Cycle Exact Measuring of Execution Times

In most cases one will measure how long certain subroutines take to execute by changing the border colors. This is usually sufficient to see how many rasters are wasted, but sometimes you want to know the exact number of cycles spent, or the routine in question takes more than a frame to execute, causing the color changes overlap in a way that makes it difficult to see where the execution starts and ends. For these cases the CIA timers come in handy:

```
;ZEN-TIMER 64, 6502tass v1.31 version. Original idea by M. Abrash. Usage:

;  jsr measure    or <start address>+0 if precompiled   to start cycle
counting
;  jsr evaluate   or <start address>+3 if precompiled   to stop counting &
print result

;Note: max cycle count range is limited to about 65.500 cycles (=roughly 3
frames)


overhead        = 19            ;cycles wasted by the timer itself during
measurement
irqs_allowed    = 0             ;1 to allow them (less accurate results)
dma_off         = 1             ;0 to allow badlines (dito)
sprites_off     = 1             ;0 to allow sprites (dito)
printout        = $400          ;0 to use $bdcd, <address> to write directly
to screen
                                ;(or some other location to look it up via
ml-mon)


;* = $1000      ;uncomment to precompile to wanted address


                jmp measure

evaluate        sei
                lda #0
                sta $dc0f
                lda vald011
                sta $d011
                lda vald015
                sta $d015
                cld
                sec
                lda #<($ffff-overhead)
                sbc $dc06
                sta locycles
                lda #>($ffff-overhead)
                sbc $dc07
```

```
.if !printout
                ldx locycles
                jsr $bdcd
                lda #13
                jsr $ffd2
                lda statusreg   ;restore (most of) st
                pha
                plp
                rts
.else

                ldy locycles    ;lame hex to petscii conversion
                ldx #$30-1
                stx ten1000s
                stx ten1000s+1
                stx ten1000s+2
                stx ten1000s+3
                stx ten1000s+4

                sec
hploop          sta temp
                inc ten1000s-$30+1,x
                tya
                sbc lo,x
                tay
                lda temp
                sbc hi,x
                bcs hdloop

                tya
                adc lo,x
                tay
                inx
                cpx #$34
                sec
                bne hploop+3

                ldx #4
print           lda ten1000s,x
                sta printout,x
                lda $d021
                eor #8
                sta (printout//$400)+$d800,x
                dex
                bpl print

                lda statusreg   ;restore (most of) st
                pha
                plp
                rts
```

```
temp            .byte 0           ;needed for hb
ten1000s        .byte 0,0,0,0,0
lo = *-$30+1
.byte <10000,<1000,<100,<10,<1
hi = *-$30+1
.byte >10000,>1000,>100,>10,>1

.fi

locycles        .byte 0
vald015         .byte 0
vald011         .byte 0
statusreg       .byte 0

measure         php             ;save st, just in case
                sei
                pla
                sta statusreg
                lda $d011
                sta vald011
                lda $d015
                sta vald015
                ldx #$00
                stx $dc0f       ;stop timer b (not really necessary, but
still)
.if dma_off
                stx $d011
.fi
.if sprites_off
                stx $d015
.fi

                dex
                cpx $d012
                bne *-3         ;wait for vblank area
                stx $dc06       ;set to $ffff
                stx $dc07
                lda #$19

.if irqs_allowed
                cli
.fi

                sta $dc0f       ;start timer b, one shot mode
                rts
```

So for example, if you had to find out how many cycles your latest uberbrilliant sprite-sorting algo takes, you could do that like this:

```
                jsr initdata   ;prepare test case for your sorting algo
```

```
                jsr measure    ;start cycle counting
                jsr sortalgo
                jsr evaluate   ;stop count & print out cycle count
```

Note that the zen-timer can't be used for really slow routines as it can only count up to about 65.500 cycles. For those routines you should use the extended timer below which chains both CIA1 timers together but thereby doesn't behave that well in an environment that uses the timer a irq (e.g the kernal - you might want to change the program to use CIA2 for that):

```
;LNG-TIMER 64, 6502tass version. Original idea by M. Abrash. Extended
;version for extra-slow routine evaluation. Doesn't like timer interrupts
;& output is in hex for simplicity's sake. Usage:

;  jsr measure    or <start adress>+0 if precompiled   to start cycle
counting
;  jsr evaluate   or <start adress>+3 if precompiled   to stop counting &
print result


overhead        = 19            ;cycles wasted by the timer itself during
measurement
dma_off         = 1             ;0 to allow badlines (dito)
sprites_off     = 1             ;0 to allow sprites (dito)
printout        = $400          ;where to write the result

;* = $1000      ;uncomment to precompile to wanted address


                jmp measure

evaluate        sei
                lda #0
                sta $dc0e
                sta $dc0f
                lda vald011
                sta $d011
                lda vald015
                sta $d015
                cld
                sec
                lda #<($ffff-overhead)
                sbc $dc04
                sta cycles
                lda #>($ffff-overhead)
                sbc $dc05
                sta cycles+1
                lda #$ff
                sbc $dc06
                sta cycles+2
```

```
                lda #$ff
                sbc $dc07
                sta cycles+3
                ldx #3
                ldy #0
 showresult     lda cycles,x
                lsr
                lsr
                lsr
                lsr
                jsr toscreen
                lda cycles,x
                and #$0f
                jsr toscreen
                dex
                bpl showresult
                lda statusreg   ;restore (most of) st
                pha
                plp
                rts

toscreen        sed             ;simple hex to hexpetscii conversion,
                cmp #$0a        ;courtesy of Frank Kontros
                adc #$30
                cld
                sta printout,y
                lda $d021
                eor #$08
                sta (printout//$400)+$d800,y
                iny
                rts

cycles         .byte 0,0,0,0,0
vald015         .byte 0
vald011         .byte 0
statusreg       .byte 0

measure         php             ;save st, just in case
                sei
                pla
                sta statusreg
                lda $d011
                sta vald011
                lda $d015
                sta vald015
                ldx #$00
                stx $dc0e       ;stop timers
                stx $dc0f
.if dma_off
                stx $d011
.fi
```

```
.if sprites_off
                stx $d015
.fi

                dex
                cpx $d012
                bne *-3         ;wait for vblank area
                stx $dc04       ;set timers to $ffffffff
                stx $dc05
                stx $dc06
                stx $dc07
                lda #$59
                sta $dc0f       ;reload and set timer b to count timer a
underflow
                lda #$11
                sta $dc0e       ;reload and start timer a, continuous mode
                rts
```