Model I No-Chip VBLANK Modification

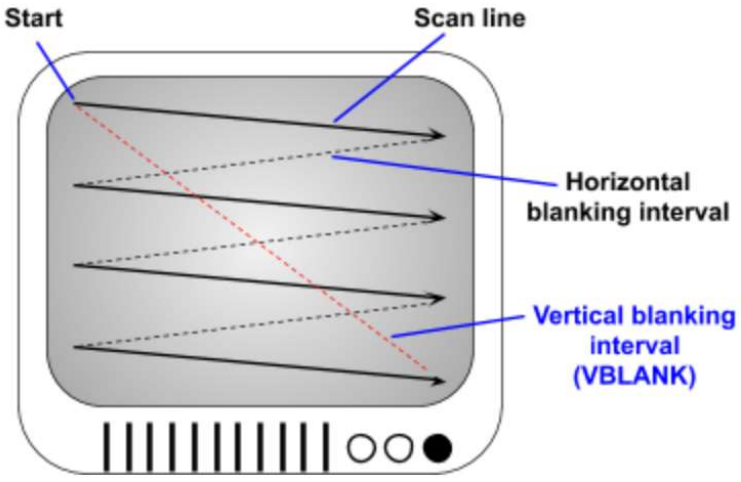
Tim Halloran

At Tandy Assembly 2019 we demonstrated a no-chip hardware modification to the TRS-80 Model I which allows software visibility to the start or end of the vertical blanking interval (VBLANK) by examination of bit `0` on port `$FF`. VBLANK detection allows for graphics without interference or tear as well as up to 192-pixel vertical resolution. The Model I could have shipped with this at absolutely no extra manufacturing cost. The mod was created by me from ideas by George Phillips. With a special thanks for the motivation to *Trash Talk* episode 21.

In this article we discuss the background and development of the VBLANK mod and present how to make it to a Model I. This mod is not immediately useful, it requires software to do anything. We present how to write software to dynamically detect and use it. Software development for this mod is aided by `trs80gp` which emulates it (with the `-m1_vblank` flag). We conclude with a short demonstration program you can try out on `trs80gp`.

Background

How did a new Model I mod come into existence in 2019? I had been programming z80 assembly for about a year and was having problems producing smooth animations on the Model I. I was using double buffering, where I draw into a memory buffer then transfer to video memory. The *transfer-to-video-memory* part was the problem. It needs to be done during the vertical blanking interval. The Model I draws the screen using 192 scan lines in a manner like that shown in the figure below (which only has 4 scan lines).



The *vertical blanking interval* (VBLANK) is the time between the end of the final scan line and the beginning of the first scan line. On a TRS-80 this goes from the bottom-right to the top-left of the screen (the red-dashed line in the figure).

Why draw my buffer to the screen during VBLANK? Because if you get it all into video memory before the beam starts the first scan line graphics animations are

smooth and tear-free to the eye. Sadly, VBLANK is not possible to detect on the Model I. I knew because I had asked George Phillips who had invented TRS-80 Beamhacking [4]. If it were possible George would have known.

Later I listened to Trash Talk episode 21 [1] where the paper *Tachistoscopic timing on theTRS·80* [3] was discussed during the topic *TRS-80s in Scientific Research*. I read the paper and asked George about VBLANK on the Model I yet again. No luck, it was a hardware mod. Thinking I might be able to discover how to do the mod, I reached out to the paper's author Robert Mapou who was amazed anyone was reading this in 2019, but sadly, had not done the mod—only used it. The trail to discover the mod created in the early 1980s was cold.

I started looking at the schematic and thinking about how the 1980s mod might have been done, keeping up an email discussion with George. An interrupt like the Model 3 would be best—but hard. George had an idea.

> Looking at the schematic I think **VDRV** would be the correct signal for VSYNC. On a Model I I'd be tempted to hook it up to one of the unused bits when reading port $FF.

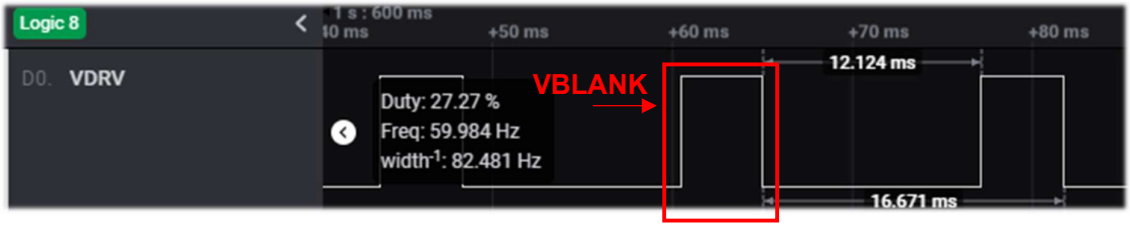
This describes the VBLANK mod in a nutshell (I choose bit `0`). Except for figuring out how to make this work inside a real Model I. That is what I worked on.

Development of the modification

With some email help and chips from Ian Mavric I had just restored 5 non-functional Model I computers to good working order. I had a moderate understanding of the motherboard and had spent many hours reading the technical reference manual when I started to work on the VBLANK mod.

Finding VBLANK inside the Model I

The first task was to look at **VDRV**. Does it really allow us to identify the start and end of VBLANK? Yes, it does. As seen below in a trace from a real Model I.



George expected **VDRV** frequency should be ~16.666 ms per cycle or 60 cycles per second. The high portion is of interest because that is VBLANK. The quarter of a cycle we observe is right for VBLANK. The display is 264 lines of which 72

of them are the vertical blanking interval which is ~27.3%. Clearly this is the signal used by the 1980s mod.

VBLANK Visibility to a I/O port (in)

According to *Tachistoscopic timing on theTRS·80* [3] the 1980s mod used port `$FA` which indicates they added logic to decode this and place the result on the data bus when

```
in a,($fa)
```

is executed by the z80. This most likely means adding chips and lots of wires. You might have missed it, but George suggested port `$FF`. Why? Because this simplifies things inside a Model I—a lot.

If you are familiar with Kitsz's *The Custom TRS-80 & Other Mysteries* on page 51 he reports port `$FF` is used for

- Cassette Data I/O (Internal)
- 80-Grafix (Programma)
- Cassette Motor Switch (Internal)
- Video Character Size Latch (Internal)
- Simutek T-Beep Addition

All of these, except Cassette Data I/O are `out` instructions. (80-Grafix uses bits `5`, `6`, and `7` to set its mode: `$20`=normal TRS-80 character set, `$A0`=use new character set, $60=program a new character set. T-Beep uses bit `0` to turn a beep `1`=on and `0`=off.)

There is a very readable description of how cassette input occurs by Barden in *More TRS-80 Assembly Language Programming* on page 224 [0]. Of interest to us, however, is

- only bit `7` is used and
- due to the precise timing necessary to correctly read from the cassette it is unlikely a *Galaxy Invasion*-like game is simultaneously on the screen.

The goal was to get the value of **VDRV** on **D0** (bit `0`) upon input on port `$FF`. In software terms, after

```
in a,($ff)
```

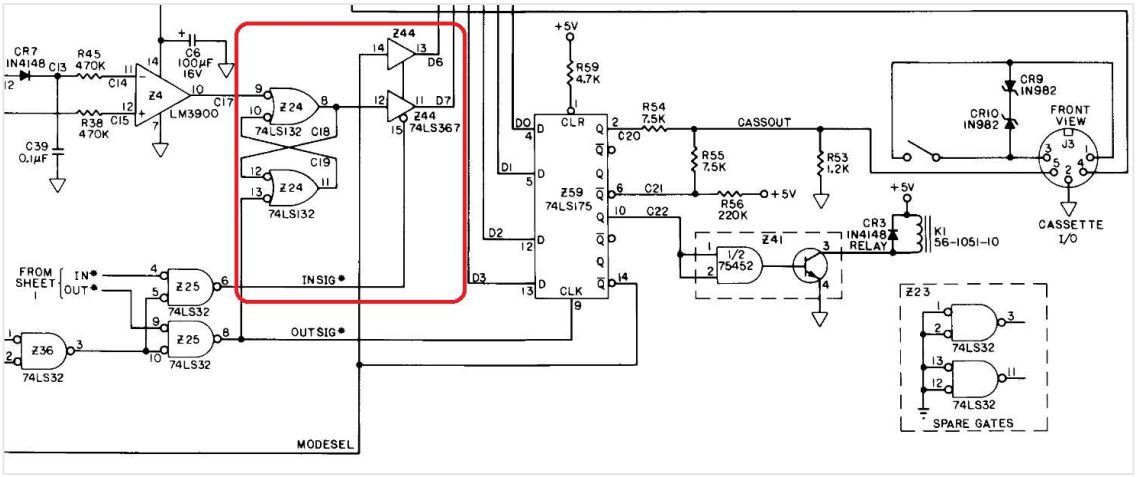
bit `0` of the accumulator would indicate if we are in VBLANK: `0`=no, `1`=yes.

Before proceeding, we review what happens during an I/O read.

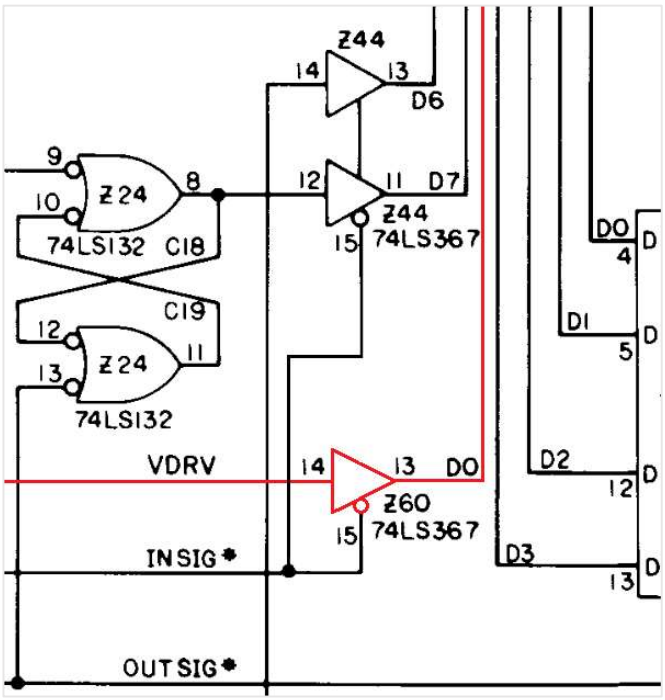
1. (First clock cycle) The address of the port is placed on the lower byte of the address bus (i.e., **A0**–**A7**).

2. (Second clock cycle) The **~IORQ** request line becomes active (i.e., goes low). For a I/O read, **~RD** will also go low with **~IORQ**.
3. (Third clock cycle) The data bus is read (i.e., **D0**–**D7**) into an 8-bit register (likely the accumulator because it saves a few t-states).

The portion of the Model I schematic we focus on is shown below.



Here **INSIG*** has already determined **A0**–**A7** refer to port `$FF`. We can also see the gate sending to **D7** on the data bus. We augment this logic as shown to the left. We use the spare gate on **Z60** to use **INSIG*** to control **VDRV** onto **D0**.

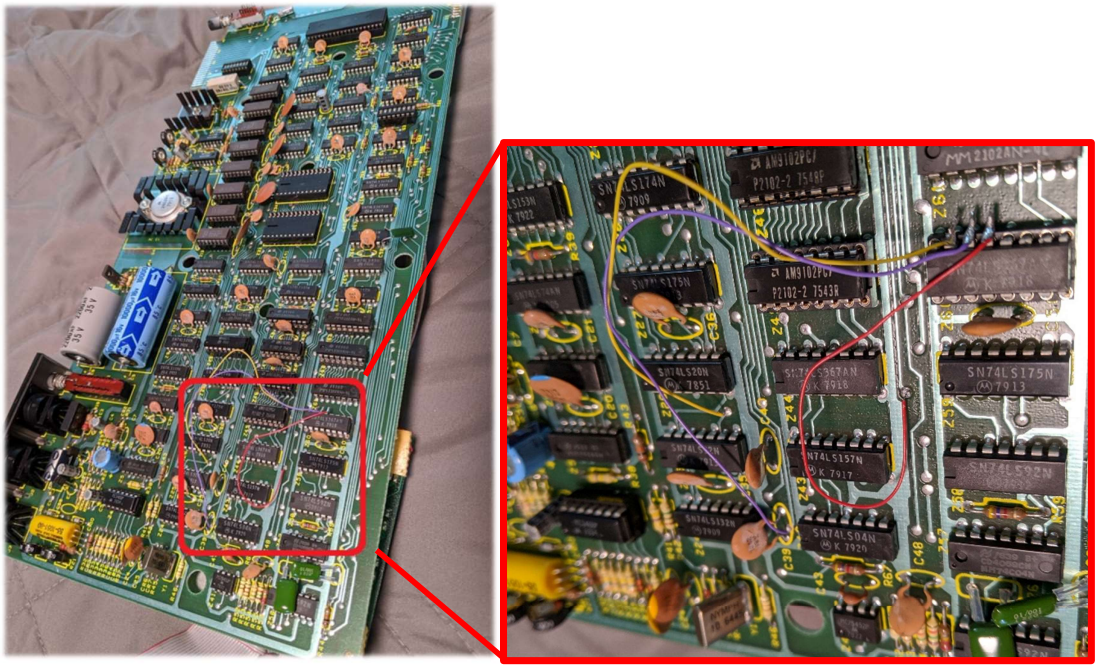


We pause to note the schematic at left could have shipped with the Model I at no additional cost. Software developers would have taken advantage of it to create graphics without interference or tear in 1977. Beamhacking, or 192-pixel vertical resolution graphics, might have been discovered decades before it was.

However, it is hard to criticize. The Model III had support, via its timer interrupt, to see the beginning of every other VBLANK (see http://48k.ca/beamhack1.html for a very readable introduction to what is going on). However, it appears little software use was made of this capability to smooth screen animation.

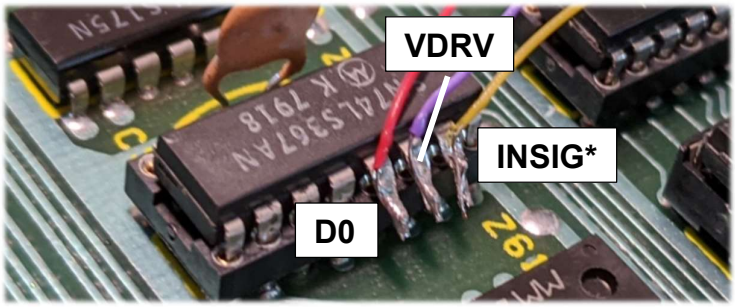
How to do the modification

The VBLANK mod uses a Z60 extra gate and three wires to top-to-bottom pass holes. The image below orients the VBLANK mod on the Model I motherboard.



The steps to take are listed below. We used wire-wrapping wire stripped by a wire-wrapping tool. All connections, however, are soldered.

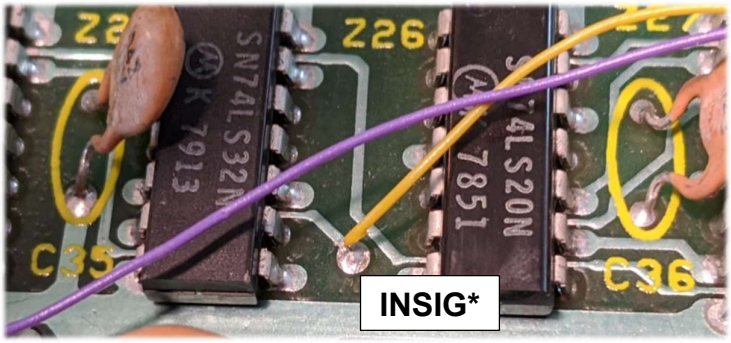
- **(Socket Z60)** De-solder and remove **Z60** and put in an 8-pin socket. We are going to use pins 12, 14, and 15, however, pin 8 (**GND**) is connected to the inputs on pins 12, 14, and 15 (under the chip). We need to ensure that pins we are using are not grounded. Therefore, when you reinsert the chip into the new socket bend out pins 13, 14, and 15.



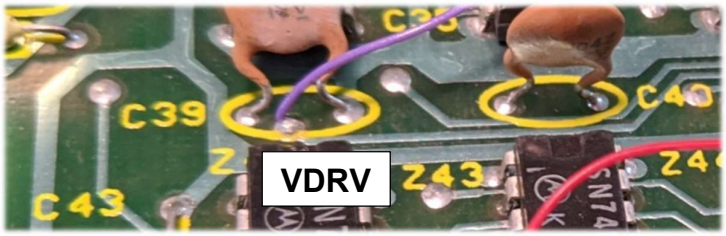
- (**INSIG*** from **Z25** pin 6) **INSIG*** from **Z25** pin 6 is available at the board cross “dot”—a board top-to-bottom cross hole which looks like a little dot of solder— between **Z25** and **Z26**. The “dot” is located between the lower

part of the two chips around pins 7 and 8. **Attach a wire between this dot and Z60 pin 15.**

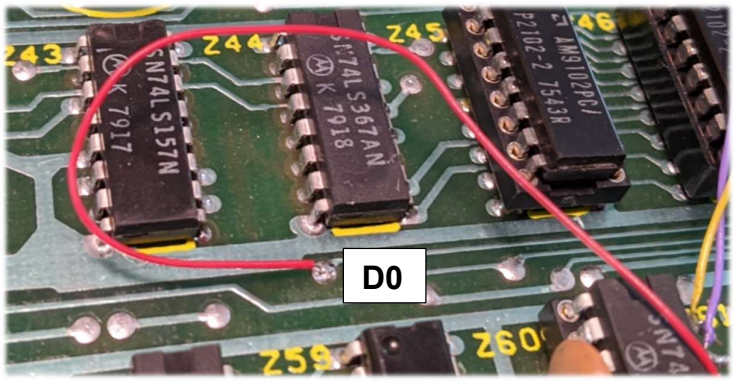
- To double check that you have the right dot test continuity from the dot to either **Z25** pin 6 or **Z44** pin 15.



- (**VDRV**) **VDRV** is available at the board cross “dot” between the posts of **C39**. Attach a wire between this dot and Z60 pin 14.
  - To double check that you have the right dot test continuity from the dot to either **Z30** pin 9 or **Z66** pin 1.



- (**D0**) **D0** is available at the board cross “dot” just below pin 8 of **Z44**. Attach a wire between this and **Z60** pin 13.
  - To double check that you have the right dot test continuity from the dot to either **Z44** pin 7 or **Z76** pin 10.



That is you need to do. Your machine should function as normal. To take advantage of this mod software is needed. We now turn our attention to that.

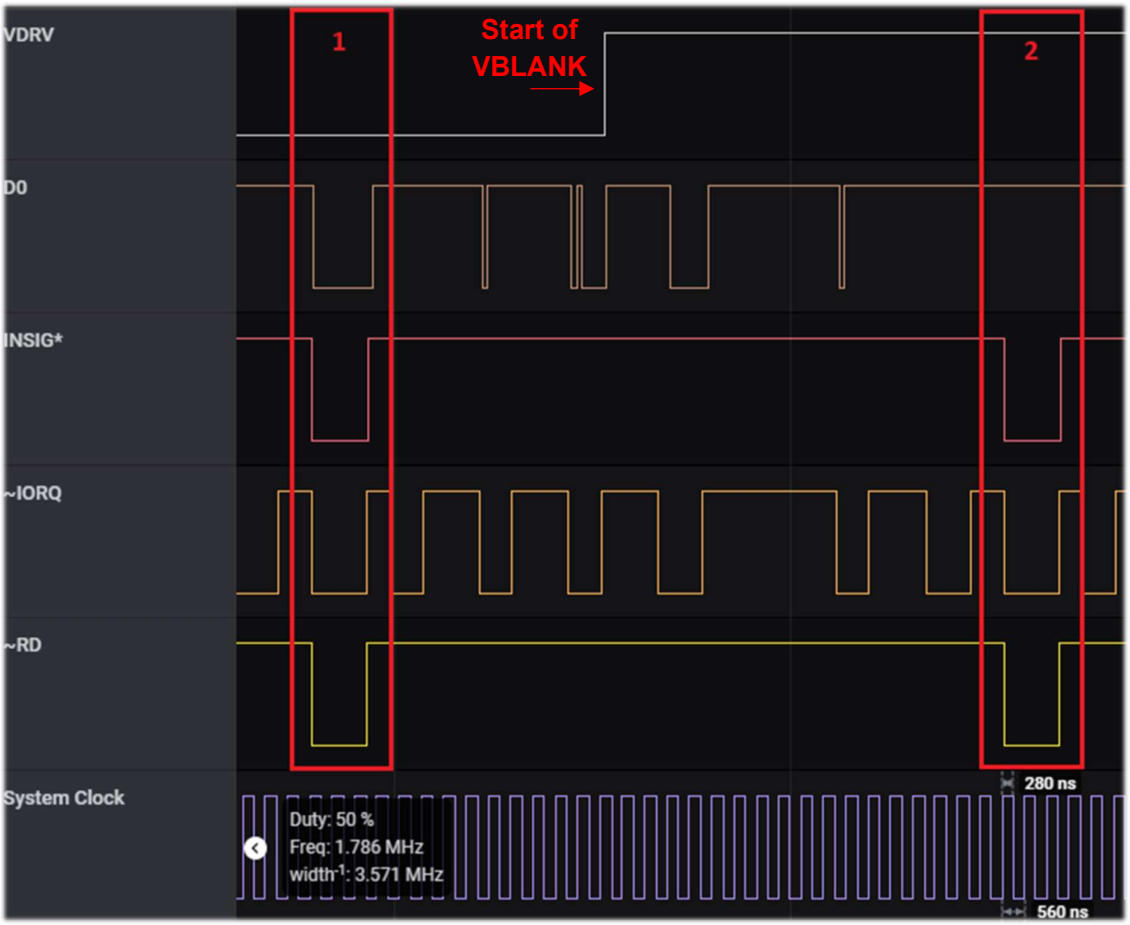
Software

The assembly code below polls our mod, pausing the program until the start of the next vertical blanking interval (VBLANK).

```
wait_in_vblank:
  in a,($ff)
  bit 0,a
  jr nz,wait_in_vblank
wait_not_in_vblank:
  in a,($ff)
  bit 0,a
  jr z,wait_not_in_vblank
```

The top loop handles the case where we catch the machine already in VBLANK. We wait the rest of this cycle out because we have no idea where we are within it. The lower loop waits out the screen being drawn. We are at the start of VBLANK when this code completes.

The logic analyzer trace below shows the `wait_not_in_vblank` loop completing.



At the red box labeled 1 in the figure we see `in a,($ff)` occurring. **D0** reports we are not in VBLANK (which ends up in bit `0` of the accumulator). However, a few clock cycles later VBLANK starts. At the red box labeled 2 in the figure we see `in a,($ff)` occurring again. Now **D0** reports we are in VBLANK and we complete the code listed above.

Another useful software routine is runtime detection of this modification. At the time this article was written only three Model I computers in the world have this modification. We want to write software that uses the VBLANK mod if it is found but works without it.

```
; Runtime check if this Model 1 has the VBLANK mod installed.
;
; On Exit: a - Is the VBLANK mod detected? 1=yes, 0=no
detect_m1_vblank:
  ld hl,1350
  ld bc,-1
  ; A Model 1 will not have a 0 in bit 0 of in $ff without the VBLANK
  ; mod. We want to observe for no less than 1/60th of a second, or
  ; 29566 t-states. The loop below is 44 t-states * 1350 = 59400
  ; or ~1/30th of a second which should be long enough.
_vblank_search_loop:
  in a,($ff)
  bit 0,a
  jr z,_vblank_found
  add hl,bc
  jr c,_vblank_search_loop
  ld a,0
  ret
_vblank_found:
  ld a,1
  ret
```

The use of this code is demonstrated in `vblank_test.asm`. This code is listed in the appendix and available for download. It detects if the VBLANK mod is installed, prints a message, and animates a line up and down on the screen. If the VBLANK mod is installed it is a smooth animation, if not interference and tearing occur—even to the text. Instructions are included to show how to run this code on the trs80gp emulator. Which brings us to our next topic.

Emulator support for this modification

The trs80gp emulator (http://48k.ca/trs80gp.html) supports the VBLANK mod. To include the mod use

```
trs80gp -m1 -m1_vblank
```

This is extremely useful for software development. It greatly reduces the edit-assemble-run cycle.

The Tandy Assembly 2019 demonstration

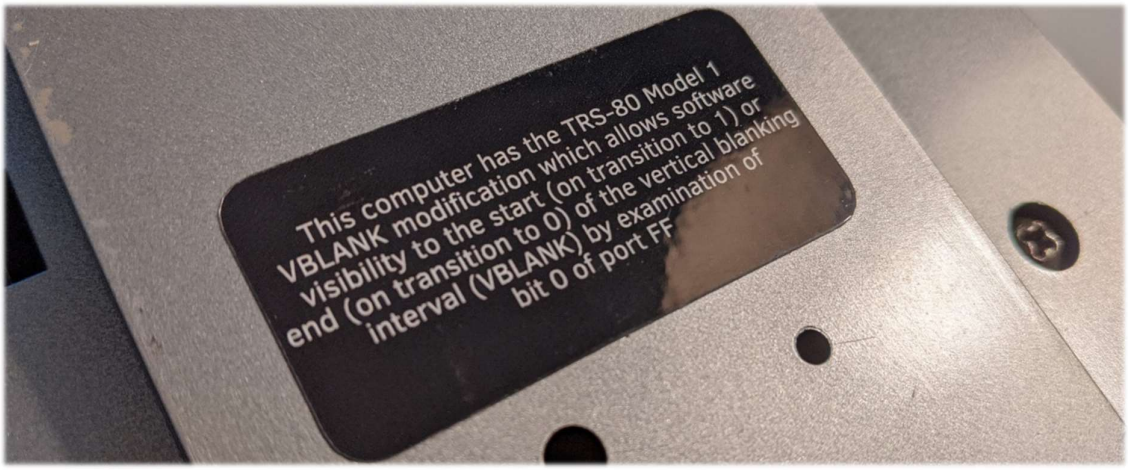
At Tandy Assembly 2019 two programs were demonstrated on Model I computers with the VBLANK mod: A Raycasting demo I wrote and George Phillips' Bouncing Ball program (see http://48k.ca/ball.html).

I was working on the Raycasting program when the VBLANK mod development started. I barely got a working version for Tandy Assembly. It would not have gotten done without the VBLANK mod support George added to the `trs80gp` emulator. Even with that it is quite slow. I made the mistake of using a lot of 16-bit mathematics on the Model I. The complete source code of this demo is online at GitHub (see links).

George demonstrated a Model I version of his Bouncing Ball program. He developed this program, as well as trs80gp emulator support for the VBLANK mod, without ever using a modified machine. George emailed cassette files to me which I loaded and ran on the only machine in the world at the time with this modification. I sent him back videos of what happened.

The VBLANK mod sticker

In the tradition of adding a mod sticker to the bottom of the machine (e.g., for the lowercase modification). An informative sticker was created to attach to VBLANK modified Model I computers. I probably have more of these stickers than there are working Model I computers in the world. If you modify yours, please reach out to me on the Facebook forums or at Tandy Assembly and I will get you one.



Links

Emulator: **trs80gp** http://48k.ca/trs80gp.html
Assembler: **zmac** http://48k.ca/zmac.html

Example code: https://github.com/hallorant/bigmit/tree/master/vblank
YouTube demo: https://youtu.be/sEr9-tCba14

Tandy Assembly 2019 exhibit and Raycaster demo code
https://github.com/hallorant/bigmit/tree/master/ta19demo

References

[0] Barden, William, Jr., *More TRS-80 Assembly Language Programming*. Radio Shack, 1982.

[1] Cetinski, Peter (host), *TRS-80s in Scientific Research*. TRS-80 Trash Talk Episode 21 (podcast). February 2018.
https://www.trs80trashtalk.com/2018/02/episode-21.html

[2] Kitsz, Dennis Bathory, *The Custom TRS-80 & Other Mysteries*. IJG Inc, 1982.

[2] Mapou, Robert L, *Tachistoscopic timing on theTRS·80*. Behavior Research Methods & Instrumentation. 1982, Vol. 14(6), 534-538.
https://link.springer.com/content/pdf/10.3758%2FBF03203418.pdf

[4] Phillips, George, *Beamhacking*. 48k.ca (website), 2009.
http://48k.ca/beamhack1.html http://48k.ca/beamhack2.html
http://48k.ca/beamhack3.html http://48k.ca/beamhack4.html
http://48k.ca/beamhack5.html http://48k.ca/beamhack6.html

Appendix: **vblank_test.asm** source code

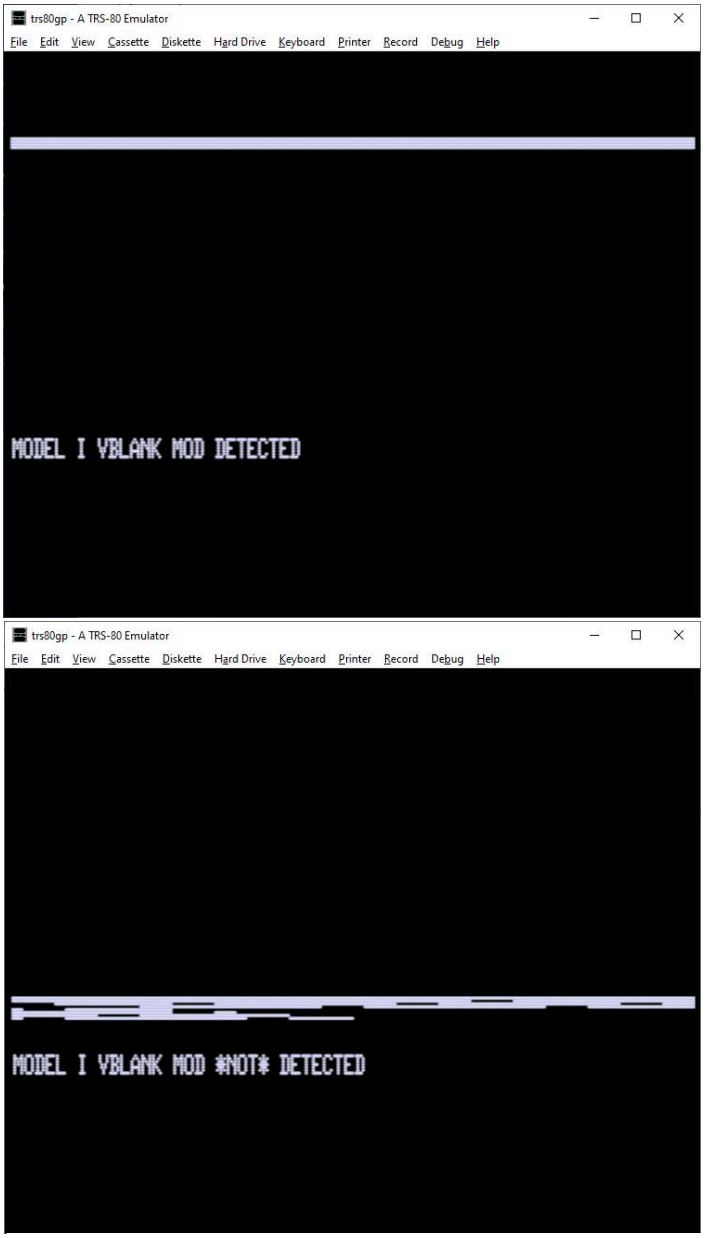
Download: https://raw.githubusercontent.com/hallorant/bigmit/master/vblank/vblank_test.asm

Assemble: `zmac --zmac vblank_test.asm` (get zmac at http://48k.ca/zmac.html)

Run: `trs80gp -m1 -m1_vblank zout/vblank_test.500.cas` (omit `-m1_vblank` for no mod)

```
; Model 1 VBLANK mod demonstration/test program.
; Dynamically detects the mod and animates a line with or without it.
  org $5200 ; change to org $4200 for 4K cassette system.
screen         equ  $3c00
blank_line     dc   64,$80
top_line       dc   64,$83
middle_line    dc   64,$8c
bottom_line    dc   64,$b0
has_vblank     defs 1 ; has VBLANK mod? 1 = yes, 0 = no.
no_vblank_txt  defb 'MODEL I VBLANK MOD *NOT* DETECTED'
no_vblank_len  equ  $-no_vblank_txt
vblank_txt     defb 'MODEL I VBLANK MOD DETECTED'
vblank_len     equ  $-vblank_txt

; Runtime check if this Model 1 has the VBLANK mod installed.
;
; On Exit: a - Is the VBLANK mod detected? 1=yes, 0=no
detect_m1_vblank:
  ld hl,1350
  ld bc,-1
  ; A Model 1 will not have a 0 in bit 0 of in $ff without the VBLANK
  ; mod. We want to observe for no less than 1/60th of a second, or
  ; 29566 t-states. The loop below is 44 t-states * 1350 = 59400
  ; or ~1/30th of a second which should be long enough.
_vblank_search_loop:
  in a,($ff)
  bit 0,a
  jr z,_vblank_found
  add hl,bc
  jr c,_vblank_search_loop
  ld a,0
  ret
_vblank_found:
  ld a,1
  ret

; Waits for the start of the next VBLANK, even if one is going on.
wait_for_vblank_start macro ?wait_in_vblank,?wait_not_in_vblank
?wait_in_vblank:
  in a,($ff)
  bit 0,a
  jr nz,?wait_in_vblank
?wait_not_in_vblank:
  in a,($ff)
  bit 0,a
  jr z,?wait_not_in_vblank
  endm

; Waits a bit to draw the next animation frame.
;
; If the VBLANK mod is installed we wait for the next VBLANK,
; otherwise a loop delay is used.
wait_a_bit macro ?no_vblank_mod,?delay,?done
  ld a,(has_vblank)
  or a ; is a == 0?
  jr z,?no_vblank_mod
  wait_for_vblank_start
  jr ?done
```

```
?no_vblank_mod
  ld hl,1200
  ld bc,-1
?delay
  add hl,bc
  jr c,?delay
?done
  endm

ldir_row macro line,row
  ld hl,line
  ld de,screen+64*row
  ld bc,64
  ldir
  endm

animate_row_down macro row
  ldir_row top_line,row
  wait_a_bit
  ldir_row middle_line,row
  wait_a_bit
  ldir_row bottom_line,row
  wait_a_bit
  ldir_row blank_line,row
  endm

animate_row_up macro row
  ldir_row bottom_line,row
  wait_a_bit
  ldir_row middle_line,row
  wait_a_bit
  ldir_row top_line,row
  wait_a_bit
  ldir_row blank_line,row
  endm

main:
  call detect_m1_vblank
  ld (has_vblank),a
  or a ; is a == 0?
  jr nz,vblank_detected

vblank_not_detected:
  ld hl,no_vblank_txt
  ld de,screen+64*11
  ld bc,no_vblank_len
  ldir
  jr start_animation

vblank_detected:
  ld hl,vblank_txt
  ld de,screen+64*11
  ld bc,vblank_len
  ldir

start_animation:
  wait_a_bit
animation_loop:
  irpc row,0123456789
  animate_row_down &row
  endm
  irpc row,9876543210
  animate_row_up &row
  endm
  jp animation_loop
  end main
```



(top) Running this program with the VBLANK mod installed. (bottom) An example of line drawing interference without the mod. The text is also interfered with from time to time (it sparkles). A YouTube video of this demo is available at https://youtu.be/sEr9-tCba14