**LAB1. I/O Ports**

**The main goal:** to learn the mobile robot system and programming environment MPLAB X, understand how work discrete input-output ports of general purpose.

**Introduction**

The mobile robot (Fig. 1) used for any laboratory work is a tracked platform with a differential control.

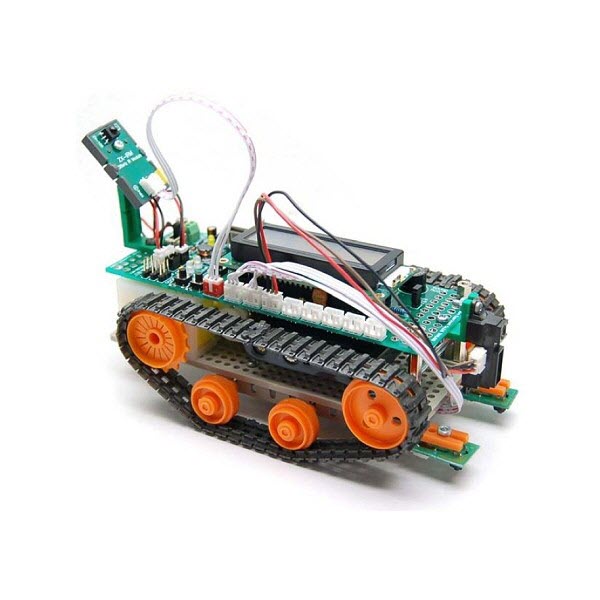


Fig.1. The mobile robot RoboPICA

The platform includes:

* control board with microcontroller PIC18F4525 (Fig.2)$
* power converters;
* power unit for control of track actuators;
* LCD-display;
* Buttons;
* sockets for connection of sensors and other devices;
* track actuators;
* set of sensors.

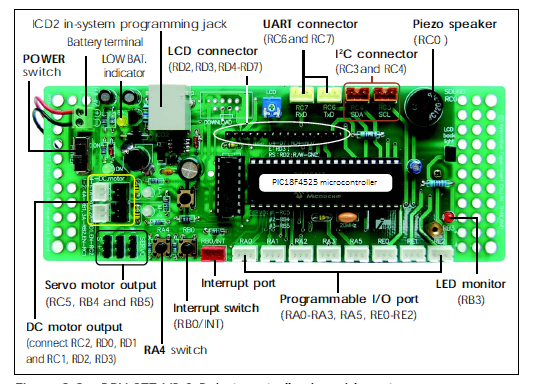
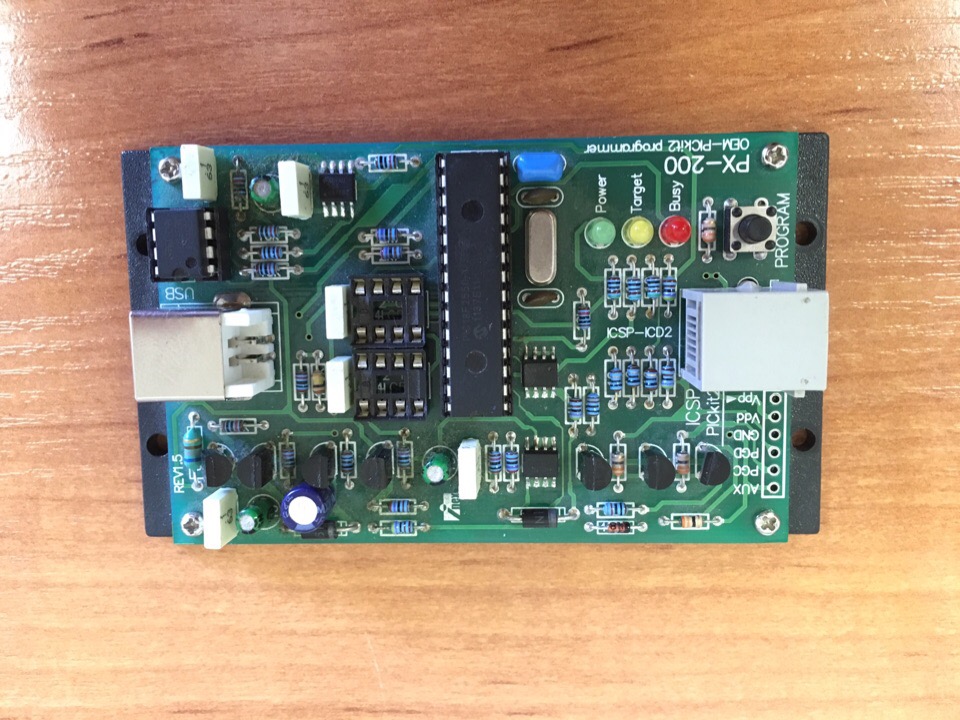


Fig.2. Robot controller board layout

To program the microcontroller, we use the programmer PICkit2 (fig. 3), this device (board) is used to transmit your C-code from PC into MCU. As software we use MPLAB X compiler XC8. The PICkit2 Microcontroller Programming software is capable of programming most of Microchip’s Flash microcontrollers.



Connect to your Robot

Connect with USB-USB B to your PC

Fig.3. PICkit2 board

* 1. **Configuration of robot**

1. Check the connection between PC and PICkit2 (the green LED «Power» should be flashed on);
2. Check the connection between PICkit2 and the robot (fig. 4)

|  |  |
| --- | --- |
| ÐÐ°ÑÑÐ¸Ð½ÐºÐ¸ Ð¿Ð¾ Ð·Ð°Ð¿ÑÐ¾ÑÑ robopica | https://pp.userapi.com/c850424/v850424430/13a11f/s0p7tX0mDQE.jpg |

Fig.4. Connection between PICkit2 and RoboPICA

1. Check the connection between RoboPICA and power source (fig. 5)

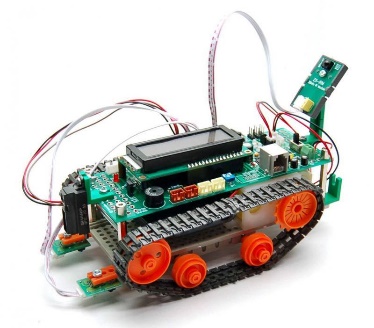




Fig.5. Connection to power source

* 1. **Opening the main program file**

1. Run the MPLAB X IDE (if it is not running).
2. Find in the Project tree project with name «**Prog\_school**».
3. Select in the architecture of working project the folder **Source Files**, clicking on the ‘**+**’ and **main.c** (double click). This is your main working file where you should write your own code (fig. 6). File **lib.c** consists of c-code of different functions that you can use.

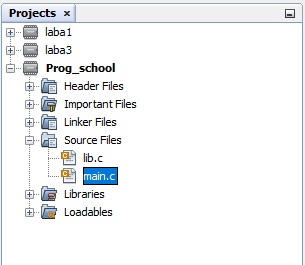


Fig. 6. Project tree

In that file you can see

|  |  |
| --- | --- |
| #include “lib.h” | It is the header file, where there are all prototypes of function that you can use. |
| void main(void) | The name of the main function, you cannot rename it or delete, because the compiler will search for it and if it do not find this function, the compiler will give you an error and you cannot run your program. |
| init\_all\_units( ); | This function is used to initialized all peripheral units in robot (LCD, motors, ADC etc.) |
| while( 1 ) | Infinite loop |

In folder «Header Files» you can find 3 files:

|  |  |
| --- | --- |
| define.h | There are some constants used for lcd-unit and also the value of MCU working frequency – #define \_XTAL\_FREQ 8000000 (it is important for delay-function and timer-unit) |
| lib.h | This file includes prototypes of functions |
| pragma.h | This file includes the settings for microcontroller. These settings will be different for different microcontrollers. |

* 1. **Working with the I/O general purpose ports**

Many of outputs of the microcontroller can be configured as to operate as digital inputs and outputs, as well as to work with other peripherals. Port I/O of 8-bit microcontroller is a group of eight or less microcontroller pins that can be configured as digital inputs or outputs. Designation of microcontroller pins that can work as digital general purpose inputs and outputs begins with a letter «R» and contains the letter corresponding to a particular port, and the number corresponding to the number of bits of the port. For example, if the output of the microcontroller contains in its designation «RB3», it means that this pin can be used as a digital input or output of the port B, and to configure it and work with - it must be used the third bit registers.

To work with the IO general purpose ports we used three registers:

* PORT\* - read the voltage level of outputs (= «to read»);
* LAT\* - set the condition of digital output (= «to write»);
* TRIS\* - set the operating mode of a particular output of the microcontroller («1» = «input», «0» = «output»).

When you work with input-output ports, special attention should be given to configuration of the microcontroller peripherals, which can also be used for the implementation of the selected digital input or output terminals of the controller. For example, when we can select output as an analog input of the controller it is necessary for providing proper operation as digital input in configuration of module ADC to disable the channel of analog input. The possibility of using the output of the controller as an analog input indicates the presence in its description designation "AN\*", where the symbol "\*" marks the number of analog inputs (fig. 7).

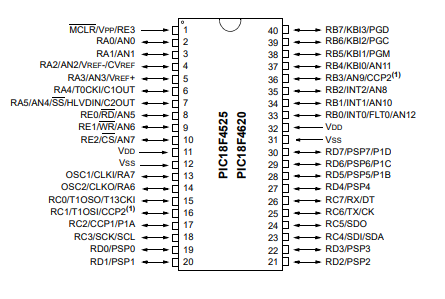


Fig.7. Pinout PIC18F4525

As an example, operating with I/O general purpose output ports, consider the task of managing an LED connected to pin RB3, with the button connected to the RB0. Listing of function **main**, implementing LED control is provided below.

Listing 1.1

void main(int argc, char\*\* argv) {

TRISB3=0; /\* Setting RB3 as a digital output \*/

TRISB0=1; /\* Setting RB0 as a digital input \*/

while (1) { /\* Infinite loop \*/

LATBbits.LATB3 =! PORTBbits.RB0;

}

}

**Exercises**

1. Write a program that implements the flashing LED.

2. Modify the program so that the LED flashes only when the button connected to RA4 is pressed.

3. Write a program that when you click on RA4 switch on piezo sound connected to the RC0 (for piezo you must be supply a square wave with a frequency of 12-18 kHz).

4. Write a program that implements the simultaneous flashing LED and piezo operation at the touch of a button connected to RA4.

5. Modify the program so that the first short pressure of the button starts blinking, the next short press adds flashing sound, the follow short press switches off flashing and sound. In this task it is necessary to exclude a situation in which the switching between modes is done spontaneously due to "bounce" contacts of the button.

**Additional information**

The simplest way to implement a delay is to use library functions **\_\_delay\_ms()** and **\_\_delay\_us()**, which provides a delay in milliseconds and microseconds, respectively.

**LAB2. ADC**

**The main goal:** to view the work of analog sensors placed on the mobile robot.

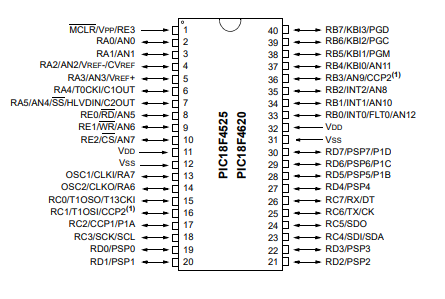
**Introduction**

List of analog sensors of the mobile robot is presented in Table 2.

Table 2. List of analog sensors

|  |  |  |
| --- | --- | --- |
| **№ sensor** | **№ channel ADC** | **Sensor** |
| 0 | AN0 | Reflected infrared sensor |
| 1 | AN1 | Reflected infrared sensor |
| 2 | AN2 | Reflected infrared sensor |
| 3 | AN3 | Distance sensor |
| 4 | AN4 | Light sensor |
| 5 | AN5 | Noise sensor |
| 6 | AN6 | Temperature sensor |
| 7 | AN7 | Potentiometer |

The ADC as input get the voltage values from sensor (fig. 8) and convert it into specific values (from 0 till 1023, because in this microcontroller 10-bit ADC unit is used, it means that output values may be changed from 0 till ).



0 … 1023

0 … 5 V

ADC

Fig. 8. ADC-unit

To work with ADC you need to initialize this unit, it will «tell» your microcontroller that you want to use it. The additional information about meaning of all setting you can find in datasheet.

Example of functions for running ADC conversion is shown in Listing 2.

Listing 2.1

void Adc\_init()

{

TRISA|=0b00101111; // *setting ports RA0, RA1, RA2, RA3, RA5 as inputs*

TRISE|=0b00000111; // *setting ports RE0, RE1, RE2 as inputs*

ADCON1bits.PCFG=0b0111; // *configuration of analog-digital ports*

ADCON1bits.VCFG=0b00; // *reference voltage Vss Vdd*

ADCON2bits.ACQT=0b111;// *conversion time 20 Tad*

ADCON2bits.ADCS=0b110;// *conversion frequency Fosc/64*

ADCON2bits.ADFM=0;// *left offset*

ADCON0bits.ADON=1; // *ADC is on*

}

Function Adc\_init() is called inside of init\_all\_units() function, so you do not need to call it in your function. This function should be called only once, remember, that without initialization you cannot get any values from any devices, and the compiler also will not tell you that it is a mistake.

To get values from sensors you need to use read\_ADC function. Instead of channel you need to write the number of sensor that you want to use. The code is shown in Listing 2.2.

Listing 2.2.

int read\_Adc( int channel )

{

ADCON0bits.CHS= channel; // *choice of analog channel*

ADCON0bits.GO\_DONE=1; // *running conversion*

while(ADCON0bits.GO\_DONE==1);

return (ADRESH<<2)+(ADRESL>>6);// *writing the result of conversion*

}

In your main-function you need to initialize variable and put values from ADC in there. The example is shown in Listining 2.3.

Listinig 2.3.

void main( )

{

Adc\_init( );

int adc\_value = 0;

while(1)

{

adc\_value = read\_Adc( 7 ); // read sensor №7

}

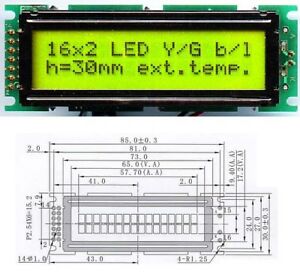
}

For interaction with the operator, as well as for displaying information useful for debugging software the robot has a character display. The LCD-display has 2 lines with 16 characters (fig. 9). To set the position of cursore you may use HEX-values:

* For 1 line: for 0x80 till 0x8f;
* For 2 line: for 0xC0 till 0xCf

0x8f

0x80



0xCf

0xC0

Fig.9. LCD-display

Listing 2.4 shows an example of functions needed for working with the display.

Listing 2.4

// Display string

void lcd\_puts(unsigned char line, const char \*p)

{

lcd\_origin(); // The transition to the 0 address LCD

lcd\_command(line); // Set address LCD 00H

while(\*p) // Check if the designator is 0

{

lcd\_putc(\*p); // Send data to LCD

p++; // Increase address by 1

}

}

// Display integer values

void inttolcd(unsigned char posi, long value) {

char buff[16];

itoa(buff,value,10);

lcd\_puts(posi,buff);

}

void main( )

{

// do not forget that it is already in init\_all\_unit()

lcd\_init();

int a = 0;

lcd\_puts(0x80,"Counter=");

while(1)

{

if( a >= 65535 ) a = 0; // to protect from overwrinting

inttolcd(0x80, a);

\_\_delay\_ms(10);

a++;

}

}

**Exercises**

1. Write a program that implements the measurement of the voltage on the potentiometer and the output values on the character display in ADC units (0-1023).

2. Modify the program so that the display shows the number of the sensor in the first line, and the result of measurement on the second line, switch between sensors is provided by short pressing the button connected to RA4. In this task it is necessary to exclude a situation in which the switching between channels is done spontaneously due to "bounce" contacts of the button.

3. Modify the program so that the switching between the channels is done by the potentiometer, wherein the potentiometer is excluded from the list of sensors from which the measurement results are displayed. In this task it is necessary to exclude a situation in which the switching between channels is done spontaneously due to "noises" of analog contacts.

**LAB 3. Interrupts**

**The main goal:** familiarize with the use of interrupts in the development of software for control system of mobile robot.

**Introduction**

Microcontrollers of PIC18 family provides the opportunity to organize two levels of interrupts:

* high priority;
* low priority, which allows during the processing of low-priority interrupts to respond to high-priority interrupt.

When creating functions with interrupt handlers, you must use reserved words ‘**interrupt’** and **‘interrupt low\_priority’** to identify the interrupt handler of high and low levels, respectively. Also, when creating the interrupt handler, it must be remembered that this function has no parameters and returns nothing.

Let’s consider the example of using only high priority interrupt from external source (button RB0/INT0). Please note that the INT0 interrupt for used family of microcontrollers has only one priority, and it is a high priority. The following listing of the program that provides the state change of the output RB3 at press the button connected to pin RB0/INT0.

Listing 3.1

void high\_interrupt\_init( void )

{

RCONbits.IPEN=1; // two-level interrupt is enabled

INTCON2bits.INTEDG0=0; // interrupt on the falling edge of the input signal INT0

INTCONbits.INT0IF=0; // reset the interrupt flag from an external source

INTCONbits.GIEH=1; // the high-level interrupt is enabled

INTCONbits.INT0IE=1;// interruption from an external source is enabled

}

void interrupt HIisr (void)

{

if (INTCONbits.INT0IF) // is true when you press the button

{

\_\_delay\_ms(30);// delay to eliminate the influence of the "bounce" of contacts

if (PORTBbits.RB0==0)

{

LATBbits.LATB3=!LATBbits.LATB3;

}

INTCONbits.INT0IF=0;

}

}

void main(void)

{

TRISBbits.RB3=0;// configure the RB3 as output

TRISBbits.RB0=1;// configure the RB0 as input

high\_interrupt\_init( );

LATBbits.LATB3=1;

while(1);

}

Note that function high\_interrupt\_init in not included into function init\_all\_units( ), so you need to call it in main.

As an example of working with timers, consider setting a timer №0 to generate interrupts at a frequency of 1 Hz. The timer №0 is used, it has the 16-bit operational mode, but remember that all registers in microcontroller is 8-bit. The most important thing is to calculate the period of time correctly. You should use the next equation for that:

|  |  |  |
| --- | --- | --- |
|  |  |  |

where 65535 – the value of overwriting for 16-bit timer, 4000000 – the operation frequency of microcontroller; prescaler – the coefficient (table 3.1), Period [s] – the desired period in seconds.

For example, if we want to get period of timer = 1 s. We will calculate

26472

0110 0111 0110 1000 103 104

After that we need to convert this decimal value into binary and then get two 8-bit values.

|  |  |  |
| --- | --- | --- |
| 55769 = | 1101 1001 | 1101 1001 |
|  | 217 | 217 |

Table 3.1. The choice of the value of pre-divider Timer 0

|  |  |  |
| --- | --- | --- |
| **The value of the pre-divider** | **T0PS<2:0>** | **Decimal values** |
| 1:256 | 111 | 8 |
| 1:128 | 110 | 7 |
| 1:64 | 101 | 6 |
| 1:32 | 100 | 5 |
| 1:16 | 011 | 4 |
| 1:8 | 010 | 3 |
| 1:4 | 001 | 2 |
| 1:2 | 000 | 1 |

The result of counting of clock pulses is stored in the registers TMR0L and TMR0H in 16-bit mode or in the register TMR0L, if the timer is configured for 9-bit mode. When operating in 16-bit mode operation read and write both the high and low registers are performed simultaneously, when accessing to the younger register. Register TMR0H is not actually the high byte of the Timer0 module in 16-bit mode. In fact this buffered value of real high byte Timer0, is not available for read or write. TMR0H is updated with the contents of the high byte Timer0 when reading the TMR0L register. This ensures that there will not be a transfer from the low to the high bit in the gap between the reading of the low and high digits. Similarly, the entry in the high byte of Timer0 occurs through the TMR0H register. The pre-recorded value in this register is transferred to the Timer0 write command to TMR0L. This allows you to update the 16-bit counter at one time and to avoid transfer during the update process.

To set the required frequency of generation of interrupts, it is necessary to calculate the duration of one count of timer. With the recommended settings for the system, bus frequency will be 10 MHz, then one count-down timer pre-divider equal to one, will be 0.1 microsecond. To set the frequency to 1 Hz 10 000 000 counts must be produced, which is impossible to achieve without the use of pre-divider.

To fulfill this task, we use the pre-divider equal to 256. In this case, one countdown timer will be up to 25.6 microsecond, but it is required approximately 39 062 countdowns to count for 1 second. The error of maintaining the frequency in this calculation will be composed of errors caused by the discreteness of the step change of duration of one frame (for this example is 0,0000128 Hz) and errors due to the time spent for the execution of commands of microcontroller, necessary to restart the timer. In this example we are setting up the interrupts so that the timer interrupt will have the lowest priority. The text of the program is shown in listing 3.2.

Listing 3.2

void interrupt low\_priority LIisr (void)

{

if (INTCONbits.TMR0IF)

{

INTCONbits.TMR0IF=0;

TMR0H=103;

TMR0L=106;

LATBbits.LATB3=!LATBbits.LATB3;

}

}

int main(int argc, char\*\* argv) {

TRISBbits.RB3=0;// configure the RB3 as output

T0CONbits.T08BIT=0;// setting timer №0 to 16-bit mode

T0CONbits.PSA=0;// pre-divider is used

T0CONbits.T0PS=0b111;//pre-divider = 256

T0CONbits.T0CS=0;// selection of the internal source clock

TMR0H=103;// the write high byte of initial value

TMR0L=106;// the write low byte of the initial value

T0CONbits.TMR0ON=1;

RCONbits.IPEN=1; // two-level interrupt is enabled

INTCON2bits.TMR0IP=0;// assign an interrupt a low priority

INTCONbits.TMR0IF=0;// reset the interrupt flag on overflow of timer 0

INTCONbits.GIEH=1; // the high-level interrupt is enabled

INTCONbits.GIEL=1; // low-level interrupts is enabled

INTCONbits.TMR0IE=1;// overflow interrupt for timer 0

while(1);

}

**Self-study task**

1. Write a program using interrupts, implements counting the number of clicks on the button connected to RB0. Display the result on the screen. Use the only high priority interrupt.

2. Supplement the program with implementation of the flashing led connected to RB3. In the interrupt handler for the timer, enter a delay of 700 MS (using the function \_\_delay\_ms). Describe the result of the work by setting the overflow interrupt timer high priority, and when assigning the interrupt to low priority. Explain the differences between programs at different levels of interrupt priority for timer.

**Additional task**

Write a program using interrupts that implements the stopwatch function with a step of 1 MS. The stopwatch is started and stopped by brief press of the button connected to RB0. The current value of the measured time and the measurement result shall be displayed.

**LAB 4**

**The main goal:** familiarization with the principle of drive control of the tracks of the robot.

**Introduction**

The DC motor is applied as drives caterpillars of the considered mobile robot. The control of the rotation speed of the tracks is performed using the power amplifier with pulse width modulation. To control each motor three pins of the microcontroller are used: two for mode control (clockwise rotation, counterclockwise rotation and braking) and one, specifying the frequency and duty cycle. The information about the control mode of the engines is shown in the table 4.1.

Table 4.1. The control mode of the engine

|  |  |
| --- | --- |
| **Mode** | **The values of the corresponding outputs** |
| The motor A rotation clockwise | RD0=0, RD1=1 |
| The motor A rotation counterclockwise | RD0=1, RD1=0 |
| The motor A braking | RD0=0, RD1=0 |
| The motor B rotation clockwise | RB1=0, RB2=1 |
| The motor B rotation counterclockwise | RB1=1, RB2=0 |
| The motor B braking | RB1=0, RB2=0 |

The frequency and duty cycle are set using the module capture/compare/PWM (CCP module). When the motors control, both CCP modules work in PWM mode, a PWM of the same frequency and phase, using the basic timer TMR2. To set the operating mode of the PWM it is necessary to determine the CCP1M bits (mode selection PWM) register ССP1CON for the first CCP module and bits CCP2M ССP2CON register for the second module, respectively. Values for duty cycle are specified in registers CCPR1L and CCPR2L. The PWM period is determined by the value of the pre-divider of the timer T2CKPS TMR2 and PR2 period value. At the recommended settings for the system bus frequency will be 10 MHz, then to generate a frequency of 20 kHz, you must set the value of the period equal to 124 for a single pre-divider. In this case, to specify the duty cycle from 0 to 100% values from 0 to 124 shall be written to the appropriate registers. Listing 4.1 presents an example of the functions that are necessary for motor control.

Listing 4.1

void motor\_init()

{

TRISDbits.RD0=0;

TRISDbits.RD1=0;

TRISBbits.RB1=0;

TRISBbits.RB2=0;

TRISCbits.TRISC1=0;

TRISCbits.TRISC2=0;

CCP1CONbits.CCP1M=0b1100; // set the operating mode of the module CCP1 (PWM)

CCP1CONbits.P1M=0b00; // involved only a single output P1A

CCP2CONbits.CCP2M=0b1111; // set the mode of operation of module CCP2(PWM)

CCPR1L=0; // set a zero duty cycle

CCPR2L=0; // set a zero duty cycle

PR2=124;// set the PWM period

T2CONbits.T2CKPS=0b00; // set the pre-divider of Timer2 module is equal to 1

T2CONbits.TMR2ON=1;// enable Timer2 module

}

To control the direction of motor rotation, you must set the ports RD0, RD1 for the engine A and RB1, RB2 for the engine B in accordance with table 4.1. An example of a function that implements drive control as shown in listing 4.2

Listing 4.2

void motor\_a\_change\_Speed (signed char speed)

{

if (speed>0) // forward movement

{

CCPR1L=speed;

PORTDbits.RD0=0;

PORTDbits.RD1=1;

}

else if (speed<0) // backward movement

{

CCPR1L =-speed;

PORTDbits.RD0=1;

PORTDbits.RD1=0;

}

else //stop

{

CCPR1L=0;

PORTDbits.RD0=0;

PORTDbits.RD1=0;

}

}

**Self-study task**

1. Write a program that implements control of a power amplifier to drive the rotation of the tracks. The duty cycle must be set with a potentiometer and displayed on the display. Determine the minimum value of the duty cycle at which the tracks move.

2. Modify the program so that the potentiometer can be set duty cycle, but the direction of movement as well. When performing this task, it is necessary to exclude situations in which the power amplifier pulses, the duty cycle which is less than the minimum value defined in task 1.

**Additional task**

Write a program that implements the movement of the robot without collisions with obstacles. For obstacle detection it is necessary to use a distance sensor. If the range of a distance sensor there is no any obstacles, the robot should move forward. When an obstacle is detected the robot must stop and begin to rotate, until within range of the proximity sensor will not appear any obstacles.

**LAB 5**

**The main goal:** create a program controls the movement of the robot along the trajectory, defined by black line on a white background.

**Introduction**

To determine the color of the surface on which the robot is moving three reflective sensor located in the front part at the bottom of the robot are used. Two sensors are located at the edges of the platforms closer to the tracks and the third is in the middle. This arrangement of sensors allows you to determine whether the robot is over the black line, and in what direction the robot moves relative to the black line.

**Self-study task**

1. Using the program developed in lab 2, determine the values of the analog-to-digital conversion of the signal of the reflective sensors for the cases of location of the sensor over a black line over a white field and on the edge of a black line.

2. Create a program that ensures the movement of the robot along the trajectory, defined by black line on a white background, with the use of relay control algorithm drives caterpillars. Make an experiment and find empirically the boundary values of the sensors with which the robot passes the road as quickly as possible and not lose the black line.

3. Modify the program developed in the previous paragraph so that the duty cycle of the PWM drives caterpillars proportionally dependent on the value of the difference of the readings of the left and right reflective sensors. Draw attention to the fact that the motion of the robot over the black line the difference between the readings of the sensors may differ significantly from zero. In this case, the program will need to be offset differential sensors. To improve the stability of the robot’s movement on straight sections of the track it is recommended to complement the control algorithm with dead zone for small values of the difference values of the sensors. Make an experiment and find empirically the gain for the proportional control law, the offset is the difference between the readings of the sensors and boundaries of the dead zone in which the robot runs track as quickly as possible and not lose the black line.

Modify the program developed in the previous paragraph, so that the duty cycle of the PWM for the drive systems of caterpillars depends not only on the value of the difference of the readings of the left and right reflective sensor, but also on this value’s speed change. Note that when using a differential component of the control law it is necessary to provide measures to reduce the influence of noise of the sensors on the generation of a control signal. Make an experiment and find empirically the gains for the proportional and differential components of the control law under which the robot passes the road as quickly as possible and not lose the black line.

**Additional task**

Create a program that ensures the movement of the robot along the trajectory, defined by black line on a white background with a detour of obstacles. When an obstacle arises on the path of a robot, the robot must go around it and return to the trajectory specified by the black line