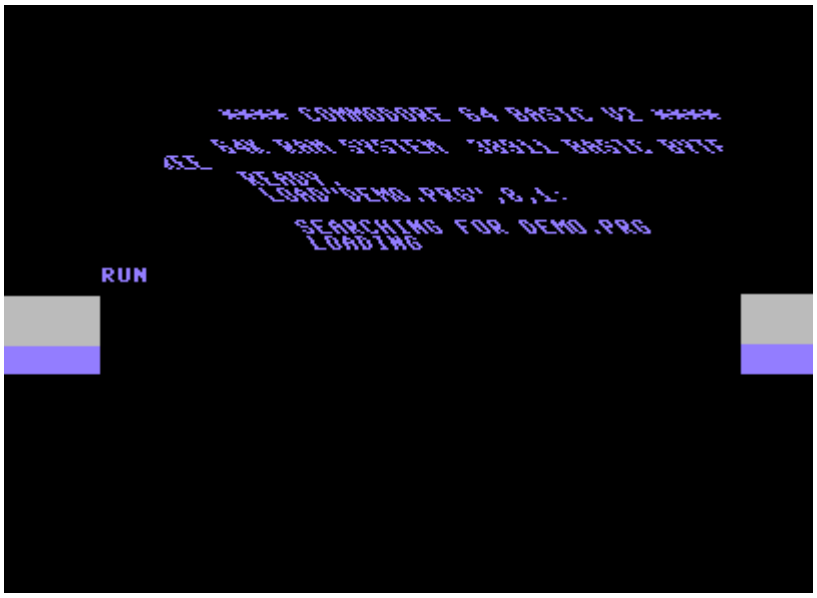
TechTech using FLI routine

Introduction

Below is an example of a tech-tech effect, achieved using a so-called FLI-routine. This is **not** a tech-tech with a FLI logo, its a simple FLI based routine to allow videoram switching at every rasterline.

The routine differs from this article in that it uses videoram banks to create the tech-tech, not character sets. One could combine these two techniques by cleverly interleaving character sets and videoram to allow for a wider sinus, perhaps even bank switching with $dd00 for even more movement. I'll leave that as an exercise for the reader ;)



I wrote this code as part of an attempt to regain my old VIC-trickery skills, it's been about 25 years since I did a tech-tech. As such, the code may not be up to today's standards. I just published it here since there was no article on tech-tech's using $d011/FLI.

Note: I don't explain FLI in detail in this article, there are other articles here which explain FLI properly.

The code can be assembled by using 64tass:

```
64tass -C -a -o techtech.prg techtech.asm
```

The Theory

I'll use the word 'logo' here to simplify things a bit. My code doesn't actually include a logo, just some text copied from the BASIC boot screen.

Since $d016 only allows us to scroll 8 pixels, we need a way to scroll more. We can achieve that using multiple videoram banks (screens). For pixels 0-7 we use a screen with the logo at its left-most position. For each 8 pixels extra, we put the logo one column to the right from the previous position in

a new videoram bank, and so on. An illustration would probably be clearer:

```
screen:  $4800     $4c00     $5000     $5400     and so on..

column:  01234567  01234567  01234567  01234567
logo:    HELLO      HELLO      HELLO      HELLO
```

Now we can select a column-offset of the logo using videoram banks (bit 7-4 of $d018) and we can use $d016 (bit 0-2) to select a pixel offset of 0-7.

For example, we want a line of the logo scrolled 21 pixels from its left-most position. We can achieve this with setting $d016 to 5 (21 & 7 == 5) and setting $d018 to $40 (21 / 8 == 2, multiply with 16 to get the correct videoram bits, and add $20 to skip the first two videoram banks which contain the character set)

In pseudo-code:

```
lda logo_offset  ; some value from the sinus table
and #7           ; ORA with $10 for multi-color
sta $d016
lda logo_offset
asl              ; divide by 8, multiply by 16 == multiply by 2
and #$f0         ; mask out lower nybble
clc
adc #$20         ; add $20 to skip the first two videoram banks
                 ; since the character set occupies those banks
sta $d018
```

This code we need to do for each (raster)line of the logo. But we have a problem...

The Problem

Manipulating $d016 on each raster line works fine, manipulating $d018 doesn't: the videoram pointer only gets updated on bad lines (every 8th raster line), so we need a way to update the videoram pointer on each raster line. That's were FLI comes in.

The Solution

Using the so-called FLI technique, we can trigger bad lines at every raster line, allowing us to manipulate the videoram pointer at every raster line.

So our basic tech-tech display routine would look like this: (loop unrolled to keep the FLI bug at three characters while using a single sprite to cover the bug)

```
; sinus = 0
lda #$1b
sta $d011
```

```
lda #$00    ; 0 & 7 == 0
sta $d016
lda #$20    ; column 0
sta $d018

; sinus = 3
; next rasterline
lda #$1c
sta $d011
lda #$03
sta $d016
lda #$20    ; still column 0
sta $d018

; sinus = 6
lda #$1d
sta $d011
lda #$06
sta $d016
lda #$20
sta $d018

; sinus = 9
lda #$1e
sta $d011
lda #$01    ; 9 & 7 == 1
sta $d016
lda #$30
sta $d018   ; column 1: 9 pixels means we need to select
            ; the second videoram bank to move the logo
            ; one column to the right

; and so on for each line of the logo, wrapping the $d011
; value around from $1f to $18
```

Now we can tech-tech the logo by storing the correct values for $d016 and $d018 in the unrolled code, using for example a sinus table. The example code contains a routine which pre-calculates the $d016 and $d018 values from a sinus table for faster performance and another unrolled loop to store these values in the FLI-routine each frame, again for performance.

The Code

Here's the actual source code, in 64tass syntax. The FLI routine needs to be unrolled, the sinus calculation can all be done in a loop, but that eats cycles, so I also wrote some unrolled code for the $d016/$d018 values updating.

Before assembling, one should probably comment out the references to the music, I would be surprised to see everyone having their HVSC at “/home/compyx/c64/HSVC” ;)

```
; vim: set et ts=8 sw=8 sts=8 syntax=64tass :
;
; Simple tech-tech using FLI, 112 pixels wide sinus, using one charset and
; 14 videoram banks in $4000-$7fff.
;
;
; 2016-04-20


        ; Music, a nice old school JCH tune
        music_sid ="/home/compyx/c64/HVSC/MUSICIANS/J/JCH/Ninjackie.sid"
        music_init = $1000
        music_play = $1003


        ; height of the tech-tech in pixels
        TECHTECH_HEIGHT = 88

        ; size of a single line of FLI code
        TECHTECH_MACRO_LEN = 15

        ; location of the FLI-bug cover sprite
        COVER_SPRITE = $7f80

        ; zero page
        zp = $14


        ; BASIC SYS line: SYS2061
        * = $0801
        .word (+), 2016
        .null $9e, ^start
+       .word 0

; Entry point
start
        jsr $fda3
        jsr $fd15
        ; jsr $ff5b
        sei
        lda #0
        sta $d020
        sta $d021
        jsr tt_setup
        lda #0
        jsr music_init
        lda #$35
        sta $01
        lda #$7f
        sta $dc0d
        sta $dd0d
```

```
        lda #0
        sta $dc0e
        lda #$01
        sta $d01a
        lda #$1b
        sta $d011
        lda #$2e
        ldx #<irq1
        ldy #>irq1
        sta $d012
        stx $fffe
        sty $ffff
        ldx #<break
        ldy #>break
        stx $fffa
        sty $fffb
        stx $fffc
        sty $fffd
        bit $dc0d
        bit $dd0d
        inc $d019
        cli
        jmp *

; IRQ: use the 'double IRQ' trick to stabilize the raster
irq1
        pha
        txa
        pha
        tya
        pha
        lda #$2e
        ldx #<irq2
        ldy #>irq2
        sta $d012
        stx $fffe
        sty $ffff
        lda #1
        inc $d019
        tsx
        cli
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
```

```
        nop
irq2
        txs
        ldx #8
-       dex
        bne -
        bit $ea
        lda $d012
        cmp $d012
        beq +
+
        ; raster is stable here
        clc
        lda #$32
        sta $d001
        adc #42
        sta $d003
        lda #0
        sta $d027
        sta $d028
        lda #%00000011  ; stretch cover sprites in X and Y direction
        sta $d017
        sta $d015
        sta $d01d
        lda #$08        ; we need to conver the first four chars on screen,
        sta $d000       ; since we have the classic three char FLI bug and
we
        sta $d002       ; need one more char to cover since we use $d016,
which
        lda #$7b        ; scrolls the FLI bug into the screen
        sta $d011
        lda #2
        sta $dd00

        ; waste some cycles to start the FLI routine at the correct time
        ldx #7
-       dex
        bne -
        nop
        nop
        nop

        ; the unrolled tech-tech FLI routine
        jsr tt_unrolled
        ; use invalid graphics mode to cover bugs
        lda #$7b
        sta $d011
        lda #$03
        sta $dd00
        lda #$15
        sta $d018
```

```
        lda #8
        sta $d016
        ldx #89         ; waste cycles until we've covered a full screen row
-       dex
        bne -
        lda #$1b
        sta $d011


        lda #$98
        ldx #<irq3
        ldy #>irq3
        sta $d012
        stx $fffe
        sty $ffff
        lda #1
        sta $d019
        pla
        tay
        pla
        tax
        pla
break   rti

irq3
        pha
        txa
        pha
        tya
        pha
        dec $d020
        jsr tt_sinus_unrolled   ; calculate the tech-tech's sinus data
        dec $d020
        jsr music_play
        lda #0
        sta $d020
        lda #$2d
        ldx #<irq1
        ldy #>irq1
        sta $d012
        stx $fffe
        sty $ffff
        lda #1
        sta $d019
        pla
        tay
        pla
        tax
        pla
        rti
```

```
; Tech-tech setup code
tt_setup
        ; copy CHARGEN
        ;
        ; For the 'logo', we use the CBM font
        lda #$32
        sta $01
        ldx #0
-       lda $d000,x
        sta $4000,x
        lda $d100,x
        sta $4100,x
        lda $d200,x
        sta $4200,x
        lda $d300,x
        sta $4300,x
        lda $d400,x
        sta $4400,x
        lda $d500,x
        sta $4500,x
        lda $d600,x
        sta $4600,x
        lda $d700,x
        sta $4700,x
        inx
        bne -

;        ldx #7
;        lda #0
;-       sta $47f8,x
;        dex
;        bpl -

        ; generate FLI-bug cover sprite
        ldx #$3f
        lda #$ff
-       sta COVER_SPRITE,x
        dex
        bpl -

        ; Set up the videoram banks for the tech-tech effect
        ;
        ; The first videoram bank at $4800 contains the 'logo' in its
        ; default (left-aligned) position, every next videoram bank contains
        ; the 'logo' shifted one column to the right. This is what makes the
        ; tech-tech effect possible.
        ;
        ; Right now, for the 'logo', we simply copy the BASIC screen data
        ; from $0400
        ldx #0
```

```
-
        lda $0400,x
        sta $4800,x
        sta $4c00 + 1,x
        sta $5000 + 2,x
        sta $5400 + 3,x
        sta $5800 + 4,x
        sta $5c00 + 5,x
        sta $6000 + 6,x
        sta $6400 + 7,x
        sta $6800 + 8,x
        sta $6c00 + 9,x
        sta $7000 + 10,x
        sta $7400 + 11,x
        sta $7800 + 12,x
        sta $7c00 + 13,x

        lda $0500,x
        sta $4900,x
        sta $4d00 + 1,x
        sta $5100 + 2,x
        sta $5500 + 3,x
        sta $5900 + 4,x
        sta $5d00 + 5,x
        sta $6100 + 6,x
        sta $6500 + 7,x
        sta $6900 + 8,x
        sta $6d00 + 9,x
        sta $7100 + 10,x
        sta $7500 + 11,x
        sta $7900 + 12,x
        sta $7d00 + 13,x
        inx
        bne -

        ; make last tech-tech line use invalid $d011 mode to mask bug
        lda #((TECHTECH_HEIGHT - 1) & 7 | $78)
        sta tt_unrolled + ((TECHTECH_HEIGHT - 1) * TECHTECH_MACRO_LEN) + 1

        ; set cover sprite pointers for each videoram bank used
        lda #$f8
        ldx #$4b
        sta zp
        stx zp + 1
        ldx #0
-
        lda #(COVER_SPRITE & $3fff) / 64
        ldy #0
        sta (zp),y
        iny
        sta (zp),y
```

```
        lda zp + 1
        clc
        adc #4
        sta zp + 1
        inx
        cpx #14
        bne -

        jsr tt_sinus_precalc

        lda #$37
        sta $01
        rts


; Precalcute sinus data
;
tt_sinus_precalc
        ldx #0
        ldy #0
-       lda sinus,y
        and #7
        sta sinus_d016,x
        sta sinus_d016 + 256,x
        lda sinus,y             ; divide by 8, multiply by 16 to get
videoram
                                ; index
        asl
        and #$f0
        adc #$20                ; C is clear
        sta sinus_d018,x
        sta sinus_d018 + 256,x
        tya
        clc
        adc #1
        tay
        inx
        bne -
        rts


        .cerror * > $0fff, "code too long"



; Link music
        * = $1000
.binary music_sid, $7e
```

```
; A single rasterline of FLI-code to display the tech-tech
;
; @param 1: row number (used to calculate the correct $d011 value)
;
; We trigger a badline condition at each rasterline to trigger a videoram
; update which we use to alter the videoram bank
;
ttmacro .macro
        lda #$18 + ((\1 + 3) & 7)
        sta $d011
        lda #$20        ; gets updated in the sinus routine
        sta $d018
        lda #$08        ; gets updated in the sinus routine
        sta $d016
        .endm


        * = $2000

; The tech-tech display routine: a simple unrolled FLI routine which also
sets
; $d016 at each line
tt_unrolled
.for row = 0, row < TECHTECH_HEIGHT, row = row + 1
        #ttmacro row
.next
        rts



        .align 256
; Unrolled sinus updating routine
;
; Uses two tables of 512 bytes each, to allow for X index overflow (we add
; an offset to each sinus_d016 and sinus_d018 table for each row, combine
that
; with X register indexing and we would go past the 256-byte mark in a
table)
tt_sinus_unrolled
_index  ldx #0

.for row = 0, row < TECHTECH_HEIGHT, row = row + 1
        lda sinus_d018 + row,x
        sta tt_unrolled + (row * TECHTECH_MACRO_LEN) + 6
        lda sinus_d016 + row,x
        sta tt_unrolled + (row * TECHTECH_MACRO_LEN) + 11
.next
        lda _index + 1
        clc
        adc #2
```

```
        sta _index + 1
        rts

; Sinus used for the tech-tech effect: 112 pixels wide since we use 14
; videoram banks
        .align 256
sinus
        .byte 55.5 + 56 * cos(range(256) * rad(360.0/256))


; Precalculated $d016 values
sinus_d016
        .fill 512, 0

; Precalculated $d018 values
sinus_d018
        .fill 512,0
```