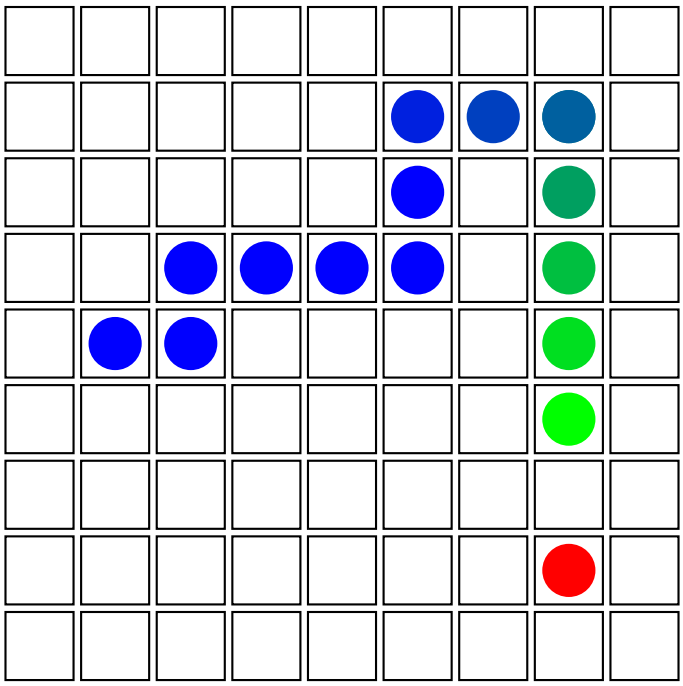
Snake

Report Project Microcontrollers

Axel Barbelanne (325318) and Elio Wanner (326429) - Group 100 - EL

May 31, 2022



1 Project Presentation

1.1 General Information and Project Requirements

For a final project in the EE-208 course, we have been asked to develop a project of our choice on a microcontroller in assembly code. The requirements given were the following:

- Use of the LED matrix as well as at least two other input/output devices
- Use of interruptions, macros
- Inclusion of a user interface
- Display of value on LCD or on the terminal

1.2 Basic description of our project

We have developed a game of snake that can be controlled either via motion control or by a remote control.

The game is shown on an 8×8 LED matrix and the score is shown on an LCD screen. This score shows the active score as well as the all-time highscore, which is also saved during power off. Other useful information can be read on the board's LED's as explained in 3.1. A button allows the user to toggle between two modes of input: an infrared distance sensor allowing for motion control with the user's hand; an infrared remote control. Both of these devices allow for the user to make the snake turn in real-time on the game. There is a reset button that always brings the player back to the starting conditions. There are also two buttons that allow the player to change the speed and hence the difficulty of the game.

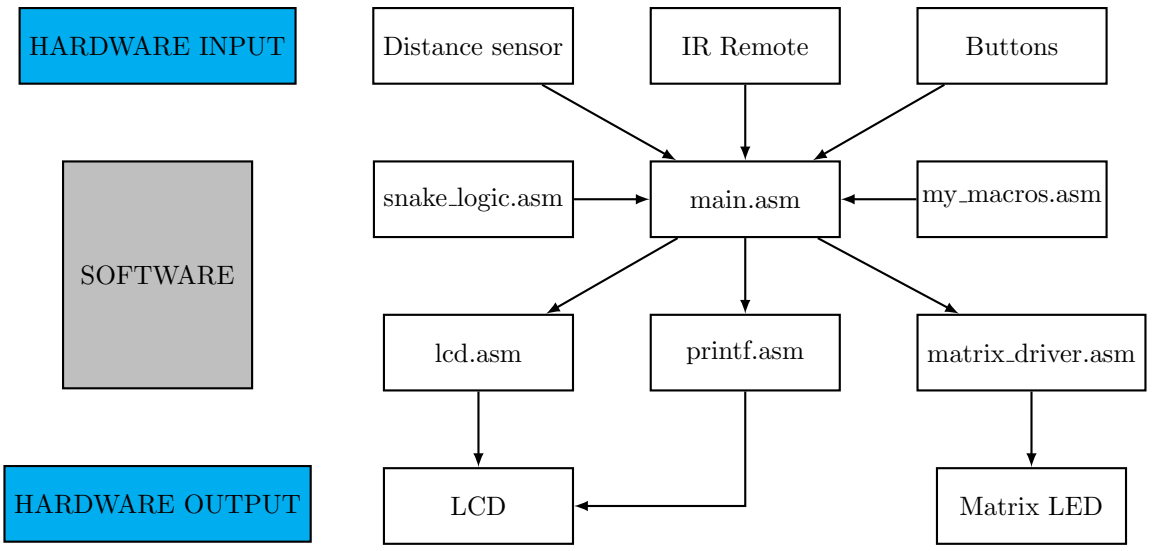
The goal of the game is to control a snake and make it eat apples to grow larger and larger, increasing your score. The player must however be careful not to make the snake collide with itself which becomes progressively harder as the snake gets longer.

2 Project Structure

2.1 General program setup

The following schematic gives an overview over the project. The flow of information is from the Hardware input layer, through the software layer to the hardware output layer.

The code for the logic of the game is in *snake_code.asm*. The *main.asm* pulls all the strings together. It receives the inputs and initiates the interrupt subroutines. The drivers for the distance sensor and the IR remote are together in a file named *input_drivers.asm*[1]



[1]Our drivers for the distance sensor, IR sensor and matrix LED are all originally based on drivers from the course

2.2 Closer inspection of *main.asm*

The file *main.asm* is the heart of the program, setting up both the hardware and software required and calling subroutines in different modules when necessary. It can be divided into three main parts: the interrupts; the definitions and inclusions; the setup (reset). The former of the three is the most important one in our code and hence the one that will be discussed in more detail.

There are five external interrupts and two counters that work as interrupts. Four of the interrupts are used by the buttons 0-3 and their respective functions are: reset; toggle mode to switch between the IR sensor and distance sensor; speed decrease; speed increase. All of these buttons are measured on a falling edge and have been debounced to keep only the first click. This has been done by skipping the content of the interruption temporarily through a *ready* bit which is set again later in the code. The fifth interruption is the IR sensor, whose protocol is presented in more detail in 2.5.1. This interruption is toggled on and off in the **EIMSK** register during a click of the *toggle* button.

The counters 0 and 1 work as output compare and overflow respectively. Counter 0 works while the active mode is the distance sensor and makes it measure at an interval of $65ms$. This short interval is there to ensure that we have multiple points of measure even if we are in the fastest game mode. By having this relatively short period of overflow, we minimise the time it takes for the sensor to update and detect a new position. This as we have realised that, in practice, the sensor does not adapt immediately when changing values. Counter 1 works with a variable output compare value which can be adjusted with the buttons 2 and 3. This allows the user to vary the length of one turn between $270ms$ and $1s$. The instructions for this interrupt sequence includes much of the game's logistics including the movement and display of the snake and the apple. This can take quite long but it always remains negligible compared to the duration of a turn and hence does not cause any issues.

2.3 Memory usage

2.3.1 Registers

The following list only includes the description of a selected number of registers.

Register	Name	Main usage
r16	w	working register*
r17	_w	working register for interrupts*
r21	a3	Status bits (see below)
r22	b0	Direction bits: Only the two LSB's are used to indicate the current direction of the snake
r23	b1	Value of turn counter output compare (speed)
r25	b3	Current score (between 3 and 64)

*register used over small distances (e.g. in macros)

a3	-	-	SPRD	GOST	IRRD	BTRD	ADRD	TOGL

SPRD: Speed button ready; **GOST**: Game over state; **IRRD**: IR sensor ready; **BTRD**: Reset and toggle buttons ready; **ADRD**: ADC/distance sensor ready; **TOGL**: Toggle state - current mode

2.3.2 SRAM

We defined three different sections in the SRAM for our program to work in.

The size of the "Display" is defined by the number of pixels times three (because every pixel is associated with three colour values). It stores the GRB (Green-Red-Blue) values for all the pixel in chronological order. This place in memory is used by the matrix driver to actually display it on the matrix.

Name	From	To	Size
Display	0x400	0x4C0	192
Apple	0x4C1	0x4C1	1
Snake	0x4C2	0x503	65

The "Apple" contains a single value ranging from zero to 63. It designates the place of the apple on the game field.

The "Snake" stores where the snake is on the game field. The head of the snake is always at 0x4C2, followed by the all the other snake parts and finishing with the tail (at 0x4C2 + length of snake). This means that not the whole reserved space in the SRAM is used. The non-used places are filled with 0xff. This was chosen as it's easily distinguishable from the rest of the snake, as the rest of the snake only has values ranging from 0 to 63. The length of this space is 65. This is one longer than the maximal length of the snake. This decision was made due to the way of how the code makes the snake move: It first copies the complete snake backwards and then deletes the tail (if no apple is eaten).

2.3.3 Internal EEPROM

We used the internal EEPROM to store the highscore. It is stored at address 200. During the intial ISP programming, the EEPROM entirely set to 0xff. Upon restart of the card it is checked if the highscore is at 0xff. If it is, it resets it to zero. Otherwise it leaves the highscore as it is. This ensures the safeguarding of the high-score even if you unplug the card and re-power it.

2.4 Game logic

There is one subroutine making the logic of the game which is called *autoAdvanceSnake*. Here are the individual logic steps chronologically:

1. Calculating the position of the next head: This is done by using the current head position and advancing it by one unit in the direction of the direction bits.
2. Check if the apple is on the same spot as the new head
3. Add new head at 0x4C2 (start of reserved section for Snake).
4. Shift the snake backwards in memory
5. Delete tail (if new head is not on apple)
6. If an apple is eaten, a new apple will be placed randomly[2]
7. Check if the snake is colliding with itself. If so, a game over is initiated

2.5 Data extraction and analysis

2.5.1 IR sensor

The IR sensor works with a protocol called NEC. This is a protocol for the remote to communicate effectively with the sensor at a short distance. The downfall of this protocol is that it takes some time (up to around $70ms$). This can be a problem as, being in an interrupt, there is a risk that it blocks other interrupts occurring in the meantime causing unwanted delays and potentially even errors. In our case however, this is not a true issue as a delay of around $70ms$ in the worst case scenario would not even be noticeable. Furthermore, the delay usually occurs in between the overflows of the game's turn counter meaning it will not affect the turn duration in any way (the counters keep going during interruptions).

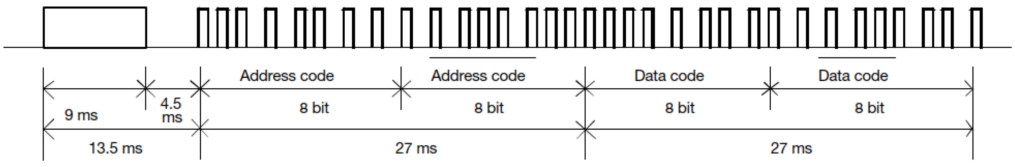


Figure 1: IR NEC protocol structure (from datasheet - /IR/dataform.pdf)

[2]For this we use a pseudo-random algorithm: we take the position of the tail, add the direction bits and subtract 10. This feels random from the point of view of the user.

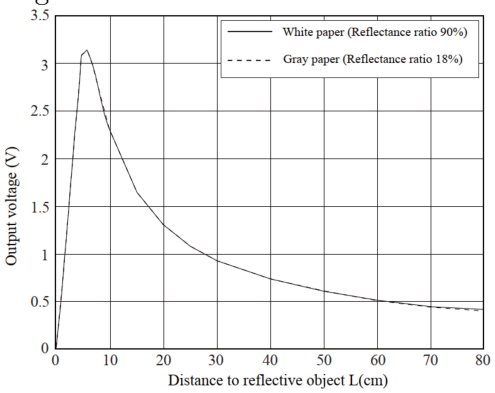
2.5.2 Distance sensor

The distance sensor we use returns a value between 0 and 1023 proportional to the distance it is from a reflective object. We used the graph on the right to find the corresponding distances [3]. Using this we were able to define critical values for turns.

In the program, a counter calls the *measure* function to make the measurement as long as the active mode is the distance sensor (and not the remote mode).

The value obtained by the sensor is delayed slightly during the measurement. To make sure to have a responsive system, we measure much more frequently than once a turn and we don't use all of the calculated values.

Figure 2: Distance sensor conversions



3 Game Setup, Control Instructions

3.1 Hardware setup

1. Connect Matrix LED: Connect GND, Vcc and Pin 7 of Port D with jumper cables.
2. Buttons: With two jumpers, connect Pin 0 and Pin 1 of Port D with Pin 0 and Pin 1 on "Switches".
3. Connect all pins of Port B with all pins of "LEDS" (use the flat cable in the kit)
4. Put the LCD panel on the long pin header on the other side of the board
5. IR receiver (M2 module): Connect GND, Vcc and Pin 7 of Port E with jumper cables
6. SHARP Distance sensor (M4 module): Connect GND, Vcc and Pin 3 of Port F with jumper cables

All of these connections can be seen on Figure 3. We added a sheet of paper on the Matrix LED to make the lights smoother and not to burn your eyes.

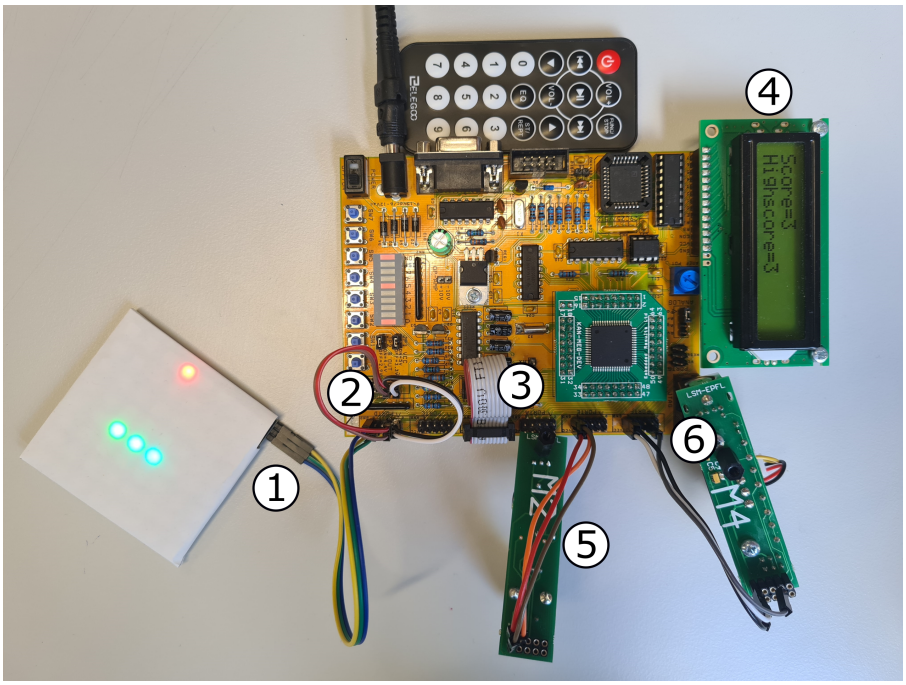


Figure 3: Full view of how cables are plugged in

[3] Graph from datasheet - /Sharp 2Y0A21/gp2y0a21yk_e-3493.pdf

We are hence using four external peripherals, two of which take a user input. There are additional user inputs from the four buttons $0-3$. These work continuously, in parallel to sensors, whereas the two sensors will never function simultaneously. Below are the functions of each button and each LED as well as a figure showing the order of the LED's

Button	Function
0	Restart game
1	Toggle game modes
2	Decreases game speed
3	Increases game speed
-	-

LED	Function
0	Control mode *
1	-
2	IR code received
3	Left motion turn
4	Left motion turn

LED	Function
5	Straight motion
6	Right motion turn
7	Right motion turn
-	-
P	Power on

*0 for remote control, 1 for motion control

Figure 4: Organisation of LED's

3.1.1 Game rules

The goal of the game is to have the highest length of snake possible. The snake consists of individual parts, which each follow its predecessor. The head (which is completely green) defines where the snake's head currently is. Exiting the game field on one side makes it loop around and the snake comes back in from the other side.

To increase the length you need to eat the apples (in red) which are placed randomly on the game field.

The game finishes when the snake bumps into itself. This is shown with a red screen.

By pressing button 0 you can restart the game.

3.1.2 Remote control mode

This is the default control mode. You change the direction of the snake by pressing one of the following buttons: "left", "right", "VOL+" (up), "VOL-"(down). These buttons control the orientation of the snake in an absolute way, meaning whatever direction you press will lead to the snake heading in the same direction on the screen. The user must be careful to not press the opposite direction of the snake's movement, as this will make the snake turn around and collide with itself, leading to a *Game Over*.

3.1.3 Motion control mode

This additional control method allows you to direct the snake with your hand by measuring the distance you are at from the sensor. There are four zones: left-turn ($5-10cm$ from the sensor), straight ($10-20cm$), right-turn ($20-\pm 50cm$) and idle ($>50cm$). In order to view the current zone of your hand, you can look at the LED's which will indicate whether you are turning left, heading straight or turning right (see 4). This mode controls the orientation of the snake in a relative way, meaning for instance that turning left will make the snake turn left compared to the direction in which it was previously headed.

3.1.4 Switching between modes

The changing of modes can be done at any time by clicking the button 1. The default state is the remote control, but a click of the reset will let the user continue in whatever mode you are currently on. The LED 0 will show you the current state you are on (see 4).

3.1.5 Resetting the game

The button 1 allows the user to restart the game. This is necessary when the player is in a game over state, but it can be done at any time.

3.1.6 Changing game speeds

By pressing on buttons 2 and 3, the player can slow down or speed up the game respectively in order to change the difficulty of the game. Note that there are minimum and maximum values for the speed.

4 Conclusion

4.1 Requirements review

- *Use of the LED matrix as well as at least two other input/output devices*
  - We have used the LED Matrix, LCD Screen, Distance sensor and the Remote Control. So we have three other i/o devices, which fulfills this requirement.
- *Use of interruptions, macros*
  - We use timer interrupts to advance the game and to use the distance sensor and external interrupts for the buttons and the remote control. We have also created our own small libraries of macros called "my_macros.asm".
- *Inclusion of a user interface*
  - We implemented two input methods (remote and motion control). Furthermore there is a LCD showing the current and the highest score, as well as LED's showing different states of the game. The main part of the user interface is the matrix LED as it displays the game.
- *Display of value on LCD or on the terminal*
  - The LCD is used to show the current score and the all-time high-score.

4.2 Issues and difficulties

We spent a lot of time on getting the IR sensor to give us data. We changed the available IR sensor file to change the start signal to an interrupt but this initially caused many problems, due in part to the synchronisation difference.

We also had some issues with the buttons which would cause mutiple interrupts when clicked. We solved this by adding *button ready* bits (see 2.3.1) and setting this to false temporarily when the button is clicked. Hence, for the next bounces the button would not be available.

Another issue we had was in the hardware of our project. We wanted to use multiple peripherals that required an access to common i/o ports. This was for instance the case with the AC/DC converter and the IR sensor on Port E, and with the buttons and matrix display on Port D. To solve this we used jumper cables to connect the peripherals

4.3 Possible improvements

There is currently no way of pausing the game. This would have been a nice to have, but we didn't think it was strictly necessary as it's a rather quick game anyways.

We also would have liked to add more original features to the game to make it more interesting and challenging to the player. We for instance thought of occasionally inverting the directions. A start menu would also have been a nice addition

We currently have most of the code existing within interrupts. While for us this is not a problem, it would most likely be neater to have this code be in the *main* subroutine in our code and have the interrupts only toggle status bits in registers on which the *main* acts

```
; file  main.asm   target ATmega128L-4MHz-STK300
; purpose timers 0,1,2 overflows

.include "macros.asm"       ; include macro definitions
.include "definitions.asm"  ; include register/constant definitions

;note: b0: ir_ready // button_ready // data_ready // toggle_mode

; === interrupt vector table ===
.org    0
    rjmp        reset
.org            INT0addr        ;reset button
    jmp         ext_int0
.org            INT1addr        ;toggle button
    jmp         ext_int1
.org            INT2addr        ;speed up button
    jmp         ext_int2
.org            INT3addr        ;slow down button
    jmp         ext_int3
.org            INT7addr        ;IR remote interrupt
    jmp         ext_int7
.org            OC0addr         ;snake turn timer
    rjmp        output_compare0
.org            OVF2addr        ;measure timer
    rjmp        overflow2
.org            ADCCaddr        ;AC/DC converter
    rjmp        ADCCaddr_sra


.include "lcd.asm"
.include "printf.asm"
.include "my_macros.asm"
.include "matrix_driver.asm"
.include "input_drivers.asm"    ;distance sensor and IR sensor drivers
.include "snake_logic.asm"      ;game logic code
.include "eeprom.asm"

.equ    TOGGLE_BIT = 0          ;mode toggle
.equ    ADC_RDY_BIT = 1     ;ADC converter ready
.equ    BTN_RDY_BIT = 2     ;toggle button ready
.equ    IR_RDY_BIT = 3          ;IR sensor ready
.equ    GO_STATE = 4            ;game over state
.equ    SPD_RDY_BIT = 5     ;speed buttons ready

.equ    TURN_R_VAL = 0x0320
.equ    TURN_L_VAL = 0x0190
.equ    TOO_FAR_VAL = 0x0070    ;value too far => ignored
.equ    DIST_PRESC = 3
```

```
49 .equ    DEF_SNAKE_TIMER = 200   ; 1 => 7.8ms
50 .equ    HIGH_SPD_LIM = 50
51 .equ    LOW_SPD_LIM = 244
52
53 ;delay times for IR protocol
54 .equ    T2 = 15532
55 .equ    T1 = 1180
56
57 ; === interrupt service routines ====
58 ext_int0:   ;reset
59     cbr         a3, (1 << GO_STATE) ;clear game over
60     rcall       setupSnake
61     rcall       setupApple
62     ldi         b1,DEF_SNAKE_TIMER  ;set default speed
63     out         OCR0,b1
64     reti
65
66 ext_int1:   ;toggle mode (IR sensor/distance sensor)
67     sbrs        a3,BTN_RDY_BIT          ;debounce (check if button is
   ready)
68     rjmp        ext_int1_end
69     cbr         a3,(1 << BTN_RDY_BIT)   ;button not ready until small
   delay
70     _EORI       a3, (1 << TOGGLE_BIT)   ;invert toggle pin
71     in          _w, EIMSK               ;toggle IR sensor interrupt
72     _EORI       _w, 0x80
73     out         EIMSK, _w
74     INVP        PORTB, 0                ;toggle mode LED
75     ext_int1_end:
76     reti
77
78 ext_int2:   ;slow down snake
79     sbrs        a3,SPD_RDY_BIT          ;debounce (check if button is
   ready)
80     rjmp        ext_int2_end
81     cbr         a3,(1 << SPD_RDY_BIT)   ;button not ready until small
   delay
82     INC_LIM10   b1, LOW_SPD_LIM         ;increase by 10 with a limit
   (slow down snake)
83     out         OCR0,b1
84     ext_int2_end:
85     reti
86
87 ext_int3:   ;speed up snake
88     sbrs        a3,SPD_RDY_BIT      ;debounce (check if button is ready)
89     rjmp        ext_int3_end
90     cbr         a3,(1 << SPD_RDY_BIT)       ;button not ready until small
   delay
91     DEC_LIM10   b1, HIGH_SPD_LIM            ;decrease by 10 with a limit
```

```
(speed up snake)
92      out         OCR0,b1
93      ext_int3_end:
94      reti
95
96  ext_int7:   ;IR sensor
97      sbrs        a3,IR_RDY_BIT
98      rjmp        end_interrupt
99      cbr         a3, (1 << IR_RDY_BIT)
100     WAIT_US     T2              ; wait for timeout
101     clc                         ; clearing carry
102     rcall       prep
103     cbi         PORTB,2         ;indicate received code
104     end_interrupt:
105     reti
106
107 overflow2:  ;counter for distance sensor measurement
108     sbrc        a3, GO_STATE
109     rjmp        after_measure
110     sbrs        a3, TOGGLE_BIT      ;check toggle mode before measurement
111     rcall       measure
112     after_measure:
113     sbr         a3, (1 << SPD_RDY_BIT)  ;make speed changing buttons
available again
114     reti
115
116 output_compare0:    ;counter for snake turns
117     sbrc        a3, GO_STATE        ;check game over state, skip if true
118     rjmp        output_compare0_end
119     sbrs        a3, TOGGLE_BIT      ;check toggle
120     rcall       change_dir
121     sbrc        a3, TOGGLE_BIT
122     sbi         PORTB,2
123     rcall       autoAdvanceSnake        ;move snake
124     rcall       loadSnakeToDisplay      ;display snake
125     rcall       loadAppleToDisplay      ;display apple
126     sbrc        a3, GO_STATE
127     rjmp        output_compare0_end     ;end if game over
128     rcall       show_on_matrix          ;shows Display in SRAM to
hardware
129     rcall       ws2812b4_reset          ;resets hardware-display
130     rcall       showScore               ;show score on LCD screen
131     sbr         a3,(1 << BTN_RDY_BIT)   ;Make buttons available again
132     sbr         a3, (1 << IR_RDY_BIT)
133     output_compare0_end:
134     reti
135
136
```

```
137 ADCCaddr_sra:
138     sbr         a3,(1 << ADC_RDY_BIT)   ;ADC data ready
139     reti
140
141 ; === initialisation (reset) ===
142 reset:
143     LDSP        RAMEND          ; load stack pointer (SP)
144     OUTI        PORTB,0xff      ; turn LEDs off
145     OUTI        DDRB,0xff       ; LED's:output
146     OUTI        DDRE,0x7f       ; IR sensor input, ADC output
147     OUTI        DDRA, 0xff      ;used by LCD
148     OUTI        DDRC, 0xff      ;used by LCD
149     OUTI        DDRD, 0b11110000;configure first two pins as inputs (for
    interrupt)
150                                 ;pin7 is used for matrix LED
151     OUTI        EIMSK, 0x8f     ;allow all interrupts at startup
152     ldi         w, 0b10101010
153     sts         EICRA, w        ;detect on falling edge
154     ldi         w, 0b11000000
155     sts         EICRB, w        ;detect on rising edge
156
157     OUTI        ASSR, (1<<AS0)  ; clock from TOSC1
158     OUTI        TCCR2,5         ; set prescaler
159
160     OUTI        TCCR0,(1<<CTC0)+5   ;clear on compare
161     ldi         b1, DEF_SNAKE_TIMER
162     out         OCR0,b1             ;set snake speed
163     OUTI        TIMSK,(1<<TOIE2) + (1<<OCIE0)
164
165     OUTI        ADCSR,(1<<ADEN)+(1<<ADIE)+6 ; AD Enable, AD int. enable,
    PS=CK/64
166     OUTI        ADMUX,3                     ; select channel POT
167     cbr         a3,ADC_RDY_BIT
168     cbr         a3, (1 << GO_STATE)         ;remomve game over state
169     sbr         a3, (1 << IR_RDY_BIT)       ;make button available
170     sbr         a3,(1 << TOGGLE_BIT)        ;default: remote
171
172     rcall       LCD_init            ; initialize the LCD
173     rcall       ws2812b4_init
174     rcall       setupSnake
175     rcall       setupApple
176
177     ldi     xl, low(200)        ;initializing highscore
178     ldi     xh, high(200)
179     rcall   eeprom_load
180     cpi     a0, 0xff            ;on first boot up eeprom is 0xff, so need
    to clear it
181     brne    PC+3
```

```
182     clr     a0
183     rcall   eeprom_store        ;loads highscore to a0
184
185     sei                 ; set global interrupt
186
187 ; === main program ===
188 main:
189     rjmp        main
```

```
/*
 * snake_logic.asm
 *
 *  Created: 03/05/2022 20:30:20
 *   Author: Elio Wanner and Axel Barbelanne
 */

game_over:
    ldi zl,low(0x0400)
    ldi zh,high(0x0400)
    ldi a0, 64

game_over_loop:
    load_led_value_i 10,0,0     ;show red screen to show failure (R=10)
    dec     a0
    brne    game_over_loop
    ldi     zl,low(0x0400)
    ldi     zh,high(0x0400)
    rcall   show_on_matrix      ;actually load red pixels on matrix
    sbr     a3, (1 << 4)

    ret

;SUBROUTINES-----------------------------------------------------------------

;set all LEDs to OFF
reset_all_LEDs:
    ldi     zl,low(0x0400)      ;Where display starts
    ldi     zh,high(0x0400)
    ldi     a0, 192             ;3*64 (3 colours, 64 pixels)
reset_ram_internal:
    ldi     w, 0x00             ;write a 0 to turn LED off
    st      z+,w
    dec     a0
    brne    reset_ram_internal  ;loop until all off
    ldi     zl,low(0x0400)      ;reset z to display start
    ldi     zh,high(0x0400)
ret

;show what is stored in the SRAM at 0x0400 + 64
show_on_matrix:
    ldi     zl,low(0x0400)
    ldi     zh,high(0x0400)
    _LDI    r0,64
show_on_matrix_loop:
    ld      a0, z+      ;load values into a0,a1,a2
    ld      a1, z+      ;because that is what the matrix driver will
```

```
output
    ld      a2, z+

    cli
    rcall   ws2812b4_byte3wr    ;output the values to display
    sei

    dec     r0
    brne    show_on_matrix_loop
    ldi     zl,low(0x0400)      ;reset z pointer
    ldi     zh,high(0x0400)
ret

;Shows the length of the snake (score) on the LCD
showScore:
    loadSnakeLength         ;b3=snake length

    ldi     xl, low(200)
    ldi     xh, high(200)
    clr     a0
    rcall   eeprom_load         ;loads highscore to a0

    cp      a0, b3
    brsh    showScore_showNow   ;if lower --> don't change highscores
    mov     a0, b3
    ldi     xl, low(200)
    ldi     xh, high(200)
    rcall   eeprom_store        ;store new highscore in eeprom

showScore_showNow:
    rcall   LCD_clear       ;Clear LCD display
    rcall   LCD_home
    PRINTF  LCD             ;call display
    .db     "Score=",FDEC,b+3, LF,"Highscore=",FDEC, a ,0   ;Show Score
on the first line and highscore on the second line

ret

;This sets the SnakeUnit in SRAM to 0xff
setupSnake:
    ldi xl,low(0x04C2)      ;where Snake starts
    ldi xh,high(0x04C2)
    ldi a0, 64
setupSnake_internal:
    ldi w, 0xff             ;set ram in SnakeUnit to 0xff (to indicate
the absence of a snake part)
    st  x+, w
    dec a0
```

```
 93     brne setupSnake_internal
 94
 95     ;INITIAL SNAKE
 96     ldi xl,low(0x04C2)      ;where Snake is
 97     ldi xh,high(0x04C2)
 98     ldi w, 0x23             ;initial length = 3 at the positions 0x23,
    0x24, 0x25
 99     st  x+, w
100     ldi w, 0x24
101     st  x+, w
102     ldi w, 0x25
103     st  x+, w
104
105     ldi     b0, 0           ;initial direction
106 ret
107
108 setupApple:
109     ldi xl,low(0x04C1)      ;where Snake is
110     ldi xh,high(0x04C1)
111     ldi w, 0x20             ;first apple location
112     st  x+, w
113 ret
114
115 ;Loads snake from snake storage in SRAM into the display (also in SRAM)
116 ;z points to the place where it will be stored
117 ;x points to the original place of the snake
118 loadSnakeToDisplay:
119     ldi     zl,low(0x0400)      ;where Display is
120     ldi     zh,high(0x0400)
121
122     ldi     xl,low(0x04C2)      ;where Snake is
123     ldi     xh,high(0x04C2)
124
125     rcall   reset_all_LEDs      ;first reset display to all OFF
126     ldi     a0,0                ;define head colour (G=40)
127     ldi     a1,40
128     ldi     a2,0
129 loopThroughSnake:
130     ld      w, x+
131
132     ldi     zl,low(0x0400)      ;where Display is
133     ldi     zh,high(0x0400)
134
135     ;add thrice, because for each colour
136     ADDZ    w
137     ADDZ    w
138     ADDZ    w
139
```

```
140     load_led_value a0,a1,a2      ;COLOUR OF THE SNAKE
141
142     tst      a1                  ;Stop colour change if fully blue
143     breq     PC+3
144     subi     a1, 5               ;Gradually change colour to blue
145     ADDI     a2, 5
146
147     ld       w, x
148     cpi      w, 0xff             ;if 0xff is found that means the snake is
    fully displayed
149     brne     loopThroughSnake
150 ret
151
152 loadAppleToDisplay:
153     ldi      zl,low(0x0400)      ;where Display is
154     ldi      zh,high(0x0400)
155
156     ldi      xl,low(0x04C1)      ;where Apple is
157     ldi      xh,high(0x04C1)
158
159     ldi      a0,50               ;Apple Colour
160     ldi      a1,0
161     ldi      a2,0
162     ld       w, x+
163     ADDZ     w
164     ADDZ     w
165     ADDZ     w
166     load_led_value a0,a1,a2
167 ret
168
169 ;   -Advances snake into the direction of the directionByte
170 ;   -Checks for self eating
171 ;   -Loops the head around if snake surpassing field limit
172 autoAdvanceSnake:
173     ldi      zl,low(0x0400)      ;where Display is
174     ldi      zh,high(0x0400)
175
176     ldi      xl,low(0x04C2)      ;where Snake is
177     ldi      xh,high(0x04C2)
178
179     ldi      yl, low(0x04C1)     ;where Apple is
180     ldi      yh, high(0x04C1)
181
182     ;load new head
183     ld       a0, x                   ;store current head position in a0
184     findNewHeadPosition     a0, b0  ;calculate new head position and put
    in w
185
```

```
186     ldi     xl,low(0x04C2)          ;reset because checkSelfEating
    modifies it
187     ldi     xh,high(0x04C2)
188     ld      a2, y                   ;get apple position
189     cp      w, a2                   ;see if new head on apple
190     brne    PC+3                    ;if on apple
191     ldi     a2, 1                   ;   a2=1
192     rjmp    PC+2
193     ldi     a2,0                    ;else: a2=0
194     st      x+, w                   ;   add new head
195
196 autoAdvanceSnake_movingInternal:    ;Shifts the snake in the memory
197     ld      a1, x
198     st      x+,a0
199     cpi     a1, 0xff
200     breq    autoAdvanceSnake_endMove
201
202     ld      a0, x
203     st      x+, a1
204     cpi     a0, 0xff
205     breq    autoAdvanceSnake_endMove
206
207     rjmp    autoAdvanceSnake_movingInternal
208 autoAdvanceSnake_endMove:
209     cpi     a2, 1       ;if on apple, don't remove tail (meaning snake
    gets longer)
210     breq    autoAdvance_setNewApple
211     ldi     w, 0xff
212     st      -x, w       ;removing tail
213     rjmp    autoAdvance_dontSetNewApple     ;only setNewApple when one is
    eaten
214 autoAdvance_setNewApple:
215     ld      w, -x
216     ldi     xl,low(0x04C1)      ;where apple is stored
217     ldi     xh,high(0x04C1)
218     add     w, b0               ;Do some pseudo random operations to
    randomize new apple position
219     subi    w, -10
220     cpi     w, 63
221     brlo    PC+2                ;check if apple is in range (not
    exceeding the display) --> jump
222     ldi     w, 2                ;if out of bounds set apple to position 2
223     st      x, w
224 autoAdvance_dontSetNewApple:
225
226 checkSelfEating                 ;check if the snake is eating itself -->
    game over
227 ret
```

```
/*
 * input_drivers.asm
 *
 *  Created: 27/05/2022 16:53:24
 *   Author: Axelb
 */

; DISTANCE SENSOR

measure:

    sbi         ADCSR,ADSC          ; AD start conversion

    in          d0,ADCL             ; read low byte first
    in          d1,ADCH             ; read high byte second

    cbr         a3,ADC_RDY_BIT      ;Wait until next ADC data ready

    ret

change_dir:
    sbi         PORTB,3
    sbi         PORTB,4
    sbi         PORTB,5
    sbi         PORTB,6
    sbi         PORTB,7
    mov         _w, b0

    CPI2        d1, d0, TOO_FAR_VAL     ; straight as default
    brlo        led_4
    CPI2        d1, d0, TURN_L_VAL      ; left turn
    brlo        led_3
    CPI2        d1, d0, TURN_R_VAL      ; right turn
    brsh        led_5
    ;show position on LED's
    led_4:
    cbi         PORTB,5
    rjmp        end_check
    led_5:
    cbi         PORTB,7
    cbi         PORTB,6
    ADDI        _w,1
    rjmp        end_check
    led_3:
    cbi         PORTB,3
    cbi         PORTB,4
    subi        _w, 1

```

```
49      end_check:
50      clr         b0
51      ANDI        _w, 0b11
52      add         b0, _w      ;sets the new direction
53      ret
54
55  ; IR REMOTE
56
57  prep:
58      PUSH4       a0, a1, a2, a3
59      PUSH4       d0, d1, d2, d3
60
61      CLR2        a3,a2           ; clear 2-byte register
62      CLR2        a1,a0
63      ldi         _w,16           ; load bit-counter
64
65  addr:
66      P2C         PINE,IR         ; move Pin to Carry (P2C, 4 cycles)
67      ROL2        a3,a2           ; roll carry into 2-byte reg (ROL2, 2
    cycles)
68      sbrc        a2,0            ; (branch not taken, 1 cycle; taken 2
    cycles)
69      rjmp        rdz_a           ; (rjmp, 2 cycles)
70      WAIT_US     (T1 - 4.5)
71      DJNZ        _w,addr         ; Decrement and Jump if Not Zero (true, 2
    cycles; false, 1 cycle)
72      jmp         next_a          ; (jmp, 3 cycles)
73
74  rdz_a:                          ; read a zero
75      WAIT_US     (2*T1 - 5.5)
76      DJNZ        _w,addr         ; Decrement and Jump if Not Zero
77
78  next_a:
79      MOV2    d1,d0, a3, a2   ; store current address
80      MOV2        a1,a0,a3,a2
81      ldi         _w,16           ; load bit-counter
82      clc
83      CLR2    a3,a2
84
85  data:
86      P2C     PINE,IR
87      ROL2        a3,a2
88      sbrc        a2,0
89      rjmp        rdz_d
90      WAIT_US     (T1 - 4.5)
91      DJNZ        _w,data
92      jmp         next_b
93
```

```
 94 rdz_d:
 95     WAIT_US      (2*T1 - 5.5)
 96     DJNZ         _w,data
 97
 98 next_b:
 99     MOV2         d3,d2,a3, a2    ; store current command
100
101 data_proc01:                     ; detect repeated code
102     _CPI             d3, 0xff
103     brne         data_proc02
104     _CPI             d2, 0xff
105     brne         data_proc02
106     _CPI             d1, 0xff
107     brne         data_proc02
108     _CPI             d0, 0xff
109     brne         data_proc02
110
111 display_repeat:
112     MOV4         a1,a0,a3,a2,c3,c2,c1,c0     ; display the last correct
    code, i.e,
113     rjmp         data_recover
114
115 data_proc02:                                 ; detect transmission error
116     com          d1
117     cpse         d0, d1
118     brne         data_recover
119     com          d3
120     cpse         d2, d3
121     brne         data_recover
122
123 display_correct:
124     com          a2                          ; complement b0 (chip
    delivers the complement)
125     com          a3
126     com          a0
127     com          a1
128     CPI2         a1, a0, 0xff00          ;check if correct address
129     brne         end_display
130     ;check each direction value individually
131     CPI2         a3, a2, 0x3dc2
132     brne         PC+3
133     ldi          b0, 0
134     rjmp         store_correct
135     CPI2         a3, a2, 0x9d62
136     brne         PC+3
137     ldi          b0, 1
138     rjmp         store_correct
139     CPI2         a3, a2, 0xdd22
```

```
140     brne        PC+3
141     ldi         b0, 2
142     rjmp        store_correct
143     CPI2        a3, a2, 0x57a8
144     ldi         b0, 3
145     store_correct:
146     MOV4        c3,c2,c1,c0,a1,a0,a3,a2     ; storing correct code to
    display/use in
147     end_display:
148     rjmp        data_recover
149
150
151 data_recover:
152     POP4        d0, d1, d2, d3
153     POP4        a0, a1, a2, a3
154     ret
```

```
; file  matrix_driver.asm   target ATmega128L-4MHz-STK300
; purpose send data to ws2812b using 4 MHz MCU and standard I/O port
;         display and parallel process (blinking LED0)
; usage: buttons on PORTD, ws2812 on PORTD (bit 1)
;        press button 0
;       a pattern is stored into memory and displayed on the array
;       LED0 blinks fast; when button0 is pressed and released, LED1
;       akcnowledges and the pattern displayed on the array moves by
;       one memory location
; warnings: 1/2 timings of pulses in the macros are sensitive
;           2/2 intensity of LEDs is high, thus keep intensities
;               within the range 0x00-0x0f, and do not look into
;               LEDs
; 20220315 AxS

;.include "macros.asm"      ; include macro definitions
;.include "definitions.asm" ; include register/constant definitions

; WS2812b4_WR0  ; macro ; arg: void; used: void
; purpose: write an active-high zero-pulse to PD1
.macro  WS2812b4_WR0
    clr u
    sbi PORTD, 7
    out PORTD, u
    nop
    nop
    ;nop    ;deactivated on purpose of respecting timings
    ;nop
.endm

; WS2812b4_WR1  ; macro ; arg: void; used: void
; purpose: write an active-high one-pulse to PD1
.macro  WS2812b4_WR1
    sbi PORTD, 7
    nop
    nop
    cbi PORTD, 7
    ;nop    ;deactivated on purpose of respecting timings
    ;nop

.endm


; ws2812b4_init     ; arg: void; used: r16 (w)
; purpose: initialize AVR to support ws2812
ws2812b4_init:
    ;OUTI   DDRD,0x02
ret
```

```
49
50 ; ws2812b4_byte3wr  ; arg: a0,a1,a2 ; used: r16 (w)
51 ; purpose: write contents of a0,a1,a2 (24 bit) into ws2812, 1 LED
   configuring
52 ;     GBR color coding, LSB first
53 ws2812b4_byte3wr:
54
55     ldi w,8
56 ws2b3_starta0:
57     sbrc a0,7
58     rjmp    ws2b3w1
59     WS2812b4_WR0            ; write a zero
60     rjmp    ws2b3_nexta0
61 ws2b3w1:
62     WS2812b4_WR1
63 ws2b3_nexta0:
64     lsl a0
65     dec w
66     brne ws2b3_starta0
67
68     ldi w,8
69 ws2b3_starta1:
70     sbrc a1,7
71     rjmp    ws2b3w1a1
72     WS2812b4_WR0            ; write a zero
73     rjmp    ws2b3_nexta1
74 ws2b3w1a1:
75     WS2812b4_WR1
76 ws2b3_nexta1:
77     lsl a1
78     dec w
79     brne ws2b3_starta1
80
81     ldi w,8
82 ws2b3_starta2:
83     sbrc a2,7
84     rjmp    ws2b3w1a2
85     WS2812b4_WR0            ; write a zero
86     rjmp    ws2b3_nexta2
87 ws2b3w1a2:
88     WS2812b4_WR1
89 ws2b3_nexta2:
90     lsl a2
91     dec w
92     brne ws2b3_starta2
93
94 ret
95
```

```
96 ; ws2812b4_reset    ; arg: void; used: r16 (w)
97 ; purpose: reset pulse, configuration becomes effective
98 ws2812b4_reset:
99     cbi PORTD, 7
100    WAIT_US 50  ; 50 us are required, NO smaller works
101 ret
```

```
/*
 * my_macros.asm
 *
 *  Created: 03/05/2022 20:59:05
 *   Author: eliow
 */

 ;Stores immediate RGB values into z register
 .macro load_led_value_i
    ldi w, @1   ; pixel 1 [GRB]
    st  z+,w
    ldi w, @0
    st  z+,w
    ldi w, @2
    st z+,w
.endmacro

;Stores RGB values from register into z register
 .macro load_led_value
    st  z+,@1   ; pixel 1 [GRB]
    st  z+,@0
    st  z+,@2
.endmacro

;finds position for new snake head from dirBits (@1) & current snake (@0)
;stores result in w
.macro findNewHeadPosition
    mov     w, @0   ;copy current Head in w
    cpi     @1, 0   ;see if direction = 0 --> plusX
    breq    plusX
    cpi     @1, 1   ;see if direction = 1 --> plusY
    breq    plusY
    cpi     @1, 2   ;see if direction = 2 --> minusX
    breq    minusX
    cpi     @1, 3   ;see if direction = 3 --> minusY
    breq    m_minusY
    rjmp    error_case

m_minusY:
    jmp minusY

plusX:
    ;move in +x
    dec     w       ;when not on right border
    CPI8    w, 0xff,7,15,23,31,39,47,55
    brne    PC + 2
    ADDI    w, 8    ;rewind when on the right border
    rjmp    finishHeadFinding
```

```
49 plusY:
50     ;move in +y
51     ADDI    w, 8    ;when not on top border
52     CPI8    w, 64,65,66,67,68,69,70,71
53     brne    PC + 2
54     subi    w, 64   ;rewind when on the top border
55     rjmp    finishHeadFinding
56 minusX:
57     ;move in -x
58     inc     w       ;when not on left border
59     CPI8    w, 8,16,24,32,40,48,56,64
60     brne    PC + 2
61     subi    w, 8    ;rewind when on the left border
62     rjmp    finishHeadFinding
63
64 minusY:
65     ;move in -y
66     subi    w, 8    ;when not on bottom border
67     CPI8    w, 0xff,0xfe,0xfd,0xfc,0xfb,0xfa,0xf9,0xf8
68     brne    PC + 2
69     ADDI    w, 64   ;rewind when on the bottom border
70     rjmp    finishHeadFinding
71
72 error_case:
73     ;This should never happen (if dirBit >3)
74 finishHeadFinding:
75     ;yey the head is succesfully found :D
76 .endmacro
77
78 ;checks if first value is equal to one of the following 8
79 .macro CPI8
80     mov     a1, @0
81
82     cpi     a1, @1
83     breq    CPI8_end
84     cpi     a1, @2
85     breq    CPI8_end
86     cpi     a1, @3
87     breq    CPI8_end
88     cpi     a1, @4
89     breq    CPI8_end
90     cpi     a1, @5
91     breq    CPI8_end
92     cpi     a1, @6
93     breq    CPI8_end
94     cpi     a1, @7
95     breq    CPI8_end
96     cpi     a1, @8
```

```
 97      breq    CPI8_end
 98  CPI8_end:
 99
100  .endmacro
101
102  .macro checkSelfEating
103      ldi     xl,low(0x04C2)      ;where Snake is
104      ldi     xh,high(0x04C2)
105      ld      w, x+               ;load in head
106  checkSelfEating_loop:
107      ld      a2, x+
108      cp      w, a2
109      breq    checkSelfEating_game_over   ;if head somewhere where snake
     already is --> game over
110      cpi     a2, 0xff
111      brne    checkSelfEating_loop        ;loop until 0xff found
112
113      ldi     xl,low(0x04C2)              ;reset x pointer
114      ldi     xh,high(0x04C2)
115      rjmp    checkSelfEating_end
116
117  checkSelfEating_game_over:
118      rcall   game_over
119  checkSelfEating_end:
120  .endmacro
121
122  .macro loadSnakeLength
123      ldi     xl,low(0x04C1)      ;where Snake is
124      ldi     xh,high(0x04C1)
125      clr     b3                  ;set length to zero
126  loadSnakeLength_loop:
127      ld      w, x+
128      cpi     w, 0xff             ;check if at end of snake
129      breq    loadSnakeLength_end
130      inc     b3                  ;add one for every snake part found
131      rjmp    loadSnakeLength_loop
132  loadSnakeLength_end:
133      dec     b3                  ;adjust, because it adds before checking
134  .endmacro
135
136   .macro IFAND   ; r1, b1, r2, b2  sets clear if ether bits are '0'
137      sbrs        @0, @1
138      sec
139      sbrs        @2, @3
140      sec
141  .endmacro
142
143  .macro CPI2     ;cpi on 2 bytes
```

```
144     push        _w
145     ldi         w,high(@2)
146     ldi         _w,low(@2)
147     CP2         @0, @1, w, _w
148     pop         _w
149 .endmacro
150 
151 .macro INC_LIM10        ;reg,lim    ;incremente de 5 avec une limite
152     INC_LIM     @0,@1
153     INC_LIM     @0,@1
154     INC_LIM     @0,@1
155     INC_LIM     @0,@1
156     INC_LIM     @0,@1
157     INC_LIM     @0,@1
158     INC_LIM     @0,@1
159     INC_LIM     @0,@1
160     INC_LIM     @0,@1
161     INC_LIM     @0,@1
162 .endmacro
163 
164 .macro DEC_LIM10        ;reg,lim    ;d?cremente de 5 avec une limite
165     DEC_LIM     @0,@1
166     DEC_LIM     @0,@1
167     DEC_LIM     @0,@1
168     DEC_LIM     @0,@1
169     DEC_LIM     @0,@1
170     DEC_LIM     @0,@1
171     DEC_LIM     @0,@1
172     DEC_LIM     @0,@1
173     DEC_LIM     @0,@1
174     DEC_LIM     @0,@1
175 .endmacro
```