Insper

Computação Embarcada - Handout - Tick! Tack! Tick!

Rafael Corsi - rafael.corsi@insper.edu.br

Março - 2018

(entregar até 26/3)

O firmware disponível em "SAME70-TickTack" configura o TimerCounter (TC) e o RTC do mircontrolador. O TC0 canal 1 é configurado para gerar uma interrupção (*TC1_Handler*) a cada 250ms (f=1/T -> de 4Hz) já o RTC é configurado para operar em modo de alarme, gerando uma interrupção (*RTC_Handler*) em um determinado momento. Inicialmente o RTC está configurado para gerar uma interrupção um minuto após o início do microcontrolador.

O TimerCounter faz com o o led pisque na frequência de 4Hz enquanto não ocorrer o alarme do RTC, após o acontecimento do alarme (interrupção do RTC) o piscar do led é desligado.

Entenda o código

Analise o código recebido, quais periféricos estão em uso ? O que o firmware faz ? Discuta com os colegas. Embarque e verifique o seu funcionamento.

```
1
```

- Quantas interrupções são usada, quais são elas ?

Periféricos envolvidos :

- Power Managment Controller (PMC)
- Parallel Input/Output Controller (PIO)
- Real Time Clock (RTC)
- Timer Counter (TC)

Timer Counter - TC

O TC é um periférico de contagem de pulsos, esses pulsos podem ser provenientes do mundo externo (abrir e fechar de portas, encoder de um motor, pulsos de um PWM) ou interno ao próprio microcontrolador, utilizando o clock como sinal de entrada. Esse contador é formado por 16 bits, limitando o valor máximo que pode contar em : 2^16 -1 = 65535.

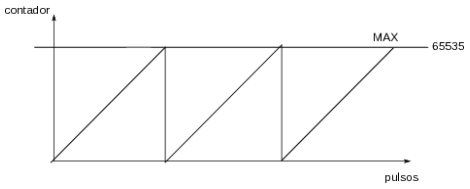

Figure 1: Timer Counter

No modo de contagem de clocks, o TC operar como um cronometro utilizando o clock do microcontrolador como sua base de tempo, ou seja, a cada pulso de clock o valor do contador é incrementado de um.

Se usarmos o clock base do sistema (300 Mhz) na entrada do TC, ou seja, 300 milhões de pulsos em um segundo, poderíamos contar até no máximo 218us (x = 65535/300M). Para possibilitar uma maior flexibilidade ao TC, podemos optar por divir o clock de entrada (300Mhz) em alguns prescales : 1/1, 1/8, 1/32, 1/128. Possibilitando que o TC conte por períodos mais longos.

A cada vez que o contador é reiniciado (quando atinge o seu valor máximo) uma interrupção (TCx_Handler) é gerada:

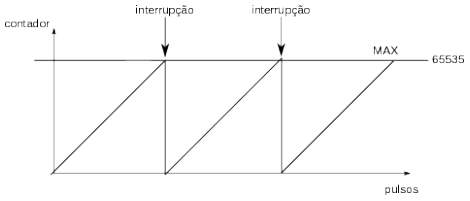

Figure 2: Timer Counter

Cada canal possui um registrador chamado de RC que possibilita o reset do contador em um valor diferente que o máximo. Esse registrador é utilizado para gerar a frequência desejada.

Cada TC possui três canais (canal 0, canal 1, canal 2), podendo cada canal operar de maneira independente, porém com a mesma configuração de prescale, conforme ilustração a seguir :

A função a seguir configura o TC0, canal 1 para operar com uma frequência de 4Hz :

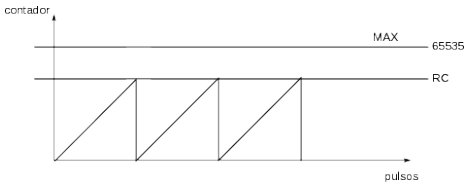

Figure 3: Timer Counter

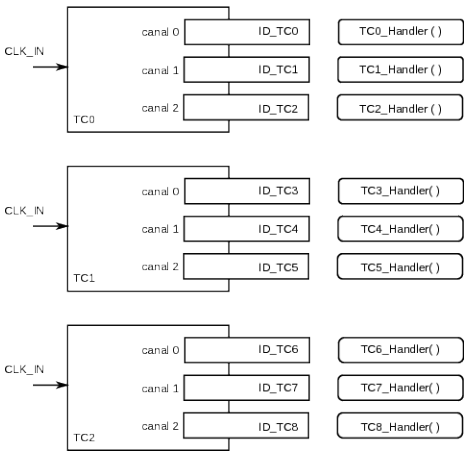

Figure 4: Timer Counter

```c
/**
* Configura TimerCounter (TC) para gerar uma interrupcao no canal (ID_TC e TC_CHANNEL)
* na taxa de especificada em freq.
*/
void TC_init(Tc * TC, int ID_TC, int TC_CHANNEL, int freq){
    uint32_t ul_div;
    uint32_t ul_tcclks;
    uint32_t ul_sysclk = sysclk_get_cpu_hz();

    uint32_t channel = 1;

    /* Configura o PMC */
    /* O TimerCounter é meio confuso
    o uC possui 3 TCs, cada TC possui 3 canais
    TC0 : ID_TC0, ID_TC1, ID_TC2
    TC1 : ID_TC3, ID_TC4, ID_TC5
    TC2 : ID_TC6, ID_TC7, ID_TC8
    */
    pmc_enable_periph_clk(ID_TC);

    /** Configura o TC para operar em  4Mhz e interrupçcão no RC compare */
    tc_find_mck_divisor(freq, ul_sysclk, &ul_div, &ul_tcclks, ul_sysclk);
    tc_init(TC, TC_CHANNEL, ul_tcclks | TC_CMR_CPCTRG);
    tc_write_rc(TC, TC_CHANNEL, (ul_sysclk / ul_div) / freq);

    /* Configura e ativa interrupçcão no TC canal 0 */
    /* Interrupção no C */
    NVIC_EnableIRQ((IRQn_Type) ID_TC);
    tc_enable_interrupt(TC, TC_CHANNEL, TC_IER_CPCS);

    /* Inicializa o canal 0 do TC */
    tc_start(TC, TC_CHANNEL);
}
```

Interrupção

Sempre que houver um reset no contador do TC, a interrupção referente ao canal é chamada, nessa interrupção fazemos a mudança no status do led.

```c
/**
 *  Interrupt handler for TC1 interrupt.
 */
  void TC1_Handler(void){
  volatile uint32_t ul_dummy;

  /****************************************************************
```

```
  * Devemos indicar ao TC que a interrupção foi satisfeita.
  ******************************************************************/
  ul_dummy = tc_get_status(TC0, 1);

  /* Avoid compiler warning */
  UNUSED(ul_dummy);

  /** Muda o estado do LED */
  if(flag_led0)
      pin_toggle(LED_PIO, LED_PIN_MASK);
}
```

Real Time Clock - RTC

O RTC é um periférico do uC que serve para contar tempo com resolução de segundos, ele possui toda a lógica interna de um relógio, contando anos, dias, meses, horas, minutos e segundos.

O RTC é configurado na função RTC_init() :

```
void RTC_init(){
    /* Configura o PMC */
    pmc_enable_periph_clk(ID_RTC);

    /* Default RTC configuration, 24-hour mode */
    rtc_set_hour_mode(RTC, 0);

    /* Configura data e hora manualmente */
    rtc_set_date(RTC, YEAR, MOUNTH, DAY, WEEK);
    rtc_set_time(RTC, HOUR, MINUTE, SECOND);

    /* Configure RTC interrupts */
    NVIC_DisableIRQ(RTC_IRQn);
    NVIC_ClearPendingIRQ(RTC_IRQn);
    NVIC_SetPriority(RTC_IRQn, 0);
    NVIC_EnableIRQ(RTC_IRQn);

    /* Ativa interrupcao via alarme */
    rtc_enable_interrupt(RTC,  RTC_IER_ALREN);
}
```

Alarme

O alarme é uma maneira de configurarmos o RTC para gerar uma interrupção em uma determinada data, no nosso caso, para um minuto apos a inicialização do micrcontrolador :

```
main(){
...
...

/* configura alarme do RTC */
rtc_set_date_alarm(RTC, 1, MOUNTH, 1, DAY);
rtc_set_time_alarm(RTC, 1, HOUR, 1, MINUTE+1, 1, SECOND);

}
```

Interrupção

Sempre que a situação do alarme for satisfeita, o RTC_Handler será chamado. Na interrupção configura a variável global flag_led0 para 0.

```
/**
 * \brief Interrupt handler for the RTC.
 */
void RTC_Handler(void)
{
  /* Qual parte do RTC gerou interrupção ? */
  uint32_t ul_status = rtc_get_status(RTC);

  /* seta led para parar de piscar */
  flag_led0 = 0;

  /* Informa que interrupção foi tratada */
  rtc_clear_status(RTC, RTC_SCCR_ALRCLR);
}
```

Programando

1. Sleep Mode

Faça o uC entrar em modo sleep enquanto estiver ocioso.

Implemente e teste no uC.

2. Flag é a melhor maneira ?

A decisão se o LED está em modo “pisca pisca” é feita por uma variável global :

```
/*****************************************************************************/
/* VAR globais                                                               */
/*****************************************************************************/
uint8_t flag_led0 = 1;
```

Dentro da interrupção do TC1 verificamos a flag :

```
    /** Muda o estado do LED */
    if(flag_led0)
        pin_toggle(LED_PIO, LED_PIN_MASK);
```

O problema aqui é que a interrupção do TC1 continua ocorrendo mesmo com o piscar do LED desativado, o que pode ter um impacto no consumo.

Proponha e implemente uma solução para essa questão.

Implemente e teste no uC.

3. Piscar durante 1 minuto e parar durante 1 minuto - cíclico

Faça com que o led pisque durante um minuto e fique um minuto sem piscar, faça isso de forma cíclica como na ilustração a baixo :

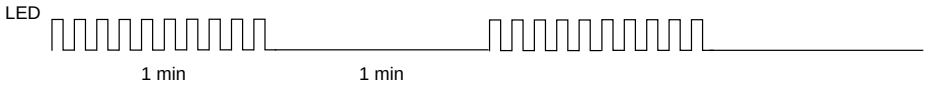

Figure 5: Led da placa

Implemente e teste no uC.

4. Várias frequências

Utilizando a placa OLED1 conectada ao kit de desenvolvimento, faça com que cada LED pisque nas frequências determinadas na tabela a baixo, utilize para cada LED um TC diferente.

LED OLED1	Frequência (Hz)
LED 1	8
LED 2	11
LED 3	17

Será preciso verificar a quais pinos os LEDs e Botões dessa placa estão conectados. Para isso abra o manual do OLED1 e verifique a pinagem.

! Implemente e teste no uC.

5. Botões

Faça com que os botões (relacionados a cada LED) pare ou inicialize o piscar dos LEDs, utilize para isso interrupção do PIO.

! Implemente e teste no uC.