How to generate
Color
video signals
in
software
using SX chips

by

Rickard Gunée

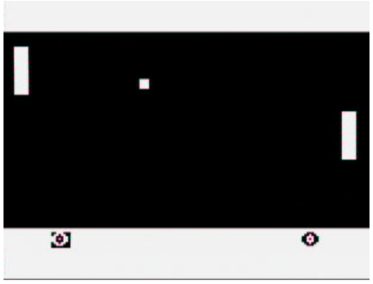
http://www.rickard.gunee.com

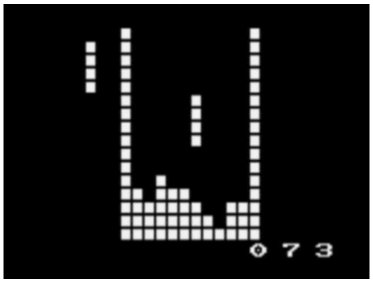
Featuring the games Tetris and Pong

Copyright note

Table of contents

1. Background


PIC16F84-based Pong



PIC16F84-based Tetris

Back in early 1998 I made some experimenting using a PIC16F84 microcontroller (3MIPS of processor power) to generate composite B&W video signals on the fly in software, with two resistors as the only video hardware. I made the two classical games Pong and Tetris with this technique and published them including source on my homepage. Since then it has been built by several hundreds of people. During the Christmas 1998-1999 I got some equipment from Scenix (nowadays known as Ubicom) and made some experiments to generate color video signals using an SX chip, but before I got any results my programmer broke down, at least that was what I believed, and I stopped developing it. In the early summer of 2001 I was told by people at Parallax that it was the early versions of the SX-chips that had a bug in them so my programmer was just fine, so they gave me some new chips and I continued my work. After some new experiments, calculating and many late hours and a bit of luck I got my TV to lock onto the color signal and by the end of summer I got a Tetris game up and running. During the fall I developed the Pong game, which was finished during the Christmas holidays 2001-2002. I didn't release the games as there were some details left to take care of. I didn't want to publish them until they were as perfect as possible due to my bad experience with my PIC-based games that were spread in early bad versions. Now in spring 2003 I decided that I shouldn't do any more improvements of the games as I don't have time to work on them and I got to stop sometime. The biggest remaining issue is that it only works good for NTSC, it is much harder to get a correct PAL signal in software, but that is a problem for someone else to solve. Another issue about the games was this text about generating color video signals that I wanted to finish before I released the games, to not get that many questions about video generation that I don't have time to answer. After reading this document you will hopefully understand how to generate color composite video signals in software. To fully understand this you need mathematical knowledge at university level, some RF-knowledge would also help a lot.

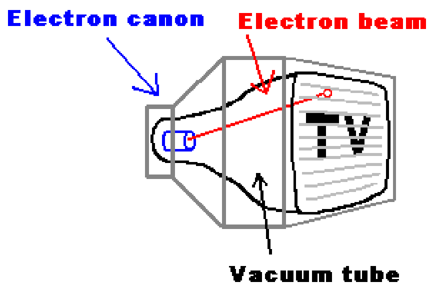
SX-Tetris

SX-Pong

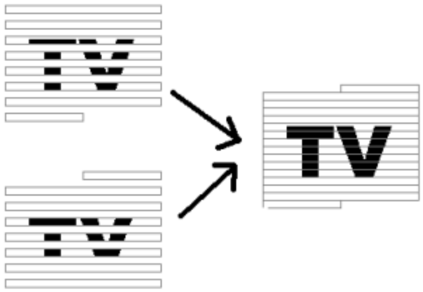
2. The composite video signal.

To understand anything about generating video signals in real-time, one must know how video-signals work in detail, so before we look at any code we'll have to talk about video signals.

2.1 How a standard TV-set works



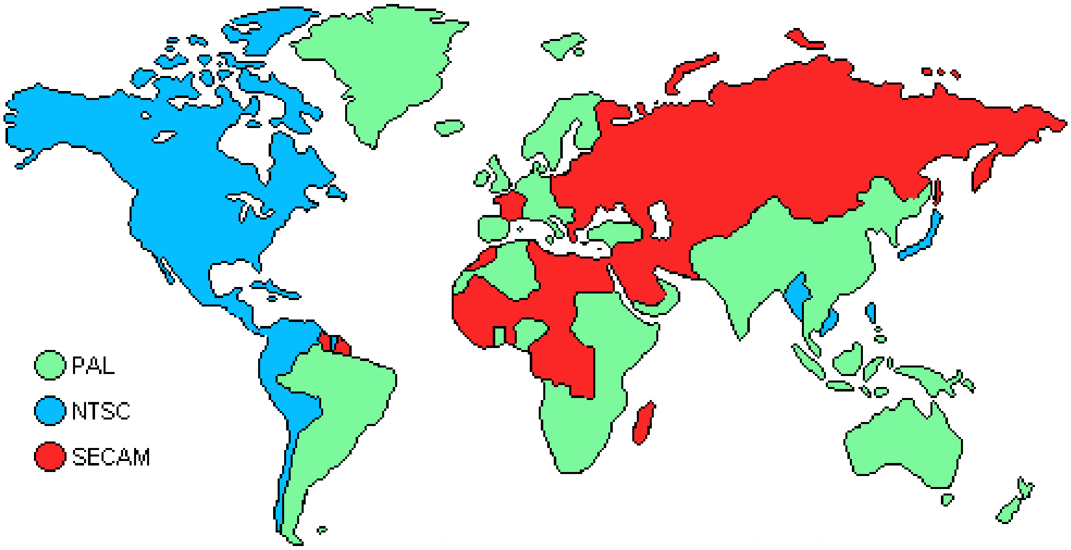
The electron beam drawing the screen



The two part images becomes one whole image.

A standard TV-set is built with a vacuum tube, which has a phosphor screen that an electron canon shoots at. When the electrons from the cannon hits the screen, light is emitted from the phosphor when the canon shoots electrons at it, and it also has a short afterglow making each pixel lit until the electron beam hits it again. The electron beam from the electron-cannon can be bent using magnets so it shoots at different parts of the screen. If this is controlled so it draws horizontal lines all over the screen repeatedly, while the intensity of the beam is modulated, an image can be drawn on the screen. The screen is redrawn 25 times per second (on a PAL system), but to reduce flickering the image is interlaced, showing first all odd lines then all even lines, so the image is partially updated 50 times per second. To get color each dot on the screen is divided into three colors: red, green and blue.

2.2 Different TV standards



A rough map over the different TV standards used on earth.

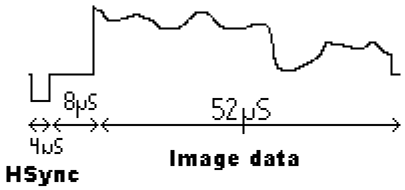
There are three major analog TV-standards: NTSC, SECAM and PAL as seen on the map above. The NTSC (Short for "National Television System Committee", but back in the early days of TV there was problems with getting the same color over the whole picture so a more evil interpretation of the letters is that it stands for "Never The Same Color") is the American TV-standard, it has only 525 scan-lines, but it has a update frequency of 30Hz. SECAM (Short for "Sequential Color And Memory", but as the French usually want to get their own solution to problems, a more evil interpretation is that it stands for "System Essentially Contrary to the American Method") is the French TV-standard, it has improved color stability and higher intensity resolution but with less color resolution, I don't know much about that standard. The European standard is PAL (Phase Alternating Lines, or as a PAL enthusiast would interpret the letters: "Perfect At Last"), it has 625 lines per frame, 25 frames per second. It is based on NTSC, but the color-coding has been improved by using a phase shift on every other line to remove the color errors that occurred with NTSC.

2.3 The information in the video signal

The image seen on the screen has different intensities. As the electron beam sweeps over the screen, the intensity that should be at the position of the beam, is sent as a voltage level in the video signal.. There is no information in this intensity information about where the electron beam is on the screen. To solve this, a synchronization pulse is sent in the beginning of each line to tell the TV that the current line is finished and move down the electron beam to the next line. (Like the <Enter> key on the keyboard, when writing a text with a computer) The TV must also know when a new image is coming, this is done by making a special synchronization pattern. (Like the "new document" function when writing a text with a computer) An image that is updated 25 times per second would be quite flickering, so therefore all even lines are drawn first and then all odd, this method shows 50 half images per second, making the picture have less flickering. The information whether the image contains even or odd lines are sent in the vertical synchronization pattern, as different patterns for odd and even images. The video signal has a voltage range 0 to 1V, where 0.3V represents black, and 1.0V is white (gray intensities have voltages between these values). Levels close to zero represent synchronization pulses.

2.4 The scan-line

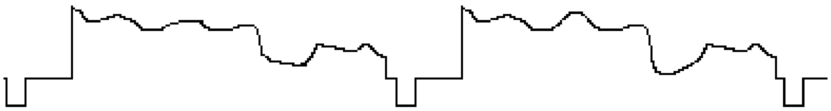
The image is divided into scan-lines, it is the most important part of the image since it contains the image data. The scan-lines are all 64us long. First a 4us long sync pulse is sent, by setting the signal level to 0V, to tell the TV that a new line is coming. The old TV's was kind of slow, so they needed 8us after the sync-pulse to get the electron beam in position. During this time the signal is kept at black level. The 8us delay is followed by the image data for 52us, drawn on the screen from the left to the right with the intensities obtained from the video signal. Black is represented by 0.3V and as the voltage increases the intensity increases, with the maximum intensity at 1.0v (white). See the image right to see the scan-line. The color information is added as two amplitude modulated sinus waves, we'll get back to that later.



"Oscilloscope"-picture of one scan-line

2.5 Putting the scan-lines together to an image

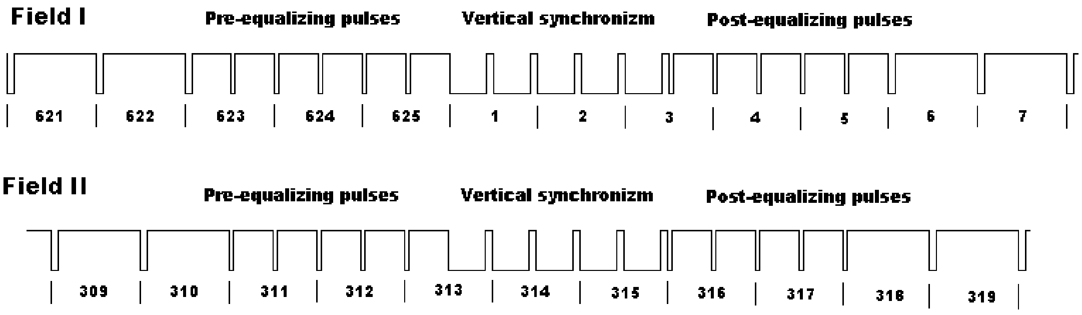
An image is built from 625scanlines, but a TV doesn't show 625 lines. Some of the lines are used for synchronization pulses, and some lines are invisible (I don't know exactly how many) because old TVs needed some time to move the electron beam from the bottom of the screen. (Those invisible lines are nowadays used for other purposes, Text-TV for example).



"Oscilloscope"-picture of several scan-lines in a video signal.

2.6 Vertical synchronization pulses.

To tell the TV that a new image is coming, a special pattern of synchronization pulses is sent. Since the picture is built from two half pictures, the pattern is different for the odd and even images. The vertical synchronization pulses looks like this:



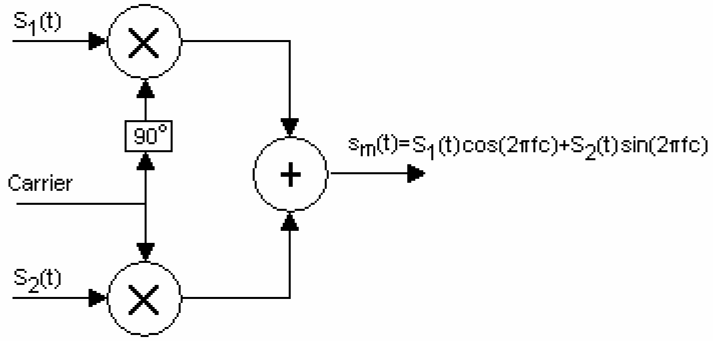
This picture shows the different vertical synchronization pulses for the two half images. The levels are 0v and 0.3v. (Numbers below signals shows scan-line number)

2.7 Color coding.

When color was introduced, it was the same problem as with any change in technology, there is always a demand for backwards compatibility that limited the new technology. For video signals this meant that a color video signal should look very much like a B&W signal so old TVs would still work. The problem was solved by overlaying the color signal with an amplitude modulated carrier on top of the video signal. In average the video signal would still be the same for B&W and it would not be noticed if the carrier had high enough frequency and the modulation also was kept to a low bandwidth.

The intensity of the TV signal is the sum of the Red, Green and Blue parts (weighted with the eyes sensitivity coefficients for those colors) in the video signal, and since that information is already given in the B&W signal then the additional color information only needs to contain two components with color difference. With the intensity sum and the two components G-R and G-B, it is possible to derive the R,B and G values. Humans have higher resolution for light intensity than for color, so using higher bandwisth for intensety than for colr variation is very appropriate. Limiting the color information to two components is especially great as it is possible to transfer two signals using quadrature modulation, making it possible to transfer color using only one carrier overlaid on the B&W video signal!

2.8 Quadrature modulation



The basic principle of quadrature coding

Quadrature modulation is a general method for modulation of a carrier. The idea is to change both amplitude and phase of the carrier to be able to send two signals with one carrier frequency. Each signal has its own carrier, one is sin ($2\pi f_c t$) and one is cos ($2\pi f_c t$), which makes it possible to reach all phases and amplitudes by modulating the voltages of the two signals. This method is not only used for TV color modulation, it is widely used, for example this is how stereo information is sent over radio also. It is a clever way to use the bandwidth to the maximum, with standard amplitude modulation only one channel is used, the other is just wasted. In order for this method to work, there must be a “pilot”, a reference signal that makes synchronizes the oscillator in the receiver with the one on the transmitter.

How the quadrature modulation is used differs slightly between PAL and NTSC. One variation is the white level as PAL where developed after NTSC and has hence more accurate coefficients to the newer more luminant phosphors used inmodern CRTs. The colors are weighted according to the eye’s sensitivity, so the green color is weighted the most, blue the least and red in the middle. Using RGB-color levels detected by the “video camera”, the luminance is calculared according to:

$$\text{PAL:}\quad Y = 0.222R + 0.707G + 0.071B$$
$$\text{NTSC:}\ Y = 0.299R + 0.587G + 0.114B$$

The Y,U,V component transformation can be described as a matrix, for PAL the matrix looks like the following.

$$\begin{bmatrix} Y \\ U_t \\ V_t \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

In NTSC the U and V components are rotated 33 degrees to minimize bandwidth for Q, the rotated components are called I and Q, calculated according this:

$$I_t = V_t \cos(33°) - U_t \sin(33°)$$
$$Q_t = V_t \cos(33°) + U_t \sin(33°)$$

For NTSC the Y,I, Q components can be described using the following conversion matrix.

$$\begin{bmatrix} Y \\ I_t \\ Q_t \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.311 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

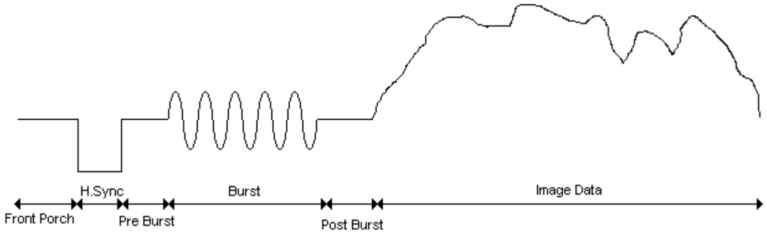
2.9 Putting it all together

The output is created with quadrature modulating as described before by modulating a cosine and a sine with the U and V (I and Q for NTSC) components and sum the result together with the lumination component Y. For PAL there is also a change in sign of the sinus component to compensate for phase error that will take out each other. (That is why it is called Phase Alternating Line). The video signal is calculated according the following.

PAL: $S(t) = Y + U_t \cos(2\pi f_C t) \pm V_t \sin(2\pi f_C t)$

NTSC: $S(t) = Y + I_t \cos(2\pi f_C t + 33°) + Q_t \sin(2\pi f_C t + 33°)$

So the color coding is just as simple as that, but there is one detail left, there must be a pilot signal in order for the quadrature modulating. In most systems using quadrature modulation, the pilot signal is sent constantly as a tone in the signal, for TVs however that would disturb the image too much. If there is an oscillator in the TV that is very stable, it would be enough to send a couple of cycles of the pilot periodically for the oscillator to tune in to, just often enough for the oscillator to keep its phase. In the B&W signal there is a gap of about 7µs between the sync pulse and where the image information starts, so it was an obvious place to put the reference carrier. This is 10-12 cycles of the color carrier (amplitude of 20IRE = 0.15V) and referred to as the “color burst”. The color burst is also shifted +-45 degrees on every scan-line for PAL.



This picture shows the scan-line including color burst.

3. Creating it in software

Generating a B&W signal is not very complicated; it is just hard work as it is a question of counting clock cycles to make the program flow take exactly the same amount of clock cycles all the time. When doing a color signal, this is even more important, if the line is one cycle too long or short (An error of 0.03% in scan line length) the TV can't lock to the color carrier at all, for a B&W video signal the timing is not this critical, most TVs can compensate for quite large errors in a B&W video signal, so you could make the scan line's length several tenths of cycles wrong without noticing as the TV compensates for it, but as our goal is to make a color video signal we are not allowed to do any errors at all. To make the job of timing easier I've created a general delay macro that delays for a given time using a minimal amount of program memory. I've also tried to use a lot of "EQU-constants" to make the code more readable and make the code possible to run for both NTSC and PAL by only changing the constants so the code is the same for both systems.

The first thing the software needs to do is output the vertical sync pulses, to tell the TV that a new frame has started. Then for the following 304 lines (254 for NTSC) it should keep each line 64us long and start each line with a horizontal sync pulse. Later on when doing a color signal a color burst must also follow after the horizontal sync pulse. During the 52us of image time the software needs to vary the voltage of the video signal between 0.3v (black) and 1v (white) as the electron beam sweeps over the screen and try to do draw something as the electron beam sweeps over the screen. This is quite easy with an SX performing 50MIPS, I've done B&W games this way using a PIC16F84 performing 3MIPS, so one could do B&W games with quite high resolution using an SX. However, generating color is much more cool, so let's talk about color generation now.

3.1 The basics for color generation

As you would know after reading the chapter about video signals, the software needs to create modulated sinus and cosines waveforms for color information and sum them together with the intensity waveform. To get a good result the sample rate needs to be much higher than the color carrier frequency, and the software must also be able to do the needed calculations for the waveform which in total would need a very powerful processor if there is no hardware to help. An SX processor performing 50MIPS would not be good enough using this method.

3.2 Mathematical tricks

However, there is fortunately a better way to do it. The color carrier part of the signal is the sum of a sinus and a cosines with the same frequency but different amplitude, this is very fortunate as the cosines could be rewritten as a sinus with it phase shifted 90 degrees compared to a cosines. Ok so what good is that, well, the sum of two sinuses with same frequency and fixed phase difference but with varying amplitude could be rewritten as one sinus with alternating phase and amplitude according to:

$$f(x) = a\sin(x) + b\cos(x) =$$
$$= \sqrt{a^2+b^2}\left(\frac{a}{\sqrt{a^2+b^2}}\sin(x) + \frac{b}{\sqrt{a^2+b^2}}\cos(x)\right)$$

The coefficients preceded cos and sin describes a point on the unit circle and could be replaced with cos and sin with the angle α according to:

$$\frac{a}{\sqrt{a^2+b^2}} = \cos(\alpha)$$
$$\frac{b}{\sqrt{a^2+b^2}} = \sin(\alpha)$$

This equals a rotation by an angle α according to:

$$f(x) = \sqrt{a^2+b^2}\left(\cos(\alpha)\sin(x) + \sin(\alpha)\cos(x)\right) =$$
$$= \sqrt{a^2+b^2}\sin(x+\alpha)$$

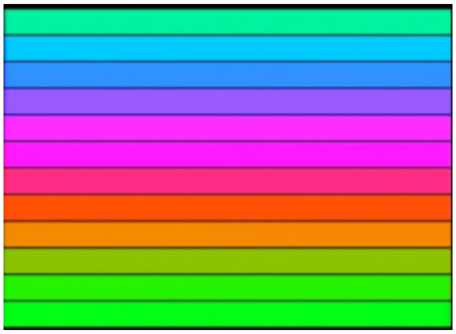
Making it possible to express the sum of an aplitude Modulated sin and cos with one sin that is both aplitude And phase modulated.

3.3 Know your hardware

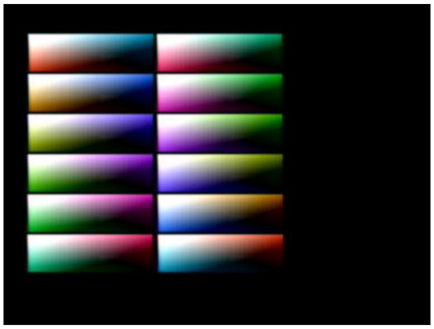
Ok we got rid of one of the components but still have one sinus that needs to be generated requiring a lot of CPU power. At the input of a TV there is a low-pass filter to limit the signal within a video signals allowed bandwidth of about 5MHz, which is very good because that means that a square wave at the color carrier frequency would look like a sinus to the TV as the high frequency components of the square wave have been filtered away. Now we are down to a square wave with changing phase, amplitude and offset, which is possible to generate in software with an SX@50MHz if the number of phases is limited and the clock frequency is a multiple of the color carrier frequency. In my projects I clock the SX with 12 times the carrier frequency for both PAL and NTSC, which gives 53.156550MHz for PAL and 42.954540 for NTSC, the over clocking of a 50MHz SX chip to 53MHz in the PAL case seems not to be a problem at all.

3.4 Our new parameters

The simplified signal with the square wave works like this: The average voltage of the signal control the lumination, the amplitude of the signal controls the whiteness and phase controls the color. When using 12 times the color carrier it is possible us get 12different colors with different variation in intensity and whiteness. The first test I made with color generation was to examine the 12 base colors available, this test I shown in the picture to the left below. The source for this test can be found in Appendix A. (This is the only one of my current programs actually performing phase alternation in PAL, sp the phase errors for PAL are not visible in this example) All possible variations for the 12 base phases can be seen here to the right below where all possible values for first and second amplitude are shown for all 12 phases and five bits. (There are25*25*12/2-25*5=3625 combinations) The source for the later is available in Appendix B.



The 12 phases, generates these 12 base colors..



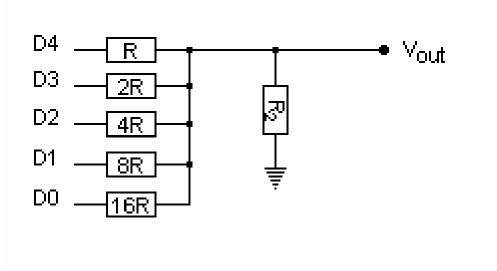
The available colors for 5bit DA and 12 phases.

3.5 Phase Alternating Line

is what PAL stands for, and that is a problem, when generating a PAL signal one should switch the phase of the signal 180degrees on every line (color burst switched 90 degrees), this is not possible with the method I generate color signals. It is possible to produce more simple graphics such as one colored horizontal lines and phase alternate, but when doing more complicated stuff (like text or graphical objects) t becomes a problem as not only is the phase alternated, so is the positions of the graphics as the graphics must be aligned with the color carrier cycles. I chose to solve this by ignoring the phase alternation, with the downside that it makes phase errors visible as they did originally with NTSC where there is no phase alternation. With NTSC this is no longer a problem as the modern TVs have become better and lock to the color carrier much better, which the PAL TVs didn't have to as their color method compensated for this problem, giving me a problem when I "cheat" when generating my video signals. I have no good solution for the problem with PAL to be software generated; it is up to someone else to figure that one out. (All pictures in this document are from the NTSC versions as they are the only pictures that are good enough to digitize with the TV-card in my computer)

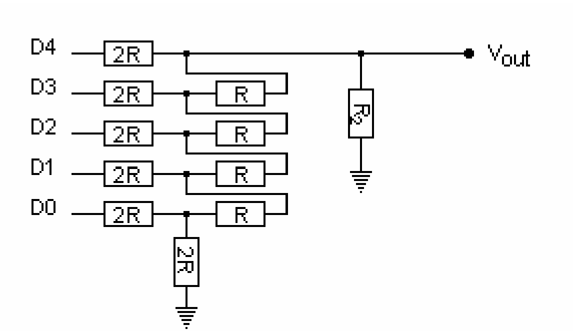
3.6 Video output hardware

To be able to generate the signal we need a DA-converter. To make this simple a resistor based DA is the way to do it. There are two kinds of resistor DAs, 2^N-nets and R2R-ladders. The 2N net is the simplest solution, it looks like this:



2^N DA converter schematic

The downside with the 2N-net is that it is very inaccurate; the R2R-ladder requires twice as many resistors but has much higher accuracy, it looks like this:



R2R DA converter schematic

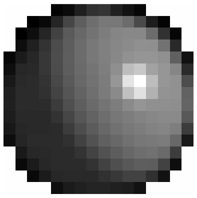
First I chose 6 bits for the DA as that is the largest number of bits that would be useful using 1% accuracy resistors, later I found that five bits is enough, the extra bit is better off in the DA, so the finished system go five bits for both sound and video. The video bits is bit 1 to 5 in my system as I already had done a optimizations in the code for using the lower 6 bits of portb making it the easiest solution, but when designing I new 5 bit system it is of course better to use bits 0 to 4.instead. Output voltage should be in the range 0 to 1.25V, which sets the values of the resistors to 220Ω and 440Ω, but as there are no such resistors, it is better to keep the 1:2 ratio and use 221Ω and 442Ω.

3.7 Limitations with colors

The color bandwidth is very low so it is not possible to change colors fast. In my games I keep the color phase constant within a graphic object and only change lumination level once every color cycle. This gives a maximum resolution of

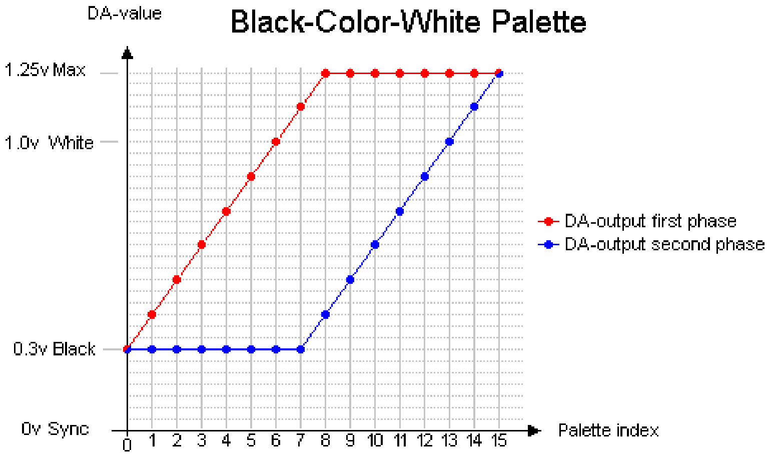
2766/12=230 pixels per scan line for PAL and 2233/12=186 pixels per scan line for NTSC. In reality not all pixels can be used as color (phase) changes cost time and thereby color cycles, and then the graphics also has to be calculated to there are not all of these pixels that actually can be used.

3.8 Use of Palette



Monochrome ball from Pong

To save memory a palette is often used in computer graphics cards. A palette is basically a color lookup table. In most cases the palette contains 2^N colors, usually 16 or 256 colors to be able to get each color into a nibble or a byte. If a picture only uses 16 different colors, then it needs 6 times less memory compared to if each byte would have been stored as three 8bit values with the RGB-components if a 16 color palette is used. In my games a palette is used to need less data for some of the graphics, a 16 color palette is used, however the lookup table doesn't store the RGB values, instead it stores high and low period values for the square wave. In other words, my palette only contains info on brightness and whiteness, the color is set by the phase of the square wave which is not stored in the palette. Only one palette is used for both my games and it stars at black level, moves to color with maximum intensity, and then moves to maximum white. (See diagram below.) This palette makes it possible to generate objects with a 3D-feeling as it is possible to make dark shadows and more illuminated parts within the same object, but the object must be "monochrome". It is possible to generate palettes with a 180 degree phase shift and get the complimentary color, but as the bandwidth is limited it is not possible to mix colors from the two phases in any order, it takes almost one color cycle for the phase change. (If the graphics is carefully planned to get few phase shifts, this could probably be used to do some really cool two colored objects)



The BCW-palette used for monochrome objects in my games

3.9 Outputting monochrome objects using palette

When showing graphics with high resolution (one intensity change per color cycle) it is not possible to calculate the graphics in real-time, so the graphics needs to be pre-calculated and stored in a buffer and then outputted from the buffer. I have created a routine that gets 4-bit graphics from the upper nibble in program ROM, translates it using a palette and store it in a buffer, consuming 31 clocks per pixel. A matching output routine, called memtovideo, which outputs data from the buffer at a speed of one pixel per color cycle (12 clock cycles). During the calculation of the next object it is not possible to show any graphics except for black or different gray colors, so therefore the layout of the graphics is very critical. In my Pong game I use three different graphic buffers, one for each paddle and one for the ball, and the graphics calculation is dynamically changed depending of where the ball is on the screen because the ball position controls where on the screen there are black surfaces that can be used for graphics calculations. In Tetris the graphics for the screws beside the graphics is calculated to the right of the playfield on the line above the one where the graphics is shown, and as both screws are identical only one graphics calculation is needed but it is outputted twice (one time on each side).

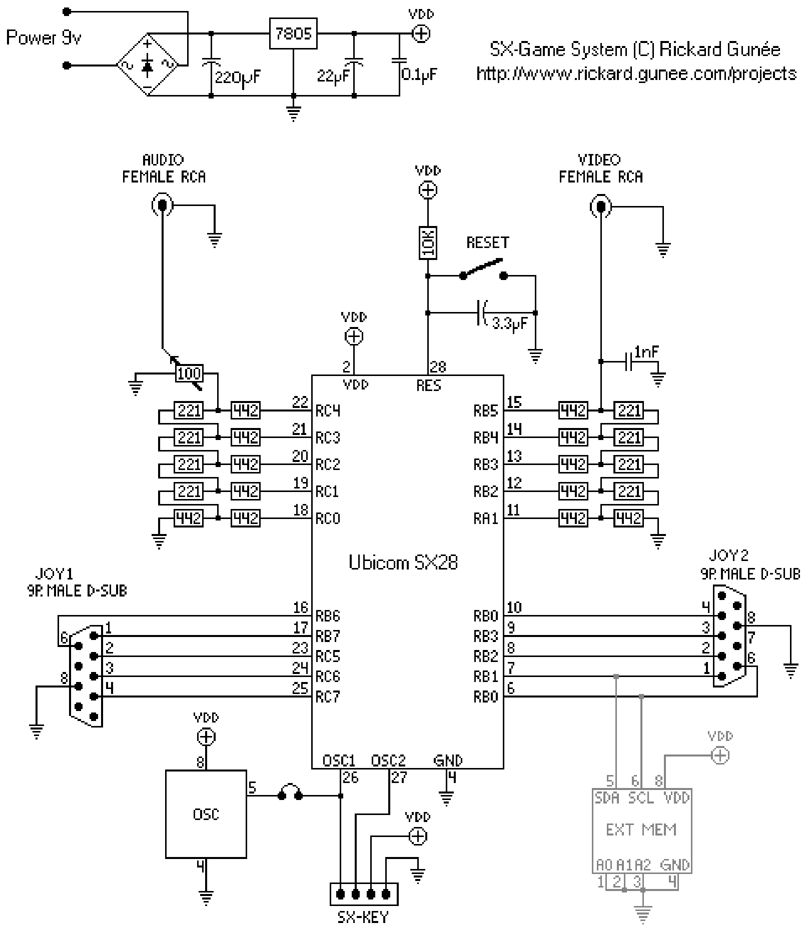
3.10 Colored text lines

The texts that appear in my games are generated on the fly; only two ROM-accesses are needed per character. First the character is read from the string stored in program ROM (low 8 bits), then this is used together with the line number to find the graphics from the font that also is stored in program ROM. Each character is 7 pixels wide, and separated by two pixels, originally the separation was three pixels but after unrolling the loop I got it down to two pixels (At the cost of program memory usage). The separation could probably be decreased to one by more unrolling at the cost of more program memory. A font is quite expensive in memory usage, so to save memory I only store the characters I use. The color generation in the text output is done by having a high and a low level for each pixel, the high level is an input parameter and the low level is always black to optimize the routine.

3.11 Emulators

Developing this kind of software is always much easier, but there are unfortunately no emulators available for color composite video signal generation with SX chips. However, there are some interesting open source stuff that might could be used as a good base for developing an SX color video game emulator.

4. Game system



Schematic over the game system

4.1 Schematic overview

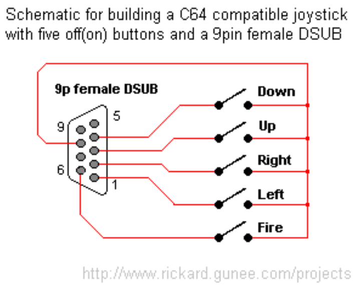
The power supply is standard, a 7805 regulates the voltage to 5v, there is a rectifier at the input to be able to run the system on both AC or DC, the voltage can be 9..15v something. Then there is a bunch of caps on the board to get rid of noice etc.

The video generation is quite simple; it is just a five bit R-2R resistor ladder. It might seem a little bit strange that I connected it to bit 1...5 instead on 0..4, but that is because when I first made the prototype it hade six bits for video and four for audio. I chose six bits first as it is the largest number of bits you should use with an R-2R ladder when using 1% tolerance resistors. Later I understood that it was not needed that many bits for video, that last bit would be better off in the audio generation. At the end of the R-2R ladder I have put one 1pF cap to get a a little bit of filtering if the TV's input has to high bandwidth as my color generation technique generates square wave that needs to be filtered so only the "sinus part" remains. The resistor ladder has an open output as it is supposed to be connected to the TV that has an 75Ohm input that ends the ladder.

The audio part is very similar to the video part, also a five bit DA using an R-2R ladder. The difference is at the end of the ladder, the audio has a 100 Ohm pot to regulate the volume. The 100kl pot also ends the ladder as the audio impedance varies a lot between different audio inputs. (1k...20kOmh)

4.2 Joysticks

The joystick inputs are extremely simple, just five pins on the chip connected directly to the joystick inputs. The joystick pins on the SX-chip have their internal pull-up resistors enabled so there is no need for external resistors. There are two joystick inputs, and as with my PIC-based games I used old C64/Amiga/Atari joysticks. If you don't have one you could build one quite easy using the schematic here to the right using five off(on) switches and a 9pin female dsub.


Schematic for building a C64 compatible joystick with five off(on) buttons and a 9pin female DSUB
9p female DSUB
9
5
6
1
Down
Up
Right
Left
Fire
http://www.rickard.gunee.com/projects

Joystick schematic

4.3 The Oscillator

One of the more tricky parts is the Oscillator. This should run at 12 times the color carrier of the TV-system. The built in generator in the SX-key programmer is not accurate enough for video generation, so an external oscillator is needed. During the development of the games I used an almost 30 years old frequency generator (as new ones cost a fortune) seen at the picture here to the right, which made the development a lot easier. There chip-oscillators available that can be programmed once just like you can program a microcontroller. See the table below for what frequencies to use.

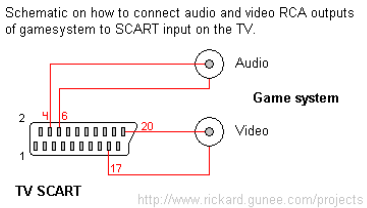
HF Signal generator

TV System	Carrier Frequency	ClockFreq = 12 x Carrier Frequency
PAL	4.4297125 MHz	53.156550 MHz
NTSC	3.579545 MHz	42.954540 MHz
PAL-N*	3.575611 MHz	42.907332 MHz
PAL-M*	3.582056 MHz	42.984672 MHz

**Note: None of the games have been tested and calibrated for PAL-M or PAL-N.*

4.4 TV connection

The system is connected to the TV's SCART input with a cable with RCA inputs and a SCART contact at the end. These cables are available in most TV stores. The best thing to do is buying a finished cable, building one is more expensive and doesn't give a good result, however if you still want to build one you should follow the schematic here to the right.

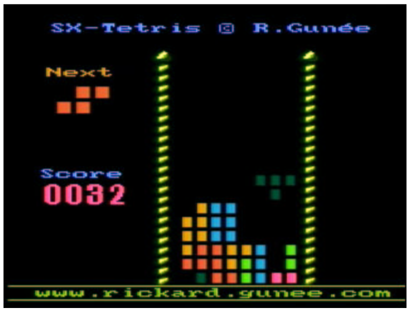

Schematic on how to connect audio and video RCA outputs of gamesystem to SCART input on the TV.
Audio
Game system
2
4
6
20
Video
1
17
TV SCART
http://www.rickard.gunee.com/projects

SCART-cable schematic

4.5 PCB

I've made a PCB design for the game system, available in Appendix E. This is quite simple as the game system is very simple, just a one layer. The PCB is stored in scale 1:1 so if you print it directly from this document you will get the size correct. The component placement is also available in Appendix E. Note that for my games you don't need the expansion memory, I might do games later on that will use it but nothing planned yet. To avoid cracking the programmer I used a 90 degree bent connector for the programmer so it lies flat on the PCB when connected. There is a jumper close to the oscillator and the programmer that selects between programmer and oscillator as both can not be connected at the same time. As mentioned before, the resistors should be 220Ω and 440Ω, but as there are no such resistors, it is better to keep the 1:2 ratio and use 221Ω and 442Ω.

5. Tetris

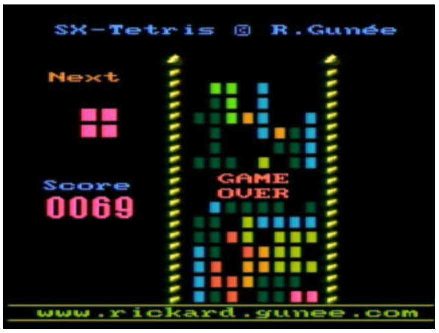


Tetris in action

The first game I made in color using SX-chips was Tetris. Tetris is an old Russian computer game where you should try to fit in block into a play-field, quite simple but really fun. All blocks are built from four bricks (the name Tetris is derived from the ancient greek word for four: "tetra"), there are seven combinations of the four bricks as seen here to the left. This version is using my PIC Game System as platform, generating a video signal in software. The video generating hardware is a 5-bit DA converter built with a few resistors. Usually the video signal is generated in video games is created with a dedicated video chips, reading the image data from a graphics memory. In this project the video signal is calculated in real-time by the microprocessor as the electron beam sweeps over the screen.

5.1 How to play the game.

When the power is turned on the game starts! (was no memory left for a fancy intro screen or similar). The score is shown left of the gamefield, and the next block to come is shown in the upper left corner of the screen. As the blocks fall down, they can be moved sideways by using the joystick (left gameport on hardware), the fall speed can temporary be increased by moving joystick down. The fire-button is used to rotate the blocks. When one horizontal line is full, then it is removed. You get points for full lines and placed blocks. As you get more points the difficult level is increased by increased block falling speed. The musics speed is increased as the game speed increases. You get game over when the playfield level has reached to the top and there is not room for more blocks (See picture here to the right).



Game over screen

5.2 The software

One of the problems for Tetris is the memory required. The size of the playfield is 16x8 bricks, to be able to keep track of thee 7 different block kinds (different color for each kind) and also be able to represent empty area, 3 bits are required for each brick. As one byte is 8 bits I chose to represent each brick as one nibble (4 bits), making the playfield 64 bytes. I chose to organize the memory making to the top 4 banks of the memory and letting each memory bank represent two columns. The main game variables are placed in the first bank, some less used data such as score and a buffer of the next block and some other misc. stuff are placed in the second bank. The two remaining banks (except for the top four bytes of the fourth bank) are used as graphics buffers when outputting data to the DA. The sound frequency and sample position are stored in the top four bytes of the fourth bank.

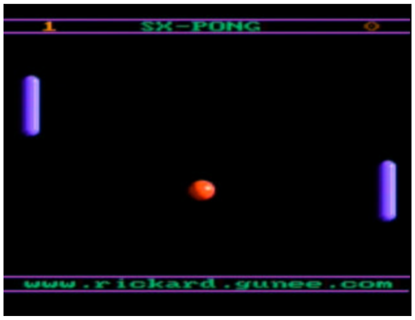
The tune Karboschka is stored in program memory as 52 notes and 52 note lengths, where the note refers to a frequency table with frequencies according to the temperated note scale (half notes differs one twelfth root of two in frequency). There is a 32-sample 4-bit sinus wave in program memory that is outputted to the audio DA at the pace of the current note translated through the frequency table. The code outputting the frequency is performed during the horizontal sync pulse, and the tune is updated at the bottom of the screen before the vertical sync. As the number of bits used for music is not very high, it sounds a little bit distorted and not very good, but better than nothing =)

Most of the game data of the game is stored as one big chunk to be able to use the program ROM more efficient. This is done by using all the 12 bits and the iread instruction, which makes it possible to store 50% more data than by using retlw, but at the cost of speed. It is hard to use 12bit data efficiently, but to make it easier I chose to separate the gamedata into one fastmem- and one slowmem-part, where the 8 lower bits of each 12-bit word is the fastmem and the upper 4 bits are the slowmem. Getting one byte from the slowmem requires two iread but the fastmem only requires one. Graphics objects are stored as 4-bit palette values, so is the music, but the font and text strings are all 8-bit values, so it is quite natural to store the 4-bit data in the upper part and the 8-bit data in the lower part.

The software is written to run for both PAL and NTSC with almost the same code, done by making all timing with constants. The constant system selects what TV system to use. In the code I have also prepared timing for PAL-M and PAL-N but they are not tested. It is not possible to generate SECAM color video signals in software with this design, so there is nothing in the code to support it. Note that the frequency which the chip should be clocked depends on your TV-system.

6. Pong

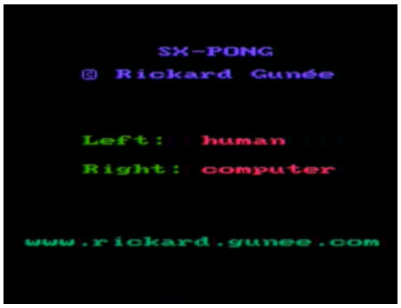
After making the tetris game, it was very easy to make a Pong game. The game Pong was the world's first video game in the early 70’s; this is a modern version of it, made with a little bit less hardware than the original version. In my version, the video signal is generated in software. The video generating hardware is a 5-bit DA converter built with a few resistors. Usually the video signal is generated in video games is created with a dedicated video chips, reading the image data from a graphics memory. In this project the video signal is calculated in real-time by the microprocessor as the electron beam sweeps over the screen.



Pong in action

6.1 How to play the game

The first screen is where you select how you want to play by moving the joystick: UP and DOWN to select *Human vs. Human*, *Human vs. Computer* or *Computer vs. Computer*. Start with FIRE. The computer vs. computer game to plays forever or until someone reset the game using the reset switch. You start serving by pressing fire, it is also possible to change direction and speed of the ball using fire. The player who has the serve will get points. If the player with the serve miss the ball, then the serve goes over to the other player. The paddles are moved up and down with the joysticks. It is possible to smash (increase speed) by pressing FIRE, and when doing so it is also possible to steer the ball by moving joystick up or down. When someone wins a game over picture will show and tell who won.


SX-PONG

© Rickard Gunée

Left: human

Right: computer

www.rickard.gunee.com

Game menu

Winner: Right

Game over screen

6.2 The software

The game logic is taken care of in the invisible lines at the top of the screen. The ball and the paddles are first generated with the *setgrapics* routine that loads the bitmap data and converts it using a palette and then writes it to the output buffer. The data in the buffer is outputted to the screen with the *memtovideo* routine. there is a delay before and after the ball is shown that varies depending on the ball position, the variation is divided into 12cycle steps to keep the phase of the signal correct. The code for the game control is mostly things to keep the ball and players within the screen, however it is not as easy as one could think as the program must always take exactly the same number of clock cycles or the TV looses its lock to the color carrier. Keeping track of all flow paths and keeping the timing is the largest problem when generating color signals in software.

The sound generation is very simple; there are two sound channels for outputting sound. The sound is called at the beginning of each scan line and outputs a sinus waveform from ROM for each channel to the audio DA, the position is updated according to the speed variable. The speed is changed according the speed change variable and thereby can be pitched up or down. A kind of bounce sound is created by pitching a ton down quickly when the ball bounces. There is also a timer variable to keep track of how many frames the sound should be active.

Most of the game data of the game is stored as one big chunk to be able to use the program ROM more efficient. This is done by using all the 12 bits and the iread instruction, which makes it possible to store 50% more data than by using retlw, but at the cost of speed. It is hard to use 12bit data efficiently, but to make it easier I chose to separate the game data into one fastmem- and one slowmem-part, where the 8 lower bits of each 12-bit word is the fastmem and the upper 4 bits are the slowmem. Getting one byte from the slowmem requires two iread but the fastmem only requires one. Graphics objects are stored as 4-bit palette values, so is the music, but the font and text strings are all 8-bit values, so it is quite natural to store the 4-bit data in the upper part and the 8-bit data in the lower part.

The software is written to run for both PAL and NTSC with almost the same code, done by making all timing with constants. The constant system selects what TV system to use. In the code I have also prepared timing for PAL-M and PAL-N but they are not tested. It is not possible to generate SECAM color video signals in software with this design, so there is nothing in the code to support it. Note that the frequency which the chip should be clocked depends on your TV-system.

7. Conclusions

It is possible to generate composite color video signals in software, but it is a lot of work and it is only possible in some special cases. NTSC is much more easy to do than PAL when doing the signal in software as phaseshifting is better done in hardware. The main reason for doing video in software is doing it for fun and that it is possible =), this form of video generation has very little commercial value as it takes huge amount of time to generate something with very poor result. Doing software based monochrome signal colored with hardware would give better result, but the best result is of course done with memory mapped graphics outputted with dedicated hardware.

Appendix A: Color test1 source code

```
;*******************************************************************************
;* SX COLOR TEST1 (C) Rickard Gunée, 2001                                      *
;*******************************************************************************
;* This is a test that shows the 12 phases as 12 colored lines.                *
;* The video signal is not 100% correct, it will not work on all TV:s, so if   *
;* your TV can't lock on the color signal or you get strange colors on the     *
;* screen then your TV probably can't run this game.                           *
;* This is an open source project and you may use this design and software     *
;* anyway you like as long as it is non comercial and you refer to the         *
;* original author with name and link to homepage.                             *
;* Use this at your own risk, don't blame me if you blow up your tv or kill    *
;* yourself or anyone else with it.                                            *
;*                                                                             *
;* For more info about project go to: http://www.rickard.gunee.com/projects    *
;*******************************************************************************

                    DEVICE    SX28,TURBO,STACKX_OPTIONX
                    RESET     start                         ;goto 'start' on reset
                    NOEXPAND

                    SYSTEM_PAL= 1
                    SYSTEM_PAL_N        = 2
                    SYSTEM_PAL_M        = 3
                    SYSTEM_NTSC         = 4

                    SYSTEM = SYSTEM_PAL ;This line selects TV-system timing to use


                    IF (SYSTEM = SYSTEM_PAL)

                              FREQ      53156550

                              TIME_2US4           EQU       128
                              TIME_4US5           EQU       239
                              TIME_27US5EQU       1463
                              TIME_29US6EQU       1574
                              TIME_64US           EQU       3405
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       89
                              TIME_SYNC           EQU       250
                              TIME_PREBURST       EQU       48
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       115

                              TOT_LINES           EQU       304
                              PRE_LINES           EQU       35
                              POST_LINESEQU       13

                              PHASESHIFT_MASK     EQU       2

                    ENDIF

                    IF (SYSTEM = SYSTEM_PAL_M)

                              FREQ      42907332

                              TIME_2US4           EQU       103
                              TIME_4US5           EQU       193
                              TIME_27US5EQU       1181
                              TIME_29US6EQU       1271
                              TIME_64US           EQU       2749
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       47
                              TIME_SYNC           EQU       202
                              TIME_PREBURST       EQU       39
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       5

                              TOT_LINES           EQU       254
                              PRE_LINES           EQU       35
                              POST_LINESEQU       13

                              PHASESHIFT_MASK     EQU       2

                    ENDIF


                    IF (SYSTEM = SYSTEM_PAL_N)

                              FREQ      42984672

                              TIME_2US4           EQU       103
                              TIME_4US5           EQU       193
                              TIME_27US5EQU       1181
                              TIME_29US6EQU       1271
                              TIME_64US           EQU       2749
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       47
                              TIME_SYNC           EQU       202
                              TIME_PREBURST       EQU       39
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       5

                              TOT_LINES           EQU       304
                              PRE_LINES           EQU       35
                              POST_LINESEQU       13

                              PHASESHIFT_MASK     EQU       2
                    ENDIF


                    IF (SYSTEM = SYSTEM_NTSC)

                              FREQ      42954540

                              TIME_2US4           EQU       103
                              TIME_4US5           EQU       193
                              TIME_27US5EQU       1181
                              TIME_29US6EQU       1271
                              TIME_64US           EQU       2748
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       47
                              TIME_SYNC           EQU       202
                              TIME_PREBURST       EQU       39
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       5

                              TOT_LINES           EQU       254
                              PRE_LINES           EQU       30
                              POST_LINESEQU       13

                              PHASESHIFT_MASK     EQU       0

                    ENDIF


                    delaytimer1         equ       08h
                    delaytimer2         equ       09h
                    temp0               equ       08h
                    temp1               equ       09h
                    temp2               equ       0Ah
                    temp3               equ       0Bh
                    temp4               equ       0Ch
                    temp5               equ       0Dh
                    temp6               equ       0Eh
                    temp7               equ       0Fh

                    stuff               equ       10h

                    black               equ       14
                    neutral             equ       14

                    frame               equ       0
                    phaseshiftequ       1

                    video               equ       RB
                    audio               equ       RC

                    TIME_HSYNC=         (TIME_PRESYNC + TIME_SYNC + TIME_PREBURST + TIME_BURST + TIME_POSTBURST)
                    TIME_IMAGE=         (TIME_TOTAL - TIME_HSYNC)
                    VISILINES           =         (TOT_LINES - PRE_LINES - POST_LINES)

;************************ vout macro **************************
;* This macro outputs a constant to the video DA              *
;**************************************************************

vout      MACRO     1
          mov       w,#(\1)
          mov       video,w
          ENDM


;*********************** voutr macro **************************
;* This macro outputs data from a register to the video DA    *
;**************************************************************

voutr     MACRO     1
          mov       w,\1
          mov       video,w
          ENDM


;************************ tnop macro **************************
;* This macro creates a delay of 3 clock cycles only using    *
;* one word of program memory.                                *
;**************************************************************

tnop      MACRO
          jmp :tnopj
:tnopj
          ENDM

;********************** setphase macro ************************
;* This is a macro for creating delay that depends of the     *
;* contents of w, it adds w to the low part of pc, and adds   *
;* nops after the jmp instruction, the number of nops is      *
;* specified as a parameter to the function                   *
;**************************************************************

setphase  MACRO     1
                    jmp       pc+w
                    REPT      \1
                    nop
                    ENDR
                    ENDM

;************************ delay macro *************************
;* This is a macro for creating delays by calling the delay   *
;* functions, it minimizes the number of program words to max *
;* 4 words. For delaytimes less than 1017 and longer than 9   *
;* it uses the short delay functions at the cost of 2-3 words *
;* for shorter delays it uses the fixed delays at a cost of 1 *
;* to 3 words, longer delays are done by a call to the short  *
;* delay functions followed by a long delay call with a total *
;* cost of 4-6 words of program memory. The macro can handle  *
;* delays from 0 to 260k cycles.                              *
;*                                                            *
;* WARNING, no guarantee that this really works correctly for *
;* all delays as it quite complex and I'm too lazy to test it *
;**************************************************************

delay     MACRO     1

:delbase
          IF (:delbase & $E00) = (delay9 & $E00)
            IF ((\1)<6)
              IF ((\1)//3)=1
              nop
            ENDIF
              IF ((\1)//3)=2
                nop
                nop
              ENDIF
              IF ((\1)/3) > 0
                REPT ((\1)/3)
                  tnop
                ENDR
              ENDIF
            ENDIF

            IF ((\1)>5) AND ((\1)<10)
              call  delay6 - ((\1)-6)
            ENDIF

            IF ((\1) > 9) AND ((\1)<1027)
              mov w,#((\1)-6)>>2
              call delay_short_0 - (((\1)-6)&3)
            ENDIF

            IF (\1) > 1026
              IF (((\1)-12)//1017)<10
              mov w,#(((((\1)-12)//1017)+1017)>>2)
                call delay_short_0 - (((((\1)-12)//1017)+1017)&3)
                mov w,#(((\1)-12)/1017)-1
              ELSE
              mov w,#((((\1)-12)//1017)>>2)
                call delay_short_0 - ((((\1)-12)//1017)&3)
                mov w,#(((\1)-12)/1017)
              ENDIF
                call delay_long
            ENDIF
          ELSE
            IF ((\1)<7)
              IF ((\1)//3)=1
              nop
            ENDIF
              IF ((\1)//3)=2
                nop
                nop
              ENDIF
              IF ((\1)/3) > 0
                REPT ((\1)/3)
                  tnop
                ENDR
              ENDIF
            ENDIF

            IF ((\1)>6) AND ((\1)<11)
              page delay6
              call delay6 - ((\1)-7)
            ENDIF

            IF ((\1) > 10) AND ((\1)<1028)
              mov w,#((\1)-7)>>2
              page delay_short_0
              call delay_short_0 - (((\1)-7)&3)
            ENDIF

            IF (\1) > 1027
              IF (((\1)-14)//1017)<10
              mov w,#(((((\1)-14)//1017)+1017)>>2)
                page delay_short_0
                call delay_short_0 - (((((\1)-14)//1017)+1017)&3)
                mov w,#(((\1)-14)/1017)-1
              ELSE
              mov w,#((((\1)-14)//1017)>>2)
                page delay_short_0
                call delay_short_0 - ((((\1)-14)//1017)&3)
                mov w,#(((\1)-14)/1017)
              ENDIF
                page delay_long
                call delay_long
            ENDIF
          ENDIF
ENDM


;********************** delay functions ***********************
;* Different delay functions to be able to create long delays *
;* using as few bytes of program memory as possible           *
;* These functions are required by the delay macro            *
;* delays with exact clock count uses no registers            *
;* short delays use temp0                                     *
;* long delays use temp0 and temp1                            *
;**************************************************************

delay9              nop                           ;1                  entrypoint of delay9 that delays 9 clocks
delay8              nop                           ;1                  entrypoint of delay8 that delays 8 clocks
delay7              nop                           ;1                  entrypoint of delay7 that delays 7 clocks
delay6              retp                          ;3                  entrypoint of delay6 that delays 6 clocks

delay_short_3       nop                           ;1                  entrypoint of delay_short_3 that delays 4*w + 8
delay_short_2       nop                           ;1                  entrypoint of delay_short_3 that delays 4*w + 7
```

```
delay_short_1       nop                           ;1                  entrypoint of delay_short_3 that delays 4*w + 6
delay_short_0       mov       temp0,w             ;1                  entrypoint of delay_short_3 that delays 4*w + 5
delay_short_m       decsz     temp0               ;1(2)               decrease counter, mainloop of delay short
                    jmp       delay_short_m       ;3                  keep looping until counnter is zero
                    retp                          ;3                  return back to caller

delay_longmov       temp1,w             ;1                  set long time counter from w
delay_long_l        mov       w,#251              ;1                  set time to delay in short delay
                    call      delay_short_3       ;1012               time to delay is 251*4+8=1012
                    decsz     temp1               ;1(2)               decrease long time counter
                    jmp       delay_long_l        ;3                  keep looping until counnter is zero
                    retp                          ;1                  return back to caller


;********************** simplecolorfa *************************
;* outputs w color cycles at (almost) maximum amplitude       *
;* Clocks: w*12 + 11 + 1                                      *
;**************************************************************

simplecolorfa       mov       temp2,w             ;1
                    mov       temp0,#63           ;2
                    mov       temp1,#black        ;2
                    skip                          ;2

;*********************** simplecolor **************************
;* outputs w color cycles                                     *
;* Clocks: w*12 + 6                                           *
;**************************************************************

simplecolor         mov       temp2,w             ;1        set colorcycle counter
simplecolor_l       voutr     temp0               ;2        set first level
                    delay     4                   ;4        delay to get 12cycle loop
                    voutr     temp1               ;2        set second level
                    decsz     temp2               ;1(2)     decrease colorcycle counter
                    jmp       simplecolor_l       ;3        do all cycles
                    retp                          ;3        get outa here


;************************* start ******************************
;* Start sequence, sets up system                             *
;**************************************************************

start               clr       fsr
clr_l               setb      fsr.4
                    clr       ind
                    incsz     fsr
                    jmp       clr_l

                    mode      $F
                    mov       !RB,#%11000001
                    mov       !RC,#%11100000
                    mode      $E
                    mov       !RA,#%0000
                    mov       !RB,#%00111110
                    mov       !RC,#%00011111

                    jmp       main


;************************* vsync ******************************
;* Performas a vertical sync pattern on the video output      *
;* Uses temp0..temp2                                          *
;**************************************************************

vsync
          IF (TOT_LINES // 2 = 0)                 ;         make sure start phase is shifted between frames
                    mov       w,#PHASESHIFT_MASK  ;1        get mask that shift phase
                    xor       stuff,w             ;1        shift phase
          ENDIF
                    mov       w,#4                ;1        odd, make 5 pulses instead
                    call      short_sync;5        clocks until sync, make those pulses,
                    mov       temp2,w             ;1        counter0=5
long_sync_l         clr       video               ;1        set video level to sync
                    delay     (TIME_27US5 - 1)    ;         30uS long sync pulse
                    vout      black               ;2        set video level to black
                    delay     (TIME_4US5 - 6)     ;         2us long black pulse
                    decsz     temp2               ;1(2)
                    jmp       long_sync_l         ;3
                    mov       w,#5                ;1        odd, make 4 pulses instead of 5
short_syncmov       temp2,w             ;1
short_sync_l        clr       video               ;1        set video level to sync
                    delay     (TIME_2US4 - 1)     ;2us long sync pulse
                    vout      black               ;2        set video level to black
                    delay     (TIME_29US6 - 6)    ;         30us long black pulse
                    decsz     temp2               ;1(2)
                    jmp       short_sync_l        ;3
                    retw      5                   ;3


;************************* hsync ******************************
;* performas a horizontal sync pulse and color burst          *
;* uses temp0                                                 *
;**************************************************************

hsync               mov       w,#PHASESHIFT_MASK  ;1        get mask that shift phase
                    xor       stuff,w             ;1        shift phase
                    delay     TIME_PRESYNC-3-1-2
                    clr       video               ;1
                    delay     TIME_SYNC-2
                    vout      neutral             ;2

IF PHASESHIFT_MASK = 0
                    delay     TIME_PREBURST-2     ;44
ELSE
                    delay     TIME_PREBURST-2-4
                    snb       stuff.phaseshift    ;1
                    jmp       nophaseshift1       ;3
                    delay     5                   ;5
nophaseshift1

ENDIF

                    mov       temp0,#12           ;2
hsyncl              vout      21                  ;2
                    delay     4                   ;4
                    vout      6                   ;2
                    decsz     temp0               ;1(2)
                    jmp       hsyncl              ;3
                    delay     2                   ;2
                    vout      neutral             ;2

IF PHASESHIFT_MASK = 0
                    delay     time_postburst - 2-3;114
ELSE
                    sb        stuff.phaseshift    ;1
                    jmp       nophaseshift2       ;3
                    delay     5                   ;5
nophaseshift2

                    delay     time_postburst - 2-3-7
ENDIF
                    retp                          ;3


;*********************** emptylines ***************************
;* Displays w empty lines, 17clocks until hsync when called   *
;* and 12 clocks until next hsync when returned               *
;**************************************************************

emptylinesmov       temp3,w
emptylines_l        delay     13                  ;13
                    call      hsync               ;
                    delay     (TIME_IMAGE-4-13)   ;
                    decsz     temp3               ;1(2)
                    jmp       emptylines_l        ;3
                    ret                           ;3


;*********************** main loop ****************************
;* This is the game main loop                                 *
;**************************************************************

main                call      vsync               ;         vertical sync, frame starts here
                    delay     17-1
                    mov       w,#PRE_LINES        ;1
                    call      emptylines;         do empty lines at top outside of screen

                    delay     12-8
                    mov       temp6,#0            ;2        set phase upwards counter to zero
                    mov       temp7,#11           ;2        set phase downwards counter to 11 (maximum phase)
                    mov       temp4,#12           ;2        set field counter to do 12 fields

mainl0              mov       temp5,#((VISILINES-12)/12)    ;2        set linecounter to number of lines in field
mainl1              call      hsync               ;         do horizontal sync pulse
                    mov       w,temp6             ;1        get phase
                    snb       stuff.phaseshift    ;1(2)     check if phase should be shifted
                    mov       w,temp7             ;1        if so get inverted phase from downwards counter
                    setphase 11                   ;         set phase from w
                    mov       w,#((TIME_IMAGE-39) / 12)     ;1
                    call      simplecolorfa
                    delay     ((TIME_IMAGE-39) // 12)
                    mov       w,temp7
                    snb       stuff.phaseshift
                    mov       w,temp6
                    setphase 11
                    decsz     temp5
                    jmp       mainl1
                    inc       temp6
                    dec       temp7
                    call      hsync
                    delay     TIME_IMAGE - 4 - 2
                    decsz     temp4
                    jmp       mainl0

                    delay     4
                    call hsync
                    delay     TIME_IMAGE - 17 - 1
                    mov       w,#POST_LINES - 1 + ((VISILINES-12)//12)
                    call      emptylines
                    jmp       main
```

Appendix B: Color test2 source code

```
;*******************************************************************************
;* SX COLOR TEST2 (C) Rickard Gunée, 2001                                      *
;*******************************************************************************
;* This is a test that shows all available colors on my SX-based gamesystem.   *
;* The video signal is not 100% correct, it will not work on all TV:s, so if   *
;* your TV can't lock on the color signal or you get strange colors on the     *
;* screen then your TV probably can't run this game.                           *
;* This is an open source project and you may use this design and software     *
;* anyway you like as long as it is non comercial and you refer to the         *
;* original author with name and link to homepage.                             *
;* Use this at your own risk, don't blame me if you blow up your tv or kill    *
;* yourself or anyone else with it.                                            *
;*                                                                             *
;* For more info about project go to: http://www.rickard.gunee.com/projects    *
;*******************************************************************************

                    DEVICE    SX28,TURBO,STACKX_OPTIONX
                    RESET     start                         ;goto 'start' on reset
                    NOEXPAND

                    SYSTEM_PAL= 1
                    SYSTEM_PAL_N        = 2
                    SYSTEM_PAL_M        = 3
                    SYSTEM_NTSC         = 4

                    SYSTEM = SYSTEM_NTSC;This line selects TV-system timing to use



                    IF (SYSTEM = SYSTEM_PAL)

                              FREQ      53156550

                              TIME_2US4           EQU       128
                              TIME_4US5           EQU       239
                              TIME_27US5EQU       1463
                              TIME_29US6EQU       1574
                              TIME_64US           EQU       3405
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       89
                              TIME_SYNC           EQU       250
                              TIME_PREBURST       EQU       48
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       114

                              TOT_LINES           EQU       304
                              PRE_LINES           EQU       35
                              POST_LINESEQU       13
                    ENDIF

                    IF (SYSTEM = SYSTEM_PAL_M)

                              FREQ      42907332

                              TIME_2US4           EQU       103
                              TIME_4US5           EQU       193
                              TIME_27US5EQU       1181
                              TIME_29US6EQU       1271
                              TIME_64US           EQU       2749
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       47
                              TIME_SYNC           EQU       202
                              TIME_PREBURST       EQU       39
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       5

                              TOT_LINES           EQU       254
                              PRE_LINES           EQU       35
                              POST_LINESEQU       13
                    ENDIF


                    IF (SYSTEM = SYSTEM_PAL_N)

                              FREQ      42984672

                              TIME_2US4           EQU       103
                              TIME_4US5           EQU       193
                              TIME_27US5EQU       1181
                              TIME_29US6EQU       1271
                              TIME_64US           EQU       2749
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       47
                              TIME_SYNC           EQU       202
                              TIME_PREBURST       EQU       39
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       5

                              TOT_LINES           EQU       304
                              PRE_LINES           EQU       35
                              POST_LINESEQU       13
                    ENDIF


                    IF (SYSTEM = SYSTEM_NTSC)

                              FREQ      42954540

                              TIME_2US4           EQU       103
                              TIME_4US5           EQU       193
                              TIME_27US5EQU       1181
                              TIME_29US6EQU       1271
                              TIME_64US           EQU       2748
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       47
                              TIME_SYNC           eQU       202
                              TIME_PREBURST       EQU       39
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       5

                              TOT_LINES           EQU       254
                              PRE_LINES           EQU       30
                              POST_LINESEQU       13
                    ENDIF


                    delaytimer1         equ       08h
                    delaytimer2         equ       09h
                    temp0               equ       08h
                    temp1               equ       09h
                    temp2               equ       0Ah
                    temp3               equ       0Bh
                    temp4               equ       0Ch
                    temp5               equ       0Dh
                    temp6               equ       0Eh
                    temp7               equ       0Fh

                    black               equ       14
                    neutral             equ       14

                    frame               equ       0

                    video               equ       RB
                    audio               equ       RC


                    TIME_HSYNC=         (TIME_PRESYNC + TIME_SYNC + TIME_PREBURST + TIME_BURST + TIME_POSTBURST)
                    TIME_IMAGE=         (TIME_TOTAL - TIME_HSYNC)
                    VISILINES           =         (TOT_LINES - PRE_LINES - POST_LINES)

;************************ vout macro **************************
;* This macro outputs a constant to the video DA              *
;**************************************************************

vout      MACRO     1
          mov       w,#(\1)
          mov       video,w
          ENDM


;*********************** voutr macro **************************
;* This macro outputs data from a register to the video DA    *
;**************************************************************

voutr     MACRO     1
          mov       w,\1
          mov       video,w
          ENDM


;************************ tnop macro **************************
;* This macro creates a delay of 3 clock cycles only using    *
;* one word of program memory.                                *
;**************************************************************

tnop      MACRO
          jmp :tnopj
:tnopj
          ENDM

;********************** setphase macro ************************
;* This is a macro for creating delay that depends of the     *
;* contents of w, it adds w to the low part of pc, and adds   *
;* nops after the jmp instruction, the number of nops is      *
;* specified as a parameter to the function                   *
;**************************************************************

setphase  MACRO     1
                    jmp       pc+w
                    REPT      \1
                    nop
                    ENDR
                    ENDM

;************************ delay macro *************************
;* This is a macro for creating delays by calling the delay   *
;* functions, it minimizes the number of program words to max *
;* 4 words. For delaytimes less than 1017 and longer than 9   *
;* it uses the short delay functions at the cost of 2-3 words *
;* for shorter delays it uses the fixed delays at a cost of 1 *
;* to 3 words, longer delays are done by a call to the short  *
;* delay functions followed by a long delay call with a total *
;* cost of 4-6 words of program memory. The macro can handle  *
;* delays from 0 to 260k cycles.                              *
;*                                                            *
;* WARNING, no guarantee that this really works correctly for *
;* all delays as it quite complex and I'm too lazy to test it *
;**************************************************************

delay     MACRO     1

:delbase
          IF (:delbase & $E00) = (delay9 & $E00)
            IF ((\1)<6)
              IF ((\1)//3)=1
              nop
            ENDIF
              IF ((\1)//3)=2
```

```
                nop
                nop
              ENDIF
              IF ((\1)/3) > 0
                REPT ((\1)/3)
                  tnop
                ENDR
              ENDIF
            ENDIF

            IF ((\1)>5) AND ((\1)<10)
              call  delay6 - ((\1)-6)
            ENDIF

            IF ((\1) > 9) AND ((\1)<1027)
              mov w,#((\1)-6)>>2
              call delay_short_0 - (((\1)-6)&3)
            ENDIF

            IF (\1) > 1026
              IF (((\1)-12)//1017)<10
              mov w,#(((((\1)-12)//1017)+1017)>>2)
                call delay_short_0 - (((((\1)-12)//1017)+1017)&3)
                mov w,#(((\1)-12)/1017)-1
              ELSE
              mov w,#((((\1)-12)//1017)>>2)
                call delay_short_0 - ((((\1)-12)//1017)&3)
                mov w,#(((\1)-12)/1017)
              ENDIF
                call delay_long
            ENDIF
          ELSE
            IF ((\1)<7)
              IF ((\1)//3)=1
              nop
           ENDIF
              IF ((\1)//3)=2
                nop
                nop
              ENDIF
              IF ((\1)/3) > 0
                REPT ((\1)/3)
                  tnop
                ENDR
              ENDIF
            ENDIF

            IF ((\1)>6) AND ((\1)<11)
              page delay6
              call delay6 - ((\1)-7)
            ENDIF

            IF ((\1) > 10) AND ((\1)<1028)
              mov w,#((\1)-7)>>2
              page delay_short_0
              call delay_short_0 - (((\1)-7)&3)
            ENDIF

            IF (\1) > 1027
              IF (((\1)-14)//1017)<10
              mov w,#(((((\1)-14)//1017)+1017)>>2)
                page delay_short_0
                call delay_short_0 - (((((\1)-14)//1017)+1017)&3)
                mov w,#(((\1)-14)/1017)-1
              ELSE
              mov w,#((((\1)-14)//1017)>>2)
                page delay_short_0
                call delay_short_0 - ((((\1)-14)//1017)&3)
                mov w,#(((\1)-14)/1017)
              ENDIF
                page delay_long
                call delay_long
            ENDIF
          ENDIF
ENDM


;********************** delay functions ***********************
;* Different delay functions to be able to create long delays *
;* using as few bytes of program memory as possible           *
;* These functions are required by the delay macro            *
;* delays with exact clock count uses no registers            *
;* short delays use temp0                                     *
;* long delays use temp0 and temp1                            *
;**************************************************************

delay9              nop                           ;1        entrypoint of delay9 that delays 9 clocks
delay8              nop                           ;1        entrypoint of delay8 that delays 8 clocks
delay7              nop                           ;1        entrypoint of delay7 that delays 7 clocks
delay6              retp                          ;3        entrypoint of delay6 that delays 6 clocks

delay_short_3       nop                           ;1        entrypoint of delay_short_3 that delays 4*w + 8
delay_short_2       nop                           ;1        entrypoint of delay_short_3 that delays 4*w + 7
delay_short_1       nop                           ;1        entrypoint of delay_short_3 that delays 4*w + 6
delay_short_0       mov       temp0,w             ;1        entrypoint of delay_short_3 that delays 4*w + 5
delay_short_m       decsz     temp0               ;1(2)     decrease counter, mainloop of delay short
                    jmp       delay_short_m       ;3        keep looping until counnter is zero
                    retp                          ;3        return back to caller

delay_longmov       temp1,w             ;1        set long time counter from w
delay_long_l        mov       w,#251              ;1        set time to delay in short delay
                    call      delay_short_3       ;1012     time to delay is 251*4+8=1012
                    decsz     temp1               ;1(2)     decrease long time counter
                    jmp       delay_long_l        ;3        keep looping until counnter is zero
                    retp                          ;1        return back to caller


;************************* start ******************************
```

```
;* Start sequence, sets up system                             *
;**************************************************************

start               clr       fsr
clr_l               setb      fsr.4
                    clr       ind
                    incsz     fsr
                    jmp       clr_l

                    mode      $F
                    mov       !RB,#%11000001
                    mov       !RC,#%11100000
                    mode      $E
                    mov       !RA,#%0000
                    mov       !RB,#%00111110
                    mov       !RC,#%00011111

                    jmp       main

;************************* vsync ******************************
;* Performas a vertical sync pattern on the video output      *
;* Uses temp0..temp2                                          *
;**************************************************************

vsync               mov       w,#4                ;1        odd, make 5 pulses instead
                    call      short_sync;5        clocks until sync, make those pulses,
                    mov       temp2,w             ;1        counter0=5
long_sync_l         clr       video               ;1        set video level to sync
                    delay     (TIME_27US5 - 1)    ;         30uS long sync pulse
                    vout      black               ;2        set video level to black
                    delay     (TIME_4US5 - 6)     ;         2us long black pulse
                    decsz     temp2               ;1(2)
                    jmp       long_sync_l         ;3
                    mov       w,#5                ;1        odd, make 4 pulses instead of 5
short_syncmov       temp2,w             ;1
short_sync_l        clr       video               ;1        set video level to sync
                    delay     (TIME_2US4 - 1)     ;2us long sync pulse
                    vout      black               ;2        set video level to black
                    delay     (TIME_29US6 - 6)    ;         30us long black pulse
                    decsz     temp2               ;1(2)
                    jmp       short_sync_l        ;3
                    retw      5                   ;3


;************************* hsync ******************************
;* performas a horizontal sync pulse and color burst          *
;* uses temp0                                                 *
;**************************************************************

hsync               delay     TIME_PRESYNC-3-1    ;85
                    clr       video               ;1
                    delay     TIME_SYNC-2         ;248
                    vout      neutral             ;2

                    delay     TIME_PREBURST-2     ;44
                    mov       temp0,#12           ;2
hsyncl              vout      5                   ;2
                    delay     4                   ;4
                    vout      24                  ;2
                    decsz     temp0               ;1(2)
                    jmp       hsyncl              ;3
                    delay     2                   ;2
                    vout      neutral             ;2
                    delay     time_postburst - 2-3 ;114
                    retp                          ;3


;*********************** emptylines ***************************
;* Displays w empty lines, 17clocks until hsync when called   *
;* and 12 clocks until next hsync when returned               *
;**************************************************************

emptylinesmov       temp3,w
emptylines_l        delay     13                  ;13
                    call      hsync               ;
                    delay     (TIME_IMAGE-4-13)   ;
                    decsz     temp3               ;1(2)
                    jmp       emptylines_l        ;3
                    ret                           ;3


;*********************** main loop ****************************
;* This is the main loop                                      *
;**************************************************************

main                call      vsync               ;         vertical sync, frame starts here

                    delay     17-1
                    mov       w,#PRE_LINES + 10   ;1
                    call      emptylines;         do empty lines at top outside of screen

                    delay     12-6
                    mov       temp3,#0            ;2        set phase upwards counter to zero
                    mov       temp4,#11           ;2        set phase downwards counter to 11 (maximum phase)
                    mov       temp5,#6            ;2        set field counter to do 6 fields
mainlp              call      hsync
                    mov       temp2,#28           ;2
                    delay     TIME_IMAGE - 2

mainl0              call      hsync
                    delay     120
                    mov       w,#7                ;1
                    add       w,temp2             ;1
                    add       w,temp2             ;1
                    mov       temp1,w             ;1
                    mov       temp0,#56           ;2
                    mov       w,temp3             ;1
                    setphase 11
mainl11             mov       w,temp1             ;1
                    mov       video,w             ;1
                    delay     3                   ;3
                    mov       w,#7                ;1
                    add       w,temp0             ;1
                    mov       video,w             ;1
                    decsz     temp0               ;1(2)
                    jmp       mainl11             ;3
                    vout      black
                    mov       w,temp4             ;1
                    setphase 11
                    delay     3
                    mov       temp0,#56           ;2
                    mov       w,temp4             ;1
                    setphase 11
mainl12             mov       w,temp1             ;1
                    mov       video,w             ;1
                    delay     3                   ;3
                    mov       w,#7                ;1
                    add       w,temp0             ;1
                    mov       video,w             ;1
                    decsz     temp0               ;1(2)
                    jmp       mainl12             ;3
                    vout      black
                    mov       w,temp3
                    setphase 11
                    delay     TIME_IMAGE - (((12*56)-2+8+11+2+2)*2) - (3+120+4+4)
                    decsz     temp2
                    jmp       mainl0

                    inc       temp3
                    dec       temp4
                    call      hsync
                    delay TIME_IMAGE - 4
                    decsz     temp5
                    jmp       mainlp

                    delay 2
                    call      hsync
                    delay     TIME_IMAGE - 17 - 1
                    mov       w,#POST_LINES - 1 + (VISILINES - 10 - (30*6))
                    call      emptylines
                    jmp       main
```

Appendix C: Tetris source code

```
;*******************************************************************************
;* SX-TETRIS (C) Rickard Gunée, 2001                                           *
;*******************************************************************************
;* This is the classical computer game tetris, outputing a color video signal  *
;* in software using a couple of resistors.                                    *
;* The video signal is not 100% correct, it will not work on all TV:s, so if   *
;* your TV can't lock on the color signal or you get strange colors on the     *
;* screen then your TV probably can't run this game.                           *
;* This is an open source project and you may use this design and software     *
;* anyway you like as long as it is non comercial and you refer to the         *
;* original author with name and link to homepage.                             *
;* Use this at your own risk, don't blame me if you blow up your tv or kill    *
;* yourself or anyone else with it.                                            *
;*                                                                             *
;* For more info about project go to: http://www.rickard.gunee.com/projects    *
;*******************************************************************************

                    DEVICE    SX28,TURBO,STACKX_OPTIONX
                    RESET     jumpstart                     ;goto 'start' on reset
                    NOEXPAND

                    SYSTEM_PAL= 1
                    SYSTEM_PAL_N        = 2
                    SYSTEM_PAL_M        = 3
                    SYSTEM_NTSC         = 4

                    SYSTEM = SYSTEM_PAL ;This line selects TV-system timing to use



                    IF (SYSTEM = SYSTEM_PAL)

                              FREQ      53156550

                              TIME_2US4           EQU       128
                              TIME_4US5           EQU       239
                              TIME_27US5EQU       1463
                              TIME_29US6EQU       1574
                              TIME_64US           EQU       3405
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       89
                              TIME_SYNC           EQU       250
                              TIME_PREBURST       EQU       48
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       114

                              TIME_LEFTGFX        EQU       80*12
                              TIME_RIGHTGFX       EQU       40*12
                              LEFTGFX_BASE        EQU       12*10

                              TOT_LINES           EQU       304
                              PRE_LINES           EQU       35
                              POST_LINESEQU       13

                              BRICK_WIDTH         EQU       7

                              BLINE_PHASE         EQU       5
                              CAP_BASE            EQU       70*12
                              CAP_PHASE           EQU       6
                              CAP_PHASEDIFF       EQU       -1
                              LEFTSCREW_PHASE     EQU       7
                              RIGHTSCREW_PHASE    EQU       0
                              GAMEFIELD_PHASE     EQU       6


                              SCORE_BASEEQU       12*6
                              SCORE_PHASE         EQU       9
                              TEXTNEXT_BASE       EQU       12*6
                              TEXTNEXT_PHASE      EQU       7
                              TEXTSCORE_BASE      EQU       12*2
                              TEXTSCORE_PHASE     EQU       1
                              NBLOCK_BASE         EQU       12*6
                              NBLOCK_PHASE        EQU       6
                              GAMEOVER_PHASE      EQU       6
                              GAMEOVER_BASE       EQU       17*12

                              STR0_BASE           EQU       38*12
                              STR0_PHASEEQU       11
                              STR1_BASE           EQU       30*12
                              STR1_PHASEEQU       7


                    ENDIF

                    IF (SYSTEM = SYSTEM_PAL_M)

                              FREQ      42907332

                              TIME_2US4           EQU       103
                              TIME_4US5           EQU       193
                              TIME_27US5EQU       1181
                              TIME_29US6EQU       1271
                              TIME_64US           EQU       2749
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       47
                              TIME_SYNC           EQU       202
                              TIME_PREBURST       EQU       39
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       5
```

```
          TOT_LINES           EQU       254
          PRE_LINES           EQU       35
          POST_LINESEQU       13

          TIME_LEFTGFX        EQU       80*12
          TIME_RIGHTGFX       EQU       40*12
          LEFTGFX_BASE        EQU       12*10

          BRICK_WIDTH         EQU       5

          BLINE_PHASE         EQU       3
          CAP_BASE            EQU       70*12
          CAP_PHASE           EQU       4
          CAP_PHASEDIFF       EQU       0
          LEFTSCREW_PHASE     EQU       5
          RIGHTSCREW_PHASE    EQU       9
          GAMEFIELD_PHASE     EQU       3


          SCORE_BASEEQU       12*6
          SCORE_PHASE         EQU       8
          TEXTNEXT_BASE       EQU       12*6
          TEXTNEXT_PHASE      EQU       6
          TEXTSCORE_BASE      EQU       12*2
          TEXTSCORE_PHASE     EQU       0
          NBLOCK_BASE         EQU       12*6
          NBLOCK_PHASE        EQU       4

          GAMEOVER_PHASE      EQU       7
          GAMEOVER_BASE       EQU       10*12

          STR0_BASE           EQU       20*12
          STR0_PHASEEQU       10
          STR1_BASE           EQU       13*12
          STR1_PHASEEQU       5
ENDIF


IF (SYSTEM = SYSTEM_PAL_N)

          FREQ      42984672

          TIME_2US4           EQU       103
          TIME_4US5           EQU       193
          TIME_27US5EQU       1181
          TIME_29US6EQU       1271
          TIME_64US           EQU       2749
          TIME_TOTALEQU       TIME_64US
          TIME_PRESYNC        EQU       47
          TIME_SYNC           EQU       202
          TIME_PREBURST       EQU       39
          TIME_BURSTEQU       144
          TIME_POSTBURST      EQU       5

          TOT_LINES           EQU       304
          PRE_LINES           EQU       35
          POST_LINESEQU       13

          TIME_LEFTGFX        EQU       80*12
          TIME_RIGHTGFX       EQU       40*12
          LEFTGFX_BASE        EQU       12*10

          TOT_LINES           EQU       304
          PRE_LINES           EQU       35
          POST_LINESEQU       13

          TIME_LEFTGFX        EQU       80*12
          TIME_RIGHTGFX       EQU       40*12
          LEFTGFX_BASE        EQU       12*10

          BRICK_WIDTH         EQU       5

          BLINE_PHASE         EQU       3
          CAP_BASE            EQU       70*12
          CAP_PHASE           EQU       4
          CAP_PHASEDIFF                 EQU       0
          LEFTSCREW_PHASE     EQU       5
          RIGHTSCREW_PHASE    EQU       9
          GAMEFIELD_PHASE     EQU       3


          SCORE_BASEEQU       12*6
          SCORE_PHASE         EQU       8
          TEXTNEXT_BASE       EQU       12*6
          TEXTNEXT_PHASE      EQU       6
          TEXTSCORE_BASE      EQU       12*2
          TEXTSCORE_PHASE     EQU       0
          NBLOCK_BASE         EQU       12*6
          NBLOCK_PHASE        EQU       4

          GAMEOVER_PHASE      EQU       7
          GAMEOVER_BASE       EQU       10*12

          STR0_BASE           EQU       20*12
          STR0_PHASEEQU       10
          STR1_BASE           EQU       13*12
          STR1_PHASEEQU       5
ENDIF


IF (SYSTEM = SYSTEM_NTSC)

          FREQ      42954545

          TIME_2US4           EQU       103
          TIME_4US5           EQU       193
          TIME_27US5EQU       1181
          TIME_29US6EQU       1271
          TIME_64US           EQU       2748
          TIME_TOTALEQU       TIME_64US
          TIME_PRESYNC        EQU       47
          TIME_SYNC           EQU       202
          TIME_PREBURST       EQU       39
          TIME_BURSTEQU       144
          TIME_POSTBURST      EQU       5

          TOT_LINES           EQU       254
          PRE_LINES           EQU       25
          POST_LINESEQU       13

          TIME_LEFTGFX        EQU       80*12
          TIME_RIGHTGFX       EQU       40*12
          LEFTGFX_BASE        EQU       12*10

          BRICK_WIDTH         EQU       5

          BLINE_PHASE         EQU       3
          CAP_BASE            EQU       70*12
          CAP_PHASE           EQU       4
          CAP_PHASEDIFF       EQU       0
          LEFTSCREW_PHASE     EQU       5
          RIGHTSCREW_PHASE    EQU       9
          GAMEFIELD_PHASE     EQU       3


          SCORE_BASEEQU       12*6
          SCORE_PHASE         EQU       8
          TEXTNEXT_BASE       EQU       12*6
          TEXTNEXT_PHASE      EQU       6
          TEXTSCORE_BASE      EQU       12*2
          TEXTSCORE_PHASE     EQU       0
          NBLOCK_BASE         EQU       12*6
          NBLOCK_PHASE        EQU       4

          GAMEOVER_PHASE      EQU       7
          GAMEOVER_BASE       EQU       10*12

          STR0_BASE           EQU       20*12
          STR0_PHASEEQU       10
          STR1_BASE           EQU       13*12
          STR1_PHASEEQU       5

ENDIF



delaytimer1         equ       08h
delaytimer2         equ       09h
temp0               equ       08h
temp1               equ       09h
temp2               equ       0Ah
temp3               equ       0Bh
temp4               equ       0Ch
temp5               equ       0Dh
temp6               equ       0Eh
temp7               equ       0Fh


joy1up              equ       RB.7
joy1down            equ       RC.5

joy1left            equ       RC.6
joy1right           equ       RC.7

joy1buttonequ       RB.6

x                   equ       $10
y                   equ       $11
kind                equ       $12
angle               equ       $13
nextkind            equ       $14
falltimer           equ       $15
oldjoy              equ       $16
joytimer            equ       $17
blockbuff           equ       $18

mixedbits           equ       $10
rnd                 equ       $11
gfxcnt              equ       $12
linecnt             equ       $13
musictimerequ       $14
songpos             equ       $15

stemp0              equ       $1B
pos                 equ       $1C
pos_l               equ       $1C
pos_h               equ       $1D
sfreq               equ       $1E
sfreq_l             equ       $1E
sfreq_h             equ       $1F

joy1button_old      equ       oldjoy.2

black               equ       14
neutral             equ       14

JTIME               equ       10

frame               equ       7
gameoverbit         equ       6

video               equ       RB
audio               equ       RC

CAP_SEP             EQU       (((BRICK_WIDTH + 2)*8*12) + (13*12))
```

```
                    TIME_HSYNCEQU       (TIME_PRESYNC + TIME_SYNC + TIME_PREBURST + TIME_BURST + TIME_POSTBURST)
                    TIME_IMAGEEQU       (TIME_TOTAL - TIME_HSYNC)
                    VISILINES           EQU       (TOT_LINES - PRE_LINES - POST_LINES)

                    TOP_LINES           EQU       (VISILINES/30)
                    STRTOCAP_LINES      EQU       (VISILINES/20)
                    BRICK_LINES         EQU       (VISILINES/25)
                    PLAYFIELD_LINES     EQU       ((BRICK_LINES+2)*16)

                    VIDEO_BUFFER        EQU       $50

                    SCORE               EQU       $3C
                    NEXTGFX             EQU       $39

                    FONT                EQU       $0        ;fastmem
                    STR0                EQU       $100      ;fastmem
                    STR0_LEN            EQU       20        ;fastmem
                    STR1                EQU       $115      ;fastmem
                    STR1_LEN            EQU       22        ;fastmem
                    STR2                EQU       $12c      ;fastmem
                    STR2_LEN            EQU       6         ;fastmem
                    STR3                EQU       $133      ;fastmem
                    STR3_LEN            EQU       6         ;fastmem
                    STR4                EQU       $13a      ;fastmem
                    STR4_LEN            EQU       5         ;fastmem
                    STR5                EQU       $140      ;fastmem
                    STR5_LEN            EQU       5         ;fastmem
                    NUMBERS             EQU       $146      ;fastmem
                    SCREW               EQU       $0        ;slowmem
                    CAP                 EQU       $40       ;slowmem
                    SINTABLE            EQU       $b9       ;slowmem
                    FREQTBL             EQU       $d9       ;slowmem
                    MUSIC               EQU       $169      ;slowmem


org 2

;*********************** add16 macro **************************
;* This is a macro to add two 16bit numbers, inputs two       *
;* arguments, each pointing at the lsb register followed by   *
;* the msb register at poistion arg+1.                        *
;* Reults is stored in registers referred to by first arg     *
;* arg1 = arg1 + arg2                                         *
;* clocks: 6                                                  *
;**************************************************************

add16     MACRO     2
          add       (\1),(\2)                     ;2
          snc                                     ;1(2)
          inc       (\1) + 1                      ;1
          add       (\1) + 1, (\2) + 1            ;2
          ENDM


;*********************** pcall macro **************************
;* This macro does the same as lcall but in 2 words           *
;**************************************************************

pjmp      MACRO     1
          page (\1)
          jmp       (\1)
          ENDM


;*********************** pcall macro **************************
;* This macro does the same as lcall but in 2 words           *
;**************************************************************

pcall     MACRO     1
          page (\1)
          call      (\1)
          ENDM


;************************ vout macro **************************
;* This macro outputs a constant to the video DA              *
;**************************************************************

vout      MACRO     1
          mov       w,#(\1)
          mov       video,w
          ENDM


;*********************** voutr macro **************************
;* This macro outputs data from a register to the video DA    *
;**************************************************************

voutr     MACRO     1
          mov       w,\1
          mov       video,w
          ENDM


;*********************** itext macro **************************
;* macro for showing a line of chars from rom                 *
;* parameters: strpointer,length,base,phase                   *
;**************************************************************

ITEXT               MACRO     4
                    mov       temp7,#8                                ;2
                    mov       temp4,#((gamedata + FONT) & $ff)        ;2
:bots_l             pcall     hsync                                   ;1+TIME_HSYNC
                    delay     (\3) - (\4)
                    mov       temp1,#(((\1) + gamedata) >> 8)         ;2
                    mov       temp3,#(((\1) + gamedata) & $FF)        ;2
                    mov       temp5,#(\2)                             ;2
                    pcall     strout                                  ;STR_LEN*8 * 12 * (w-1) + 42 + 1
                    inc       temp4                                   ;1
                    delay     TIME_IMAGE-((((\2)-1)*8*12) + 44 + 1) - (\3) + (\4) - (2+2+2+1+4+1)
                    decsz     temp7                                   ;1(2)
                    jmp       :bots_l                                 ;3
                    delay     2
                    pcall     hsync
                    ENDM

;************************ tnop macro ***************************
;* This macro creates a delay of 3 clock cycles only using    *
;* one word of program memory.                                 *
;***************************************************************

tnop      MACRO
:tnopf
                    IF (:tnopf & %111000000000) = ((:tnopf+1) & %111000000000)
                              jmp :tnopf + 1
                    ELSE
                              nop
                              nop
                              nop
                    ENDIF
          ENDM

;********************** setphase macro ************************
;* This is a macro for creating delay that depends of the      *
;* contents of w, it adds w to the low part of pc, and adds    *
;* nops after the jmp instruction, the number of nops is       *
;* specified as a parameter to the function                    *
;***************************************************************

setphase  MACRO     1
                    jmp       pc+w
                    REPT      \1
                    nop
                    ENDR
                    ENDM

;************************ delay macro **************************
;* This is a macro for creating delays by calling the delay   *
;* functions, it minimizes the number of program words to max *
;* 4 words. For delaytimes less than 1017 and longer than 9   *
;* it uses the short delay functions at the cost of 2-3 words *
;* for shorter delays it uses the fixed delays at a cost of 1 *
;* to 3 words, longer delays are done by a call to the short  *
;* delay functions followed by a long delay call with a total *
;* cost of 4-6 words of program memory. The macro can handle  *
;* delays from 0 to 260k cycles.                              *
;*                                                            *
;* WARNING, no guarantee that this really works correctly for *
;* all delays as it quite complex and I'm too lazy to test it *
;**************************************************************

delay     MACRO     1

:delbase
          IF (:delbase & $E00) = (delay9 & $E00)
            IF ((\1)<6)
              IF ((\1)//3)=1
              nop
            ENDIF
              IF ((\1)//3)=2
                nop
                nop
              ENDIF
              IF ((\1)/3) > 0
                REPT ((\1)/3)
                    TNOP
                ENDR
              ENDIF
            ENDIF

            IF ((\1)>5) AND ((\1)<10)
              call  delay6 - ((\1)-6)
            ENDIF

            IF ((\1) > 9) AND ((\1)<1027)
              mov w,#((\1)-6)>>2
              call delay_short_0 - (((\1)-6)&3)
            ENDIF

            IF (\1) > 1026
              IF (((\1)-12)//1017)<10
              mov w,#(((((\1)-12)//1017)+1017)>>2)
                call delay_short_0 - (((((\1)-12)//1017)+1017)&3)
                mov w,#(((\1)-12)/1017)-1
              ELSE
              mov w,#((((\1)-12)//1017)>>2)
                call delay_short_0 - ((((\1)-12)//1017)&3)
                mov w,#(((\1)-12)/1017)
              ENDIF
                call delay_long
            ENDIF
          ELSE
            IF ((\1)<7)
              IF ((\1)//3)=1
              nop
            ENDIF
              IF ((\1)//3)=2
                nop
                nop
              ENDIF
              IF ((\1)/3) > 0
                REPT ((\1)/3)
```

```
                    TNOP
                ENDR
              ENDIF
            ENDIF

            IF ((\1)>6) AND ((\1)<11)
              page delay6
              call delay6 - ((\1)-7)
            ENDIF

            IF ((\1) > 10) AND ((\1)<1028)
              mov w,#((\1)-7)>>2
              page delay_short_0
              call delay_short_0 - (((\1)-7)&3)
            ENDIF

            IF (\1) > 1027
              IF (((\1)-14)//1017)<10
              mov w,#(((((\1)-14)//1017)+1017)>>2)
                page delay_short_0
                call delay_short_0 - (((((\1)-14)//1017)+1017)&3)
                mov w,#(((\1)-14)/1017)-1
              ELSE
              mov w,#((((\1)-14)//1017)>>2)
                page delay_short_0
                call delay_short_0 - ((((\1)-14)//1017)&3)
                mov w,#(((\1)-14)/1017)
              ENDIF
                page delay_long
                call delay_long
            ENDIF
          ENDIF
ENDM


;********************** delay functions ***********************
;* Different delay functions to be able to create long delays *
;* using as few bytes of program memory as possible           *
;* These functions are required by the delay macro            *
;* delays with exact clock count uses no registers            *
;* short delays use temp0                                     *
;* long delays use temp0 and temp1                            *
;**************************************************************

delay9              nop                           ;1                  entrypoint of delay9 that delays 9 clocks
delay8              nop                           ;1                  entrypoint of delay8 that delays 8 clocks
delay7              nop                           ;1                  entrypoint of delay7 that delays 7 clocks
delay6              retp                          ;3                  entrypoint of delay6 that delays 6 clocks

delay_short_3       nop                           ;1                  entrypoint of delay_short_3 that delays 4*w + 8
delay_short_2       nop                           ;1                  entrypoint of delay_short_3 that delays 4*w + 7
delay_short_1       nop                           ;1                  entrypoint of delay_short_3 that delays 4*w + 6
delay_short_0       mov       temp0,w             ;1                  entrypoint of delay_short_3 that delays 4*w + 5
delay_short_m       decsz     temp0               ;1(2)               decrease counter, mainloop of delay short
                    jmp       delay_short_m       ;3                  keep looping until counnter is zero
                    retp                          ;3                  return back to caller

delay_longmov       temp1,w             ;1                  set long time counter from w
delay_long_l        mov       w,#251              ;1                  set time to delay in short delay
                    call      delay_short_3       ;1012               time to delay is 251*4+8=1012
                    decsz     temp1               ;1(2)               decrease long time counter
                    jmp       delay_long_l        ;3                  keep looping until counnter is zero
                    retp                          ;1                  return back to caller


;17
readsong  mov       m,#((MUSIC + gamedata) >> 8)            ;1
                    mov       w,#(MUSIC + gamedata) & $FF             ;1
                    add       w,songpos                               ;1
                    snc                                               ;1(2)
                    mov       m,#((MUSIC + gamedata) >> 8) + 1        ;1
                    iread                                             ;4
                    mov       w,m                                     ;1
                    inc       songpos                                 ;1
                    ret                                               ;3

;16
readfreqtbl         mov       m,#((FREQTBL + gamedata) >> 8);1
                    mov       w,#(FREQTBL + gamedata) & $FF           ;1
                    add       w,temp0                                 ;1
                    snc                                               ;1(2)
                    mov       m,#((FREQTBL + gamedata) >> 8)+1        ;1
                    iread                                             ;4
                    mov       w,m                                     ;1
                    ret                                               ;3




;************************ readjoy1 ****************************
;* Reads joy1 bits from RC and RB and combining them to w     *
;* temp register 0 used                                       *
;* clocks: 12 + 1                                             *
;**************************************************************

readjoy1  mov       w,RC                ;1
                    and       w,#%11100000        ;1
                    mov       temp0,w             ;1
                    mov       w,<>RB              ;1
                    and       w,#%00001100        ;1
                    or        w,temp0             ;1
                    retp                          ;3


;*********************** memtovideo ***************************
;* outputs data from memory to video output                   *
;* number of clocks: w*12 + 7 + 1                             *
;* temp register 0 used                                       *
;**************************************************************

memtovideomov       temp0,w             ;1        set pixelcounter
mtvl0               mov       w,ind               ;1        get lower level byte from mem
                    mov       video,w             ;1        send to video output
                    setb      fsr.5               ;1        select upper bank
                    mov       w,ind               ;1        get upper level byte from mem
                    inc       fsr                 ;1        point at next pixel
                    clrb      fsr.5               ;1        select lower bank
                    nop                           ;1
                    mov       video,w             ;1        send to video output
                    decsz     temp0               ;1(2)     decrease pixel counter
                    jmp       mtvl0               ;3        keep looping until all pixels are done
                    vout      BLACK               ;2        set black color
                    retp                          ;3        get outa here


;********************** setgraphics ***************************
;* outputs data from memory to video output                   *
;* number of clocks: w*31 + 5 +1                              *
;* temp0 = bitmap rom-pointer bit 0..7                        *
;* temp1 = bitmap rom-pointer bit 8..11                       *
;* temp2 = palette rom-pointer bit 0..7                       *
;* fsr = pointr to memory where to store graphics             *
;* Note: bits 8..11 of palette pointer is in the constant     *
;* called PALETTE_PAGE, all palettes should be placed within  *
;* this page. fsr,temp0 and temp1 are modyfied                *
;**************************************************************

setgraphics         mov temp3,w                   ;1        set pixelcounter
sgl0                mov m,temp1                   ;2        set page
                    mov w,temp0                   ;1        get image pointer
                    iread                         ;4        read pixeldata from rom
                    mov w,m                       ;1        get slowmem nibble
                    add w,temp2                   ;1        select palettte, assuming all palettes within the same page
                    mov m,#PALETTE_PAGE           ;1        select page
                    iread                         ;4        read palette
                    mov ind,w                     ;1        remember first level
                    setb fsr.5          ;1        select second level bank
                    and w,#$C0          ;1        mask out two upper bits
                    mov ind,w                     ;1        store second level two upper bits
                    rr ind                        ;1        move upper bits into correct position (1/2)
                    rr ind                        ;1        move upper bits into correct position (2/2)
                    mov w,m                       ;1        get second level lower nibble
                    or ind,w                      ;1        stor second level lower nibble
                    clrb fsr.5          ;1        get back to first level bank
                    inc fsr                       ;1        point at next pixel memory position
                    inc temp0                     ;1        point at next nibble
                    snz                           ;1(2)
                    inc temp1                     ;1        if page overflow, go to next page
                    decsz temp3                   ;1(2)     decrease pixel counter
                    jmp sgl0                      ;3        keep looping until all pixels are done
                    retp                          ;3        get outa here



;************************* blocks *****************************
;* get compressed brick data                                  *
;* clock cycles: 20                                           *
;**************************************************************

blocks              and       w,#%1111
                    add       pc,w                ;3
                    retw      $50                 ;3
                    retw      $44                 ;3
                    retw      $D0                 ;3
                    retw      $0C                 ;3
                    retw      $D0                 ;3
                    retw      $0C                 ;3
                    retw      $D0                 ;3
                    retw      $3C                 ;3
                    retw      $D0                 ;3
                    retw      $CC                 ;3
                    retw      $F4                 ;3
                    retw      $C0                 ;3
                    retw      $5C                 ;3
                    retw      $C0                 ;3
                    retw      $00                 ;3
                    retw      $6C                 ;3


;*********************** brickposcheck ************************
;* Check if out of bounds, calculate address to brick and     *
;* mask to unwanted nibble                                    *
;* clock cycles: 20                                           *
;* uses temp0..temp3                                          *
;* temp0 = x-position                                         *
;* temp1 = y-position                                         *
;* returns out of bounds as a nonzero value in temp3          *
;* returns bitmask in w                                       *
;**************************************************************

;20 clocks
brickposcheck       mov       w,#$F8              ;1        get illegal x-positions
                    and       w,temp0             ;1        mask out illegal x-bits for x-position
                    mov       temp3,w             ;1        store illegal bits for later
                    mov       w,#$F0              ;1        get illegal y-positions
                    and       w,temp1             ;1        mask out illegal y-bits for y-position
                    or        temp3,w             ;1        combine iillegal x- and y- bits and store in temp3
                    mov       w,<>temp0           ;1        get x-position and swap nibbles
                    and       w,#$60              ;1        only keep former bit 1 and 2
                    add       w,temp1             ;1        add y-position
                    or        w,#$90              ;1        set bit 4 and 7 to get correct address
                    mov       fsr,w               ;1        set file select register to calculated pointer
                    mov       w,#$F0              ;1        get bitmask for left brick
                    snb       temp0.0             ;1(2)     check if x-pos is odd
                    mov       w,#$0F              ;1        yes, get bitmask for right brick instead
                    ret                           ;3        get outa here


;************************* setbrick ***************************
;* Sets a brick on the position temp0, temp1 with color temp3 *
;* clock cycles: 34                                           *
;* uses temp0..temp3                                          *
;* temp0 = x-position                                         *
;* temp1 = y-position                                         *
;* temp2 = color                                              *
;**************************************************************

setbrick  call      brickposcheck                 ;20       calc address, check out of bounds and get bitmask
                    test      temp3               ;1        out of bounds ? (1/2)
                    sz                            ;1(2)     out of bounds ? (2/2)
                    jmp       delay9              ;3        yes, out of bounds, do delayed return
                    and       ind,w               ;1        clear wanted nibble
                    mov       w,temp2             ;1        get color
                    snb       temp0.0             ;1        check if x is odd
                    mov       w,<>temp2           ;1        yes, get color with swapped nibbles instead
                    or        ind,w               ;1        set color
                    ret                           ;3        get outa here


;************************ checkbrick **************************
;* Checks if there is a brick on the position temp0, temp1    *
;* returns nonxzero value of there is a brick and zero if the *
;* position is clear                                          *
;* clock cycles: 33                                           *
;* uses temp0..temp3                                          *
;* temp0 = x-position                                         *
;* temp1 = y-position                                         *
;**************************************************************

checkbrickcall      brickposcheck       ;20
                    test      temp3               ;1        out of bounds ? (1/2)
                    sz                            ;1(2)     out of bounds ? (2/2)
                    jmp       delay8              ;3        yes, out of bounds, do delayed return
                    not       w                   ;1        invert bits to get wanted nibble
                    and       w,ind               ;1        get wanted nibble from playfield
                    ret                           ;3        get outa here


;*********************** checksetblock *************************
;* If bit 0 in temp6 is set then the the block in blockbuff    *
;* is drawn in the playfiled on position x,y with color temp2  *
;* using 221 clocks                                            *
;* If bit 0 in temp6 is clear then the the block in blockbuff  *
;* is checked for collitions on position x,y in the playfield  *
;* returning result in temp7 using 217 clocks                  *
;* tempregs 0..7 are used                                      *
;* The reason of combining these two operations is that they   *
;* are very similar, combining them will save program mem      *
;* checkblock calls checksetblock with temp6.0 cleared (221+1) *
;* setblock calls checksetblock with temp6.0 set (223+1)       *
;***************************************************************

checkblockclrb      temp6.0             ;1        set checking (clear setting)
                    skip                          ;2        don't set setting
setblock  setb      temp6.0                       ;1        set setting
checksetblock       clr       temp7               ;1        clear result
                    mov       temp4,#blockbuff    ;2        point temp4 at block buffer
                    mov       temp5,#4            ;2        each block has 4 bricks

setblock_lmov       fsr,temp4           ;2        set file select register to block buffer pointer
                    mov       w,x                 ;1        get block base x-position
                    add       w,ind               ;1        add relative brick position
                    mov       temp0,w             ;1        store brick x-position
                    inc       fsr                 ;1        point at next buffer x-position
                    mov       w,y                 ;1        get block base y-position
                    add       w,ind               ;1        add relative brick y-position
                    mov       temp1,w             ;1        store brick y-position
                    sb        temp6.0             ;1(2)
                    call      checkbrick;33
                    snb       temp6.0             ;1(2)
                    call      setbrick            ;34
                    or        temp7,w             ;1        store result from check
                    add       temp4,#2            ;2        update buffer pointer to next brick
                    decsz     temp5               ;1(2)     decrease brick counter
                    jmp       setblock_l;3        keep loopin until all 4 bris are out

                    bank      $00
                    retp                          ;3        get outa here

jumpstart pjmp      start


;********************** simplecolorfa ************************
;* outputs w color cycles at (almost) maximum amplitude       *
;* Clocks: w*12 + 11 + 1                                      *
;**************************************************************

simplecolorfa       mov       temp2,w             ;1
                    mov       temp0,#63           ;2
                    mov       temp1,#black        ;2
                    skip                          ;2


;*********************** simplecolor *************************
;* outputs w color cycles                                     *
;* Clocks: w*12 + 5 + 1                                       *
;**************************************************************

simplecolor         mov       temp2,w             ;1        set colorcycle counter
simplecolor_l       voutr     temp0               ;2        set first level
                    delay     4                   ;4        delay to get 12cycle loop
                    voutr     temp1               ;2        set second level
                    decsz     temp2               ;1(2)     decrease colorcycle counter
                    jmp       simplecolor_l       ;3        do all cycles
                    retp                          ;3        get outa here

;************************ makeblock ***************************
;* Get compressed coordinates from rom and genereate rotated  *
;* uncompressed coordinates in block buffer in ram            *
;* Clocks: 204+1                                              *
;**************************************************************

mbexpand  and       w,#3                          ;1        mask out current block
                    add       pc,w                ;3        get to correct value
                    retw      0                   ;3        return value
                    retw      1                   ;3        return value
                    retw      2                   ;3        return value
                    retw      -1                  ;3        return value

makeblock bank      $00                           ;1        set bank 0
                    mov       w,<<kind            ;1        relative address = kind*2
                    call      blocks              ;9        get block x-data from table
                    mov       temp0,w             ;1        store inn x-data temporary register (temp0)
                    mov       w,<<kind            ;1
                    or        w,#1                ;1        relative address = kind*2 + 1
                    call      blocks              ;9        get block y-data from table
                    mov       temp1,w             ;1        store inn y-data temporary register (temp1)
                    mov       fsr,#blockbuff      ;2        point at block buffer
                    mov       w,angle             ;1        get angle
                    and       w,#3                ;1        limit angle to 0..3
                    mov       temp4,w             ;1        store in local angle
                    mov       w,kind              ;1        what kind of block do we have ?
                    and       w,#%00000111        ;1        check lower bits of kindword
                    snz                           ;1(2)     zero ? (non rotatable square)
                    clr       temp4               ;1        yes, do not rotate, set angle to zero
                    mov       w,>>temp4           ;1        get bit2 of angle
                    xor       w,temp4             ;1        xor it with bit2
                    mov       temp2,w             ;1        and store result in temp2 (temp2.0 is set for angle 1&2)
                    sb        temp4.0             ;1(2)     if (angle = 0) or (angle = 2)
                    jmp       mbnoswap            ;3        then don't swap x and y
                    mov       w,temp1             ;1        else do swap x and y by xoring
                    xor       w,temp0             ;1        .
                    xor       temp1,w             ;1        .
                    xor       temp0,w             ;1        .
mbnoswapc mov       temp3,#4                      ;2        4 bricks in each block, set counter to 4
                    clr       temp4               ;1        a register vcontaining zero is needed for later

makeblock_l         mov       w,temp0             ;1        get x-data
                    call      mbexpand            ;10
                    snb       temp2.1             ;1        if (angle = 2) or (angle = 3)
                    mov       w,temp4 - w         ;1        then mirror angle
                    mov       ind,w               ;1        store in buffer
                    inc       fsr                 ;1        point at next position in buffer
                    mov       w,temp1             ;1        get y-data
                    call      mbexpand            ;10
                    snb       temp2.0             ;1        if (angle = 1) or (angle = 2)
                    mov       w,temp4 - w         ;1        then mirror angle
                    mov       ind,w               ;1        store in buffer
                    inc       fsr                 ;1        point at next position in buffer
                    rr        temp0               ;1        temp0 = temp0 >> 2
                    rr        temp0               ;1        .
                    rr        temp1               ;1        temp1 = temp1 >> 2
                    rr        temp1               ;1        .
                    decsz     temp3               ;1(2)     decrease brick counter
                    jmp       makeblock_l         ;3        keep looping until all bricks are done

                    bank      $00                 ;1        set bank 0
                    retp                          ;3        get outa here

mbnoswap  jmp       mbnoswapc                     ;3        time portal to get 6 clocks if x and y are not swapped

;************************* start ******************************
;* Start sequence, sets up system                             *
;**************************************************************

start


                    clr       fsr
clr_l               setb      fsr.4
                    clr       ind
                    incsz     fsr
                    jmp       clr_l

                    bank      $20
                    mov       musictimer,#1

                    bank      $00
                    mov       joytimer,#JTIME     ;2
                    mov       x,#4
                    mov       y,#2
                    mov       kind,#$12
                    mov       falltimer,#50

                    mode      $F
                    mov       !RB,#%11000001
                    mov       !RC,#%11100000
                    mode      $E
                    mov       !RA,#%0000
                    mov       !RB,#%00111110
                    mov       !RC,#%00011111

                    bank      NEXTGFX
                    mov       NEXTGFX,#$9F
                    mov       NEXTGFX+1,#$F9

                    bank      $20
                    pjmp      main
```

```
;*********************** updatemusic **************************
;* Music player, called once per frame, playing "karboshka"   *
;* from rom, each position of the tune is stored as a note    *
;* followed by a length. The length is multyplied with the    *
;* gamespeed making it play faster with the speed of the game *
;* Each note is translated to a "frequency" from a table in   *
;* ROM, notes are separated with a short (2 frames) pause     *
;* clocks: 130 + 1                                            *
;**************************************************************

updatemusic         bank      $20                 ;1
                    decsz     musictimer;1        decrease note/pause length timer
                    jmp       musicnochnote       ;3        if not zero, don't update note info

                    snb mixedbits.gameoverbit     ;1        no music if gameover
                    jmp       musicpausep

                    bank      $60                 ;1
                    mov       w,sfreq_l ;1
                    or        w,sfreq_h ;1        is frequency zero ?

                    sz                            ;1        yes, don't make a pause
                    jmp       musicpause;3        no, make a pause (i.e. set freq to zero for two frames)

                    bank      $20                 ;1
                    call      readsong            ;17       get next nibble of the song from rom, the note (i.e position
in frequency table)
                    bank      $60                 ;1
                    mov       temp0,w             ;1        temp0 = 1*freqtablepos
                    add       temp0,w             ;1        temp0 = 2*freqtablepos
                    add       temp0,w             ;1        temp0 = 3*freqtablepos
                    call      readfreqtbl         ;16       get bit 0..3 from rom
                    mov       sfreq_l,w           ;1        store in high byte of frequency
                    inc       temp0               ;1        point at next position in rom
                    call      readfreqtbl         ;16       get bit 4..7 from rom
                    swap      sfreq_l             ;1        swap nibbles to be ready for next nibble
                    or        sfreq_l,w           ;1        or the two nibbles together
                    swap      sfreq_l             ;1        swap nibbles to get back correct order of nibbles
                    inc       temp0               ;1        point at next position in rom
                    call      readfreqtbl         ;16       get bit 8..11 from rom
                    mov       sfreq_h,w           ;1        store in high byte of frequency

                    bank      $20                 ;1
                    mov       temp1,#11 ;2        temp1 = 11
                    sub       temp1,(SCORE+1)     ;2        temp1 = 11-speed = note baselength
                    call      readsong            ;17       get next nibble of the song from rom, the notelength
                    mov       temp0,w             ;1        put notelength in temp0 to be able to do tests
                    mov       w,>>temp1           ;1        w = (11-speed)/2
                    test      temp0               ;1        update flags according to notelength
                    clc                           ;1        clear  carry  to  prevent  a  set  carrybiit  pollution  of  the
speed multyplier
                    sz                            ;1        check if notelenth is larger than zero
                    rl        temp1               ;1        temp1 = (11-speed)*2
                    snb       temp0.1             ;2(4)     check if notelength is 1
                    add       temp1,w             ;1        temp1 = speed*2 + speed/2 = speed * 2.5
                    mov       w,temp1             ;1        w = temp1 = lengthmultiplier (1, 2 or 2.5)
                    mov       musictimer,w        ;1        set notelength to (11-speed)*lengthmultiplier

                    mov       w,songpos           ;1        get song position
                    xor       w,#104              ;1        xor with songlength
                    snz                           ;1        if result is zero then we have reached the end of the song
                    clr       songpos             ;1        and should restart the song
                    pjmp      main                ;4        get back to main


musicnochnote       delay     130-12              ;         delay to keep timing when no change of frequency
                    pjmp      main                ;4        get back to main

musicpausep         bank      $60
                    delay     3

musicpauseclr       sfreq_l             ;1        clear low byte of rfrequency
                    clr       sfreq_h             ;1        clear high byte of frequency
                    clr       pos_l               ;1
                    clr       pos_h               ;1

                    bank      $20                 ;1
                    mov       musictimer,#2       ;2        pause is for two cycles
                    delay     130-25              ;         delay to keep timing when setting pause
                    pjmp      main                ;4        get back to main

;*********************** vrealsound **************************
;* vrealsound is called from vsync and calls realsound every  *
;* second vsync cycle as vsync is called at twice the speed   *
;* as sync, so vrealsound is dependent of realsound           *
;* clocks: 39 + 1                                             *
;**************************************************************

vrealsoundsb        temp2.0             ;1(2)
                    jmp       realsound ;35
                    delay     35-1-3
                    retp                          ;3

;************************ realsound ***************************
;* realsound is called from hsync to output sound data to the *
;* sound DA, the sound is based on a 16bit sin signal in rom. *
;* clocks: 35 + 1                                             *
;**************************************************************

realsound mov       w,fsr                                             ;1
                    bank      $60                                     ;1
                    mov       stemp0,w                                ;1
                    mov       m,#((SINTABLE+gamedata) >> 8)           ;1        point at corrent page for sintable
                    add16     pos,sfreq                               ;6        update  sintable  position  according  to
speed
                    and       pos_h,#31                               ;2        keep sample position in range 0..31
                    mov       temp0,pos_h                             ;2        get high part i wave position
                    add       temp0,#((SINTABLE+gamedata )& $FF)      ;2        add low part of pointer to sintable and
position
                    mov       w,temp0                                 ;1        the sum, the low pointer should be in w
                    iread                                             ;4        read from rom
                    mov       w,m                                     ;1        get high nibble, i.e w=sin(pos)
                    mov       temp0,w                                 ;1        temp0 = 7 + sin(pos)
                    add       temp0,#7                                ;2
                    mov       w,<<temp0                               ;1        w = (7 + sin(pos))*2
                    mov       audio,w                                 ;1        output value to audio DA
                    mov       fsr,stemp0                    ;2
                    retp                                              ;3


PALETTE_BCW         EQU $0              ;word-mem

gamedata2 dw $808,$80f,$816,$81d,$824,$82b,$832,$839,$f3f,$67f,$d7f,$3bf,$abf,$1ff,$8ff,$fff

PALETTE_PAGE        EQU (( gamedata2 + PALETTE_BCW)>>8)

ORG       $200

;************************* vsync ******************************
;* Performas a vertical sync pattern on the video output      *
;* Uses temp0..temp2                                          *
;**************************************************************

vsync               mov       w,#4                          ;1        odd, make 5 pulses instead
                    call      short_sync          ;5        clocks until sync, make those pulses,
                    mov       temp2,w                       ;1        counter0=5
long_sync_l         clr       video                         ;1        set video level to sync
                    delay     (TIME_27US5 - 1)              ;         30uS long sync pulse
                    vout      black                         ;2        set video level to black
                    call      vsound                        ;43
                    delay     (TIME_4US5 - 6 - 43);         2us long black pulse
                    decsz     temp2                         ;1(2)
                    jmp       long_sync_l                   ;3
                    mov       w,#5                          ;1        odd, make 4 pulses instead of 5
short_syncmov       temp2,w                       ;1
short_sync_l        clr       video                         ;1        set video level to sync
                    call      vsound                        ;43
                    delay     (TIME_2US4 - 43 - 1)          ;2us long sync pulse
                    vout      black                         ;2        set video level to black
                    delay     (TIME_29US6 - 6)              ;         30us long black pulse
                    decsz     temp2                         ;1(2)
                    jmp       short_sync_l                  ;3
                    retw      5                             ;3
vsound              pjmp      vrealsound          ;40                           ;3

;************************* hsync ******************************
;* performas a horizontal sync pulse and color burst          *
;* uses temp0                                                 *
;**************************************************************

hsync               delay     TIME_PRESYNC-3-1              ;85
                    clr       video                         ;1

                    call      sound                         ;39
                    delay     TIME_SYNC-2-39
                    vout      neutral                       ;2

                    delay     TIME_PREBURST-2               ;44
                    mov       temp0,#12                     ;2
hsyncl              vout      6                             ;2
                    delay     4                             ;4
                    vout      21                            ;2
                    decsz     temp0                         ;1(2)
                    jmp       hsyncl                        ;3
                    delay     2                             ;2
                    vout      neutral                       ;2
                    delay     time_postburst - 2-3;114
                    retp                                    ;3
sound               pjmp      realsound                     ;36


;*********************** emptylines ***************************
;* Displays w empty lines, 17clocks until hsync when called   *
;* and 12 clocks until next hsync when returned               *
;**************************************************************

emptylinesmov       temp3,w
emptylines_l        delay     13                            ;13
                    call      hsync                         ;
                    delay     (TIME_IMAGE-4-13)             ;
                    decsz     temp3                         ;1(2)
                    jmp       emptylines_l                  ;3
                    ret                                     ;3


;******************** brickcolortable *************************
;* function to get phase and amplitude of block w             *
;**************************************************************

brickcolortable     add       pc,w                ;3
                    retw      (black)<<2;3 black
                    retw      $D0                 ;3 purple-pink
                    retw      $A6                 ;3 green-cyan
                    retw      $A6                 ;3 green-cyan
                    retw      $C1                 ;3 red
                    retw      $F2                 ;3 orange
                    retw      $F3                 ;3 yellow
                    retw      $F4                 ;3 green
                    retw      $E7                 ;3 blue


;*********************** main loop ****************************
;* This is the game main loop                                 *
```

```
;***************************************************************

main                bank      $20
                    clr       linecnt
                    clr       gfxcnt
                    call      vsync               ;         vertical sync, frame starts here


                    snb mixedbits.gameoverbit
                    pjmp gameover

                    call      hsync               ;643      first line starts here

;------------ Remove block from playfield ------------- 430 clocks

                    pcall     makeblock           ;205      create current block in buffer
                    clr       temp2               ;1        set color 0
                    pcall     setblock            ;224      set black block at current position, to remove block from
playfield

;---------------- Handle block falling ---------------- 987 clocks

                    decsz     falltimer           ;1(2)     decrease falltimer
                    jmp       nofall              ;3        if falltimer hasn't reached zero, then don't fall
                    bank      $20                 ;1        set bank 1
                    inc       rnd

                    mov       temp0,#10 ;2
                    sub       temp0,SCORE+1       ;2
                    mov       w,<<temp0 ;1
                    bank      $00                 ;1        set bank 0
                    mov       falltimer,w         ;1        set falltimer to gamespeed
                    rl        falltimer ;1

                    inc       y                   ;1        move block one step down
                    pcall     checkblock;222      check if there was anything in the way
                    test      temp7               ;1        was fall possible ? (1/2)
                    snz                           ;1(2)     was fall possible ? (2/2)
                    jmp       fallok              ;3        yes, continue

                    bank      $00                 ;1        set bank 0
                    mov       w,kind              ;1        set color of current block
                    and       w,#%00000111        ;1
                    mov       temp2,w             ;1
                    inc       temp2               ;1
                    dec       y                   ;1
                    pcall     setblock            ;224      place the block on playfield
                    mov       temp0,nextkind      ;2        store next kind in a tempreg to be able to transer to store
in kind later
                    bank      $20                 ;1        set bank1

                    xor       w,rnd               ;1        compare nextkind with rnd
                    snz                           ;1        equal ?
                    inc       rnd                 ;1        yes, boring, increase rnd to avoid same block twice

                    mov       w,rnd               ;1        get a new nextblock
                    and       w,#7                ;1

                    bank      $00                 ;1        set bank0
                    mov       nextkind,w;1

                    mov       w,temp0             ;1        get old nextkind
                    and       w,#%11111000        ;1        clear lower bits just leaving the scrollposition
                    or        nextkind,w;1        combine it with the new nextkind

                    mov       w,#%11111000        ;1
                    and       kind,w              ;1
                    mov       w,temp0             ;1
                    and       w,#%00000111        ;1
                    or        kind,w              ;1

                    clr       angle               ;1        set angle to zero
                    mov       y,#2                ;2        set y-position to 2
                    mov       x,#4                ;2        set x-position to 4
                    pcall     makeblock           ;205      decompress the new block
                    pcall     checkblock;222

                    bank      $20
                    test      temp7               ;1
                    sz                            ;1(2)
                    setb      mixedbits.gameoverbit
                    pcall     incpoints           ;60
                    bank      $00                 ;1
                    jmp       donefall            ;3        and we are done with handle of fall

nofall              delay     242-4               ;235      there was no fall, delay to get timing correct
fallok              delay     987-242             ;732      fall was ok so we don't need to restore fall, delay to get
timing correct
donefall

;-------------- Handle joystick motion ---------------- 23 clocks

                    pcall     readjoy1  ;13
                    not       w                   ;1
                    snz                           ;1(2)
                    jmp       nojoy               ;3
                    xor       w,oldjoy  ;1
                    snz                           ;1(2)
                    jmp       joyok               ;3
                    decsz     joytimer  ;1(2)
                    jmp       nojoytimeout        ;3
joyok               mov       joytimer,#JTIME     ;2

;----------------- Joystick down ? -------------------- 3 clocks

                    mov       w,#1                ;1        prepare end of fall
                    sb        joy1down  ;1(2)     joy down ?
                    mov       falltimer,w         ;1        set falltimer to end of fall
```

```
;----------------- Joystick left ? -------------------- 231 clocks

                    snb       joy1left            ;1(2)     check if joystick moved left
                    jmp       noleft              ;3        if not, do nothing
                    dec       x                   ;1        else try to move block one step left
                    pcall     checkblock;222      check if there was anything in the way
                    test      temp7               ;1        was motion possible ? (1/2)
                    sz                            ;1(2)     was motion possible ? (1/2)
                    inc       x                   ;1        no, retract move
                    jmp       doneleft            ;3        we are done here

noleft              delay     231-4               ;226      no left motion, wait to get timing correct
doneleft

;----------------- Joystick right ? -------------------- 231 clocks

                    snb       joy1right ;1(2)     check if joystick moved right
                    jmp       noright             ;3        if not, do nothing
                    inc       x                   ;1        else try to move block one step right
                    pcall     checkblock;222      check if there was anything in the way
                    test      temp7               ;1        was motion possible ? (1/2)
                    sz                            ;1(2)     was motion possible ? (1/2)
                    dec       x                   ;1        no, retract move
                    jmp       doneright ;3        we are done here

noright             delay     231-4               ;226      no right motion, wait to get timing correct
doneright


;----------------- Joystick button ? -------------------- 436 clocks

                    snb       joy1button;1(2)     check if joystick buttton pressed
                    jmp       norotate            ;3        if not, do nothing
                    inc       angle               ;1        else try to rotate block

                    pcall     makeblock ;205
                    pcall     checkblock;222      check if there was anything in the way
                    test      temp7               ;1        was rotation possible ? (1/2)
                    sz                            ;1(2)     was rotation possible ? (1/2)
                    dec       angle               ;1        no, retract rotation
                    jmp       donerotate;3        we are done here

nojoy               delay     5
nojoytimeout        delay     433+231+231+3+23-18-(433-4)
norotate  delay     436-4
donerotate
;------------ place block in playfield -------------

                    pcall     makeblock ;205

                    delay     TIME_IMAGE-430-987-23-3-231-231-436-205

;*********************** Line 2 ****************************

                    call      hsync               ;643


;---------------- Find full line ---------- 1032 clocks

                    mov       temp2,#15           ;2        examine 15 lines (all except for the most upper one)
                    mov       fsr,#$9F            ;2        start at bottom left corner of playfield
                    clr       temp4               ;1        clear emptyline memory
fline0              mov       temp1,#4            ;2        set byte counter to 4 bytes (per line)
                    mov       temp0,#8            ;2        set brick counter to 8 bricks (per line)

fline1              mov       w,ind               ;1        get byte from playfield
                    and       w,#$F0              ;1        mask out one nibble
                    sz                            ;1(2)     if nibble is not zero
                    dec       temp0               ;1        then decrease brick counter
                    mov       w,ind               ;1        get byte from playfield
                    and       w,#$0F              ;1        mask out the other nibble
                    sz                            ;1(2)     if nibble is not zero
                    dec       temp0               ;1        then decrease brick counter
                    add       fsr,#$20            ;2        point one byte right
                    decsz     temp1               ;1        decrease byte counter
                    jmp       fline1              ;3        do all (four) bytes

                    mov       w,temp2             ;1        get line number
                    test      temp0               ;1        how many nibbles was not zero ? (1/2)
                    snz                           ;1(2)     was all nibbles non zero then line is full
                    mov       temp4,w             ;1        then remember this as a full line (store line number)
                    add       fsr,#-$81           ;2        point one line up and 4 bytes left
                    decsz     temp2               ;1(2)     decrease line counter
                    jmp       fline0              ;3        do all (15) lines

                    mov       temp1,temp4         ;2        line number of empty line is number of lines to move
                    mov       w,--temp4           ;1        get line number of the line above the one to remove
                    add       fsr,w               ;1        point at that line
                    not       temp4               ;1        temp4 = ~linenumbers
                    add       temp4,#15+1         ;2        temp4 = lines left = 15 + (1 + ~line number) = 15 - line
number
                    test      temp1               ;1        check if there are lines to move
                    snz                           ;1(2)     if no lines to move

;---------------- Remove full line ----------- 731 clocks

                    jmp       nomline             ;3        then don't move any lines

mline0              mov       w,#4                ;1        4 bytes per line
                    mov       temp3,w             ;1        4 bytes on this line to be moved
                    mov       temp5,w             ;1        4 bytes on upper line to move (this will be used later on in
the code)

mline1              mov       w,ind               ;1        get one byte of line
                    inc       fsr                 ;1        get to next line
                    mov       ind,w               ;1        store byte at next line
                    add       fsr,#$1F            ;2        point one byte left and one line up
                    decsz     temp3               ;1(2)     decrease inne loop counter
                    jmp       mline1              ;3        do all four bytes

                    add       fsr,#-$81           ;2        point one line up
                    decsz     temp1               ;1(2)     decrease outer loop counter
                    jmp       mline0              ;3        do all lines

                    mov       w,#$20              ;1        $20 steps between bytes to clear
clrul               clr       ind                 ;1        clear byte
                    add       fsr,w               ;1        next byte
                    decsz     temp5               ;1(2)     decrease byte counter
                    jmp       clrul               ;3        do all 4 bytes

                    pcall     tenpoints           ;56

                    jmp       mlinec              ;3        skip delay

nomline             delay     731-653             ;731-(43*15+3+1+2+2)

mlinec              test      temp4               ;1        are there lines to remove ?
                    snz                           ;1(2)
                    jmp       nodline             ;3

dline0              delay     43-4                ;38       delay 5 to get 9 clocks in loop to get same length as mline1
loop
                    decsz     temp4               ;1(2)     decrease delay counter
                    jmp       dline0              ;3        do all delay lines
                    delay     2                   ;2        2 clocks to fast, compensate
nodline

;--------------- create next block graphics ------------

                    pcall     makenext            ;184

;-------------- remaining delay to fill line ----------- 244 clocks
                    bank      $00                 ;1
                    mov       w,kind              ;1        set color
                    and       w,#%00000111        ;1
                    mov       temp2,w             ;1
                    inc       temp2               ;1

                    pcall     readjoy1            ;13
                    mov       oldjoy,w            ;1

                    pcall     setblock            ;224      put block at current position on playfield
                    bank      $00                 ;1

                    delay     TIME_IMAGE - 731 -1032 - 184 - 244 - 17 - 1

;************************ Lines 3..64 ************************

bfgovr              mov       w,#PRE_LINES  -2 + TOP_LINES  ;1
                    call      emptylines          ;         do empty lines at top outside of screen

                    delay     12-5
                    ITEXT     STR0,STR0_LEN,STR0_BASE,STR0_PHASE
                    delay     TIME_IMAGE - 17-1

                    mov       w,#STRTOCAP_LINES
                    call      emptylines

                    delay     12-6

                    mov       temp5,#11                               ;2
                    mov       temp3,#((gamedata + CAP) & $FF)         ;2
                    mov       temp4,#((gamedata + CAP) >> 8);2
cap_l               call      hsync                                   ;643
                    delay     (CAP_BASE) - (CAP_PHASE) - 9 - 347 - 7
                    mov       temp0,temp3                             ;2
                    mov       temp1,temp4                             ;2
                    mov       temp2,#((PALETTE_BCW + gamedata2) & $FF);2
                    mov       fsr,#VIDEO_BUFFER                       ;2
                    mov       w,#11                                   ;1
                    pcall     setgraphics                             ;11*31+5+1 = 347
                    mov       temp3,temp0                             ;2
                    mov       temp4,temp1                             ;2
                    mov       fsr,#VIDEO_BUFFER                       ;2
                    mov       w,#11                                   ;1
                    pcall     memtovideo                    ;140
                    delay     CAP_SEP - 140 - 2 -1 + (CAP_PHASEDIFF)
                    mov       fsr,#VIDEO_BUFFER                       ;2
                    mov       w,#11                                   ;1
                    pcall     memtovideo                    ;140
                    delay     (TIME_IMAGE) - (CAP_BASE) + (CAP_PHASE) - (CAP_SEP) - 140 - 4 - (CAP_PHASEDIFF)
                    decsz     temp5                                   ;1(2)
                    jmp       cap_l                                   ;3

                    mov       temp7,#16                               ;2
field_l             call      hsync

                    pcall     leftgfx

;163
                    mov       temp0,#$A0          ;2        set temp0 to bottom left corner of playfield
                    sub       temp0,temp7                   ;2        subtract linecounter to get top left
                    mov       temp1,#VIDEO_BUFFER+8         ;2        set temp1 to playfield buffer
                    mov       temp3,#4                      ;2        set field read counter to 4
fieldread_l         mov       fsr,temp0                     ;2        set fsr to playfield pointer
                    mov       temp2,ind                     ;2        read two blocks of playfield
                    mov       fsr,temp1                     ;2        set fsr to video buffer pointer
                    mov       w,temp2                       ;1        get playfield data
                    and       w,#$F                         ;1        mask out left block color
                    call      brickcolortable               ;9        get phase and amplitude of color
                    mov       ind,w                         ;1        store phase and color in video buffer
                    inc       fsr                           ;1        update local pointer to point at next position in
video buffer
                    mov       w,<>temp2           ;1                  get blocks in swapped order
                    and       w,#$F                         ;1        mask out right block color
                    call      brickcolortable               ;9        get phase and amplitude of color
                    mov       ind,w                         ;1        store phase and color in video buffer
                    inc       temp1                         ;1        update real video buffer pointer
                    inc       temp1                         ;1        update real video buffer pointer (again)
                    add       temp0,#$20          ;2        move playfield pointer one step right
                    decsz     temp3                         ;1(2)     decrease field read counter
                    jmp       fieldread_l                   ;3        loop until entire line is read
                    clr       fsr                           ;1        as the fsr was manipulated, reset it back first
page

                    delay     (((BRICK_WIDTH + 2)*8*12)+5+1+ 7+11+11)-163

                    pcall     rightgfx

                    delay     TIME_IMAGE - TIME_RIGHTGFX - TIME_LEFTGFX - (((BRICK_WIDTH + 2)*8*12)+5+1+ 7+11+11) - 2

                    mov       temp6,#BRICK_LINES
linel               call      hsync                         ;643      11

                    pcall     leftgfx
                    delay     11-GAMEFIELD_PHASE
                    bank      $20                           ;1
                    mov       w,temp7                       ;1        get blockline
                    and       w,#%1110                      ;1        dont care about even or odd
                    xor       w,#%1000                      ;1        check if line is 6 or 7
                    sz                                      ;1(2)
                    jmp       nogameover          ;3        if not don't show gameover
                    page      showgameover                  ;1        prepare page for jmp
                    snb       mixedbits.gameoverbit         ;1(2)     check if game is over
                    jmp       showgameover                  ;3        if so, show game over
                    skip                                    ;(2)
nogameoverjmp       nogameover2                   ;3
nogameover2

                    mov       fsr,#VIDEO_BUFFER+8 ;2
                    mov       temp0,#BRICK_WIDTH  ;2
                    mov       temp1,#black                  ;2
                    mov       w,#8                          ;1
                    pcall     outputcol           ;((BRICK_WIDTH + 2)*8*12)+5+1
showgameoverret
                    delay     GAMEFIELD_PHASE
                    pcall     rightgfx

                    delay     TIME_IMAGE - TIME_RIGHTGFX - TIME_LEFTGFX - (((BRICK_WIDTH + 2)*8*12)+5+1+ 7+11+11) - 4


                    decsz     temp6               ;1(2)
                    jmp       linel               ;3
                    delay     2

                    call      hsync
                    pcall     leftgfx

                    delay     (((BRICK_WIDTH + 2)*8*12)+5+1+ 7+11+11)
                    pcall     rightgfx
                    delay     TIME_IMAGE - TIME_RIGHTGFX - TIME_LEFTGFX - (((BRICK_WIDTH + 2)*8*12)+5+1+ 7+11+11)

                    call      hsync
                    pcall     leftgfx
                    delay     (((BRICK_WIDTH + 2)*8*12)+5+1+ 7+11+11)
                    pcall     rightgfx
                    delay     TIME_IMAGE - TIME_RIGHTGFX - TIME_LEFTGFX - (((BRICK_WIDTH + 2)*8*12)+5+1+ 7+11+11) - 4 -
1

                    page      field_l
                    decsz     temp7
                    jmp       field_l
                    delay     2

                    call      hsync

                    pjmp      nextmain

;*********************** outputcol ****************************
;* shows w number of colorfields, each BLOCKWIDTH cycles wide *
;* (this is the thing that shows the colors in the playfield) *
;*                                                            *
;* number of clocks: ((temp0 + 2) * w * 12) + 5               *
;* used tempregs: 0..5                                        *
;*                                                            *
;* input:                                                     *
;*   w = number of fields                                     *
;*   temp0 = fieldlength, must be odd (phase lost when even)  *
;*   temp1 = neutral level                                    *
;*   fsr = pointer to field data (contens is destroyed)       *
;* output:                                                    *
;*   none                                                     *
;*                                                            *
;* local use of tempregs:                                     *
;*   temp2 is used as color loop counter                      *
;*   temp3 is used as field loop counter                      *
;*   temp4 is used as temp storage for intensity calculations *
;*   temp5 is used as temp storage for phase                  *
;*                                                            *
;* comments:                                                  *
;*   This routine is optimized to get as short gaps between   *
;*   fields as possible, these optimizations assumes some     *
;*   limitations of the input data to be able to get the gap  *
;*   down to only 2 color cycles (24 clocks)                  *
;*   Field color is stored as bytes, where each byte has      *
;*   bit 0..2 as phase bit and bit 2..5 is intensity, note    *
;*   that this means that there is and overlap of phase and   *
;*   intensity                                                *
;*   Remaining phase = 7-phase = 7+(-phase) = 7+(/phase+1) =  *
;*   -1+/phase+1 = /phase = phase xor 7 (assuming 3 bit calc) *
;*   this calculation (xor by 7) is done in the inner loop as *
;*   it was the only place where there was free clock cycles  *
;*   so to get a correct result the inner loop needs to be    *
;*   executed an odd number of times, which makes the field-  *
;*   length required to be an odd number or else phase will   *
;*   be lost.                                                 *
;*   The x in the Phase comment field is the phase value read *
;*   it can be values 0..6                                    *
;*                                                            *
;**************************************************************

                                                  ;Clocks   Phase     Comment
outputcol mov       temp3,w                       ;1        9         set field counter
ocolx1              mov       w,ind               ;1        10        get phase and intensity
                    and       w,#7                ;1        11        mask out phase
                    setphase 7          ;3+7-x    0         set phase
                    mov       temp5,w             ;1        10-x      remember phase for later
                    mov       temp2,temp0         ;2        11-x      set color loop counter
ocolx0              vout      black               ;2        1-x       set first half to black level
                    xor       temp5,#7            ;2        3-x       invert all bits in phase
                    mov       w,>>ind             ;1        5-x       get color and phase shifted right one step
                    mov       temp4,w             ;1        6-x       store shifted value in a tempreg
                    mov       w,>>temp4           ;1        7-x       shift value one more step right
                    mov       video,w             ;1        8-x       and set video output to the intensity
                    decsz     temp2               ;1(2)     9-x       decrease color cycle counter
                    jmp       ocolx0              ;3        10-x      and loop until all color cycles are done
                    voutr     temp1               ;2        11-x      set neutral level
                    mov       w,temp5             ;1        1-x       get remaining phase = 7-phase = phase xor 7
                    setphase 7          ;3+x      2-x       set remaining phase to return to original phase
                    inc       fsr                 ;1        5+x-x     point at next field byte
                    decsz     temp3               ;1(2)     6         decrease field cycle counter
                    jmp       ocolx1              ;3        7         and loop until all fields are done
                    retp                          ;3        8         return back home


;************************* leftgfx ***************************
;* handle graphics at left side of playfield                  *
;**************************************************************

leftgfx             delay     LEFTGFX_BASE

                    bank      $20                 ;1        select bank $20 to be able to read linecnt
                    mov       w,<>linecnt         ;1        get line number with swapped nibbles
                    and       w,#$F               ;1        mask out most significant nibble of linecount to get section
number
                    jmp       pc+w                ;3        jump to correct section of 16 lines
                    jmp       textnext_line       ;3        text "NEXT "
                    jmp       nextblock_line      ;3        bricks preview
                    jmp       nextblock_line      ;3        bricks preview
                    jmp       nextblock_line      ;3        bricks preview
                    jmp       black_line;3        black lines between preview and points text
                    jmp       textscore_line      ;3        text "POINTS"
                    jmp       showpoints_line     ;3        display points
                    jmp       black_line;3        black lines at the bottom
                    jmp       black_line;3        black lines at the bottom
                    jmp       black_line;3        black lines at the bottom
                    jmp       black_line;3        black lines at the bottom
                    jmp       black_line;3        black lines at the bottom
                    jmp       black_line;3        black lines at the bottom

black_linedelay     TIME_LEFTGFX - LEFTSCREW_PHASE - LEFTGFX_BASE - 9 - 4 - 3 - 3 - 104

showscrew mov       fsr,#VIDEO_BUFFER             ;2
                    mov       w,#8                ;1
                    pcall     memtovideo;104      output left screw graphics
                    delay     LEFTSCREW_PHASE
                    retp


;************************ rightgfx ***************************
;* handle graphics at right side of playfield                 *
;**************************************************************

rightgfx  delay     11-RIGHTSCREW_PHASE
                    mov       fsr,#VIDEO_BUFFER   ;2
                    mov       w,#8                ;1
                    pcall     memtovideo;104      output right screw graphics
                    bank      $20                 ;1
                    inc       linecnt             ;1        update linecounter

                    mov       temp1,#((gamedata + SCREW) >> 8)        ;2        set page of graphics
                    mov       w,<>linecnt         ;1        get linenumber with swapped nibbles (multiplied by 16)
                    and       w,#$70              ;1        reboove unwanted bits to get (line%8)*16
                    mov       temp0,w             ;1        store in temp0
                    clc                           ;1        clear carry
                    rr        temp0               ;1        rotate right to get (line%8)*8
                    add       temp0,#((gamedata + SCREW) & $FF)       ;2        add graphics base
                    snc                           ;1(2)     check for page overflow
                    inc       temp1               ;1        point at next page
                    mov       temp2,#((PALETTE_BCW + gamedata2) & $FF) ;2       point at correct palette
                    mov       fsr,#VIDEO_BUFFER   ;2        point at video buffer position where to store graphics
                    mov       w,#8                                    ;1        graphics is 8 pixels wide
                    pcall     setgraphics         ;254      translate graphics into

                    delay     TIME_RIGHTGFX + RIGHTSCREW_PHASE -11 - 104 - 21 - 254 - 4 - 3
                    retp                          ;3


;************************* strout *****************************
;* output characters from string in rom using a font in rom   *
;* temp0 used as character temp storage                       *
;* temp2 used as character counter                            *
;* temp1:temp3 = pointer to string                            *
;* temp4 = line (0..7) + FONT_BASE                            *
;* temp5 = length                                             *
;* clocks: 8 * 12 * w + 44 + 1                                 *
;**************************************************************

strout_cl
```

```
strout_l  vout      black                         ;2        pixel three to seven
                    delay     2                   ;2
                    rr        temp0               ;1
                    snc                           ;1(2)
                    mov       w,#53               ;1
                    mov       video,w
                    decsz     temp2               ;1(2)
                    jmp       strout_l            ;3

strout              mov       m,temp1             ;2        set character page

                    vout      black               ;2        pixel one starts here

                    mov       w,temp3             ;1        get pointer to characters
                    iread                         ;4        read one character
                    add       w,temp4             ;1        update according to line and fontbase
                    mov       m,#((gamedata + FONT) >> 8)   ;1        set font-page
                    iread                         ;4        read character pixels from font
                    mov       temp0,w             ;1        store character pixels in temp0

                    mov       w,#black            ;1
                    rr        temp0               ;1
                    snc                           ;1(2)
                    mov       w,#53               ;1
                    inc       temp3               ;1        point at next character
                    mov       video,w             ;1

                    snz                           ;1(2)
                    inc       temp1               ;1
                    delay     2
                    vout      black               ;2        pixel three starts here

                    mov       temp2,#5            ;2
                    rr        temp0               ;1
                    snc                           ;1(2)
                    mov       w,#53               ;1
                    mov       video,w             ;1
                    decsz     temp5               ;1(2)
                    jmp       strout_cl           ;3
                    vout      black
                    retp                          ;3


;************************ incpoints ***************************
;* add one point to score                                     *
;* clocks: 59+1                                               *
;**************************************************************

incpoints mov       fsr,#SCORE+3                  ;2
                    mov       temp0,#4            ;2

incpoints_l         inc       ind                 ;1
                    mov       w,#%1010            ;1
                    xor       w,ind               ;1
                    sz                            ;1(2)
                    jmp       nocarry_l           ;3
                    clr       ind
                    dec       fsr                 ;1
                    decsz     temp0               ;1(2)
                    jmp       incpoints_l         ;3
                    delay     7                   ;7
                    retp                          ;3

nocarry_l delay     7
                    decsz     temp0               ;1(2)
                    jmp       nocarry_l           ;3
                    retp


;************************ tenpoints ***************************
;* add ten points to score                                    *
;* note: this routine requires incpoints                      *
;* clocks: 56                                                 *
;**************************************************************

tenpoints mov       fsr,#SCORE+2                            ;2
                    mov       temp0,#3                      ;2
                    jmp       incpoints_l                   ;3


;************************ textlines ***************************
;* routine to output line with text in leftgraphics field     *
;* clocks: 537                                                *
;**************************************************************

textlines mov       w,linecnt                               ;1        get line number
                    and       w,#%1100                      ;1(2)     check bit 2,3
                    sz                                      ;1        if bits are zero, don't do next test
                    xor       w,#%1100                      ;1        toggle bit 2,3 to check if values are %11
                    snz                                     ;1(2)     if bits are %00 or %11
                    jmp       notext                        ;3        then lines should be empty

                    mov       temp4,#((gamedata + FONT) & $ff) - 4    ;2        set temp4 to fontbase - 4 (compensating
line starting at 4)
                    mov       w,linecnt                     ;1        get linecounter
                    and       w,#$F                         ;1        get last significant nibble (sectionline)
                    add       temp4,w                       ;1        update pointer according to line number
                    mov       temp5,#6                      ;2        always output 6 characters
                    jmp       strout                        ;525      output text


notext              delay     526                           ;523      delay to keep timing correct if no text
                    ret                                     ;3        get back to left graphics and show screw


;************************ makenext ****************************
;* create graphics for next block to be used in leftgraphics  *
;* clocks: 183 + 1                                            *
;**************************************************************

xsel                and       w,#3                          ;1 remove unwanted bits
                    jmp       pc+w                          ;3 select correct returnvalue
                    retw      2                             ;3 bit 1
                    retw      4                             ;3 bit 2
                    retw      8                             ;3 bit 3
                    retw      1                             ;3 bit 0

ysel                and       w,#3
                    jmp       pc+w
                    retw      (NEXTGFX   & $7F) | $80
                    retw      (NEXTGFX+1 & $7F)
                    retw      (NEXTGFX+1 & $7F) | $80
                    retw      (NEXTGFX   & $7F)

makenext  bank      NEXTGFX                                 ;1
                    clr       NEXTGFX                       ;1
                    clr       NEXTGFX+1                     ;1
                    bank      $00                           ;1
                    clc                                     ;1
                    mov       w,<<nextkind                  ;1
                    pcall     blocks                        ;10
                    mov       temp0,w                       ;1
                    mov       w,<<nextkind                  ;1
                    or        w,#1                          ;1
                    call      blocks                        ;9
                    page      makenext_l          ;1
                    mov       temp1,w                       ;1
                    mov       temp3,#4                      ;2

makenext_lmov       w,temp0                       ;1
                    call      xsel                          ;10
                    mov       temp2,w                       ;1
                    mov       w,temp1                       ;1
                    call      ysel                          ;10
                    mov       fsr,w                         ;1
                    snb       fsr.7                         ;1(2)
                    swap      temp2                         ;1
                    clrb      fsr.7                         ;1
                    or        ind,temp2                     ;2
                    rr        temp0                         ;1
                    rr        temp0                         ;1
                    rr        temp1                         ;1
                    rr        temp1                         ;1
                    decsz     temp3                         ;1(2)
                    jmp       makenext_l          ;3
                    retp                                    ;3

;********************** textnext_line ************************
;* handles field with text "next" in leftgraphics             *
;**************************************************************

textnext_line       delay     TEXTNEXT_BASE-TEXTNEXT_PHASE            ;         delay to set phase of text
                    mov       temp3,#((STR2 + gamedata) & $FF)        ;2        set lower pointer to "Next" text
                    mov       temp1,#((STR2 + gamedata) >> 8)         ;2        set upper pointer to "Next" text
                    call      textlines                               ;537      output 6 charcters to left field
                    delay     TIME_LEFTGFX  -  LEFTSCREW_PHASE  -  LEFTGFX_BASE  -  9  -  4  -  3  -  3  -  104  -  3  -  544  -
TEXTNEXT_BASE + TEXTNEXT_PHASE
                    jmp showscrew                                     ;3        get back to left graphics and show screw


;********************** textscore_line ***********************
;* handles field with text "score" in leftgraphics            *
;**************************************************************

textscore_line      delay     TEXTSCORE_BASE-TEXTSCORE_PHASE;         delay to set phase of text
                    mov       temp3,#((STR3 + gamedata) & $FF)        ;2        set lower pointer to "Score" text
                    mov       temp1,#((STR3 + gamedata) >> 8)         ;2        set upper pointer to "Score" text
                    call      textlines                               ;537      output 6 charcters to left field
                    delay     TIME_LEFTGFX  -  LEFTSCREW_PHASE  -  LEFTGFX_BASE  -  9  -  4  -  3  -  3  -  104  -  3  -  544  -
TEXTSCORE_BASE + TEXTSCORE_PHASE
                    jmp showscrew                                     ;3        get back to left graphics and show screw


;********************** showpoints_line **********************
;* handles field with score in leftgraphics                   *
;**************************************************************

showpoints_line     delay     SCORE_BASE-SCORE_PHASE

                    mov       w,linecnt                               ;1        get line number
                    and       w,#$0F                                  ;1        get      last      significant      nibble
(sectionline)
                    mov       temp4,w                                 ;1        store sectionline in text line register
(temp4)

                    mov       temp1,#4                                ;2        set character counter
                    mov       fsr,#SCORE                    ;2        point att score
                    add       temp4,#((gamedata + NUMBERS) & $FF)     ;2        add pointer to font to linenumber

stroutp_clmov       m,#((gamedata + NUMBERS) >> 8);1        set font-page
                    mov       w,<>ind                                 ;1        get digit x 16
                    inc       fsr                                     ;1        point at next digit
                    add       w,temp4                                 ;1        point at correct line in font (according
to current sectionline)
                    snc                                               ;1(2)     detect page overflow
                    mov       m,#(((gamedata + NUMBERS) >> 8)+1)      ;1        if overflow set next font-page
                    iread                                             ;4        read character pixels from font
                    mov       temp0,w                                 ;1        store character pixels in temp0
                    mov       temp2,#8                                ;2        each character is 7 pixels wide
                    delay     7                                       ;7        delay to keep phase

stroutp_l vout      black                                             ;2        set video output to black level
                    delay     2                                       ;2        delay to make colorcycle 12 clocks
                    rr        temp0                                   ;1        rotate font data
                    snc                                     ;1(2)     check  if  bit  set,  if  not  keeep  black
leevel in w
                    mov       w,#53                         ;1        else if bit set set gigit intensity
                    mov       video,w                       ;1        output selected level to video output
                    decsz     temp2                         ;1(2)     decrease pixel counter
                    jmp       stroutp_l                     ;3        loop until all pixels are done
                    vout      black                         ;2        set video level to black

                    decsz     temp1                         ;1(2)     decreasee digit counter
                    jmp       stroutp_cl          ;3        loop until all digs are done

                    delay     TIME_LEFTGFX  -  LEFTSCREW_PHASE  -  LEFTGFX_BASE  -  9  -  4  -  3  -  3  -  104  -  3  -  487  -
SCORE_BASE+SCORE_PHASE
                    jmp showscrew                           ;3        get back to left graphics and show screw


;********************** nextblock_line ***********************
;* handles fields showing next block                          *
;**************************************************************

nextblock_line      delay     NBLOCK_BASE-NBLOCK_PHASE
                    mov       w,gfxcnt                      ;1        get gfxccounter
                    and       w,#$0C                        ;1        check  if  line  is  0..2  of  the  12-line
brick
                    snz                                     ;1(2)     if not, get on with the brick drawing
                    jmp       nonextg                       ;3        if  one  of  the  first  lines,  then  it
should be black

                    mov       temp0,gfxcnt                  ;2
                    bank      NEXTGFX                       ;1

                    mov       w,NEXTGFX                     ;1        get graphics for next block
                    snb       temp0.5                       ;1(2)     check if brickline second half
                    mov       w,NEXTGFX+1                   ;1        yes, it was, get next next graphics
                    mov       temp1,w             ;1        store in temp1
                    snb       temp0.4             ;1(2)     check if blickline is odd
                    swap      temp1               ;1        yes, swap nibbles

                    bank      $00                 ;1        set bank for next kind
                    mov       w,++nextkind        ;1        get next kind of block
                    and       w,#%00001111        ;1
                    pcall     brickcolortable     ;10       translate kind to phase and amplitude
                    page      nextl0              ;1        brickcolortabe destroy page register, restore it
                    mov       temp2,w             ;1        store phase and amplitude in temp2
                    mov       temp3,#4            ;2        we have 4 brick positions to convert
                    mov       fsr,#VIDEO_BUFFER + $28       ;2        point at beginning of buffer

nextl0              mov       w,temp2             ;1        get phase and amplitude of next block
                    rr        temp1               ;1        rotate block data to get next bit
                    sc                            ;1(2)     check if bit was set
                    mov       w,#(black)<<2       ;1        if not set black intensity
                    mov       ind,w               ;1        store phase and color in buffer
                    inc       fsr                 ;1        point at next buffer position
                    decsz     temp3               ;1(2)     decrease brick counter
                    jmp       nextl0              ;3        keep looping until all bricks are done

                    mov       fsr,#VIDEO_BUFFER + $28       ;2        point at beginning of buffer
                    mov       temp0,#7            ;2        each brick is 7 cycles
                    mov       temp1,#black        ;2        set black level (between bricks)
                    mov       w,#4                ;1        we have 4 brick positions
                    pcall     outputcol           ;438      output colors to video output
                    jmp       donenextg           ;3

nonextg             delay     512                 ;508      empty line, delay to keep timing

donenextg bank      $20                           ;1
                    inc       gfxcnt              ;1        increase counter
                    mov       w,gfxcnt            ;1        get counter value
                    and       w,#3                ;1        check bit 0 and bit 1
                    sz                            ;1(2)     if not zero upper part was not increased
                    setb      gfxcnt.0            ;1        if zero, jump to one by setting bit 0

                    delay     TIME_LEFTGFX - LEFTSCREW_PHASE - LEFTGFX_BASE - 9 - 4 - 3 - 3 - 104 - 3         -   524   -
NBLOCK_BASE+NBLOCK_PHASE      ;         delay to make points section 551 cycles
                    jmp showscrew                                     ;3        get back to left graphics and
show screw


;*********************** showgameover ************************
;* show "GAME OVER" text in playfield                          *
;**************************************************************

showgameover        delay     GAMEOVER_BASE-GAMEOVER_PHASE            ;
                    mov       temp3,#((STR5 + gamedata) & $FF)        ;2        set lower pointer to "Game" text
                    mov       temp1,#((STR4 + gamedata) >> 8)         ;2        set upper pointer to "Over" text
                    mov       w,#((STR4 + gamedata) & $FF)            ;1        get lower address to "Over"
                    snb       temp7.0                       ;1(2)     check if section is odd
                    mov       temp3,w                       ;1        odd  line,  set  lower  pointer  to  "Over"
text
                    mov       temp4,#((gamedata + FONT -1 +8) & $ff)  ;2        set temp4 to fontbase - 4 (compensating
line starting at 4)
                    mov       w,temp6                       ;1        get linecounter
                    and       w,#7                          ;1        get      last      significant      nibble
(sectionline)
                    snb       temp6.3                       ;1(2)     check if linenr 1arger than 7
                    jmp       emptygoverl                   ;3        if found, skip it

                    sub       temp4,w                       ;1        update pointer according to line number
                    mov       temp5,#5                      ;1        always output 4 characters
                    call      strout                        ;429      output text
                    delay     ((BRICK_WIDTH + 2)*8*12)+35-444-26+GAMEOVER_PHASE-GAMEOVER_BASE
                    pjmp      showgameoverret               ;4        get back to mainloop

emptygoverl         delay     ((BRICK_WIDTH + 2)*8*12)+35-15-26+GAMEOVER_PHASE-GAMEOVER_BASE
                    pjmp      showgameoverret               ;4        get back to mainloop

gameover  pcall     hsync
                    bank      $00
                    snb       joy1button                    ;1(2)     check if joystick buttton pressed
                    jmp       nogovb                                  ;3         if not, do nothing
                    page      start                                   ;1
                    sb        joy1button_old                          ;1(2)
                    jmp       start                                   ;3
nogovb
                    pcall     readjoy1                                ;13
                    mov       oldjoy,w                                ;1

                    delay     TIME_IMAGE-1-20
                    pcall     hsync
                    delay     TIME_IMAGE-1-4
                    pjmp      bfgovr


nextmain  delay     12-BLINE_PHASE
                    mov       w,#(TIME_IMAGE-1-12-1-12-1-4) / 12
                    pcall     simplecolorfa

                    delay     ((TIME_IMAGE-1-12-1-12-1-4) // 12) + BLINE_PHASE
                    pcall     hsync

                    delay     (TIME_IMAGE - 5)

                    ITEXT     STR1,STR1_LEN,STR1_BASE,STR1_PHASE

                    delay     (TIME_IMAGE-1)

                    pcall     hsync

                    delay     4+12-BLINE_PHASE
                    mov       w,#(TIME_IMAGE-1-12-4-1-18-1) / 12
                    pcall     simplecolorfa
                    delay     ((TIME_IMAGE-1-12-4-1-18-1) // 12) + BLINE_PHASE

                    mov       w,#POST_LINES + VISILINES - STRTOCAP_LINES - PLAYFIELD_LINES - 11 - 1 - 9 -1 - 10
                    pcall     emptylines

                    bank      $20
                    inc       rnd

                    pjmp      updatemusic                   ;131

;************************** gamedata ***************************
;* Game data: graphics, music, wavetables etc...              *
;***************************************************************
;

ORG       $600

gamedata
dw $000,$000,$200,$300,$300,$400,$300,$000,$000,$000,$200,$47e,$400,$500,$400,$100        ; $000..$00f
dw $000,$200,$400,$700,$800,$a18,$418,$200,$21c,$436,$963,$c7f,$963,$863,$463,$200        ; $010..$01f
dw $27f,$446,$c16,$91e,$616,$646,$47f,$200,$23c,$466,$903,$603,$673,$466,$25c,$100        ; $020..$02f
dw $263,$477,$67f,$67f,$46b,$263,$163,$000,$063,$267,$46f,$47b,$373,$363,$163,$000        ; $030..$03f
dw $03e,$063,$063,$063,$063,$263,$03e,$000,$03f,$066,$066,$03e,$036,$066,$067,$200        ; $040..$04f
dw $43c,$266,$00c,$018,$030,$066,$03c,$000,$07e,$27e,$45a,$a18,$318,$218,$03c,$000        ; $050..$05f
dw $063,$063,$063,$263,$463,$a36,$b1c,$a00,$363,$263,$036,$01c,$036,$263,$463,$a00        ; $060..$06f
dw $c00,$a00,$61e,$430,$33e,$233,$06e,$000,$200,$400,$c3e,$a63,$603,$663,$43e,$300        ; $070..$07f
dw $238,$030,$23e,$433,$a33,$c33,$a6e,$600,$600,$500,$43e,$263,$17f,$203,$43e,$900        ; $080..$08f
dw $a00,$600,$66e,$533,$533,$43e,$230,$11f,$218,$400,$51c,$518,$518,$518,$53c,$400        ; $090..$09f
dw $407,$206,$166,$036,$21e,$536,$567,$400,$400,$400,$437,$27f,$16b,$06b,$06b,$000        ; $0a0..$0af
dw $200,$200,$23b,$266,$266,$166,$166,$000,$000,$000,$13e,$363,$463,$563,$63e,$600        ; $0b0..$0bf
dw $700,$700,$73b,$66e,$606,$506,$40f,$300,$100,$000,$f7e,$d03,$c3e,$b60,$a3f,$a00        ; $0c0..$0cf
dw $90c,$90c,$93f,$a0c,$a0c,$b6c,$c38,$d00,$f00,$100,$833,$033,$133,$933,$06e,$a00        ; $0d0..$0df
dw $900,$000,$d63,$a6b,$06b,$27f,$c36,$000,$a00,$d00,$063,$736,$e1c,$036,$363,$000        ; $0e0..$0ef
dw $130,$018,$03e,$063,$07f,$003,$03e,$000,$03e,$041,$059,$045,$045,$059,$041,$03e        ; $0f0..$0ff
dw $050,$068,$008,$058,$088,$0d0,$0c0,$098,$0c8,$000,$0f8,$000,$048,$010,$028,$0d8        ; $100..$10f
dw $0b0,$0f0,$088,$000,$0e0,$0e0,$0e0,$0e0,$010,$0c0,$098,$078,$0a0,$070,$0c0,$080        ; $110..$11f
dw $010,$090,$0d8,$0b0,$088,$088,$010,$078,$0b8,$0a8,$000,$038,$138,$088,$0e8,$0d0        ; $120..$12f
dw $000,$000,$050,$050,$078,$0b8,$0c0,$088,$000,$028,$028,$318,$030,$020,$400,$040        ; $130..$13f
dw $040,$060,$020,$048,$600,$000,$03c,$07e,$be7,$0e7,$0e7,$de7,$0e7,$0e7,$0e7,$0e7        ; $140..$14f
dw $0e7,$0e7,$0e7,$ee7,$07e,$13c,$160,$078,$07e,$37e,$070,$070,$570,$070,$070,$070        ; $150..$15f
dw $a70,$170,$b70,$d70,$170,$070,$038,$07c,$0e6,$4e6,$1e6,$1e6,$0f0,$270,$070,$378        ; $160..$16f
dw $138,$23c,$01c,$10e,$0fe,$0fe,$23e,$27f,$0e7,$4e7,$1e7,$3e0,$07c,$2fc,$0e0,$1e7        ; $170..$17f
dw $2e7,$2e7,$0e7,$3e7,$17f,$43e,$1f8,$2f8,$1f8,$0fc,$1fc,$0ec,$1ee,$5ee,$2e6,$6e7        ; $180..$18f
dw $0ff,$7ff,$1ff,$6e0,$0e0,$5e0,$0ff,$4ff,$207,$207,$077,$4ff,$1e7,$3e7,$0e0,$2e0        ; $190..$19f
dw $0e7,$1e7,$2e7,$2e7,$07e,$33c,$13c,$47e,$1e7,$2e7,$107,$07f,$1ff,$0e7,$1e7,$5e7        ; $1a0..$1af
dw $2e7,$6e7,$0e7,$7e7,$17e,$63c,$0fe,$5fe,$0e0,$4e0,$2e0,$260,$070,$470,$170,$330        ; $1b0..$1bf
dw $038,$238,$038,$138,$21c,$21c,$03c,$37e,$1e7,$4e7,$1e7,$2e7,$1e7,$07e,$1ff,$0e7        ; $1c0..$1cf
dw $1e7,$0e7,$0e7,$0e7,$07e,$03c,$03c,$07e,$0e7,$0e7,$0e7,$0e7,$0e7,$0e7,$0e7,$0ff        ; $1d0..$1df
dw $0fe,$0e0,$0e7,$0e7,$07e,$03c                                                          ; $1e0..$1e5
```

Appendix D: Pong source code

```
;*******************************************************************************
;* SX-PONG (C) Rickard Gunée, 2001                                             *
;*******************************************************************************
;* This is the classical videogame pong, outputing a color video signal in     *
;* software using a couple of resistors.                                       *
;* The video signal is not 100% correct, it will not work on all TV:s, so if   *
;* your TV can't lock on the color signal or you get strange colors on the     *
;* screen then your TV probably can't run this game.                           *
;* This is an open source project and you may use this design and software     *
;* anyway you like as long as it is non comercial and you refer to the         *
;* original author with name and link to homepage.                             *
;* Use this at your own risk, don't blame me if you blow up your tv or kill    *
;* yourself or anyone else with it.                                            *
;*                                                                             *
;* For more info about project go to: http://www.rickard.gunee.com/projects    *
;*******************************************************************************

                    DEVICE    SX28,TURBO,STACKX_OPTIONX

                    RESET     jumpstart           ;goto start on reset
                    NOEXPAND

                    SYSTEM_PAL= 1
                    SYSTEM_PAL_N        = 2
                    SYSTEM_PAL_M        = 3
                    SYSTEM_NTSC         = 4

                    SYSTEM = SYSTEM_PAL ;*** This line selects TV-system timing to use ***




                    delaytimer1         equ       08h
                    delaytimer2         equ       09h
                    temp0               equ       08h
                    temp1               equ       09h
                    temp2               equ       0Ah
                    temp3               equ       0Bh
                    temp4               equ       0Ch
                    temp5               equ       0Dh
                    temp6               equ       0Eh
                    temp7               equ       0Fh

                    joy                 equ       RC

                    joy1up              equ       RB.7
                    joy1down  equ       RC.5
                    joy1left  equ       RC.6
                    joy1right equ       RC.7
                    joy1buttonequ       RB.6

                    joy2up              equ       RA.2
                    joy2down  equ       RA.3
                    joy2left  equ       RA.0
                    joy2right equ       RA.1
                    joy2buttonequ       RC.7

                    y1                  equ       $10
                    y2                  equ       $11
                    mixedbits equ       $12

                    ballx               equ       $13
                    ballx_l             equ       $13
                    ballx_h             equ       $14
                    gamekind  equ       $13

                    bally               equ       $15
                    bally_l             equ       $15
                    bally_h             equ       $16


                    ballx_speed         equ       $17
                    ballx_speed_l       equ       $17
                    ballx_speed_h       equ       $18

                    bally_speed         equ       $19
                    bally_speed_l       equ       $19
                    bally_speed_h       equ       $1A

                    p1                  equ       $1B
                    p2                  equ       $1C
                    state               equ       $1D

                    oldj1               equ       $1E
                    oldj2               equ       $1F

                    soundtemp0equ       $10
                    soundtemp1equ       $11

                    wave1pos  equ       $12
                    wave1pos_lequ       $12
                    wave1pos_hequ       $13
                    wave1speedequ       $14
                    wave1speed_l        equ       $14
                    wave1speed_h        equ       $15
                    wave1speeddif       equ       $16
```

```
wave1speeddif_l     equ       $16
wave1speeddif_h     equ       $17
wave1timerequ       $18

wave2pos  equ       $19
wave2pos_lequ       $19
wave2pos_hequ       $1A
wave2speedequ       $1B
wave2speed_l        equ       $1B
wave2speed_h        equ       $1C
wave2speeddif       equ       $1D
wave2speeddif_l     equ       $1D
wave2speeddif_h     equ       $1E
wave2timerequ       $1F

black               equ       14
neutral             equ       14


VIDEO_BUFFER        equ       $F0
JTIME               equ       10


frame               equ       0
line                equ       1
gameoverbit         equ       2

video               equ       RB
audio               equ       RC

joy1up_oldequ       oldj2.7
joy1down_old        equ       oldj1.5
joy1left_old        equ       oldj1.6
joy1right_old       equ       oldj1.7
joy1button_old      equ       oldj2.6

joy2up_oldequ       oldj1.2
joy2down_old        equ       oldj1.3
joy2left_old        equ       oldj1.0
joy2right_old       equ       oldj1.1
joy2button_old      equ       oldj1.7


IF (SYSTEM = SYSTEM_PAL)

          FREQ      53156550

          TIME_2US4           EQU       128
          TIME_4US5           EQU       239
          TIME_27US5EQU       1463
          TIME_29US6EQU       1574
          TIME_64US           EQU       3405
          TIME_TOTALEQU       TIME_64US
          TIME_PRESYNC        EQU       89
          TIME_SYNC           EQU       250
          TIME_PREBURST       EQU       48
          TIME_BURSTEQU       144
          TIME_POSTBURST      EQU       112

          LEFT_SPACEEQU       120
          RIGHT_SPACE         EQU       144
          TOT_LINES           EQU       304
          PRE_LINES           EQU       35
          POST_LINESEQU       19

          LEFTPAD_PHASE       EQU       1
          RIGHTPAD_PHASE      EQU       10
          BALL_PHASEEQU       4
          LEFTSCORE_BASE      EQU       (12*20)
          LEFTSCORE_PHASE     EQU       9
          RIGHTSCORE_BASE     EQU       (12*18)
          RIGHTSCORE_PHASE    EQU       4

          TTEXT_BASE          EQU       (12*87)
          TTEXT_PHASE         EQU       8
          BTEXT_BASEEQU       (12*30)
          BTEXT_PHASE         EQU       8
          TEXTLINE_PHASE      EQU       6
          PADMID_PHASE        EQU       1
          PADEND_PHASE        EQU       7
          WINTEXT_BASE        EQU       (12*65)
          WINTEXT_PHASE       EQU       7
          INITTEXT1_BASE      EQU       (12*92)
          INITTEXT1_PHASE     EQU       11
          INITTEXT2_BASE      EQU       (12*56)
          INITTEXT2_PHASE     EQU       11
          INITTEXT3_BASE      EQU       (12*57)
          INITTEXT3_PHASE     EQU       5
          INITTEXT4_BASE      EQU       (12*57)
          INITTEXT4_PHASE     EQU       5
          INITTEXT5_BASE      EQU       (12*28)
          INITTEXT5_PHASE     EQU       8
ENDIF

IF (SYSTEM = SYSTEM_PAL_M)

          FREQ      42907332

          TIME_2US4           EQU       103
          TIME_4US5           EQU       193
          TIME_27US5EQU       1181
          TIME_29US6EQU       1271
          TIME_64US           EQU       2749

          TIME_TOTALEQU       TIME_64US
          TIME_PRESYNC        EQU       47
          TIME_SYNC           EQU       202
          TIME_PREBURST       EQU       39
          TIME_BURSTEQU       144
          TIME_POSTBURST      EQU       5

          TOT_LINES           EQU       254
          PRE_LINES           EQU       35
          POST_LINESEQU       19

          LEFT_SPACEEQU       (12*12)
          RIGHT_SPACE         EQU       (12*7)

          LEFTPAD_PHASE       EQU       1
          RIGHTPAD_PHASE      EQU       10
          BALL_PHASEEQU       4
          LEFTSCORE_BASE      EQU       (12*20)
          LEFTSCORE_PHASE     EQU       9
          RIGHTSCORE_BASE     EQU       (12*15)
          RIGHTSCORE_PHASE    EQU       4

          TTEXT_BASEEQU       (12*65)
          TTEXT_PHASE         EQU       8
          BTEXT_BASEEQU       (12*13)
          BTEXT_PHASE         EQU       8
          TEXTLINE_PHASE      EQU       6
          PADMID_PHASE        EQU       1
          PADEND_PHASE        EQU       7
          WINTEXT_BASE        EQU       (12*48)
          WINTEXT_PHASE       EQU       7
          INITTEXT1_BASE      EQU       (12*75)
          INITTEXT1_PHASE     EQU       11
          INITTEXT2_BASE      EQU       (12*40)
          INITTEXT2_PHASE     EQU       11
          INITTEXT3_BASE      EQU       (12*40)
          INITTEXT3_PHASE     EQU       5
          INITTEXT4_BASE      EQU       (12*40)
          INITTEXT4_PHASE     EQU       5
          INITTEXT5_BASE      EQU       (12*13)
          INITTEXT5_PHASE     EQU       8

ENDIF


IF (SYSTEM = SYSTEM_PAL_N)

          FREQ      42984672

          TIME_2US4           EQU       103
          TIME_4US5           EQU       193
          TIME_27US5EQU       1181
          TIME_29US6EQU       1271
          TIME_64US           EQU       2749

          TIME_TOTALEQU       TIME_64US
          TIME_PRESYNC        EQU       47
          TIME_SYNC           EQU       202
          TIME_PREBURST       EQU       39
          TIME_BURSTEQU       144
          TIME_POSTBURST      EQU       5

          TOT_LINES           EQU       304
          PRE_LINES           EQU       35
          POST_LINESEQU       19

          LEFT_SPACEEQU       (12*12)
          RIGHT_SPACE         EQU       (12*7)

          LEFTPAD_PHASE       EQU       1
          RIGHTPAD_PHASE      EQU       10
          BALL_PHASEEQU       4
          LEFTSCORE_BASE      EQU       (12*20)
          LEFTSCORE_PHASE     EQU       9
          RIGHTSCORE_BASE     EQU       (12*15)
          RIGHTSCORE_PHASE    EQU       4

          TTEXT_BASEEQU       (12*65)
          TTEXT_PHASE         EQU       8
          BTEXT_BASEEQU       (12*13)
          BTEXT_PHASE         EQU       8
          TEXTLINE_PHASE      EQU       6
          PADMID_PHASE        EQU       1
          PADEND_PHASE        EQU       7
          WINTEXT_BASE        EQU       (12*48)
          WINTEXT_PHASE       EQU       7
          INITTEXT1_BASE      EQU       (12*75)
          INITTEXT1_PHASE     EQU       11
          INITTEXT2_BASE      EQU       (12*40)
          INITTEXT2_PHASE     EQU       11
          INITTEXT3_BASE      EQU       (12*40)
          INITTEXT3_PHASE     EQU       5
          INITTEXT4_BASE      EQU       (12*40)
          INITTEXT4_PHASE     EQU       5
          INITTEXT5_BASE      EQU       (12*13)
          INITTEXT5_PHASE     EQU       8



ENDIF


IF (SYSTEM = SYSTEM_NTSC)

          FREQ      42954540

          TIME_2US4           EQU       103
          TIME_4US5           EQU       193
          TIME_27US5EQU       1181
          TIME_29US6EQU       1271
          TIME_64US           EQU       2748
```

```
                              TIME_TOTALEQU       TIME_64US
                              TIME_PRESYNC        EQU       47
                              TIME_SYNC           EQU       202
                              TIME_PREBURST       EQU       39
                              TIME_BURSTEQU       144
                              TIME_POSTBURST      EQU       5

                              TOT_LINES           EQU       254
                              PRE_LINES           EQU       30
                              POST_LINESEQU       16

                              LEFT_SPACEEQU       (12*12)
                              RIGHT_SPACE         EQU       (12*7)

                              LEFTPAD_PHASE       EQU       1
                              RIGHTPAD_PHASE      EQU       10
                              BALL_PHASEEQU       4
                              LEFTSCORE_BASE      EQU       (12*20)
                              LEFTSCORE_PHASE     EQU       9
                              RIGHTSCORE_BASE     EQU       (12*15)
                              RIGHTSCORE_PHASE    EQU       4

                              TTEXT_BASEEQU       (12*65)
                              TTEXT_PHASE         EQU       8
                              BTEXT_BASEEQU       (12*13)
                              BTEXT_PHASE         EQU       8
                              TEXTLINE_PHASE      EQU       6
                              PADMID_PHASE        EQU       1
                              PADEND_PHASE        EQU       7
                              WINTEXT_BASE        EQU       (12*48)
                              WINTEXT_PHASE       EQU       7
                              INITTEXT1_BASE      EQU       (12*75)
                              INITTEXT1_PHASE     EQU       11
                              INITTEXT2_BASE      EQU       (12*40)
                              INITTEXT2_PHASE     EQU       11
                              INITTEXT3_BASE      EQU       (12*40)
                              INITTEXT3_PHASE     EQU       5
                              INITTEXT4_BASE      EQU       (12*40)
                              INITTEXT4_PHASE     EQU       5
                              INITTEXT5_BASE      EQU       (12*13)
                              INITTEXT5_PHASE     EQU       8
                    ENDIF


                    TIME_HSYNCEQU       (TIME_PRESYNC + TIME_SYNC + TIME_PREBURST + TIME_BURST + TIME_POSTBURST)
                    TIME_IMAGEEQU       (TIME_TOTAL - TIME_HSYNC)

                    BALLAREA_WIDTH      EQU       ((TIME_IMAGE - (119+119+167 + RIGHT_SPACE + LEFT_SPACE + 12 + 2 + 4 + 
9 + 1))/12)

                    MID_LINES           EQU       (TOT_LINES - PRE_LINES - POST_LINES)
                    PLAYFIELD_LINES     EQU       (MID_LINES-22)
                    PAD_ENDSIZE         EQU       5
                    PAD_MIDSIZE         EQU       (PLAYFIELD_LINES / 5)
                    PAD_SIZE            EQU       (PAD_MIDSIZE + (PAD_ENDSIZE*2))
                    BALL_LINESEQU       16

                    BALL_BUFFER         EQU       $D0
                    RPAD_BUFFER         EQU       $90
                    LPAD_BUFFER         EQU       $50

                    WAVEPAD_FREQ        EQU       256
                    WAVEPAD_DIF         EQU       12
                    WAVEPAD_LEN         EQU       20

                    WAVEWALL_FREQ       EQU       196
                    WAVEWALL_DIF        EQU       8
                    WAVEWALL_LEN        EQU       24

                    WAVEMISS_FREQ       EQU       256
                    WAVEMISS_DIF        EQU       0
                    WAVEMISS_LEN        EQU       150

                    WAVESERV_FREQ       EQU       512
                    WAVESERV_DIF        EQU       16
                    WAVESERV_LEN        EQU       20

                    STATE_PL1_SERVE     EQU       0
                    STATE_PL2_SERVE     EQU       1
                    STATE_PL1_GAME      EQU       2
                    STATE_PL2_GAME      EQU       3
                    STATE_PL1_WON       EQU       4
                    STATE_PL2_WON       EQU       5

          ;game data locations, locations are given relative to the base of gamedata
          ;fastmem refers to the bit 0..7 of each program memory position
          ;slowmem refers to the bit 8..11 of each program memory position

                    FONT      EQU $0    ;fastmem
                    STR0      EQU $100  ;fastmem
                    STR0_LEN  EQU 8     ;fastmem
                    STR1      EQU $109  ;fastmem
                    STR1_LEN  EQU 16    ;fastmem
                    STR2      EQU $11a  ;fastmem
                    STR2_LEN  EQU 22    ;fastmem
                    STR3      EQU $131  ;fastmem
                    STR3_LEN  EQU 14    ;fastmem
                    STR4      EQU $140  ;fastmem
                    STR4_LEN  EQU 14    ;fastmem
                    STR5      EQU $14f  ;fastmem
                    STR5_LEN  EQU 7     ;fastmem
                    STR6      EQU $157  ;fastmem
                    STR6_LEN  EQU 7     ;fastmem
                    STR7      EQU $15f  ;fastmem
                    STR7_LEN  EQU 9     ;fastmem
                    STR8      EQU $169  ;fastmem
                    STR8_LEN  EQU 9     ;fastmem
                    NUMBERS   EQU $173  ;fastmem
                    BALL      EQU $0    ;slowmem
                    PADDLE    EQU $c0   ;slowmem
                    SINTABLE  EQU $118  ;slowmem

                    EMPTYGFX  EQU $138

;*********************** add16 macro **************************
;* This is a macro to add two 16bit numbers, inputs two       *
;* arguments, each pointing at the lsb register followed by   *
;* the msb register at poistion arg+1.                        *
;* Reults is stored in registers referred to by first arg     *
;* arg1 = arg1 + arg2                                         *
;* clocks: 6                                                  *
;**************************************************************

add16               MACRO     2
                    add       (\1),(\2)                     ;2
                    snc                                     ;1(2)
                    inc       (\1) + 1                      ;1
                    add       (\1) + 1, (\2) + 1            ;2
                    ENDM

;*********************** sub16 macro **************************
;* This is a macro to sub two 16bit numbers, inputs two       *
;* arguments, each pointing at the lsb register followed by   *
;* the msb register at poistion arg+1.                        *
;* Reults is stored in registers referred to by first arg     *
;* arg1 = arg1 - arg2                                         *
;* clocks: 6                                                  *
;**************************************************************

sub16               MACRO     2
                    sub       (\1),(\2)                     ;2
                    sc                                                ;1(2)
                    dec       (\1) + 1                      ;1
                    sub       (\1) + 1, (\2) + 1            ;2
                    ENDM


;*********************** neg16 macro **************************
;* This is a macro to negate one 16bit number, inputs one     *
;* argument, pointing at the lsb register followed by the msb *
;* register at poistion arg+1.                                *
;* arg1 = -arg1                                               *
;* clocks: 5                                                  *
;**************************************************************

neg16               MACRO     1
                    not       (\1)
                    not       (\1)+1
                    inc       (\1)
                    snz
                    inc       (\1)+1
                    ENDM


;*********************** add168 macro *************************
;* This is a macro to add one 16bit register with one 8 bit   *
;* constant, inputs two argument: register, constant          *
;* register argument pointing at the lsb register followed by *
;* the msb register at poistion arg+1.                        *
;* Reults is stored in registers referred to by first arg     *
;* arg1 = arg1 + arg2                                         *
;* clocks: 4                                                  *
;**************************************************************

add16l8             MACRO     2
                    add       (\1),#(\2)                    ;2
                    snc                                               ;1(2)
                    inc       (\1) + 1                      ;1
                    ENDM


;*********************** sub168 macro *************************
;* This is a macro to sub one 8bit constant from one 16bit    *
;* register, inputs two argument: register, constant          *
;* register argument pointing at the lsb register followed by *
;* the msb register at poistion arg+1.                        *
;* Reults is stored in registers referred to by first arg     *
;* arg1 = arg1 - arg2                                         *
;* clocks: 4                                                  *
;**************************************************************

sub16l8             MACRO     2
                    sub       (\1),#(\2)                    ;2
                    sc                                                ;1(2)
                    dec       (\1) + 1                      ;1
                    ENDM


;*********************** mov16l macro *************************
;* This is a macro to set one 16bit register to one 16bit     *
;* constant, inputs two parameters: register,constant where   *
;* register argument pointing at the lsb register followed by *
;* the msb register at poistion arg+1                         *
;* arg1 = arg2                                                *
;* clocks: 4                                                  *
;**************************************************************

mov16l              MACRO     2
                    mov       (\1),#(\2)&$FF
                    mov       (\1)+1,#(\2)>>8
                    ENDM

;************************* dosound1 ***************************
;* This is a macro initializes a sound in sound channel one   *
;* taking three constant parameters: speed,speeddifference    *
;* and length                                                 *
;* clocks: 10                                                 *
;**************************************************************

dosound1  MACRO     3
                    mov16l    wave1speed,(\1)                         ;4        start frequency
                    mov16l    wave1speeddif,(\2)            ;4        frequency change speed
                    mov       wave1timer,#(\3)              ;2        sound length
                    ENDM

;************************* dosound2 ***************************
;* This is a macro initializes a sound in sound channel two   *
;* taking three constant parameters: speed,speeddifference    *
;* and length                                                 *
;* clocks: 10                                                 *
;**************************************************************

dosound2  MACRO     3
                    mov16l    wave2speed,(\1)                         ;4        start frequency
                    mov16l    wave2speeddif,(\2)            ;4        frequency change speed
                    mov       wave2timer,#(\3)              ;2        sound length
                    ENDM


;*********************** pcall macro **************************
;* This macro does the same as lcall but in 2 words           *
;* clocks: 4                                                  *
;**************************************************************

pjmp      MACRO     1
          page (\1)
          jmp       (\1)
          ENDM


;*********************** pcall macro **************************
;* This macro does the same as lcall but in 2 words           *
;* clocks: 4                                                  *
;**************************************************************

pcall     MACRO     1
          page (\1)
          call      (\1)
          ENDM


;************************ vout macro **************************
;* This macro outputs a constant to the video DA              *
;* clocks: 2                                                  *
;**************************************************************

vout      MACRO     1
          mov       w,#(\1)
          mov       video,w
          ENDM


;*********************** voutr macro **************************
;* This macro outputs data from a register to the video DA    *
;* clocks: 2                                                  *
;**************************************************************

voutr     MACRO     1
          mov       w,\1
          mov       video,w
          ENDM


;************************ tnop macro **************************
;* This macro creates a delay of 3 clock cycles only using    *
;* one word of program memory.                                *
;* clocks: 3                                                  *
;**************************************************************

tnop      MACRO
          jmp :tnopj
:tnopj
          ENDM

;********************** setphase macro ************************
;* This is a macro for creating delay that depends of the     *
;* contents of w, it adds w to the low part of pc, and adds   *
;* nops after the jmp instruction, the number of nops is      *
;* specified as a parameter to the function                   *
;* clocks: w+3                                                *
;**************************************************************

setphase  MACRO     1
                    jmp       pc+w
                    REPT      \1
                    nop
                    ENDR
                    ENDM

;************************ delay macro *************************
;* This is a macro for creating delays by calling the delay   *
;* functions, it minimizes the number of program words to max *
;* 6 words. For delaytimes less than 1017 and longer than 9   *
;* it uses the short delay functions at the cost of 2-3 words *
;* for shorter delays it uses the fixed delays at a cost of 1 *
;* to 3 words, longer delays are done by a call to the short  *
;* delay functions followed by a long delay call with a total *
;* cost of 4-6 words of program memory. The macro can handle  *
;* delays from 0 to 260k cycles.                              *
;*                                                            *
;* WARNING, no guarantee that this really works correctly for *
;* all delays as it quite complex and I'm too lazy to test it *
;**************************************************************
```

```
delay     MACRO     1
          IF (\1) < 0
            ERROR 'Negative delay'
          ENDIF
:delbase
          IF (:delbase & $E00) = (delay9 & $E00)
            IF ((\1)<6)
              IF ((\1)//3)=1
              nop
            ENDIF
              IF ((\1)//3)=2
                nop
                nop
              ENDIF
              IF ((\1)/3) > 0
                REPT ((\1)/3)
                  tnop
                ENDR
              ENDIF
            ENDIF

            IF ((\1)>5) AND ((\1)<10)
              call  delay6 - ((\1)-6)
            ENDIF

            IF ((\1) > 9) AND ((\1)<1027)
              mov w,#((\1)-6)>>2
              call delay_short_0 - (((\1)-6)&3)
            ENDIF

            IF (\1) > 1026
              IF (((\1)-12)//1017)<10
              mov w,#(((((\1)-12)//1017)+1017)>>2)
                call delay_short_0 - (((((\1)-12)//1017)+1017)&3)
                mov w,#(((\1)-12)/1017)-1
              ELSE
              mov w,#((((\1)-12)//1017)>>2)
                call delay_short_0 - (((((\1)-12)//1017)&3)
                mov w,#(((\1)-12)/1017)
              ENDIF
                call delay_long
            ENDIF
          ELSE
            IF ((\1)<7)
              IF ((\1)//3)=1
              nop
            ENDIF
              IF ((\1)//3)=2
                nop
                nop
              ENDIF
              IF ((\1)/3) > 0
                REPT ((\1)/3)
                  tnop
                ENDR
              ENDIF
            ENDIF

            IF ((\1)>6) AND ((\1)<11)
              page delay6
              call delay6 - ((\1)-7)
            ENDIF

            IF ((\1) > 10) AND ((\1)<1028)
              mov w,#((\1)-7)>>2
              page delay_short_0
              call delay_short_0 - (((\1)-7)&3)
            ENDIF

            IF (\1) > 1027
              IF (((\1)-14)//1017)<10
              mov w,#(((((\1)-14)//1017)+1017)>>2)
                page delay_short_0
                call delay_short_0 - (((((\1)-14)//1017)+1017)&3)
                mov w,#(((\1)-14)/1017)-1
              ELSE
              mov w,#((((\1)-14)//1017)>>2)
                page delay_short_0
                call delay_short_0 - (((((\1)-14)//1017)&3)
                mov w,#(((\1)-14)/1017)
              ENDIF
                page delay_long
                call delay_long
            ENDIF
          ENDIF

ENDM


;********************** delay functions ***********************
;* Different delay functions to be able to create long delays *
;* using as few bytes of program memory as possible           *
;* These functions are required by the delay macro            *
;* delays with exact clock count uses no registers            *
;* short delays use temp0                                     *
;* long delays use temp0 and temp1                            *
;**************************************************************

delay9              nop                           ;1        entrypoint of delay9 that delays 9 clocks
delay8              nop                           ;1        entrypoint of delay8 that delays 8 clocks
delay7              nop                           ;1        entrypoint of delay7 that delays 7 clocks
delay6              retp                          ;3        entrypoint of delay6 that delays 6 clocks

delay_short_3       nop                           ;1        entrypoint of delay_short_3 that delays 4*w + 8
delay_short_2       nop                           ;1        entrypoint of delay_short_3 that delays 4*w + 7
delay_short_1       nop                           ;1        entrypoint of delay_short_3 that delays 4*w + 6
delay_short_0       mov       temp0,w             ;1        entrypoint of delay_short_3 that delays 4*w + 5
```

```
delay_short_m       decsz     temp0               ;1(2)     decrease counter, mainloop of delay short
                    jmp       delay_short_m       ;3        keep looping until counnter is zero
                    retp                          ;3        return back to caller

delay_longmov       temp1,w             ;1        set long time counter from w
delay_long_l        mov       w,#251              ;1        set time to delay in short delay
                    call      delay_short_3       ;1012     time to delay is 251*4+8=1012
                    decsz     temp1               ;1(2)     decrease long time counter
                    jmp       delay_long_l        ;3        keep looping until counnter is zero
                    retp                          ;1        return back to caller


;*********************** jumpstart ****************************
;* Jumps to real start routine, required as chip must start   *
;* on page 0                                                  *
;**************************************************************

jumpstart pjmp      start


;*********************** memtovideo ***************************
;* outputs data from memory to video output                   *
;* temp register 0 used                                       *
;* clocks: w*12 + 7 + 1                                       *
;**************************************************************

memtovideomov       temp0,w             ;1        set pixelcounter
mtvl0               mov       w,ind               ;1        get lower level byte from mem
                    mov       video,w             ;1        send to video output
                    setb      fsr.5               ;1        select upper bank
                    mov       w,ind               ;1        get upper level byte from mem
                    inc       fsr                 ;1        point at next pixel
                    clrb      fsr.5               ;1        select lower bank
                    nop                           ;1
                    mov       video,w             ;1        send to video output
                    decsz     temp0               ;1(2)     decrease pixel counter
                    jmp       mtvl0               ;3        keep looping until all pixels are done
                    vout      BLACK               ;2        set black color
                    retp                          ;3        get outa here


;********************** setgraphics ***************************
;* outputs data from memory to video output                   *
;* temp0 = bitmap rom-pointer bit 0..7                        *
;* temp1 = bitmap rom-pointer bit 8..11                       *
;* temp2 = palette rom-pointer bit 0..7                       *
;* fsr = pointr to memory where to store graphics             *
;* Note: bits 8..11 of palette pointer is in the constant     *
;* called PALETTE_PAGE, all palettes should be placed within  *
;* this page. fsr,temp0 and temp1 are modyfied                *
;* clocks: w*31 + 5 +1                                        *
;**************************************************************

setgraphics         mov temp3,w                   ;1        set pixelcounter
sgl0                mov m,temp1                   ;2        set page
                    mov w,temp0                   ;1        get image pointer
                    iread                         ;4        read pixeldata from rom
                    mov w,m                       ;1        get slowmem nibble
                    add w,temp2                             ;1        select palettte, assuming all palettes within the
same page
                    mov m,#PALETTE_PAGE           ;1        select page
                    iread                         ;4        read palette
                    mov ind,w                     ;1        remember first level
                    setb fsr.5          ;1        select second level bank
                    and w,#$C0          ;1        mask out two upper bits
                    mov ind,w                     ;1        store second level two upper bits
                    rr ind                        ;1        move upper bits into correct position (1/2)
                    rr ind                        ;1        move upper bits into correct position (2/2)
                    mov w,m                       ;1        get second level lower nibble
                    or ind,w                      ;1        stor second level lower nibble
                    clrb fsr.5          ;1        get back to first level bank
                    inc fsr                       ;1        point at next pixel memory position
                    inc temp0                     ;1        point at next nibble
                    snz                           ;1(2)
                    inc temp1                     ;1        if page overflow, go to next page
                    decsz temp3                   ;1(2)     decrease pixel counter
                    jmp sgl0                      ;3        keep looping until all pixels are done
                    retp                                    ;3        get outa here


;********************** simplecolorfa *************************
;* outputs w color cycles at maximum amplitude                *
;* Clocks: w*12 + 11 + 1                                      *
;**************************************************************

simplecolorfa       mov       temp2,w             ;1
                    mov       temp0,#63           ;2
                    mov       temp1,#black        ;2
                    skip                          ;2


;*********************** simplecolor **************************
;* outputs w color cycles                                     *
;* Clocks: w*12 + 5 + 1                                       *
;**************************************************************

simplecolor         mov       temp2,w             ;1        set colorcycle counter
simplecolor_l       voutr     temp0               ;2        set first level
                    delay     4                   ;4        delay to get 12cycle loop
                    voutr     temp1               ;2        set second level
                    decsz     temp2               ;1(2)     decrease colorcycle counter
                    jmp       simplecolor_l       ;3        do all cycles
                    retp                          ;3        get outa here

;************************* strout *****************************
;* output characters from string in rom using a font in rom   *
;* temp0 used as character temp storage                       *
;* temp2 used as character counter                            *
;* temp1:temp3 = pointer to string                            *
;* temp4 = line (0..7) + FONT_BASE                            *
;* temp5 = length                                             *
;* clocks: 8 * 12 * w + 44 + 1                                 *
;**************************************************************

strout_cl
strout_l  vout      black                         ;2        pixel three to seven
                    delay     2                   ;2
                    rr        temp0               ;1
                    snc                           ;1(2)
                    mov       w,#53               ;1
                    mov       video,w
                    decsz     temp2               ;1(2)
                    jmp       strout_l            ;3

strout              mov       m,temp1             ;2        set character page

                    vout      black               ;2        pixel one starts here

                    mov       w,temp3             ;1        get pointer to characters
                    iread                         ;4        read one character
                    add       w,temp4             ;1        update according to line and fontbase
                    mov       m,#((gamedata + FONT) >> 8)   ;1        set font-page
                    iread                         ;4        read character pixels from font
                    mov       temp0,w             ;1        store character pixels in temp0

                    mov       w,#black            ;1
                    rr        temp0               ;1
                    snc                           ;1(2)
                    mov       w,#53               ;1
                    inc       temp3               ;1        point at next character
                    mov       video,w             ;1

                    snz                           ;1(2)
                    inc       temp1               ;1
                    delay     2
                    vout      black               ;2        pixel three starts here

                    mov       temp2,#5            ;2
                    rr        temp0               ;1
                    snc                           ;1(2)
                    mov       w,#53               ;1
                    mov       video,w             ;1
                    decsz     temp5               ;1(2)
                    jmp       strout_cl           ;3
                    vout      black
                    retp                          ;3


;************************ charout *****************************
;* output character from font in rom                          *
;* temp0 used as character temp storage                       *
;* temp1 = intensity                                          *
;* temp3:temp2 = pointer to char                              *
;* clocks: 112 + 1                                            *
;**************************************************************

charout             mov m,temp3                   ;2        set high part of pointer (i.e. page)
                    mov w,temp2                   ;1        set low part of pointer
                    iread                         ;4        read character data from rom
                    mov temp0,w                   ;1        store character data in temp0
                    mov temp2,#8                  ;2        character is 8 pixels wide
charout_l vout black                    ;2        start with black level
                    delay     2                   ;2        delay to keep phase
                    rr        temp0               ;1        rotate character data
                    snc                           ;1(2)     if lsb was zero, keep black
                    mov       w,temp1             ;2        get amplitude of character
                    mov       video,w             ;1        output to video DA
                    decsz     temp2               ;1(2)     decrease pixel counter
                    jmp       charout_l           ;3        go all pixels
                    vout      black               ;2        set black level
                    retp                          ;3        get outa here


;************************ mul_12 ******************************
;* multiply contents of w with constant 12, result in w       *
;* destroys temp2                                             *
;* clocks: 13                                                 *
;**************************************************************

mul_12              mov       temp2,w             ;1        temp2 = w
                    add       temp2,w             ;1        temp2 = w*2
                    add       temp2,w             ;1        temp2 = w*3
                    clc                           ;1        carry must be cleared before rotation
                    rl        temp2               ;1        temp2 = w*3*2
                    rl        temp2               ;1        temp2 = w*3*4 = w*12
                    mov       w,temp2             ;1        result in w (one could optimize one word and clock here, but
I'm to lazy to recalculate the timing of the main program)
                    ret                           ;3        return back


;********************** makepaddle ***************************
;* Creates paddle graphics one a line, based on y position.   *
;* This is used for both left and right paddle.               *
;* fsr = pointer to graphics buffer                           *
;* temp0 = temporary storage, contents destroyed              *
;* temp1 = temporary storage, contents destroyed              *
;* temp2 = y-pos of paddle                                    *
;* temp7 = line number                                        *
;* clocks: 314 + 1 clocks                                     *
;**************************************************************

makepaddleclr       temp1               ;1        clear output register
                    mov       temp0,temp2         ;2        temp0 = y
                    sub       temp0,temp7         ;2        temp0 = y - checkline
                    sc                            ;1(2)     check if result is negative, y < checkline
                    setb      temp1.0             ;1        no, y > checkline, set first bit
                    mov       temp0,temp2         ;2        temp0 = y
                    add       temp0,#PAD_ENDSIZE  ;2        temp0 = y + endsize
                    sub       temp0,temp7         ;2        temp0 = y + endsize - checkline
                    sc                            ;1(2)     check if result is negative, y+endsize < checkline
                    setb      temp1.1             ;1        no, y+endsize > checkline, set second bit
                    mov       temp0,temp2         ;2
                    add       temp0,#PAD_ENDSIZE + PAD_MIDSIZE        ;2
                    sub       temp0,temp7         ;2
                    sc                            ;1(2)
                    clrb      temp1.0             ;1
                    mov       temp0,temp2         ;2
                    add       temp0,#PAD_ENDSIZE*2 +PAD_MIDSIZE       ;2
                    sub       temp0,temp7         ;2
                    sc                            ;1(2)
                    clrb      temp1.1             ;1
                    mov       w,temp1             ;1
                    add       pc,w                ;3        select what to do according to result
                    jmp       paddle_black        ;3        0 - no paddle
                    jmp       paddle_bottom       ;3        3 - top part of paddle
                    jmp       paddle_top;3        1 - bottom part of paddle
                                                  ;         2 - middle part of paddle

paddle_middle       delay     19
                    mov       temp0,#((gamedata + PADDLE + 40) & $FF) ;2        set low rom pointer to paddle graphics
                    mov       temp1,#((gamedata + PADDLE + 40) >> 8)  ;2        set high rom pointer to paddle graphics
                    mov       temp2,#((PALETTE_BCW + gamedata2) & $FF);2        set palette
                    mov       w,#8                                              ;1        pad is 8 pixels wide
                    jmp       setgraphics                                       ;253

paddle_bottom       mov       temp3,#PAD_ENDSIZE                      ;2        set bottom reference
                    mov       temp0,#((gamedata + PADDLE + 48) & $FF) ;2        set low rom pointer to paddle graphics
                    mov       temp1,#((gamedata + PADDLE + 48) >> 8)  ;2        set high rom pointer to paddle graphics
                    jmp       paddle_bottom_j                                   ;3

paddle_topmov       temp3,#PAD_ENDSIZE*2 + PAD_MIDSIZE      ;2        set top reference
                    mov       temp0,#((gamedata + PADDLE) & $FF)      ;2        set low rom pointer to paddle graphics
                    mov       temp1,#((gamedata + PADDLE) >> 8)       ;2        set high rom pointer to paddle graphics
                    jmp       paddle_bottom_j                                   ;3

paddle_bottom_j     add       temp2,temp3         ;2        temp2 = y + size
                    sub       temp2,temp7         ;2        temp2 = y + size - linenumber

                    clc                           ;1
                    rl        temp2               ;1
                    rl        temp2               ;1
                    mov       w,<<temp2           ;1        w = (y + size - linenumber)*8
                    add       temp0,w             ;1        select line in graphics
                    snc                           ;1(2)     check for overflow
                    inc       temp1               ;1        if overflow, change to next page
                    mov       temp2,#((PALETTE_BCW + gamedata2) & $FF);2        set palette
                    mov       w,#8                ;1        pad is 8 pixels wide
                    jmp       setgraphics         ;253

paddle_black        delay     16
                    mov       temp0,#((gamedata + EMPTYGFX) & $FF)    ;2        set low rom pointer to paddle graphics
                    mov       temp1,#((gamedata + EMPTYGFX) >> 8)     ;2        set high rom pointer to paddle graphics
                    mov       temp2,#((PALETTE_BCW + gamedata2) & $FF);2        set palette
                    mov       w,#8                                    ;1        pad is 8 pixels wide
                    jmp       setgraphics                             ;253


;************************ makeball ****************************
;* Creates ball graphics on a line                            *
;* Requires makepad as it reuses routines for empty line to   *
;* save rom memory (as both paddle and ball are 12 pexels)    *
;* fsr = pointer to graphics buffer                           *
;* temp0 = temporary storage, contents destroyed              *
;* temp1 = temporary storage, contents destroyed              *
;* temp2 = temporary storage, contents destroyed              *
;* temp7 = line number                                        *
;* clocks: 421+1 clocks                                       *
;**************************************************************

makeball  bank $00
                    mov       temp0,#((gamedata + BALL) & $FF)        ;2        set low rom pointer to ball graphics
                    mov       temp1,#((gamedata + BALL) >> 8)         ;2        set high rom pointer to ball graphics

                    mov       temp2,bally_h                           ;2
                    sub       temp2,temp6                             ;2
                    snc                                               ;1(2)
                    jmp       noball_j1                               ;3

                    mov       temp2,bally_h                           ;2
                    add       temp2 ,#BALL_LINES                      ;2
                    sub       temp2,temp6                             ;2
                    sc                                                ;1(2)
                    jmp       noball_j2                               ;3

                    mov       w,temp2                                 ;1
                    call      mul_12                                  ;13
                    add       temp0,w                                 ;1        select line in graphics
                    snc                                               ;1(2)     check for overflow
                    inc       temp1                                   ;1        if overflow, change to next page
                    mov       temp2,#((PALETTE_BCW + gamedata2) & $FF);2        set palette
                    mov       fsr,#BALL_BUFFER                        ;2        point at ball buffer
                    mov       w,#12                                   ;1        pad is 12 pixels wide
                    jmp       setgraphics                             ;377

noball_j1 delay     8
noball_j2 delay     11
                    mov       fsr,#BALL_BUFFER                        ;2        point at ball buffer
                    mov       temp0,#((gamedata + EMPTYGFX) & $FF)    ;2        set low rom pointer to paddle graphics
                    mov       temp1,#((gamedata + EMPTYGFX) >> 8)     ;2        set high rom pointer to paddle graphics
                    mov       temp2,#((PALETTE_BCW + gamedata2) & $FF);2        set palette
                    mov       w,#12                                   ;1        pad is 8 pixels wide
                    jmp       setgraphics                             ;377
```

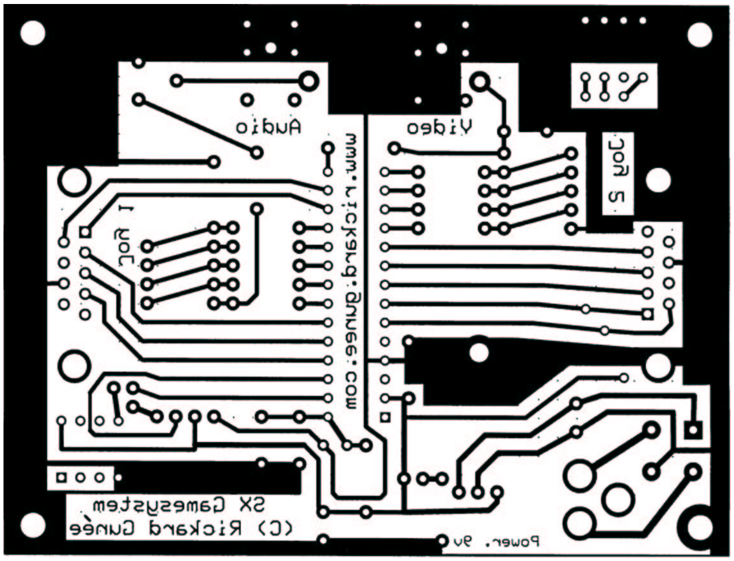
```
;*********************** itext macro **************************
;* macro for showing a line of chars from rom                 *
;* parameters: strpointer,length,base,phase                   *
;**************************************************************

ITEXT               MACRO     4
                    mov       temp7,#8                                ;2
                    mov       temp4,#((gamedata + FONT) & $ff)        ;2
:bots_l             pcall     hsync                                   ;1+TIME_HSYNC
                    delay     (\3) - (\4)
                    mov       temp1,#(((\1) + gamedata) >> 8)         ;2
                    mov       temp3,#(((\1) + gamedata) & $FF)        ;2
                    mov       temp5,#(\2)                             ;2
                    pcall     strout                                  ;STR_LEN*8 * 12 * (w-1) + 42 + 1
                    inc       temp4                                   ;1
                    delay     TIME_IMAGE-((((\2)-1)*8*12) + 44 + 1) - (\3) + (\4) - (2+2+2+1+4+1)
                    decsz     temp7                                   ;1(2)
                    jmp       :bots_l                                 ;3
                    delay     2
                    pcall     hsync
                    ENDM


;************************ makeball ****************************
;* shows initscreen where user can select gametype            *
;**************************************************************

                    INITLINES EQU       9+9+9+44+9+14+9+49+9
initscreenbank      $00                           ;1
                    mov       w,rc                          ;1
                    and       w,#%11100000                  ;1
                    mov       oldj1,w                       ;1
                    mov       w,ra                          ;1
                    and       w,#%00001111                  ;1
                    or        oldj1,w                       ;1
                    mov       oldj2,rb            ;2

                    pcall     vsync

                    ;delay    TIME_IMAGE - 17 - 2

                    mov       w,#PRE_LINES + ((MID_LINES - INITLINES)/2)
                    pcall     emptylines

                    delay     12-5
                    itext     STR0,STR0_LEN,INITTEXT1_BASE,INITTEXT1_PHASE
                    delay     TIME_IMAGE - 17 - 2

                    mov       w,#9
                    pcall     emptylines

                    delay     12-5
                    itext     STR1,STR1_LEN,INITTEXT2_BASE,INITTEXT2_PHASE
                    delay     TIME_IMAGE - 17 - 2

                    mov       w,#44
                    pcall     emptylines

                    delay     12-5
                    mov       temp7,#8                                ;2
                    mov       temp4,#((gamedata + FONT) & $ff)        ;2
iscrl_l             pcall     hsync
                    delay     INITTEXT3_BASE - INITTEXT3_PHASE
                    mov       temp1,#((STR6 + gamedata) >> 8)         ;2
                    mov       temp3,#((STR6 + gamedata) & $FF)        ;2
                    mov       temp5,#STR6_LEN                         ;2
                    call      strout                                  ;STR_LEN*8*12+44
                    delay     (12*4) - 7 + 6
                    bank      $00                                     ;1
                    mov       w,#((STR7 + gamedata) & $FF)            ;1
                    snb       gamekind.1                    ;1(2)
                    mov       w,#((STR8 + gamedata) & $FF)            ;1
                    mov       temp3,w                                 ;1
                    mov       temp1,#((STR7 + gamedata) >> 8)         ;2        ;2
                    mov       temp5,#STR7_LEN                         ;2
                    call      strout                                  ;STR_LEN*8*12+44
                    inc       temp4                                   ;1
                    delay     TIME_IMAGE-((STR6_LEN-1)*8*12+44) - INITTEXT3_BASE + INITTEXT3_PHASE - (2+2+2+1+4+(12*4)-
7+6+7+2+1) - ((STR7_LEN-1)*8*12+44)
                    decsz     temp7                                   ;1(2)
                    jmp       iscrl_l                                 ;3
                    delay     2
                    pcall     hsync
                    delay     TIME_IMAGE - 17 - 2

                    mov       w,#14
                    pcall     emptylines

                    delay     12-5
                    mov       temp7,#8                                ;2
                    mov       temp4,#((gamedata + FONT) & $ff)        ;2
iscrr_l             pcall     hsync
                    delay     INITTEXT4_BASE - INITTEXT4_PHASE
                    mov       temp1,#((STR5 + gamedata) >> 8)         ;2
                    mov       temp3,#((STR5 + gamedata) & $FF)        ;2
                    mov       temp5,#STR5_LEN                         ;1
                    call      strout                                  ;STR_LEN*8*12+44
                    delay     (12*4) - 8 + 6
                    mov       w,#((STR7 + gamedata) & $FF)            ;1

                    bank      $00                                     ;1
                    test      gamekind
                    sz                                                ;1(2)
                    mov       w,#((STR8 + gamedata) & $FF)            ;1
                    mov       temp3,w                                 ;1
                    mov       temp1,#((STR7 + gamedata) >> 8)         ;2
                    mov       temp5,#STR7_LEN                         ;2
                    call      strout                                  ;STR_LEN*8*12+44
                    inc       temp4                                   ;1
                    delay     TIME_IMAGE-((STR5_LEN-1)*8*12+44) - INITTEXT4_BASE + INITTEXT4_PHASE - (2+2+2+1+4+(12*4)-
7+6+7+2+1) - ((STR7_LEN-1)*8*12+44)
                    decsz     temp7                                   ;1(2)
                    jmp       iscrr_l                                 ;3
                    delay     2
                    pcall     hsync
                    delay     TIME_IMAGE - 17 - 2

                    mov       w,#49
                    pcall     emptylines

                    delay     12-5
                    itext     STR2,STR2_LEN,INITTEXT5_BASE,INITTEXT5_PHASE
                    delay     TIME_IMAGE - 17 - 2

                    mov       w,#POST_LINES + ((MID_LINES - INITLINES)/2) + ((MID_LINES - INITLINES)//2) - 1  ;1
                    pcall     emptylines

                    delay     12-1

                    pcall     hsync

                    bank      $00                           ;1
                    sb        joy1up_old          ;1(2)
                    jmp       initnoupwarp                  ;1
                    sb        joy1up                        ;1(2)
                    dec       gamekind                      ;1
                    mov       w,gamekind          ;1
                    xor       w,#$FF                        ;1
                    mov       w,#3                          ;1
                    snz                                     ;1(2)
                    mov       gamekind,w          ;1
initnoupwarpr
                    sb        joy1down_old                  ;1(2)
                    jmp       initnodownwarp                ;1
                    sb        joy1down                      ;1(2)
                    inc       gamekind                      ;1
                    mov       w,gamekind          ;1
                    xor       w,#4                          ;1
                    snz                                     ;1(2)
                    clr       gamekind                      ;1
initnodownwarpr
                    delay     TIME_IMAGE - 10 - 8 - 6 - 9 - 4

                    sb        joy1button_old                ;1(2)
                    skip                                    ;2
                    snb       joy1button          ;1(2)
                    jmp       initscreen          ;3
                    clrb      joy1button_old                ;1
                    pjmp      premain

initnoupwarp        jmp       initnoupwarpr
initnodownwarp      jmp       initnodownwarpr


org $200
;************************* vsync ******************************
;* Performas a vertical sync pattern on the video output      *
;* Uses temp0..temp2                                          *
;**************************************************************

vsync               mov       w,#4                          ;1        odd, make 5 pulses instead
                    call      short_sync          ;5        clocks until sync, make those pulses,
                    mov       temp2,w                       ;1        counter0=5
long_sync_l         clr       video                         ;1        set video level to sync
                    delay     (TIME_27US5 - 1)              ;         30uS long sync pulse
                    vout      black                         ;2        set video level to black
                    call      vsound                        ;65
                    delay     (TIME_4US5 - 6 - 65);         2us long black pulse
                    decsz     temp2                         ;1(2)
                    jmp       long_sync_l                   ;3
                    mov       w,#5                          ;1        odd, make 4 pulses instead of 5
short_syncmov       temp2,w                       ;1
short_sync_l        clr       video                         ;1        set video level to sync
                    call      vsound                        ;65
                    delay     (TIME_2US4 - 65 - 1)          ;2us long sync pulse
                    vout      black                         ;2        set video level to black
                    delay     (TIME_29US6 - 6)              ;         30us long black pulse
                    decsz     temp2                         ;1(2)
                    jmp       short_sync_l                  ;3
                    retw      5                             ;3
vsound              pjmp      vrealsound          ;62

;************************* hsync ******************************
;* performas a horizontal sync pulse and color burst          *
;* uses temp0                                                 *
;**************************************************************

hsync               delay     TIME_PRESYNC-3-1              ;85
                    clr       video                         ;1

                    call      sound                         ;61
                    delay     TIME_SYNC-2-61                ;248
                    vout      neutral                       ;2

                    delay     TIME_PREBURST-2               ;44
                    mov       temp0,#12                     ;2
hsyncl              vout      6                             ;2
                    delay     4                             ;4
                    vout      21                            ;2
                    decsz     temp0                         ;1(2)
                    jmp       hsyncl                        ;3
                    delay     2                             ;2
                    vout      neutral                       ;2
                    delay     time_postburst - 2-3;114
                    retp                                    ;3
```

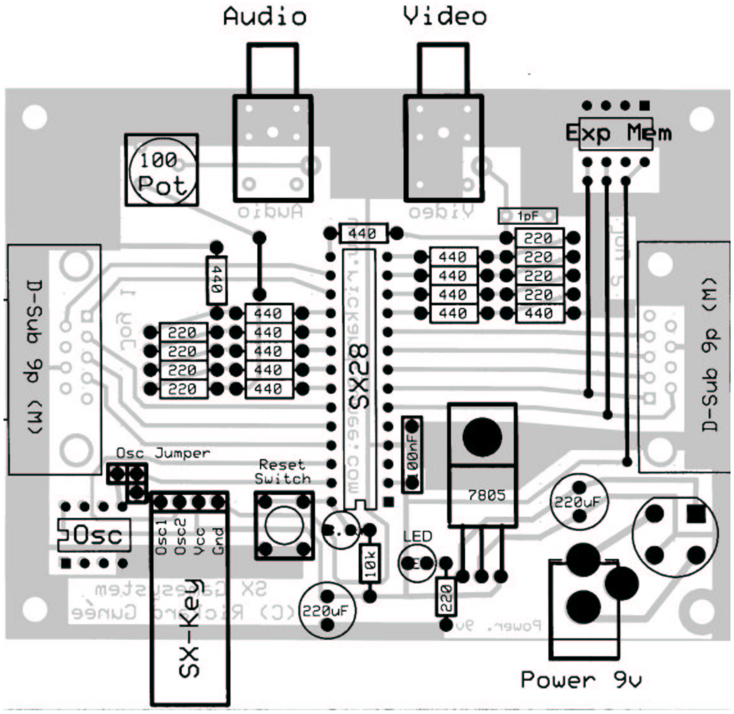
```
sound               pjmp      realsound                     ;58


;*********************** emptylines **************************
;* Displays w empty lines, 17 + 1 clocks until hsync when     *
;* called and 12 clocks until next hsync when returned        *
;*************************************************************

emptylinesmov       temp3,w                       ;1
emptylines_l        delay     13                            ;13
                    call      hsync                         ;643
                    delay     (TIME_IMAGE-4-13)             ;
                    decsz     temp3                         ;1(2)
                    jmp       emptylines_l                  ;3
                    retp                                    ;3

checkpad  mov       temp1,w                                 ;1
                    mov       temp0,bally_h                 ;2
                    add       temp0,#BALL_LINES             ;2
                    cjb       temp0,temp1,cpwarp            ;4(6)
                    add       temp1,#PAD_ENDSIZE*2 + PAD_MIDSIZE      ;2
                    mov       w,bally_h                     ;1
                    mov       w,temp1-w                     ;1
                    ret                                     ;3
cpwarp              delay     1
                    clc                                     ;1
                    retp                                    ;3

;*********************** warpzone2 ***************************
;* In some places, mostly in the game logic, jump and delay   *
;* needs to be done, these delays are placed here             *
;*************************************************************

rightwarp delay     68 - 6 - 3
                    jmp       rightwarpr

leftwarp  delay     68 - 6 - 3
                    jmp       leftwarpr

topwarp             delay     23 - 6 - 3
                    jmp       topwarpr

bottomwarpdelay     23 - 6 - 3
                    jmp       bottomwarpr

noservewarp11       delay     2
noservewarp21       delay     24-3-3
                    jmp       server
noservewarp12       delay     2
noservewarp22       delay     21-3-3
                    jmp       server

noup1warp jmp       noup1warpr
nodown1warp         jmp       nodown1warpr
noup2warp jmp       noup2warpr
nodown2warp         jmp       nodown2warpr
noscrewup1warp      jmp       noscrewup1warpr
noscrewdown1warp jmpnoscrewdown1warpr
noscrewup2warp      jmp       noscrewup2warpr
noscrewdown2warp jmpnoscrewdown2warpr
nonegxswarp         jmp       nonegxswarpr
nonegyswarp         jmp       nonegyswarpr


nosmash2warp        delay     13
                    jmp       nosmash2warpr
nosmash1warp        delay     13
                    jmp       nosmash1warpr

warpgover sb        joy1button_old
                    jmp       warpgover_j
                    sb        joy1button
                    jmp       warptostart
warpgover_j         delay     (68+68+23+23+14+14) - (4+4+3)
                    jmp       warpgoverr
warptostart         pjmp      start

warpmove  snb       state.0                                 ;1(2)
                    jmp       serve2                        ;3
serve1              mov       bally_h,y1          ;2        set ball y to paddle 1 y
                    mov       ballx_h,#1          ;2        set ball x to most right
                    add       bally_h,#((PAD_ENDSIZE*2 + PAD_MIDSIZE)/2) - (BALL_LINES / 2)         ;2        put   ball
in center of paddle
                    snb       mixedbits.6                   ;1(2)     coputer player ?
                    jmp       doserve1                      ;3        yes, do serve automatically
                    sb        joy1button_old                ;1(2)     check for previous firebutton
                    jmp       noservewarp11                 ;3        was recently pressed, don't serve
                    snb       joy1button          ;1(2)     check for fire button
                    jmp       noservewarp21                 ;3        no fire button, don't serve
doserve1  clr       ballx_speed_h                           ;1        clear high byte of x-speed
                    mov       bally_speed_l,#$80            ;2        set y-speed
                    mov       ballx_speed_l,#$80            ;2        set x-speed
                    mov       bally_speed_h,#$FF            ;2        set negative y-direction
                    setb      state.1                       ;1
                    bank      $20                           ;1
                    dosound1 WAVESERV_FREQ,WAVESERV_DIF,WAVESERV_LEN  ;10
                    bank      $00                           ;1
                    jmp       server                        ;3        get on with it

serve2              nop                                     ;1        delay nop to get both vcases equal length
                    mov       bally_h,y2          ;2        set ball y to paddle 1 y
                    mov       ballx_h,#BALLAREA_WIDTH-1     ;2        set ball x to most right
                    add       bally_h,#((PAD_ENDSIZE*2 + PAD_MIDSIZE)/2) - (BALL_LINES / 2)         ;2        put   ball
in center of paddle
                    snb       mixedbits.7                   ;1(2)     coputer player ?
                    jmp       doserve2                      ;3        yes, do serve automatically
                    sb        joy2button_old                ;1(2)     check for previous firebutton
                    jmp       noservewarp12                 ;3        was recently pressed, don't serve
                    snb       joy2button          ;1(2)     check for fire button
                    jmp       noservewarp22                 ;3        no fire button, don't serve
doserve2  clr       bally_speed_h                           ;1        clear high byte of y-speed
                    mov       bally_speed_l,#$80            ;2        set y-speed
                    mov       ballx_speed_l,#$80            ;2        set x-speed
                    mov       ballx_speed_h,#$FF            ;2        set negative x-direction
                    bank      $20                           ;1
                    dosound1 WAVESERV_FREQ,WAVESERV_DIF,WAVESERV_LEN  ;10
                    bank      $00                           ;1
                    setb      state.1                       ;1
server
                    delay     3-3+12+14+68+68+23+23-3-37
                    jmp       warpmover


;*********************** main loop ****************************
;* This is the game main loop                                 *
;**************************************************************

premain             test      gamekind                      ;premain sets the game kind when arriving from initscreen
                    sz
                    setb      mixedbits.7
                    snb       gamekind.1
                    setb      mixedbits.6
main
                    call      vsync                         ;         vertical sync, frame starts here

                    call      hsync                         ;line 1
                    bank $00                                ;1

                    jb        joy1up,noup1warp              ;2(4)     joy1 up pressed ?
                    csa       y1,#PLAYFIELD_LINES-PAD_SIZE-1 ;3(4)    yes, can paddle 1 move up ?
                    inc       y1                            ;1        yes, increase y-pos of paddle 1
                    nop                                     ;1        delay to keep phase
noup1warprjb        joy1down,nodown1warp;2(4)     joy1 down pressed ?
                    csb       y1,#1                         ;3(4)     yes, can paddle 1 move down ?
                    dec       y1                            ;1        yes, decrease y-pos of paddle 1
                    nop                                               ;1        delay to keep phase
nodown1warpr        jb        joy2up,noup2warp                        ;2(4)     joy2 up pressed ?
                    csa       y2,#PLAYFIELD_LINES-PAD_SIZE-1;3(4)     yes, can paddle 2 move up ?
                    inc       y2                                      ;1        yes, increase y-pos of paddle 2
                    nop                                               ;1        delay to keep phase
noup2warprjb        joy2down,nodown2warp          ;2(4)     joy2 down pressed ?
                    csb       y2,#1                                   ;3(4)     yes, can paddle 2 move down ?
                    dec       y2                                      ;1        yes, decrease y-pos of paddle 2
                    nop                                               ;1        delay to keep phase
nodown2warpr

                    mov       w,state                                 ;1        get state
                    and       w,#$FE                                  ;1        remove player bit
                    snz                                               ;1(2)     check if zero
                    jmp       warpmove                                ;3        yes, serve, dont move ball
                    jb        state.2,warpgover                       ;2(4)     if gameover don't move ball and stuff

                    add16     ballx,ballx_speed                       ;6        move ball in x-direction
                    jnb       ballx_speed_h.7,nonegxswarp             ;2(4)     check if x-direction is negative
                    mov       w,ballx_h                               ;1
                    and       w,#$E0                                  ;1
                    xor       w,#$E0                                  ;1
                    snz                                               ;1(2)
                    clr       ballx_h                                 ;1        if upper 1/4, then clear
nonegxswarpr
                    add16     bally,bally_speed                       ;6        move ball in y-direction
                    jnb       bally_speed_h.7,nonegyswarp             ;2(4)     check if x-direction is negative
                    mov       w,bally_h                               ;1
                    and       w,#$E0                                  ;1
                    xor       w,#$E0                                  ;1
                    snz                                               ;1(2)
                    clr       bally_h                                 ;1        if upper 1/8, then clear
nonegyswarpr
                    cjae      ballx_h,#1,leftwarp                     ;4(6)     check if x less than one
                    mov       w,y1                                    ;1        get left pad y -position
                    call      checkpad                                ;19       check if ball hits paddle
                    jnc       miss2                                   ;2(4)     if player missed ball, handle score etc
                    neg16     ballx_speed                             ;5        yes, change x-direction of ball
                    mov       ballx_h,#1                    ;2        stop ball from beeing out of bounds
                    bank      $20                                     ;1
                    dosound1 WAVEPAD_FREQ,WAVEPAD_DIF,WAVEPAD_LEN     ;10
                    bank      $00                                     ;1
                    jb        joy1button,nosmash1warp                 ;2(4)     joy1 right pressed ?
                    add16l8   ballx_speed,$80                         ;4        add more speed in y-direktion
                    jb        joy1up,noscrewup1warp                   ;2(4)     joy1 up pressed ?
                    add16l8   bally_speed,$60                         ;4        add more speed in y-direktion
                    nop                                               ;1        one cycle delay to get in sync with warp
delay
noscrewup1warpr     jb        joy1down,noscrewdown1warp               ;2(4)     joy1 down pressed ?
                    sub16l8   bally_speed,$60                         ;4        subtract more speed in y-direktion
                    nop                                               ;1        one cycle delay to get in sync with warp
delay
noscrewdown1warpr
nosmash1warpr       jmp       nomiss2                                 ;3        the ball bounced, get on with the game

miss2               snb       state.0                                 ;1(2)     was it the left player that served
                    inc       p2                                      ;1        yes, increase player one's points
                    mov       state,#STATE_PL2_SERVE                  ;2        player 1 is going to serve
                    csne      p2,#9                                   ;3(4)     if  player  1  got  10point,  show  winner
screen
                    setb      state.2                                 ;1
                    bank      $20                                     ;1
                    dosound2 WAVEMISS_FREQ,WAVEMISS_DIF,WAVEMISS_LEN  ;10
                    bank      $00                                     ;1
                    delay     44-24
nomiss2
leftwarpr cjbe      ballx_h,#BALLAREA_WIDTH-1,rightwarp               ;4(6)     check if x larger than playfield width
                    mov       w,y2                                    ;1        get left pad y -position
                    call      checkpad                                ;19       check if ball hits paddle
                    jnc       miss1                                   ;2(4)     if player missed ball, handle score etc
                    neg16     ballx_speed                             ;5        yes, change x-direction of ball
                    mov       ballx_h,#BALLAREA_WIDTH-1               ;2        stop ball from beeing out of bounds
                    bank      $20                                               ;1
                    dosound1 WAVEPAD_FREQ,WAVEPAD_DIF,WAVEPAD_LEN     ;10
                    bank      $00                                     ;1
                    jb        joy2button,nosmash2warp                 ;2(4)     joy1 right pressed ?
                    sub16l8   ballx_speed,$80                         ;4        add more speed in y-direktion
                    jb        joy2up,noscrewup2warp                   ;2(4)     joy1 up pressed ?
                    add16l8   bally_speed,$60                         ;4        add more speed in y-direktion
                    nop                                               ;1        one cycle delay to get in sync with warp
delay
noscrewup2warpr     jb        joy2down,noscrewdown2warp               ;2(4)     joy1 down pressed ?
                    sub16l8   bally_speed,$60                         ;4        subtract more speed in y-direktion
                    nop                                               ;1        one cycle delay to get in sync with warp
delay
noscrewdown2warpr
nosmash2warpr       jmp       nomiss1                                 ;3        the ball bounced, get on with the game

miss1               sb        state.0                                 ;1(2)     was it the right player that served
                    inc       p1                                      ;1        yes, increase player two's points
                    mov       state,#STATE_PL1_SERVE                  ;2        player 2 is going to serve
                    csne      p1,#9                                   ;3(4)     if  player  2  got  10point,  show  winner
screen
                    setb      state.2
                    bank      $20                                     ;1
                    dosound2 WAVEMISS_FREQ,WAVEMISS_DIF,WAVEMISS_LEN  ;10
                    bank      $00                                     ;1
                    delay     44-24
nomiss1

rightwarprcjae      bally_h,#1,bottomwarp                   ;4(6)     check if y less than one
                    neg16     bally_speed                             ;5        yes, change y-direction of ball
                    mov       bally_h,#1                    ;2        stop ball from beeing out of bounds
                    bank      $20                                     ;1
                    dosound2 WAVEWALL_FREQ,WAVEWALL_DIF,WAVEWALL_LEN  ;10
                    bank      $00                                     ;1
bottomwarpr
                    cjbe      bally_h,#PLAYFIELD_LINES-1-BALL_LINES,topwarp     ;4(6)     check   if   y   larger   than
playfield height
                    neg16     bally_speed                             ;5        yes, change y-direction of ball
                    mov       bally_h,#PLAYFIELD_LINES-1-BALL_LINES   ;2        stop ball from beeing out of bounds
                    bank      $20                                     ;1
                    dosound2 WAVEWALL_FREQ,WAVEWALL_DIF,WAVEWALL_LEN  ;10
                    bank      $00                                     ;1

topwarpr
warpmover
warpgoverr
                    bank      $00                           ;1
                    mov       w,rc                          ;1
                    and       w,#%11100000                  ;1
                    mov       oldj1,w                       ;1
                    mov       w,ra                          ;1
                    and       w,#%00001111                  ;1
                    or        oldj1,w                       ;1
                    mov       oldj2,rb            ;2

                    delay     TIME_IMAGE - 17 - 1 - 28 - 6 - 12 - 68 - 68 - 23 - 23 - 9

                    mov       w,#PRE_LINES - 1
                    call      emptylines

                    delay     11
                    pcall     hsync
                    delay     12 - TEXTLINE_PHASE
                    mov       w,#((TIME_IMAGE-14) / 12)-1
                    pcall     simplecolorfa
                    delay     TEXTLINE_PHASE + ((TIME_IMAGE-14) // 12)

                    pcall     hsync
                    delay     TIME_IMAGE-5

                    mov       temp7,#7
                    mov       temp4,#((gamedata + FONT) & $ff)
tops_l              pcall     hsync
                    delay     LEFTSCORE_BASE - LEFTSCORE_PHASE
                    mov       temp3,#((gamedata + NUMBERS) >> 8)      ;2        let temp3 point at correct page
                    mov       w,#63                                   ;1        high intensity
                    snb       state.0                                 ;1(2)     left players serve ?
                    mov       w,#40                                   ;1        no, lower intensity
                    mov       temp1,w                                 ;1        set intensity
                    clc                                               ;1
                    mov       w,<<p1                                  ;1        get points*2
                    mov       temp2,w                                 ;1        temp2 = points*2
                    rl        temp2                                   ;1        temp2 = points*4
                    rl        temp2                                   ;1        temp2 = points*8
                    add       temp2,#((gamedata + NUMBERS) & $FF) + 7 ;2        add low part of point to numbers
                    sub       temp2,temp7                             ;2        select line in font
                    pcall     charout                                 ;113      output number

                    delay     TTEXT_BASE-TTEXT_PHASE-(LEFTSCORE_BASE - LEFTSCORE_PHASE + 128)
                    mov       temp1,#((gamedata + STR0) >> 8)         ;2
                    mov       temp3,#((STR0 + gamedata) & $FF)        ;2
                    mov       temp5,#STR0_LEN                         ;2
                    pcall     strout
                    inc       temp4                                   ;1
                    delay     TIME_IMAGE-((STR0_LEN-1)*8*12+45+1)-TTEXT_BASE+TTEXT_PHASE-10-(RIGHTSCORE_BASE           +
RIGHTSCORE_PHASE + 128)

                    mov       temp3,#((gamedata + NUMBERS) >> 8)      ;2        let temp3 point at correct page
                    mov       w,#63                                   ;1        high intensity
                    sb        state.0                                 ;1(2)     right players serve ?
                    mov       w,#40                                   ;1        no, lower intensity
                    mov       temp1,w                                 ;1        set intensity
                    clc                                               ;1
                    mov       w,<<p2                                  ;1        get points*2
                    mov       temp2,w                                 ;1        temp2 = points*2
                    rl        temp2                                   ;2        temp2 = points*4
                    rl        temp2                                   ;2        temp2 = points*8
                    add       temp2,#((gamedata + NUMBERS) & $FF) + 7 ;2        add low part of point to numbers
                    sub       temp2,temp7                             ;2        select line in font
                    pcall     charout                                 ;113      output number

                    delay     RIGHTSCORE_BASE + RIGHTSCORE_PHASE - 1 - 1


                    page      tops_l
                    decsz     temp7                                   ;1(2)
                    jmp       tops_l                                  ;3

                    delay     2
                    pcall     hsync
                    delay     TIME_IMAGE - 1

                    pcall     hsync
                    delay     12 - TEXTLINE_PHASE
                    mov       w,#((TIME_IMAGE-22) / 12)-1
                    pcall     simplecolorfa
                    delay     TEXTLINE_PHASE + ((TIME_IMAGE-22) // 12)

                    mov       temp7,#PLAYFIELD_LINES                  ;2
                    mov       temp6,w                                 ;1

                    snb       state.2                                 ;1(2)
                    jmp       gameover                                ;3
                    jmp       playfield_l                             ;3

gameover  nop
                    pcall     hsync
                    delay     TIME_IMAGE - 17 - 2
                    mov       w,#(PLAYFIELD_LINES - 10) / 2;epty lines at the top
                    pcall     emptylines
                    delay     12 - 5

                    mov       temp7,#8                                ;2
                    mov       temp4,#((gamedata + FONT) & $ff)        ;2
gov_l               pcall     hsync
                    delay     WINTEXT_BASE-WINTEXT_PHASE
                    mov       w,#((STR3 + gamedata) & $FF)            ;1
                    sb        state.0                                 ;1(2)
                    mov       w,#((STR4 + gamedata) & $FF)            ;1
                    mov       temp3,w                                 ;1
                    mov       temp1,#((gamedata + STR3) >> 8)         ;2
                    mov       temp5,#STR3_LEN                         ;2
                    pcall     strout
                    inc       temp4                                   ;1
                    delay     TIME_IMAGE-((STR3_LEN-1)*8*12+45+1)-WINTEXT_BASE+WINTEXT_PHASE-12-1
                    decsz     temp7
                    jmp       gov_l
                    delay     2
                    pcall     hsync

                    delay     TIME_IMAGE - 17 - 2
                    mov       w,#((PLAYFIELD_LINES - 9) / 2) + ((PLAYFIELD_LINES - 9) // 2)
                    pcall     emptylines

                    delay     12 - 3 - 1
                    jmp       playfield_e


playfield_l         pcall     hsync

                    delay     LEFT_SPACE+12-LEFTPAD_PHASE
                    mov       fsr,#LPAD_BUFFER                        ;2
                    mov       w,#8                                    ;1
                    pcall     memtovideo                    ;104

                    delay     LEFTPAD_PHASE
                    bank      $00                                     ;1        select correct bank
                    mov       temp5,ballx_h                           ;2
                    mov       temp0,ballx_h                           ;2        temp0 = ball_xh
                    sub       temp0,#(36+1)                           ;2        temp0 = ball_xh-(36+1)
                    sc                                                ;1(2)     if x<(36+1)
                    jmp       doball                                  ;3      don't make graphics
                    sub       temp5,#36                               ;2        remove used pixels
                    pcall     makeball                                ;422      create graphics
                    delay     (36*12) - (1+2+422 +1+2+2+1)            ;         make section 40 pixels wide

                    bank      $00                                     ;1        select correct bank
                    mov       temp0,ballx_h                           ;2        temp0 = ball_xh
                    sub       temp0,#(36+28+1)                        ;2        temp0 = ball_xh-(36+28+1)
                    sc                                                ;1(2)     if x<(36+28+1)
                    jmp       doball                                  ;3      don't make graphics
                    sub       temp5,#28                               ;2        remove used pixels
                    mov       temp2,y2                                ;2        use player 2 y-position
                    mov       fsr,#RPAD_BUFFER                        ;2        point at left video buffer
                    pcall     makepaddle                              ;318      create graphics
                    delay     (28*12) - (1+2+2+2+318 +1+2+2+1)

                    bank      $00                                               ;1        select correct bank
                    mov       temp0,ballx_h                           ;2        temp0 = ball_xh
                    sub       temp0,#(36+28+28+1)                     ;2        temp0 = ball_xh-(36+28+28+1)
                    sc                                                ;1(2)     if x<(36+28+28+1)
                    jmp       doball                                  ;3      don't make graphics
                    sub       temp5,#28                               ;2        remove used pixels
                    mov       temp2,y1                                ;2        use player 1 y-position
                    mov       fsr,#LPAD_BUFFER                        ;2        point at left video buffer
                    pcall     makepaddle                    ;318      create graphics

                    delay     (28*12) - (1+2+2+2+318) + 3

doball              dec       temp6                                   ;1

                    mov       temp0,temp5                             ;2
ball_left_l         delay     8
                    decsz     temp0
                    jmp       ball_left_l
                    delay     12-BALL_PHASE
                    mov       fsr,#BALL_BUFFER                        ;2
                    mov       w,#12                                   ;1
                    pcall     memtovideo                    ;152

                    delay     BALL_PHASE
                    mov       temp0,#BALLAREA_WIDTH - (36+28+28)      ;2
                    sub       temp0,temp5                             ;2
ball_right_l        delay     8
                    decsz     temp0
                    jmp       ball_right_l

                    bank      $00                                     ;1        select correct bank
                    mov       temp0,ballx_h                           ;2        temp0 = ball_xh
                    sub       temp0,#93                               ;2        temp0 = ball_xh-76
                    snc                                               ;1(2)     if x<76
                    jmp       endball                                 ;3      don't make graphics
                    mov       temp2,y1                                ;2        use player 1 y-position
                    mov       fsr,#LPAD_BUFFER                        ;2        point at left video buffer
                    pcall     makepaddle                    ;318      create graphics
                    delay     (28*12) - (1+2+2+318 +1+2+2+1)

                    bank      $00                                     ;1        select correct bank
                    mov       temp0,ballx_h                           ;2        temp0 = ball_xh
                    sub       temp0,#65                               ;2        temp0 = ball_xh-64
                    snc                                               ;1(2)     if x<38
                    jmp       endball                                 ;3      don't make graphics
                    mov       temp2,y2                                ;2        use player 2 y-position
                    mov       fsr,#RPAD_BUFFER                        ;2        point at left video buffer
                    pcall     makepaddle                    ;318      create graphics
                    delay     (28*12) - (1+2+2+318 +1+2+2+1)

                    bank      $00                                     ;1        select correct bank
                    mov       temp0,ballx_h                           ;2        temp0 = ball_xh
                    sub       temp0,#37                               ;2        temp0 = ball_xh-37
                    snc                                               ;1(2)     if x<38
                    jmp       endball                                 ;3      don't make graphics
                    pcall     makeball                                ;422      create graphics
                    delay     (36*12) - (1+422) + 3                   ;         make section 40 pixels wide

endball             delay     12-RIGHTPAD_PHASE
                    mov       fsr,#RPAD_BUFFER                        ;2
                    mov       w,#8                                    ;1
                    pcall     memtovideo                    ;104

                    delay     RIGHT_SPACE + (TIME_IMAGE - (119+119+167 + RIGHT_SPACE + LEFT_SPACE + (BALLAREA_WIDTH *
12) + 4 + 2 + 9 + 12 + 1)) + RIGHTPAD_PHASE

                    decsz     temp7
                    jmp       playfield_l

                    delay     2
playfield_e         pcall     hsync
                    delay     12 - TEXTLINE_PHASE
                    mov       w,#((TIME_IMAGE-14) / 12) - 1
                    pcall     simplecolorfa
                    delay     TEXTLINE_PHASE + ((TIME_IMAGE-14) // 12)
                    pcall     hsync
                    delay     TIME_IMAGE-5

                    mov       temp7,#8
                    mov       temp4,#((gamedata + FONT) & $ff)
bots_l              pcall     hsync
                    delay     BTEXT_BASE-BTEXT_PHASE
                    mov       temp1,#((gamedata + STR2) >> 8)         ;2
                    mov       temp3,#((STR2 + gamedata) & $FF)        ;2
                    mov       temp5,#STR2_LEN                         ;1
                    pcall     strout
                    inc       temp4                                   ;1
                    delay     TIME_IMAGE-((STR2_LEN-1)*8*12+45+1)-BTEXT_BASE+BTEXT_PHASE-10-1
                    decsz     temp7
                    jmp       bots_l

                    delay     2
                    pcall     hsync
                    delay     TIME_IMAGE-1
                    pcall     hsync
                    delay     12 - TEXTLINE_PHASE
                    mov       w,#((TIME_IMAGE-32) / 12)-1
                    pcall     simplecolorfa
                    delay     TEXTLINE_PHASE + ((TIME_IMAGE-32) // 12)

                    mov       w,#POST_LINES-1
                    pcall     emptylines

                    delay     12-1
                    pcall     hsync

                    bank      $20                                     ;1
                    test      wave1timer                    ;1
                    sz                                                ;1(2)
                    jmp       nosoundch1                    ;3
                    dec       wave1timer                    ;5
                    sub16     wave1speed,wave1speeddif                ;6
nosoundch1r

                    test      wave2timer                    ;1
                    sz                                                ;1(2)
                    jmp       nosoundch2                    ;3

                    dec       wave2timer                    ;5
                    sub16     wave2speed,wave2speeddif                ;6
nosoundch2r


                    bank      $00                                     ;1
                    mov       temp0,bally_h                           ;2
                    sub       temp0,#(PAD_SIZE / 2) - (BALL_LINES / 2) ;2
                    sc                                          ;1(2)
                    clr       temp0                             ;1

                    sb        mixedbits.6                       ;1        coputer player ?
                    jmp       nocomputerlwarp                   ;3        no, don't play automatically

                    cjb       temp0,y1,nocup1                   ;4(6)     joy1 up simulated ?
                    csa       y1,#PLAYFIELD_LINES-PAD_SIZE-1;3(4)     yes, can paddle 1 move up ?
                    inc       y1                                ;1        yes, increase y-pos of paddle 1
nocup1              cja       temp0,y1,nocdown1warp             ;4(6)     joy1 down simulated ?
                    csb       y1,#1                             ;3(4)     yes, can paddle 1 move down ?
                    dec       y1                                ;1        yes, decrease y-pos of paddle 1
                    nop                                         ;1
nocdown1warpr
nocomputerlwarpr
                    sb        mixedbits.7                       ;1        coputer player ?
                    jmp       nocomputerrwarp                   ;3        no, don't play automatically
                    cjb       temp0,y2,nocup2                   ;4(6)     joy2 up simulated ?
                    csa       y2,#PLAYFIELD_LINES-PAD_SIZE-1;3(4)     yes, can paddle 1 move up ?
                    inc       y2                                ;1        yes, increase y-pos of paddle 2
nocup2              cja       temp0,y2,nocdown2warp             ;4(6)     joy2 down simulated ?
                    csb       y2,#1                             ;3(4)     yes, can paddle 1 move down ?
                    dec       y2                                ;1        yes, decrease y-pos of paddle 2
                    nop
nocdown2warpr
nocomputerrwarpr
                    delay     TIME_IMAGE - 74
                    pjmp      main

nocomputerlwarp     delay     3
                    jmp       nocomputerlwarpr

nocomputerrwarp     delay     3
                    jmp       nocomputerrwarpr

nocdown1warp        jmp       nocdown1warpr
nocdown2warp        jmp       nocdown2warpr

nosoundch1mov16l    wave1speed,0                      ;4        set soundspeed to zero to make channel silent
                    delay     3                                 ;3        delay to keep timing
                    jmp       nosoundch1r                       ;3        get back

nosoundch2mov16l    wave2speed,0                      ;4        set soundspeed to zero to make channel silent
                    delay     3                                 ;3        delay to keep timing
                    jmp       nosoundch2r                       ;3        get back

;************************* start ******************************
;* Start sequence, sets up system                             *
;**************************************************************

start               clr       fsr
clr_l               setb      fsr.4
                    clr       ind
                    incsz     fsr
                    jmp       clr_l

                    mov       fsr,#$70
clr_l2              setb      fsr.4
                    mov       ind,#black
                    incsz     fsr
                    jmp       clr_l2

                    mode      $F
                    mov       !RB,#%11000001
                    mov       !RC,#%11100000
                    mode      $E
                    mov       !RA,#%0000
                    mov       !RB,#%00111110
                    mov       !RC,#%00011111

                    bank $00
                    mov       w,#(PLAYFIELD_LINES / 2) - (PAD_SIZE / 2)
                    mov       y1,w
                    mov       y2,w

                    pjmp      initscreen


;************************* gamedata2 **************************
;* Only one palette in this gamedata section                  *
;**************************************************************

gamedata2

llevel = 0
REPT 8
          pal_secphase        =  BLACK
          pal_firstphase      = (BLACK + (((63-BLACK)*llevel)/7))
          dw pal_firstphase | ((pal_secphase << 8) & %111100000000) | ((pal_secphase << 2) & %11000000)
          llevel = llevel + 1
ENDR

llevel = 0
REPT 8
          pal_secphase        = (BLACK + (((63-BLACK)*llevel)/7))
          pal_firstphase      = 63
          dw pal_firstphase | ((pal_secphase << 8) & %111100000000) | ((pal_secphase << 2) & %11000000)
          llevel = llevel + 1
ENDR

PALETTE_BCW         EQU $0
PALETTE_PAGE        EQU (( gamedata2 + PALETTE_BCW)>>8)


;************************* gamedata ***************************
;* Game data: graphics, music, wavetables etc...              *
;**************************************************************
```

```
org $600

gamedata
dw $000,$000,$000,$000,$100,$300,$300,$100,$000,$000,$000,$07e,$000,$000,$000,$300        ; $000..$00f
dw $500,$600,$700,$700,$500,$118,$018,$000,$000,$018,$218,$500,$600,$718,$818,$800        ; $010..$01f
dw $83c,$666,$203,$003,$073,$166,$35c,$500,$60f,$706,$806,$806,$846,$866,$57f,$000        ; $020..$02f
dw $063,$267,$36f,$57b,$673,$763,$863,$800,$93e,$963,$763,$263,$163,$263,$33e,$500        ; $030..$03f
dw $63f,$766,$866,$a3e,$b06,$906,$80f,$400,$23f,$366,$366,$53e,$636,$766,$867,$b00        ; $040..$04f
dw $d3c,$a66,$90c,$618,$230,$366,$33c,$400,$663,$763,$863,$a6b,$c6b,$a7f,$936,$700        ; $050..$05f
dw $263,$363,$336,$41c,$536,$663,$763,$800,$900,$800,$81e,$730,$23e,$333,$36e,$300        ; $060..$06f
dw $500,$600,$73e,$763,$803,$863,$83e,$600,$238,$230,$33e,$333,$433,$533,$76e,$700        ; $070..$07f
dw $700,$800,$83e,$563,$17f,$203,$33e,$300,$43c,$566,$606,$71f,$706,$706,$70f,$300        ; $080..$08f
dw $000,$200,$26e,$333,$333,$43e,$530,$61f,$707,$706,$536,$16e,$066,$166,$267,$300        ; $090..$09f
dw $318,$300,$41c,$518,$618,$518,$33c,$000,$007,$006,$166,$236,$31e,$336,$367,$400        ; $0a0..$0af
dw $400,$200,$037,$07f,$06b,$06b,$06b,$100,$200,$200,$23b,$266,$166,$066,$066,$000        ; $0b0..$0bf
dw $000,$000,$13e,$363,$463,$263,$03e,$000,$000,$100,$43b,$666,$766,$73e,$306,$00f        ; $0c0..$0cf
dw $000,$200,$53b,$76e,$806,$806,$70f,$200,$10c,$30c,$53f,$70c,$80c,$a6c,$938,$500        ; $0d0..$0df
dw $200,$300,$533,$633,$933,$c33,$96e,$700,$200,$300,$563,$66b,$86b,$a7f,$936,$700        ; $0e0..$0ef
dw $230,$318,$43e,$663,$77f,$803,$83e,$700,$23e,$241,$359,$545,$645,$759,$741,$53e        ; $0f0..$0ff
dw $150,$260,$308,$440,$638,$630,$620,$300,$0f8,$1f8,$300,$348,$4a0,$570,$3a8,$068        ; $100..$10f
dw $0d0,$078,$200,$220,$2e0,$2b8,$0f0,$080,$000,$1e8,$3e8,$4e8,$5e8,$610,$6d0,$7a0        ; $110..$11f
dw $770,$7a8,$668,$6d0,$578,$410,$390,$1e0,$0b8,$f80,$d80,$c10,$b70,$ac0,$ab0,$900        ; $120..$12f
dw $958,$958,$aa0,$ab8,$bb8,$c80,$dd0,$f18,$000,$048,$0a0,$090,$098,$0d8,$000,$058        ; $130..$13f
dw $058,$0a0,$0b8,$0b8,$080,$0d0,$018,$000,$028,$080,$088,$0d8,$000,$000,$048,$048        ; $140..$14f
dw $0a0,$090,$098,$0d8,$018,$000,$028,$028,$080,$088,$0d8,$018,$000,$000,$098,$098        ; $150..$15f
dw $0e0,$0b0,$068,$0b8,$000,$000,$000,$000,$070,$070,$0c0,$0b0,$0c8,$0e0,$0d8,$080        ; $160..$16f
dw $0d0,$000,$000,$01c,$036,$063,$06b,$063,$036,$01c,$000,$018,$01c,$018,$018,$018        ; $170..$17f
dw $018,$07e,$000,$03e,$063,$060,$038,$00c,$066,$07f,$000,$03e,$063,$060,$03c,$060        ; $180..$18f
dw $063,$03e,$000,$038,$03c,$036,$033,$07f,$030,$078,$000,$07f,$003,$003,$03f,$060        ; $190..$19f
dw $063,$03e,$000,$01c,$006,$003,$03f,$063,$063,$03e,$000,$07f,$063,$030,$018,$00c        ; $1a0..$1af
dw $00c,$00c,$000,$03e,$063,$063,$03e,$063,$063,$03e,$000,$03e,$063,$063,$07e,$060        ; $1b0..$1bf
dw $030,$01e,$000                                                                         ; $1c0..$1c2


;*********************** vrealsound ***************************
;* vrealsound is called from vsync and calls realsound every  *
;* second vsync cycle as vsync is called at twice the speed   *
;* as sync, so vrealsound is dependent of realsound           *
;* clocks: 58                                                 *
;**************************************************************

vrealsoundsb        temp2.0             ;1(2)
                    jmp       realsound ;3 + 54
                    delay     56                  ;53
                    retp                          ;3

;************************ realsound ***************************
;* realsound is called from hsync to output sound data to the *
;* sound DA, the sound outputted is two cahnnels of sound     *
;* mixed together, the sound is based on a 16bit sinus signal *
;* in rom.                                                    *
;* clocks: 54                                                 *
;**************************************************************

realsound bank      $20                                     ;1
                    mov       m,#((SINTABLE+gamedata) >> 8)           ;1        point at corrent page for sintable
                    add16     wave1pos,wave1speed                     ;6        update  sintable  position  according  to
speed
                    and       wave1pos_h,#31                          ;2        keep sample position in range 0..31
                    mov       soundtemp0,wave1pos_h                   ;2        get high part i wave position
                    add       soundtemp0,#((SINTABLE+gamedata )& $FF) ;2        add low part of pointer to sintable and
position
                    mov       w,soundtemp0                            ;1        the sum, the low pointer should be in w
                    iread                                             ;4        read from rom
                    mov       w,m                                     ;1        get high nibble
                    mov       soundtemp0,w                            ;1        store in temporary register
                    mov       w,#$F0                                  ;1        set sign extend mask
                    snb       soundtemp0.3                            ;1(2)     check for sign bit
                    or        soundtemp0,w                            ;1        sign-bit was set, do sign extend
                    add       soundtemp0,#15                          ;2
                    mov       m,#((SINTABLE+gamedata) >> 8)           ;1        point at corrent page for sintable
                    add16     wave2pos,wave2speed                     ;6        update  sintable  position  according  to
speed
                    and       wave2pos_h,#31                          ;2        keep sample position in range 0..31
                    mov       soundtemp1,wave2pos_h                   ;2        get high part i wave position
                    add       soundtemp1,#((SINTABLE+gamedata) & $FF) ;2        add low part of pointer to sintable and
position
                    mov       w,soundtemp1                            ;1        the sum, the low pointer should be in w
                    iread                                             ;4        read from rom
                    mov       w,m                                     ;1        get high nibble
                    mov       soundtemp1,w                            ;1        store in temporary register
                    snb       soundtemp1.3                            ;1(2)     check for sign bit
                    or        w,#$F0                                  ;1        sign-bit was set, do sign extend
                    add       w,soundtemp0                            ;1        add the two channels
                    mov       audio,w                                 ;1        output to audio DA converter
                    bank      $00                                     ;1
                    retp                                              ;3
```

Appendix E: Game System PCB layout

The mirrored PCB-layout in scale 1:1, also including the component layout:




Audio
Video
100 Pot
Exp Mem
1pF
440
220
D-Sub 9p (M)
SX28
100nF
7805
220uF
Osc Jumper
Reset Switch
Osc
Osc1
Osc2
Vcc
Gnd
SX-Key
10k
LED
Power 9v