Digitala projekt HT 2007
PONG

Emil Klasson, Fredrik Eneroth
Grupp 3

2007-12-04

Digitala projekt

Abstract

This document describes our own construction and implementation of the first video game ever made - Pong. The purpose of the game is to bounce the ball back and forth in a game field. To do this two players handles one paddle each on each side of the field. If a player misses the ball, the other player scores.

Innehåll

1 Inledning

Vem känner inte till det gamla klassiska datorspelet Pong, med två racketar och en boll som ska styras mellan racketerna? Vi tyckte att det var ett utmärkt tillfälle att skapa vår egen version av detta, och få uppleva spelet på nytt.

1.1 Bakgrund

Syftet med kursen Digitala projekt (EDI021) är att ge eleven en inblick i hur samverkan mellan hårdvara och mjukvara fungerar. Gruppen fick i uppgift att bestämma en konstruktionsuppgift som sedan skulle konstrueras, byggas och testas.

1.2 Mål

Valet av konstruktionsuppgift blev ett spel, det gamla anrika Pong mer specifikt. Utöver processorn skulle ett antal tryckknappar och en grafisk LCD-display resultera i en alldeles egen version av spelet.

1.3 Disposition av rapporten

Först i rapporten beskrivs hårdvaran som använts, varpå mjukvaran beskrivs. Avslutningsvis presenteras genomförandet av projektet, med resultat, reflektioner och slutsats.

2 Krav på systemet

Kraven som sattes upp på systemet var:

- Två spelare skall kunna spela.
- Skall klara av poängräkning.
- De två racketerna ska kunna styras samtidigt.
- Skall kunna starta om spelet med en resetknapp.

3 Hårdvara

1. AVR ATMega16
2. Batron 128x64 - Grafisk LCD display
3. Oscillator
4. Knappar

3.1 AVR ATMega16

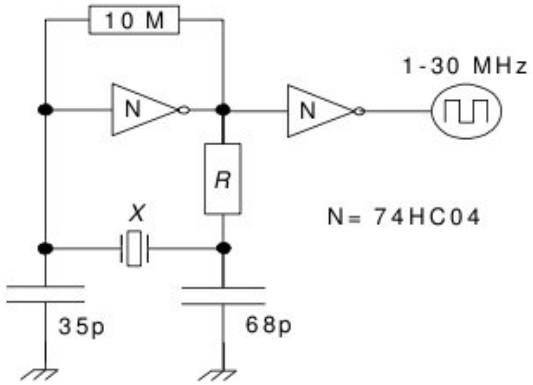
ATMega16 passade bra till vårt projekt eftersom den innehåller i princip allt som behövdes, såsom klocka, RAM-minne, EEPROM, JTAG gränssnitt och A/D-omvandlare. Klockfrekvensen på den interna oscillatorn ligger på 8 MHz vilket är fullt tillräckligt för vårat system.

3.2 Batron 128x64

Batron-displayen består utav två displayer på vardera 64x64 pixlar, som styrs med hjälp av varsin KS0108b LCD styrkontroll. För att hantera de två olika segmenten används två styrsignaler som bestämmer vilken av dom som skall vara aktiv. Pixlarna i displayen är uppdelade så att i x-led är det 128 pixlar, och i y-led är det upplagt som så att de 64 pixlarna är uppdelade i 8 stycken pages på vardera 8 pixlar.

3.3 Oscillator

En 1 MHz kristall användes för att bygga våran oscillator enligt figur 1 som genererar externa timeravbrott till processorn med kontinuerliga tidsintervall. Syftet med den var att få en exakt klocka för att kunna bestämma tidsintervall för olika händelser.



Figur 1: Oscillator där $R = 10k\Omega$

3.4 Knappar

Sex stycken knappar, fungerande som strömbrytare användes till:

- Spelare 1 - styrning upp och ner.
- Spelare 2 - styrning upp och ner.
- Enterknapp.
- Resetknapp.

För att detta skall fungera fick alla knappar kopplas till pull-down-motstånd, förutom resetknappen som är aktiv låg, och därmed får använda sig av ett pull-up-motstånd.

4 Genomförande

Efter att ha gjort en omfattande undersökning om hur AVR ATMega16-processorn och Batron-displayen fungerade påbörjades uppritningen av kopplingsschemat, se figur 3 sist i rapporten. Detta gjordes i programmet PowerLogic som är ett kraftfullt verktyg där man kan importera komponenter och sedan skapa förbindelser mellan olika kopplingspunkter, till exempel processorn och grafikstyrningskretsen.

Nästa steg var att förverkliga kopplingsschemat. Först skapades anslutningen mellan processorn och displayen med hjälp av virning som är mindre tidskrävande än lödning, men fullt tillräckligt för den här sortens signaler. Anslutningen mellan knapparna och processorn anslöts med ett pull-down-motstånd kopplat till varje knapp, förutom resetknappen som är aktiv låg och därför kopplades till ett pull-up-motstånd. En oscillator på 1 MHz kopplades sedan för att skapa timeravbrott till processorn.

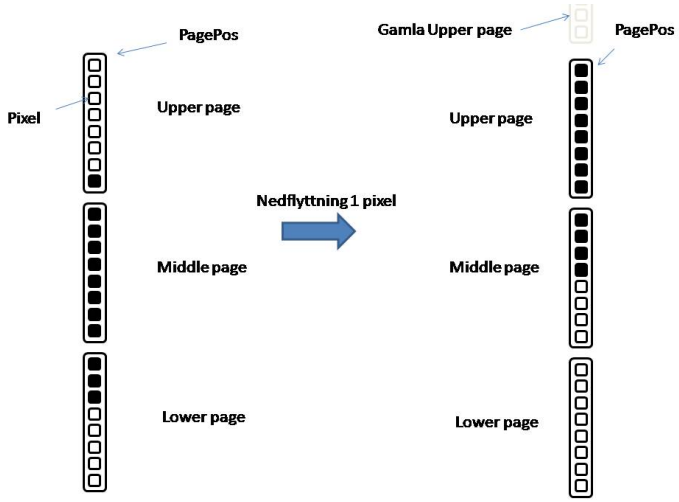
När systemet var uppkopplat påbörjades programmeringen av mjukvaran. Detta steg var det mest tidskrävande steget som krävde en bra förståelse för hur styrningen av displayen fungerade och hur processorn hanterade avbrott med mera. Vid själva utvecklingen av Pong skapades olika delar av programmet stegvis. Först togs spelplanen fram, sedan utvecklades rackerterna och dess styrning, därefter bollen och dess rörelse, och till sist poängberäkning.

Utritningen av racketerna och bollen sker genom att den sista biten, beroende på vilket håll racketen, och/eller bollen skall flyttas, tas bort och den nya sätts ut på andra sidan.

För racketerna blev det komplicerat i och med att vi bestämde oss för att de skulle vara tolv pixlar stora, och därmed kunde sträcka sig över tre olika pages på displayen. Detta hanterar vi genom att hålla koll på de tre aktiva pagesen i en struktur, dvs vi hade en upper, middle och lower där vi sparade bitmönstret för den pagen. För att kunna veta vart pagen befinner sig finns även en variabel pagepos som anger vart upper ska ritas ut. Genom att alltid ha minst en pixel i upper pagen kunde rackerterna sedan lätt ritas ut. Se figur 2 för ett exempel på hur ett pagebyte går till. Då inget pagebyte behöver ske skiftas pixlarna i de aktiva pagesen.

Bollen hålls även den i struktur där vi sparar vart nästa bit ska ritas ut och vilken nästa bit att ta bort är samt x- och y-riktning för bollen. När bollen träffar en vägg eller racket ändras riktningen och de två referenserna

till nästa pixel att rita samt pixeln som ska tas bort ändras.



Figur 2: Page byte när endast nedersta biten finns i upper och den ska flyttas nedåt.

5 Resultat

Efter ett par veckors utvecklande stod då systemet färdigt, och en boll studsade som tänkt mellan två racketar inuti en utritad spelplan. Racketarna styrs av knappar på kopplingsbrädan, och precis som det ursprungligen fungerade så ökar bollens hastighet efterhand och det blir mer komplicerat att rädda densamma. Efter tre missade bollar på en sida utses den andra som vinnare, och skrivs även ut som sådan på skärmen, varpå spelet startas om genom ett tryck på enterknappen i mitten. Härmed ansågs kraven uppfyllda, och projektet genomfört.

6 Reflektioner och slutsats

Det svåra i projektet var att komma igång, då vanan i att läsa datablad inte riktigt var inövad till fullo. Efterhand stod det dock klart att databladen var riktigt viktiga att kontrollera noggrannt för att få det att fungera som tänkt.

Ur utvecklingssynpunkt var det att få koll på hur man fick LCD-displayen att fungera som var det mest tidskrävande. Med detta relativt stora (ur tidsperspektiv) hinder ur vägen flöt arbetet på bra.

Tankarna inför kursens start var att det skulle bli kul och spännande att se om man lyckades att slå ihop kunskaper man fått av tidigare kurser man läst, såsom Digitalteknik, Datorteknik, Elektronik och programmeringskurser, till en helhet och utveckla ett helt system med både hård- och mjukvara. Nu i efterhand kan det sägas att det var precis så kul och intressant som förutspått, om inte ännu mer, när man faktiskt såg resultat och delmål uppfyllas efterhand.

7 Referenser

Referenser

[1] *ATmega16 High-performance AVR 8-bit Microcontroller* http://www.it.lth.se/datablad/Processors/ATmega16.pdf.

[2] *Starta med Power Logic* http://www.it.lth.se/digp/PDF_files/powerlogic/power_logic.pdf.

[3] *Bygga oscillatorer* http://www.it.lth.se/digp/PDF_files/oscillators.pdf

[4] *Batron 128 X 64 Graphic LCD.* http://www.it.lth.se/datablad/display/Batron128x64.pdf

[5] *KS0108b/HD61202Graphic LCD-Driver.* http://www.it.lth.se/datablad/display/ks0108b.pdf

8 Bilaga A

8.1 Headerfiler

8.1.1 buttons.h

```
/*
 * Define Port B
 */
#define BUTTON_CTRL_PORT PORTB
#define BUTTON_CTRL_PIN PINB
#define BUTTON_CTRL_DDR DDRB

#define P1UP  (PINB & 0x01)
#define P1DOWN  (PINB & 0x04)
#define ENTER   (PINB & 0x08)
#define P2UP (PINB & 0x10)
#define P2DOWN (PINB & 0x20)
#define TIME_DELAY 100;

/*
 * Init button port
 */
void init_buttons();

/*
 * Returns the bits from pin B
 */
unsigned char get_buttons();
```

8.1.2 datatypes.h

```
typedef struct paddle {
unsigned char x;
/* Position of upper page (1-7) */
unsigned char pagepos;

unsigned char upper;
```


```
unsigned char middle;
unsigned char lower;

unsigned char points;
} paddle;

unsigned char p1_timer, p2_timer;
```

8.1.3 font.h

```
#define digit_0a 0x7c
#define digit_0b 0x44
#define digit_0c 0x7c

#define digit_1a 0x44
#define digit_1b 0x7c
#define digit_1c 0x40

#define digit_2a 0x74
#define digit_2b 0x54
#define digit_2c 0x5c

#define digit_3a 0x54
#define digit_3b 0x54
#define digit_3c 0x7c

#define digit_4a 0x1c
#define digit_4b 0x10
#define digit_4c 0x7c

#define digit_5a 0x5c
#define digit_5b 0x54
#define digit_5c 0x74

#define digit_6a 0x7c
#define digit_6b 0x54
#define digit_6c 0x74

#define digit_7a 0x04
```

```
#define digit_7b 0x04
#define digit_7c 0x7c

#define digit_8a 0x7c
#define digit_8b 0x54
#define digit_8c 0x7c

#define digit_9a 0x5c
#define digit_9b 0x54
#define digit_9c 0x7c

#define letter_w_a 0x7c
#define letter_w_b 0x20
#define letter_w_c 0x10
#define letter_w_d 0x20
#define letter_w_e 0x7c

#define letter_i_a 0x7c

#define letter_n_a 0x7c
#define letter_n_b 0x0c
#define letter_n_c 0x10
#define letter_n_d 0x60
#define letter_n_e 0x7c

#define letter_e_a 0x7c
#define letter_e_b 0x54
#define letter_e_c 0x54
#define letter_e_d 0x44

#define letter_r_a 0x7c
#define letter_r_b 0x34
#define letter_r_c 0x5c

#define letter_p_a 0x7c
#define letter_p_b 0x14
#define letter_p_c 0x1c
```

8.1.4 game.h

```
#include "datatypes.h"
#define DIRECTION_LEFT  -1
#define DIRECTION_RIGHT 1
#define DIRECTION_UP 1
#define DIRECTION_DOWN -1
#define MAX_Y  61
#define MIN_Y 10

#define FIELD_LEFT_LINE 5
#define FIELD_RIGHT_LINE 122
#define FIELD_TOP_LINE 1
#define FIELD_BOTTOM_LINE 7

#define LEFT_PADDLE_X 7
#define RIGHT_PADDLE_X 120
#define PADDLE_SIZE 12

#define HIT 1
#define MISS 0
#define LARGE_ACC  10
#define SMALL_ACC  5
#define MAX_SPEED  15


typedef struct ball_struct {
unsigned char x_pos;
unsigned char y_pos;

unsigned char x_pos_old;
unsigned char y_pos_old;

unsigned char speed_timer;
char x_direction;
char y_direction;

unsigned char delta_x;
unsigned char delta_y;
```


```
} ball_struct;

void draw_field(void);

void draw_paddle(paddle*);

void draw_ball(ball_struct*);

void init_game_field();
```

8.1.5 lcd.h

```
/*
 * Define Port A for Databus
 */
#define LCD_DATA_PORT PORTA
#define LCD_DATA_PIN PINA
#define LCD_DATA_DDR DDRA

/*
 * Define Port D for control signals
 */
#define LCD_CTRL_PORT PORTD
#define LCD_CTRL_PIN PIND
#define LCD_CTRL_DDR DDRD

/*
 * Define control signals to LCD-Display
 */
#define LCD_CS1P PD0
#define LCD_CS2P PD1
#define LCD_RST PD4
#define LCD_RW PD5
#define LCD_RS PD6
#define LCD_E  PD7

/*
```


```
 * Define
 */
#define SET_CS1 (LCD_CTRL_PORT |= (1 << LCD_CS1P))
#define CLR_CS1 (LCD_CTRL_PORT &= ~(1 << LCD_CS1P))

#define SET_CS2 (LCD_CTRL_PORT |= (1 << LCD_CS2P))
#define CLR_CS2 (LCD_CTRL_PORT &= ~(1 << LCD_CS2P))

#define SET_RST (LCD_CTRL_PORT |= (1 << LCD_RST))
#define CLR_RST (LCD_CTRL_PORT &= ~(1 << LCD_RST))

#define SET_RW (LCD_CTRL_PORT |= (1 << LCD_RW))
#define CLR_RW (LCD_CTRL_PORT &= ~(1 << LCD_RW))

#define SET_RS (LCD_CTRL_PORT |= (1 << LCD_RS))
#define CLR_RS (LCD_CTRL_PORT &= ~(1 << LCD_RS))

#define SET_EN (LCD_CTRL_PORT |= (1 << LCD_E))
#define CLR_EN (LCD_CTRL_PORT &= ~(1 << LCD_E))

#define LCD_CS0 SET_CS1;CLR_CS2;
#define LCD_CS1 CLR_CS1;SET_CS2;
#define LCD_NOCS CLR_CS1;CLR_CS2;

#define DISPLAY_STATUS_BUSY 0x80
#define DISPLAY_SET_X 0xB8
#define DISPLAY_SET_Y 0x40
#define DISPLAY_START_LINE 0xC0
#define DISPLAY_ON_CMD 0x3E
#define ON 0x01

/*
 * Function to write data to LCD-Display
 */
void lcdWriteData(unsigned char);
```

8.2 Källkod

8.2.1 buttons.c

```
#include "buttons.h"
#include "datatypes.h"
#include <avr/io.h>

paddle* left_paddle, *right_paddle;

/*
 * Initialize Buttons
 */
void init_buttons(paddle* left, paddle* right)
{
BUTTON_CTRL_DDR = 0x00;
left_paddle = left;
right_paddle = right;
}

unsigned char get_buttons() {

return BUTTON_CTRL_PIN;
}

/*
 * Check if any buttons are pressed
 */
void check_buttons(void)
{
if (P1UP != 0 && p1_timer == 0)  {
move_paddle_up(left_paddle);
p1_timer = TIME_DELAY;

} else if (P1DOWN != 0 && p1_timer == 0) {
move_paddle_down(left_paddle);
p1_timer = TIME_DELAY;
}
```


```
if (P2UP != 0 && p2_timer == 0) {
move_paddle_up(right_paddle);
p2_timer = TIME_DELAY;

} else if (P2DOWN != 0 && p2_timer == 0) {
move_paddle_down(right_paddle);
p2_timer = TIME_DELAY;
}

}
```

8.2.2 game.c

```
#include "game.h"
#include "buttons.h"
#include <avr/io.h>
#include <avr/delay.h>

ball_struct* ball;
paddle* left_paddle, *right_paddle;
unsigned char y_page, y_offset, step = 1;
unsigned char x_count, y_count;

unsigned char move_ball(unsigned char ball_timer)
{

/*
 * Check if ball is allowed to move.
 */
if (ball_timer == 0) {

/*
 * Check if ball hits the paddles
 */
if (ball->x_direction == 1 && ball->x_pos >= RIGHT_PADDLE_X
&& check_hit(right_paddle) == HIT) {

ball->x_direction = -1;
ball->x_pos_old = ball->x_pos - 1;
```


```
ball->x_pos -= 3;

}
else if (ball->x_direction == -1 && ball->x_pos <= LEFT_PADDLE_X
&& check_hit(left_paddle) == HIT) {
ball->x_direction = 1;
ball->x_pos_old = ball->x_pos + 1;
ball->x_pos += 3;

}

/*
   * Check if ball hits the wall
 */
if (ball->y_direction == 1 && ball->y_pos == MAX_Y) {
ball->y_direction = -1;
ball->y_pos_old = ball->y_pos + 1;

} else if (ball->y_direction == -1 && ball->y_pos == MIN_Y) {
ball->y_direction = 1;
ball->y_pos_old = ball->y_pos;

} else if (ball->x_direction == 1 && ball->x_pos == FIELD_RIGHT_LINE) {
left_paddle->points++;
write_digit(left_paddle->points, FIELD_LEFT_LINE, 0);
if (left_paddle->points == 3)
show_winner(1);
new_ball(1);
} else if (ball->x_direction == -1 && ball->x_pos == FIELD_LEFT_LINE) {
right_paddle->points++;
write_digit(right_paddle->points, FIELD_RIGHT_LINE - 3, 0);
if (right_paddle->points == 3)
show_winner(2);
new_ball(-1);
}

draw_ball(ball);
ball_timer = ball->speed_timer;
}
```


```
return ball_timer;
}

void new_ball(char dir)
{
unsigned char points_p1, points_p2;
clear_ball();
points_p1 = left_paddle->points;
points_p2 = right_paddle->points;
init_paddles();
left_paddle->points = points_p1;
right_paddle->points = points_p2;
init_game_field();
init_ball(dir);
}

void show_winner(unsigned char player)
{
lcdCls();
write_letter('w',44,4);
write_letter('i',50,4);
write_letter('n',52,4);
write_letter('n',58,4);
write_letter('e',64,4);
write_letter('r',69,4);
write_letter('p',78,4);
write_digit(player,82,4);
while (ENTER == 0);

/*
 * Clear screen
 */
lcdCls();

/*
 * Init Paddles
 */
init_paddles(left_paddle, right_paddle);

/*
```


```
 * Initialize the game field
 */
init_game_field();
init_ball(1);

_delay_ms(100);
}

/*
 * Calculate if ball hits the paddle
 */
int check_hit(paddle* pad)
{
unsigned char pad_y, upper_data;

pad_y = pad->pagepos * 8;

upper_data = pad->upper;
while ((upper_data & 0x01) == 0x00) {
upper_data >>= 1;
pad_y += 1;
}

if (ball->y_pos >= pad_y && ball->y_pos <= pad_y + 11) {
switch ((ball->y_pos - pad_y) / 3 + 1) {
case (1) :
if (ball->delta_x > 1 && (ball->delta_x < 5 || ball->y_direction == -1))
ball->delta_x += 1 * ball->y_direction;
else if (ball->delta_x == 1 && ball->y_direction == 1)
ball->delta_x += 1;

if (ball->delta_x == 4 && ball->y_direction == 1)
ball->y_direction = -1;

if (ball->speed_timer > MAX_SPEED)
if (ball->y_direction == -1)
ball->speed_timer -= LARGE_ACC;
else
ball->speed_timer += SMALL_ACC;
return HIT;
```


```
break;
case (2) :
if (ball->delta_x > 1 && (ball->delta_x < 5 || ball->y_direction == 1))
ball->delta_x += 1 * ball->y_direction;
else if (ball->delta_x == 1 && ball->y_direction == 1)
ball->delta_x += 1;

if (ball->delta_x == 5 && ball->y_direction == 1)
ball->y_direction = -1;

if (ball->speed_timer > MAX_SPEED)
if (ball->y_direction == -1)
ball->speed_timer -= SMALL_ACC;
else
ball->speed_timer += 0;
return HIT;
break;
case (3) :
if (ball->delta_x > 1 && (ball->delta_x < 5 || ball->y_direction == 1))
ball->delta_x -= 1 * ball->y_direction;
else if (ball->delta_x == 1 && ball->y_direction == -1)
ball->delta_x += 1;

if (ball->delta_x == 5 && ball->y_direction == -1)
ball->y_direction = 1;

if (ball->speed_timer > MAX_SPEED)
if (ball->y_direction == 1)
ball->speed_timer -= SMALL_ACC;
else
ball->speed_timer += 0;
return HIT;
break;
case (4) :
if (ball->delta_x > 1 && (ball->delta_x < 5 || ball->y_direction == 1))
ball->delta_x -= 1 * ball->y_direction;
else if (ball->delta_x == 1 && ball->y_direction == -1)
ball->delta_x += 1;

if (ball->delta_x == 4 && ball->y_direction == -1)
```


```
ball->y_direction = 1;

if (ball->speed_timer > MAX_SPEED)
if (ball->y_direction == 1)
ball->speed_timer -= LARGE_ACC;
else
ball->speed_timer += SMALL_ACC;
return HIT;
break;
}
}

return MISS;
}

/*
 * Clear ball and paddles.
 */
void clear_ball()
{
unsigned char i;

for(i = 1; i < 8; i++){
lcdGoTo(left_paddle->x - 1, i);
lcdWriteData(0x00);
lcdWriteData(0x00);
lcdGoTo(right_paddle->x, i);
lcdWriteData(0x00);
lcdWriteData(0x00);
}
}

void draw_field(void)
{
unsigned char x;
unsigned char y;


/* Draw TOP line */
lcdGoTo(FIELD_LEFT_LINE,FIELD_TOP_LINE);
```

```
for (x = FIELD_LEFT_LINE; x < FIELD_RIGHT_LINE; x++)
lcdWriteData(0x01);

/* Draw BOTTOM line */
lcdGoTo(FIELD_LEFT_LINE, FIELD_BOTTOM_LINE);
for (x = FIELD_LEFT_LINE; x < FIELD_RIGHT_LINE; x++)
lcdWriteData(0x80);

/* Draw LEFT line */
for (y = 1; y < 8; y++) {
lcdGoTo(FIELD_LEFT_LINE,y);
lcdWriteData(0xff);
}

/* Draw RIGHT line */
for (y = 1; y < 8; y++) {
lcdGoTo(FIELD_RIGHT_LINE,y);
lcdWriteData(0xff);
}
}

void init_game_field()
{
draw_field();
draw_paddle(left_paddle);
draw_paddle(right_paddle);
ball = malloc(sizeof(ball));

write_digit(left_paddle->points, FIELD_LEFT_LINE, 0);
write_digit(right_paddle->points, FIELD_RIGHT_LINE - 3, 0);
}

void init_ball(char dir)
{
unsigned char data = 0x03;

/*
 *Set start values for ball
 */
ball->x_pos = 63;
```


```
ball->y_pos = 36;
ball->speed_timer = 40;
ball->x_direction = dir;
ball->y_direction = 1;
ball->delta_x = 5;
ball->delta_y = 1;
x_count = ball->delta_x;
y_count = ball->delta_y;

calc_y();
data <<= y_offset;

if (y_offset == 7) {
lcdGoTo(ball->x_pos, y_page);
lcdWriteData(data);
lcdWriteData(data);
lcdGoTo(ball->x_pos, y_page + 1);
lcdWriteData(0x01);
lcdWriteData(0x01);
} else {
lcdGoTo(ball->x_pos, y_page);
lcdWriteData(data);
lcdWriteData(data);
}
ball->x_pos_old = ball->x_pos;
ball->y_pos_old = ball->y_pos-1;

ball->x_pos += 2;

while (ENTER == 0);
}

void init_paddles(paddle* l_pad, paddle* r_pad)
{
left_paddle = l_pad;
right_paddle = r_pad;
/*
 * Set the initial cordinates for the paddles
 */
left_paddle->x = 7;
```

```
left_paddle->pagepos = 3;
left_paddle->upper = 0xc0;
left_paddle->middle = 0xff;
left_paddle->lower  = 0x03;
left_paddle->points  = 0;


right_paddle->x = 120;
right_paddle->pagepos = 3;
right_paddle->upper = 0xc0;
right_paddle->middle = 0xff;
right_paddle->lower  = 0x03;
right_paddle->points  = 0;
}

void move_paddle_up(paddle* pad) {

/*
 * Control so paddle isn't at top
 */
if (!(pad->pagepos == 1 && pad->upper == 0xfe)) {

if (pad->upper == 0xff) {
pad->pagepos -= 1;
pad->upper = 0x80;
pad->middle = 0xff;
pad->lower = 0x07;

} else if (pad->lower == 0x00) {
pad->middle >>= 1;
pad->upper >>=1;
pad->upper |= 0x80;

} else if (pad->lower != 0x00) {
pad->upper >>= 1;
pad->upper |= 0x80;
pad->middle = 0xff;
pad->lower >>= 1;

} else {
```


```
write_digit(1, 20, 0);
}
draw_paddle(pad);
}
}

void move_paddle_down(paddle* pad) {

  /*
 * Control so paddle isn't at bottom
 */
if (!(pad->pagepos == 6 && pad->upper == 0xf8)) {

if (pad->upper == 0x80) {

// Ugly hack in lack of time...
pad->upper = 0x00;
draw_paddle(pad);

pad->pagepos += 1;
pad->upper = 0xff;
pad->middle = 0x0f;
pad->lower = 0x00;

} else if (pad->upper < 0xf8) {
pad->upper <<= 1;
pad->middle = 0xff;
pad->lower <<= 1;
pad->lower |= 0x01;

} else if (pad->lower == 0x00) {
pad->upper <<= 1;
pad->middle <<= 1;
pad->middle |= 0x01;

} else {
write_digit(2, 20,0);
}
```


```
draw_paddle(pad);
}
}

void draw_paddle(paddle* pad)
{
  lcdGoTo(pad->x, pad->pagepos);
// Check if we are in top page so we add field bit
if (pad->pagepos == 1)
lcdWriteData(pad->upper | 0x01);
else
lcdWriteData(pad->upper);

lcdGoTo(pad->x, pad->pagepos+1);
     // Check if field bit is required.
if (pad->pagepos == 6)
lcdWriteData(pad->middle | 0x80);
else
lcdWriteData(pad->middle);

// check so we don't write at page 8.
if (pad->pagepos < 6) {
lcdGoTo(pad->x, pad->pagepos+2);
// Check if field bit is required.
if(pad->pagepos == 5)
lcdWriteData(pad->lower | 0x80);
else
lcdWriteData(pad->lower);
}
}

/*
 * Calculate page and offset of the current y-position.
 * Offset is how many bits from LSB the position is.
 */
void calc_y() {
y_page = ball->y_pos/8;
y_offset = ball->y_pos%8;
}
```


```
void draw_ball(ball_struct* ball)
{
unsigned char data = 0x03, borders = 0x00;
calc_y();
data <<= y_offset;

if (y_page == 1)
borders = 0x01;
else if (y_page == 7)
borders = 0x80;
else
borders = 0x00;

if(x_count == y_count) {
x_count = 1;
y_count = 1;
}

else if((x_count == 4 && y_count == 2) || (x_count == 2 && y_count == 4)) {
x_count = x_count/2;
y_count = y_count/2;
}

if (x_count > 0) {
if (y_offset == 7) {
lcdGoTo(ball->x_pos_old, y_page);
lcdWriteData(0x00 | borders);
lcdGoTo(ball->x_pos_old, y_page+1);

if (y_page == 6)
lcdWriteData(0x00 | 0x80);
else if (y_page == 1)
lcdWriteData(0x00);
else
lcdWriteData(0x00 | borders);

lcdGoTo(ball->x_pos, y_page);
lcdWriteData(data | borders);
lcdGoTo(ball->x_pos, y_page+1);
```


```
if (y_page == 6)
lcdWriteData(0x01 | 0x80);
else
lcdWriteData(0x01 | borders);

} else {
lcdGoTo(ball->x_pos_old, y_page);
lcdWriteData(0x00 | borders);
lcdGoTo(ball->x_pos, y_page);
lcdWriteData(data | borders);
}
ball->x_pos_old = ball->x_pos - ball->x_direction;
ball->y_pos_old = ball->y_pos;
ball->x_pos += ball->x_direction;

x_count -= 1;
if (y_count == 0)
x_count = ball->delta_x;


} else if (y_count > 0) {
if (y_offset == 7 && ball->y_direction == -1) {
lcdGoTo(ball->x_pos_old, y_page + 1);
// Add borders when on page 6
if (y_page == 6) {
lcdWriteData(0x00 | 0x80);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page + 1);
lcdWriteData(0x00 | 0x80);
} else {
lcdWriteData(0x00);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page + 1);
lcdWriteData(0x00);
}

lcdGoTo(ball->x_pos_old, y_page);
lcdWriteData(0xc0 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page);
lcdWriteData(0xc0 | borders);

} else if (y_offset > 0 && ball->y_direction == -1) {
```


```
lcdGoTo(ball->x_pos_old, y_page);
lcdWriteData(data >> 1 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page);
lcdWriteData(data >> 1 | borders);

} else if (y_offset < 6 && ball->y_direction == 1) {
lcdGoTo(ball->x_pos_old, y_page);
lcdWriteData(data << 1 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page);
lcdWriteData(data << 1 | borders);

} else if (y_offset == 0 && ball->y_direction == -1) {
lcdGoTo(ball->x_pos_old, y_page - 1);
if (y_page == 2) {
lcdWriteData(0x80 | 0x01);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page - 1);
lcdWriteData(0x80 | 0x01);

} else {

lcdWriteData(0x80 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page - 1);
lcdWriteData(0x80 | borders);
}

lcdGoTo(ball->x_pos_old, y_page);
lcdWriteData(0x01 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page);
lcdWriteData(0x01 | borders);


} else if (y_offset == 6 && ball->y_direction == 1) {

lcdGoTo(ball->x_pos_old, y_page);
lcdWriteData(data << 1 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page);
lcdWriteData(data << 1 | borders);

lcdGoTo(ball->x_pos_old, y_page + 1);
if (y_page == 6) {
```


```
lcdWriteData(0x01 | 0x80);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page + 1);
lcdWriteData(0x01 | 0x80);

} else {
lcdWriteData(0x01 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page + 1);
lcdWriteData(0x01 | borders);
}


} else if (y_offset == 7 && ball->y_direction == 1) {
lcdGoTo(ball->x_pos_old, y_page);
lcdWriteData(0x00 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page);
lcdWriteData(0x00 | borders);

lcdGoTo(ball->x_pos_old, y_page + 1);

if (y_page == 6) {
lcdWriteData(0x03 | 0x80);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page + 1);
lcdWriteData(0x03 | 0x80);

} else {
lcdWriteData(0x03 | borders);
lcdGoTo(ball->x_pos_old + ball->x_direction, y_page + 1);
lcdWriteData(0x03 | borders);
}
}

else
write_digit(1,4,0);

ball->y_pos += ball->y_direction;

y_count -= 1;
if(y_count == 0) {
x_count = ball->delta_x;
y_count = ball->delta_y;
```


```
}
}
}
```

8.2.3 lcd.c

```
#include "lcd.h"
#include "font.h"
#include <avr/io.h>
#include <stdlib.h>
#include <avr/delay.h>


unsigned char lcd_x, lcd_y;

/*
 * Enable E signal to display for delay() time.
 */
void e_strobe(void)
{
SET_EN;
delay();
CLR_EN;
}

void lcdInit(void)
{
LCD_DATA_DDR = 0xFF;
LCD_CTRL_DDR = 0xFF;
CLR_RST;
delay();
SET_RST;
}

/*
 * Wait for 750 ns
 */
void delay(void)
{
```


```
 _delay_loop_1(2);
}

/*
 * Make sure the display is ready for new instructions
 */
void lcdWait(void)
{
LCD_DATA_DDR = 0x00;
CLR_RS;
SET_RW;
do {
delay();
e_strobe();
} while((LCD_DATA_PIN & DISPLAY_STATUS_BUSY));
}


void lcdWriteCmd(unsigned char x)
{
lcdWait();
CLR_RS;
CLR_RW;
LCD_DATA_DDR = 0xFF;
LCD_DATA_PORT = x;
e_strobe();
}


void lcdWriteData(unsigned char data)
{
if(lcd_x < 64)
{LCD_CS0}
else
{LCD_CS1}

lcdWait();
SET_RS;
CLR_RW;
LCD_DATA_DDR = 0xFF;
```


```
LCD_DATA_PORT = data;
e_strobe();
lcd_x++;

if(lcd_x > 127)
lcd_x = 0;

LCD_NOCS;

SET_RW;

}

void write_digit(unsigned char digit, unsigned char x, unsigned char y)
{

lcdGoTo(x,y);

switch(digit) {

case 0:
lcdWriteData(digit_0a);
lcdWriteData(digit_0b);
lcdWriteData(digit_0c);
break;

case 1:
lcdWriteData(digit_1a);
lcdWriteData(digit_1b);
lcdWriteData(digit_1c);
break;
case 2:
lcdWriteData(digit_2a);
lcdWriteData(digit_2b);
lcdWriteData(digit_2c);
break;
case 3:
lcdWriteData(digit_3a);
lcdWriteData(digit_3b);
lcdWriteData(digit_3c);
```

```
break;
case 4:
lcdWriteData(digit_4a);
lcdWriteData(digit_4b);
lcdWriteData(digit_4c);
break;
case 5:
lcdWriteData(digit_5a);
lcdWriteData(digit_5b);
lcdWriteData(digit_5c);
break;
case 6:
lcdWriteData(digit_6a);
lcdWriteData(digit_6b);
lcdWriteData(digit_6c);
break;
case 7:
lcdWriteData(digit_7a);
lcdWriteData(digit_7b);
lcdWriteData(digit_7c);
break;
case 8:
lcdWriteData(digit_8a);
lcdWriteData(digit_8b);
lcdWriteData(digit_8c);
break;
case 9:
lcdWriteData(digit_9a);
lcdWriteData(digit_9b);
lcdWriteData(digit_9c);
break;

}
}

void write_letter(char letter, unsigned char x, unsigned char y)
{
lcdGoTo(x,y);

switch(letter)
```

```
{

case 'w':
lcdWriteData(letter_w_a);
lcdWriteData(letter_w_b);
lcdWriteData(letter_w_c);
lcdWriteData(letter_w_b);
lcdWriteData(letter_w_a);
break;

case 'i':
lcdWriteData(letter_i_a);
break;

case 'n':
lcdWriteData(letter_n_a);
lcdWriteData(letter_n_b);
lcdWriteData(letter_n_c);
lcdWriteData(letter_n_d);
lcdWriteData(letter_n_e);
break;

case 'e':
lcdWriteData(letter_e_a);
lcdWriteData(letter_e_b);
lcdWriteData(letter_e_c);
lcdWriteData(letter_e_d);
break;

case 'r':
lcdWriteData(letter_r_a);
lcdWriteData(letter_r_b);
lcdWriteData(letter_r_c);
break;

case 'p':
lcdWriteData(letter_p_a);
lcdWriteData(letter_p_b);
lcdWriteData(letter_p_c);
break;
```


```
}
}

void lcdGoTo(unsigned char x, unsigned char y)
{
lcd_x = x;
lcd_y = y;
if(lcd_x > 63)
{
LCD_CS1;
lcdWriteCmd(DISPLAY_SET_X | lcd_y);
lcdWriteCmd(DISPLAY_SET_Y | (lcd_x - 64));
}
else
{
LCD_CS0;
lcdWriteCmd(DISPLAY_SET_X | lcd_y);
lcdWriteCmd(DISPLAY_SET_Y | lcd_x);
LCD_CS1;
lcdWriteCmd(DISPLAY_SET_X | lcd_y);
lcdWriteCmd(DISPLAY_SET_Y | 0 );
}
LCD_CS0;
lcdWriteCmd(DISPLAY_START_LINE | 0);
LCD_CS1;
lcdWriteCmd(DISPLAY_START_LINE | 0);
LCD_NOCS;
}


void lcdOn(void)
{
LCD_CS0;
lcdWriteCmd(DISPLAY_ON_CMD | ON);
LCD_CS1;
lcdWriteCmd(DISPLAY_ON_CMD | ON);
LCD_NOCS;
}

void lcdCls(void)
```

```
{
unsigned char x, y;
for (y = 0; y < 8; y++)
{
lcdGoTo(0,y);
for (x = 0; x < 128; x++)
lcdWriteData(0x00);
}
lcdGoTo(0,0);
}
```

8.2.4 pong.c

```
#include "datatypes.h"
#include <stdio.h>
#include <avr/signal.h>
#include <avr/interrupt.h>

unsigned char ball_timer;

/*
 * Interrupt method that is executed every 10 ms.
 */
ISR(TIMER1_COMPA_vect)
{
// Reset the counter.
TCNT1 = 0x0000;

// Count down our timers.
if (p1_timer > 0)
p1_timer -= 1;
if (p2_timer > 0)
p2_timer -= 1;
if (ball_timer > 0)
ball_timer -=1;

}

int main(void)
```


```
{
int  i;
paddle*  left_paddle, *right_paddle;

/*
 * Init
 */
init_timer();

// enable timer interrupts
TIMSK = 0x10;

p1_timer = 1;
p2_timer = 1;
/*
 * Init Display
 */
lcdInit();
lcdOn();
lcdCls();

/*
 * Init Paddles
 */
left_paddle  = malloc(sizeof(paddle));
right_paddle  = malloc(sizeof(paddle));
init_paddles(left_paddle, right_paddle);

/*
 * Initialize the game field
 */
init_game_field();
init_ball(1);
ball_timer = 0;

/*
 * Initialize Buttons
 */
init_buttons(left_paddle, right_paddle);
```


```
/*
 * Run the game
 */
while(1) {
check_buttons();
ball_timer = move_ball(ball_timer);
  }
}
```

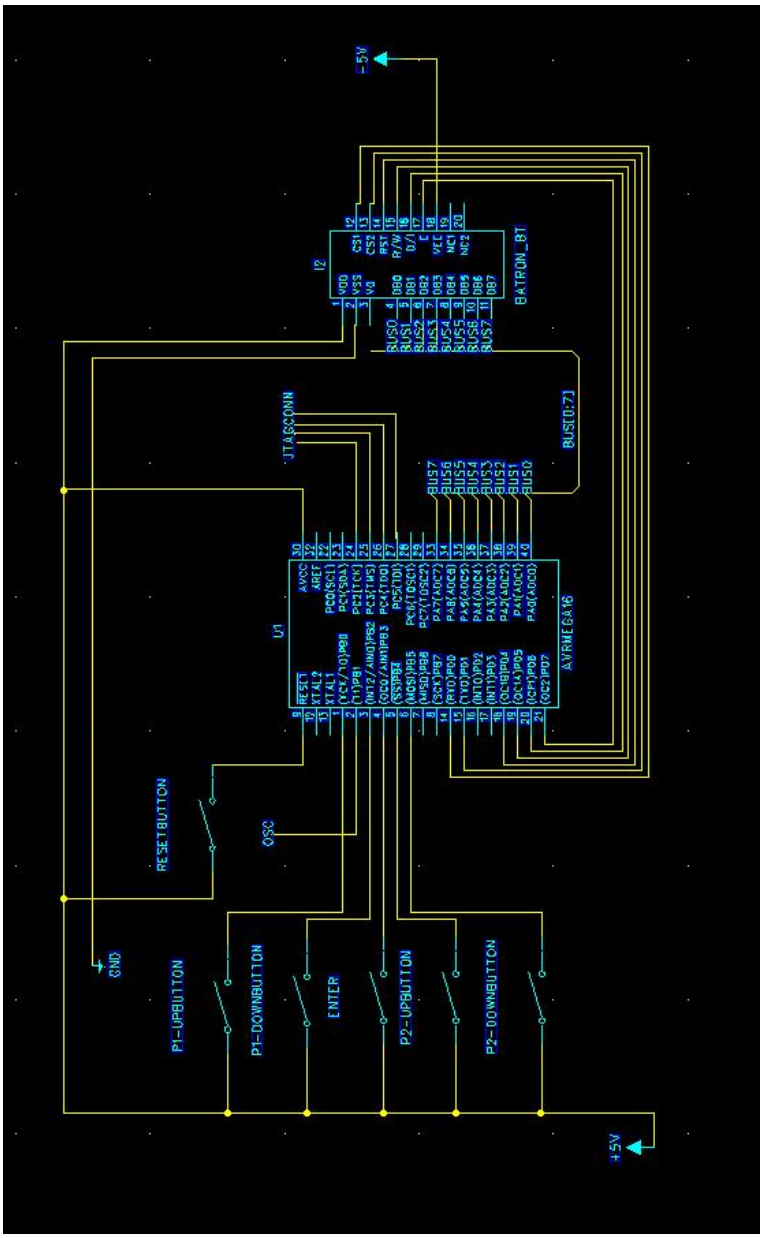
8.2.5 timer.c

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

void init_timer( void )
{
unsigned char sreg;
// Activate external clock on PB1
TCCR1B = 0x07;

// Set timer interrupt at 10 000 counts
OCR1A = 0x03e8; //2710

// Enable global interrupts
SREG = 0x82;
}
```



Figur 3: Kopplingsschema