Projektowanie układów elektronicznych

Raport końcowy

Wykonany projekt: Robot typu Light Follower

Wykonanie: Wojciech Tyczyński

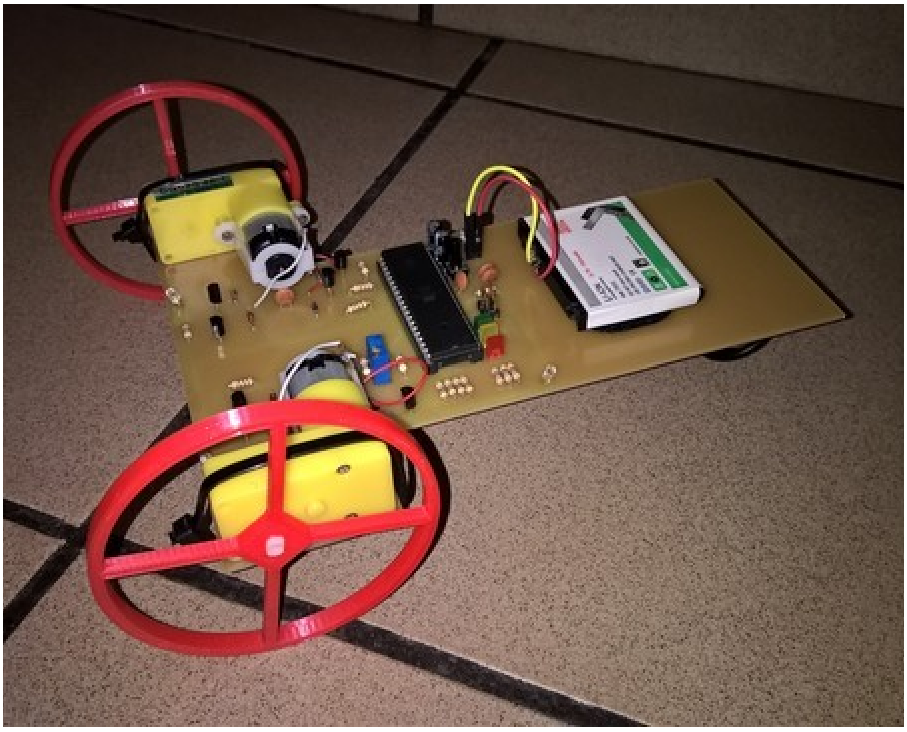
Numer albumu: 104015

Wydział Elektroniki i Telekomunikacji

Politechnika Poznańska

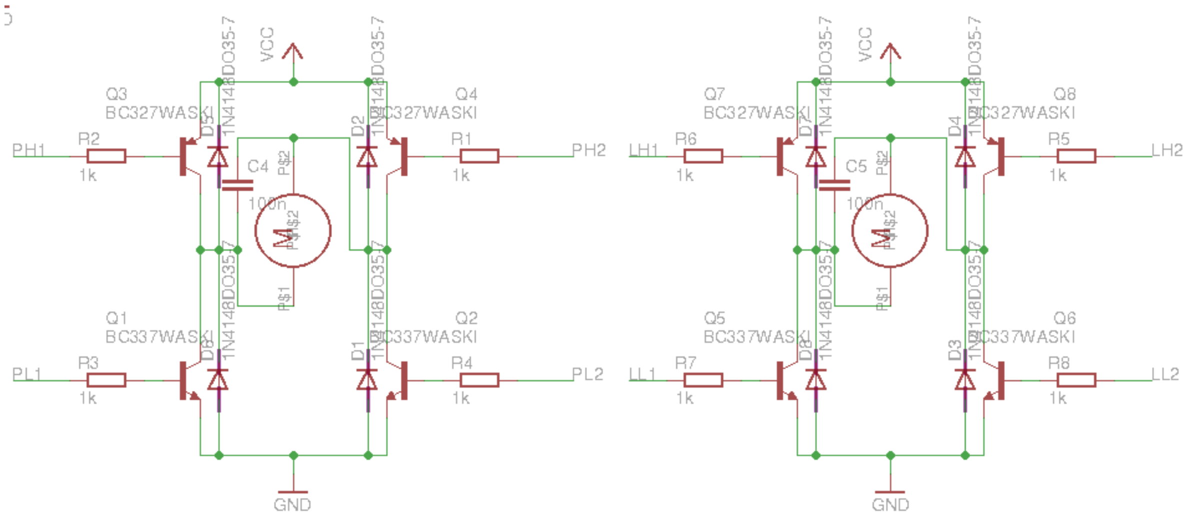
Poznań 2016

1. Zdjęcie wykonanego robota.

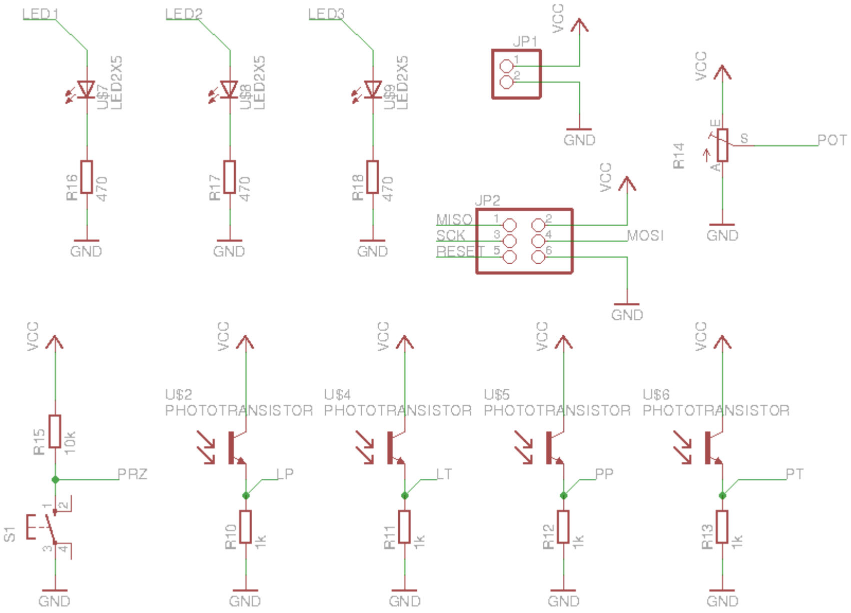


Rys. 1: Wykonany robot - efekt końcowy

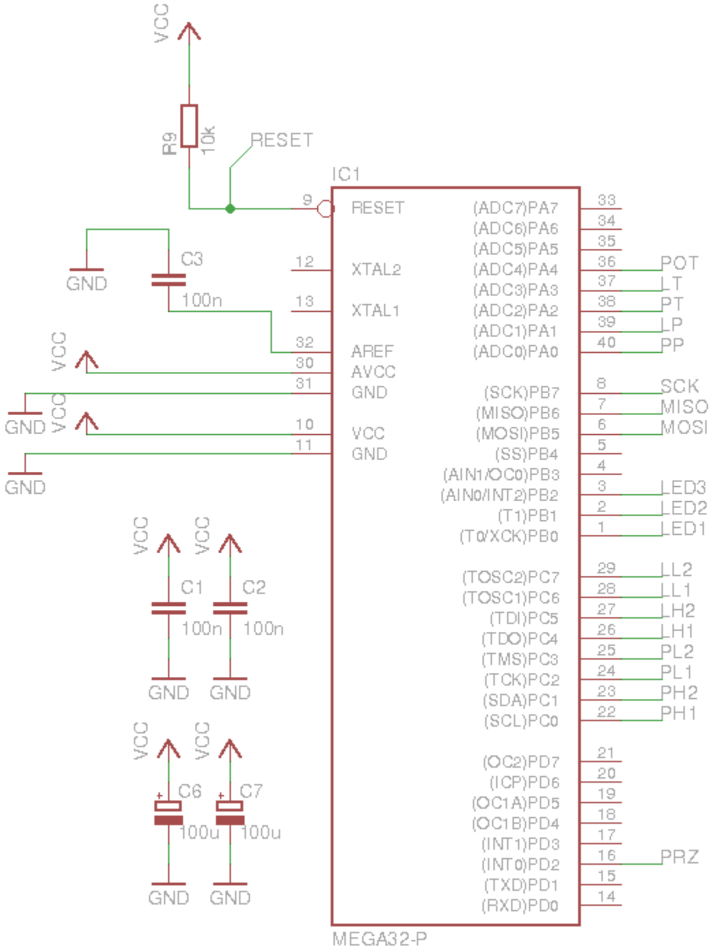
2. Schemat robota.



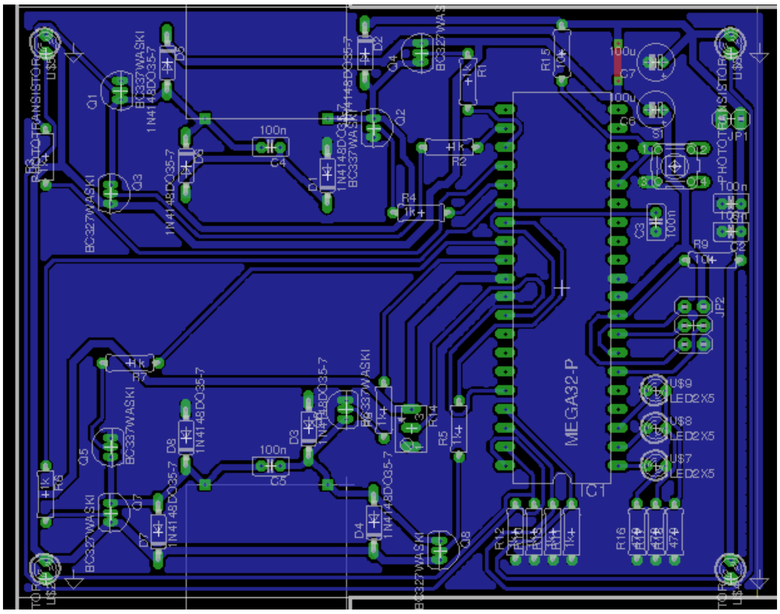
Rys. 2: Silniki



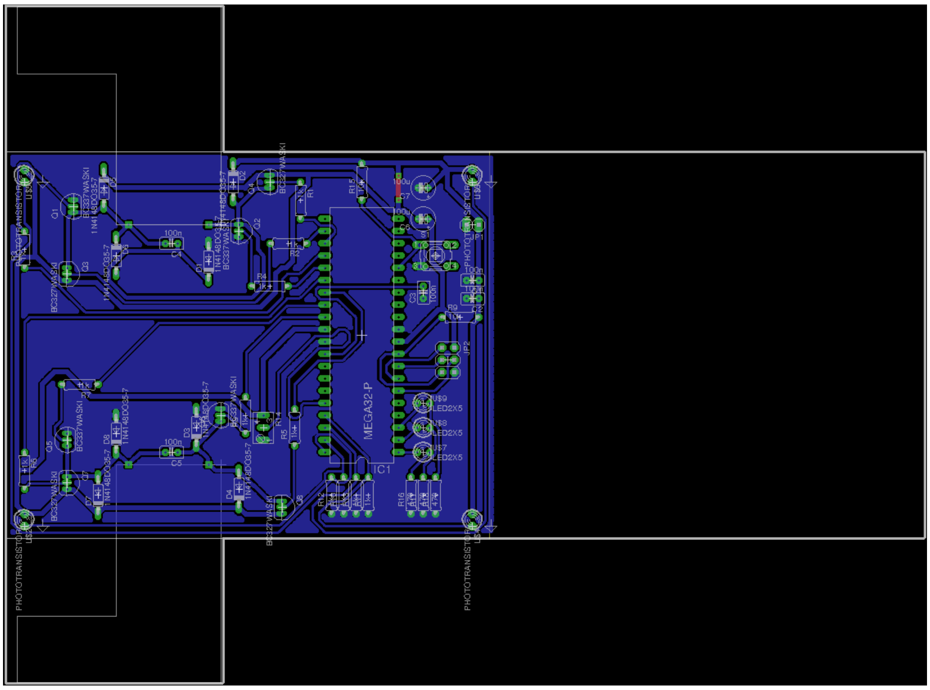
Rys. 3: Fototranzystory, LEDy, przycisk oraz piny do zasilania i SPI



Rys. 4: Schemat połączeń mikrokontrolera ATmega 32A

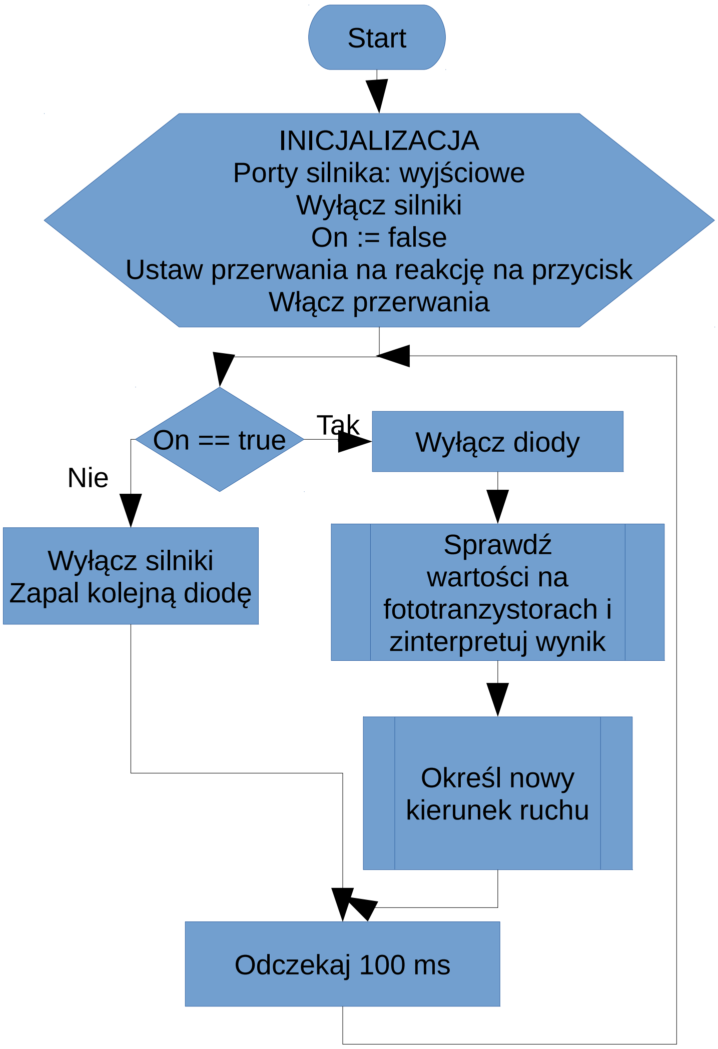


Rys. 5: Schemat połączeń na płytce (widok na elementy)

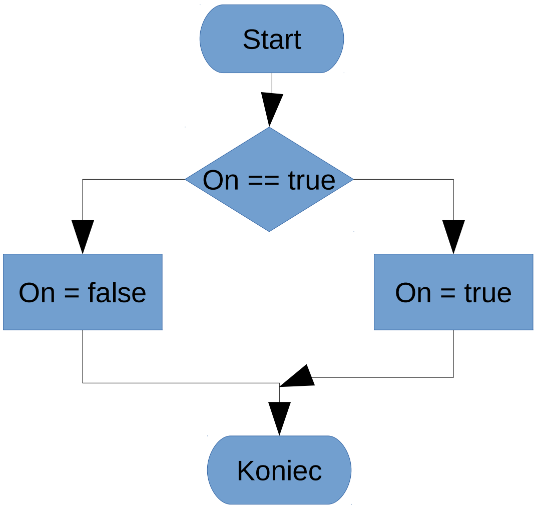


Rys. 6: Schemat połączeń na płytce (widok na całą płytkę)

3. Schemat blokowy opracowanego kodu



Schemat 1: Pętla główna



Schemat 2: Przerwanie

4. Kod programu

Listing 1. Nagłówki

```
/*
 * main.h
 *
 *  Created on: 6 lut 2016
 *      Author: tykus
 */

#ifndef MAIN_H_
#define MAIN_H_

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

#define POTENTIOMETER_CH 4
#define POTENTIOMETER_PORT PORTA
#define POTENTIOMETER_DDR DDRA

#define PHOTOTRANSISTOR_DDR DDRA
#define PHOTOTRANSISTOR_PORT PORTA
#define PHOTOTRANSISTOR_LT_CH 3
#define PHOTOTRANSISTOR_PT_CH 2
#define PHOTOTRANSISTOR_LP_CH 1
#define PHOTOTRANSISTOR_PP_CH 0
#define NUM_OF_PHOTOTRANSISTORS 4

#define ENGINE_DDR DDRC
#define ENGINE_PORT PORTC
#define ENGINE_PH1_CH 0
#define ENGINE_PH2_CH 1
#define ENGINE_PL1_CH 2
#define ENGINE_PL2_CH 3
#define ENGINE_LH1_CH 4
#define ENGINE_LH2_CH 5
#define ENGINE_LL1_CH 6
#define ENGINE_LL2_CH 7

#define LED_DDR DDRB
#define LED_PORT PORTB
#define LED1_CH 0
#define LED2_CH 1
#define LED3_CH 2

#define PRZ_DDR DDRD
#define PRZ_PORT PORTD
#define PRZ_CH 2

#define SET_BIT(ADDRESS,BIT) (ADDRESS |= (1 << BIT))
#define CLEAR_BIT(ADDRESS,BIT) (ADDRESS &= ~(1 << BIT))
#define FLIP_BIT(ADDRESS,BIT) (ADDRESS ^= (1 << BIT))
#define CHECK_BIT(ADDRESS,BIT) (ADDRESS & (1 << BIT))

#define SET_BITMASK(x,y) (x |= (y))
#define CLEAR_BITMASK(x,y) (x &= (~y))
#define FLIP_BITMASK(x,y) (x ^= (y))
#define CHECK_BITMASK(x,y) (x & (y))
```

```
#define DIRECTION_NONE 0
#define DIRECTION_FORWARD 1
#define DIRECTION_BACKWARD 2
#define DIRECTION_LEFT 3
#define DIRECTION_RIGHT 4
#define DIRECTION_UNDEFINED 5

void adcConfig(char channel);
uint16_t adcMeasure(void);
char direction(uint16_t measures[]);

volatile uint8_t on;

void initialize(void);
char checkPhototransistors(void);
char setDirection(char actualDirection, char measureResult);

#endif /* MAIN_H_ */
```

Listing 2. Funkcje pomocnicze

```
/*
 * utils.c
 *
 *  Created on: 12 lut 2016
 *      Author: tykus
 */

#include "main.h"

/**
 * Configure ADC.
 * channel - channel number for ADC (0-7)
 */
void adcConfig(char channel)
{
    // napiecie odniesienia = Vcc
    // kanal w zakresie od 0 do 7
    ADMUX = (1 << REFS0) | (channel & 0x07);
    // preskaler = 8
    ADCSRA = (1 << ADEN) | (1 << ADPS1) | (1 << ADPS0);
}

/**
 * Get value from ADC (on channel specified earlier in adcConfig).
 * return result of measure
 */
uint16_t adcMeasure(void)
{
    ADCSRA |= (1 << ADSC);
    while (ADCSRA & (1 << ADSC))
    {   // oczekiwanie na koniec konwersji
    }
    return ADC;
}

/**
 * Try to set actual
 */
char direction(uint16_t measures[])
{
```

```
uint8_t i;
uint8_t maxVal1 = 0;
uint8_t maxVal2 = 0;

for (i = 0; i < NUM_OF_PHOTOTRANSISTORS; ++i)
{
    if (measures[i] > measures[maxVal1])
    {
        maxVal2 = maxVal1;
        maxVal1 = i;
    }
    else if (measures[i] > measures[maxVal2])
    {
        maxVal2 = i;
    }
}

if (measures[maxVal1] < measures[maxVal2])
{
    char tmp = maxVal1;
    maxVal1 = maxVal2;
    maxVal2 = tmp;
}
char result = DIRECTION_UNDEFINED;

if (measures[maxVal1] < 0x0003)
{
    result = DIRECTION_NONE;
}

else if (measures[maxVal1] > 10 * measures[maxVal2])
{
    switch (maxVal1)
    {
    case PHOTOTRANSISTOR_LP_CH:
        result = DIRECTION_LEFT;
        break;
    case PHOTOTRANSISTOR_PP_CH:
        result = DIRECTION_RIGHT;
        break;
    case PHOTOTRANSISTOR_LT_CH:
        result = DIRECTION_RIGHT;
        break;
    case PHOTOTRANSISTOR_PT_CH:
        result = DIRECTION_LEFT;
        break;
    default:
        break;
    }
}
else if ((maxVal1 == PHOTOTRANSISTOR_LP_CH
        && maxVal2 == PHOTOTRANSISTOR_PP_CH)
        || (maxVal2 == PHOTOTRANSISTOR_LP_CH
                && maxVal1 == PHOTOTRANSISTOR_PP_CH))
{
    result = DIRECTION_FORWARD;
}
else if ((maxVal1 == PHOTOTRANSISTOR_LT_CH
        && maxVal2 == PHOTOTRANSISTOR_PT_CH)
        || (maxVal2 == PHOTOTRANSISTOR_LT_CH
                && maxVal1 == PHOTOTRANSISTOR_PT_CH))
    {
        result = DIRECTION_BACKWARD;
    }
    else if ((maxVal1 == PHOTOTRANSISTOR_LP_CH
            && maxVal2 == PHOTOTRANSISTOR_LT_CH)
            || (maxVal2 == PHOTOTRANSISTOR_LP_CH
                    && maxVal1 == PHOTOTRANSISTOR_LT_CH))
    {
        result = DIRECTION_LEFT;
    }
    else if ((maxVal1 == PHOTOTRANSISTOR_PT_CH
            && maxVal2 == PHOTOTRANSISTOR_PP_CH)
            || (maxVal2 == PHOTOTRANSISTOR_PT_CH
                    && maxVal1 == PHOTOTRANSISTOR_PP_CH))
    {
        result = DIRECTION_RIGHT;
    }

    return result;
}

/**
 * First initialization.
 * By default:
 * - robot is turned off
 * - engines are turned off
 * - LEDs are turned off
 * Also interruption is set.
 */
void initialize(void)
{
    on = FALSE;
    SET_BITMASK(LED_DDR, 0x07);
    SET_BITMASK(LED_PORT, 0x00);     // Leds turn off
    SET_BITMASK(ENGINE_DDR, 0xFF);   // Engines turn off
    SET_BITMASK(ENGINE_PORT, 0x33); // (for NPN - ON is 1, for PNP - ON is 0)

    SET_BIT(GICR, INT0);
    SET_BIT(MCUCR, ISC00);
    SET_BIT(MCUCR, ISC01);  // ISC00 i ISC01 - reakcja na zbocze narastające
    sei();
}

/**
 * Check states of all phototransistors.
 */
char checkPhototransistors(void)
{
    uint16_t measures[NUM_OF_PHOTOTRANSISTORS];
    adcConfig(PHOTOTRANSISTOR_LP_CH);
    measures[0] = adcMeasure();
    adcConfig(PHOTOTRANSISTOR_LT_CH);
    measures[1] = adcMeasure();
    adcConfig(PHOTOTRANSISTOR_PP_CH);
    measures[2] = adcMeasure();
    adcConfig(PHOTOTRANSISTOR_PT_CH);
    measures[3] = adcMeasure();

    return direction(measures);
}
```

```
/**
 * Change direction
 */
char setDirection(char actualDirection, char measureResult)
{
    if (measureResult != DIRECTION_UNDEFINED
            && actualDirection != measureResult)
    {
        ENGINE_PORT = 0x33;
        _delay_ms(10);
        switch (measureResult)
        {
        case DIRECTION_FORWARD:
            ENGINE_PORT = 0xAA;
            break;

        case DIRECTION_BACKWARD:
            ENGINE_PORT = 0x55;
            break;

        case DIRECTION_LEFT:
            ENGINE_PORT = 0x5A;
            break;

        case DIRECTION_RIGHT:
            ENGINE_PORT = 0xA5;
            break;

        default:
            ENGINE_PORT = 0x33;
            break;
        }
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}
```

Listing 3. Główna pętla i przerwanie

```
/*
 * main.c
 *
 *  Created on: 4 lut 2016
 *      Author: tykus
 */

#include "main.h"

const int DELAY = 100;

/**
 * Switch robot on or off
 */
ISR(INT0_vect)
{
    if (on == TRUE)
    {
        on = FALSE;
    }
    else
    {
        on = TRUE;
    }
}

/**
 * Main loop function
 */
int main(void)
{
    initialize();

    char oldDir = DIRECTION_NONE;
    char newDir = DIRECTION_NONE;
    char state = 1;
    char changeDir = FALSE;

    while (TRUE)
    {
        if (on)
        {
            LED_PORT = 0x00;
            if (changeDir == TRUE)
            {
                oldDir = newDir;
            }
            newDir = checkPhototransistors();
            changeDir = setDirection(oldDir, newDir);
        }
        else
        {
            oldDir = DIRECTION_NONE;
            newDir = DIRECTION_NONE;
            changeDir = FALSE;
            ENGINE_PORT = 0x33;
            LED_PORT = state;
            state <<= 1;
            if (state > 4)
            {
                state = 1;
            }
        }
        _delay_ms(DELAY);
    }
}
```