TS (Tile & Sprite) Configuration Programming Notes

Theodore (Alex) Evans

December 26, 2019

Contents

Chapter 1

Introduction

In my opinion, TSconf is a very modern add-on over the beloved ZX Spectrum, which brings long-awaited and necessary elements in the form of color to a point, hardware sprites, and so on. This will be discussed in this article.

Tile-sprite configuration (TSconf) can be divided into the following logical groups:

1. Graphic accelerator
    - Use of tiles
    - Output and control of sprites
2. Memory manager
3. Direct memory access unit (DMA)
4. Interrupt system
5. Cache
6. Ports management

Let us gradually consider all these points in order.

1.1 Graphic subsystem

Tao says: The screen we see is the output window of the resolution specified in the system, which displays a block of 512x512 video memory pages in the specified mode according to the given output coordinates. TSU is a device that collects data from video, tile and sprite memory, which processes and presents it at the current position of the line drawing in a given resolution / desired mode.

This unit is an extension (add-on) above the standard 6912 Spectrum screen. More precisely, the 6912 mode is part of the whole family of system permissions:

256x192
320x200
320x240
360x288

These permissions can be used in different video modes:

ZX
16c
256c
Text

To display the color for a given mode, the first block of the system's internal memory is used - the palette memory . This block represents 512 bytes and stores the full color palette of the system - two-byte data of color components for 256 colors.

The full palette is divided into groups of 16 colors (32 bytes) for 16c mode, which gives us 16 palettes with 16 colors.

The first palette is number 0.

When you turn on the system, a set of 16 colors for the ZX mode is loaded into the general palette, which is located in the last, 15th palette.

Each mode sets its own characteristics of displaying color information on the screen:

ZX - limits the output to 16 colors of the standard ZX Spectrum palette;
16c - in this mode only 16 colors are used - one of the 16 available palettes;
256s - provides the ability to display all the loaded colors in the palette
Text - text mode, allows you to display text in color.

In this case, it was a question of horizontal mobility. Let's talk about vertical:

In addition to the listed modes, the system allows the use of layered display of graphical information. The standard Spectrum screen is the base screen, but it is not the lowest.

So, the location of the layers that form the screen:

1. Border . It is a monochromatic fullscreen layer. Set the color of the curb.
2. Basic (main) screen. May be included in any resolution / mode of the above.
3. The layers of the graphic accelerator are displayed in the resolution specified for the base screen in 16 color mode:
   - sprite layer 0
   - tile layer 0
   - sprite layer 1
   - tile layer 1
   - sprite layer 2

Thus, we get 7 layers that make up a single, visible for the user screen.

All the listed layers use colors specified in a common palette of 256 colors.

Tile and sprite accelerator layers work only in 16c mode, using the 0th color of the palette set for them as transparent. Thanks to TSU, there is no need to save video memory data under tiles and sprites, since the display on the screen is literally collected at the output of each line without changing the contents of the video memory.

Each graphic element of these two types of layers represents a block of at least 8x8 pixels, and for each of them it is necessary to specify a palette, one of all of the available 16.

The tile layer is a map describing the location of the graphic elements specified as an image. Thus, the position of the tile on the screen directly depends on its position in the map.

Tile cardIt has a size of 64x64 tiles (4096 tiles in general), up to 4 palettes from 4 groups of all 16 palettes can be used for one layer. For each tile it is possible to set its own (of these 4) palette.

In total, we have two such layers organized in the same way.

Sprites are graphics, organized by the type of tile, but having large (multiple of 8) sizes - from 8x8 to 64x64 pixels. Sprites also have zero color transparency.

A feature of the sprites is that for each sprite, you can specify both your own palette and the position on the screen right up to the point.

To handle the sprites, the system uses the second block of internal memory

, representing the next 512 bytes of memory, which stores " sprite handles"-data from 6 bytes, describing each sprite. The maximum number of descriptors in this memory is 85 pieces.

In the following articles I will describe in more detail about working with the devices listed at the beginning of the article.

PS: for those who do not want to wait and want details

Chapter 2

DMA

Development

[74_New]Given such a large memory size, a fast data transfer facility is needed.

TSconf offers us such a tool that allows you to transfer data in memory without the participation of the processor.

Tao says: The transfer speed is 7 MHz, copying takes place two bytes (16 bits), provided that there is no access to the CPU, video, or a TCU during this clock cycle.

On average: 4 bytes - 2 clocks, dma refers to the RAM for 1 beat of 7 MHz, 16 bits, 2 calls are needed for transfer. We get : DMA speed 7 MB/s

DMA can copy data from the following sources: What can we use?

Src	Destination	
RAM	RAM	RAM (Src) is copied to RAM (Dst)
BLT	RAM	Pixels from RAM (Src) are copied to RAM (Dst) if they non zero
SPI	RAM	SPI data is copied to RAM (Dst)
RAM	SPI	RAM (Src) is copied to SPI
IDE	RAM	IDE data is copied to RAM (Dst)
RAM	IDE	RAM (Src) is copied to IDE
FILL	RAM	RAM (Dst) is filled with word from RAM (Src)
RAM	CRAM	RAM (Src) is copied to CRAM (Dst)
RAM	SFILE	RAM (Src) is copied to SFILE (Dst)

S_ALGN	Source Address Alignment
D_ALGN	Destination Address Alignment
0	After each burst address keeped as is
1	After each burst lower bits of address restored to their initial values before burst, upper bits increased by 1

A_SZ	Address Alignment Size
0	256 bytes (8 lower address bits) alignment
1	512 bytes (9 lower address bits) alignment

A_SZ	Blitting Bitness
0	4 bits per pixel, transparent color is 0 from 16 (4'b0)
1	8 bits per pixel, transparent color is 0 from 256 (8'b0)

Direct copying, copying mode for images with transparency (transparency is not copied - overlaying data over), copying from SPI devices (for example - sd-card), copying from the hard drive, filling the memory with the specified 16 bits (clearing the screen - also fill color) copying to the system's internal memory for the palette and for sprites.

Copying is done by filling in the ports of the DMA system:

- source data start address, source data page;
- address of the beginning of the receiver, the page for receiving data;
- the amount of data transmitted at one time (the length of the burst), the number of such transmission units (burst);
- setting the copy mode that starts the transfer

The maximum length for one transmission is 512 bytes (256 times 2 bytes), the maximum number of transmission blocks is 256. The maximum possible ONE data transfer is: 256 * 512 bytes = 131072 bytes for one copy.

Naturally, the length can be set different.

Considering the fact that the transmission takes place in 2 bytes (16 bits), at a minimum we can copy 2 bytes of data, that there are 4 points in 16c mode (one byte is 2 points), or 2 points in 256 colors mode. All addresses of operations are even.

Copy Modes: These bits are used to set system actions after copying a single data block (burst) of a given length:

- S_ALGN, D_ALGN: these bits are responsible for the "alignment" of the source / destination address. In the state 0, the address does not change, in the state 1- the address will be restored to the initial one with an increase in the high address depending on the A_SZ bit. This mode is useful to us when copying graphics.

- A_SZ - data alignment method: 256/512 bytes after each burst, which is useful for copying 4-bit (16 color) or 8-bit graphics

For more information about #27af (DMACtrl launch port):

DMACtrl R/W - S_ALGN D_ALGN A_SZ DDEV[2:0]

Here, bit 7 is R / W, the 6th bit is not used, 5 is source alignment, 4 is receiver alignment, 3 is alignment type (512/256), then 3 bits of the device.

Dao says: The address for DMA does not take into account the state of the two high-order bits of the 16-bit address. Accordingly, #c000 = #0000. You can use both #c000 and #0000 for the address - they are equivalent. When copying data, the location in the memory (address) for the source / receiver must be a multiple of 2.

So how do you manage all this goodness?

DMA ports must be programmed (#xxAF): High byte of source port:

#1Aaf, #1Baf - address (low DMASAddrL, high DMASAddrH bytes)

#1Caf - DMASAddrX start page High byte of receiver port

#1Daf, #1Eaf - address (low DMADAddrL, older DMADAddrH bytes)

#1Faf - DMADAddrX start page

Port of one burst transmission length: #26af, DMALen

Port of transmission burst number: #28af, DMANum Transmission

control / transmission status port: #27af, DMACtrl

Send data to this port #27af (DMACtrl) launches DMA transfer

Examples:

Copy the image of 256x256 pixels in 16c mode to the screen.

The original image is located at #c000 of the glasspat_page of the page, copied to the address of the #c000 of the screen page.

Let's get down to business in a smarter way than simply programming each port:

```
ld bc, #1aaf    ; DMASAddrL
xor a
out (c),a
inc b
out (c),a
```

So, we copy the graphics with alignment (moving to the next line on the screen) after each burst.

We note that A_SZ is turned off, which is necessary for 16 colors:

```
                ld hl,glasspat_copy
                call set_ports
                ret

glasspat_page   equ #22
Vid_page        equ #80

glasspat_copy   db #1a,0
                db #1b,0
                db #1c,glasspat_page
                db #1d,0
                db #1e,0
                db #1f,Vid_page
                db #26,256/4-1
                db #28,256-1
                db #27,DMA_RAM + DMA_DALGN
                db #ff
```

To do this, use the routine to issue data to the port (set_ports):

```
set_ports       ld c,#AF
.m1             ld b,(hl)
                inc hl
                inc b
                jr z,dma_stats
                outi
                jr .m1

dma_stats       ld b,high DMASTATUS
```

```
                in a,(c)
                AND #80
                jr nz,$-4
                ret
```

After setting the value for port 27 the transfer starts. We need to control the employment of DMA, for this we use reading this same port for a busy signal (DMA_ACT bit). Setting this bit to 0 indicates that the transfer is complete.

Cleaning (fill) screen:

```
                ld bc,PAGE3
                ld a,Vid_page
                out (c),a
                ld hl,0         ;00 - color set in the palette
                ld (#c000),hl
                ld hl,fill_screen
                jp set_ports

fill_screen     defb #1a,0      ;
                defb #1b,0      ;
                defb #1c,Vid_page       ;
                defb #1d,0      ;
                defb #1e,0      ;
                defb #1f,Vid_page       ;

                defb #28,200    ;
                defb #26,#ff    ;
                defb #27,%00000100    ; DMA_FILL
                db #ff
```

In this case, we include the video page from the address #c000 and fill it with the color we need (in this case, color No. 00), after which we start the fill. Copying occurs from the address 0000 (remember that the system is on the drum as the state of the older two bits and one of the youngest) is filled with a length of 256 * 2, 200 bursts.

Cleaning the tile layer:

```
clear_tileset   ld bc,PAGE3
```

```
                ld a,Tile_page
                out (c),a
                ld hl,0
                ld (#c000),hl
                ld hl,tileset_clr
                jp set_ports
tileset_clr
                defb #1a,0      ;
                defb #1b,0      ;
                defb #1c,Tile_page      ;
                defb #1d,0      ;
                defb #1e,0      ;
                defb #1f,Tile_page      ;
                defb #28,#ff    ;
                defb #26,#3f    ;
                defb #27,%00000100
                db #ff
```

Copying the palette to the palette memory:

```
                ld hl,pals
                jp set_ports

pals            db #1a,low pal
                db #1b,high pal
                db #1c,2
                db #1d,0
                db #1e,0
                db #1f,0
                db #26,#10
                db #28,0
                db #27,DMA_RAM_CRAM     ; #84 - copy from RAM to CRAM
                db #ff

                align 2
pal             incbin "palette.tga.pal"
```

In this case, copy 16 colors (one palette), from the second page to the beginning of the palette memory.

[Question-M]Questions

:? Why is there DMA_DALGN?
! Since the image is 256 pixels wide, after the last 256 points we have to place a new piece of data from a new line. This bit will help us.

? Why is the DMANum port (#28af) such a strange value: 256-1?
! Considering that the maximum number of burst is 256, and the minimum is 1, it is assumed that the value sent to the port 0 is the number of burst 1 piece, and the sent value 255 is respectively 256 bursts.

? Why is the DMALen port (#26af) given such a strange value: 256 / 4-1?
! Since we have an image size of 256 points in 16-color mode, we get 2 points in one byte, and 4 points in two bytes. Again, the value 0 for this port is the minimum length of 2 bytes.

? Is it worth waiting for the end of the transfer in the read loop of the DMASTATUS port?
! If you are sure that the shipment will have time to work out before the start of a new one - it is not necessary at all. Moreover, it is recommended that the waiting cycle be set BEFORE a new DMA transaction, so that the z80 continues its computational work while the DMA fulfills the task.

? I sent the palette, but it is not really that color at all!
! Check the location of the palette - it should be stored in memory at an even address. ALIGN 2 to help you. Or, perhaps, not there copied?

? Picture sent to the screen, and he moved down the lines!
! The size of the picture should be a multiple of 2m bytes - 2 points (256 c), 4 points (16 c)

? and DMA when copying the page itself switches?
! Of course! And the source and receiver.

Chapter 3

Graphics

Development

[Big-Bang-Television-Static-590x353]Sprites, tiles ... Perhaps this would be enough for us ...

But under these layers is the base graphics layer .

Tao says: The graphics layer displays data that resides in memory pages.

The first page to display (its address is necessarily a multiple of 8 for 16 colors, 16 for a 256 color mode, total length - 8/16 pages) is indicated by the port VPage (#01af). The way this memory is mapped is set by the VConfig port bits, which sets the resolution and color depth.

The memory display window is a block with the dimensions of the specified resolution and is displayed by the positions X (0-511) and Y (0-511), which are indicated by the pairs of ports GXOffs and GYOffs. The window is looped around the edges in the display.

The colors of the display are given by the palette, the number is selected by the first notebook of the PalSel register (#06af).

So, we have a screen with our own internal scrolling.

The output to the screen is a data record in the video memory page at the desired address.

Depending on the selected color mode, recording one byte allows you to output one (256 colors) or two (16 color mode) dots at a time. In the 16 color mode, the highest tetrad is responsible for the color number in the

RRES[1:0]	Pixels
00	256x192
01	320x200
10	320x240
11	360x288

VM[2:0]	Mode
00	ZX
01	16c
10	256c
11	Text

palette for the left point, the youngest - for the right one.

In 256 color mode - the byte of the dot indicates the color in the palette.

What is the choice of resolution? All this is set by the VConfig port bits (#00af):

VConfig RRES[1:0] NOGFX NOTSU - - VM[1:0]
RRES - 7,6 bits - resolution
VM[1:0] 1, 0 bits - color mode

In addition, this port offers us a bit to enable / disable graphics display and the bits on / off of the sprite-tile engine.

In the first case - when turning off the graphics, we will see a clean screen of the color specified by the border. In the second - sprite and tile layers will not be displayed.

Disabling unused layers leads to the release of system resources and will allow, for example, to transfer more DMA data per line.

Let's display something on the screen.

The easiest way, I think, is to organize the output of a picture using DMA - let the iron work, the processor thinks at this time, and the person is resting :)

So, we will display the image in 16 color mode, 256x256 pixels in size.

To do this, you must prepare the ports, indicating the desired resolution, color gamut, address of the video memory page:

```
ld hl,init_ts
```

```
                jp set_ports

Vid_page        equ #40

init_ts         db high VCONFIG,VID_16C+VID_320X240
                db high VPAGE,Vid_page
                db high PALSEL,0

        include "tsconfig.asm"
```

Note that the palette for graphics has the number 0. I recall that the standard initial palette of the zx spectrum is located in the last, 15th palette.

```
                ld hl,pic_copy
                call set_ports
                jr $
screen_page     equ #80 ; page number in which the image is stored for output
pic_copy
                db #1a,0
                db #1b,0
                db #1c,screen_page
                db #1d,0
                db #1e,0
                db #1f,Vid_page
                db #26,256/4-1
                db #28,256-1
                db #27,DMA_RAM + DMA_DALGN
                db #ff
```

Everything, the image is on the screen, starting from the 0x0 coordinate.

It's very simple to move this image around the screen by setting indents by programming ports of offsets for the base graphics layer - GXOffs / GYOffs (#02af-03af/#04af-05af):

```
offs1           ld a,0
                inc a
                and #7f
                ld (offs1+1),a
                ld bc,GXOFFSL
```

```
out (c),a
ld bc,GYOFFSL
out (c),a
halt
jr offs1
```

In this case, the offsets are written for the lower 8 bits, and the offset is in the range of 0-127.

[Question-Mark-Dude1-300x300]Questions and Answers

:? Why on the screen after turning on the mode and selecting the video page a color NOISE?
! The initial contents of the memory have a random nature, which is displayed on the screen. Clean up!

? What is the address of the point y = 1?
! Depending on the selected color mode:
With 16 colors: y = 0: #c000; y = 1: #c100
With 256 colors: y = 0: #c000; y = 1: #c200

? I do not understand what is displayed!
! The screen start page should be a multiple of 8. Should the palette be loaded and selected by the PalSel port

? Only a narrow bar is displayed!
! Depending on the selected mode, up to 262,144 bytes of memory are used, which does not fit into 16k at all. Switch the video page and draw on!

Chapter 4

Interrupts

Development

[14_1] TSconf has an extended system of interrupts that can be triggered if there are such states as: beam arrival at a given screen position, beam arrival at the beginning of a line, line display on the screen, completion of data transfer.

Tao says: The system has three types of masked interrupts that can be called at an address that has a high byte — the address in register I, and the youngest — its type:

1. #FF - frame
2. #FD - lowercase (Line)
3. #FB - DMA.

The processing of these interrupts can be switched by the INTMask port (#2A af), changing the state of the bits:

0 - Frame, 1 - Line, 2 - DMA, which leads to an on / off call to handlers.

State of bits: 0 - disabled / 1 - enabled.

In the case of the arrival of several events at the same time, the interrupt with the lower number will be processed first.

How can we manage these capabilities? Consider in order:

Frame A

frame interrupt in the standard Spectrum comes at the very beginning of the screen drawing. For TSconf, this parameter can be set using three ports:

- VSINT: the high and low bytes of these two ports (VSINTH - #24af, VSINTL - #23af) allow you to specify the vertical position of the beam, which allows us to specify the entire area of the screen lines: 0 - 319;
- HSINT: port indicating the horizontal position of the beam (position in the line): 0–223 in 3.5 MHz cycles,

If the position of the beam coincides, a frame interrupt handler will be called. If you specify values outside the specified range, the interrupt will not be generated.

So let's give an example in which the frame interrupt handler #BE will be installed:

```
im2_init
                xor a
                ld bc,HSINT
                out (c),a
                ld bc,VSINTL
                out (c),a
                ld bc,VSINTH
                out (c),a

                ld a,#be
                ld i,a
                ld hl,int
                ld (#beff),hl
                im 2
                ret
```

In this case, the positions for calling the frame interrupt handler were set to 0 (upper left corner of the screen, leftmost position). When the ray of the screen rendering arrives at this position, the transition will be made to the personnel interrupt handler code by the int tag.

In this case, this handler will be called with a frame pulse frequency of 50 Hz.

And if we need more?

Consider the situation when we need to call the personnel handler at the beginning of the screen, and at the beginning of the line at number 128.

Why? So that the upper part of the screen was colored red!

To do this, we will need to specify when we need to call the handler in each next time. In the handler, the position of the screen 0 (the first line of the screen) will indicate in which place of the screen the following personnel interruption will be triggered:

```
int
                push af
                push bc
                push hl
                ld a,128
                ld bc,VSINTL
                out (c),a
                xor a
                ld bc,VSINTH
                out (c),a
                ld hl,int128
                ld (#beff),hl
                ld a,#f2
                ld bc,BORDER
                out (c),a
                pop hl
                pop bc
                pop af
                ei
                ret
```

Here we indicate that the next personnel interrupt should be called in the vertical position of the beam on line 128, and the handler will be the int128 subroutine. Set the red color from the standard zx spectrum palette (palette #0f, color 2 - for the border color, simply specify the desired color from the entire 256-cell palette).

When the beam hits the beginning of line 128, the subroutine int128 is called:

```
int128
                push af
                push bc
                push hl
```

```
xor a
ld bc,VSINTL
out (c),a
ld bc,VSINTH
out (c),a
ld hl,int
ld (#beff),hl
ld a,#f0
ld bc,BORDER
out (c),a
pop hl
pop bc
pop af
ei
ret
```

in which we indicated that the next personnel interruption will be processed by the int subroutine at the beam position 0, and set the border color to 0 of the standard palette.

We note that the HSINT horizontal position port does not change now and is set to 0, therefore the call to the handlers is always at the beginning of the line. I suggest you play with it yourself :)

Great, we can already get to the right place on the screen. But what if you need a certain subroutine to execute EVERY line? To do this, we have a tool in the form of a string interrupt.

Line interrupts

This type of interrupt will call its handler when the beam is at position 0 of each line of the screen (whether it is visible or not on the screen is not important for the system).

To use it, you must do the following:

- set the address of the line interrupt handler
- allow INTMASK port processing (#2Aaf)

So, turn on the bits of the INTMASK port, which allow processing of both personnel and line interrupts, set the address of the handler:

```
ld      hl,line_proc
ld      (#befd),hl
```

```
                ld      bc,INTMASK
                ld      a,%00000011
                out     (c),a
```

We note to ourselves that both personnel and line interrupts are now allowed. The first handler will be processed with a smaller number - i.e. first personnel, then lower case.

Change the task. Let each line on the border be visible strip of a different color.

Disable the frame handler, leaving only the lowercase one:

```
                ld      hl,line_proc
                ld      (#befd),hl

                ld      bc,INTMASK
                ld      a,%00000010
                out     (c),a
```

Add a line interrupt handler:

```
line_proc
                exx
                ex af, af'
lp1             ld a,#ff
                inc a
                ld (lp1+1),a
                or #f0
                ld bc,BORDER
                out (c),a
                ex af, af'
                exx
                ei
                ret
```

Since this handler will be called very often, it is worth minimizing the processor clock used by switching to an alternate set of registers. Then we increase the color number counter in the palette and send it to the system (border port #0Faf).

The peculiarity of these interrupt systems is that using the HALT “stop the processor” command so familiar to us will only wait for the next personnel / line interrupt, that is, for the first example, drawing a full screen takes 2 HALTs and 320 HALT for a lowercase :)

DMA

Handler to complete the transfer will be called after the transfer is completed. Your KO ;)

Turn on its processing and specify the address of the handler. As usual:

```
ld      hl,dma_proc
ld      (#befb),hl

ld      bc,INTMASK
ld      a,%00000100
out     (c),a
```

Note that in this case, we have completely disabled frame and line interrupts, and their handlers will NOT be called.

[Question-Mark-Dude1-300x300]Questions and Answers

:? Cho music when lowercase spoils?
! Alas, the standard pt3 player uses the stack in operation, and when calling the line processor, the data at the address in the SP is replaced with its call address. And spoil the muse data.

Accordingly, we cause the player to turn off the lowercase ones (for example, outside the screen, set up personnel), or use the psg player. ... or playing a non-stackable player.

? And how do you play with fullscreen effects?
! 1) Heh, lowercase interrupts are called every line. Accordingly, we obtain a lower frequency frequency of 15625 hertz, and each line is thrown into the audio coking port. Naman? And do not forget the previous question
2) The case, if only the 15625 sound, and the personnel interruption is long enough, then allow (nested) interrupts in the personnel handler. Plus, you can move the personnel int (for example) to the middle of the line, so that they would not come at the same time as the lower case.

List of Figures

List of Tables