**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

**Інститут комп’ютерних технологій, автоматики та метрології**

**Кафедра "Комп'ютеризовані системи автоматики"**



**ЗВІТ**

про виконання **лабораторної роботи №7**

з курсу **«Мікроконтролери, частина 1»**

**«Реалізація програмних затримок за допомогою**

**таймерів мовою асемблер»**

**Виконав:**

студент групи ІР-22

Яцків О. О.

Варіант 18

**Прийняла:**

к.т.н., старший викладач КСА

Лагун І. І.

Львів — 2023

**Порядок виконання**

1. Згідно варіанту завдання (таблиця 1) зібрати у пакеті симуляції Proteus схему на основі МК ATmega2560 та написати програму мовою Асемблер в Atmel Studio для реалізації вказаного завдання.

2. Залити програму в Arduino-Kit.

3. Отримані результати представити викладачу

**Завдання (варіант 18)**

1. Два набори LED-ів згідно натиску відповідної кнопки відпрацьовують вказані алгоритми і гаснуть. При цьому алгоритми можуть працювати незалежно один від одного. Натиск відповідної кнопки перезапускає вказаний алгоритм, який зараз працює, спочатку.

2. Часові інтервали вимірюються за допомогою таймерів, при цьому паралельно виконується робота в основній програмі.

3. При натиску кнопок звучить сигнал бузера.

4. Програма має бути написана мовою Асемблер.

Кнопки PK3 і PK4, світлодіоди port-L і port-A, затримка 0.4 сек., таймер T3, алгоритми 2 і 5:

**Алгоритм 2**

Лінійка з 8-ми одноколірних світлодіодів. При натисканні кнопки світлодіоди почергово блимають від 7-виводу порту до 0.

P0←P1←P2←P3←P4←P5←P6←P7

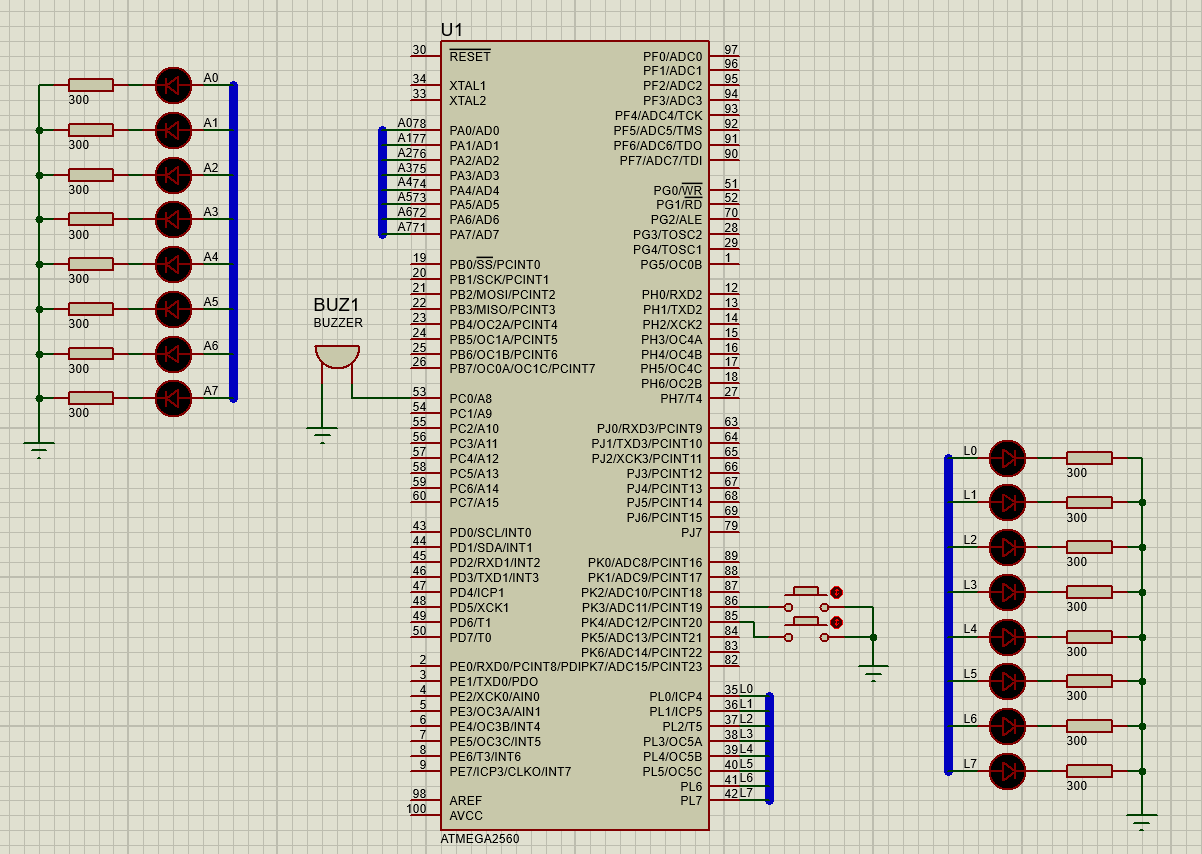
**Алгоритм 5**

Лінійка з 8-ми одноколірних світлодіодів. При натисканні кнопки світлодіоди починають почергово блимати через один від 0-виводу порту до 6, а потім далі від 1 до 7.

P0→ P2→ P4→ P6→ P1→ P3→ P5→P7

**Виконання роботи**

У пакеті симуляції Proteus зібрав необхідну схему на основі мікроконтролера ATMega2560:



Програма мовою Assembly для реалізації вказаного алгоритму:

.macro outi

; This line loads the constant value specified by the macro's first parameter (@1)

; into a register called "\_swap". This is done so that the constant value can be

; easily written to the specified register, which is either an IO register (address

; range 0x00-0x3F) or a memory-mapped register (address range 0x40-0xFF).

ldi \_swap, @1

; This line starts an if statement that checks if the address specified in

; the macro's second parameter (@0) is in the IO register address range (less than 0x40).

.if @0 < 0x40 ; register has max of 0x3F address

; If the address is in the IO register address range, this line writes the value

; in "\_swap" to the specified IO register (@0) using the "out" instruction.

out @0, \_swap

.else

; This line writes the value in "\_swap" to the specified memory-mapped register

; (@0) using the "sts" (store direct to memory) instruction.

sts @0, \_swap

.endif

.endmacro

; macros to write from register

.macro outr

mov \_swap, @1

.if @0 < 0x40

out @0, \_swap

.else

sts @0, \_swap

.endif

.endmacro

; These lines are defining symbolic names for a number of registers in the AVR microcontroller's register file.

; These symbolic names can be used throughout the code to refer to specific registers without having to remember

; the actual register numbers.

; registers

.def \_low =r10 ; I/O ports states

.def \_high =r11

.def \_swap =r20 ; register for outi and outr

.def \_time\_counter =r12

.def \_timestamp =r13

.def \_logic =r0 ; 1=HIGH - button1; 2=HIGH - button2

.def \_counter1 =r19

.def \_counter\_algo1 =r16

.def \_counter\_algo2 =r17

.def \_button\_status =r18

.equ OCRA\_value =781

; =====================================================SRAM==========================================================================

.DSEG

; DATA SEGMENT

; =====================================================FLASH==========================================================================

; Define the CODE SEGMENT (CSEG) for the vector of interrupts

.CSEG

; Define the origin of the program code to address 0x000

; This is where the program starts running after power on or reset

.org 0x000

; Jump to the setup routine

; This routine is responsible for initializing the hardware and setting up the configuration

rjmp setup

; Define the origin of the Timer/Counter 3 interrupt

.org $40

; Jump to the TIMER3\_COMPA label

; This is the interrupt service routine that will be executed when Timer/Counter 3 triggers an interrupt

rjmp TIMER3\_COMPA

setup:

clr \_low

clr \_time\_counter

clr \_timestamp

ser \_swap

mov \_high, \_swap

mov \_counter\_algo1, \_high

mov \_counter\_algo2, \_high

out DDRA, \_high

out PORTA, \_low

sts DDRL, \_high

sts PORTL, \_low

sbi DDRC, 0

cbi PORTC,0

outr DDRK, \_low

outi PORTK, 0b11000

outi OCR3AH, High(OCRA\_value)

outi OCR3AL, Low(OCRA\_value)

outi TCCR3A, 0x00

outi TCCR3B, (1 << WGM32) | (1 << CS32) | (1 << CS30)

outi TIMSK3, (1 << OCIE3A)

; WGM32: This bit enables the CTC (Clear Timer on Compare Match) mode, which causes TC3 to reset to zero

; when it matches the value in OCR3A

; CS32 and CS30: These bits set the clock prescaler to 1024, meaning that the timer will increment once

; every 1024 clock cycles. This determines the frequency at which TC3 will run.

; OCIE3A: This bit enables the Output Compare A Match Interrupt for TC3, which will generate an interrupt

; when TC3 matches the value in OCR3A.

sei ; set global interruption flag

main:

cp \_timestamp, \_time\_counter

breq main

mov \_timestamp, \_time\_counter

clt

lds r21, PINK

andi r21, 0b11000

mov r19, r21

andi r19, 0b1000 ; First button is clicked

brne main\_b1\_nc

set

mov r19, \_button\_status

andi r19, 0b1000 ; First button is clicked before

brne main\_b1\_e

ori \_button\_status, 0b1000

ldi \_counter\_algo1, 0

rjmp main\_b1\_e

main\_b1\_nc:

andi \_button\_status, ~0b1000

main\_b1\_e:

mov r19, r21

andi r19, 0b10000; Second button is clicked

brne main\_b2\_nc

set

mov r19, \_button\_status

andi r19, 0b10000; Second button is clicked before

brne main\_b2\_e

ori \_button\_status, 0b10000

ldi \_counter\_algo2, 0

rjmp main\_b2\_e

main\_b2\_nc:

andi \_button\_status, ~0b10000

main\_b2\_e:

brtc buzzer\_dis

sbi PORTC, 0

rjmp main

buzzer\_dis:

cbi PORTC, 0

rjmp main ; loop main

; DELAYS

; TIMER3 COMPARE MATCH A

TIMER3\_COMPA:

cli

push r1

push r30

push r31

inc \_time\_counter

ldi r30, 10

cp r30, \_time\_counter

brne algo2\_upd\_e

clr \_time\_counter

cpi \_counter\_algo1, 0xff

breq algo1\_upd\_e

ldi r30, Low(algorithm1\*2)

ldi r31, High(algorithm1\*2)

mov r1, \_counter\_algo1

add r30,r1

adc r31, \_low

lpm r1, Z

sts PORTL, r1

inc \_counter\_algo1

cpi \_counter\_algo1, 9

brne algo1\_upd\_e

ldi \_counter\_algo1, 0xff

algo1\_upd\_e:

cpi \_counter\_algo2, 0xff

breq algo2\_upd\_e

ldi r30, Low(algorithm2\*2)

ldi r31, High(algorithm2\*2)

mov r1, \_counter\_algo2

add r30,r1

adc r31, \_low

lpm r1, Z

out PORTA, r1

inc \_counter\_algo2

cpi \_counter\_algo2, 9

brne algo2\_upd\_e

ldi \_counter\_algo2, 0xff

algo2\_upd\_e:

pop r31

pop r30

pop r1

sei

reti

algorithm1:

.db\

0b10000000,\

0b01000000,\

0b00100000,\

0b00010000,\

0b00001000,\

0b00000100,\

0b00000010,\

0b00000001,\

0,0

algorithm2:

.db\

0b00000001,\

0b00000100,\

0b00010000,\

0b01000000,\

0b00000010,\

0b00001000,\

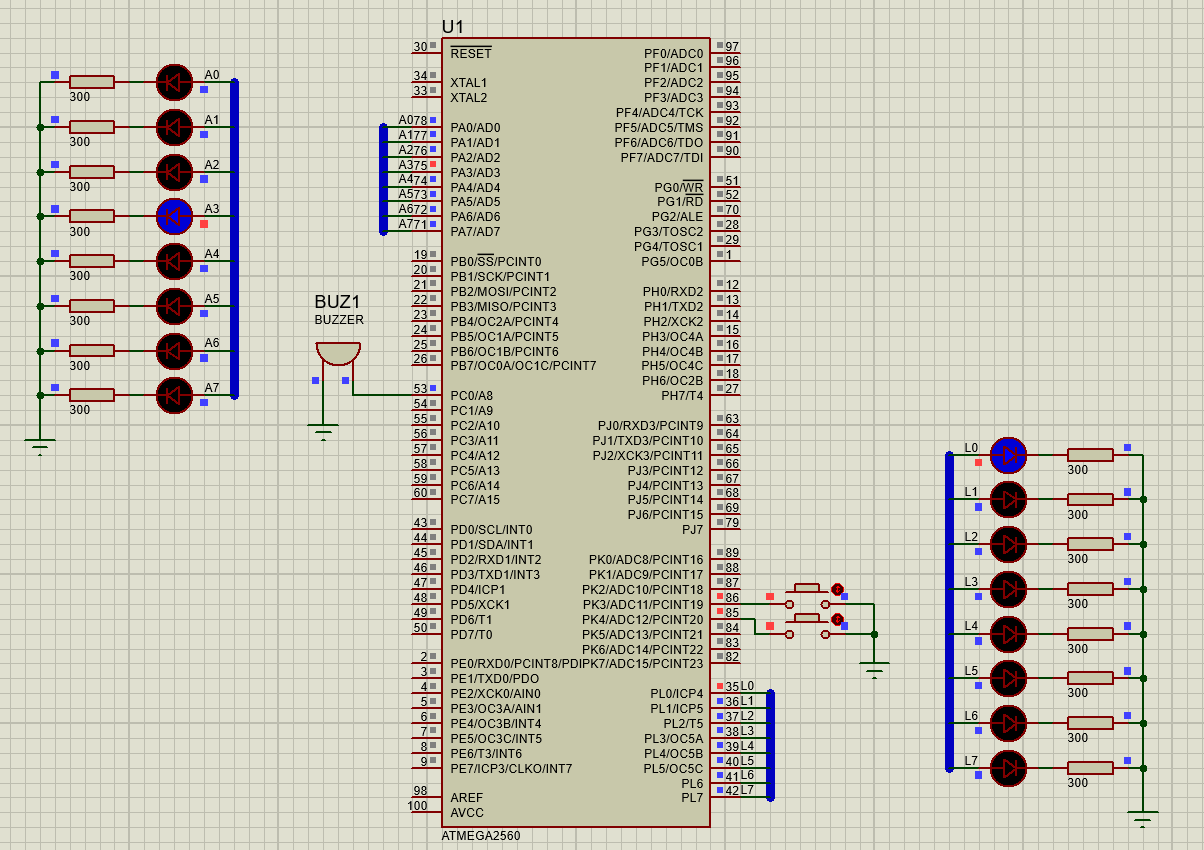
0b00100000,\

0b10000000,\

0,0

; EEPROM

.ESEG



Демонстрація виконання програми на МК

**Висновок:**

На цій лабораторній роботі я ближче познайомився з мовою Assembly та написав на ній свою першу програму на мікроконтролер із використанням таймера, застосувавши і свої попередні навички.