D.Y.S.P. using a cycle table

Introduction

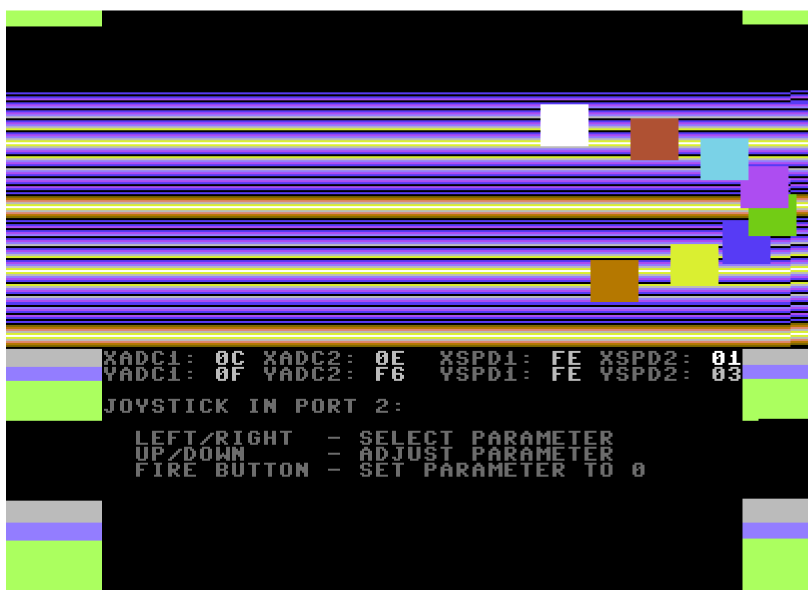
This article describes a way to do a d.y.s.p. with eight sprites, using flexible cycle-wasting in the loop. This method differs from the $d017-approach in that we can use the full 21 lines of a sprite. Unfortunately this means our loop gets a little more complicated due to the fact that the number of cycles on a given raster line is now dependent on the number of sprites on that line.

Getting and assembling the code

The code is hosted at GitHub. It requires 64tass. Once you've cloned the repo, a simple

```
make
```

can be used to assemble to code into a runnable .prg file.



Opening the border with flexible timing

The raster routine itself is fairly simple, we manipulate $d011 so we don't get any bad lines, which in turn allows us to display all eight sprites in the side-border:

```
dysp
        ldy #8
        ldx #0
-       lda d021_table,x
        dec $d016
        sta $d021
        sty $d016
        lda d011_table,x
```

```
        sta $d011

        ; this is the interesting bit, for each raster line
        ; we alter the branch so it wastes the correct number
        ; of cycles for the next iteration of the loop:
        lda timing,x
        sta _delay + 1
_delay  bpl * + 2
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        bit $ea
        inx
        cpx #DYSP_HEIGHT
        bne -
        rts
```

By altering the branch instruction to skip a variable amount of bytes of code, we can determine the number of cycles the loop must waste on a raster line. Generally speaking, the more sprites we have on a line, the more cycles the VIC eats, which means we have to branch further to skip more cycles.

The “cpx #$e0 … bit $ea” code is used to waste cycles with an accuracy of a single cycle, I use the same trick to time VSP's (with a few more “cpx #$e0” instructions).

Here's how it works (simplified):

```
; here we waste 2 + 2 + 3 = 7 cycles:
.1000   bpl $1002
.1002   cpx #$e0      ; 2
.1004   cpx #$e0      ; 2
.1006   bit $ea       ; 3

; adjusting the branch, we can waste 6 cycles:
.1000   bpl $1003     ; we branch into the argument of CPX #$e0 at $1003
.1002   cpx #$e0      ; which means we execute CPX #$e0 at $1003,
.1004   cpx #$e0      ; then CPX #$24 at $1005
.1006   bit $ea       ; and finally NOP at $1007

; so, for the above code , the CPU executes this:
.1000   bpl $1003
; .1002 gets skipped
.1003   cpx #$e0      ; 2
.1005   cpx #$24      ; 2
.1007   nop           ; 2
```

So for each additional byte we branch over, we either end at BIT $EA (3 cycles) or NOP (2 cycles). Now

we can waste anywhere between 0 (no sprites) and 17 cycles (all 8 sprites).

Calculating how many cycles we use

cycle table

To determine how many cycles we need to waste, we need to know how many cycles each combination of sprites uses. That's where the 'cycles' table comes in, it gives us the amount of cycles sprites use for each 'sprite enable'/$d015 value. I used a little tool to determine those values, using brute force: for each combination I adjusted the cycle-delay until I got the proper value.

The tool is also on GitHub, but be warned, the code is a little messy.

sprite enable table

Now to determine which values to pick from the 'cycles' table and store in the 'timing' table, we need yet another table, which I call the 'sprite enable' table. This table is cleared each frame, and then populated using ORA for each sprite:

For each sprite, at the proper Y-position, we ORA 21 values of the sprite enable table with the bit value for that particular sprite: $01 for sprite 0, $02 for sprite 1, up to $80 for sprite 7.

An illustration might help: suppose we have sprite 0 at Y-offset 0, sprite 1 at Y-offset 3 and sprite 2 at Y-offset 5:

```
          Sprite ORA values  Sprite enable
Y-offset  spr0   spr1  spr2  table result
--------  -----  ----  ----  -------------
00          01                 01
01          01                 01
02          01                 01
03          01     02          03
04          01     02          03
05          01     02    04    07
06          01     02    04    07
```

loop timing table

Using this result, we can use the 'sprite enable' table values as an index into the cycles table to get the proper value for the 'timing' table used in the border-loop:

```
        ldx #0
-       ldy sprite_enable,x
        lda cycles,y
        sta timing,x
        inx
        cpx #DYSP_HEIGHT
```

```
        bne -
        rts
```

Done

And there you have it, we now have a D.Y.S.P. with flexible Y-positions. Naturally all this calculating eats cycles, which is why my code uses a lot of unrolled loops.

The Code

Finally, the code. It also contains a user interface, allowing the user to change the DYSP's movements with a joystick in port 2.

```
; vim: set et ts=8 sw=8 sts=8 syntax=64tass :

; D.Y.S.P. using pre-calculated cycle table
;
; 2016-04-01

        music_sid = "Blitter.sid"
        music_init = $1000
        music_play = $1003

        DYSP_HEIGHT = 128

        JOY_UP = $01
        JOY_DOWN = $02
        JOY_LEFT = $04
        JOY_RIGHT = $08
        JOY_FIRE = $10

        zp = $10

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


; Initialization code
;
; Set up sprites, initialize VIC, set up IRQ handlers
start
        jsr $fda3
        jsr $fd15
        jsr $ff5b
        sei
```

```
        ; set sprite colors
        clc
        ldx #0
-       txa
        adc #1
        sta $d027,x
        inx
        cpx #8
        bne -
        ; create an example sprite in the tape buffer
        ldx #$3f
        lda #$ff
-       sta $0340,x
        dex
        bpl -
        lda #($340 / 64)
        ldx #7
-       sta $07f8,x
        dex
        bpl -
        ; set up interface text
        ldx #0
-       lda iface_text,x
        sta $0400,x
        lda #$0b
        sta $d800,x
        inx
        bne -
-       lda iface_text + 256,x
        sta $0500,x
        lda #$0b
        sta $d900,x
        inx
        cpx #$40
        bne -
        lda #0
        jsr music_init
        lda #$35
        sta $01
        lda #$7f
        sta $dc0d
        sta $dd0d
        ldx #0
        stx $dc0e
        stx $dd0e
        stx $3fff
        lda #$01
        sta $d01a
        lda #$1b
        sta $d011
        lda #$29
```

```
        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 handlers: $d020 changes are used to show the raster time used by the
;
;  various routines.
;
;---------------------------------------------------------------------------
-;

        ; avoid timing critical loops to cross page boundaries
        .align 256
irq1
        ; 'Double IRQ' method to stabilize raster
        pha
        txa
        pha
        tya
        pha
        lda #$2a
        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
        nop
irq2
        txs
        ldx #8
-       dex
        bne -
        bit $ea
        lda $d012
        cmp $d012
        beq +
+
        ldx #$10
-       lda sprite_positions,x
        sta $d000,x
        dex
        bpl -
        lda #$ff
        sta $d015
        ldx #$14
-       dex
        bne -
        nop
        jsr dysp        ; the actual DYSP loop
        lda #0
        sta $d021
        sta $d015
        dec $d020
        ; responde to user input
        jsr joystick2
        jsr update_iface
        dec $d020
        jsr param_highlight
        dec $d020
        jsr music_play
        lda #0
        sta $d020
        lda #$f9
        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
        ldx #7
-       dex
        bne -
        stx $d011
        ldx #30
-       dex
        bne -
        lda #$1b
        sta $d011
        dec $d020
        ; calculate X and Y movement of the DYSP
        jsr dysp_x_sinus
        jsr dysp_y_sinus
        dec $d020
        ; clear the 'sprite enable' table
        jsr dysp_clear_timing_fast
        dec $d020
        ; calculate the 'sprite enable' values for raster line of the DYSP
        jsr dysp_calc_timing_fast
        lda #0
        sta $d020
        lda #$29
        ldx #<irq1
        ldy #>irq1
        sta $d012
        stx $fffe
        sty $ffff
        lda #1
        sta $d019
        pla
        tay
        pla
        tax
        pla
        rti

; sprite positions: values for $d000-$d010
sprite_positions
        .byte $00, $a0
        .byte $18, $a0
```

```
        .byte $30, $a0
        .byte $48, $a0

        .byte $60, $a0
        .byte $78, $a0

        .byte $90, $a0
        .byte $a8, $a0
        .byte $00

;-----------------------------------------------------------------------------
-;
;                      User interface/DYSP control code
;
;-----------------------------------------------------------------------------
-;

; DYSP sinus control parameters
dysp_x_idx1     .byte 0
dysp_x_idx2     .byte 0

dysp_y_idx1     .byte 0
dysp_y_idx2     .byte 0


params          ; offset 0 of the parameters, used by the joystick routine

dysp_x_adc1     .byte 12
dysp_x_adc2     .byte 9
dysp_x_spd1     .byte 2
dysp_x_spd2     .byte 3

dysp_y_adc1     .byte $0f
dysp_y_adc2     .byte $f6
dysp_y_spd1     .byte $fe
dysp_y_spd2     .byte $3


; colorram locations of the parameter values
param_colram
        .word $d807, $d811, $d81c, $d826
        .word $d82f, $d839, $d844, $d84e


; index in the parameter list for the joystick routine
param_index     .byte 0

; Text for the user interface
iface_text
        .enc screen
```

```
        ;      0123456789abcdef0123456789abcdef01234567
        .text "xadc1: 00 xadc2: 00  xspd1: 00 xspd2: 00"
        .text "yadc1: 00 yadc2: 00  yspd1: 00 yspd2: 00"
        .text "                                        "
        .text "joystick in port 2:                     "
        .text "                                        "
        .text "  left/right  - select parameter        "
        .text "  up/down     - adjust parameter        "
        .text "  fire button - set parameter to 0      "
iface_text_end

; Translate A into hexadecimal digits in A (bit 7-4) and X (bit 3-0)
hex_digits
        pha
        and #$0f
        cmp #$0a
        bcs +
        adc #$3a
+       sbc #$09
        tax
        pla
        lsr
        lsr
        lsr
        lsr
        cmp #$0a
        bcs +
        adc #$3a
+       sbc #$09
        rts

; Update the interface's parameter display
update_iface
        lda dysp_x_adc1
        jsr hex_digits
        sta $0407
        stx $0408
        lda dysp_x_adc2
        jsr hex_digits
        sta $0411
        stx $0412
        lda dysp_x_spd1
        jsr hex_digits
        sta $041c
        stx $041d
        lda dysp_x_spd2
        jsr hex_digits
        sta $0426
        stx $0427

        lda dysp_y_adc1
```

```
        jsr hex_digits
        sta $042f
        stx $0430
        lda dysp_y_adc2
        jsr hex_digits
        sta $0439
        stx $043a
        lda dysp_y_spd1
        jsr hex_digits
        sta $0444
        stx $0445
        lda dysp_y_spd2
        jsr hex_digits
        sta $044e
        stx $044f
        rts



; highlight the currently adjustable parameter
param_highlight
        ; clear param highlighting
        ldx #0
-       lda param_colram,x
        sta zp
        lda param_colram + 1,x
        sta zp + 1
        ldy #0
        lda #$0f
        sta (zp),y
        iny
        sta (zp),y
        inx
        inx
        cpx #16
        bne -
        ; highlight current param
        lda param_index
        asl
        tax
        lda param_colram,x
        sta zp
        lda param_colram + 1,x
        sta zp + 1
        ldy #0
        lda #$01
        sta (zp),y
        iny
        sta (zp),y

        rts
```

```
; Check user input from joystick #2
joystick2
        lda #8
        beq +
        dec joystick2 + 1
        rts
+       lda $dc00
        sta zp
        and #%00011111
        eor #%00011111
        bne +
        rts
+       lda #8
        sta joystick2 + 1
        lda zp
        and #JOY_UP
        beq joy2_up
        lda zp
        and #JOY_DOWN
        beq joy2_down
        lda zp
        and #JOY_LEFT
        beq joy2_left
        lda zp
        and #JOY_RIGHT
        beq joy2_right
        lda zp
        and #JOY_FIRE
        beq joy2_fire
        rts
joy2_up
        ldx param_index
        inc params,x
        rts
joy2_down
        ldx param_index
        dec params,x
        rts
joy2_left
        lda param_index
        sec
        sbc #1
        and #7
        sta param_index
        rts
joy2_right
        lda param_index
        clc
        adc #1
        and #7
```

```
        sta param_index
        rts
joy2_fire
        ldx param_index
        lda #0
        sta params,x
        rts



; Calculate DYSP Y-movement
dysp_y_sinus
        ldx dysp_y_idx1
        ldy dysp_y_idx2
        ; unroll loop for speed:
.for index = 0, index < 8, index = index + 1
        lda ysinus,x
        clc
        adc ysinus,y
        adc #$32
        sta sprite_positions + 1 + (index * 2)
.if index < 7
        ; only needed 7 times
        txa
        clc
        adc dysp_y_adc1
        tax
        tya
        clc
        adc dysp_y_adc2
        tay
.endif
.next
        lda dysp_y_idx1
        clc
        adc dysp_y_spd1
        sta dysp_y_idx1
        lda dysp_y_idx2
        clc
        adc dysp_y_spd2
        sta dysp_y_idx2
        rts


; temp storage for $d010 calculations
xmsb_tmp .byte 0

; Calculate DYSP X-movement using two sinus tables added together
dysp_x_sinus
        lda #0
        sta xmsb_tmp
```

```
        ldx dysp_x_idx1
        ldy dysp_x_idx2

        ; once again unroll loop for speed
.for index = 0, index < 8, index = index + 1
        lda xsinus_256,x
        clc
        adc xsinus_96,y
        sta sprite_positions + (index * 2)
        bcc +
        lda xmsb_tmp
        ora #(1 << index)
        sta xmsb_tmp
+
.if index < 7
        ; this section is only needed 7 times
        txa
        clc
        adc dysp_x_adc1
        tax
        tya
        clc
        adc dysp_x_adc2
        tay
.endif
.next
        ; store  $d010 value in the IRQ handler
        lda xmsb_tmp
        sta sprite_positions + 16

        lda dysp_x_idx1
        clc
        adc dysp_x_spd1
        sta dysp_x_idx1
        lda dysp_x_idx2
        clc
        adc dysp_x_spd2
        sta dysp_x_idx2
        rts



        .align 256      ; avoid page boundary crossing in raster bars

; The actual DYSP routine:
;
; The access to the 'timing' table is what makes this possible. It contains,
; for each raster line, the number of cycles the sprites use. By storing
that
; value in the BPL argument we can waste between 0 and 17 cycles inclusive.
```

```
;
; Unrolling this loop and altering the code which calculates the cycle waste
; values (storing them directly in the unrolled code, not in a table), we
can
; easily add three raster splits.
dysp
        ldy #8
        ldx #0
-       lda d021_table,x
        dec $d016
        sta $d021
        sty $d016
        lda d011_table,x
        sta $d011

        lda timing,x
        sta _delay + 1
_delay  bpl * + 2
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        cpx #$e0
        bit $ea
        inx
        cpx #DYSP_HEIGHT
        bne -
        rts

.cerror * > $0fff, "code section too large!"

        * = $2000
ysinus
        .byte ((DYSP_HEIGHT - 24) / 4)  + 0.5 + ((DYSP_HEIGHT - 24) / 4) *
sin(range(256) * rad(360.0/256))

xsinus_256
        .byte 127.5 + 128 * sin(range(256) * rad(360.0/256))
xsinus_96
        .byte 47.5 + 48 * sin(range(256) * rad(360.0/256))



        .align 256

; The 'sprite enable' table, this is where the number of active sprites per
; raster line is stored. The values in this table are used as index into the
; 'cycles' table to get the proper amount of cycles to skip in the DYSP loop
```

```
dysp_sprite_enable
        .fill DYSP_HEIGHT, 0



        .align 256
d011_table
.for row = 0, row < DYSP_HEIGHT, row = row + 1
        .byte $18 + ((row + 3) & 7)
.next

        .align 256

; Raster bar colors
d021_table
        .byte $06, $00, $06, $04, $00, $06, $04, $0e
        .byte $00, $06, $04, $0e, $0f, $00, $06, $04
        .byte $0e, $0f, $07, $00 ,$06, $04, $0e, $0f
        .byte $07, $01, $07, $0f, $0e, $04, $06, $00
        .byte $07, $0f, $0e, $04, $06, $00, $0f, $0e
        .byte $04, $06, $00, $0e, $04, $06, $00, $04
        .byte $06, $00, $06, $00, $09, $08, $0a, $0f
        .byte $07, $01, $07, $0f, $0a, $08, $09, $00
        .byte $06, $00, $06, $04, $00, $06, $04, $0e
        .byte $00, $06, $04, $0e, $0f, $00, $06, $04
        .byte $0e, $0f, $07, $00 ,$06, $04, $0e, $0f
        .byte $07, $01, $07, $0f, $0e, $04, $06, $00
        .byte $07, $0f, $0e, $04, $06, $00, $0f, $0e
        .byte $04, $06, $00, $0e, $04, $06, $00, $04
        .byte $06, $00, $06, $00, $09, $08, $0a, $0f
        .byte $07, $01, $07, $0f, $0a, $08, $09, $00



        .align 256
; cycle delay table
timing
        .fill 2, 0      ; don't touch this, raster code starts early
        .fill DYSP_HEIGHT - 2, 0





        .align 256
; number of cycles to skip in the branch
cycles
;       skip cycles     $d015                   sprite(s) active

        ; $00-$07
        .byte 0         ; $00 - %00000000       no sprites
        .byte 3         ; $01 - %00000001                     0
```

```
        .byte 5          ; $02 - %00000010                 1
        .byte 5          ; $03 - %00000011                 1 0

        ; $04-$07
        .byte 5          ; $04 - %00000100               2
        .byte 7          ; $05 - %00000101               2   0
        .byte 7          ; $06 - %00000110               2 1
        .byte 7          ; $07 - %00000111               2 1 0

        ; $08-$0b
        .byte 5          ; $08 - %00001000             3
        .byte 8          ; $09 - %00001001             3     0
        .byte 9          ; $0a - %00001010             3   1
        .byte 9          ; $0b - %00001011             3   1 0

        ; $0c-$0f
        .byte 7          ; $0c - %00001100             3 2
        .byte 9          ; $0d - %00001101             3 2   0
        .byte 9          ; $0e - %00001110             3 2 1
        .byte 9          ; $0f - %00001111             3 2 1 0

        ; $10-$13
        .byte 5          ; $10 - %00010000           4
        .byte 7          ; $11 - %00010001           4       0
        .byte 10         ; $12 - %00010010           4     1
        .byte 10         ; $13 - %00010011           4     1 0

        ; $14-$17
        .byte 9          ; $14 - %00010100           4   2
        .byte 11         ; $15 - %00010101           4   2   0
        .byte 11         ; $16 - %00010110           4   2 1
        .byte 11         ; $17 - %00010111           4   2 1 0

        ; $18-$1b
        .byte 7          ; $18 - %00011000           4 3
        .byte 10         ; $19 - %00011001           4 3     0
        .byte 11         ; $1a - %00011010           4 3   1
        .byte 11         ; $1b - %00011011           4 3   1 0

        ; $1c-$1f
        .byte 9          ; $1c - %00011100           4 3 2
        .byte 11         ; $1d - %00011101           4 3 2   0
        .byte 11         ; $1e - %00011110           4 3 2 1
        .byte 11         ; $1f - %00011111           4 3 2 1 0

        ; $20-$2f
        .byte $05, $08, $09, $09
        .byte $09, $0c, $0c, $0c
        .byte $09, $0c, $0d, $0d
        .byte $0b, $0d, $0d, $0d
```

```
        ; $30-$3f
        .byte $07, $09, $0c, $0c
        .byte $0b, $0d, $0d, $0d
        .byte $09, $0c, $0d, $0d
        .byte $0b, $0d, $0d, $0d

        ; $40-$4f
        .byte $05, $07, $0a, $0a
        .byte $0a, $0b, $0b, $0b
        .byte $0a, $0d, $0e, $0e
        .byte $0b, $0e, $0e, $0e

        ; $50-$5f
        .byte $09, $0b, $0e, $0e
        .byte $0d, $0f, $0f, $0f
        .byte $0b, $0e, $0f, $0f
        .byte $0d, $0f, $0f, $0f

        ; $60-$6f
        .byte $07, $0a, $0b, $0b
        .byte $0b, $0e, $0e, $0e
        .byte $0b, $0e, $0f, $0f
        .byte $0d, $0f, $0f, $0f

        ; $70-$7f
        .byte $09, $0b, $0e, $0e
        .byte $0d, $0f, $0f, $0f
        .byte $0b, $0e, $0f, $0f
        .byte $0d, $0f, $0f, $0f

        ; $80-$8f
        .byte $05, $08, $09, $09
        .byte $09, $0c, $0c, $0c
        .byte $09, $0d, $0d, $0d
        .byte $0c, $0d, $0d, $0d

        ; $90-$9f
        .byte $09, $0c, $0f, $0f
        .byte $0d, $10, $10, $10
        .byte $0c, $0f, $10, $10
        .byte $0d, $10, $10, $10

        ; $a0-$af
        .byte $09, $0c, $0d, $0d
        .byte $0d, $10, $10, $10
        .byte $0d, $10, $11, $11
        .byte $0f, $11, $11, $11

        ; $b0-$bf
        .byte $0b, $0d, $10, $10
        .byte $0f, $11, $11, $11
```

```
        .byte $0d, $10, $11, $11
        .byte $0f, $11, $11, $11

        ; $c0-$cf
        .byte $07, $09, $0c, $0c
        .byte $0c, $0d, $0d, $0d
        .byte $0c, $0f, $10, $10
        .byte $0d, $10, $10, $10

        ; $d0-$df
        .byte $0b, $0d, $10, $10
        .byte $0f, $11, $11, $11
        .byte $0d, $10, $11, $11
        .byte $0f, $11, $11, $11

        ; $e0-$ef
        .byte $09, $0c, $0d, $0d
        .byte $0d, $10, $10, $10
        .byte $0d ,$10, $11, $11
        .byte $0f, $11, $11, $11

        ; $f0-$ff
        .byte $0b, $0d, $10, $10
        .byte $0f, $11, $11, $11
        .byte $0d, $10, $11, $11
        .byte $0f, $11, $11, $11


; Clear the 'sprite enable' table, unrolled for speed
dysp_clear_timing_fast
        lda #0
.for row = 0, row < DYSP_HEIGHT, row = row + 1
        sta dysp_sprite_enable + row
.next
        rts


; Calculate the 'sprite enable' values for each raster line of the DYSP
;
; For each sprite, we ORA 21 bytes of the table with the bitmask for that
; particular sprite. The result of these calculations is used to look up the
; number of cycles to waste in the DYSP raster code
;
; Again unrolled for speed, but still takes a lot of raster time
dysp_calc_timing_fast
        lda sprite_positions + 1
        sec
        sbc #$32
        tax
.for row = 0, row < 21, row = row + 1
        lda dysp_sprite_enable + row,x
```

```
        ora #1
        sta dysp_sprite_enable + row,x
.next
        lda sprite_positions + 3
        sec
        sbc #$32
        tax
.for row = 0, row < 21, row = row + 1
        lda dysp_sprite_enable + row,x
        ora #2
        sta dysp_sprite_enable + row,x
.next
        lda sprite_positions + 5
        sec
        sbc #$32
        tax
.for row = 0, row < 21, row = row + 1
        lda dysp_sprite_enable + row,x
        ora #4
        sta dysp_sprite_enable + row,x
.next
        lda sprite_positions + 7
        sec
        sbc #$32
        tax
.for row = 0, row < 21, row = row + 1
        lda dysp_sprite_enable + row,x
        ora #8
        sta dysp_sprite_enable + row,x
.next
        lda sprite_positions + 9
        sec
        sbc #$32
        tax
.for row = 0, row < 21, row = row + 1
        lda dysp_sprite_enable + row,x
        ora #16
        sta dysp_sprite_enable + row,x
.next
        lda sprite_positions + 11
        sec
        sbc #$32
        tax
.for row = 0, row < 21, row = row + 1
        lda dysp_sprite_enable + row,x
        ora #32
        sta dysp_sprite_enable + row,x
.next
        lda sprite_positions + 13
        sec
        sbc #$32
```

```
        tax
.for row = 0, row < 21, row = row + 1
        lda dysp_sprite_enable + row,x
        ora #64
        sta dysp_sprite_enable + row,x
.next
        lda sprite_positions + 15
        sec
        sbc #$32
        tax
.for row = 0, row < 21, row = row + 1
        lda dysp_sprite_enable + row,x
        ora #128
        sta dysp_sprite_enable + row,x
.next

        ; update actual cycle skip table
.for row = 0, row < DYSP_HEIGHT, row = row + 1
        ldy dysp_sprite_enable + row
        lda cycles,y
        sta timing + 2 + row
.next

        rts




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

From:
https://codebase64.org/ - **Codebase 64 wiki**

Permanent link:
https://codebase64.org/doku.php?id=base:dysp_cycle_table&rev=1461678129

Last update: **2016-04-26 15:42**