PROJECT – TETRIS

Course: EITF40 Digital and Analogue Projects

Author: Wang Yue and Johannes Seidel

Teacher: Bertil Lindvall

1. ABSTRACT

This paper describes the developing process of a Tetris Game starting with designing the game, choosing the right components, assembling the hardware and writing the code. The work is based on the microcontroller Atmega16, GDM12864H Display, buttons and JTAG interface. The main task of this project was to generate a working communication between those four components and of course the program itself. The programming process contained the design of the menu and gaming page, creating the different Tetris block types, moving and rotating the block types, controlling and manipulating the speed of the falling blocks, clearing the row and at last increasing the score. How these tasks were solved is described in the following report.

2. Contents

1. Abstract 1
2. Contents 2
3. Introduction 3
4. Requirments 4
5. Hardware 4
   5.1. Microcontroller 4
   5.2. Dislplay 5
   5.3. Layout of the Prototype 5
6. Software 6
   6.1. Main program 7
   6.2. Display 8
   6.3. Pages 9
   6.4. Tetris 10
   6.5. Interrupts 12
7. Table of Figures 13
8. Appendix 14

3. INTRODUCTION

Tetris is puzzle video game, which was designed and invented by Alexey Pajitnov. It has been released on June 6, 1984 and is nowadays available for nearly every video game console and many other devices like graphing calculators, mobile phones, portable media players and PDAs.[1]

However the aim of the game is to score as many points as possible by completing a whole row with square parts of the seven different tetrominoes (Figure 1). "Polyominos are formed by joining unit squares along their edges. A free polyomino is a polyomino considered up to congruence. That is, two free polyominos are the same if there is a combination of translations, rotations, and reflections that turns one into the other. A free tetromino is a free polyomino made from four squares".[2] In the following report the tetromino is always called a block.

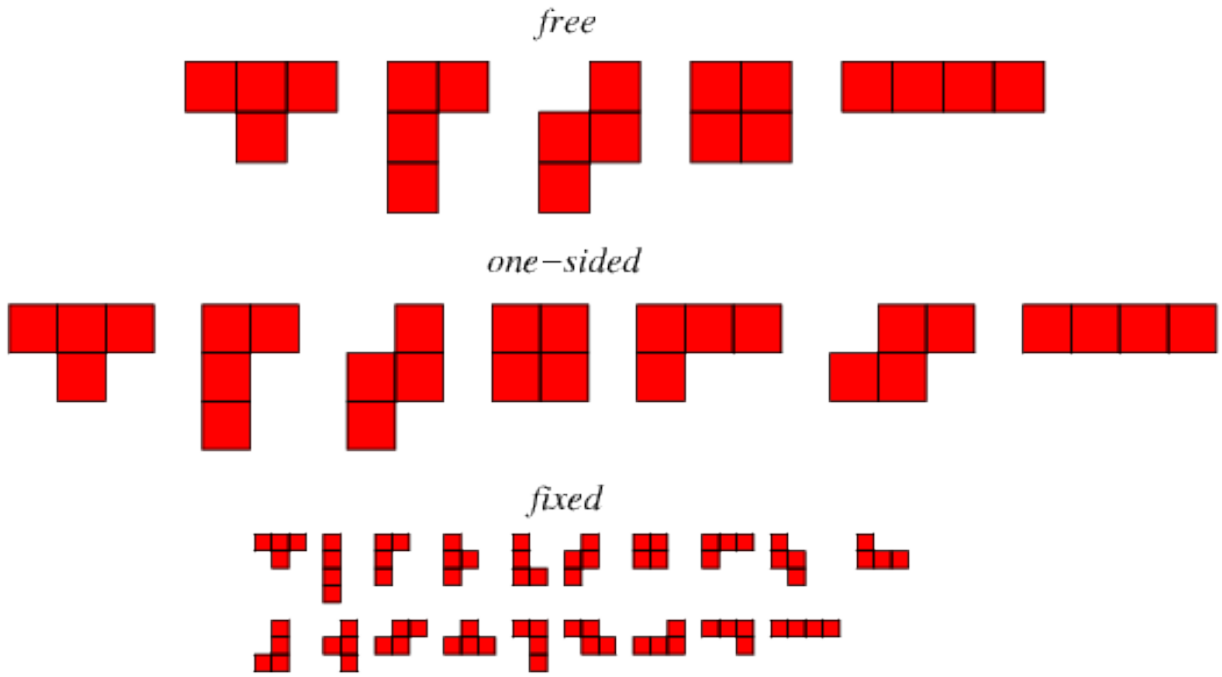


FIGURE 1 TETROMINOES

The blocks are falling down in the game area and can be moved and rotated so that the game player can place them where he wants them to be. If a row is completed it will be deleted and all the upper blocks will fall down. The Game is ended if it isn't possible to place a block without colliding with another block.

[1] http://en.wikipedia.org/wiki/Tetris
[2] http://en.wikipedia.org/wiki/Tetromino

4. REQUIRMENTS

Before starting with the work some ideas and thoughts had to be developed. The following points show the main ideas about the hardware and layout in a short way:

- four buttons for control
- one display to show the game
- one microcontroller as the calculating unit + the JTAG interface
- one prototype circuit board

In the second step the most important requirements of the software were summarized:

- Create a start page with “Play”, “Highscore” and “Quit”
- Create a game page that showing level, score, next block and the gaming area
- Program a library for the characters
- Read and write the display
- Create the different types of blocks
- Make the blocks falling down in different speeds
- Rotate and move the blocks without collision with other blocks
- Delete a completed row and move down all of the upper blocks

5. HARDWARE

Before starting with programming, the hardware had to be chosen. In this case some thoughts about the microcontroller and display were developed. The following part describes the chosen hardware in more detail.

5.1. MICROCONTROLLER

The requirements on a microcontroller to program a Tetris game are not really high, so nearly every microcontroller could be used. That’s why an inexpensive and simple microcontroller is totally sufficient. For that reason an Atmega16, which was also available at the university, was selected.

The Atmega16 is an 8-bit controller, which has 16 kB In-System programmable flash. Its EEPROM is 512 Byte big and as an interface a JTAG can be used. Tow 8-bit timers with separate prescalers and compare modes are included, that is useful especially for controlling the speed of the falling blocks. The controller has 32 programmable I/O Lines which can be used to connect peripheral components, for example display and buttons. [3]

[3] http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf page 1

5.2. DISLPLAY

The main requirement of the display was a screen that is big enough to show the game page described in chapter 4. Also it was important that it was available in a short time because of the limited time for the project.

After comparing a few displays the GDM12864HLCM was selected. The resolution of this display is about 64x128. It is divided in two equal 64x64 arrays (Figure 2) which are controlled each by one chip. In x-direction the display is divided into 8 pages that consist out of one byte (8 bit). The communication between a microcontroller and the display is parallel. Two different types of commands can be used. On the one hand instruction command is needed to tell the display whether it should read or write and to send x- and y-address. With this command the display also can be turned on and off. On the other hand there are data commands where the data has to be written/read.

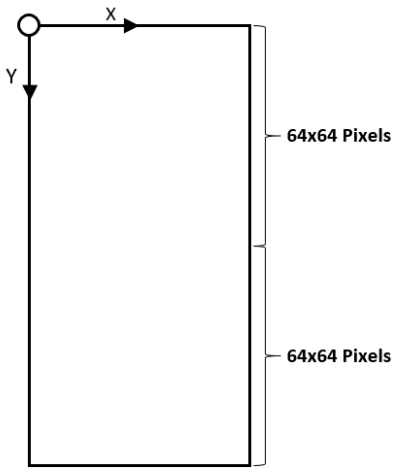


FIGURE 2 DISPLAY

5.3. LAYOUT OF THE PROTOTYPE

Before connecting all the pins of the display and buttons to the microcontroller, a schematic (Figure 3 Schematic) was made and the schematic is shortly described in this part.

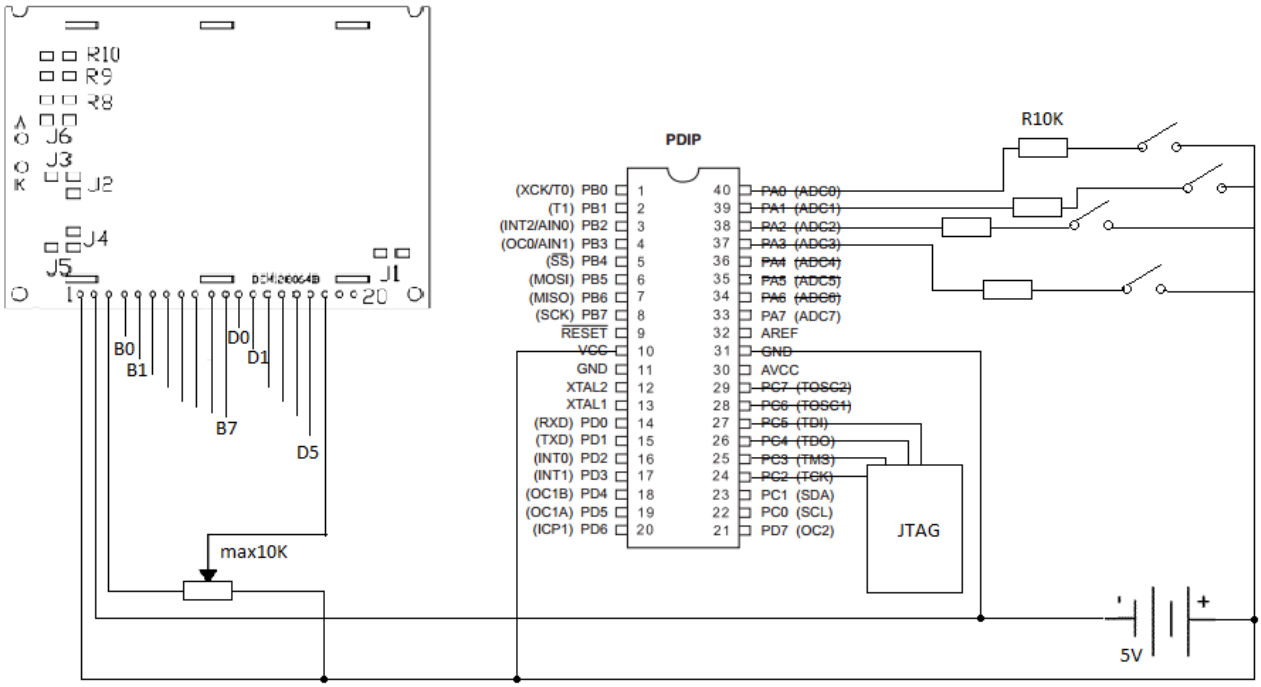


FIGURE 3 SCHEMATIC

As you can see in the schematic, the first four pins of port A were used as input for the four buttons. If a button is pressed a voltage of 5 V between the corresponding pin and ground will be set. Port B and D are connected to the display while port B handles the data and port D handles the instructions, which were explained in chapter 5.2.

	Pin No.	Symbol	Level	Description
	1	Vdd	5.0V	Supply voltage for logic and LCD (+)
	2	Vss	0V	Ground
	3	V0	-	Operating voltage for LCD (variable)
B0-B7:	4~11	DB0~DB7	H/L	Data bit 0~7
D0	12	CS2	L	Chip select signal for IC2
D1	13	CS1	L	Chip select signal for IC1
D2	14	/RES	L	Reset signal
D3	15	R/W	H/L	H: read (MUP<－ module),L: write (MPU－ >module)
D4	16	D/I	H/L	H: data, L: instruction code
D5	17	E	H, H L	Chip enable signal
	18	VEE	-	Operating voltage for LCD (variable)
	19	A	4.2V	Backlight power supply
	20	K	0V	Backlight power supply

FIGURE 4 PIN DESCRIPTION OF THE DISPLAY[4]

Between pin 3, 18 and ground a potentiometer is inserted. When changing the resistance, the contrast or rather the angle of the liquid crystals is changed. Port C enables the communication with a pc while a JTAG is used as an interface.

6. SOFTWARE

For programming the microcontroller the language c was selected. As developing environment Atmel Studio 6 was used. The following chapters explain the software and describe in depth how the code was programmed.

The program is divided into one main program and three subprograms, as shown in Figure 5:

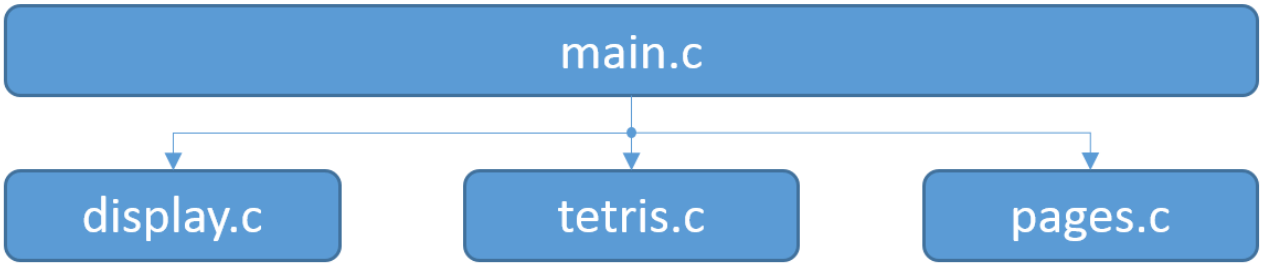


FIGURE 5 PROGRAM OVERVIEW

[4] http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Display/GDM12864H.pdf page 3

6.1.MAIN PROGRAM

The main program is written as short as possible, it primarily contains the initialization and the continuously performs the while loop of the Tetris game. For the initialization, port B and D are set as output by writing 0b11111111 to them and port A is set as input by writing 0b00000000 to it (see chapter 5.3). Also the display is turned on and cleared in the initialization. After that, the interrupts are enabled (see chapter 6.5) and the Start Page is shown using the function “Show_Start_Page” from the subprogram “pages.c”. Then the function “Menu” is called from the subprogram “tetris.c”, so that play, highscore or quit can be selected using the buttons down and right.

In two nested while loops the start column and start row are defined and the functions “Create_Blocks”, “show_next_block”, “Move_Rotate_Block”, “clear_row” and “check_game_over” are continuously executed and performed. If the global variable “game_over” is set to 1 by the function “check_game_over”, the characters “GAME OVER” will be shown on the display.

6.2. DISPLAY

The subprogram display.c was created to summarize all functions that relate to the communication between the display and microcontroller. The following table shows all its including functions and a short description of them in pseudo code.

LCD_E_Toggle	Sets the enable bit of the instruction command from high to low (Pin D5) and waits for the delay.
LCD_Delay	Counts from 0 to 9
LCD_Clear_Display	Clears the display by writing zero to every pixel

LCD_Display_On	Turns the display on by writing `0b00111111 to port B;`
	`Write D/I and R/W (Pin D3, D4) to 0 and`
	`Selecting the chip by writing 1 to D0 – D1`
	`+ LCD_E_Toggle`

LCD_Display_Off	Turns the display off by writing `0b00111110 to port B;`
	`Write D/I and R/W (Pin D3, D4) to 0 and`
	`Selecting the chip by writing 1 to D0 – D1`
	`+ LCD_E_Toggle`

LCD_Set_X_Address	Sets x-address by writing 0b10111000 + page to port B (page is an unsigned 8 bit integer from 0 to 7), Writes D/I and R/W (Pin D3, D4) to 0 + LCD_E_Toggle
LCD_Set_Y_Address	Sets y-address by writing `0b01000000` + add to port B (`add` is an unsigned 8 bit integer from 0 to 63), Writes D/I and R/W (Pin D3, D4) to 0 + LCD_E_Toggle

LCD_Read_Byte	Reads a byte of the display, needs x and y value (address) by writing 0b11111111 to port B;
	selecting chip: if y smaller than 64 write 1 to D1 + LCD_Delay
	else write 1 to D0 + LCD_Delay (chip 2), y=y-64;
	`LCD_Set_Y_Address(63-y);`
	`LCD_Set_X_Address(x);`
	Write 0b00000000 to port B,

LCD_Write_Byte	Write a byte by telling the address (x, y) and the data to write; selecting chip: if y smaller than 64 write 1 to D1 + LCD_Delay else write 1 to D0 + LCD_Delay (chip 2), y=y-64;

TABLE 1 FUNCTION DESCRIPTION OF DISPLAY .C

6.3.PAGES

Pages.c was created to collect all necessary functions for displaying the start and game page. This also includes functions that produce frames, patterns, characters and blocks using the library called font.h.

Show_Start_Page	Shows the start page including “PLAY”, “HIGH SCORE and “QUIT”; To display the word “Tetris” an array from font.h is used. To display other characters, the function Put_Character is used (appendix I).
Show_Game_Page	Displays the game page including “Level”, “Score” and a box for showing the next block, pattern and the gaming area (appendix II).
Show_Frame	Shows one of the three frames (appendix I) surrounding “PLAY”, “HIGH SCORE” or “QUIT”, in order to specify which function is chosen. The templates of the three frames are saved as data in font.h. Writes ones to pixels for the frames.
Delete_Frame	Deletes one of the three frames (appendix I) outside “PLAY”, “HIGH SCORE” or “QUIT”. The templates of the three frames are saved as data in font.h. Writes zeros to pixels for the frames.
Put_Character	Displays different characters, one at a time. The data needed for displaying the characters are saved in font.h. The characters have a size of 8x7 (pixel).
Reverse_Byte	Reverses one selected byte, this is necessary because of the direction the display works.
Create_Blocks	Shows all 7 types of different blocks in the game area. The data needed for displaying the blocks are saved in font.h. The position of the created block is recorded into global array “falling_units”, which keeps track of the falling block.
Create_Blocks_Next	Similar to Create_Blocks, but it shows all types of blocks in the box which displays the next block.
Display_Unit	Shows one unit on the display. The whole display is divided in to 16x32 units. Each unit is 4x4 pixels. Each block consists of 4 units, so the function is useful when creating blocks (refer to Appendix iii) One page is divided into two units in x direction. When displaying a unit on one side (left or right) of a page, keeps the data on the other side of the page unchanged.

Display_Score	Shows the score(integer) while
	hundred = score/100;
	ten = (score – hundred*100)/10;
	one = score%10;

	hundred is written into page 3, ten into page 4 and one into page 5.
Display_Level	Does the same as “Display_Score” except that the input number “level” is displayed at the indicated place for level.
Display_Highscore	Does the same like “Display_Score” except that the high score is displayed at the middle of the display.

TABLE 2 FUNCTION DESCRIPTION OF PAGES.C

6.4. TETRIS

Tetris.c contains all the important functions of the game like for example moving and rotating the blocks. The following table describes the functions in more detail.

Menu	This function is needed to control which frame at the start page should be displayed. As a default frame0 surrounding “PLAY” is displayed. If the button right is pressed it will clear the display and show the game page. Every time button down is pressed the previous frame will be set zero and the next frame will be set one. If right button is pressed while frame1 is one, the display is cleared and the high score is shown. If frame2 is one and the right button is pressed the display will be shut down.
Move_Rotate_Block	With this function blocks can be moved or rotated. For example if the button up is pressed and the function collision_rotate returns a zero, the current block will be rotated by 90 degree counter-clockwise. The rotation is realized by rotating the matrix for displaying the block. The matrix can be rotated as shown in the following table:

row	column	new row	new column
0	0	3	0
0	1	2	0
0	2	1	0
0	3	0	0
1	0	3	1
1	1	2	1
1	2	1	1
1	3	0	1
2	0	3	2
2	1	2	2
2	2	1	2
2	3	0	2
3	0	3	3
3	1	2	3
3	2	1	3
3	3	0	3

TABLE 3 MATRIX TRANSFORMATION

	If the right button is pressed and the function collision_right returns a zero the current block will be deleted and all units of the current block will be moved to right neighbor unit. The opposite applies when the left button is pressed.
random	This function was copied and modified from http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&p=511923. It creates a random number between 0 and 7 which is assigned to the characters A – G using the ascii-code.
show_next_block	Shows a random next block in the box above the game area. The random block type is determined by random(), and then the block type is assigned to the globle variable next_block_type.
collision_left	Collision_left checks if it is possible to move block to the left. If there is a displayed unit left to the falling units it will return a one otherwise it will return a zero.
collision_right	Is nearly the same as collision_left, except that it checks the right side of the block
collision_down	Is also analogical to collision_right and collision_left
collision_rotate	Gets the position of the rotated block without showing the block. Checks if the rotated blocked collides with other displayed units. If there is a collision return 1, else return 0.
block_to_gaming_array	This function records the position of a block when it can't fall down anymore. The position is written into gaming_array, which records all stable displayed units in the game area.
clear_row	If full_row returns a 1 the completed row will be deleted, the gaming array will move one unit down and the score will increase by one and be displayed. After score reaches a defined number (Table 4) the level will be set higher. Table 4 Score - Level
full_row	If a complete row is made this function will return a one.
write_highscore	Write the high score to the EEPROM of microcontroller as described in http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf page 19.
read_highscore	Read the high score from the EEPROM of microcontroller as described in http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf page 20.

Score	10	20	30	50	70	100	140	190	250	320
Level	2	3	4	5	6	7	8	9	10	11

check_game_over	Writes one to the global variable game_over when a one is written to the first row of the gaming_array. This will cause that "GAME OVER" is displayed (Figure 6) on the screen.	 Figure 6 Game over

FIGURE 7 FUNCTION DESCRIPTION OF TETRIS.C

6.5.INTERRUPTS

The interrupts are needed to manipulate the speed of the falling down blocks. Therefore the interrupt vector TIMER1_COMPA_vect was used. It generated an interrupt when a number, which was defined before, is reached. The system frequency was set to 2 MHz so a prescaler was needed. That's why CS10 and CS12 were set so the clock source was divided by 1024. [5]

The time to count for one was calculated as it is shown in equation 1. So the prescaler was divided by the frequency. After this the falling down time for each level was defined (Table 5). Then the count needed for different falling time is calculated as shown in equation 2.

$$t_1 = \frac{prescaler}{f} \quad (1)$$

$$\text{count to} = \frac{t_2}{t_1} \quad (2)$$

Time2	count to	score
0,80	1563	10
0,72	1406	20
0,63	1230	30
0,55	1074	50
0,47	918	70
0,38	742	100
0,30	586	140
0,22	430	190
0,13	254	250
0,10	195	320

TABLE 5 LEVEL DEFINITION

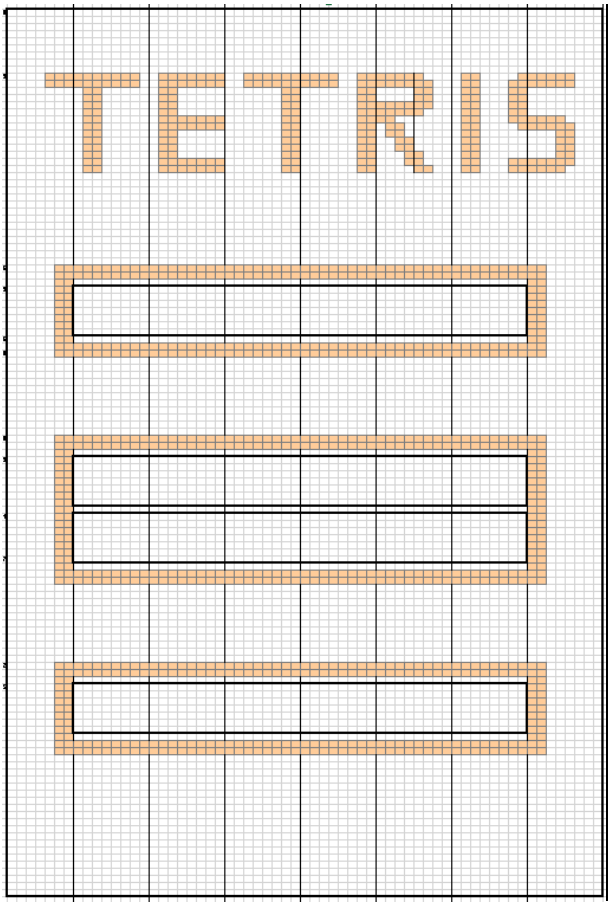
[5] http://www.engblaze.com/microcontroller-tutorial-avr-and-arduino-timer-interrupts/

7. TABLE OF FIGURES

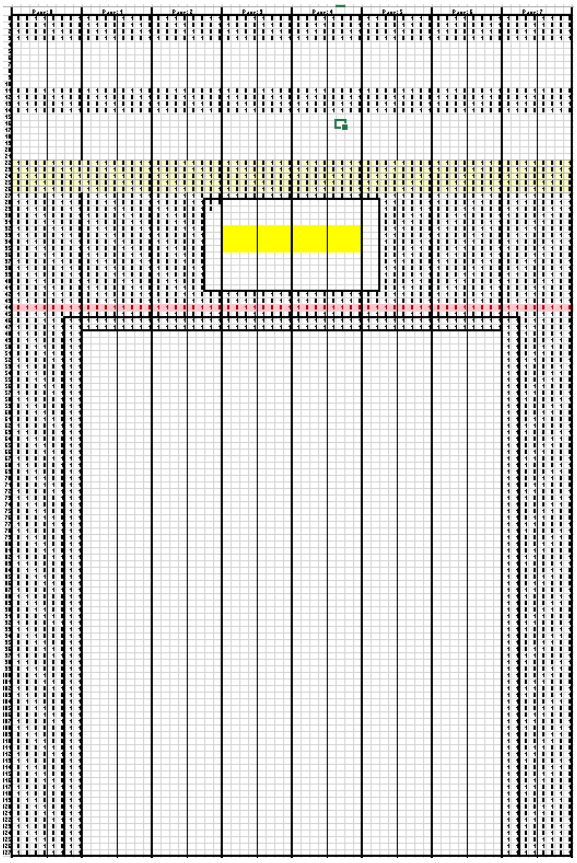
Figure 1 Tetrominoes 3
Figure 2 Display 5
Figure 3 Schematic 5
Figure 4 Pin description of the display 6
Figure 5 Program overview 6
Figure 6 Game over 12
Figure 7 Function description of tetris.c 12

8. APPENDIX

APPENDIX I - START PAGE



APPENDIX II - GAME PAGE



APPENDIX III UNITS FOR DRAWING UP THE BLOCKS