Interrupções e Arduino

Definição

Em nível de hardware, uma interrupção é um sinal de algum dispositivo ou software que indica à CPU interrromper o processo que está executando e passe a executar um outro processo. Por exemplo, isso ocorre quando uma tecla do teclado é acionada, o programa que está em execução (processo) no processador é interrompido e um novo processo é executado para tratar a tecla pressionada.

Interrupções no Arduino

No arduino, não são todos os pinos que podem receber uma interrupção, no Arduino Uno, são permitidos interrupções apenas nos pinos 2 e 3, veja a tabela dos pinos habilitavéis para interrupção:

Modelo	Pinos digitais utilizáveis para interrupções
Uno, Nano, Mini, outro baseado em 328	2, 3
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, outro baseado em 32u4	0, 1, 2, 3, 7
Zero	Todos os pinos digitais, exceto 4
MKR1000 Rev.1	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Vencimento	todos os pinos digitais
101	Todos os pinos digitais (Apenas pinos 2, 5, 7, 8, 10, 11, 12, 13 trabalham com CHANGE)

Usando Interrupções

As interrupções são úteis para que as tarefas sejam execuadas automaticamente em programas de microcontroladores e podem ajudar a resolver problemas de temporização. Podendo ser utilizado ler um codificador rotativo ou monitorar a entrada do usuário.

Rotinas de serviço de interrupção (Interrupt Service Routines - ISR)

Os ISRs são tipos especiais de funções que possuem algumas limitações únicas que a maioria das outras funções não possuem. Um ISR não pode ter nenhum parâmetro, e eles não devem retornar nada.

Geralmente, um ISR deve ser tão curto e rápido quanto possível. Se programa tem vários ISRs , apenas um pode ser executado de cada vez, outras interrupções serão executadas depois que o atual terminar em uma ordem que depende da prioridade que eles possuem.

Dentro da função que será chamada na interrupção, a função delay() não funcionará e a função *millis()* não será incrementado

***delayMicroseconds()* não usa nenhum contador, portanto ela funcionará normalmente durante uma interrupçao**

Variáveis tipicamente globais são usadas para passar dados entre um ISR e o programa principal. Para garantir que as variáveis compartilhadas entre um ISR e o programa principal sejam atualizadas corretamente, declare-as como volatile .

Você deve declarar como volátil quaisquer variáveis que você utilizar dentro da função chamada pela interrupção .

Sintaxe:

attachInterrupt (digitalPinToInterrupt (pino), ISR, modo); // recomendado

attachInterrupt (interrupção, ISR, modo); // não recomendado

attachInterrupt (pino, ISR, modo); // Não recomendado para Arduino Due, Zero, MKR1000 , 101

Onde:

pino: o número do pino conectado (no Arduino Uno só poderá ser os pinos 2 e 3);

ISR: o ISR para ligar quando ocorrer a interrupção; Esta função não deve ter parâmetros e não devolve nada. Esta função às vezes é referida como uma rotina de serviço de interrupção.

modo: define quando a interrupção deve ser acionada. Quatro constantes são predefinidas como valores válidos:

- **LOW:** dispara a interrupção sempre que o pino tiver nível baixo;
- **CHANGE**: dispara a interrupção quando houver mudança no estado do pino;
- **RISING:** dispara a interrupção quando o pino tiver uma mudança do estado LOW para HIGH;

- **FALLING**: dispara a interrupção quando o pino tiver uma mudança do estado HIGH para LOW;

Exemplo:

```
const byte ledPin = 13 ;
const byte interruptPin = 2 ;
volatile byte state = LOW ;

void setup ( ) {
   pinMode ( ledPin , OUTPUT ) ;
   pinMode ( interruptPin , INPUT_PULLUP ) ;
   attachInterrupt ( digitalPinToInterrupt ( interruptPin ) , blink , CHANGE ) ;
}

void loop ( ) {
   digitalWrite ( ledPin , state ) ;
}

void blink ( ) {
   state = ! state ;
}
```