Projekt zaliczeniowy

Systemy czasu rzeczywistego

Grzegorz Łagocki

7716

12.07.2024

Opis zadania

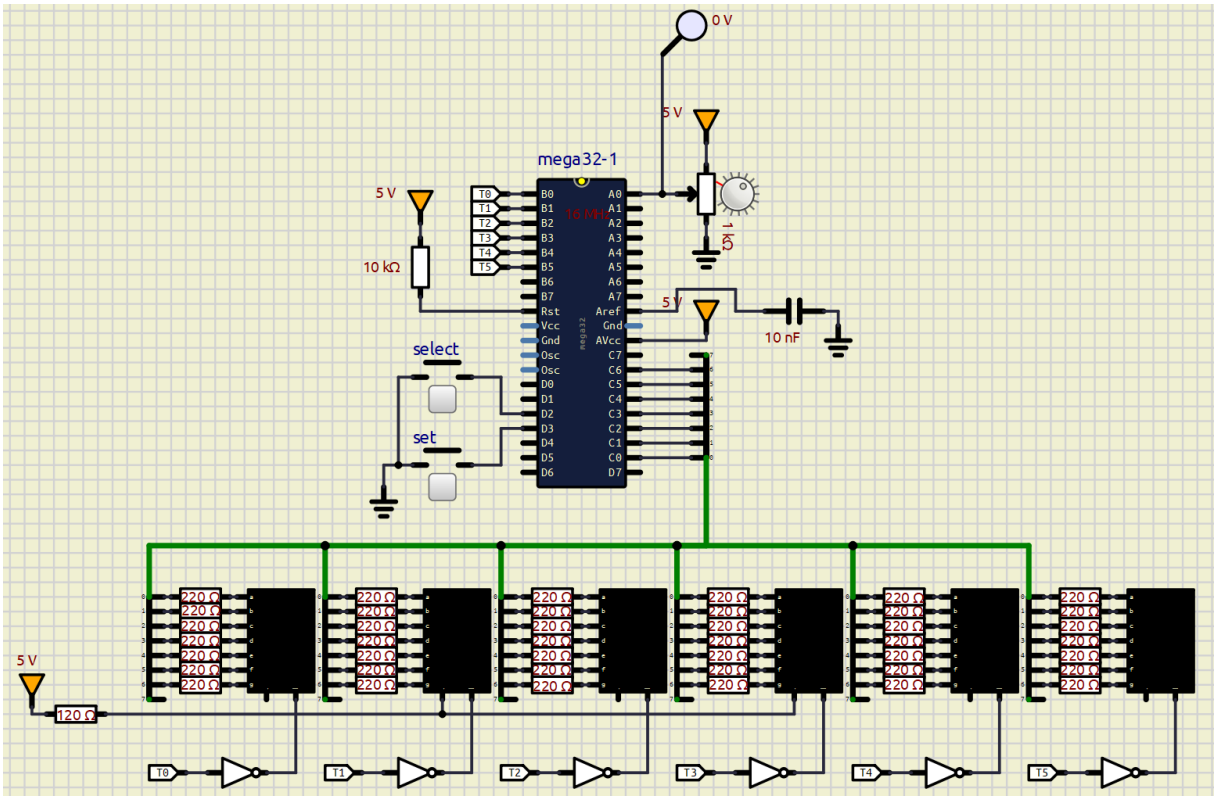
Celem ćwiczenia jest zasymulowanie układu zegara składającego się z sześciu wyświetlaczy 7-segmentowych sterowanych mikrokontrolerem ATmega32. Pierwsze dwa wyświetlają godziny, kolejne minuty i sekundy. W układzie znajdują się dwa przyciski:

- SELECT (pin D2/INT0) – wybór parametru do ustawienia godziny/minuty/sekundy
- SET (pin D3/INT1) – uruchomienie zegarka

Do ustawienia ww. parametrów służy potencjometr (pin A0). Odczytana wartość ADC jest przeskalowywana i ustawiana. Wyświetlacze są multipleksowane z częstotliwością 180 Hz (6 * 30 Hz złudzenia płynnego obrazu).

Układ elektryczny

Układ elektryczny zaprojektowano i zasymulowano w programie SimulIDE 1.1.0-SR0. Głównym układem jest mikrokontroler ATmega32. Do pinów **C[0-6]** podłączono równolegle **anody** poszczególnych segmentów wyświetlaczy poprzez rezystory 220Ω. Do pinów **B[0-5]** podłączono wspólne **katody** wyświetlaczy, które są multipleksowane przez mikrokontroler. W pierwotnej wersji symulacji użyte zostały tranzystory, jednak zostały zastąpione inwerterami logicznymi z powodu problemów z wydajnością symulacji. Do pinu **A0 (ADC kanał 0)** podłączono środkowe wyprowadzenie **potencjometru** (regulowany dzielnik napięcia 0-VCC). Do wejścia **D2/INT0** oraz **D3/INT1** podłączono przyciski zwierane do masy. Do wejścia AREF podłączony jest kondensator filtrujący napięcie odniesienia dla mikrokontrolera.



Rysunek 1: Schemat elektryczny układu

Program sterujący mikrokontrolerem

Kod użyty do zaprogramowania mikrokontrolera ATmega32 został napisany w języku C i skompilowany za pomocą WinAVR-20100110. Składa się z plików:

1. ***main.c*** – plik główny programu
2. ***adc.c***, ***adc.h*** – obsługa pomiaru napięcia
3. ***clock.c***, **clock.h** – obsługa mechanizmu zegara (funkcje odmierzania czasu i ustawiania)
4. ***led7seg.c***, ***led7seg.h*** – obsługa wyświetlaczy 7-segmentowych
5. ***makefile*** – do kompilacji

main.c – plik główny programu

W pliku **main.c** zostają zainicjalizowane wyświetlacze, mechanizm zegara i pomiaru adc. W pętli głównej odświeżane są wyświetlacze. Flaga *clock_tick* jest wystawiana co sekundę w przerwaniu w pliku **clock.c**. W zależności od ustawionego trybu (RUN lub ADJ) czas jest inkrementowany lub przypisywane są wartości przeskalowane z odczytanego napięcia.

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

#include "led7seg.h"
#include "clock.h"
#include "adc.h"

#define LIMIT(arg, val) ((arg < val) ? arg : val) // makro ograniczajace
wartosc

/*************************************************************/
// funkcja glowna programu
/*************************************************************/
int main(void)
{
    // inicjalizacja
    init_led7seg(); // inicjalizacja wyswietlaczy
    init_clock(); // inicjalizacja zegarka
    init_adc(); // inicjalizacja ADC

    sei(); // zezwolenie globalne na przerwania

    while(1)
    {
        refresh_displays();

        if (clock_tick)
        {
            clock_tick = 0;

            if (clock_mode == RUN)
```

```
            {
                add_second();
            }
        }

        if (clock_mode == RUN) // ponizej kod dla ustawiania zegarka pomijany
w trybie RUN
            continue;

        uint16_t adc = read_adc(); // odczyt adc

        switch (cursor_adj) // ustawienie zegarka w zaleznosci odq pozycji
kursora
        {
            case 0:
                clock_set_hours(adc / 43);
                break;
            case 1:
                clock_set_minutes(LIMIT(adc, 1003) / 17); // ograniczenie
wartosci adc do 1003 (59*17)
                break;
            default:
                clock_set_seconds(LIMIT(adc, 1003) / 17);
                break;
        }
    }

    return 0;
}
```

adc.c, ***adc.h*** – obsługa pomiaru napięcia

Zawiera dwie proste funkcje do inicjalizacji i odczytu wartości ADC.

```
#ifndef ADC_H_
#define ADC_H_

void init_adc(void); // inicjalizacja ADC
uint16_t read_adc(void); // pomiar ADC

#endif
```

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

#include "adc.h"


/*************************************************************/
```

```c
// inicjalizacja ADC
/*************************************************************/
void init_adc(void)
{
    DDRA &= ~(1 << PA0); // PA0 jako wejscie
    ADCSRA |= (1 << ADEN) | (1 << ADPS0) | (1 << ADPS1) | (1 << ADPS2); //
wlaczenie ADC, preskaler 128
    ADMUX |= (1 << REFS0); // napiecie odniesienia AVCC
}

/*************************************************************/
// pomiar ADC
/*************************************************************/
uint16_t read_adc(void)
{
    ADCSRA |= (1 << ADSC); // start konwersji

    while(!(1 << ADIF)); //oczekiwanie na koniec konwersji

    return (ADCH << 8) | ADCL; // zwrocenie wyniku 10-bitowego z obu rejestrow
}
```

clock.c, **clock.h** – obsługa mechanizmu zegara (funkcje odmierzania czasu i ustawiania)

Aktualny czas jest przechowywany w zmiennej *current_time* (struktura *time*). Dostępne są funkcje ustawiające i zwiększające poszczególne jej pola. W przypadku przekroczenia wartości maksymalnej zwiększany jest kolejny parametr (np. minuta w przypadku 60 sekund). W funkcji inicjalizacyjnej ustawiany jest timer 1 na 1Hz w trybie CTC oraz przerwania INT0 i INT1 do obsługi przycisków. Funkcja *refresh_displays* zapisuje dane do bufora w pliku **led7seg.c** i realizuje miganie podczas ustawiania poprzez zapisanie -1 do bufora (nie wyświetla wartości ujemnych).

```c
#ifndef CLOCK_H_
#define CLOCK_H_

enum modes { RUN, ADJ }; // tryby pracy zegarka: RUN - odmierzanie czasu, ADJ
- ust. czasu
extern volatile uint8_t clock_mode; // aktualny tryb pracy, domyślnie
uruchomiony
extern volatile uint8_t clock_tick; // flaga ustawiana w przerwaniu do uzycia
w petli glownej
extern volatile uint8_t cursor_adj; // aktualna pozycja kursora

// struktura do przechowywania aktualnego czasu
typedef struct
{
    uint8_t seconds,
        minutes,
```

```
        hours;
} time;
extern time current_time; // zmienna przechowujaca aktualny czas

// deklaracje funkcji
void init_clock(void); // inicjalizacja zegarka

void clock_set_seconds(uint8_t seconds); // funkcja ustawiajaca sekundy
void clock_set_minutes(uint8_t minutes); // funkcja ustawiajaca minuty
void clock_set_hours(uint8_t hours); // funkcja ustawiajaca godzine

void add_second(void); // funkcja zwiekszajaca czas o 1s
void add_minute(void); // funkcja zwiekszajaca czas o 1m
void add_hour(void); // funkcja zwiekszajaca czas o 1h

void refresh_displays(void); // odswieza dane w buforze wyswietlaczy

void clock_set_mode(enum modes mode); // ustawia tryb pracy zegara

#endif
```

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

#include "clock.h"
#include "led7seg.h"
#include "adc.h"

#define CURSOR_NEXT         cursor_adj = (cursor_adj + 1) % 3; // kolejna
pozycja kursora
#define CURSOR_RESET        cursor_adj = 2; // reset kursora


volatile uint8_t clock_mode = RUN; // aktualny tryb pracy, domyślnie
uruchomiony
volatile uint8_t clock_tick; // flaga ustawiana w przerwaniu do uzycia w petli
glownej
volatile uint8_t cursor_adj; // aktualna pozycja kursora
volatile uint8_t blinker; // okresla stan migania przy ustawianiu zegarka

time current_time; // aktualny czas


/*************************************************************/
// inicjalizacja zegarka
/*************************************************************/
void init_clock(void)
{
```

```
    // inicjalizacja timera 1 i ustawienie na 1 Hz (do odmierzania sekund)
    TCCR1B |=  (1 << WGM12); // tryb CTC
    TCCR1B |= (1 << CS12); // preskaler 256
    OCR1A = 31249; // (1 / czestotliwosc) / (1 / (F_CPU / preskaler))) - 1
    TIMSK |= (1 << OCIE1A); // zezwolenie na przerwanie timera 1 przy
przepelnieniu

    // inicjalizacja przerwan INT0 i INT1 dla przycisków
    PORTD |= (1 << PD2) | (1 << PD3); // wlaczenie rezystora pull-up dla wejsc
z przyciskami
    MCUCR |= (1 << ISC01) | (1 << ISC11); // zbocze opadajace dla INT0 i INT1
    GICR |= (1 << INT0) | (1 << INT1); // wlaczenie przerwan dla INT0 i INT1

    clock_set_mode(RUN); // ustawienie trybu RUN (praca normalna)
}

/*************************************************************/
// funkcja ustawiajaca sekundy
/*************************************************************/
void clock_set_seconds(uint8_t seconds)
{
    current_time.seconds = seconds;
}

/*************************************************************/
// funkcja ustawiajaca minuty
/*************************************************************/
void clock_set_minutes(uint8_t minutes)
{
    current_time.minutes = minutes;
}

/*************************************************************/
// funkcja ustawiajaca godzine
/*************************************************************/
void clock_set_hours(uint8_t hours)
{
    current_time.hours = hours;
}

/*************************************************************/
// funkcja zwiekszajaca czas o 1s
/*************************************************************/
void add_second(void)
{
    if (current_time.seconds < 59)
        current_time.seconds++;
    else
    {
```

```
        current_time.seconds = 0;
        add_minute();
    }
}

/*************************************************************/
// funkcja zwiekszajaca czas o 1m
/*************************************************************/
void add_minute(void)
{
    if (current_time.minutes < 59)
        current_time.minutes++;
    else
    {
        current_time.minutes = 0;
        add_hour();
    }
}

/*************************************************************/
// funkcja zwiekszajaca czas o 1h
/*************************************************************/
void add_hour(void)
{
    if (current_time.hours < 23)
        current_time.hours++;
    else
        current_time.hours = 0;
}

/*************************************************************/
// odswieza dane w buforze wyswietlaczy
/*************************************************************/
void refresh_displays(void)
{
    led7seg_stop();

    buffer[0] = current_time.hours / 10;
    buffer[1] = current_time.hours % 10;
    buffer[2] = current_time.minutes / 10;
    buffer[3] = current_time.minutes % 10;
    buffer[4] = current_time.seconds / 10;
    buffer[5] = current_time.seconds % 10;

    if (clock_mode == ADJ && blinker)
    {
        uint8_t idx = cursor_adj * 2;
        buffer[idx] = buffer[idx + 1] = -1;
    }
```

```
    led7seg_start();
}

/*************************************************************/
// ustawia tryb pracy zegara
/*************************************************************/
void clock_set_mode(enum modes mode)
{
    clock_mode = mode;

    if (mode == RUN)
        CURSOR_RESET
    else // if (mode == ADJ)
        CURSOR_NEXT
}

/*************************************************************/
// obsluga przerwania timer1 (do odmierzania sekund)
/*************************************************************/
ISR(TIMER1_COMPA_vect)
{
    clock_tick = 1; // ustawienie flagi do uzycia w petli glownej
    blinker ^= 1; // zmiana stanu
}

/*************************************************************/
// obsluga nacisniecia przycisku SELECT
/*************************************************************/
ISR(INT0_vect)
{
    clock_set_mode(ADJ);
}

/*************************************************************/
// obsluga nacisniecia przycisku SET
/*************************************************************/
ISR(INT1_vect)
{
    clock_set_mode(RUN);
}
```

led7seg.c, ***led7seg.h*** – obsługa wyświetlaczy 7-segmentowych

Wzorce dla wyświetlaczy zapisano w tablicy *segments***.** Multipleksowanie odbywa się z częstotliwością 180Hz (6 * 30Hz dla płynnego oprazu). Jest realizowane w wektorze przerwania *timera 0*.

```
#ifndef LED7SEG_H_
```

```h
#define LED7SEG_H_

extern volatile uint8_t buffer[6]; // bufor cyfr dla wyswietlaczy


void init_led7seg(void); // inicjalizacja wyswietlaczy

void led7seg_start(void); // wlaczenie multipleksowania
void led7seg_stop(void); // wylaczenie multipleksowania

#endif
```

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

#include "led7seg.h"


#define SELECT_DISPLAY(x)  PORTB = (1 << x); // wybor wyswietlacza
(multipleksowanie)
#define SET_DIGIT(num)     PORTC = ((num < 0 || num > 9) ? 0 : segments[num]);
// ustawienie libczby na wyswietlaczu
#define CURSOR_NEXT        cursor = (cursor + 1) % 6; // przesuniecie kursora


volatile uint8_t cursor = 0; // aktualnie wybrany wyswietlacz przy
multipleksowaniu
volatile uint8_t buffer[6]; // bufor cyfr dla wyswietlaczy

// tablica wzorcow dla wyswietlacza 7-segmentowego
volatile const uint8_t segments[10] =
{
//    a           b           c           d           e           f
    g
    (1 << PC0) | (1 << PC1) | (1 << PC2) | (1 << PC3) | (1 << PC4) | (1 <<
PC5)             , // 0
                 (1 << PC1) | (1 <<
PC2)                                                  , // 1
    (1 << PC0) | (1 << PC1)             | (1 << PC3) | (1 << PC4)
|              (1 << PC6), // 2
    (1 << PC0) | (1 << PC1) | (1 << PC2) | (1 << PC3)
|                          (1 << PC6), // 3
                 (1 << PC1) | (1 << PC2)             | (1 << PC5) | (1 <<
PC6)             , // 4
    (1 << PC0) |             (1 << PC2) | (1 << PC3) |             (1 <<
PC5) | (1 << PC6), // 5
    (1 << PC0) |             (1 << PC2) | (1 << PC3) | (1 << PC4) | (1 <<
PC5) | (1 << PC6), // 6
```

```
    (1 << PC0) | (1 << PC1) | (1 <<
PC2)                                                    , // 7
    (1 << PC0) | (1 << PC1) | (1 << PC2) | (1 << PC3) | (1 << PC4) | (1 <<
PC5) | (1 << PC6), // 8
    (1 << PC0) | (1 << PC1) | (1 << PC2) | (1 << PC3) |             (1 <<
PC5) | (1 << PC6)  // 9
};


/*************************************************************/
// inicjalizacja wyswietlaczy
/*************************************************************/
void init_led7seg(void)
{
    // PB[0-5] jako wyjscia - do wyboru wyswietlacza
    DDRB |= (1 << PB0) | (1 << PB1) | (1 << PB2) | (1 << PB3) | (1 << PB4) |
(1 << PB5);

    // PC[0-6] jako wyjscia - sterowanie wyswietlaczem 7-segmentowym
    DDRC |= (1 << PC0) | (1 << PC1) | (1 << PC2) | (1 << PC3) | (1 << PC4) |
(1 << PC5) | (1 << PC6);

    // inicjalizacja timera 0 i ustawienie na 180 Hz (do multipleksowania)
    TCCR0 |= (1 << WGM01); // tryb CTC
    TCCR0 |= (1 << CS02) | (1 << CS00); // preskaler 1024
    OCR0 = 171; // (1 / czestotliwosc) / (1 / (F_CPU / preskaler))) - 1
    led7seg_start();
}

/*************************************************************/
// wlaczenie multipleksowania
/*************************************************************/
void led7seg_start(void)
{
    TIMSK |= (1 << OCIE0); // zezwolenie na przerwanie timera 0
}

/*************************************************************/
// wylaczenie multipleksowania
/*************************************************************/
void led7seg_stop(void)
{
    TIMSK &= ~(1 << OCIE0); // brak zezwolenia na przerwanie timera 0
}

/*************************************************************/
// obsluga przerwania timer0 (do multipleksowania)
/*************************************************************/
ISR(TIMER0_COMP_vect)
```

```
{
    SELECT_DISPLAY(cursor);   // aktywacja wyswietlacza
    SET_DIGIT(buffer[cursor]); // ustawienie liczby na wyswietlaczu
    CURSOR_NEXT; // przejscie do nastepnego wyswietlacza
}
```

makefile – do kompilacji

Ustawiono mikrokontroler ATmega32, 8MHz i ścieżki do plików.

```
# Hey Emacs, this is a -*- makefile -*-
#----------------------------------------------------------------------------
# WinAVR Makefile Template written by Eric B. Weddington, J□rg Wunsch, et al.
#
# Released to the Public Domain
#
# Additional material for this makefile was written by:
# Peter Fleury
# Tim Henigan
# Colin O'Flynn
# Reiner Patommel
# Markus Pfaff
# Sander Pool
# Frederik Rouleau
# Carlos Lamas
#
#----------------------------------------------------------------------------
# On command line:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF.
#
# make extcoff = Convert ELF to AVR Extended COFF.
#
# make program = Download the hex file to the device, using avrdude.
#                Please customize the avrdude settings below first!
#
# make debug = Start either simulavr or avarice as specified for debugging,
#              with avr-gdb or avr-insight as the front end for debugging.
#
# make filename.s = Just compile filename.c into the assembler code only.
#
# make filename.i = Create a preprocessed source file for use in submitting
#                   bug reports to the GCC project.
#
# To rebuild project do "make clean" then "make all".
```

```
#----------------------------------------------------------------------------


# MCU name
MCU = atmega32


# Processor frequency.
#     This will define a symbol, F_CPU, in all source code files equal to the
#     processor frequency. You can then use this symbol in your source code to
#     calculate timings. Do NOT tack on a 'UL' at the end, this will be done
#     automatically to create a 32-bit value in your source code.
#     Typical values are:
#         F_CPU =  1000000
#         F_CPU =  1843200
#         F_CPU =  2000000
#         F_CPU =  3686400
#         F_CPU =  4000000
#         F_CPU =  7372800
#         F_CPU =  8000000
#         F_CPU = 11059200
#         F_CPU = 14745600
#         F_CPU = 16000000
#         F_CPU = 18432000
#         F_CPU = 20000000
F_CPU = 8000000


# Output format. (can be srec, ihex, binary)
FORMAT = ihex


# Target file name (without extension).
TARGET = main


# Object files directory
#     To put object files in current directory, use a dot (.), do NOT make
#     this an empty or blank macro!
OBJDIR = .


# List C source files here. (C dependencies are automatically generated.)
SRC = $(TARGET).c  led7seg.c clock.c adc.c


# List C++ source files here. (C dependencies are automatically generated.)
CPPSRC =
```

```
# List Assembler source files here.
#     Make them always end in a capital .S.  Files ending in a lowercase .s
#     will not be considered source files but generated files (assembler
#     output from the compiler), and will be deleted upon "make clean"!
#     Even though the DOS/Win* filesystem matches both .s and .S the same,
#     it will preserve the spelling of the filenames, and gcc itself does
#     care about how the name is spelled on its command-line.
ASRC =


# Optimization level, can be [0, 1, 2, 3, s].
#     0 = turn off optimization. s = optimize for size.
#     (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = s


# Debugging format.
#     Native formats for AVR-GCC's -g are dwarf-2 [default] or stabs.
#     AVR Studio 4.10 requires dwarf-2.
#     AVR [Extended] COFF format requires stabs, plus an avr-objcopy run.
DEBUG = dwarf-2


# List any extra directories to look for include files here.
#     Each directory must be seperated by a space.
#     Use forward slashes for directory separators.
#     For a directory that has spaces, enclose it in quotes.
EXTRAINCDIRS =


# Compiler flag to set the C Standard level.
#     c89   = "ANSI" C
#     gnu89 = c89 plus GCC extensions
#     c99   = ISO C99 standard (not yet fully implemented)
#     gnu99 = c99 plus GCC extensions
CSTANDARD = -std=gnu99


# Place -D or -U options here for C sources
CDEFS = -DF_CPU=$(F_CPU)UL


# Place -D or -U options here for ASM sources
ADEFS = -DF_CPU=$(F_CPU)


# Place -D or -U options here for C++ sources
CPPDEFS = -DF_CPU=$(F_CPU)UL
#CPPDEFS += -D__STDC_LIMIT_MACROS
#CPPDEFS += -D__STDC_CONSTANT_MACROS
```

```
#---------------- Compiler Options C ----------------
#  -g*:          generate debugging information
#  -O*:          optimization level
#  -f...:        tuning, see GCC manual and avr-libc documentation
#  -Wall...:     warning level
#  -Wa,...:      tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CFLAGS = -g$(DEBUG)
CFLAGS += $(CDEFS)
CFLAGS += -O$(OPT)
CFLAGS += -funsigned-char
CFLAGS += -funsigned-bitfields
CFLAGS += -fpack-struct
CFLAGS += -fshort-enums
CFLAGS += -Wall
CFLAGS += -Wstrict-prototypes
#CFLAGS += -mshort-calls
#CFLAGS += -fno-unit-at-a-time
#CFLAGS += -Wundef
#CFLAGS += -Wunreachable-code
#CFLAGS += -Wsign-compare
CFLAGS += -Wa,-adhlns=$(<:%.c=$(OBJDIR)/%.lst)
CFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
CFLAGS += $(CSTANDARD)


#---------------- Compiler Options C++ ----------------
#  -g*:          generate debugging information
#  -O*:          optimization level
#  -f...:        tuning, see GCC manual and avr-libc documentation
#  -Wall...:     warning level
#  -Wa,...:      tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CPPFLAGS = -g$(DEBUG)
CPPFLAGS += $(CPPDEFS)
CPPFLAGS += -O$(OPT)
CPPFLAGS += -funsigned-char
CPPFLAGS += -funsigned-bitfields
CPPFLAGS += -fpack-struct
CPPFLAGS += -fshort-enums
CPPFLAGS += -fno-exceptions
CPPFLAGS += -Wall
CPPFLAGS += -Wundef
#CPPFLAGS += -mshort-calls
#CPPFLAGS += -fno-unit-at-a-time
#CPPFLAGS += -Wstrict-prototypes
```

```
#CPPFLAGS += -Wunreachable-code
#CPPFLAGS += -Wsign-compare
CPPFLAGS += -Wa,-adhlns=$(<:%.cpp=$(OBJDIR)/%.lst)
CPPFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
#CPPFLAGS += $(CSTANDARD)


#---------------- Assembler Options ----------------
#  -Wa,...:   tell GCC to pass this to the assembler.
#  -adhlns:   create listing
#  -gstabs:   have the assembler create line number information; note that
#             for use in COFF files, additional information about filenames
#             and function names needs to be present in the assembler source
#             files -- see avr-libc docs [FIXME: not yet described there]
#  -listing-cont-lines: Sets the maximum number of continuation lines of hex
#       dump that will be displayed for a given single line of source input.
ASFLAGS = $(ADEFS) -Wa,-adhlns=$(<:%.S=$(OBJDIR)/%.lst),-gstabs,--listing-
cont-lines=100


#---------------- Library Options ----------------
# Minimalistic printf version
PRINTF_LIB_MIN = -Wl,-u,vfprintf -lprintf_min

# Floating point printf version (requires MATH_LIB = -lm below)
PRINTF_LIB_FLOAT = -Wl,-u,vfprintf -lprintf_flt

# If this is left blank, then it will use the Standard printf version.
PRINTF_LIB =
#PRINTF_LIB = $(PRINTF_LIB_MIN)
#PRINTF_LIB = $(PRINTF_LIB_FLOAT)


# Minimalistic scanf version
SCANF_LIB_MIN = -Wl,-u,vfscanf -lscanf_min

# Floating point + %[ scanf version (requires MATH_LIB = -lm below)
SCANF_LIB_FLOAT = -Wl,-u,vfscanf -lscanf_flt

# If this is left blank, then it will use the Standard scanf version.
SCANF_LIB =
#SCANF_LIB = $(SCANF_LIB_MIN)
#SCANF_LIB = $(SCANF_LIB_FLOAT)


MATH_LIB = -lm


# List any extra directories to look for libraries here.
```

```
#     Each directory must be seperated by a space.
#     Use forward slashes for directory separators.
#     For a directory that has spaces, enclose it in quotes.
EXTRALIBDIRS =



#---------------- External Memory Options ----------------

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# used for variables (.data/.bss) and heap (malloc()).
#EXTMEMOPTS = -Wl,-Tdata=0x801100,--defsym=__heap_end=0x80ffff

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# only used for heap (malloc()).
#EXTMEMOPTS = -Wl,--section-start,.data=0x801100,--defsym=__heap_end=0x80ffff

EXTMEMOPTS =



#---------------- Linker Options ----------------
#  -Wl,...:     tell GCC to pass this to linker.
#    -Map:      create map file
#    --cref:    add cross reference to  map file
LDFLAGS = -Wl,-Map=$(TARGET).map,--cref
LDFLAGS += $(EXTMEMOPTS)
LDFLAGS += $(patsubst %,-L%,$(EXTRALIBDIRS))
LDFLAGS += $(PRINTF_LIB) $(SCANF_LIB) $(MATH_LIB)
#LDFLAGS += -T linker_script.x



#---------------- Programming Options (avrdude) ----------------

# Programming hardware
# Type: avrdude -c ?
# to get a full listing.
#
AVRDUDE_PROGRAMMER = stk500v2

# com1 = serial port. Use lpt1 to connect to parallel port.
AVRDUDE_PORT = com1    # programmer connected to serial device

AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
#AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep


# Uncomment the following if you want avrdude's erase cycle counter.
```

```
# Note that this counter needs to be initialized first using -Yn,
# see avrdude manual.
#AVRDUDE_ERASE_COUNTER = -y

# Uncomment the following if you do /not/ wish a verification to be
# performed after programming the device.
#AVRDUDE_NO_VERIFY = -V

# Increase verbosity level.  Please use this when submitting bug
# reports about avrdude. See <http://savannah.nongnu.org/projects/avrdude>
# to submit bug reports.
#AVRDUDE_VERBOSE = -v -v

AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)



#---------------- Debugging Options ----------------

# For simulavr only - target MCU frequency.
DEBUG_MFREQ = $(F_CPU)

# Set the DEBUG_UI to either gdb or insight.
# DEBUG_UI = gdb
DEBUG_UI = insight

# Set the debugging back-end to either avarice, simulavr.
DEBUG_BACKEND = avarice
#DEBUG_BACKEND = simulavr

# GDB Init Filename.
GDBINIT_FILE = __avr_gdbinit

# When using avarice settings for the JTAG
JTAG_DEV = /dev/com1

# Debugging port used to communicate between GDB / avarice / simulavr.
DEBUG_PORT = 4242

# Debugging host used to communicate between GDB / avarice / simulavr,
normally
#     just set to localhost unless doing some sort of crazy debugging when
#     avarice is running on a different computer.
DEBUG_HOST = localhost
```

```
#============================================================================


# Define programs and commands.
SHELL = sh
CC = avr-gcc
OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
SIZE = avr-size
AR = avr-ar rcs
NM = avr-nm
AVRDUDE = avrdude
REMOVE = rm -f
REMOVEDIR = rm -rf
COPY = cp
WINSHELL = cmd


# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = --------  end  --------
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_COFF = Converting to AVR COFF:
MSG_EXTENDED_COFF = Converting to AVR Extended COFF:
MSG_FLASH = Creating load file for Flash:
MSG_EEPROM = Creating load file for EEPROM:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling C:
MSG_COMPILING_CPP = Compiling C++:
MSG_ASSEMBLING = Assembling:
MSG_CLEANING = Cleaning project:
MSG_CREATING_LIBRARY = Creating library:




# Define all object files.
OBJ = $(SRC:%.c=$(OBJDIR)/%.o) $(CPPSRC:%.cpp=$(OBJDIR)/%.o)
$(ASRC:%.S=$(OBJDIR)/%.o)

# Define all listing files.
LST = $(SRC:%.c=$(OBJDIR)/%.lst) $(CPPSRC:%.cpp=$(OBJDIR)/%.lst)
$(ASRC:%.S=$(OBJDIR)/%.lst)
```

```
# Compiler flags to generate dependency files.
GENDEPFLAGS = -MMD -MP -MF .dep/$(@F).d


# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mmcu=$(MCU) -I. $(CFLAGS) $(GENDEPFLAGS)
ALL_CPPFLAGS = -mmcu=$(MCU) -I. -x c++ $(CPPFLAGS) $(GENDEPFLAGS)
ALL_ASFLAGS = -mmcu=$(MCU) -I. -x assembler-with-cpp $(ASFLAGS)




# Default target.
all: begin gccversion sizebefore build sizeafter end

# Change the build target to build a HEX file or a library.
build: elf hex eep lss sym
#build: lib


elf: $(TARGET).elf
hex: $(TARGET).hex
eep: $(TARGET).eep
lss: $(TARGET).lss
sym: $(TARGET).sym
LIBNAME=lib$(TARGET).a
lib: $(LIBNAME)



# Eye candy.
# AVR Studio 3.x does not check make's exit code but relies on
# the following magic strings to be generated by the compile job.
begin:
    @echo
    @echo $(MSG_BEGIN)

end:
    @echo $(MSG_END)
    @echo


# Display size of file.
HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
ELFSIZE = $(SIZE) --mcu=$(MCU) --format=avr $(TARGET).elf

sizebefore:
```

```
    @if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_BEFORE); $(ELFSIZE);
\
    2>/dev/null; echo; fi

sizeafter:
    @if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE);
\
    2>/dev/null; echo; fi



# Display compiler version information.
gccversion :
    @$(CC) --version



# Program the device.
program: $(TARGET).hex $(TARGET).eep
    $(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH) $(AVRDUDE_WRITE_EEPROM)


# Generate avr-gdb config/init file which does the following:
#     define the reset signal, load the target file, connect to target, and
set
#     a breakpoint at main().
gdb-config:
    @$(REMOVE) $(GDBINIT_FILE)
    @echo define reset >> $(GDBINIT_FILE)
    @echo SIGNAL SIGHUP >> $(GDBINIT_FILE)
    @echo end >> $(GDBINIT_FILE)
    @echo file $(TARGET).elf >> $(GDBINIT_FILE)
    @echo target remote $(DEBUG_HOST):$(DEBUG_PORT)  >> $(GDBINIT_FILE)
ifeq ($(DEBUG_BACKEND),simulavr)
    @echo load  >> $(GDBINIT_FILE)
endif
    @echo break main >> $(GDBINIT_FILE)

debug: gdb-config $(TARGET).elf
ifeq ($(DEBUG_BACKEND), avarice)
    @echo Starting AVaRICE - Press enter when "waiting to connect" message
displays.
    @$(WINSHELL) /c start avarice --jtag $(JTAG_DEV) --erase --program --file
\
    $(TARGET).elf $(DEBUG_HOST):$(DEBUG_PORT)
    @$(WINSHELL) /c pause

else
    @$(WINSHELL) /c start simulavr --gdbserver --device $(MCU) --clock-freq \
```

```
    $(DEBUG_MFREQ) --port $(DEBUG_PORT)
endif
    @$(WINSHELL) /c start avr-$(DEBUG_UI) --command=$(GDBINIT_FILE)



# Convert ELF to COFF for use in debugging / simulating in AVR Studio or
VMLAB.
COFFCONVERT = $(OBJCOPY) --debugging
COFFCONVERT += --change-section-address .data-0x800000
COFFCONVERT += --change-section-address .bss-0x800000
COFFCONVERT += --change-section-address .noinit-0x800000
COFFCONVERT += --change-section-address .eeprom-0x810000



coff: $(TARGET).elf
    @echo
    @echo $(MSG_COFF) $(TARGET).cof
    $(COFFCONVERT) -O coff-avr $< $(TARGET).cof


extcoff: $(TARGET).elf
    @echo
    @echo $(MSG_EXTENDED_COFF) $(TARGET).cof
    $(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof



# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
    @echo
    @echo $(MSG_FLASH) $@
    $(OBJCOPY) -O $(FORMAT) -R .eeprom -R .fuse -R .lock $< $@

%.eep: %.elf
    @echo
    @echo $(MSG_EEPROM) $@
    -$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" \
    --change-section-lma .eeprom=0 --no-change-warnings -O $(FORMAT) $< $@ ||
exit 0

# Create extended listing file from ELF output file.
%.lss: %.elf
    @echo
    @echo $(MSG_EXTENDED_LISTING) $@
    $(OBJDUMP) -h -S -z $< > $@
```

```
# Create a symbol table from ELF output file.
%.sym: %.elf
    @echo
    @echo $(MSG_SYMBOL_TABLE) $@
    $(NM) -n $< > $@



# Create library from object files.
.SECONDARY : $(TARGET).a
.PRECIOUS : $(OBJ)
%.a: $(OBJ)
    @echo
    @echo $(MSG_CREATING_LIBRARY) $@
    $(AR) $@ $(OBJ)


# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(OBJ)
%.elf: $(OBJ)
    @echo
    @echo $(MSG_LINKING) $@
    $(CC) $(ALL_CFLAGS) $^ --output $@ $(LDFLAGS)


# Compile: create object files from C source files.
$(OBJDIR)/%.o : %.c
    @echo
    @echo $(MSG_COMPILING) $<
    $(CC) -c $(ALL_CFLAGS) $< -o $@


# Compile: create object files from C++ source files.
$(OBJDIR)/%.o : %.cpp
    @echo
    @echo $(MSG_COMPILING_CPP) $<
    $(CC) -c $(ALL_CPPFLAGS) $< -o $@


# Compile: create assembler files from C source files.
%.s : %.c
    $(CC) -S $(ALL_CFLAGS) $< -o $@


# Compile: create assembler files from C++ source files.
%.s : %.cpp
    $(CC) -S $(ALL_CPPFLAGS) $< -o $@
```

```
# Assemble: create object files from assembler source files.
$(OBJDIR)/%.o : %.S
    @echo
    @echo $(MSG_ASSEMBLING) $<
    $(CC) -c $(ALL_ASFLAGS) $< -o $@


# Create preprocessed source for use in sending a bug report.
%.i : %.c
    $(CC) -E -mmcu=$(MCU) -I. $(CFLAGS) $< -o $@


# Target: clean project.
clean: begin clean_list end

clean_list :
    @echo
    @echo $(MSG_CLEANING)
    $(REMOVE) $(TARGET).hex
    $(REMOVE) $(TARGET).eep
    $(REMOVE) $(TARGET).cof
    $(REMOVE) $(TARGET).elf
    $(REMOVE) $(TARGET).map
    $(REMOVE) $(TARGET).sym
    $(REMOVE) $(TARGET).lss
    $(REMOVE) $(SRC:%.c=$(OBJDIR)/%.o)
    $(REMOVE) $(SRC:%.c=$(OBJDIR)/%.lst)
    $(REMOVE) $(SRC:.c=.s)
    $(REMOVE) $(SRC:.c=.d)
    $(REMOVE) $(SRC:.c=.i)
    $(REMOVEDIR) .dep


# Create object files directory
$(shell mkdir $(OBJDIR) 2>/dev/null)


# Include the dependency files.
-include $(shell mkdir .dep 2>/dev/null) $(wildcard .dep/*)


# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion \
build elf hex eep lss sym coff extcoff \
clean clean_list program debug gdb-config
```

Wnioski

Symulacja działa prawidłowo, tj. czas jest odmierzany co jedną sekundę oraz można ustawić parametry przy pomocy potencjometru. Dalsza optymalizacja jest możliwa poprzez ustawienie krótkich funkcji jako *inline* lub jako makra (dyrektywy preprocesora). Podział projektu na pliki znacząco ułatwił pisanie kodu. Repozytorium GIT dostępne pod adresem: grzesieklagocki/SCR_zaliczeniowy: Projekt zaliczeniowy z Systemów Czasu Rzeczywistego (github.com)