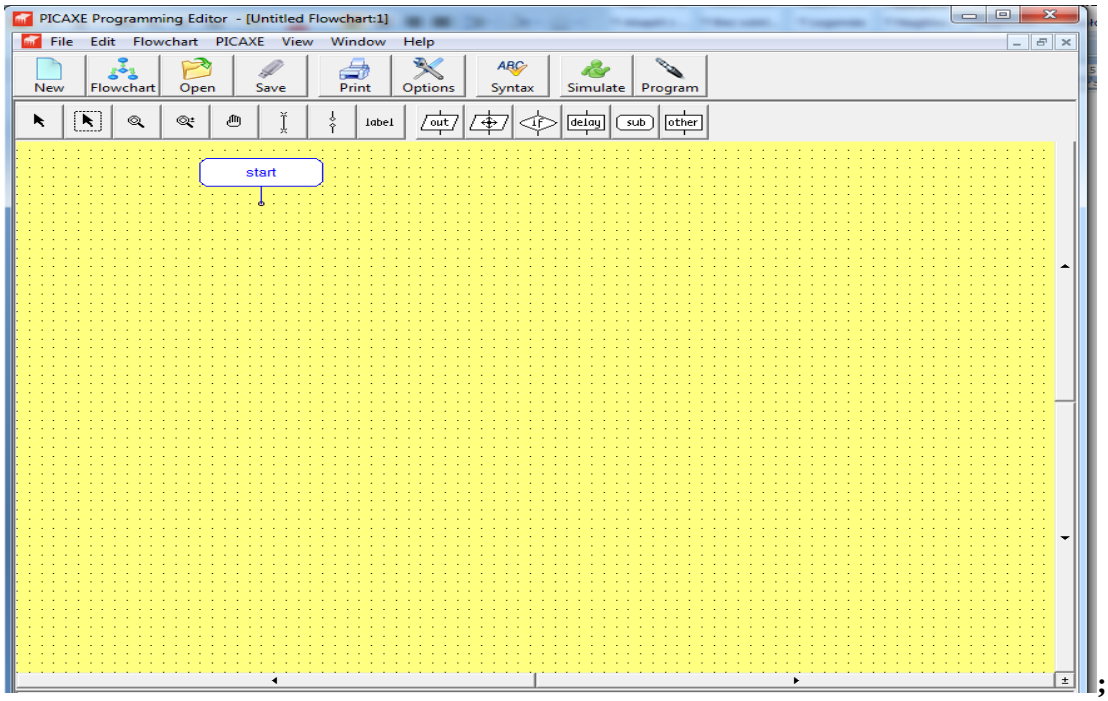


Fig. 15. Tag Floechart –the graphical way of building a program in the PICAXE Programming Editor

How to create and simulate a program in the Basic language in the Picaxe Programming Editor, will be explained in the following steps.

Step1. Starting the Picaxe Programming Editor and selecting a proper chip of the PICAXE family (Fig. 16).

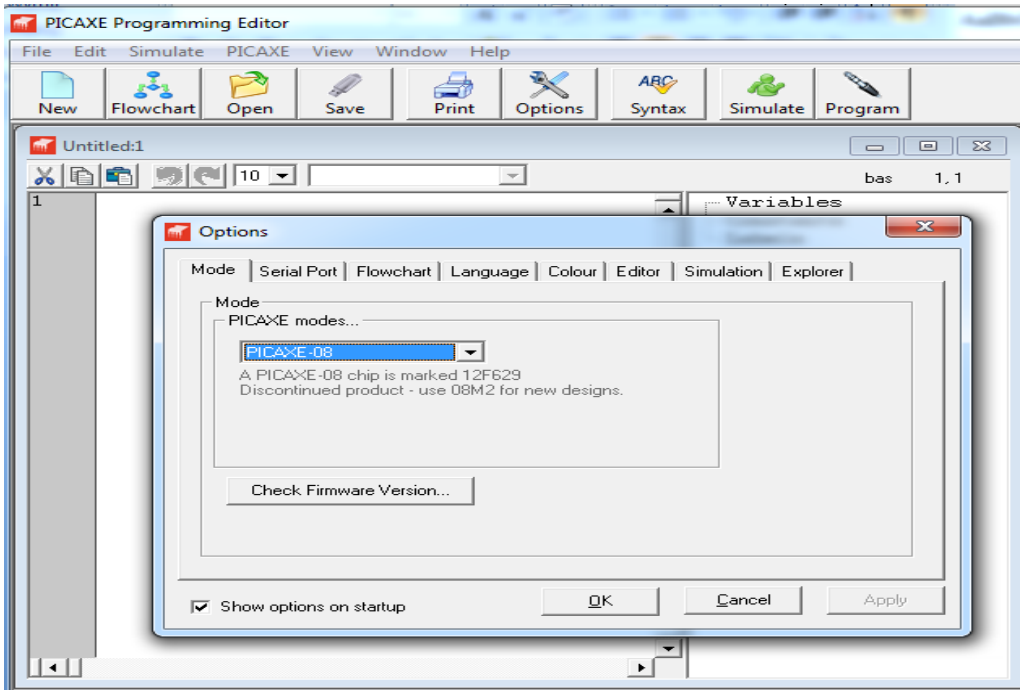


Fig. 16. Starting the Picaxe Programming Editor and selecting a proper chip of the PICAXE family

Step2. Choosing the communication port with PC, notebook, etc. In the case of using the USB-RS232 adapter, you should select a suitable COM... (Fig. 17).

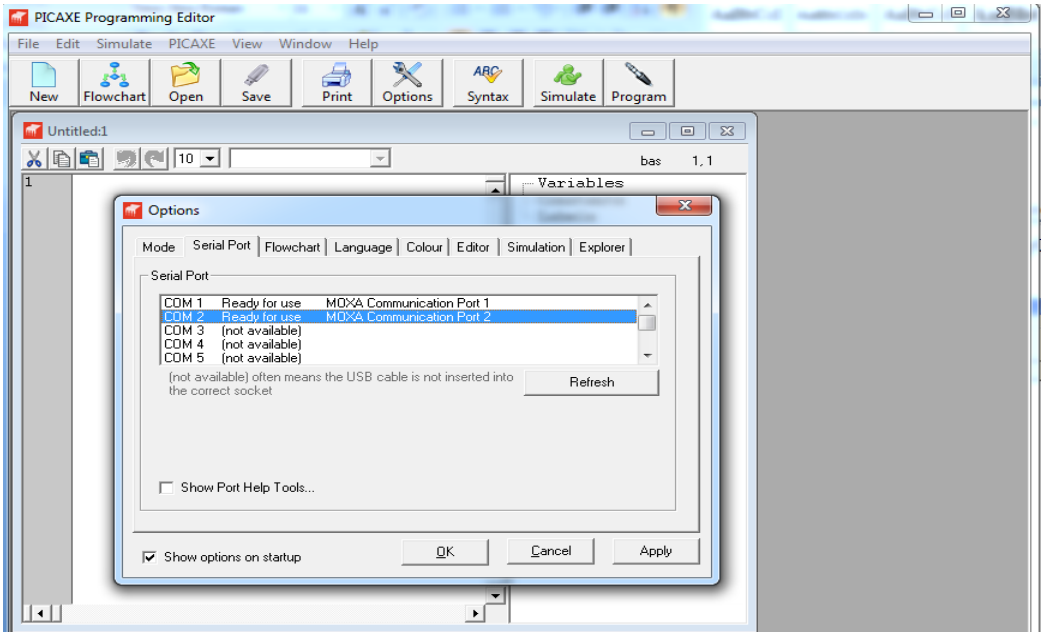


Fig. 17. Choosing the communication port with PC, notebook, etc.

Step3. ABC-Syntax tag checks the program validity (Fig. 18).

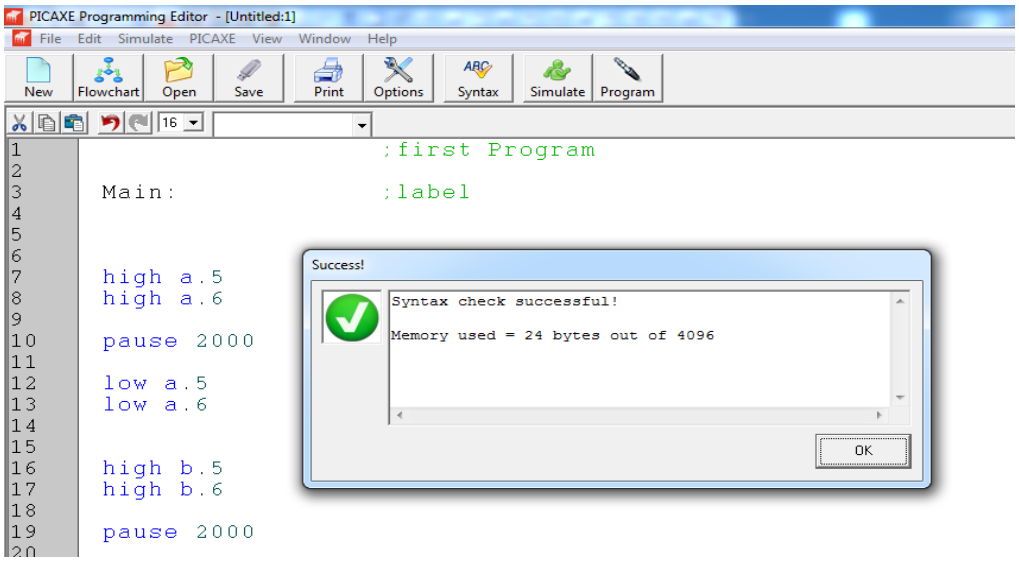


Fig. 18. ABC-Syntax tag checks the program validity

Step4. Simulating a program in the PICAXE Programming Editor (Fig. 19).

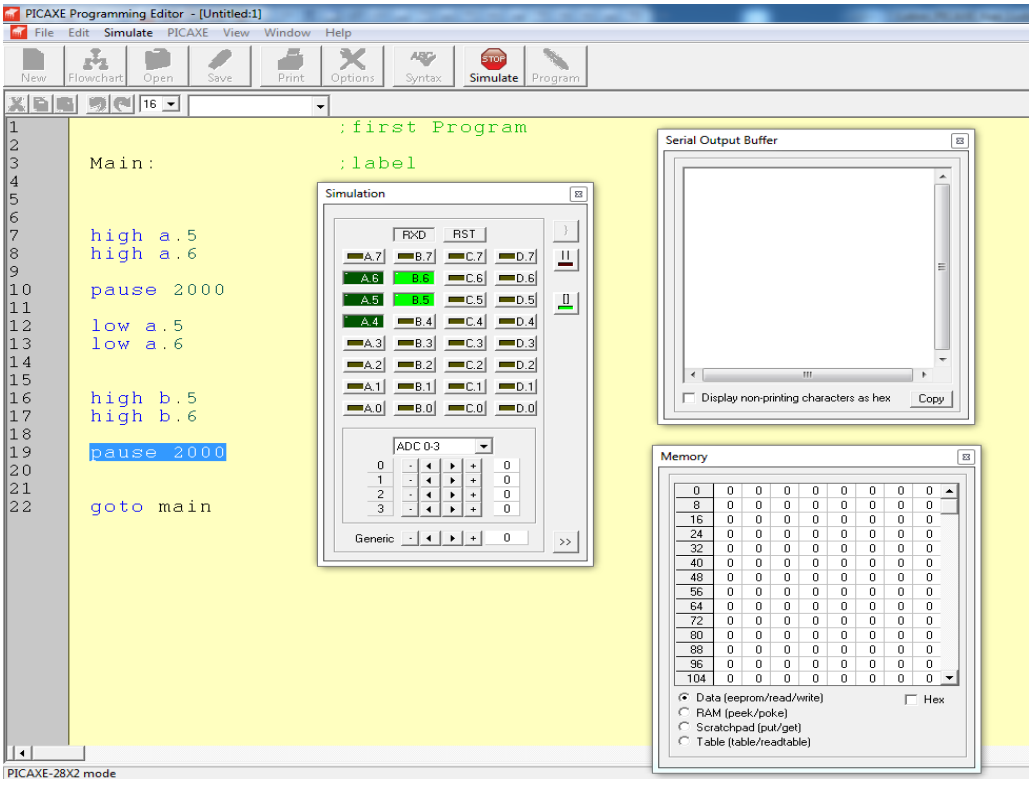


Fig. 19. Simulating a program in the PICAXE Programming Editor

Example. Write a program for the PICAXE-40X2 (Fig. 11 and Table 19) for lighting and extinguishing LEDs on ports A, B, C, and D for four different cases. The cases depend on the selection signals a.0, b.0, c.0, and d.0. Do the simulation of this program in the PICAXE Programming Editor.

```
start0:
    if pina.0=1 then start1
    if pinb.0=1 then start2
    if pinc.0=1 then start3
    if pind.0=1 then start4
start1:
    high a.7
    high b.7
    high c.7
        pause 1000
    low a.7
    high c.6
        pause 1000
    low b.7
    high c.5
        pause 1000
    low c.7
    high c.4
    high d.1
        pause 1000
    low c.6
    high b.3
        pause 1000
    low c.5
    high a.3
        pause 1000
    low c.5
    high a.2
        pause 1000
    low c.4
    high a.1
        pause 1000
    low c.3
    high b.1
        pause 1000
    low b.3
    high c.1
        pause 1000
    low a.3
    high d.0
        pause 1000
    low a.2
        pause 1000
    low a.1
        pause 1000
    low b.1
        pause 1000
    low c.1
        pause 1000
    low d.1
goto start0
start2:
    high b.4
    pause 750
    low b.4
    high b.3
    pause 750
    low b.3
```

```
    high b.2
    pause 750
    low b.2
    high b.1
    pause 750
    low b.1
goto start0
start3:
    high c.4
    pause 500
    low c.4
    high c.3
    pause 500
    low c.3
    high c.2
    pause 500
    low c.2
    high c.1
    pause 500
    low c.1
goto start0
start4:
    high a.7
    high b.7
    high c.7
    high d.7
        pause 500
    high a.6
    high b.6
    high c.6
    high d.6
        pause 600
    high a.5
    high b.5
    high c.5
    high d.5
        pause 700
    high a.4
    high b.4
    high c.4
    high d.4
        pause 800
    high a.3
    high b.3
    high c.3
    high d.3
        pause 900
    high a.2
    high b.2
    high c.2
    high d.2
        pause 1000
    high a.1
    high b.1
    high c.1
    high d.1
goto start0
```

4. PRACTICAL UTILIZATION OF PICAXE IN CONTROLLING STEPPER AND DC MOTORS

4.1. STEPPER MOTORS

Stepper motors are an electromechanical device that converts electrical pulses (generator signals) into mechanical discrete movements [5]. Most stepper motors are located in hardware (printers, scanners, hard disc drives HDD, FDD - floppy disc drives, recorders, CDs and DVDs etc). In each computer set there are many stepper motors. Stepper motors have the following properties [5]:

- the speed is proportional to the frequency of the input pulses,
- repeatability of movement and precise positioning,
- the stepper motor in the positioning system can operate without feedback;
- the angular or linear distance between adjacent positions is constant and is called a step;
- by electronic methods or using the microprocessor system, a step can be divided into any number of microsteps.

4.1.1. Switching of stepper motor windings

Stepper motors are electronically commutated machines and work with pulse power from a DC source (controller or electronic commutator), with the stepping positioning of the magnetic field. Figure 20 shows a block diagram of the basic elements of stepper motor controlling. Both a stator and a rotor have poles. By pulsing the stator windings (phases) in a suitable sequence, it is possible to control the motor direction of movement and speed.

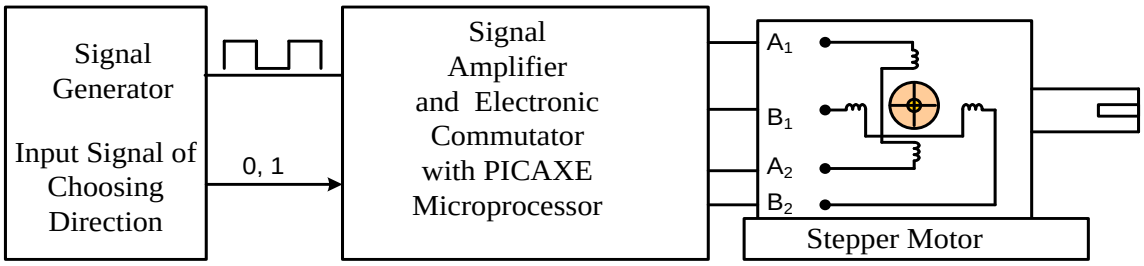


Fig. 20. Basic elements of a stepper motor controlling

Figure 2 shows the schematics of stator windings with shared and unshared windings (two phases). The two phases of stator windings have:

- unshared two single windings (4 terminals) - one winding/phase,
- four independent windings (8 terminals) - two windings/phase,
- with two windings connected in parallel (4 terminals) - two windings/phase,
- with two windings connected in series (6 pin) - two windings/phase.

Sometimes the unipolar stepper motors with four independent windings are named as a four phase motor [5].

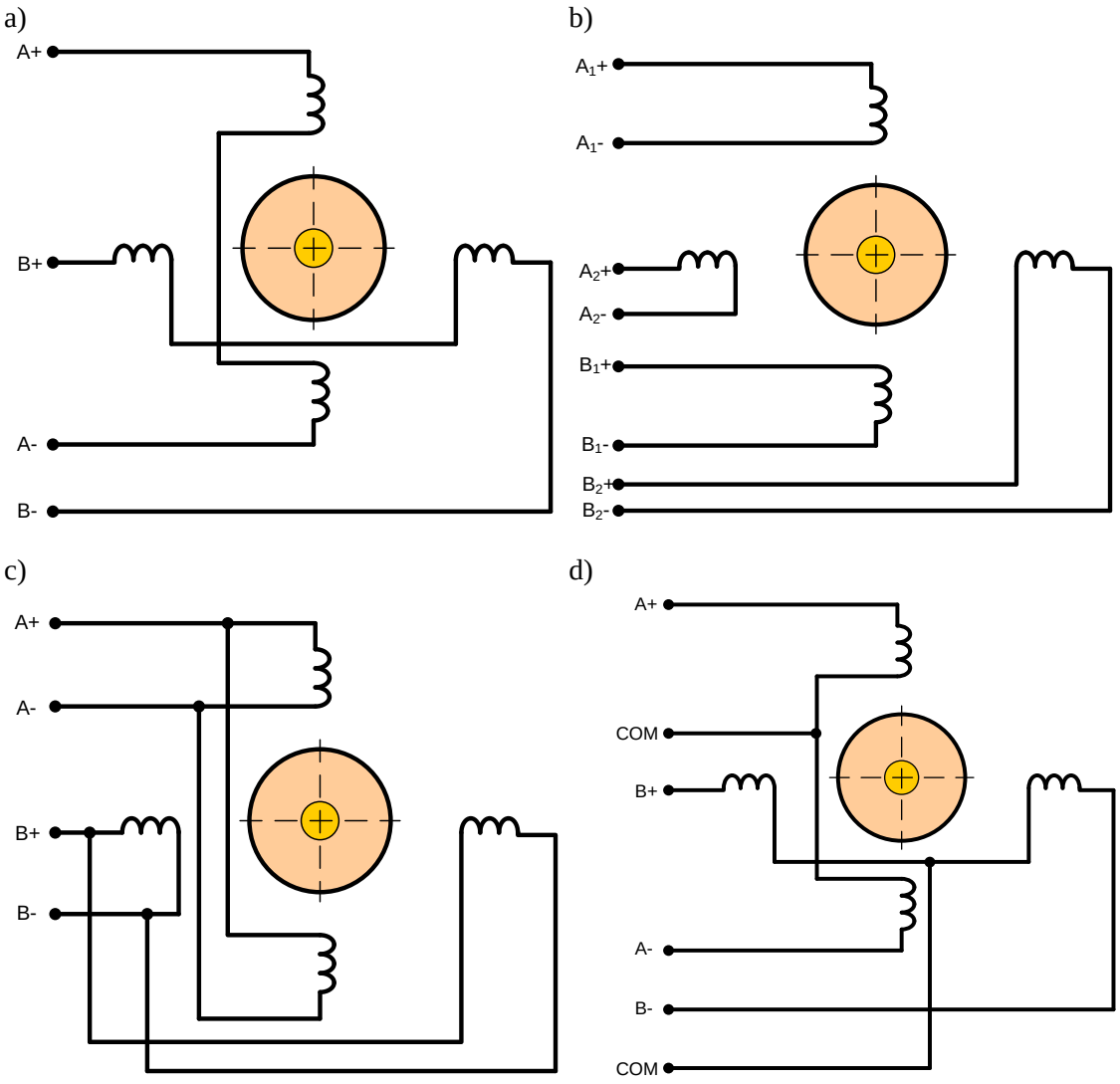


Fig. 21. Stepper motor stator windings a) with two unshared stator winding (4 terminals), b) four independent windings (8 terminals), c) with two windings connected in parallel (4 terminals), d) with two windings connected in series (6 pin)

The majority of stepper motors are built with two unshared stators winding called bipolar. Stepper motors with shared winding are called unipolar. Figure 22 shows a simplified construction of the variable and hybrid stepper motor [5].

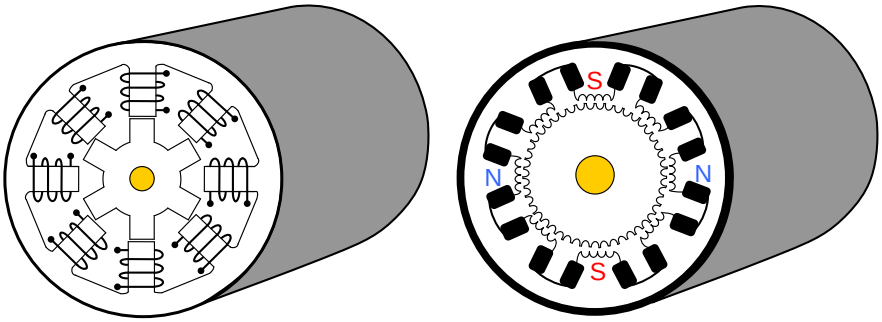


Fig. 22. Simplified construction of a) variable stepper motor , b) hybrid stepper motor

In Figure 23 a simplified construction of a rotor with permanent magnets is shown.

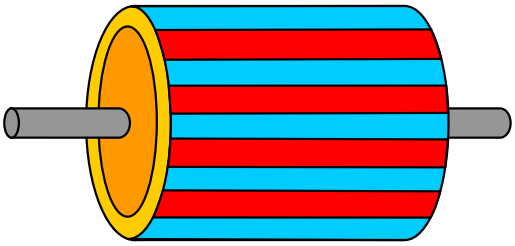


Fig. 23. Simplified construction of a rotor with permanent magnets

In Figure 24 a simplified way of controlling the stepper motor position is shown [5].

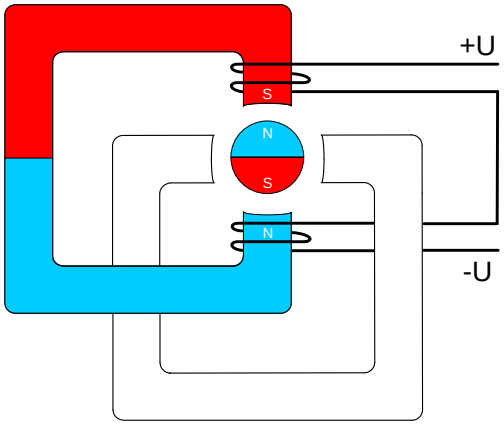
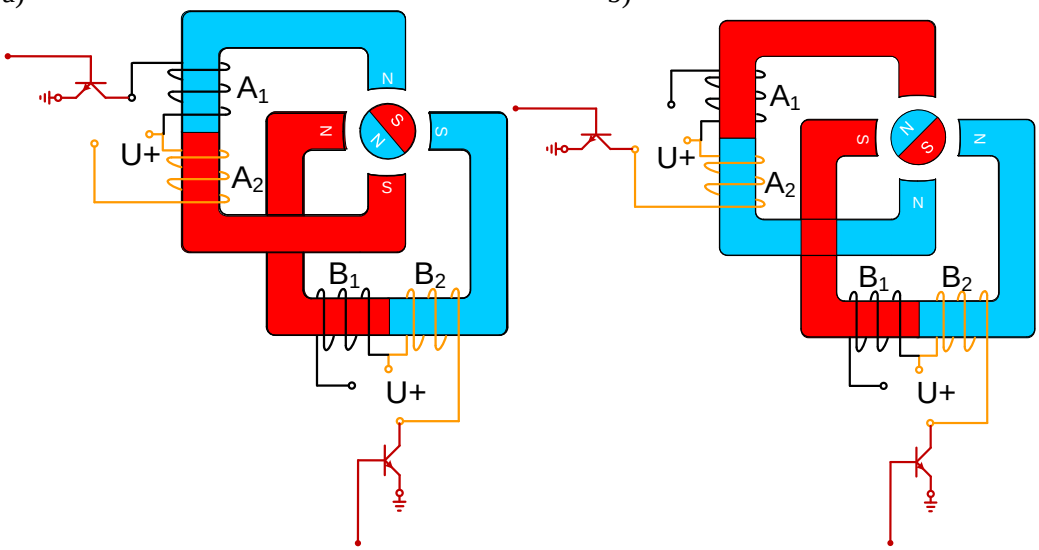


Fig. 24. Simplified way of controlling a stepper motor position

Figure 25 shows 4 steps how to control individual windings of the stepper motor [5].

a) b)



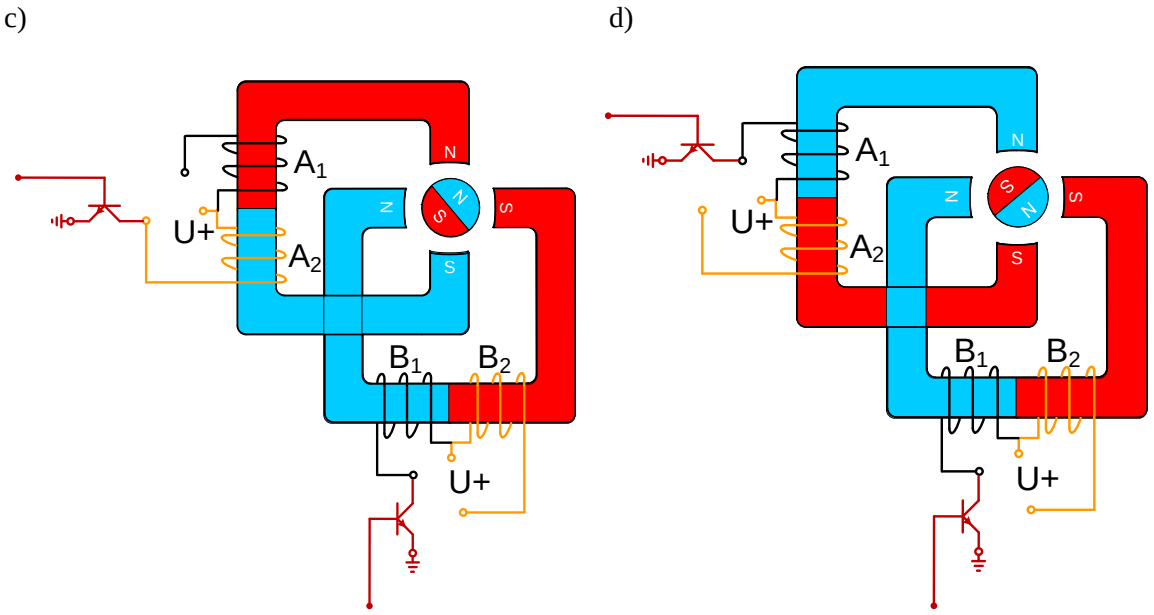


Fig. 25. Steps of controlling the individual stepper motor windings a) step 1, b) step 2, c) step 3, d) step 4

In Figure 26 the electrical circuit of a simplified way of controlling the unipolar stepper motor with four independent windings is shown.

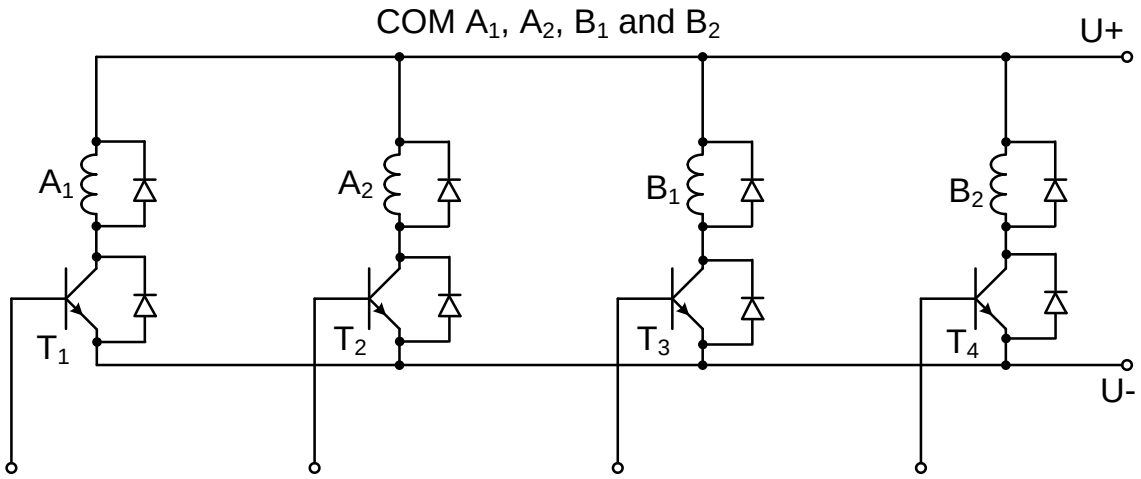


Fig. 26. The electrical circuit of a simplified way of controlling the unipolar stepper motor with four independent windings

In Figure 27 the electrical circuit of a simplified way of controlling the unipolar stepper motor with two windings connected in series per phase. The electrical circuit (Fig. 27) is similar to the circuit shown in Figure 26.

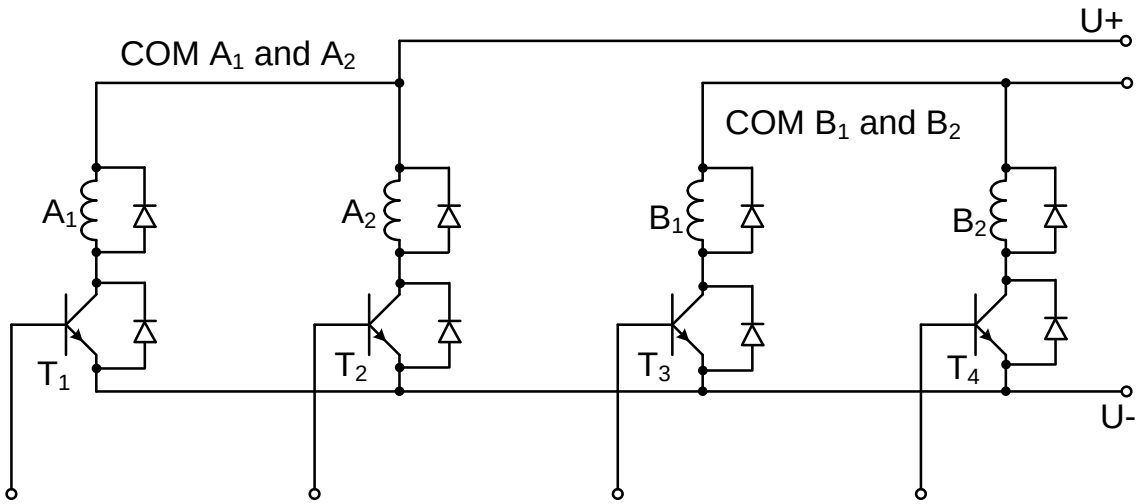


Fig. 27. Electrical circuit of a simplified way of controlling the unipolar stepper motor with two windings connected in series per phase

Figure 28 presents the electrical circuit of a simplified way of controlling the bipolar stepper motor.

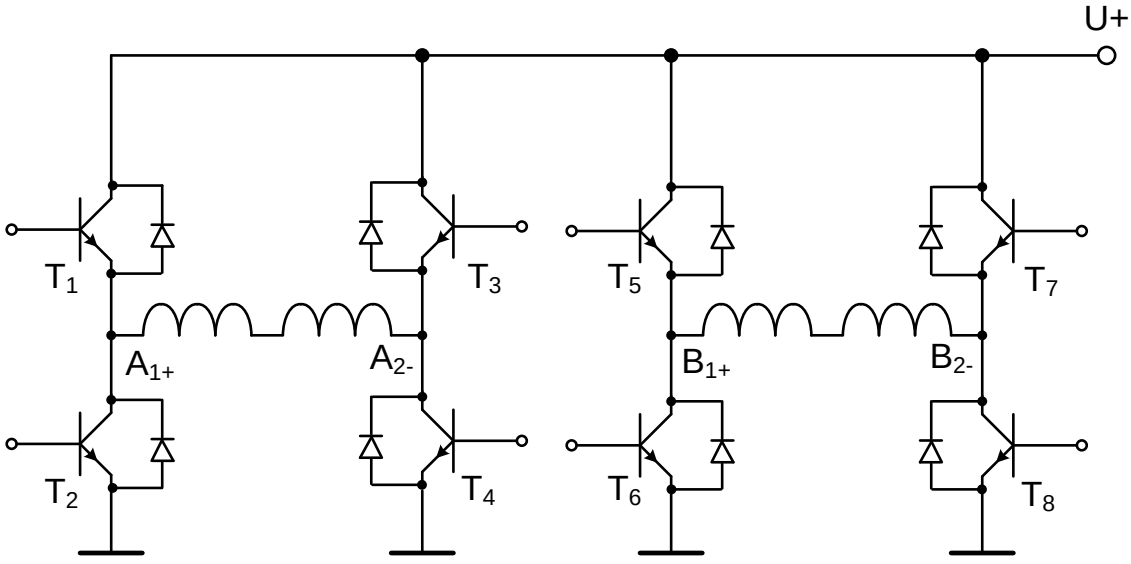


Fig. 28. Electrical circuit of a simplified way of controlling the bipolar stepper motor

4.2. MODES OF CONTROLLING THE STEPPER MOTOR

Generally there are four modes of controlling the stepper motor:

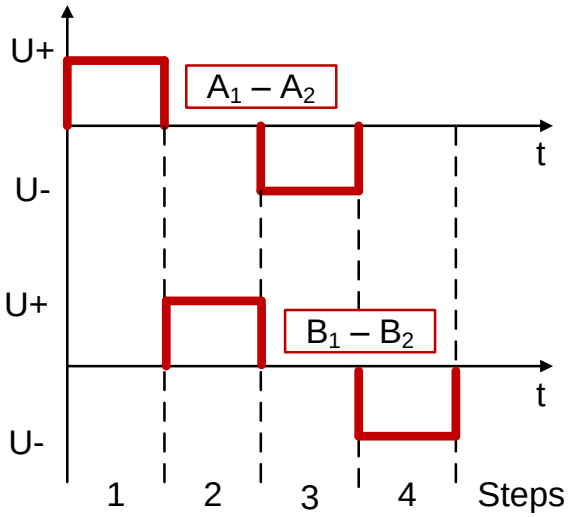
- wave drive (a one winding is on),
- full step drive (two windings are on),
- half step drive (a one winding or two windings are on),
- microstepping.

4.2.1. The wave drive

Both in bipolar and unipolar stepper motors (in the wave drive at the same time) only one phase is on. So, in unipolar stepper motors only 25 % of windings energized the magnetic circuit. In bipolar stepper motors only 50 % of windings (a one phase - two windings) energized magnetic circuit. It means that in the wave drive at any time it is not produced the maximum of electromagnetic torque. If we take into account the same magnetic dimensions and the same number of windings per pole, it means that the electromagnetic torque that is produced by the unipolar stepper motor is lower than in the case of the bipolar one.

Figure 29 shows states in which a bipolar and a unipolar stepper motor windings are supplied in the wave drive mode.

a)



b)

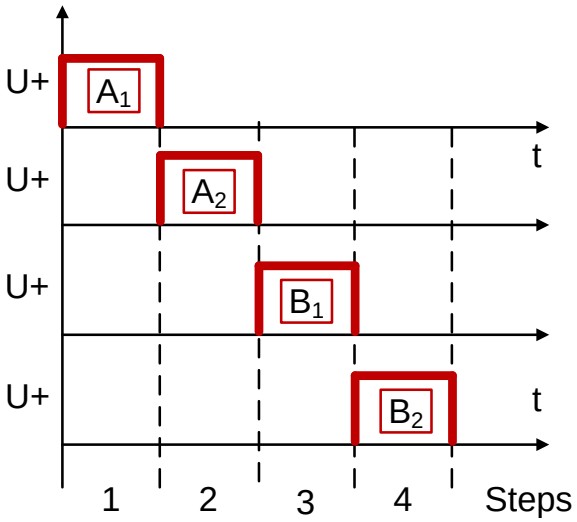


Fig. 29. Sequences for the wave drive mode for a) the bipolar stepper motor, b) the unipolar stepper motor

4.2.2. The full step drive

Both in bipolar and unipolar stepper motors (in the full step drive at any time) two windings are always on. So, in the unipolar stepper motors only 50 % of windings energized the magnetic circuit and in bipolar stepper motors 100 % of windings energized magnetic circuit. It means that in the full step drive at any time in the unipolar stepper motor it is not produced the maximum of electromagnetic torque.

Figure 30 shows the states in which a bipolar and a unipolar stepper motor windings are supplied in the wave drive mode.

a) b)

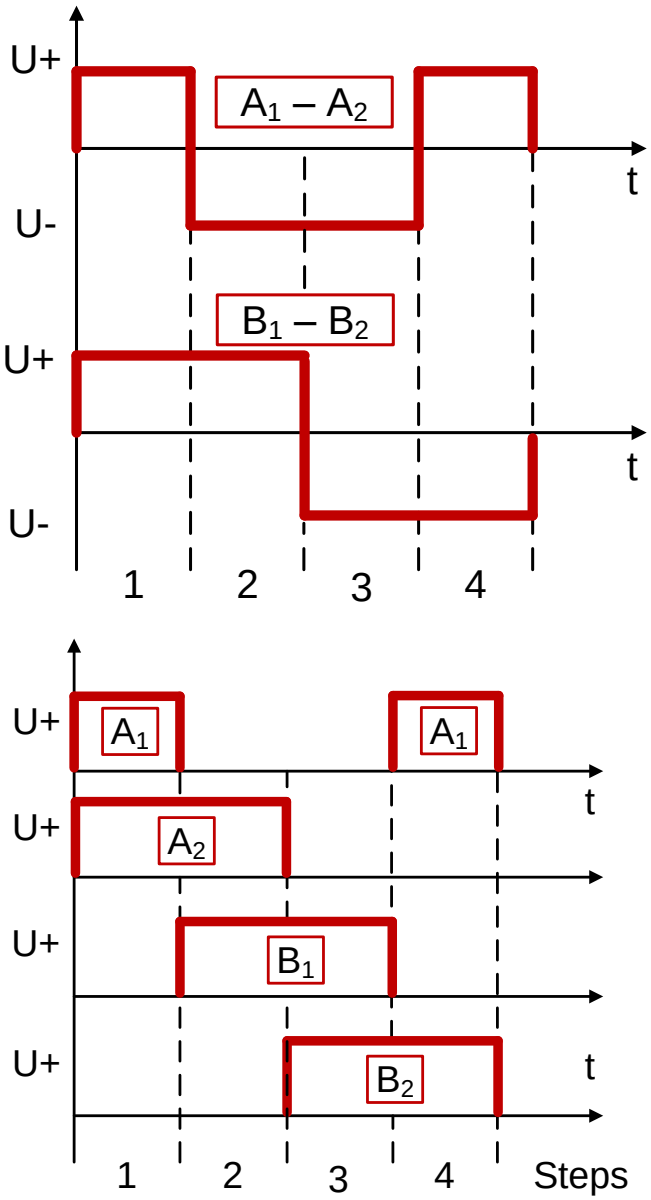


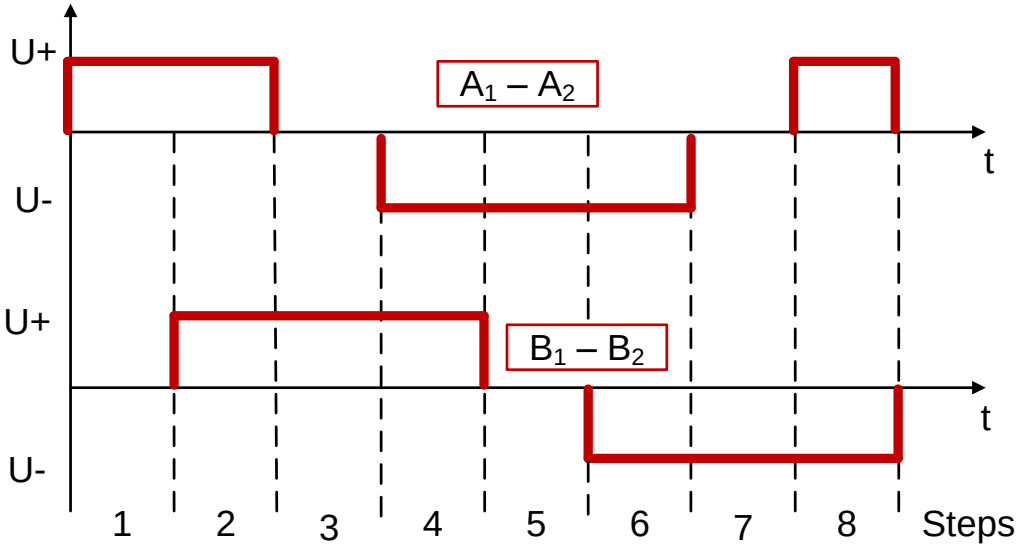
Fig. 30. Sequences for the full step drive mode for a) the bipolar stepper motor, b) the unipolar stepper motor

4.2.3. The half step drive

Both in bipolar and unipolar stepper motors (in the half step drive at any time) one or two windings are always on. So, the method is a combination of wave and full step drives, it means that one or two windings are always on. So, in unipolar stepper motors only 25 % or 50 % of windings energized the magnetic circuit, in bipolar stepper motors 50 % or 100 % of windings energized the magnetic circuit.

Figure 31 shows states in which bipolar and unipolar stepper motor windings are supplied in half step drive mode.

a)



b)

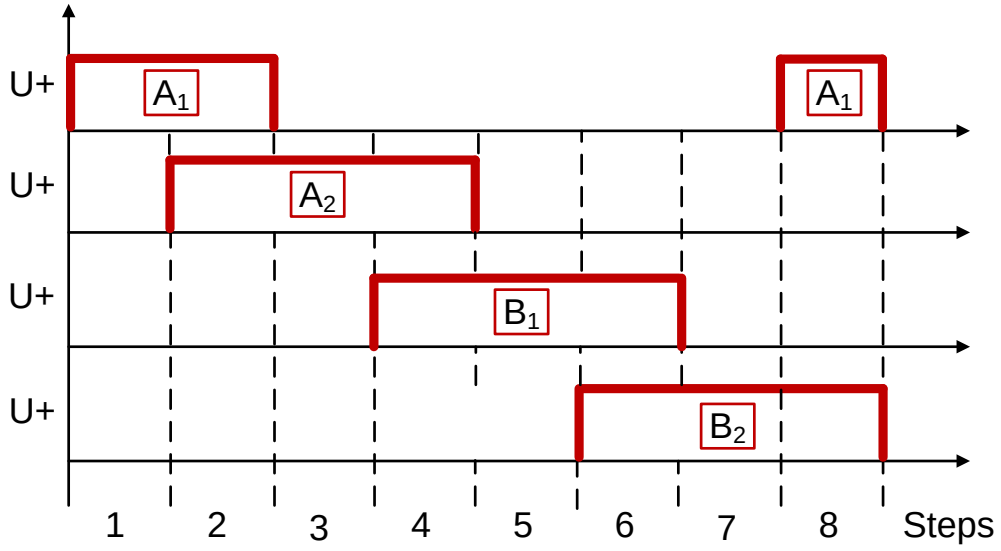


Fig. 31. Sequences for the half step drive mode for a) the bipolar stepper motor, b) the unipolar stepper motor

4.3. EXAMPLE OF STEPPER MOTOR CONTROLLING

Example 1. Write a program for the PICAXE-14M (Fig. 15 and Table 9) for continuous controlling the four-winding unipolar motor with shared windings and the wave drive control method of stepping.

```
; Continuous unipolar motor control: wave drive control, direction of rotation - right
loop:                ; start of loop
        high portc 4 ; high state: port C output 4
        pause 100    ; maintaining a high state: Port C output 4 for 0.1 s
        low portc 4  ; low state: port C output 4

        high portc 5 ; high state: port C output 5
        pause 100    ; maintaining a high state: port C output 5 for 0.1 s
        low portc 5  ; low state: port C output 5

        high portc 6 ; high state: port C output 6
        pause 100    ; maintaining a high state: Port C output 6 for 0.1 s
        low portc 6  ; low state: port C output 6

        high portc 7 ; high state: port C output 7
        pause 100    ; maintaining a high state for 0.1 s: port C output 7
        low portc 7  ; low state: port C output 7
goto loop            ; the end of the loop and go to the loop "loop"
```

Example 2. Write a program for the PICAXE-28X (Fig. 5 and Table 8) for a number of 15 cycles of controlling the four-windings unipolar motor and the full step control method of stepping. In this program the **For...next** loop should be used to repeat a section of the code.

```
begin :
        For  b1=1 to 15     ; For...next  - loop, the b1 is the most significant byte of
                            ; the w0 word
            high portc 4
            pause 150
            low portc 4

            high portc 5
            pause 150
            low portc 5
            high portc 6
```

```
            pause 150
            low portc 6

            high portc 7
            pause 150
            low portc 7

        next b1                 ; repeat a section between for ... next
    end
```

4.4. THE DC MOTOR

Every electric motor runs when a current-carrying through a conductor (windings) that is placed in an external magnetic field. The electromagnetic torque is proportional to the current in the winding and to the strength of the external magnetic field. In Figure 32 a simplified working principle of a DC motor is shown.

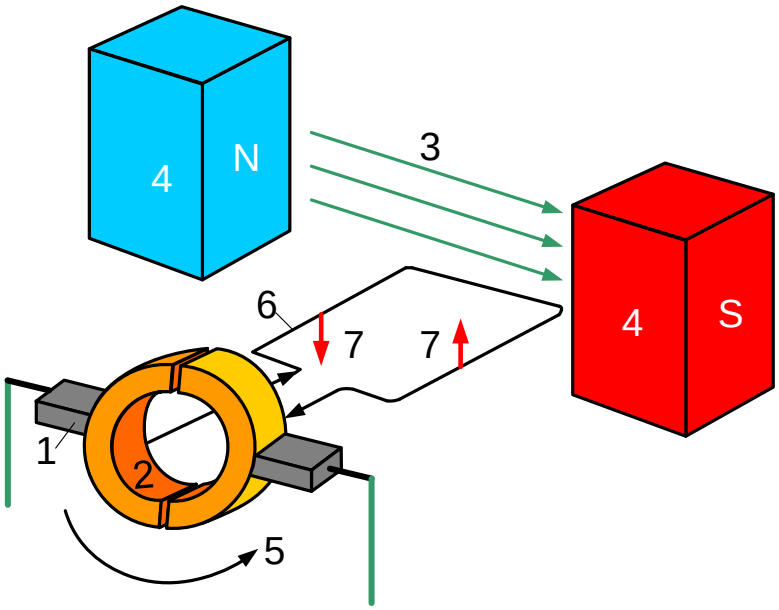


Fig. 32. Simplified working principle of a DC motor

In Figure 1 the individual elements of the DC motor represent:
1 – brush,
2 – mechanical commutator,
3 – magnetic flux,
4 – south and north poles,
5 – direction of movement,
6 – coil (winding) with current,
7 – supply.

Figure 33 shows how to control two DC motors. The electrical circuit enables independent changing of the direction of the rotational speed of two DC motors.

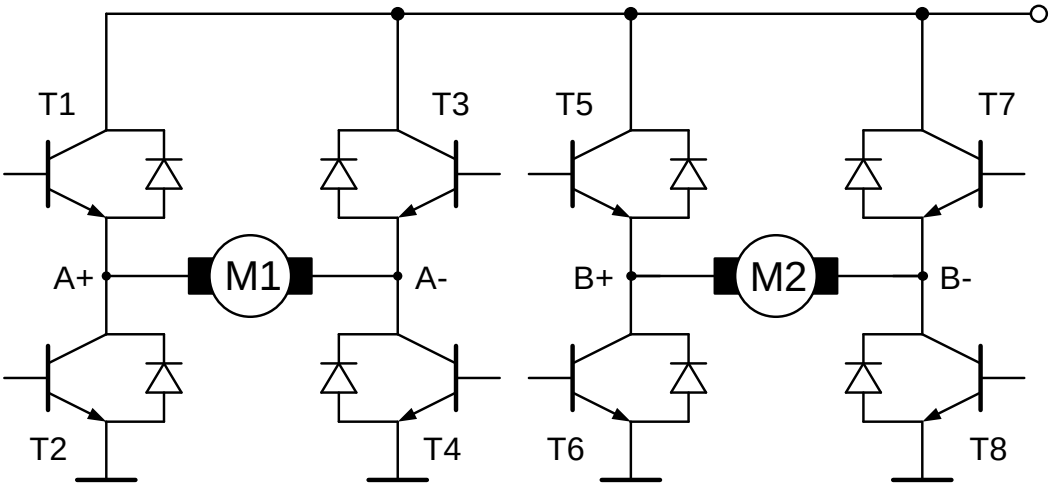


Fig. 33. Independent changing of the direction of the rotational speed of M1 and M2 DC motors

In the electrical circuit (Fig. 33), changing the direction of speed rotation of the DC motors can be achieved by any 18- 40 pin PICAXE microprocessor and a special IC high and low side driver (up to 600 V - for example IR2110 [6] or for low voltage up to 36 V and 0.6 A L293D [7] – Figs. 34 and 35). The four IC high and low side drivers can control double-four power transistors T1-T2 and T3 - T4 or T5-T6 and T7 - T8. Figure 32 illustrates the simple way of control of a DC motor is used to control the speed and change of moving a direction of the Buggy vehicle by the two DC motors [2]. Both the view on the Buggy vehicle and the location of the most important elements are shown in Figure 34 [2]. The vehicle uses the 18-pin PICAXE microprocessors (a detailed view with the description of the pins is shown in Fig. 7 and in the Tables 10 and 11).

Figure 34 shows a schematic control system that consists of two A and B DC motors powered by L293D, two A and B LEDs, two micro-switches (sensor A and B) and the PICAXE - 18M [4].

The control system (Figs. 34 and 35) consists of the following elements:

- 2 DC motors A and B
- 2 LEDs A and B
- piezo sounder
- 2 micro-switches (mechanical sensor A and B).

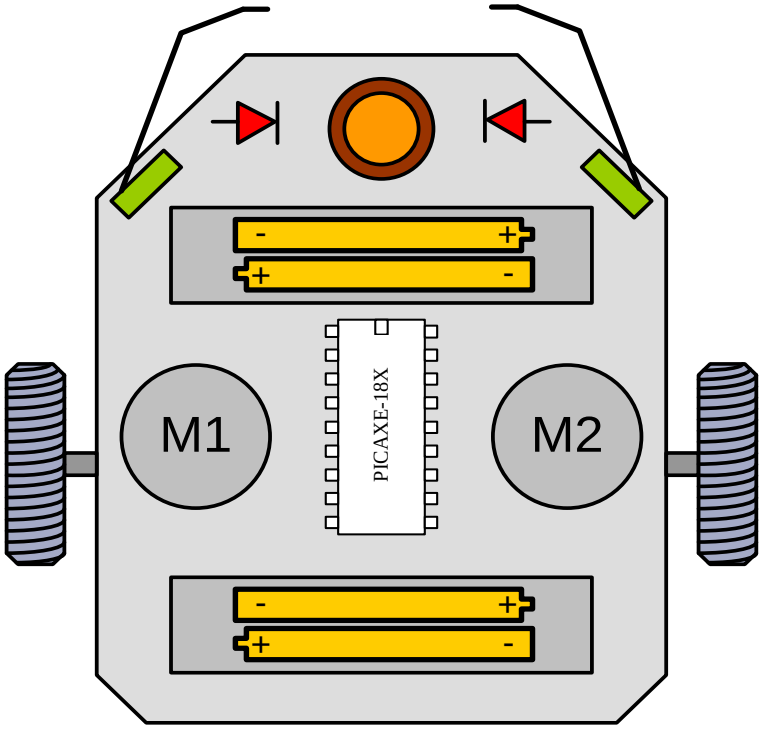


Fig. 34. View on the location of the most important elements of the Buggy vehicle

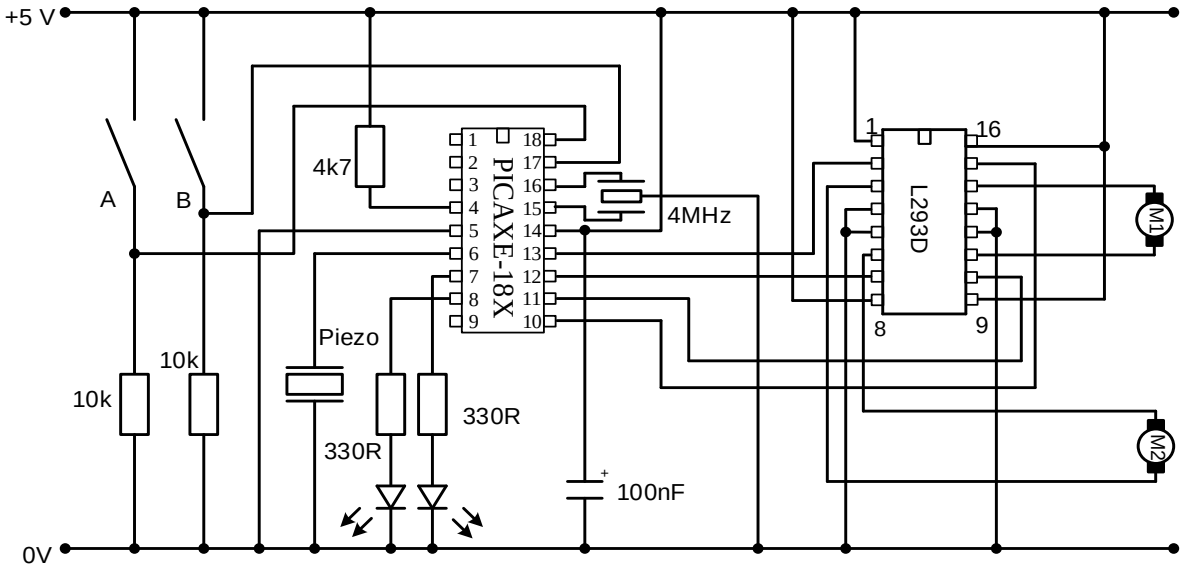


Fig. 35. Schema of the Buggy vehicle control system

Example 1. Write a program to control the Buggy vehicle using the PICAXE 18X chip.

```
SYMBOL LED_L = OUTPUT2          ; LED A
SYMBOL LED_P = OUTPUT1          ; LED B
SYMBOL SENSOR_L = INPUT1        ; micro-switch A
SYMBOL SENSOR_P = INPUT0        ; micro-switch B
```

```
symbol L_T   = output5          ;DC motor A -back
symbol L_P = output4                ;DC motor A -forward
symbol P_T   = output7          ; DC motor B -back
symbol P_P = output6                ; DC motor B -forward
symbol piezo = output0          ; piezo sounder

main:
      high L_P                  ; DC motor A is working - forward
      high P_P                  ; DC motor B is working - forward
    if SENSOR_L = 1 then left   ; if the Buggy hits an obstacle with the left arm of
                                ; the micro-switch A than jump to the "left" label
    if SENSOR_P = 1 then right  ; if the Buggy hits an obstacle with the right arm of
                                ; the micro-switch B then jump to the "right" label
      goto main                 ; jump to the "main" label

right:
      high LED_P                ; the LED B lights
      low LED_L                 ; the LED A is off
      gosub back                ; jump to the "back" label
      high L_T                  ; DC motor A is working - back
      high P_P                  ; DC motor B is working - forward
      pause 700                 ; wait 700 ms
      low L_T                   ; DC motor A is off
      pause 200                 ; wait 200 ms
      high L_P                  ; DC motor A is working - forward
      goto main                 ; jump to "main" label
left:
      high LED_L                ; the LED A lights
      low LED_P                 ; the LED B is off
      gosub back                ; jump to the "back" label,
      high P_T                  ; DC motor B is working - back
      high L_P                  ; DC motor A is working - forward
      pause 700                 ; wait 200 ms
      low P_T                   ; DC motor B is stopped
      pause 200                 ; wait 200 ms
      high P_P                  ; DC motor B is working - forward
      goto main                 ; jump to the "main" label
back:                           ; go back for 700 ms
      low L_P                   ; DC motor A is stopped
      low P_P                   ; DC motor B is stopped
      pause 200                 ; wait 200 ms
      high L_T                  ; DC motor A is working - back
      high P_T                  ; DC motor B is working - back
      sound piezo,(100,70)      ; generates soud for 700 ms
      low L_T                   ; DC motor A is stopped
      low P_T                   ; DC motor B is stopped
      pause 200                 ; wait 200 ms
      return
```

Example 2. Write a program to control the Buggy vehicle with the PICAXE-18M2 (Fig. 7 and Table 11) with utilization of LED signalization and piezo sounder.

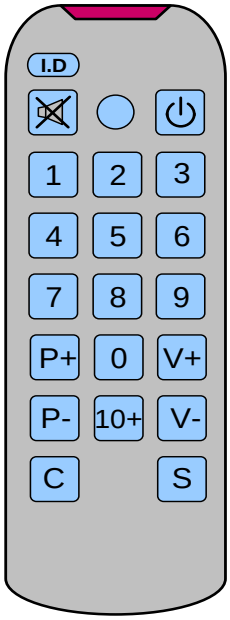
```
symbol C_L=INPUT2        ; micro-switch A
symbol C_P=INPUT7        ; micro-switch B
symbol PIEZO=B.0         ; piezo sounder
symbol LED_L=B.1         ; LED A
symbol LED_P=B.2         ; LED A
symbol L_P=B.4           ; DC motor A -forward
symbol L_T=B.5           ; DC motor A -back
symbol P_P=B.6           ; DC motor B -forward
symbol P_T=B.7           ; DC motor B -back
main:
        goto main2
main2:
        high L_P
        high P_P
        if C_L=1 then left
        if C_P=1 then right
        goto main2
left:
        high LED_L
        low L_P
        low P_P
        gosub bum
        high L_T
        high P_T
        pause 1000
        low L_T
        pause 500
        low P_T
        low LED_L
        goto main2

right:
        high LED_P
        low L_P
        low P_P
        gosub bum
        high L_T
        high P_T
        pause 1000
        low P_T
        pause 500
        low L_T
        low LED_P
        goto main2

bum:
tune 0, 1, ($40,$6B,$6A,$69,$68,$67,$66,$65,$64,$63,$62,$61,$E0,$C0,$E0,$C0,$E0)
return
```

4.5. THE IR REMOTE CONTROLS

Some PICAXE chips have the ability to control electronic devices using an infrared remote control. Figure 36 shows a view of the PICAXE Infra Red Remote Control (PIRRC) with the location of buttons and their descriptions [2, 4]. To remote control any device using the PIRRC it is required any infrared sensor (e.g. LED020 PICAXE infrared sensor).



Key	Value
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
P+	10
0	11
V+	12
P-	13
10+	14
V-	15
Mute	16
Power	17

Fig. 36. PICAXE IR remote control with the location of buttons and their descriptions

In Figure 37 a schema of the Buggy vehicle control system with the LED020 PICAXE infrared sensor (instead of two mechanical micro-switches A and B - Fig. 35) is shown.

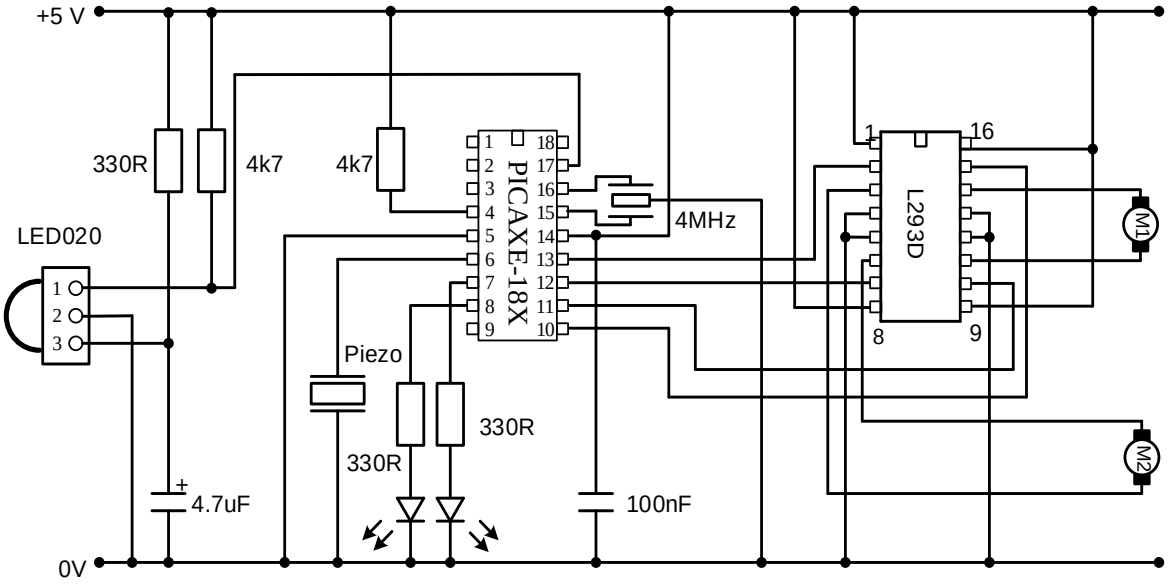
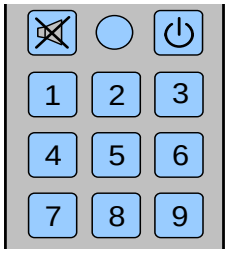


Fig. 37. Schema of the Buggy vehicle control system with the LED020 PICAXE infrared sensor

Example 1. Change the program from Example 1 to control the Buggy vehicle using the PICAXE 18X chip and the LED020 PICAXE infrared sensor (Fig. 37). In the program, use the following keys of the PICAXE Infra Red Remote Control:



2 - DC motor A and B move the Buggy vehicle forward,
4 - DC motor A and B move the Buggy vehicle left,
6 - DC motor A and B move the Buggy vehicle right,
8 - DC motor A and B move the Buggy vehicle back,
Power (value 17) - stop the Buggy vehicle.

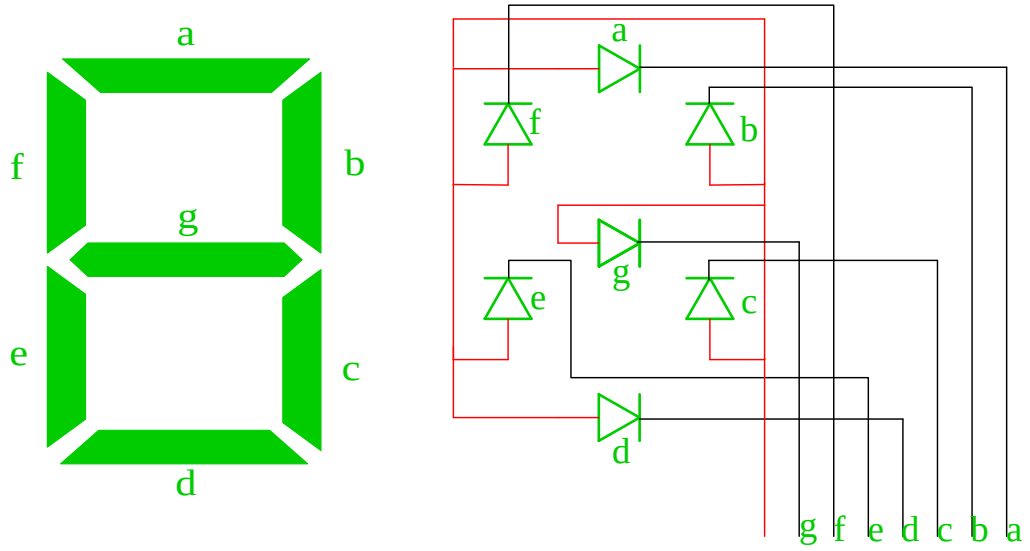
```
SYMBOL LED_L = OUTPUT2          ; LED A
SYMBOL LED_P = OUTPUT1          ; LED B
symbol L_T   = output5          ; DC motor A -back
symbol L_P = output4                    ; DC motor A -forward
symbol P_T   = output7          ; DC motor B -back
symbol P_P = output6                    ; DC motor B -forward
symbol piezo = output0          ; piezo sounder
main:
        if infra=2 than forward     ; DC motor A and B move the Buggy vehicle forward
        if infra=4 than left        ; DC motor A and B move the Buggy vehicle left
        if infra=6 than right       ; DC motor A and B move the Buggy vehicle right
        if infra=8 than back        ; DC motor A and B move the Buggy vehicle forward
        if infra=17 than stop       ; DC motor A and B are stopped
        goto main                   ; jump to the "main" label
forward:
        high L_P                    ; DC motor A is working - forward
        high P_P                    ; DC motor B is working - forward
        goto main                   ; jump to the "main" label
stop:
        low L_P                     ; DC motor A is stopped
        low P_P                     ; DC motor B is stopped
        low L_T                     ; DC motor A is stopped
        low P_T                     ; DC motor B is stopped
        goto main                   ; jump to the "main" label
```

```
right:
        high LED_P              ;the LED B lights
        low LED_L               ; the LED A is off
        high L_T                ; DC motor A is working - back
        high P_P                ; DC motor B is working - forward
        sound piezo,(100,70)    ; sounds for 700 ms
        pause 700               ; wait 700 ms
        low L_T                 ; DC motor A is off
        pause 200               ; wait 200 ms
        high L_P                ; DC motor A is working - forward
        goto forward            ; jump to the "forward" label
left:
        high LED_L              ; the LED A lights
        low LED_P               ; the LED B is off
        high P_T                ; DC motor B is working - back
        high L_P                ; DC motor A is working - forward
        sound piezo,(100,70)    ; sounds for 700 ms
        pause 700               ; wait 200 ms
        low P_T                 ; DC motor B is stopped
        pause 200               ; wait 200 ms
        high P_P                ; DC motor B is working - forward
        goto forward            ; jump to the "forward" label
back:                           ; go back for 700 ms
        low L_P                 ; DC motor A is stopped
        low P_P                 ; DC motor B is stopped
        pause 200               ; wait 200 ms
        high L_T                ; DC motor A is working - back
        high P_T                ; DC motor B is working - back
        sound piezo,(100,70)    ; sounds for 700 ms
        high L_T                ; DC motor A back
        high P_T                ; DC motor B back
        goto main               ; jump to the "main" label
```

4.6. PICAXE COOPERATION WITH THE SEVEN-SEGMENT DISPLAY

Very frequently microprocessors communicate with the user with various types of LED displays. One of the most popular is the seven-segment LED display. The structure and method of assembling and placing of each diode is shown in Figure 37. The even-segment displays (Fig. 37) consist of LED diodes, that are connected with a common anode or common cathode [8].

a)



b)

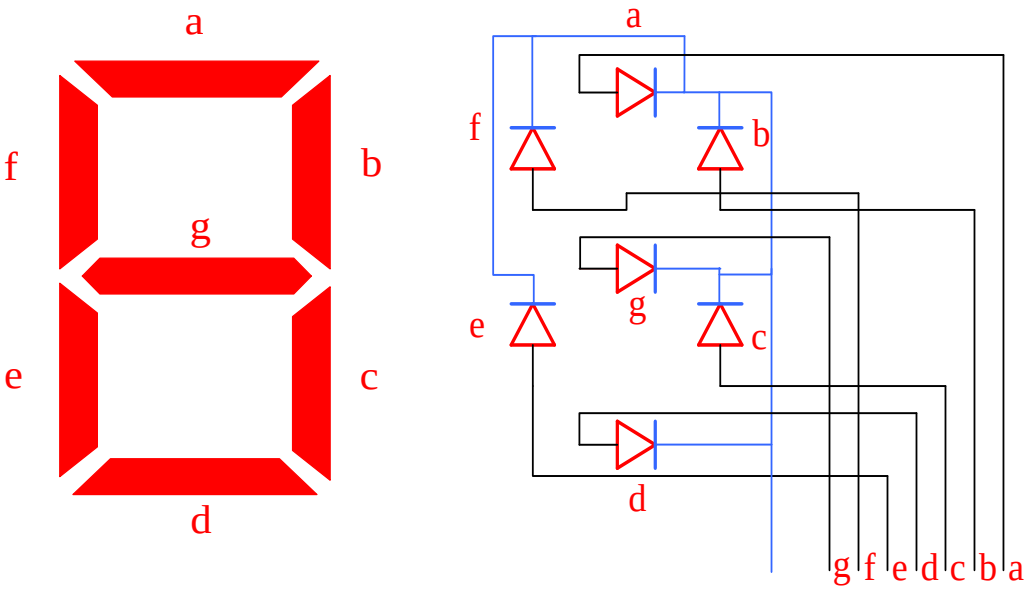


Fig. 37. Construction and method of connection and location of LED diodes in the seven-segment displays a) with common anode, b) with common cathode

Figure 38 shows a control system with four seven-segment LED displays, BCD to a seven-segment decoder (7447) and the PICAXE-40X2.

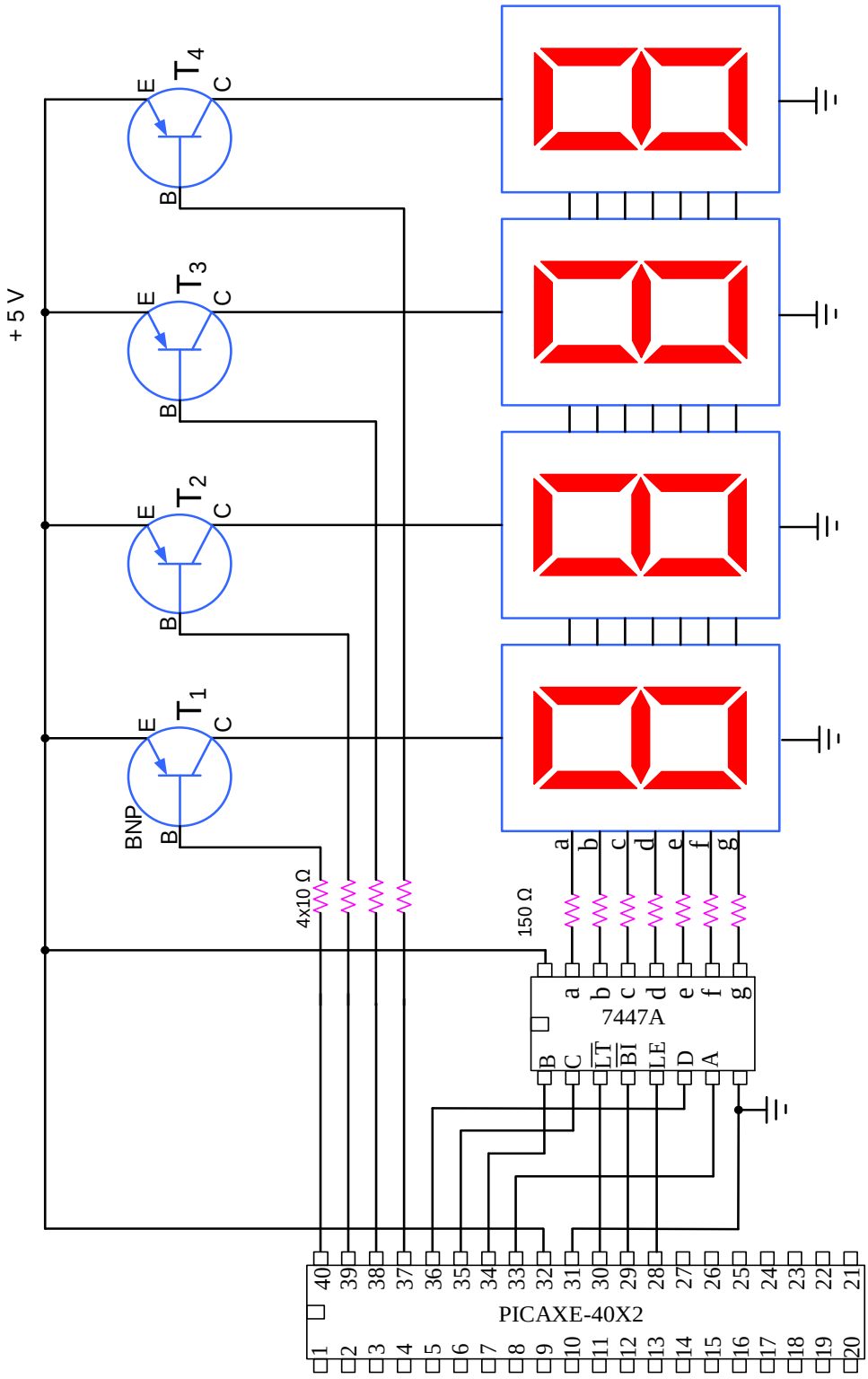


Fig. 38. Control system with four seven-segment LED displays, BCD to a seven-segment decoder (7447) and the PICAXE-40X2

A Truth table of inputs and outputs BCD to a seven-segment decoder is presented in Table 21.

Table 21. Truth table of inputs and outputs of the 7447 BCD to a seven-segment decoder

Inputs				Outputs							Display
D	C	D	C	a	b	c	d	e	f	g	
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	2
0	0	1	1	0	0	0	0	1	1	0	3
0	1	0	0	1	0	0	1	1	0	0	4
0	1	0	1	0	1	0	0	1	0	0	5
0	1	1	0	1	1	0	0	0	0	1	6
0	1	1	1	0	0	0	1	1	1	1	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	0	1	1	0	0	9

Task. 1. Using the PICAXE-40X2 (details is shown in Fig. 11 and presented in Table 19) and the truth table of inputs and outputs BCD to a seven-segment decode (Table 21), write an example program that displays the number 1234 on the four seven-segment LED displays (Fig. 38).

REFERENCES

1. T. Jabłoński, K. Pławsiuk: Programowanie mikrokontrolerów PIC w języku C. Wydawnictwo BTC, W-wa 2005.
2. http://www.picaxe.com (in 2013).
3. D. Lincoln: Programming and customizing PICAXE Microcontroller. McCraw-Hill, New York 2006.
4. PICAXE Manual 1, PICAXE Manual 2, PICAXE Manual 3: instructions in pdf. http://www.picaxe.com
5. Stepper Motor Basics. Industrial Circuits Application Note. http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf (in 2013).
6. http://www.irf.com/product-info/datasheets/data/ir2110.pdf
7. http://www.alldatasheet.com/datasheet-pdf/pdf/22432/STMICROELECTRONICS/L293D.html
8. R. Tokheim: Digital Electronics. McGraw-Hill International edition. NY 2008.