

Timer

→ 3 funções:

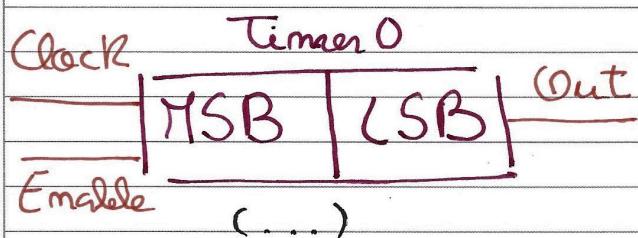
1. int timer_test_read_config(uint8_t timer,
enum timer_status_field field)

2. int timer_test_time_base(uint8_t timer,
uint32_t freq)

3. int timer_test_int(uint8_t timer)

• Every PC "has" an i8254, an integrated circuit (IC) with 3 timers.

• The 3 timers are identical and operate independently of one another.



• Se o timer estiver enabled, o reloj do counter diminui com cada "pulse" da clock.

• O reloj do out depende do counter e da configuração de operação do timer.

→ Operating mode 3 = Square Wave generator

$$\text{frequência} = \frac{\text{Clock}}{\text{dine}} \rightarrow \text{freq. da clock}$$

↓ realça

da frequência para da freq. materna

→ Programar com timer

• Cada timer tem um contador de 16 bits para os quais se pode ler ou escrever.

• Existe com único control register para os quais se pode escrever, que é usado para configurar as operações de todos os timers.

Programas com timer implica:

1 → especifica a modo de operação escrevendo a control word para o control register

2 → dar load da regra inicial da counter, escrevendo para o control register

Control Word:

F 6

Select Counter
00 → 0
01 → P
P0 → 2

5 4

Initialization mode
01 → LSB
10 → HSB
Pp → LSB, HSB

3 2 1

Operation mode
000 → 0 P00 → 4
00P → 1 P0P → 5
0P0 → 2
0PP → 3

0

Counting base
0 → Binary
1 → BCD

F and 6, especificam a timer a programar

0 especifica se o counter é binário ou BCD

1, 2 e 3, especificam o modo de operação

5 e 6, apesar dos caímentos terem 16 bits, é parcialmente inicializada 8 bits de cada regras, logo para dar load da regra inicial da counter é necessário introduzir os LSB e os HSB separadamente.

→ ler as configurações de um timer

Escrive com comando especial: Read-Back command

F6

Read-Back Command → ff (distinção entre uma control word e um read-back command)

5

Read Counter value → 0 (ler counter value)
1 (m ler counter value)

4

Read Programmed value → 0 (ler programmed value)
1 (m ler programmed value)

3

Select Counter 1

2

Select Counter 2

1

Select Counter 3

0

Reserved

→ Após enviar o Read-Back Command para o control register, pode ler-se o counter value ou o programmed mode no counter.

Status Byte:

bits 0 a 5 da última control word colocada no control register, a 7 a initialization mode, operation mode e a counting base.

F → Output → regra da contagem do timer

6 → Null Counter

5, 4 → Type of Access

3, 2, 1 → Programmed mode

0 → BCD

→ O timer 0 usa a IRQ line 0 do interrupt controller.

→ PC's interrupt hardware

Os hardware interrupts são processados pela priority interrupt controller (PIC).

1º Notifica os device drivers (DD) interessados quando o interrupt ocorre, para isto é necessário suscender o interrupt.

sys_irq_set_policy(int irq_line, int policy, int *hook_id)
[irq_line] → irq line da device

[policy] → informa o GTH que pode mandar o interrupt para o EOI

[hook_id] → no momento do input: um id para o Kernel usar ma interrupt notification
Como output: um id para ser usada pelo DD dentro Kernel calls da interrupt

2º Para modificar os DD sobre a ocorrência de um interrupt o GTH utiliza notificações.

→ os processos recebem e emitem mensagens para comunicar um com os outros

→ a notif. é um tipo especial de mensagem utilizada para comunicar da Kernel para um user process

3º desuspende, modificações da device na final:

```

int ipe_status;
message msg;
while (...) {
    if ((n = driver_recieve (Any, &msg, &ipe_status)) != 0) {
        printf ("driver_recieve failed with: %d", n);
        continue;
    }
    if (is_ipe_notify (ipe_status)) {
        switch (_ENDPOINT_P(msg.m_source)) {
            case Hardware: hardware interrupt notification
                verifica se a → if (msg.m_notify & ing-not) {
                    msg está a ser (...)}
                    da disponibilidade break;
                    que é quando ing_lime
                    pertence. Case default: não tem hardware com interrupt
                    break;
        }
    }
}

```

elre { } m receber uma notif só como mensagem standard, m não espera das mensagens standard, faz m receber uma mensagem

↑
 message received

driver_recieve (Any, &msg, &ipe_status)

receber (notif) notif/
 mensagem de qualquer dispositivo

↳ ipe status

_ENDPOINT_P(msg.m_source)
 msg.m_source tem como end point com address q
 comunica a destino e origem da mensagem

`int (timer_get_conf)(uint8_t timer, uint8_t *st)`

→ função: ler a config. do timer "timer"

1º → Configurar de o timer é realizada

2º → Como queremos sair a configuração do timer, temos de enviar a read-back word de modo a que Read-Programmed - redere esteja a 0

3º → Enviar o Read-Back command para o control register

4º → Ler a config. a partir do timer e coloca no st.

Nota: Comandos para escrever: `sys_outb(port, command)`
Comando para ler: `sys_inb(port, status)`

!! O status não em 32 bits mas só os 8 mais significativos, logo circun-20 tem só 8 bits
sys_outb(sys_outb(0x12, 0x00) q denotam só os 8 bits mais significativos.

`int (timer_display_conf)(uint8_t timer, uint8_t st, enum timer_status_field field)`

→ função: organiza as configurações do timer numa struct enum timer_status_field para de forma serem imprimida na tela.

1º → Organizar o status de maneira a retinar aquela que interessa

2º → Criar um objecto do tipo enum timer_status_field_val.

enum timer_status_field_val conf;

Switch(field){

case Tsf-all:

conf.bYTE = ...;

break;

enum timer_status_field:

- tsf-all
- tsf-initial
- tsf-mode
- tsf-base

enum timer_status_field_real:

→ uint8_t bgte

- enum timer_init_im-mode
- uint8t count-mode
- bool bcd

3º Usa a função timer-point-config() para dar print à configuração.

(int(timer->ref-frequency))(uint8-t timer, uint32-t freq)

→ função: muda a frequência do timer "timer"

1º A frequência colocada tem de estar entre 1G e a frequência loaded no timer, caso contrário o nº não é representável em 16 bits que se fizer a divisão

2º Vamos trabalhar c/ o operating mode 3 que é o modo default dos timers no Timix, logo aí continuam a control word que vamos guardar o operating mode e o BCD atuais.

→ Lembrando anteriormente os bits 0 a 5 da status são iguais aos bits 0 a 5 da control word

a) escrever a read-back word e oca o status

3º Construir a control word. Somente queremos alterar a freq. Bit 5 e 4 tem de estar a 1

4º Encontrar a control word para o control register

5º Calcular a freq. à qual temos de dar laod para os timers

$$\text{dine} = \frac{\text{clock}}{\text{freq}}$$

6º Encontrar o dine para os controlo registos da timer "timers". Pº a LSBe de seguido a MSB.

`int(timer-test-int)(uint8-t timer)`

→ função: subscrever, lidar e dessubscrever interrupções geradas pelos timers.

7º Subscrever a dispositiva. IRQ line do timer 0 é 0.

→ necessário criar uma mask.

$$\text{uint8-t bit_ma} = (\text{mº entre } 0 \text{ e } 7);$$

$$\text{uint32-t irq_ret} = \text{Bit}(\text{bit_ma});$$

→ mask_id = bit_ma;

→ Policy = IRQ-REENABLE

2º ter a categoria que frequência está a 60Hz.

3º Criar a interrupt handle. Este só regui incrementa um reacíonel global counter a cada interrupt. (Timer gera um interrupt a cada ciclo de relógico, logo quando o counter for igual a 60 param a reação).

4º Coloca como condição de paragem que $\text{counter} < \text{frequency} * \text{time}$)

5º desvelar crecer a interrupt.



by Staples

9