# Let’s Make a Retro Game

Episode 14 -TMS9918A Graphics

In this episode we are going to cover some more technical details on some of the platforms we are targeting, specifically how graphics are handled on the following systems:

- Colecovision
- MSX
- Spectravideo
- Sega SG-1000/SC-3000

This information can also cover:

TI-99/4A, Memotech MTX, Creativision (Dick Smith Wizzard), Einstein, Sord M5

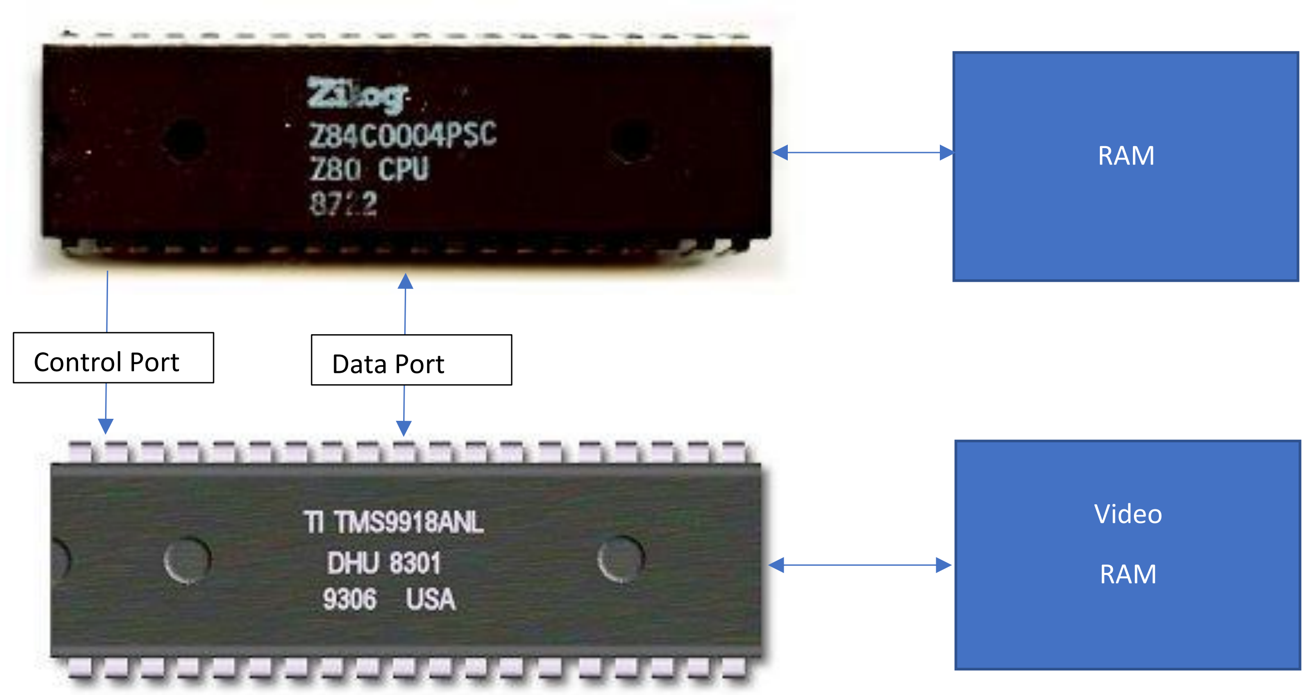
Each of these systems have a main processor (all bar one uses a Z80A CPU) with their own RAM, but this processor does not handle anything to do with creating output on the screen i.e. graphics. To do that all these systems actually have a 2nd CPU from Texas Instruments called the TMS9918A.

The TMS processor has its own RAM area, often called VRAM (for Video Ram), key points to understand how this works are:

- The TMS Processor has direct access to only to its own RAM, and
- Likewise, the Z80 only has direct access to its RAM

So unlike other machines, like the Commodore 64 and Spectrum you can’t just write a value to a memory location and it will change what happens on the screen.

So, for your code to control what is happening on the screen, the TMS processor has a number of ports (think of them as serving windows) that you use to send information to or receive information from the video ram.



This may seem strange, especially to programmers of other systems, but it is actually modelled on how a lot of arcade games were created, with different processors controlling different things, like video and sound.

The advantages of this are:

- The main processor i.e. the Z80 does not have to do anything to draw the screen each frame.
- The main processor does not have to share it’s Ram with anything so there is no delay accessing the Ram at any stage.
- The video ram is completely dedicated to the TMS processor, and with 16k reasonably high resolution (for 8-bit standards), colour and of course 32 hardware sprites, very faithful recreations of early arcade games are possible.

The disadvantages of this are:

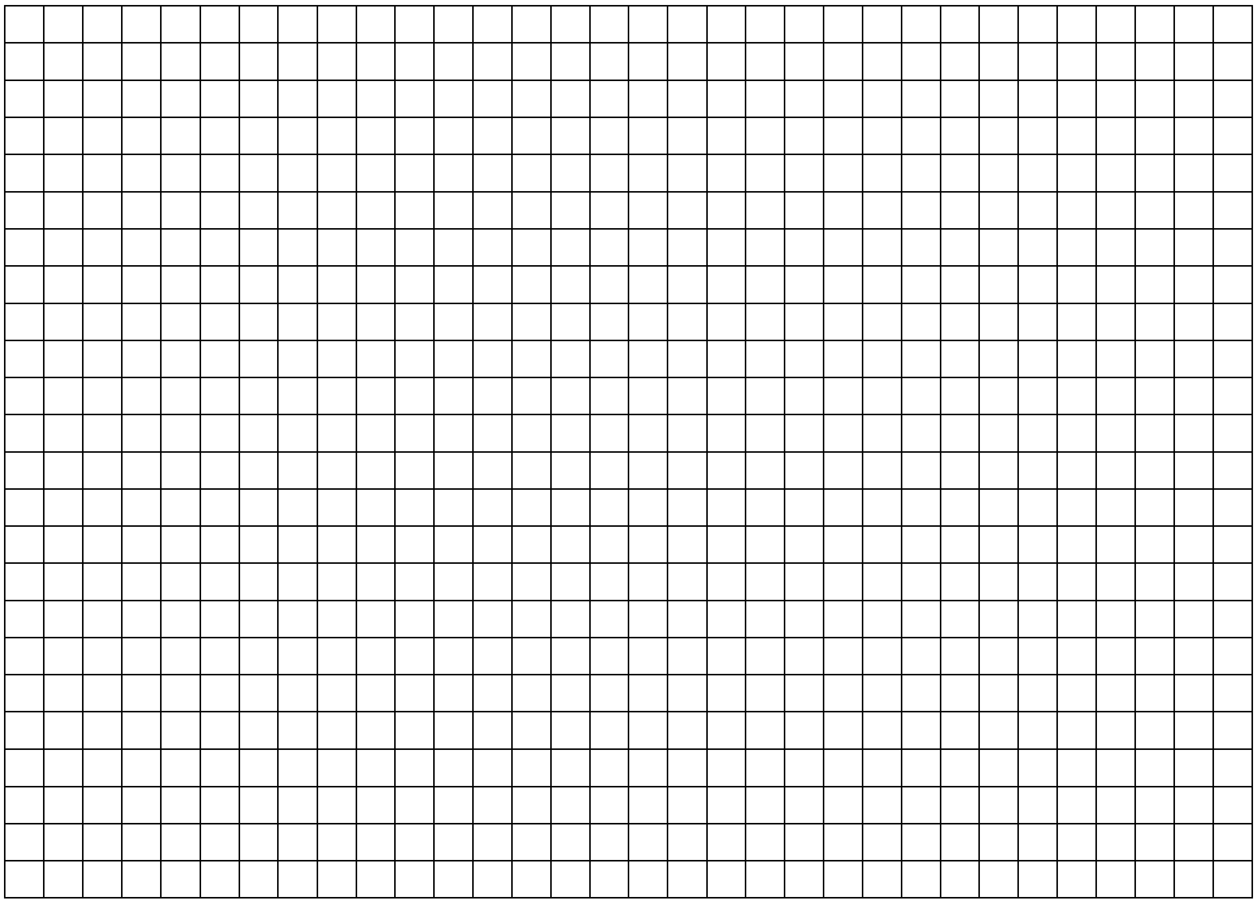
- There are limits to how fast information can be sent to or read from video using the ports to the TMS processor, so you can’t change large amounts of video ram each frame. This is why scrolling can be difficult to achieve on TMS based systems.

## Video Ram Break Down

The TMS processor actually has a number of different modes available as follows:

### Graphics Modes I & II

These two modes have a screen resolution of 256 x 192 pixels, broken up into 8 x 8 pixel tiles i.e 32 x 24 or 768 tiles.



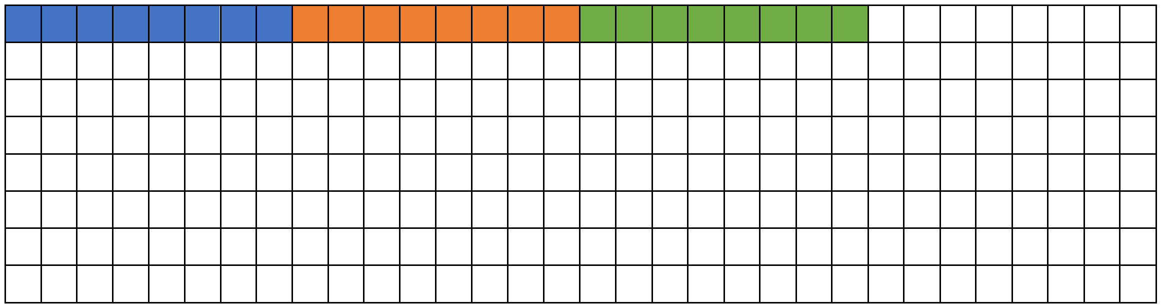
Each tile has 8 bytes of Ram that define the pattern or shape e.g.

```
00011000
00111100
01111110
11011011
11111111
00100100
01011010
10100101
```

Graphics Mode I only has one table of 256 tiles that are used for the whole screen, whereas Mode II has 768 tile patterns, so each tile can be unique.

How each mode handles colour is different as well.

- Mode I groups each of the tile patterns into blocks of 8, each of these blocks have the same foreground and background colour.



- Mode II has another 8 bytes for each pattern. Each of the bytes describes the foreground and background colour for a row of the pattern.

The two modes seem similar but differ greatly in the amount of colour and pattern detail that can be displayed.

Mode I harks back to when memory was more expensive, and if the TMS chip is only supplied with 4K of Video Ram.

Mode II uses 12k just for the patterns and colour information, it needs more to hold the pattern positions and sprite table.

Both of these modes support the 32 hardware sprites.

## Text Mode

Text Mode is just that designed purely to display text, and can display 40 columns across 24 rows, a total of 960 tiles. The patterns for each of the tiles come from a table of 256 patterns.

Each tile has 8 bytes of Ram that define the shape but only the most significant 6 bits (the right 8 bits) define the pattern e.g.

```
XX001100
XX010010
XX010010
XX100001
XX111111
XX100001
XX100001
XX100001
```

The positions marked with an X are ignored.

There is only a single foreground and background colour for the whole screen and sprites are not available in this mode.

### Multicolour Mode

The Multicolour Mode is quite different from the other modes in that it allows unrestricted use of the 16 available colours, but at a much lower resolution of 64 x 48 pixels.

Sprites are also available in this mode.

## Hardware Sprites

In all modes (other than text mode), 32 hardware sprites are available, these sit in order in front of the graphics/background layer.

All sprites can be either 8x8 or 16x16 pixels in size, plus also can be displayed double size i.e. the pixels are doubled in size.

Each sprite can be a different one of the 15 colours. Multi-coloured sprites are created by stacking multiple sprites on top of each other.

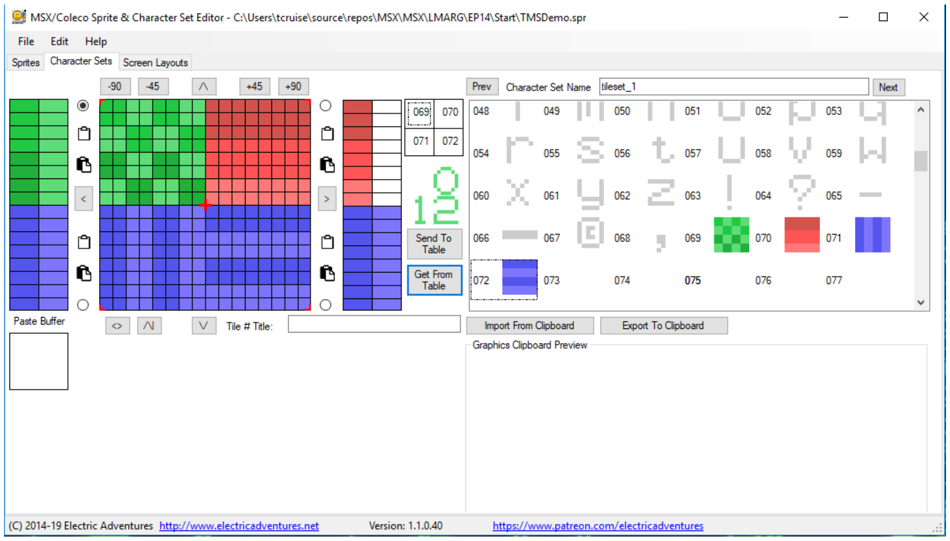
There can be a maximum of four sprites displayed in a row on the screen, but with some tricks up to eight in a row can be achieved with minor flickering.

These hardware sprites, along with the high resolution of the background tiles, are what allow TMS systems to cover early 80’s arcade games so well.

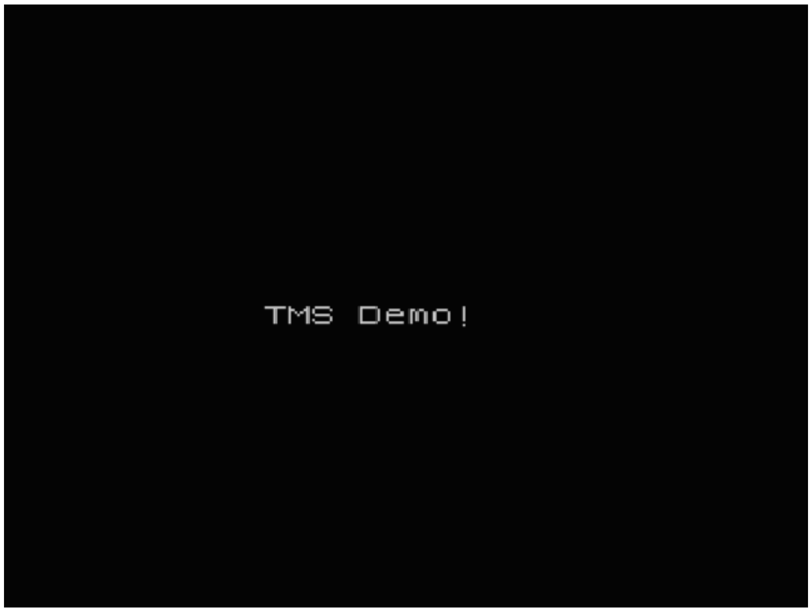
## Example Code – TMS Demo

For this episode we will not advance the game we have been working on but instead have a look at some examples of what the TMS graphics chip can do i.e. a demo.

In our starting directory we have our base template for each system, along with a set of tile patterns to get us started.


MSX/Coleco Sprite & Character Set Editor - C:\Users\tcruise\source\repos\MSX\MSX\LMARG\EP14\Start\TMSDemo.spr
File
Edit
Help
Sprites
Character Sets
Screen Layouts
Prev
Character Set Name
tileset_1
Next
Send To Table
Get From Table
Paste Buffer
Tile # Title:
Import From Clipboard
Export To Clipboard
Graphics Clipboard Preview
(C) 2014-19 Electric Adventures
http://www.electricadventures.net
Version: 1.1.0.40
https://www.patreon.com/electricadventures

Compiling and running this starting template in our emulator should get you a simple start-up screen with a title as follows:



Let’s add some things to our template, nothing fancy just a couple of items that will help visualise how the TMS chip uses its video memory to draw the display and some simple techniques to add some nice visual effects.

## Animating a tile pattern

For the top 256 tile positions of our screen we are going to set them all to be the same pattern (in this case tile pattern 69). Add this section of code after the label TITLESCREEN: and after the highlighted text below:

```
    ; now setup our initial screen layout
    ; top 1/3 of the screen
    LD HL,VRAM_NAME
    LD BC,256
    LD A,69
    CALL FILVRM
```

That’s not that interesting so let’s add some animation, add this code to our interrupt routine:

```
; This is our routine called every VDP interrupt during the title
screen
; - Do all VDP writes here to avoid corruption
OUTPUT_VDP_TITLE:
    ; do our pattern and colour animations
    LD A,(WAIT)
    CP 0
    JR Z, OVT1
    DEC A
    LD (WAIT),A
    RET
OVT1:
    LD A,8
    LD (WAIT),A
    ; 1. animate the pattern in tile 69
    LD HL,69*8
    CALL SETWRT
    LD B,4
    LD A,(LASTPATTERN1)
LP1:
    OUT (DATA_PORT),A
    OUT (DATA_PORT),A
    XOR 0ffh
    DJNZ LP1
    XOR 0ffh
    LD (LASTPATTERN1),A
    RET
```

We need to reserve two bytes of ram for our counter and last pattern storage:

```
ORG RAMSTART

WAIT: ds 1
LASTPATTERN1: ds 1
```

And it’s always good to initialise it to a known value when we start, in our INITRAM function:

```
; Initialise any RAM we will be using
INITRAM:
    LD A,204
    LD (LASTPATTERN1),A
    LD A,8
    LD (WAIT),A
    RET
```

This will change the pattern being displayed for our tile and thus change all 256 positions at one time, compile it and give it a go.

You don’t have to just work with a single pattern with a bit of planning you can get quite a nice detailed animation going with more tiles e.g. 4 tiles arranged 2 x 2.

## Animating the colour table

For the middle 256 tile positions of our screen, I used tile pattern 70, which has changing bands of colour (different shades of red) running down it. Plus, I added a border around our text using tile pattern 72 as follows:

```
    ; middle 1/3 of the screen
    LD HL,VRAM_NAME + 256
    LD BC,256
    LD A,70
    CALL FILVRM

    ; our initial text
    LD HL,VRAM_NAME + 32*12 + 10
    CALL SETRD
    LD HL,TITLE_TEXT
    CALL OUTPUT_TEXT

    ; add a box of characters around the text
    LD HL,VRAM_NAME + 256 + 32*2 + 6
    LD BC,20
    LD A,72
    CALL FILVRM
    LD HL,VRAM_NAME + 256 + 32*3 + 6
    CALL SETWRT
    LD A,72
    OUT (DATA_PORT),A
    LD HL,VRAM_NAME + 256 + 32*3 + 25
    CALL SETWRT
    LD A,72
    OUT (DATA_PORT),A
    LD HL,VRAM_NAME + 256 + 32*4 + 6
    CALL SETWRT
    LD A,72
    OUT (DATA_PORT),A
    LD HL,VRAM_NAME + 256 + 32*4 + 25
    CALL SETWRT
    LD A,72
    OUT (DATA_PORT),A
    LD HL,VRAM_NAME + 256 + 32*5 + 6
    CALL SETWRT
    LD A,72
    OUT (DATA_PORT),A
    LD HL,VRAM_NAME + 256 + 32*5 + 25
    CALL SETWRT
    LD A,72
    OUT (DATA_PORT),A
    LD HL,VRAM_NAME + 256 + 32*6 + 6
    LD BC,20
    LD A,72
    CALL FILVRM
```

It not only looks nice it also shows the higher resolution colour abilities of the TMS chip.

Now even though that doesn’t look too bad, lets jazz it up by animating the colour bands for tile pattern 70, by adding code to our interrupt routine before the RET statement as follows:

```
    ; 2. animate the palette entries on the 2nd zone
    LD HL,VRAM_COLOR + 0800h + 70 * 8
    CALL SETWRT
    LD A,(LASTPATTERN2)
    INC A
    CP 3
    JR NZ,OVT2
    XOR A
OVT2:
    LD (LASTPATTERN2),A
    LD HL,COLOURTABLE
    LD C,A
    LD B,0
    XOR A
    ADC HL,BC
    LD A,(HL)
    OUT (DATA_PORT),A
    OUT (DATA_PORT),A
    OUT (DATA_PORT),A
    INC HL
    LD A,(HL)
    OUT (DATA_PORT),A
    OUT (DATA_PORT),A
    OUT (DATA_PORT),A
    INC HL
    LD A,(HL)
    OUT (DATA_PORT),A
    OUT (DATA_PORT),A
    RET
```

Just below that add a little bit of data that will control which colours we will cycle through as follows:

```
COLOURTABLE:
    DB 096,128,144,096,128
```

We need to reserve a byte of ram for our counter:

```
ORG RAMSTART

WAIT: ds 1
LASTPATTERN1: ds 1
LASTPATTERN2: ds 1
```

And once again initialise it to a known value in our INITRAM function:

```
INITRAM:
    LD A,204
    LD (LASTPATTERN1),A
    XOR A
    LD (LASTPATTERN2),A
```

## More colour table animation

For the last 256 tile positions of our screen, I used tile pattern 71, which has vertical bands as follows:

```
    ; last 1/3 of the screen
    LD HL,VRAM_NAME + 512
    LD BC,256
    LD A,71
    CALL FILVRM
```

Let’s add one more piece of code to our interrupt routine as follows:

```
    ; 3. animate the palette entries in the 3rd zone
    LD A,(LASTPATTERN3)
    ; swap upper and lower number by rotating 4 times
    RLCA
    RLCA
    RLCA
    RLCA
    LD (LASTPATTERN3),A
    LD HL,VRAM_COLOR + 01000h + 71*8
    LD BC,8
    CALL FILVRM
    RET
```

We need to reserve a byte of ram for our counter:

```
ORG RAMSTART
```

```
WAIT: ds 1
LASTPATTERN1: ds 1
LASTPATTERN2: ds 1
LASTPATTERN3: ds 1
```

And once again initialise it to a known value in our INITRAM function:

```
    LD A,045h
    LD (LASTPATTERN3),A
```

So now by swapping the two colours the tile uses we can introduce an animation, that makes it look like that section of the screen is scrolling (although depending on how your eyes see it, it will be either going left to right or right to left ☺).

So now we have ended up with a very simple demo that demonstrates some techniques that can really jazz up your games on TMS systems. In our next part we will have a proper look at how the TMS processor handles sprites.