Detect NTSC/PAL

Since you cannot rely on $02a6 to detect NTSC/PAL, you better do the check yourself. The theory behind these checks is simply that PAL and NTSC systems have different amounts of rasterlines, which can thus serve as the basis for a detection check. For alternative approach, especially useful if you also need to use TOD, check Efficient TOD initialisation page.

J0x variant

This snippet was written by J0x, as a result of a discussion on CSDb, which can be found in this thread. (In order to make it 105% reliable, you would have to catch NMI first or so.)

```
l1 lda $d012
l2 cmp $d012
   beq l2
   bmi l1
   cmp #$20
   bcc ntsc
```

Graham's variant

The easiest way to determine cycles/rasterline is to count the rasterlines:

```
312 rasterlines -> 63 cycles per line [PAL: 6569 VIC]
263 rasterlines -> 65 cycles per line [NTSC: 6567R8 VIC]
262 rasterlines -> 64 cycles per line [NTSC: 6567R56A VIC]
```

Counting the lines:

```
w0  LDA $D012
w1  CMP $D012
    BEQ w1
    BMI w0
```

Result in Akku (low byte of rasterlinecount-1):

```
#$37 -> 312 rasterlines
#$06 -> 263 rasterlines
#$05 -> 262 rasterlines
```

A slight improvement may be:

```
w0  LDA $D012
w1  CMP $D012
    BEQ w1
```

```
    BMI w0
    AND #$03
```

...and the results in the akku:

```
#$03 -> 312 rasterlines
#$02 -> 263 rasterlines
#$01 -> 262 rasterlines
```

EDIT: This check assumes that the machine is not a Drean PAL-N, which has 312 lines and 65 cycles per line.

Sokrates' variant

Graham's variant with additional Drean PAL-N detection. First count rasterlines, then count cycles to differ between PAL and PAL-N:

```
    SEI
    LDX #$00
w0  LDA $D012
w1  CMP $D012
    BEQ w1
    BMI w0
    AND #$03
    CMP #$03
    BNE detectionDone ; done for NTSC
    TAY
countCycles
    INX
    LDA $D012
    BPL countCycles
    CPX #$5E  ; VICE values: PAL-N=$6C PAL=$50
          ; so choose middle value $5E for check
    BCC isPAL
    INY ; is PAL-N
isPAL
    TYA
detectionDone
    ...
```

Results in the accumulator:

```
#$01: 262 rasterlines and 64 cycles per line [NTSC: 6567R56A VIC] (OLD NTSC)
#$02: 263 rasterlines and 65 cycles per line [NTSC: 6567R8 VIC]
#$03: 312 rasterlines and 63 cycles per line [PAL: 6569 VIC]
#$04: 312 rasterlines and 65 cycles per line [Drean PAL-N: 6572 VIC]
```

TLR's more advanced variant

TLR says: my routine is designed for the purpose of hardware analysis and emulator evaluation. I therefore require the number of cycles per line to be measured directly, not relying on the number of raster lines. That routine should work with any timing encountered as long as there are more than 256 raster lines on the system.

There are no known hardware with timings different from those listed by Graham so normally you can use his method. It will for almost all intents and purposes be equally accurate and shorter.

EDIT: nojoopa points out that the Drean PAL-N machine has 65 cycles per line and 312 raster lines so my statement about the known machines was wrong.

Code to be assembled with DASM:

```
;*******************************************************************************
;*
;* FILE  modesplit.asm
;* Copyright (c) 2010 Daniel Kahlin <daniel@kahlin.net>
;* Written by Daniel Kahlin <daniel@kahlin.net>
;*
;* DESCRIPTION
;*
;******
    processor 6502

LINE        equ    56

    seg.u    zp
;*******************************************************************************
;*
;* SECTION  zero page
;*
;******
    org    $02
ptr_zp:
    ds.w    1
tm1_zp:
    ds.b    1
tm2_zp:
    ds.b    1
    seg    code
    org    $0801
;*******************************************************************************
;*
;* Basic line!
;*
;******
StartOfFile:
    dc.w    EndLine
```

```
    dc.w    0
    dc.b    $9e,"2069 /T.L.R/",0
;           0 SYS2069 /T.L.R/
EndLine:
    dc.w    0

;*****************************************************************************
;*
;* SysAddress... When run we will enter here!
;*
;******
SysAddress:
    sei
    lda    #$7f
    sta    $dc0d
    lda    $dc0d
    jsr    check_time
    sta    cycles_per_line
    stx    num_lines

    jsr    test_present

    sei
    lda    #$35
    sta    $01

    ldx    #6
sa_lp1:
    lda    vectors-1,x
    sta    $fffa-1,x
    dex
    bne    sa_lp1

    lda    cycles_per_line
    sec
    sbc    #63
    bcc    sa_fl1        ;<63, fail
    cmp    #66-63        ;>=66, fail
    bcs    sa_fl1
    tax
    lda    time1,x
    sta    is_sm1+1

    jsr    test_prepare
    jsr    wait_vb

    lda    #$1b | (>LINE << 7)
    sta    $d011
    lda    #<LINE
    sta    $d012
    lda    #1
```

```
    sta    $d019
    sta    $d01a

    cli
sa_lp2:
    if    0
; be evil to timing to provoke glitches
    inx
    bpl    sa_lp2
    inc    $4080,x
    dec    $4080,x
    endif
    jmp    sa_lp2

vectors:
    dc.w    nmi_entry,0,irq_stable

sa_fl1:
nmi_entry:
    sei
    lda    #$37
    sta    $01
    jmp    $fe66

cycles_per_line:
    dc.b    0
num_lines:
    dc.b    0
time1:
    dc.b    irq_stable2_pal, irq_stable2_ntscold, irq_stable2_ntsc

;*****************************************************************************
;*
;* NAME  irq_stable, irq_stable2
;*
;******
irq_stable:
    sta    accstore    ; 4
    sty    ystore        ; 4
    stx    xstore        ; 4
    inc    $d019        ; 6
    inc    $d012        ; 6
is_sm1:
    lda    #<irq_stable2_pal    ; 2
    sta    $fffe        ; 4
    tsx             ; 2
    cli             ; 2
;10 * nop for PAL (63)
;11 * nop for NTSC (65)
    ds.b    11,$ea        ; 22
; guard
```

```
is_lp1:
    sei
    inc    $d020
    jmp    is_lp1

;28 cycles for NTSC (65)
;27 cycles for NTSCOLD (64)
;26 cycles for PAL (63)
irq_stable2_ntsc:
    dc.b    $2c        ; bit <abs>
irq_stable2_ntscold:
    dc.b    $24        ; bit <zp>
irq_stable2_pal:
    dc.b    $ea
    jsr    twelve        ; 12
    jsr    twelve        ; 12
;---
    txs            ; 2
    dec    $d019        ; 6
    dec    $d012        ; 6
    lda    #<irq_stable    ; 2
    sta    $fffe        ; 4    =46
    lda    $d012        ; 4
    eor    $d012        ; 4
    beq    is2_skp1    ; 2 (3)
is2_skp1:

    jsr    test_perform
accstore    equ    .+1
    lda    #0
xstore  equ    .+1
    ldx    #0
ystore  equ    .+1
    ldy    #0
    rti

;*******************************************************************************
;*
;* NAME  wait_vb
;*
;******
wait_vb:
wv_lp1:
    bit    $d011
    bpl    wv_lp1
wv_lp2:
    bit    $d011
    bmi    wv_lp2
    rts

;*******************************************************************************
```

```
;*
;* NAME  check_time
;*
;* DESCRIPTION
;*   Determine number of cycles per raster line.
;*   Acc = number of cycles.
;*   X = LSB of number of raster lines.
;*
;******
check_time:
    lda    #0
    sta    $dc0e
    jsr    wait_vb
;--- raster line 0
    lda    #$fe
    sta    $dc04
    sta    $dc05        ; load timer with $fefe
    lda    #%00010001
    sta    $dc0e        ; start one shot timer
ct_lp1:
    bit    $d011
    bpl    ct_lp1
;--- raster line 256
    lda    $dc05        ; timer msb
    eor    #$ff        ; invert
; Acc = cycles per line
;--- scan for raster wrap
ct_lp2:
    ldx    $d012
ct_lp3:
    cpx    $d012
    beq    ct_lp3
    bmi    ct_lp2
    inx
; X = number of raster lines (LSB)
twelve:
    rts

;*****************************************************************************
;*
;* NAME  test_present
;*
;******
test_present:
    lda    #14
    sta    646
    sta    $d020
    lda    #6
    sta    $d021

    lda    #<label_msg
```

```
    ldy    #>label_msg
    jsr    $ab1e

    lda    #1
    sta    646

    lda    #NAME_POS
    sta    $d3
    lda    #<name_msg
    ldy    #>name_msg
    jsr    $ab1e
    lda    #CONF_POS
    sta    $d3
    lda    #0
    ldx    cycles_per_line
    jsr    $bdcd
    inc    $d3
    lda    #1
    ldx    num_lines
    jsr    $bdcd

    rts

NAME_POS    equ    2
CONF_POS    equ    32
name_msg:
    dc.b    "modesplit",29,"r01",0
label_msg:
    dc.b    147,"01234567890123456789012345678901234567890123456789",19,0

;***************************************************************************
;*
;* NAME  test_prepare
;*
;******
test_prepare:

; set up screen
    ldx    #0
prt_lp1:
    lda    #$5f
    sta    $0428,x
    sta    $0500,x
    sta    $0600,x
    sta    $06e8,x
    lda    #14
    sta    $d828,x
    sta    $d900,x
    sta    $da00,x
    sta    $dae8,x
    inx
```

```
    bne    prt_lp1

    jsr    adjust_timing

    lda    #$17
    sta    $d018

    rts



;***************************************************************************
;*
;* NAME  adjust_timing
;*
;******
adjust_timing:
    lda    cycles_per_line
    sec
    sbc    #63
    tax
    lda    time2,x
    sta    tm1_zp
    lda    time3,x
    sta    tm2_zp
    lda    #<test_start
    sta    ptr_zp
    lda    #>test_start
    sta    ptr_zp+1
    ldx    #>[test_end-test_start+255]
at_lp1:
    ldy    #0
    lda    #$d8        ; cld
    cmp    (ptr_zp),y
    bne    at_skp1
    iny
    cmp    (ptr_zp),y
    bne    at_skp1
    dey
    lda    tm1_zp
    sta    (ptr_zp),y
    iny
    lda    tm2_zp
    sta    (ptr_zp),y
at_skp1:
    inc    ptr_zp
    bne    at_lp1
    inc    ptr_zp+1
    dex
    bne    at_lp1
```

```
    rts

; eor #$00 (2),  bit $ea (3),  nop; nop (4)
time2:
    dc.b    $49, $24, $ea
time3:
    dc.b    $00, $ea, $ea

;******
; end of line marker
    mac    EOL
    ds.b    2,$d8
    endm
;******
; One 8 char high chunk
    mac    CHUNK
    ldy    #$08
    bne    .+4
.lp1:
    ds.b    9,$ea
    sty    $d016
    lda    #7
    sta    $d021
    lda    #6
    sta    $d021
    ds.b    7,$ea
    EOL

    jsr    {1}

    ldx    #$1b
    stx    $d011
    sty    $d016
    ds.b    1,$ea
    EOL

    iny
    cpy    #$10
    bne    .lp1
    endm
    align    256
test_start:
;*************************************************************************
;*
;* NAME  test_perform
;*
;******
test_perform:
    ds.b    6,$ea
; start 1
    CHUNK     section1
```

```
; start 2
    CHUNK     section2
; start 3
    CHUNK     section3
; end
    bit    $ea
    ds.b    4,$ea
    lda    #$1b
    sta    $d011
    lda    #$08
    sta    $d016
    lda    #7
    sta    $d021
    lda    #6
    sta    $d021
    rts

    align    256


;*******************************************************************************
;*
;* NAME  section1
;*
;******
section1:
    repeat    6
    EOL
    ds.b    2,$ea
    ldx    #$1b
    stx    $d011
    sty    $d016
    tya
    ora    #%00010000
    sta    $d016        ; mc
    ldx    #$5b        ; illegal text
    stx    $d011
    ldx    #$3b        ; bitmap
    stx    $d011
    and    #%11101111
    sta    $d016        ; hires
    ldx    #$7b
    stx    $d011        ; illegal bitmap1
    ldx    #$5b
    stx    $d011        ; ECM
    ds.b    3,$ea
    bit    $ea
    repend
    rts

;*******************************************************************************
```

```
;*
;* NAME  section2
;*
;******
section2:
    repeat    6
    EOL
    ds.b    2,$ea
    ldx    #$1b
    stx    $d011
    sty    $d016

    ldx    #$5b
    stx    $d011
    ldx    #$1b
    stx    $d011
    ds.b    16,$ea
    bit    $ea
    repend
    rts

;*********************************************************************************
;*
;* NAME  section3
;*
;******
section3:
    repeat    6
    EOL
    ds.b    2,$ea
    ldx    #$1b
    stx    $d011
    sty    $d016
    tya
    ora    #%00010000
    sta    $d016
    and    #%11101111
    sta    $d016
    ds.b    15,$ea
    bit    $ea
    repend
    rts

test_end:

; eof
```

TWW's Variant

Count's number of cycles on one scan with CIA timer and uses the 2 LSBs from the high byte of the CIA Timer to determine model. This reliably detects PAL, NTSC, NTSC2 and DREAN. routine exits with result in A. Make sure no interrupts occur during the runtime of the routine.

```
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Detect PAL/NTSC
    //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // 312 rasterlines -> 63 cycles per line PAL        => 312 * 63 = 19656
Cycles / VSYNC  => #>76  %00
    // 262 rasterlines -> 64 cycles per line NTSC V1    => 262 * 64 = 16768
Cycles / VSYNC  => #>65  %01
    // 263 rasterlines -> 65 cycles per line NTSC V2    => 263 * 65 = 17095
Cycles / VSYNC  => #>66  %10
    // 312 rasterlines -> 65 cycles per line PAL DREAN  => 312 * 65 = 20280
Cycles / VSYNC  => #>79  %11

DetectC64Model:

    // Use CIA #1 Timer B to count cycled in a frame
    lda #$ff
    sta $dc06
    sta $dc07  // Latch #$ffff to Timer B

    bit $d011
    bpl *-3    // Wait untill Raster > 256
    bit $d011
    bmi *-3    // Wait untill Raster = 0

    ldx #%00011001
    stx $dc0f  // Start Timer B (One shot mode (Timer stops automatically
when underflow))

    bit $d011
    bpl *-3    // Wait untill Raster > 256
    bit $d011
    bmi *-3    // Wait untill Raster = 0

    sec
    sbc $dc07  // Hibyte number of cycles used
    and #%00000011
    rts
```

Older routine

Ninja/The dreams presented a PAL/NTSC routine in Go64!/CW Issue 06/2000, together with a short article. Technically, there are no obvious benefits using this one compared to the shorter one above,

but it might still serve an educational purpose (especially since it was supplement code for an article, which is also included below).

A reliable PAL/NTSC check!

From Go64!/CW Issue 06/2000.

By Wolfram Sang (Ninja/The Dreams - www.the-dreams.de)

“Where am I from?” - this quite philosophic question is not uninteresting for C64 programmers. Even those without a SuperCPU may learn something here.

Our beloved Commodores exist in PAL- and NTSC versions, what means they have differences in their cycle clock and picture generation. Let us check the facts:

System	VIC-Type-No.	Cycles per rasterline	Raster lines per screen	Screen-refreshrate	Clock cycle
NTSC	6567	65	263	~60Hz	1022727 Hz
PAL	6569	63	312	~50Hz	985248 Hz

A program which relies on this must know on which kind of system it is currently running. The kernal offers the memory location $02A6 for that (0 = NTSC, 1 = PAL), which is accepted as reliable enough by many programmers. Unfortunately, this method has two major disadvantages: First it can lead to wrong results, since this location is only set in the Reset routine ($ff5b - $ff6a). Afterwards the value can be modified (by mistake or intentionally), for example with a simple POKE command. Second, this value will always indicate an NTSC machine when a SuperCPU is running and has been in 20 MHz mode during Reset. To explain that, let's take a further look at the system detection method of the kernal routine.

A good idea...

As you can see from the above table, a PAL-C64 generates more rasterlines than an NTSC one, exactly 312 instead of 263. To check this, just write a number of a rasterline only existent on PAL into the latch $D011/12 (for example $137 = dec. 311). In the Interrupt Request Register (IRR) at $D019 bit 0 we can see whether there have been the same values in the latch and in the real current rasterline. We just wait until one entire screen has been built up and then look if the rasterline $137 was displayed. Remember: This is only possible on PAL systems! So we have an absolutely sure decision criterium.

... badly realized

Why this so sure method now fails with a SuperCPU? Well, this is the fault of Commodore. The responsible kernal routine doesn't wait until a full screen has been built up, it performs some other tasks meanwhile (clear screen etc.). On a standard C64 it is absolutely sure that these tasks take more time than one screen build-up. Unfortunately, the developers at Commodore couldn't know that

a 20 MHz turbo board, introduced 15 years later, would be too fast to ever reach rasterline $137. The following routine solves the problem. It delivers a 100% sure identification of the system, even with a SuperCPU in turbo mode. However, only 6510 opcodes were used, so that this routine will also work on an unexpanded C64.

The improved version

Let's look at our new routine step by step. First we disable the interrupts and set the NMI vector to an RTI - we need silence for the detection of the video mode, any interrupt could lead to a wrong result. Now we wait until $d012 equals zero, meaning that we are either in rasterline 0 or 256. Which one doesn't matter, this check only prevents that the routine is called when the rasterline is close to $137, which could lead to an error under certain circumstances. Now we write the number of our test-line into the latch register. Afterwards we reset the bit 0 in the IRR to remove a maybe existing old request. With the following waiting loop we make sure that an entire screen was really built up. Since we check the rasterline itself for this, this routine will probably still work in 15 years with the 200 MHz ultra turbocard. Now the only thing we need to do is mask out the crucial bit in the IRR, reset $D019 and leave the routine. The value in the accu now represents PAL or NTSC, just like in $02A6, but much more reliable.

Finished!

As you can see, this check is neither long nor complicated. I urgently recommend to use this one instead of $02A6. Because who likes an X to be treated as a U?

Source Code

```
 ; Reliable PAL/NTSC-Detector by Ninja/The Dreams/TempesT
 ; for Go64!/CW issue 06/2000 (detailed description there)
 ; This routine can't be fooled like $02a6 and works also with a SCPU


              include standard.c64

nmivec        = $0318                    ; NMI-vector

              org $0801

              adr $080b, 64
              byt $9e,"2061",0,0,0       ; Basic-line

jmp_in:
              lda #lo(text)
              ldy #hi(text)
              jsr $ab1e                  ; print startup-message

              jsr palntsc                ; perform check
              sta $02a6                  ; update KERNAL-variable
              beq ntsc                   ; if accu=0, then go to NTSC
```

```
              lda #lo(pal_text)
              ldy #hi(pal_text)           ; otherwise print PAL-text
              jmp $ab1e                   ; and go back.
ntsc:
              lda #lo(ntsc_text)
              ldy #hi(ntsc_text)          ; print NTSC-text
              jmp $ab1e                   ; and go back.

palntsc:
              sei                         ; disable interrupts
              ldx nmivec
              ldy nmivec+1                ; remember old NMI-vector
              lda #lo(rti)
              sta nmivec
              lda #hi(rti)                ; let NMI-vector point to
              sta nmivec+1                ; a RTI
wait:
              lda $d012
              bne wait                    ; wait for rasterline 0 or 256
              lda #$37
              sta $d012
              lda #$9b                    ; write testline $137 to the
              sta $d011                   ; latch-register
              lda #$01
              sta $d019                   ; clear IMR-Bit 0
wait1:
              lda $d011                   ; Is rasterbeam in the area
              bpl wait1                   ; 0-255? if yes, wait
wait2:
              lda $d011                   ; Is rasterbeam in the area
              bmi wait2                   ; 256 to end? if yes, wait
              lda $d019                   ; read IMR
              and #$01                    ; mask Bit 0
              sta $d019                   ; clear IMR-Bit 0
              stx nmivec
              sty nmivec+1                ; restore old NMI-vector
              cli                         ; enable interrupts
              rts                         ; return

rti:          rti                         ; go immediately back after
                                          ; a NMI


              cascii
text:
              byt $93,$05,$0e,$0d
              byt "Reliable PAL/NTSC-Detector",$0d
              byt "by Ninja/The Dreams in 2000",$0d
              byt "for GO64!/CW-Magazine.",$0d,$0d
              byt $9b,"You have a ",0
pal_text:
              byt "PAL-machine.",$0d,$05,0
```

```
ntsc_text:
              byt "NTSC-machine.",$0d,$05,0

              end $0801
```

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

Permanent link:
https://codebase64.org/doku.php?id=base:detect_pal_ntsc

Last update: **2019-05-15 20:11**