Timers Counters e Interrupts

As abordagens efectuadas neste tutorial consideram-se com base da utilização das placas Arduino que incorporam os microcontroladores Atmel AVR.

O assunto é muito vasto e complexo, quer a nível de programação, quer a nível do entendimento da arquitectura destes itens nos chips. Por essa razão, trata-se apenas de um ligeiro apontamento e de uma ideia do que é e como funcionam.

A utilização destas técnicas são imprescíndiveis e extremamente vantajosas quando se pretende construir um projecto electrónico onde se verificam interacções com o exterior, nos dois sentidos (input e ouput) como é o caso dos robots.

Quando um computador (sem processamento paralelo!) está em funcionamento e a executar as instruções desenvolvidas/previstas de um programa informático, essa execução é efectuada seguindo o fluxo dessas instruções.

A questão que se põe é quando ou se for necessário saber, por exemplo, se um sensor é activado e como tratar a situação. Num fluxo de programação, pode-se evocar a qualquer momento uma rotina que efectue a leitura do

respectivo pin de input e consoante a natureza dessa leitura, isto é, se pelos valores lidos, o sensor enviou ao computador dados significativos, ou, por outro lado, essa leitura não traduz qualquer conteúdo.

No entanto, enquadrar o nosso propósito nesta técnica não se mostra adequado, dado o tempo gasto para ler os inputs e tratar a respectiva informação, com que intervalo de tempo, em que parte do programa se deve prever o desencadear dessas leituras, o tempo perdido em leituras sem conteúdo, a perda do input por não ter sido detectado no momento certo, etc.. Todos estes processos acontecem a altas velocidades, em micro-segundos, dependendo da frequência do relógio do sistema, configuração (por software) dos timers, etc..

O ideal seria dispôr de um processo à margem do processamento normal e sem afectar em vão o fluxo de instruções que o processador está a executar, e garantir a leitura e o respectivo tratamento desses inputs.

É evidente que esta questão é fundamental para o processador garantir a execução atempada de outro tipo de eventos.

O processo é um pouco à semelhança da pessoa que está no local de trabalho a executar o seu plano de actividades,

nas quais se inclui a tarefa de atender o telefone. Não faria sentido algum o empregado estar sempre, ou de vez em quando, a pegar no telefone, e muito menos com a preocupação de quando é que o mesmo vai tocar, se tocar. O que o empregado deve fazer é:

1. execução normal do seu plano de actividades,
2. quando o telefone tocar,
3. interromper o que está a fazer,
4. sinalizar o ponto do trabalho onde foi interrompido,
5. atender o telefone,
6. dar seguimento ao que resultar desse atendimento,
7. desligar o telefone,
8. voltar imediata e exactamente ao ponto do trabalho interrompido + 1,
9. prosseguir com a sua execução.

Nos computadores a lógica é, em termos simplistas, a mesma, com as complexidades que advém de a implementar em sistemas informáticos, na vertente do software e do hardware!

Efectivamente, um dos meios para resolver o problema é o recurso aos timers e interrupts.

Interrupts:

Em termos simples, um interrupt é um sinal que interrompe a actividade normal do processador. Este sinal de interrupção (interrupt request signal) pode ser a nível interno do CPU, através de um timer, ou externo através da alteração do estado do pin de input ligado a um periférico (sensor, receptor rc, etc.).

Se o CPU foi configurado para esta funcionalidade, reage com:

1. verifica-se um interrupt,
2. fecho de instrução que está a executar,
3. guarda os valores de contadores, status de informação, registos, flags, etc.,
4. começa a executar o código (ISR) associado ao interrupt,
5. fim da execução do código associado ao interrupt,
6. recupera os valores dos contadores, status de informação, flags, etc.,
7. recomeçar o processamento na posição do código que estava a executar, imediatamente antes da interrupção.

O programa principal que está a ser executado "nem tem" que se dar conta do que aconteceu. Houve um momento em que algo aconteceu, mas que não tem consequências

directas... a menos que estivesse a executar instruções onde o factor tempo (na ordem dos micro-segundos...) é crucial!

Os interrupts podem ocorrer em qualquer altura da execução do código de um programa (normalmente isso ocorre entre duas instruções) e é um processo totalmente independente desta execução.

Entretanto, ao recorrer a estas técnicas tem que se ter muito cuidado, por inúmeras razões, pois se existe a hipótese de o CPU estar a executar um programa, e, em qualquer momento o interrupt interromper o mesmo, pode dar-se o caso de perda ou alteração de valores gravados em determinados registos.

Quando um interrupt se verifica, é evocada uma função ISR - Interrupt Service Routine"

Quando é evocada uma função ISR, todos os restantes interrupts são desactivados e são automaticamente activados logo que esse ISR finaliza. Não é necessário explicitar no código as funções interrupt e noInrerrupts.

Não esquecer que as variáveis dentro do código de uma ISR devem ser declaradas com o tipo "volatile":

por exemplo:

volatile unsigned long PulseCounts;

Esta questão tem a ver com a forma como o compilador reconhece e trata as variáveis no momento em que converte o código fonte em código máquina para o CPU o poder executar. Esta particularidade constituí uma matéria muito extenta, e tem a ver com os processos de optimização observados no momento da compilação do código.

É importante que o código que consta num ISR associado a um interrupt não seja muito extenso.

Em termos de programação de um computador, o que se passa, é que, enquanto está a ser processada pelo CPU, se uma flag muda de valor, é porque houve um interrupt que foi entretanto activado, executou um determinado bloco de código (ISR), e guardou algures os valores que daí resultaram, para quando e se forem necessários poderem ser utilizados pelo programa principal.

Tudo se vai complicando quando existem diversos interrupts, prioridades, origens diferentes para um mesmo interrupt, interacções entre os mesmos, etc.

Os interrupts podem ser "Hardware interrupts", por exemplo em resposta a um pin de input quando muda de high para low, ou vice-versa, e “Software interrupts” quando ocorre em resposta a uma instrução enviada por software.

Como controlar os interrupts no Arduino:

-interrupt

-noInrerrupts

o programa principal terá este aspecto:

```
void setup()
{
      //todo o conjunto de definições, atributos,etc.
      //próprios deste bloco
}
void loop()
{
  noInterrupts();
      //aqui devem ser tratados os passos tendo em conta
      //que tudo o que acontecer pode-se tornar crítico
  interrupts();
      // código corrente
}
```

Como utilizar interrupts externos no Arduino:

-attachInterrupt

-detachInterrupt

Síntaxe:

attachInterrupt(interrupt, function, mode)

attachInterrupt(pin, function, mode) (só para *Arduino Due*)

Parâmetros

interrupt:	the number of the interrupt (*int*)	
pin:	the pin number	(*Arduino Due only*)
function:	the function to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an *interrupt service routine*.	
mode:	defines when the interrupt should be triggered. Four contstants are predefined as valid values: **LOW** to trigger the interrupt whenever the pin is low, **CHANGE** to trigger the interrupt whenever the pin changes value **RISING** to trigger when the pin goes from low to high, **FALLING** for when the pin goes from high to low.	

Exemplo:

```
int pin = 13;
volatile int state = LOW;

void setup()
{
  pinMode(pin, OUTPUT);
  attachInterrupt(0, blink, CHANGE);
}

void loop()
{
  digitalWrite(pin, state);
}
```

External interrupts pin assignments - Arduino Mega2560

Interrupt:	Pin:
0	2
1	3
2	21
3	20
4	19
5	18

Exemplo de como implementar um interrupt num programa para Arduino:

```
#include "avr/interrupt.h"
void setup(void)
{
        pinMode(2, INPUT);
        digitalWrite(2, HIGH);
        sei();
        EIMSK |= (1 << INT0);
        EICRA |= (1 << ISC01);
}
void loop(void)
{
        //código normal do nosso programa
}
ISR(EXT_INT0_vect)
{
        digitalWrite(13, !digitalRead(13));
}
```

Se estivermos a trabalhar com interrupts externos, pode-se, em alternativa, usar o seguinte método:

```
void setup()
{
        attachInterrupt(0, minhaISR,FALLING);
}
```

Timers

Um timer é um processo que nos permite efectuar a medida de um determinado intervalo de tempo.
Os timers funcionam independentemente do outro código que está a ser executado e com base no incremento um registo contador. Este contador conta até um determinado valor, dependendo do seu tamanho. Quando atinge o valor máximo o contador entra em overflow, reseta e volta ao valor zero, ao mesmo tempo que activa uma flag que poderá ser usada para se saber que isso aconteceu, ou associar o timer a um interrupt que desencadeará a execução de uma ISR, que por sua vez fará o reset da flag e o processo repetir-se-á.

Como se incrementa o ***counter***:

O contador (*counter register*) tem que ter sempre acesso a um relógio "clock source" que está constantemente a gerar um sinal.
Este pode ser um clock source externo, mas geralmente usa-se o relógio interno do chip como clock source.

Sempre que o timer detecta este sinal incrementa o contador em um.

A medida possível mais pequena de usar é o período desse relógio.

O período (T) é o inverso da frequência (f) de um dispositivo.

Assim, se a frequência do relógio do microcontrolador é de 16 MHz, significa que:

o sistema gera 16 * 10^6 = 16000000 ciclos por segundo

T=1/f = 1/16000000 = 0,0000000625 segundos, que é a resolução máxima conseguida.

Como o exemplo é de um timer de 16 bits, significa que a cada 0,0041 segundos (=0,0000000625 * 65535) um ISR pode ser executado.

Porque, nalguns casos, estes podem ser períodos rápidos demais para na prática poderem ser utilizados, existem técnicas que possibilitam aumentar o valor dos mesmos (ver tabela 16.5).

Tipos de timers:

Depende do chip que se uilizar. No caso do AVR ATmega328 que se encontra instalado, por exemplo, no Arduino duemilanove.

Timer0

De 8 bits, o que significa que pode registar valores até 255. É utilizado em algumas funções do Arduino, por exemplo, delay() pelo que se deve considerar e ter muito cuidado no desenvolvimento de um programa.

Timer1

De 16 bits, o que significa que pode registar valores até 65535 (=2^16 -1). Tem que se ter o mesmo cuidado que o anterior porque é utilizado por algumas bibliotecas, por exemplo, *"Servo.h"*

Timer2

De 8 bits, o que significa que pode registar valores até 255. É utilizado em algumas funções do Arduino, por exemplo, tone() pelo que se deve considerar e ter muito cuidado no desenvolvimento de um programa.

No caso do Arduino Mega2560, temos mais 3 timers:

Timer3

Timer4

Timer5

De 16 bits, o que significa que pode registar valores até 65535.

Configuração dos timers:

Para se poder utilizar os timers, tem que se proceder à sua definição (no programa que se está a desenvolver) e gravar nos registos do chip os respectivos valores. Cada timer tem um determinado número de registos. Dois destes registos guardam os valores anteriores de setup: TCCR1A e TCCR1B (Timer/Counter Control Register)

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Para começar a utilizar o timer, as mais importantes configurações são os últimos 3 bits em TCCR1B, CS12, CS11 e CS10. Actuando sobre estes bits podemos obter diversas combinações e pôr o timer a correr com diversas velocidades.

Por defeito, estes bits estão definidos com zero.

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Para mais informações e detalhes sobre estas matérias e projectos de robótica:

O meu sítio na internet: *http://carlos-ch-santos.net/*

O meu blog: *http://carlosch5908.wordpress.com/*