Universidade Tecnológica Federal do Paraná – UTFPR
Bacharelado em Ciência da Computação

BCC34C – Sistemas Microcontrolados

Prof. Frank Helbert Borsato

Interrupções

- **Interrupção:**
  - **É um processo pelo qual um dispositivo externo ou interno pode interromper a execução de uma determinada tarefa do microcontrolador e solicitar a execução de outra**
    - Elas permitem que um programa responda a determinados eventos no momento em que eles ocorrem

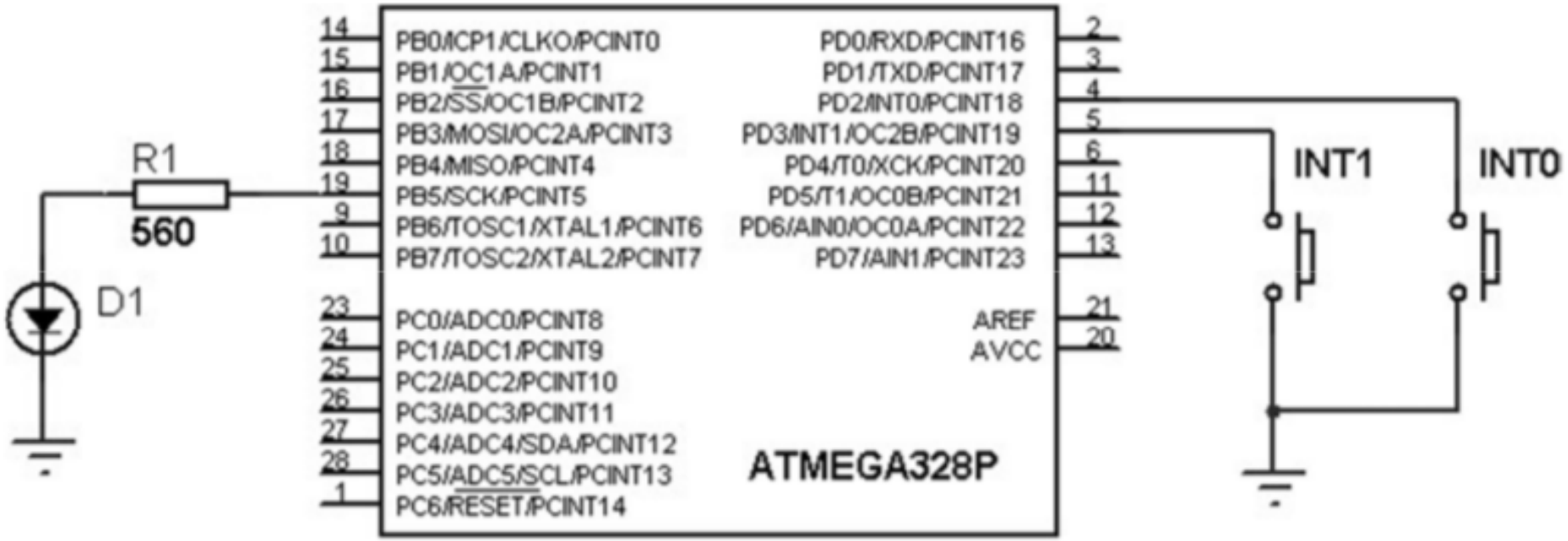


Fig. 6.2 – Circuito para emprego das interrupções externas INT0 e INT1.

```
ISR(INT0_vect);
ISR(INT1_vect);
//-------------------------------------------------------------------------------
int main()
{
     DDRD  = 0x00;        //PORTD entrada
     PORTD = 0xFF;        //pull-ups habilitados
     DDRB  = 0b00100000;  //somente pino do LED como saída
     PORTB = 0b11011111;  //desliga LED e habilita pull-ups

     UCSR0B = 0x00;       /*necessário desabilitar RX e TX para trabalho com os pinos
                                                          do PORTD no Arduino*/
     EICRA = 1<<ISC01;//interrupções externas: INT0 na borda de descida, INT1 no nível zero.
     EIMSK = (1<<INT1) | (1<<INT0);//habilita as duas interrupções
     sei();          //habilita interrupções globais, ativando o bit I do SREG

     while(1){}
}
//-------------------------------------------------------------------------------
ISR(INT0_vect) //interrupção externa 0, quando o botão é pressionado o LED troca de estado
{
     cpl_bit(PORTB,LED);
}
//-------------------------------------------------------------------------------
ISR(INT1_vect) //interrupção externa 1, mantendo o botão pressionado o LED pisca
{
     cpl_bit(PORTB,LED);
     _delay_ms(200);      //tempo para piscar o LED
}
//===============================================================================
```

Interrupções

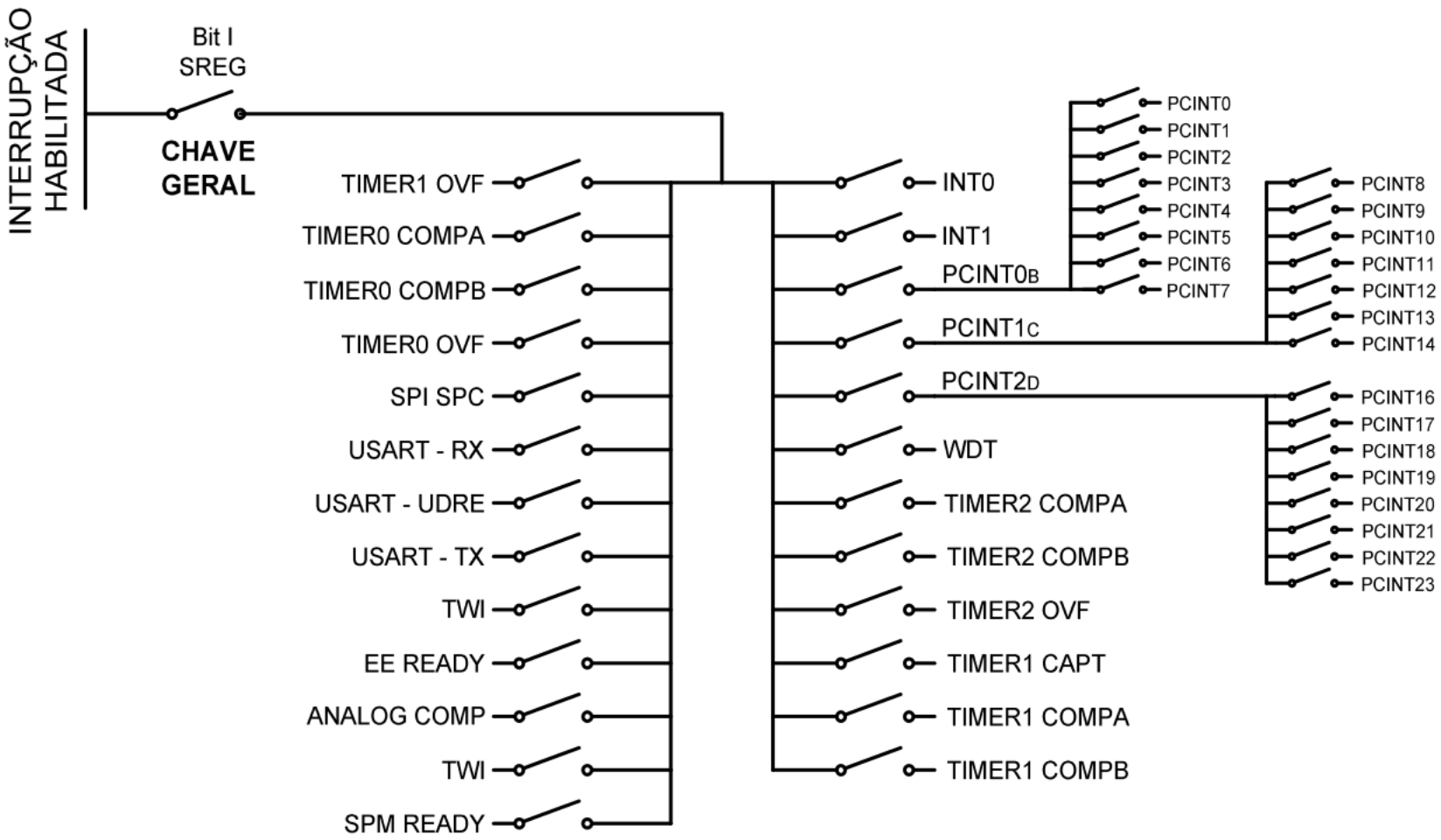


Fig. 6.1 – Chaves de habilitação das interrupções.

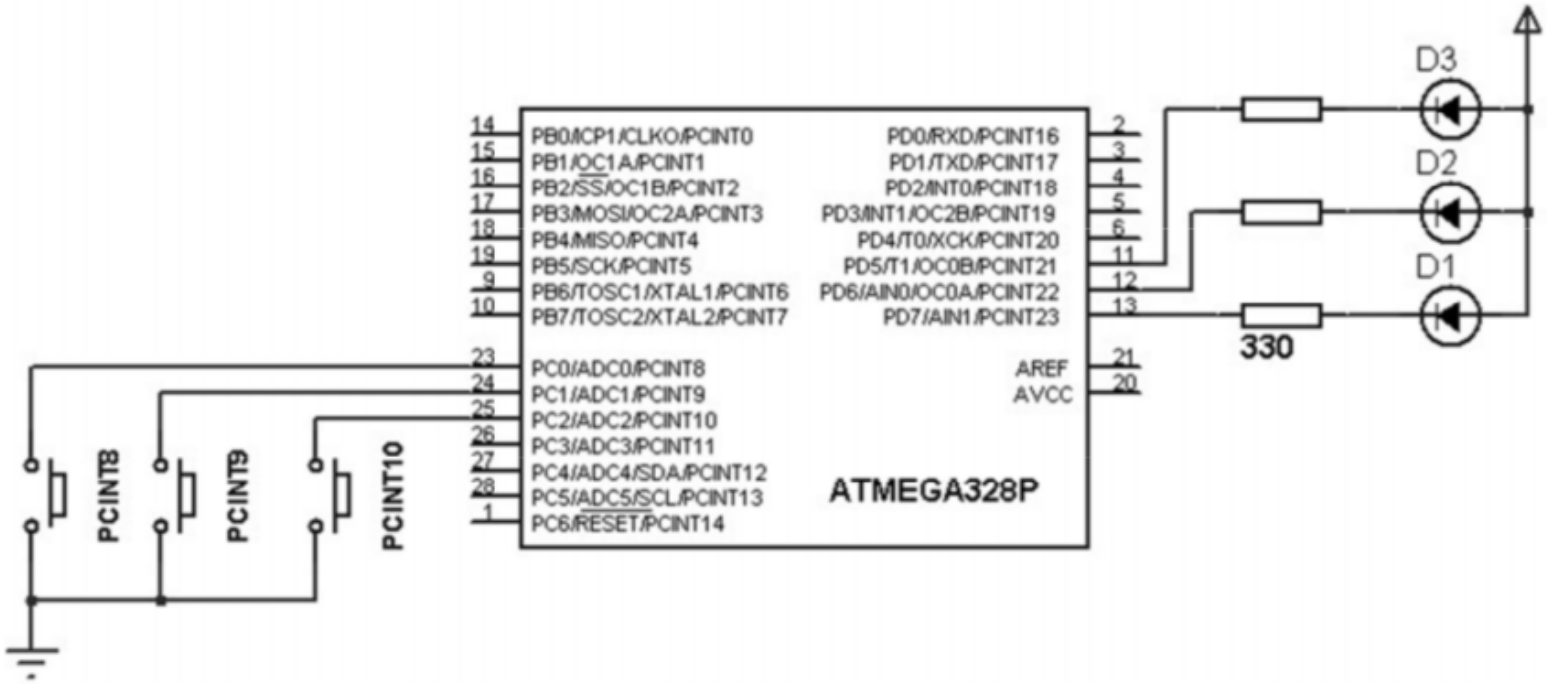


Fig. 6.3 – Circuito para teste das interrupções externas PCINT8:10.

Interrupções

- **Interrupções no ATMega328**
  - **As interrupções no AVR são 'vetoradas'**
    - Cada interrupção tem um endereço de atendimento fixo na memória de programa
      - » **Se cada interrupção não possuísse esse endereço fixo:**
        - » Um único endereço seria destinado para todas as interrupções
        - » Para executar a tarefa de uma determinada interrupção, no caso de haver mais de uma habilitada, o programa teria que testar qual foi a interrupção ocorrida
      - » **Havendo um endereço fixo por interrupção, o programa se torna mais eficiente**

Interrupções

- **Interrupções no ATMega328**
  - **O endereço de cada interrupção na memória de programa do ATmega328**
    - Possui 26 diferentes tipos de interrupções
    - A ordem dos endereços determina o nível de prioridade das interrupções
      - » **Quanto menor o endereço do vetor de interrupção, maior será a sua prioridade**
    - Por exemplo:
      - » **A interrupção INT0 tem prioridade sobre a INT1**

Tab. 6.1 – Interrupções do ATmega328 e seus endereços na memória de programa.

Vetor	End.	Fonte	Definição da Interrupção	Prioridade
1	0x00	RESET	Pino externo, Power-on Reset, Brown-out Reset e Watchdog Reset	
2	0x01	INT0	interrupção externa 0	
3	0x02	INT1	interrupção externa 1	
4	0x03	PCINT0	interrupção 0 por mudança de pino	
5	0x04	PCINT1	interrupção 1 por mudança de pino	
6	0x05	PCINT2	interrupção 2 por mudança de pino	
7	0x06	WDT	estouro do temporizador *Watchdog*	
8	0x07	TIMER2 COMPA	igualdade de comparação A do TC2	
9	0x08	TIMER2 COMPB	igualdade de comparação B do TC2	
10	0x09	TIMER2 OVF	estouro do TC2	
11	0x0A	TIMER1 CAPT	evento de captura do TC1	
12	0x0B	TIMER1 COMPA	igualdade de comparação A do TC1	
13	0x0C	TIMER1 COMPB	igualdade de comparação B do TC1	
14	0x0D	TIMER1 OVF	estouro do TC1	
15	0x0E	TIMER0 COMPA	igualdade de comparação A do TC0	
16	0x0F	TIMER0 COMPB	igualdade de comparação B do TC0	
17	0x10	TIMER0 OVF	estouro do TC0	
18	0x11	SPI, STC	transferência serial completa - SPI	
19	0x12	USART, RX	USART, recepção completa	
20	0x13	USART, UDRE	USART, limpeza do registrador de dados	
21	0x14	USART, TX	USART, transmissão completa	
22	0x15	ADC	conversão do ADC completa	
23	0x16	EE_RDY	EEPROM pronta	
24	0x17	ANA_COMP	comparador analógico	
25	0x18	TWI	interface serial TWI – I2C	
26	0x19	SPM_RDY	armazenagem na memória de programa pronta	

Power-on Reset: ocorre na energização enquanto a fonte de alimentação estiver abaixo da tensão limiar de power-on reset (V POT).
Reset externo: ocorre quando o pino de reset é aterrado (0 V) por um determinado período de tempo.
Watchdog Reset: ocorre quando o watchdog está habilitado e o seu contador atinge o valor limite.
Brown-out Reset: ocorre quando a tensão de alimentação cair abaixo do valor definido para o brown-out reset (V BOT) e o seu detector estiver habilitado.

Interrupções

- **Toda vez que uma interrupção ocorre, a execução do programa é interrompida:**
  - A CPU completa a instrução em andamento
    - » **Carrega na pilha o endereço da próxima instrução que seria executada (endereço de retorno)**
  - Desvia para a posição de memória correspondente à interrupção
  - O código escrito no endereço da interrupção é executado até o programa encontrar o código RETI (Return from Interruption)
    - » **É carregado no PC o endereço de retorno armazenado na pilha**
    - » **O programa volta a trabalhar a partir do ponto que parou antes da ocorrência da interrupção**

Interrupções

- **Ao atender uma interrupção, o microcontrolador desabilita todas as outras interrupções que possam estar habilitadas**
  - Zerando o bit I do SREG (chave geral das interrupções)
- **Ao retornar da interrupção (encontrar a instrução RETI)**
  - Ele coloca novamente o bit I em 1, permitindo que outras interrupções sejam atendidas

- **Se uma ou mais interrupções ocorrerem neste período**
  - Os seus bits de sinalização serão colocados em 1 e o microcontrolador as tratará de acordo com a ordem de prioridade de cada uma
    - » **O detalhe é que o AVR executará sempre uma instrução do programa principal antes de atender qualquer interrupção em espera**

Interrupções

- **Ao atender uma interrupção, o microcontrolador desabilita todas as outras interrupções que possam estar habilitadas**
  - Zerando o bit I do SREG (chave geral das interrupções)

SREG – *STATUS REGISTER*

Bit	7	6	5	4	3	2	1	0
SREG	I	T	H	S	V	N	Z	Z
Lê/Escreve	L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

Bit 7 – I: *Global Interrupt Enable.*
Bit 6 – T: *Bit Copy Storage.*
Bit 5 – H: *Half Carry Flag.*
Bit 4 – S: *Sign Bit*, S = N ⊕ V.
Bit 3 – V: *Two's Complement Overflow Flag.*
Bit 2 – N: *Negative Flag.*
Bit 1 – Z: *Zero Flag.*
Bit 0 – C: *Carry Flag.*

Interrupções

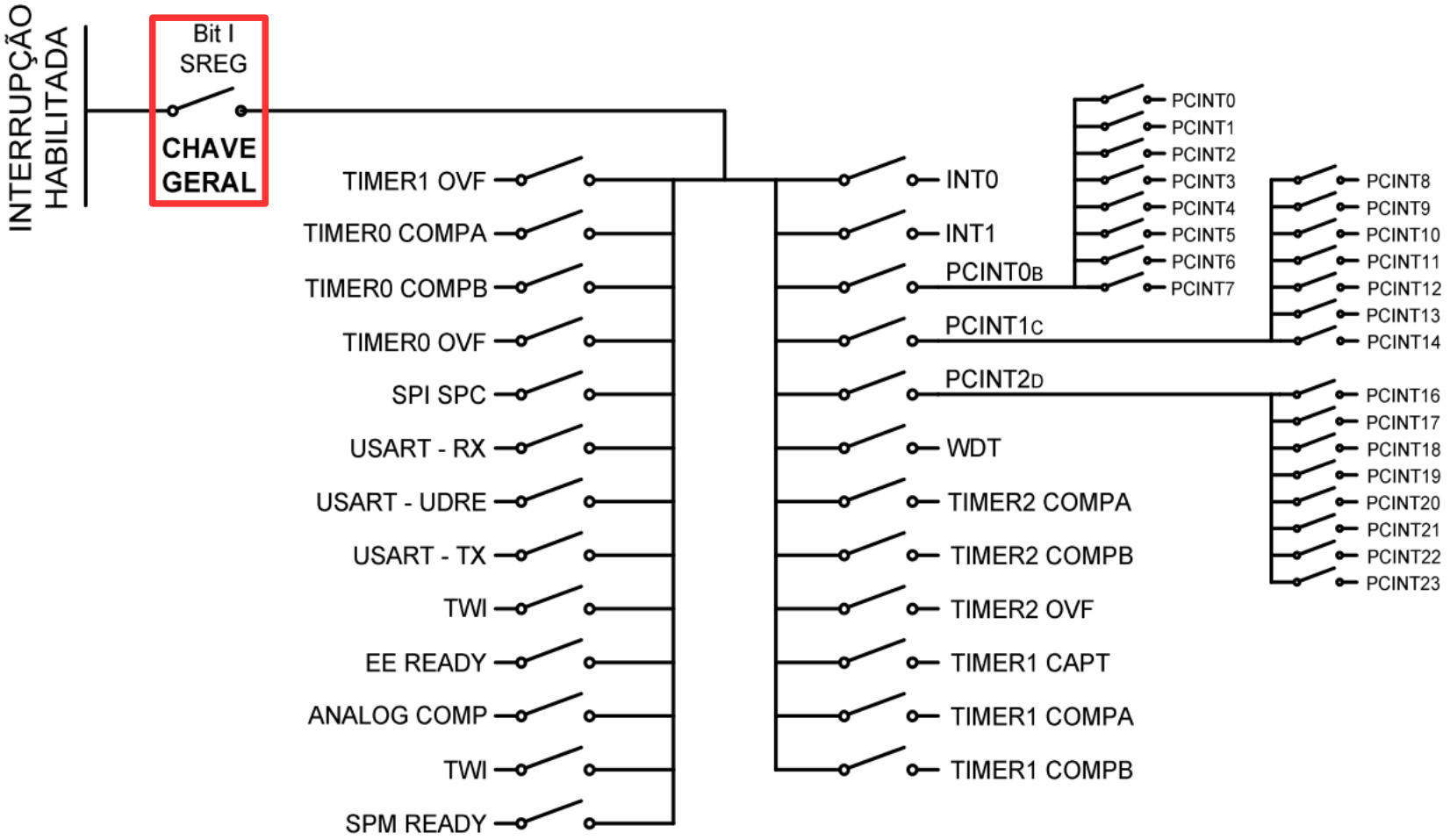


Fig. 6.1 – Chaves de habilitação das interrupções.

Interrupções

- **O AVR possui dois tipos de interrupção**
  - **O primeiro ativa um bit de sinalização (flag) indicando que a interrupção ocorreu, o qual é mantido em 1 até que a interrupção seja atendida**
    - Sendo zerado automaticamente pelo hardware
    - Alternativamente, o flag pode ser limpo pela escrita de 1 no bit correspondente
    - O bit de sinalização permite que várias interrupções fiquem ativas enquanto uma está sendo atendida e sejam processadas por ordem de prioridade

Interrupções

- **O AVR possui dois tipos de interrupção**
  - **O segundo tipo de interrupção é disparada quando o evento que a gera está presente**
    - Não existem bits de sinalização e a interrupção só é atendida se sua condição existir quando a chave geral das interrupções estiver ativa
    - Nesse caso, não há fila de espera, pois não há sinalização da ocorrência da interrupção
      - **Este é o caso das interrupção externas por nível, por exemplo**

Interrupções

- **Quando uma interrupção é executada o registrador de estado (SREG) não é automaticamente preservado nem restaurado ao término de sua execução**
  - **Em assembly:**
    - O registrador SREG e outros empregados no programa principal, e também em sub-rotinas (interrupção ou não)
      - » **Devem ter seus conteúdos salvos antes da execução da sub-rotina e restaurados ao término dessa**
      - » **Caso contrário, o programa pode apresentar respostas imprevisíveis, visto que os valores dos registradores podem ter sido alterados fora da sequência normal do programa**
    - Os comandos PUSH e POP são perfeitos para salvar e restaurar valores de registradores

Interrupções

.DEF reg_F= r16 // Registrador temporário para SREG

```
; -- Interrupt Service Routine --
PCINT0:
   push reg_F
   in reg_F,SREG
   push r16
   ; Load
   ; Do Something
   ; Store
   pop r16
   out SREG,reg_F
   pop reg_F
   reti
; ----------------------------------------------------
```

Interrupções

- Quando uma interrupção é executada o registrador de estado (SREG) não é automaticamente preservado nem restaurado ao término de sua execução
  - Em C:
    - O salvamento dos registradores é feito automaticamente pelo compilador de forma transparente ao programador

Interrupções

- **No compilador AVR-GCC**
  - **A tabela de vetores de interrupção é predefinida para apontar para rotinas de interrupção com nomes predeterminados**
    - Usando o nome apropriado, a rotina será chamada quando ocorrer a interrupção correspondente

Interrupções

Código (função)

```
int main()                {//aqui vai o programa principal                }
ISR(INT0_vect)            {//interrupção externa 0                        }
ISR(INT1_vect)            {//interrupção externa 1                        }
ISR(PCINT0_vect)          {//interrupção 0 por mudança de pino             }
ISR(PCINT1_vect)          {//interrupção 1 por mudança de pino             }
ISR(PCINT2_vect)          {//interrupção 2 por mudança de pino             }
ISR(WDT_vect)             {//estouro do temporizador Watchdog              }
ISR(TIMER2_COMPA_vect)    {//igualdade de comparação A do TC2              }
ISR(TIMER2_COMPB_vect)    {//igualdade de comparação B do TC2              }
ISR(TIMER2_OVF_vect)      {//estouro do TC2}                               }
ISR(TIMER1_CAPT_vect)     {//evento de captura do TC1                      }
ISR(TIMER1_COMPA_vect)    {//igualdade de comparação A do TC1              }
ISR(TIMER1_COMPB_vect)    {//igualdade de comparação B do TC1              }
ISR(TIMER1_OVF_vect)      {//estouro do TC1                                }
ISR(TIMER0_COMPA_vect)    {//igualdade de comparação A do TC0              }
ISR(TIMER0_COMPB_vect)    {//igualdade de comparação B do TC0              }
ISR(TIMER0_OVF_vect)      {//estouro do TC0                                }
ISR(SPI_STC_vect)         {//transferência serial completa - SPI           }
ISR(USART_RX_vect)        {//USART, recepção completa                      }
ISR(USART_UDRE_vect)      {//USART, limpeza do registrador de dados        }
ISR(USART_TX_vect)        {//USART, transmissão completa                   }
ISR(ADC_vect)             {//conversão do ADC completa                     }
ISR(EE_READY_vect)        {//EEPROM pronta                                 }
ISR(ANALOG_COMP_vect)     {//comparador analógico                          }
ISR(TWI_vect)             {//interface serial TWI                          }
ISR(SPM_READY_vect)       {//armazenagem na memória de programa pronta}
```

Interrupções

- **Cada interrupção é habilitada por um bit específico no seu registrador de controle**
  - A interrupção é efetivamente ativada quando o bit I do registrador SREG for colocado em 1

- **A chave geral é utilizada para ligar ou desligar todas as interrupções ativas de uma única vez**
  - No AVR-GCC:
    - » **A função sei( ) liga a chave geral das interrupções**
    - » **A função cli( ) a desliga**
  - Os bits individuais das interrupções serão vistos em momentos oportunos

Interrupções

- **Interrupções Externas**
  - **São empregadas para avisar o microcontrolador que algum evento externo a ele ocorreu**
    - Pode ser um botão pressionado ou um sinal de outro sistema digital
    - Muitas vezes, elas são empregadas para retirar a CPU de um estado de economia de energia
      - **Para 'despertar' o microcontrolador**

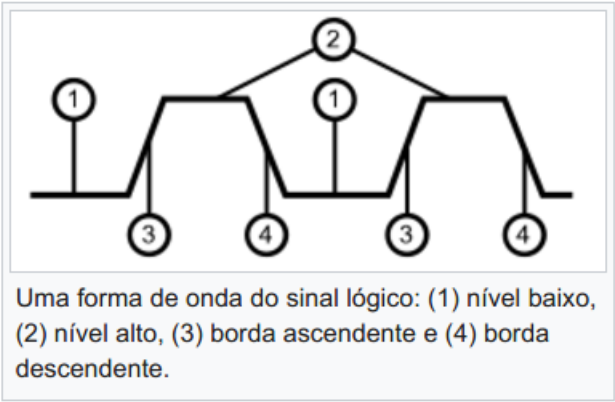
Interrupções

- **Interrupções Externas**
  - **Todos os pinos de I/O do ATmega328 podem gerar interrupções externas por:**
    - **Mudança de estado lógico no pino**
      - » **O nome delas nos pinos são PCINT0 até PCINT23**

  - Os dois pinos, **INT0** e **INT1**, que podem gerar interrupções:
    1) **Mudança de estado lógico no pino**
    2) **Na borda de subida (Rising edge)**
    3) **Na borada descida (Falling Edge)**
    4) **Na manutenção do nível do estado lógico no pino**

Interrupções

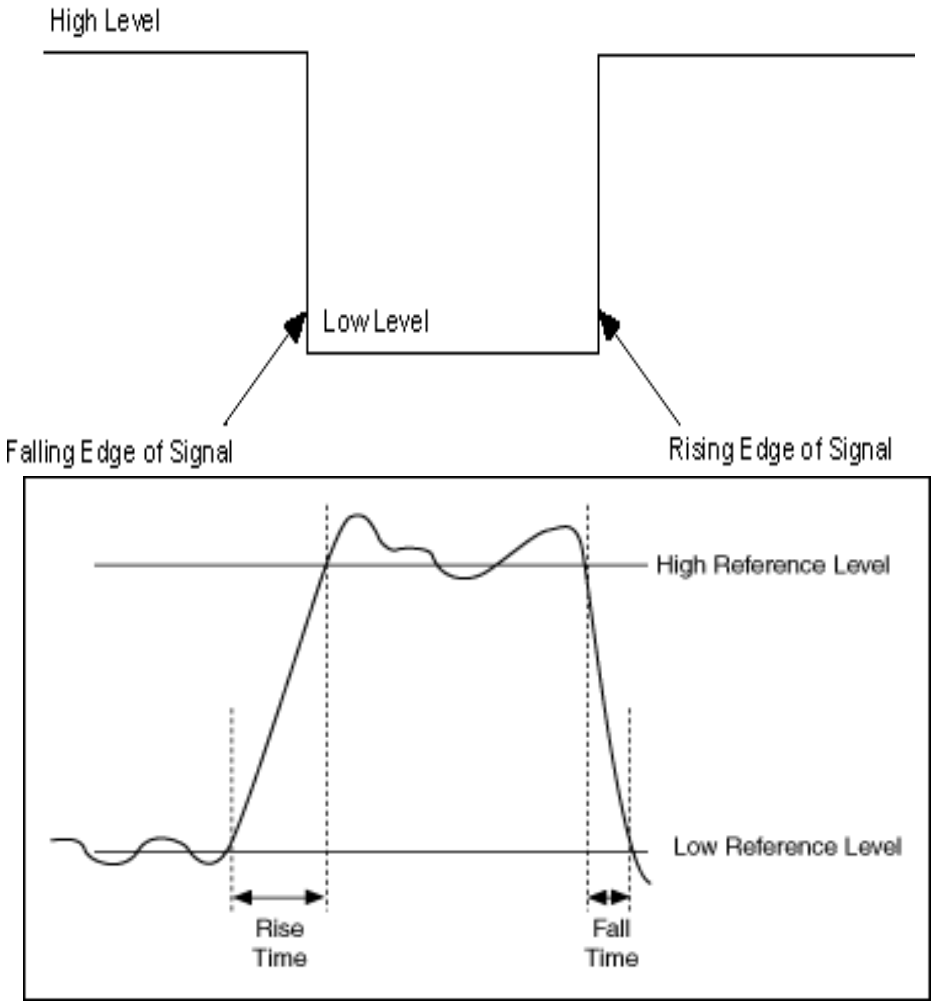
- **As alterações de lógica são acionadas pelo borda ascendente ou pela borda descendente**
  - **O diagrama dado é um exemplo do pulso prático e, portanto, introduzimos dois novos termos que são:**
    - **Borda ascendente/subida (Rising edge):** a transição de uma baixa tensão (nível 1 no diagrama) para uma alta tensão (nível 2).
    - **Borda descendente/descida (Falling Edge)**: a transição de uma alta tensão para uma baixa.



Uma forma de onda do sinal lógico: (1) nível baixo, (2) nível alto, (3) borda ascendente e (4) borda descendente.

https://en.wikipedia.org/wiki/Digital_signal

Interrupções



http://www.avrfreaks.net/forum/rising-edge-definition

Interrupções

- **O registrador de controle EICRA**
  - **Extern Interrupt Control Register A**
    - Contém os bits responsáveis pela configuração das interrupções externas INT0 e INT1

Bit		7	6	5	4	3	2	1	0
	EICRA	-	-	-	-	ISC11	ISC10	ISC01	ISC00
Lê/Escreve		L	L	L	L	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Interrupções

Tab. 6.2 – Bits de configuração da forma das interrupções nos pinos INT1 e INT0.

ISC11	ISC10	Descrição
0	0	Um nível baixo em INT1 gera um pedido de interrupção.
0	1	Qualquer mudança lógica em INT1 gera um pedido de interrupção.
1	0	Uma borda de decida em INT1 gera um pedido de interrupção.
1	1	Uma borda de subida em INT1 gera um pedido de interrupção.
ISC01	**ISC00**	**Descrição**
0	0	Um nível baixo em INT0 gera um pedido de interrupção.
0	1	Qualquer mudança lógica em INT0 gera um pedido de interrupção.
1	0	Uma borda de decida em INT0 gera um pedido de interrupção.
1	1	Uma borda de subida em INT0 gera um pedido de interrupção.

Interrupões

- **As interrupções são habilitadas nos bits INT1 e INT0 do registrador EIMSK**
  - **External Interrupt Mask Register**
    - Se o bit I do registrador SREG (Status Register) também estiver habilitado

Bit		7	6	5	4	3	2	1	0
	EIMSK	-	-	-	-	-	-	INT1	INT0
Lê/Escreve		L	L	L	L	L	L	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Interrupões

- **O registrador EIFR**
  - **External Interrupt Flag Register**
    - Contém os dois bits sinalizadores que indicam se alguma interrupção externa ocorreu
      - » **São limpos automaticamente pela CPU após a interrupção ser atendida**
      - » **Alternativamente, esses bits podem ser limpos pela escrita de 1 lógico**

Bit		7	6	5	4	3	2	1	0
	EIFR	-	-	-	-	-	-	INTF1	INTF0
Lê/Escreve		L	L	L	L	L	L	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Interrupções

- **O registrador PCICR**
  - **Pin Change Interrupt Control Register**
    - É responsável pela habilitação de todas as interrupções externas nos pino de I/O
      - » **Do PCINT0 até o PCINT23, as quais são divididas entre os três PORTs do microcontrolador:**

Bit		7	6	5	4	3	2	1	0
	PCICR	-	-	-	-	-	PCIE2	PCIE1	PCIE0
Lê/Escreve		L	L	L	L	L	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

- **PCIE0 habilita a interrupção por qualquer mudança nos pinos do PORTB (PCINT0:7)**
- **PCIE1 nos pinos do PORTC (PCINT8:14)**
- **PCIE2 nos pinos do PORTD (PCINT16:23)**

Interrupções

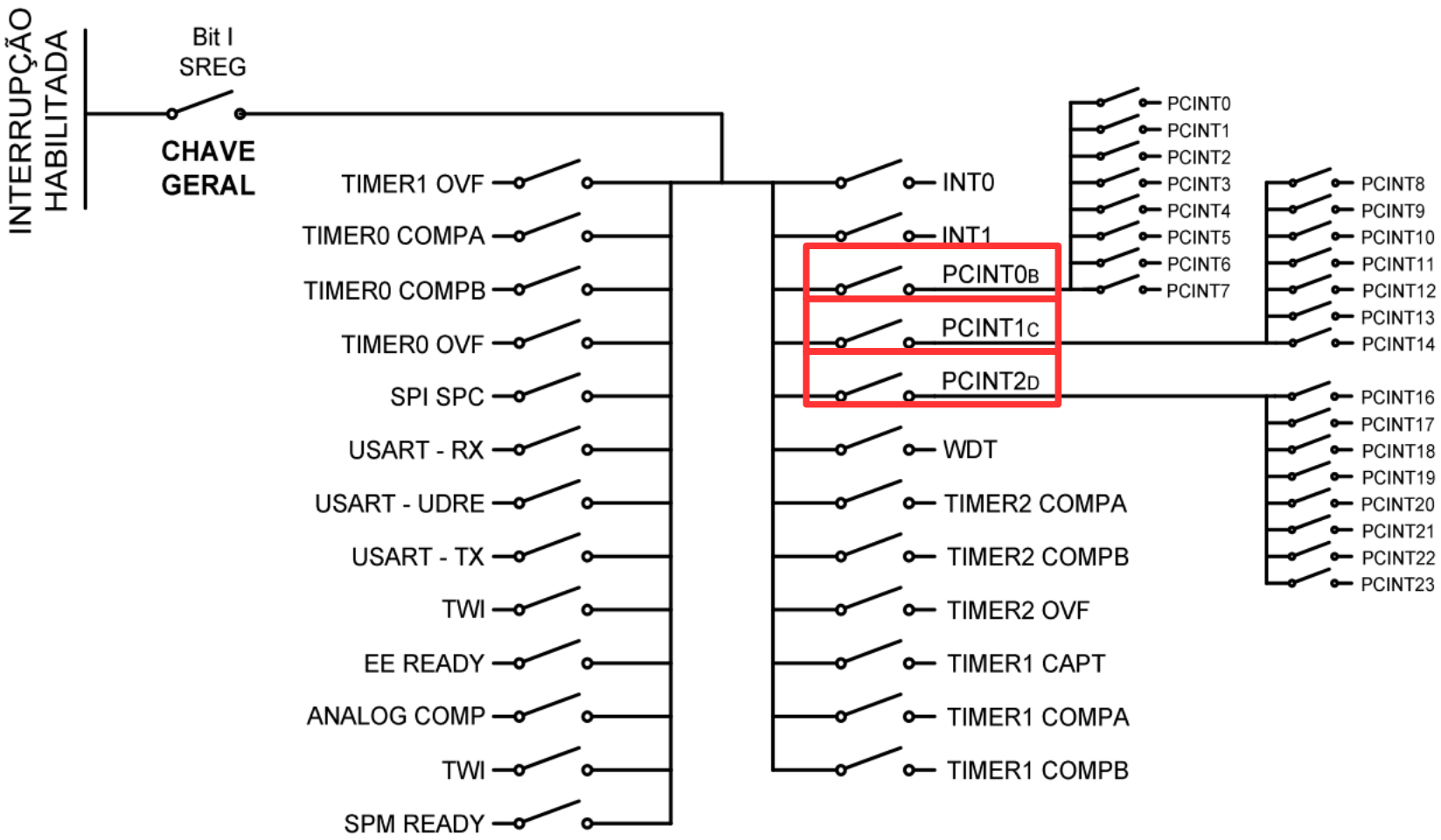


Fig. 6.1 – Chaves de habilitação das interrupções.

Interrupções

- **O registrador PCIFR**
  - **Pin Change Interrupt Flag Register**
    - Quando ocorrer um interrupção em algum dos pinos
      - » **O bit de sinalização do PORT no registrador PCIFR é ativo:**

Bit	7	6	5	4	3	2	1	0
PCIFR	-	-	-	-	-	PCIF2	PCIF1	PCIF0
Lê/Escreve	L	L	L	L	L	L/E	L/E	L/E
Valor Inicial	0	0	0	0	0	0	0	0

- **PCIF0 para o PORTB**
- **PCIF1 para o PORTC**
- **PCIF2 para o PORTD**

Interrupções

- **Existem várias chaves ligadas às chaves de interrupção dos PORTs:**
  - PCINT0B
    - **PCINT0**
    - **PCINT1**
    - **PCINT2**
    - **PCINT3**
    - **PCINT4**
    - **PCINT5**
    - **PCINT6**
    - **PCINT7**

Interrupções

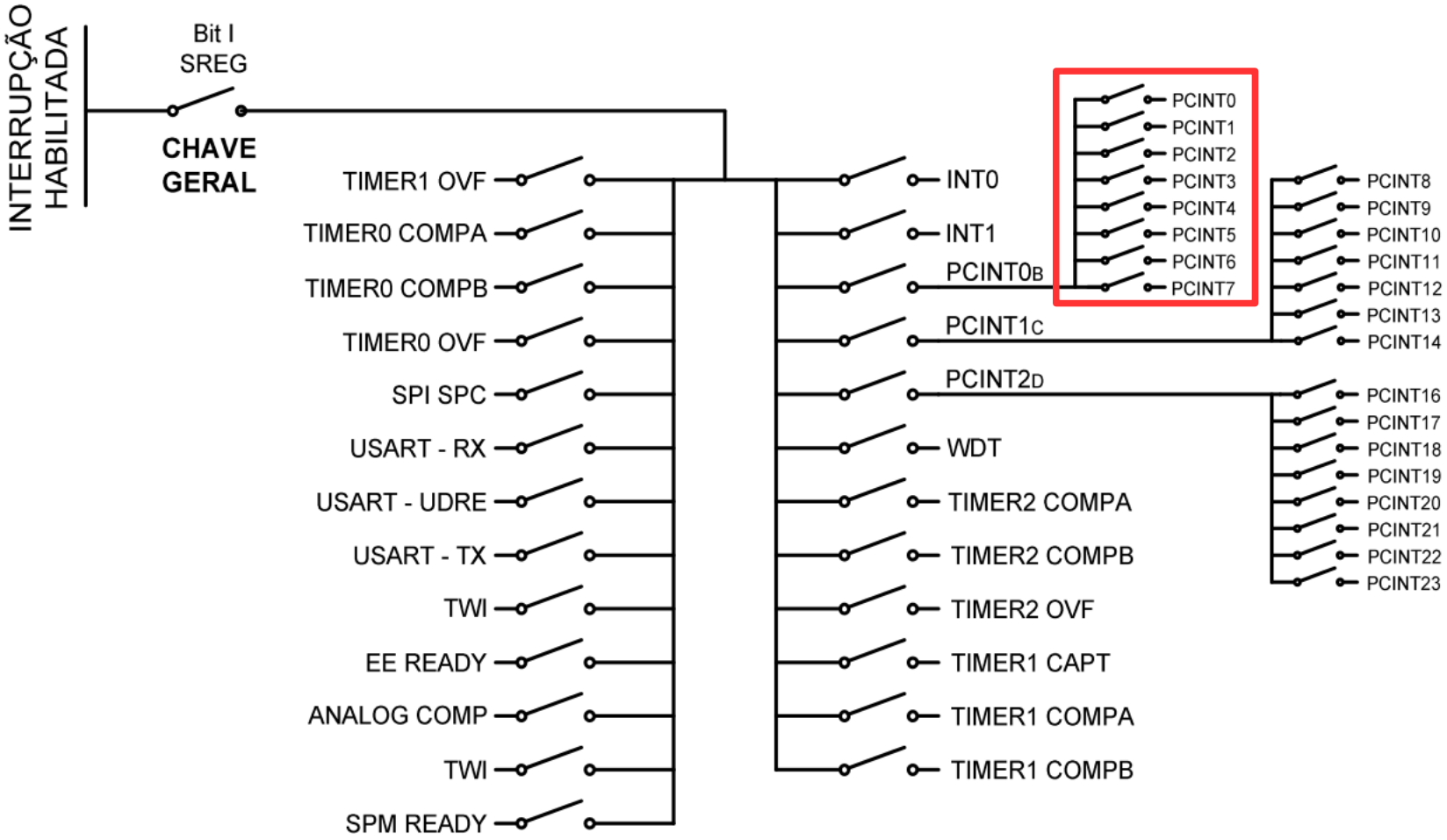


Fig. 6.1 – Chaves de habilitação das interrupções.

Interrupções

- **PCINT0...PCINT7 estão ligados ao PORTB**
  - **A habilitação individual dos pinos é feita nos registradores PCMSK0**

Bit		7	6	5	4	3	2	1	0
	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

- **A habilitação individual dos pinos do PORTB é feita nos registradores PCMSK0**

Interrupções

- **Existem várias chaves ligadas às chaves de interrupção dos PORTs:**
  - PCINT0c
    - **PCINT8**
    - **PCINT9**
    - **PCINT10**
    - **PCINT11**
    - **PCINT12**
    - **PCINT13**
    - **PCINT14**

Interrupções

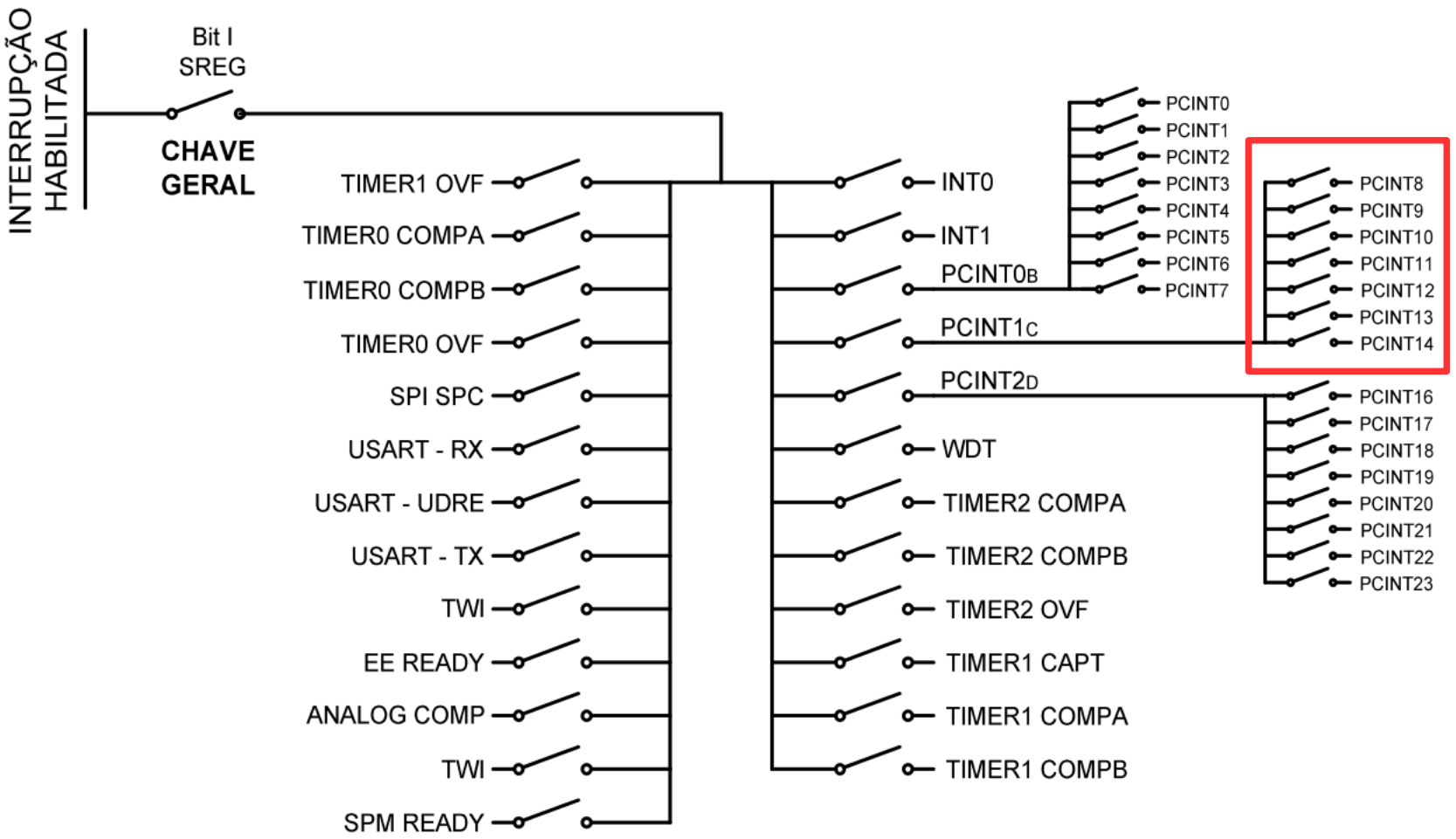


Fig. 6.1 – Chaves de habilitação das interrupções.

Interrupções

- **PCINT8...PCINT14 estão ligados ao PORTC**
  - **A habilitação individual dos pinos é feita nos registradores PCMSK1**

Bit		7	6	5	4	3	2	1	0
	PCMSK1	-	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Interrupções

- **Existem várias chaves ligadas às chaves de interrupção dos PORTs:**
  - PCINT0c
    - **PCINT16**
    - **PCINT17**
    - **PCINT18**
    - **PCINT19**
    - **PCINT20**
    - **PCINT21**
    - **PCINT22**
    - **PCINT23**

Interrupções

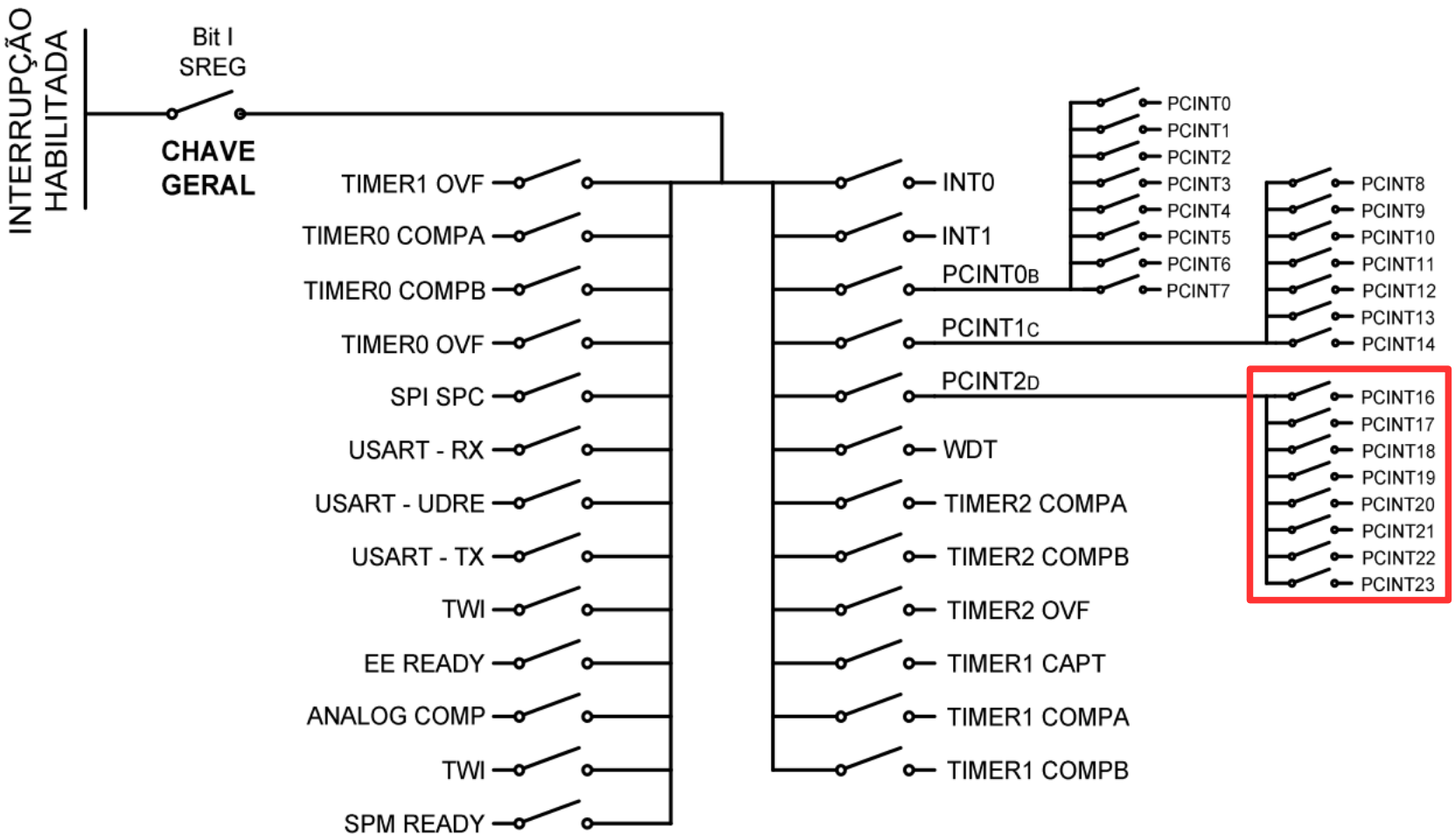


Fig. 6.1 – Chaves de habilitação das interrupções.

Interrupções

- **PCINT16...PCINT23 estão ligados ao PORTD**
  - **A habilitação individual dos pinos é feita nos registradores PCMSK2**

Bit		7	6	5	4	3	2	1	0
	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Interrupções

- **Para ilustrar a operação das duas interrupções externas**
  - **INT0 e INT1 configuradas, respectivamente, para:**
    - Gerar interrupção por nível e por transição

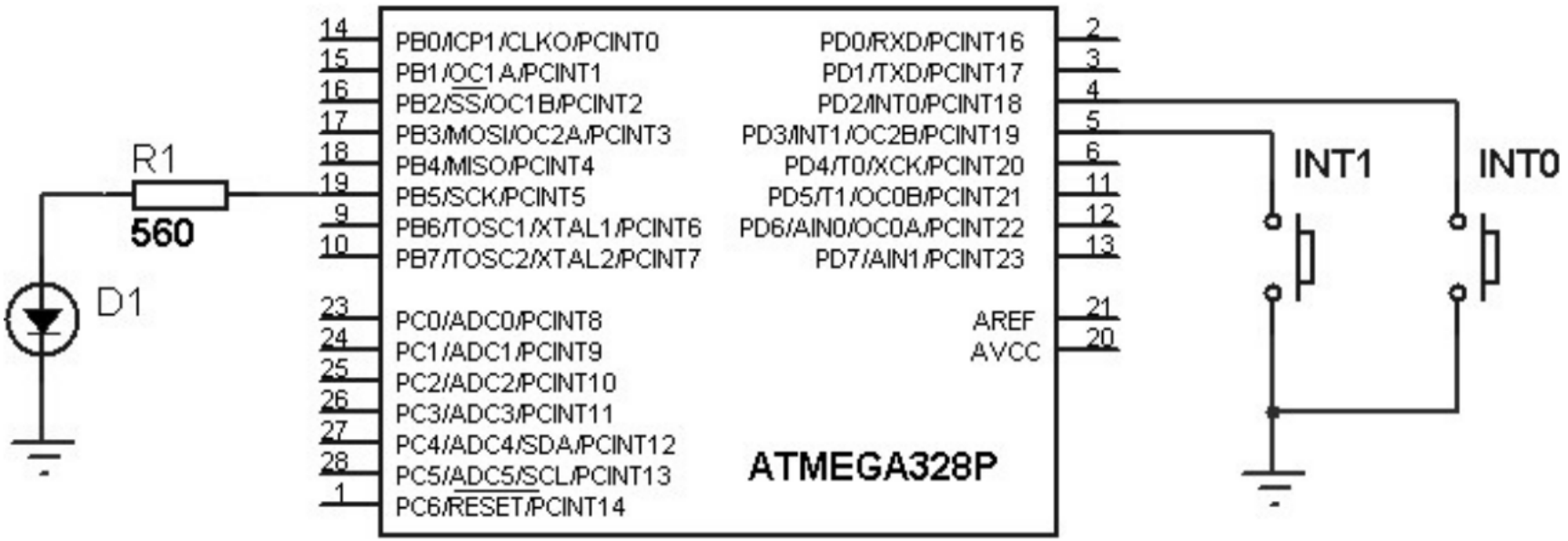


Fig. 6.2 – Circuito para emprego das interrupções externas INT0 e INT1.

Interrupções

- **Para ilustrar a operação das duas interrupções externas**
  - **INT0 e INT1 configuradas, respectivamente, para:**
    - Um botão irá trocar o estado do LED (se ligado, desliga e vice-versa)
    - Outro botão mantido pressionado piscará o LED

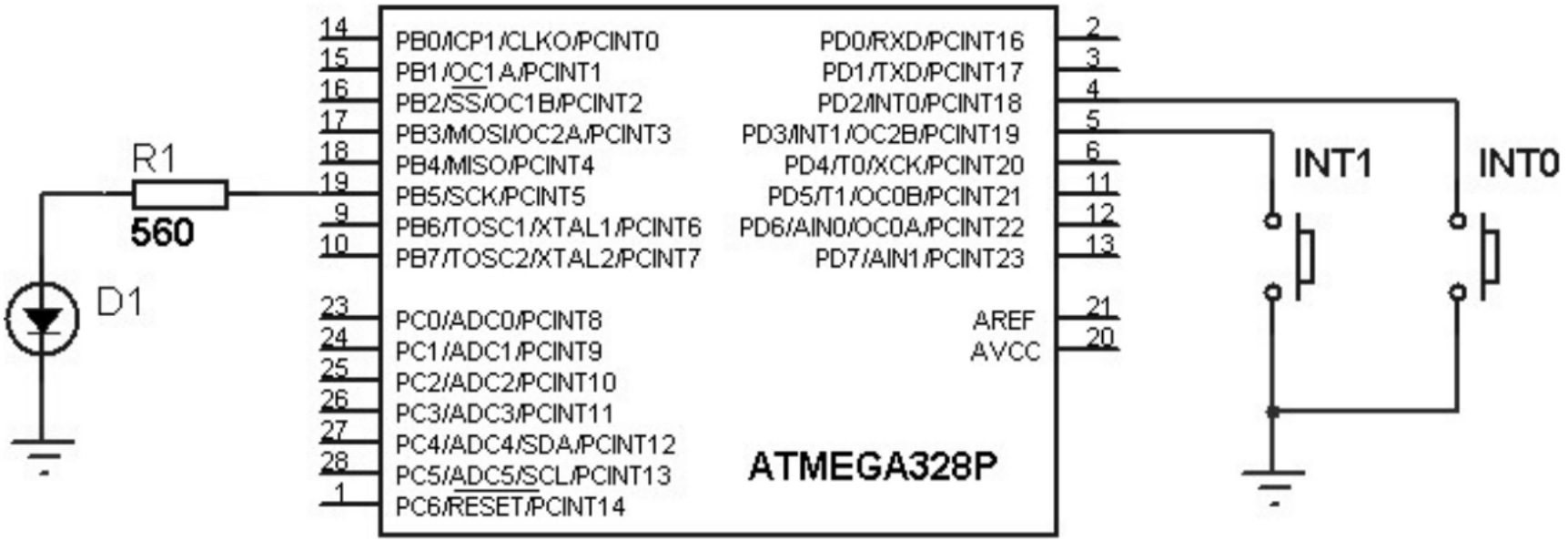


Fig. 6.2 – Circuito para emprego das interrupções externas INT0 e INT1.

INTO_1.c

```
//=========================================================================== //
// HABILITANDO AS INTERRUPÇÕES INT0 e INT1 POR TRANSIÇÃO E NÍVEL, RESPECTIVAMENTE //
//=========================================================================== //
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

//Definições de macros - empregadas para o trabalho com bits
#define   set_bit(Y,bit_x) (Y|=(1<<bit_x))    //ativa o bit x da variável
#define   clr_bit(Y,bit_x) (Y&=~(1<<bit_x))   //limpa o bit x da variável
#define   tst_bit(Y,bit_x) (Y&(1<<bit_x))     //testa o bit x da variável
#define   cpl_bit(Y,bit_x) (Y^=(1<<bit_x))    //troca o estado do bit x

#define LED PB5                               //LED está no pino PB5
```

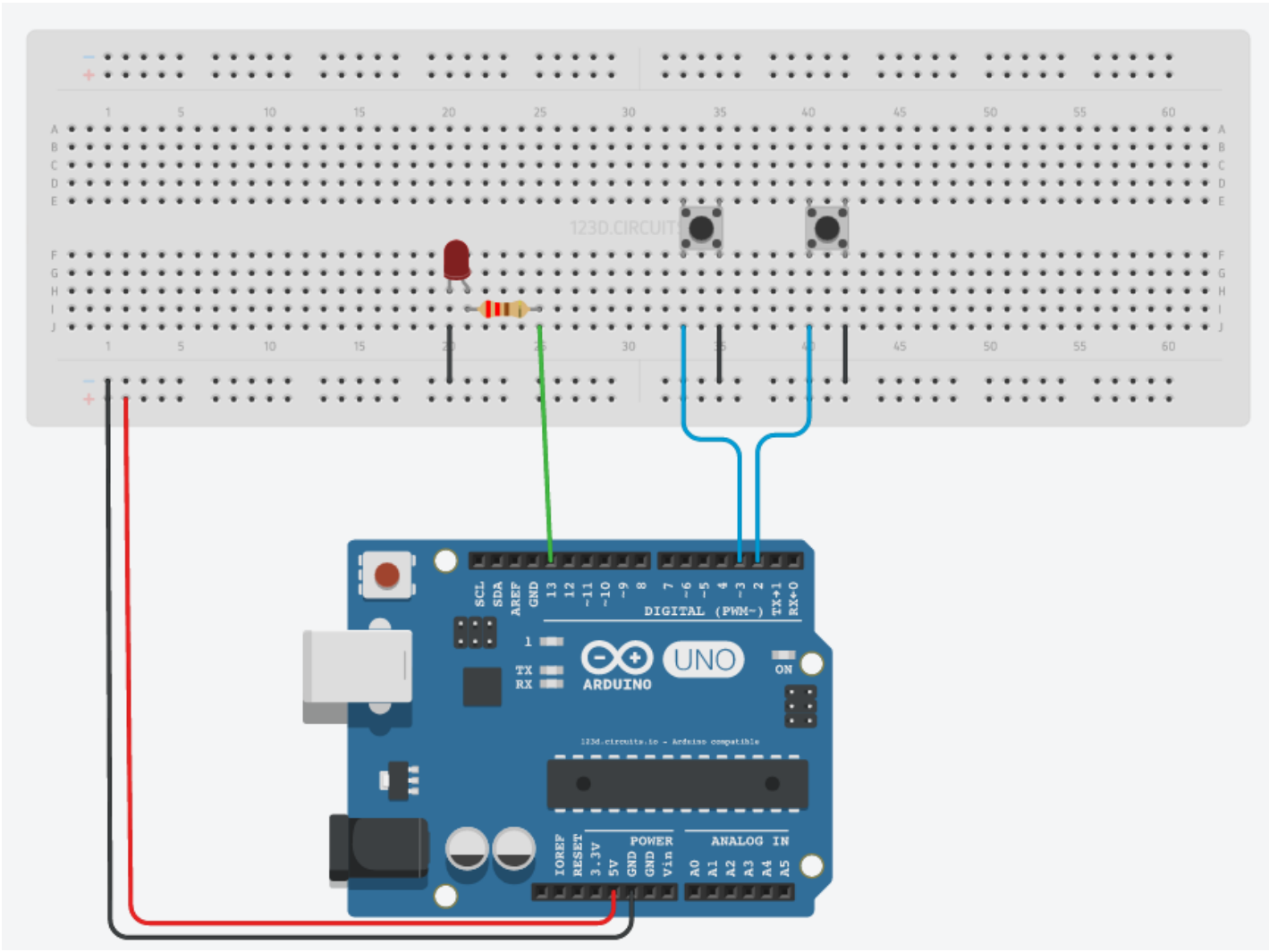
```
// mudança necessária para funcionar no simulador Tinkercad
// interrupções externas: INT0 na borda de descida, INT1 na borda de subida
EICRA = (1<<ISC11) | (1<<ISC10) | (1<<ISC01);
```

```
ISR(INT0_vect);
ISR(INT1_vect);
//-------------------------------------------------------------------
int main()
{
      DDRD  = 0x00;         //PORTD entrada
      PORTD = 0xFF;         //pull-ups habilitados
      DDRB  = 0b00100000;   //somente pino do LED como saída
      PORTB = 0b11011111;   //desliga LED e habilita pull-ups

      UCSR0B = 0x00;        /*necessário desabilitar RX e TX para trabalho com os pinos
                                                          do PORTD no Arduino*/
      EICRA = 1<<ISC01;//interrupções externas: INT0 na borda de descida, INT1 no nível zero.
      EIMSK = (1<<INT1) | (1<<INT0);//habilita as duas interrupções
      sei();          //habilita interrupções globais, ativando o bit I do SREG

      while(1){}
}
//-------------------------------------------------------------------------------
ISR(INT0_vect)  //interrupção externa 0, quando o botão é pressionado o LED troca de estado
{
      cpl_bit(PORTB,LED);
}
//-------------------------------------------------------------------------------
ISR(INT1_vect)  //interrupção externa 1, mantendo o botão pressionado o LED pisca
{
      cpl_bit(PORTB,LED);
      _delay_ms(200);       //tempo para piscar o LED
}
//===============================================================================
```

https://www.tinkercad.com/things/3F3FHaAfgsV



Interrupções

- **O uso das interrupções externas por mudança em qualquer pino de I/O**
  - **Quando um botão do PORTC é pressionado, o LED definido para o pino, liga ou desliga**
    - As interrupções são por mudança de estado no pino
    - É gerado um pedido de interrupção sempre que se pressiona e se solta o botão

Interrupções

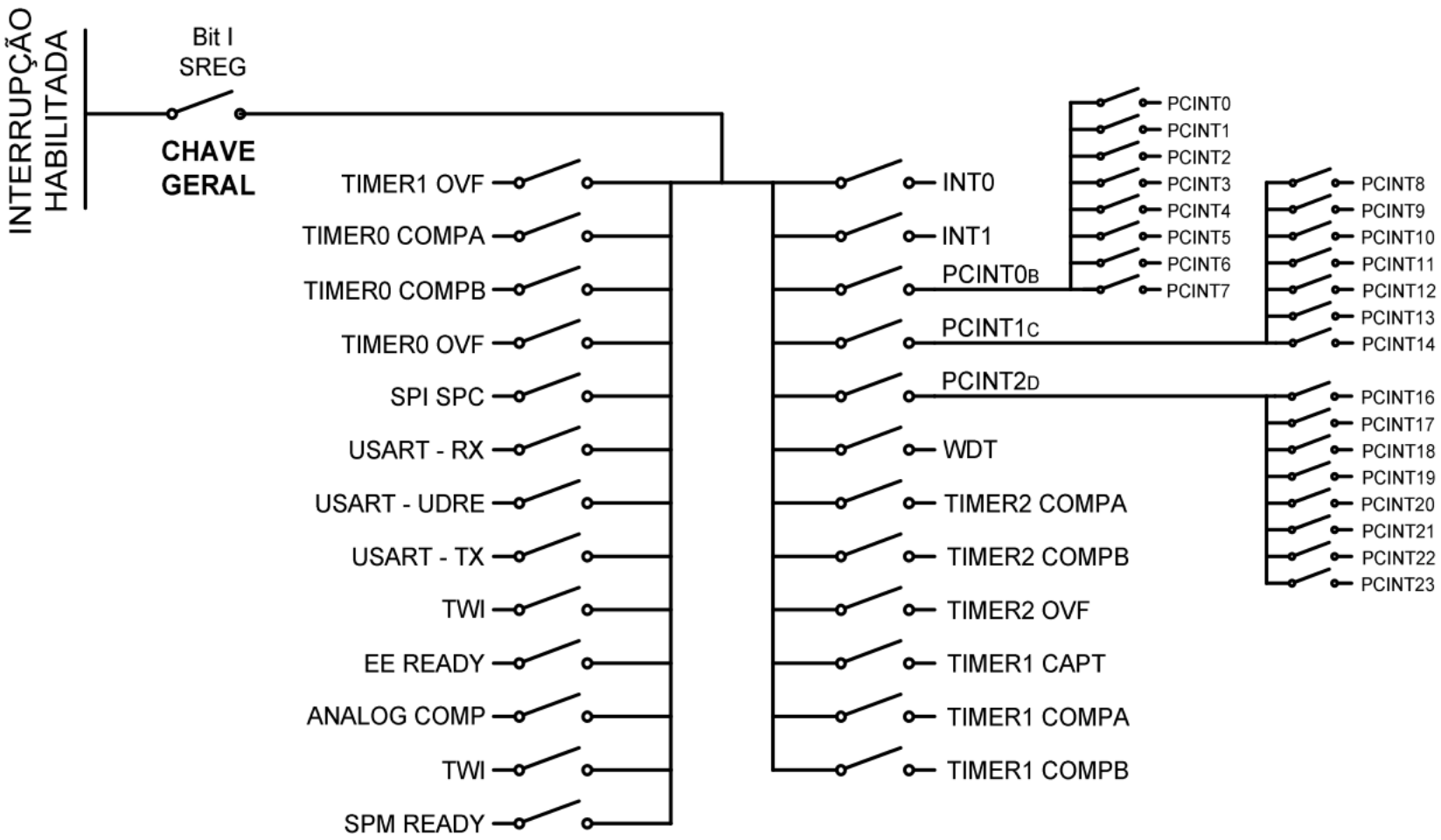


Fig. 6.1 – Chaves de habilitação das interrupções.

Interrupções

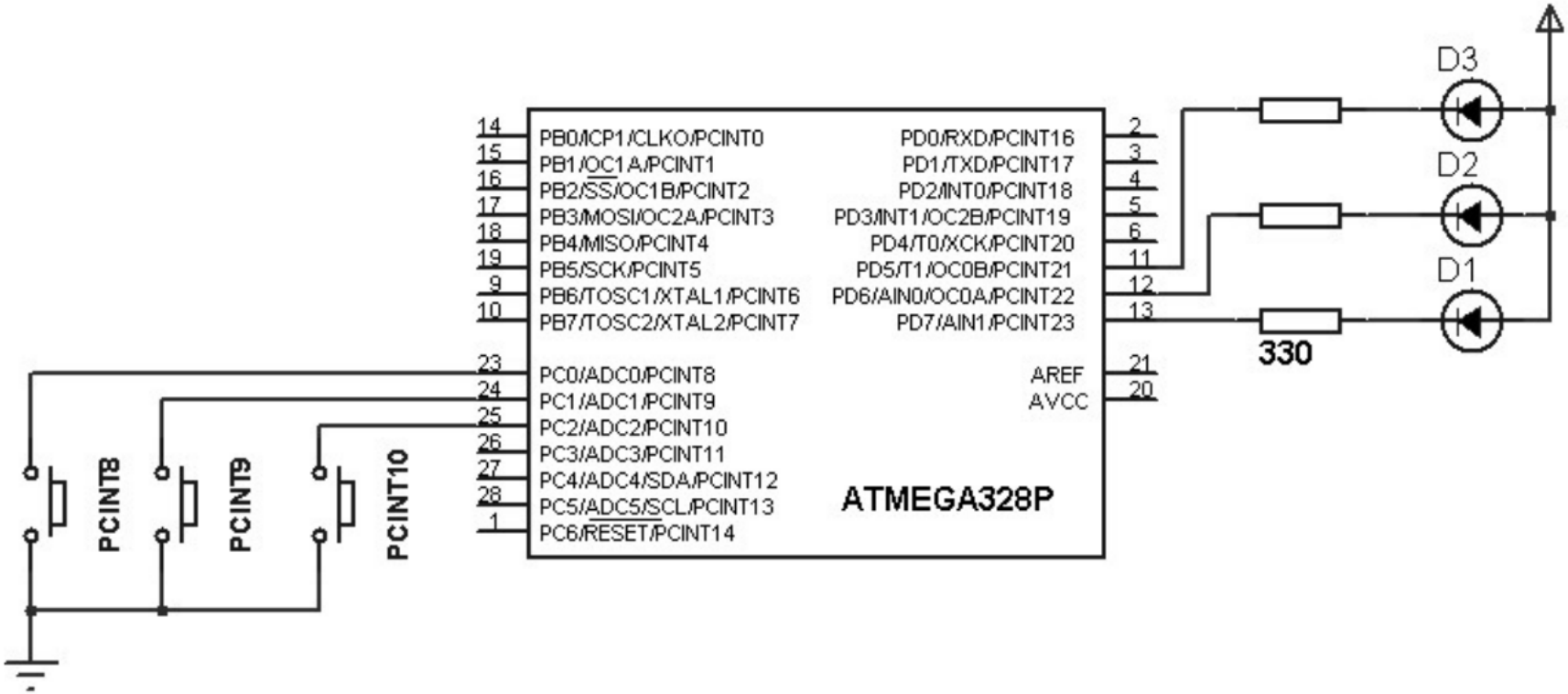
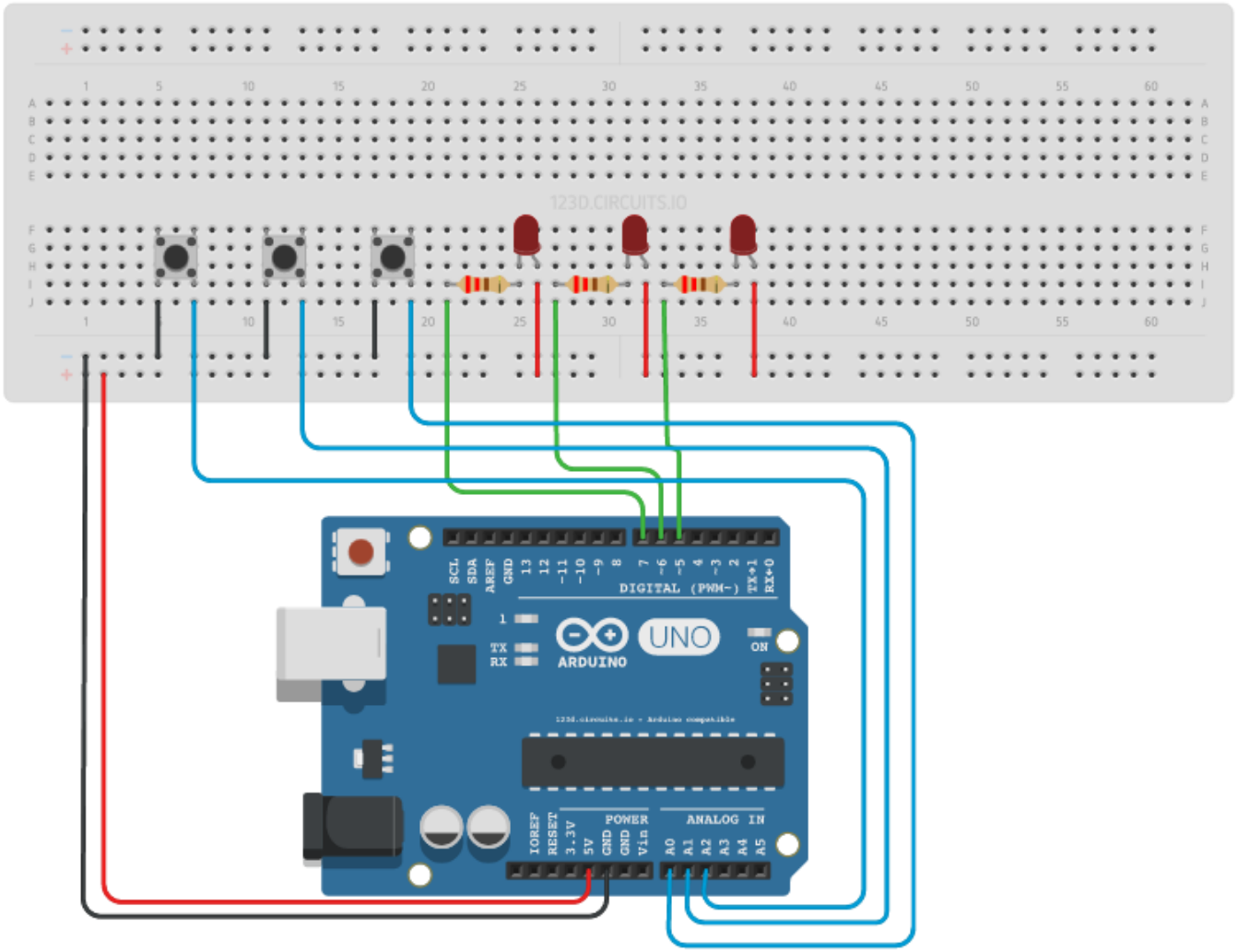


Fig. 6.3 – Circuito para teste das interrupções externas PCINT8:10.

https://www.tinkercad.com/things/ceSSVQe7JBL



Interrupções

- **O uso das interrupções externas por mudança em qualquer pino de I/O**
  - **Quando um botão do PORTC é pressionado, o LED definido para o pino, liga ou desliga**
    - Existindo ruído no acionamento do botão, o resultado será imprevisível
      - » **Colocado um atraso de 200 ms no final da rotina de interrupção**
      - » **Para que o botão seja pressionado rapidamente, ativando a interrupção somente na borda de descida do sinal**
        - » Caso contrário, o estado do LED seria aleatório
    - Como a interrupção externa PCINT1 só possui um endereço na memória de programa
      - » **É necessário testar qual foi o pino responsável pela interrupção**

def_principais.h (arquivo de cabeçalho do programa principal)

```
#ifndef _DEF_PRINCIPAIS_H
#define _DEF_PRINCIPAIS_H

#define F_CPU 16000000UL       //frequência de trabalho
#include <avr/io.h>            //definições do componente especificado
#include <avr/interrupt.h>     //define algumas  macros para as interrupções
#include <util/delay.h>        //biblioteca para o uso das rotinas de delay

//Definições de macros para o trabalho com bits
#define   set_bit(y,bit)  (y|=(1<<bit))//coloca em 1 o bit x da variável Y
#define   clr_bit(y,bit)  (y&=~(1<<bit))//coloca em 0 o bit x da variável Y
#define   cpl_bit(y,bit)  (y^=(1<<bit))//troca o estado lógico do bit x da variável Y
#define   tst_bit(y,bit)  (y&(1<<bit))//retorna 0 ou 1 conforme leitura do bit

#define   LED0 PD5
#define   LED1 PD6
#define   LED2 PD7

#endif
```

PCINTx.c (programa principal)

```
//=============================================================================  //
//   Cada vez que um botão é pressionado o LED correspondente troca de estado     //
//============================================================================= //
#include "def_principais.h"//inclui arquivo com as definições principais

ISR(PCINT1_vect);

int main()
{
     DDRC =  0x00;        //PORTC como entrada, 3 botões
     PORTC = 0xFF;        //habilita pull-ups
     DDRD  = 0b11100000;  //pinos PD5:7 do PORTC como saída (leds)
     PORTD = 0xFF;        //apaga leds e habilita pull-ups dos pinos não utilizados

     PCICR = 1<<PCIE1;//habilita interrupção por qualquer mudança de sinal no PORTC
     PCMSK1 = (1<<PCINT10)|(1<<PCINT9)|(1<<PCINT8);/*habilita os pinos PCINT8:10 para
                                                     gerar interrupção*/
     sei();               //habilita as interrupções

     while(1){}
}
//-----------------------------------------------------------------------------
ISR(PCINT1_vect)
{
     //quando houver mais de um pino que possa gerar a interrupção é necessário testar qual foi
     if(!tst_bit(PINC,PC0))
          cpl_bit(PORTD,LED0);
     else if(!tst_bit(PINC,PC1))
          cpl_bit(PORTD,LED1);
     else if(!tst_bit(PINC,PC2))
          cpl_bit(PORTD,LED2);

     _delay_ms(200);
}
//=============================================================================
```

Interrupções

- **O registrador PCICR**
  - **Pin Change Interrupt Control Register**
    - É responsável pela habilitação de todas as interrupções externas nos pino de I/O
      - **» Do PCINT0 até o PCINT23, as quais são divididas entre os três PORTs do microcontrolador:**

Bit		7	6	5	4	3	2	1	0
	PCICR	-	-	-	-	-	PCIE2	PCIE1	PCIE0
Lê/Escreve		L	L	L	L	L	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

- **PCIE0 habilita a interrupção por qualquer mudança nos pinos do PORTB (PCINT0:7)**
- **PCIE1 nos pinos do PORTC (PCINT8:14)**
- **PCIE2 nos pinos do PORTD (PCINT16:23)**

Interrupções

- **PCINT8...PCINT14 estão ligados ao PORTC**
  - **A habilitação individual dos pinos é feita nos registradores PCMSK1**

Bit		7	6	5	4	3	2	1	0
	PCMSK1	-	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Interrupções

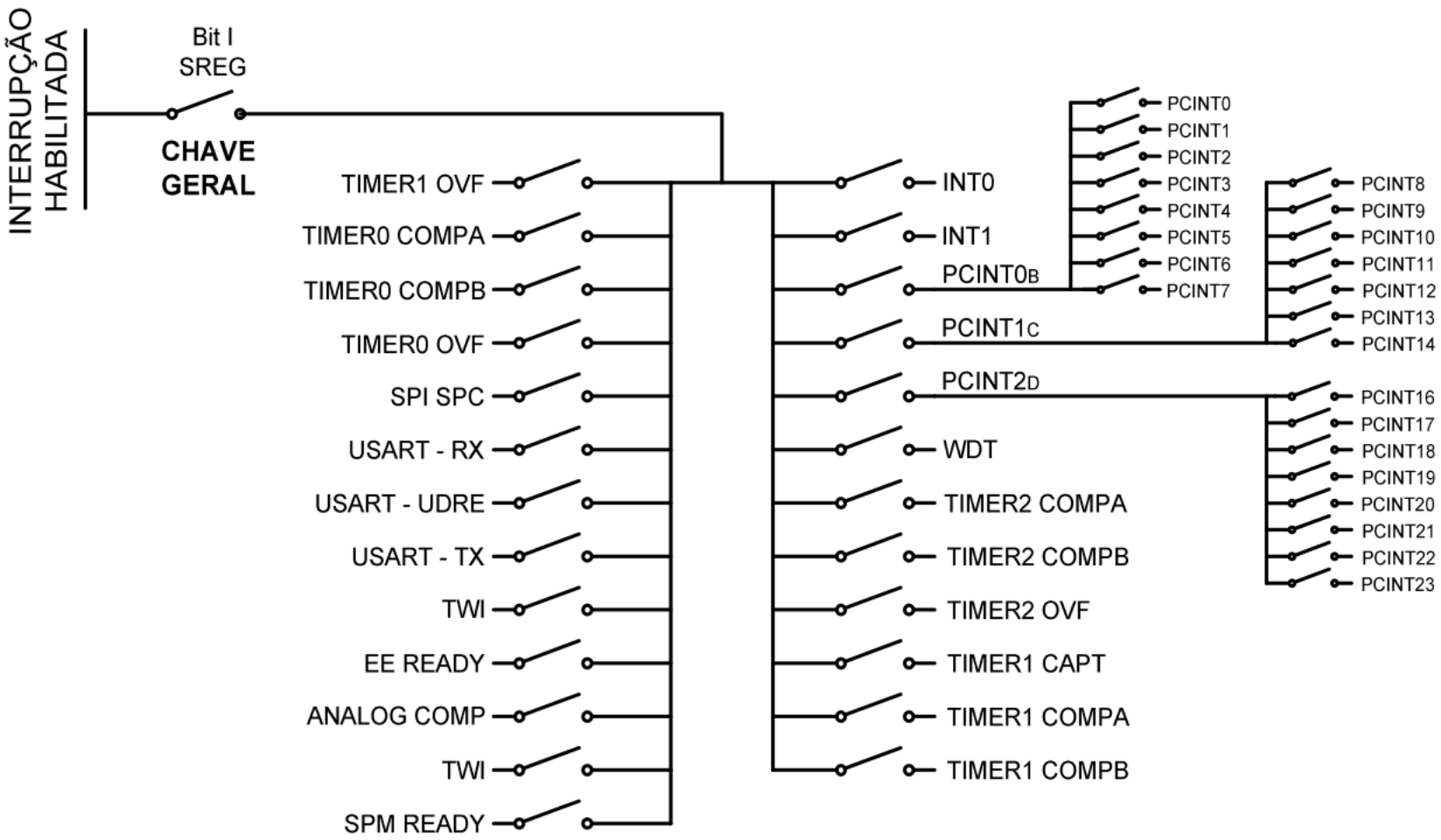


Fig. 6.1 – Chaves de habilitação das interrupções.

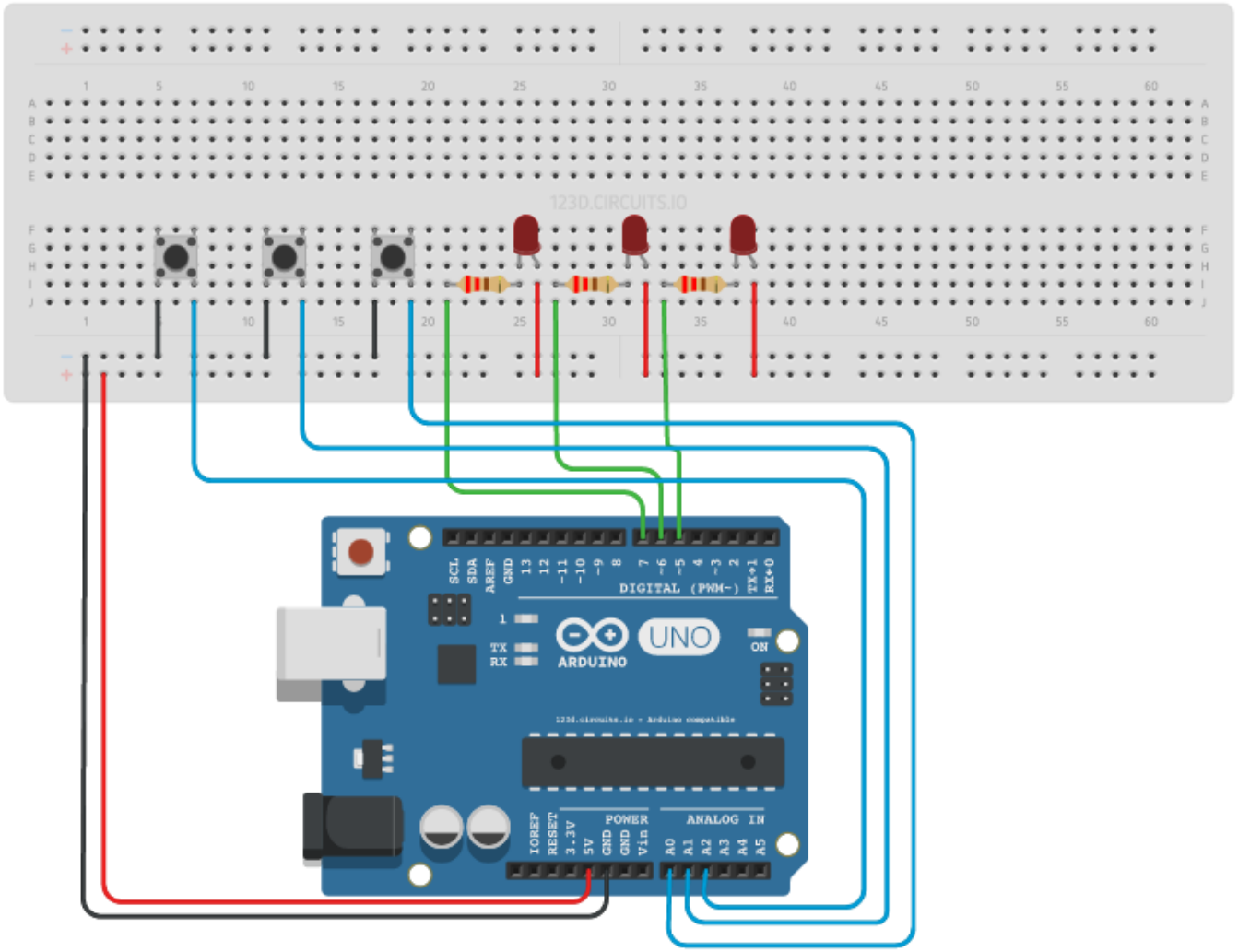
Interrupões

- **O registrador PCICR**
  - **Pin Change Interrupt Control Register**
    - É responsável pela habilitação de todas as interrupções externas nos pino de I/O
      - » **Do PCINT0 até o PCINT23, as quais são divididas entre os três PORTs do microcontrolador:**

Bit		7	6	5	4	3	2	1	0
	PCICR	-	-	-	-	-	PCIE2	PCIE1	PCIE0
Lê/Escreve		L	L	L	L	L	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

- **PCIE0 habilita a interrupção por qualquer mudança nos pinos do PORTB (PCINT0:7)**
- **PCIE1 nos pinos do PORTC (PCINT8:14)**
- **PCIE2 nos pinos do PORTD (PCINT16:23)**

https://www.tinkercad.com/things/ceSSVQe7JBL



Interrupções

Elaborar um programa em C para apresentar em um display de 7 segmentos, um número aleatório[1] entre 1 e F quando um botão for pressionado, ou seja, crie um dado eletrônico (hexa).

a) Empregue o circuito da Figura 1 e utilize o código Display_7Seg.c como referência

b) Utilizar a interrupção externa quando o botão for pressionado

Obs.: [1]Na verdade, criar um número puramente aleatório é difícil, o mais fácil é um pseudoaleatório. Neste exercício, o objetivo é não empregar as bibliotecas padrão do C. A ideia é utilizar o botão para gerar o evento de sorteio do número. Dessa forma, um contador pode ficar contando continuamente de 1 até F e, quando o botão for pressionado, o número da contagem será selecionado.

Interrupções

Elaborar um programa em C para apresentar em um display de 7 segmentos, um número aleatório[1] entre 1 e F quando um botão for pressionado, ou seja, crie um dado eletrônico (hexa).

c) Utilizar um flag no código:

volatile unsigned char flag=0;

https://en.wikipedia.org/wiki/Volatile_(computer_programming)

Interrupções

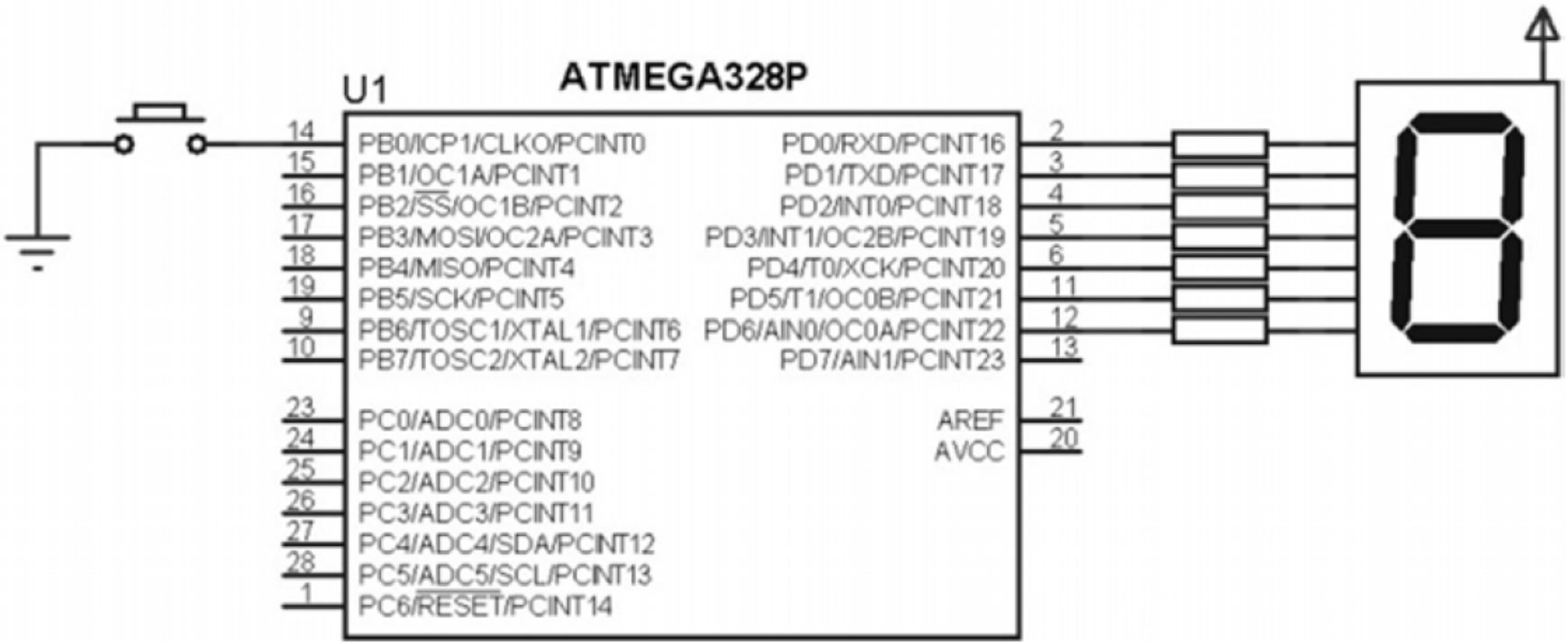
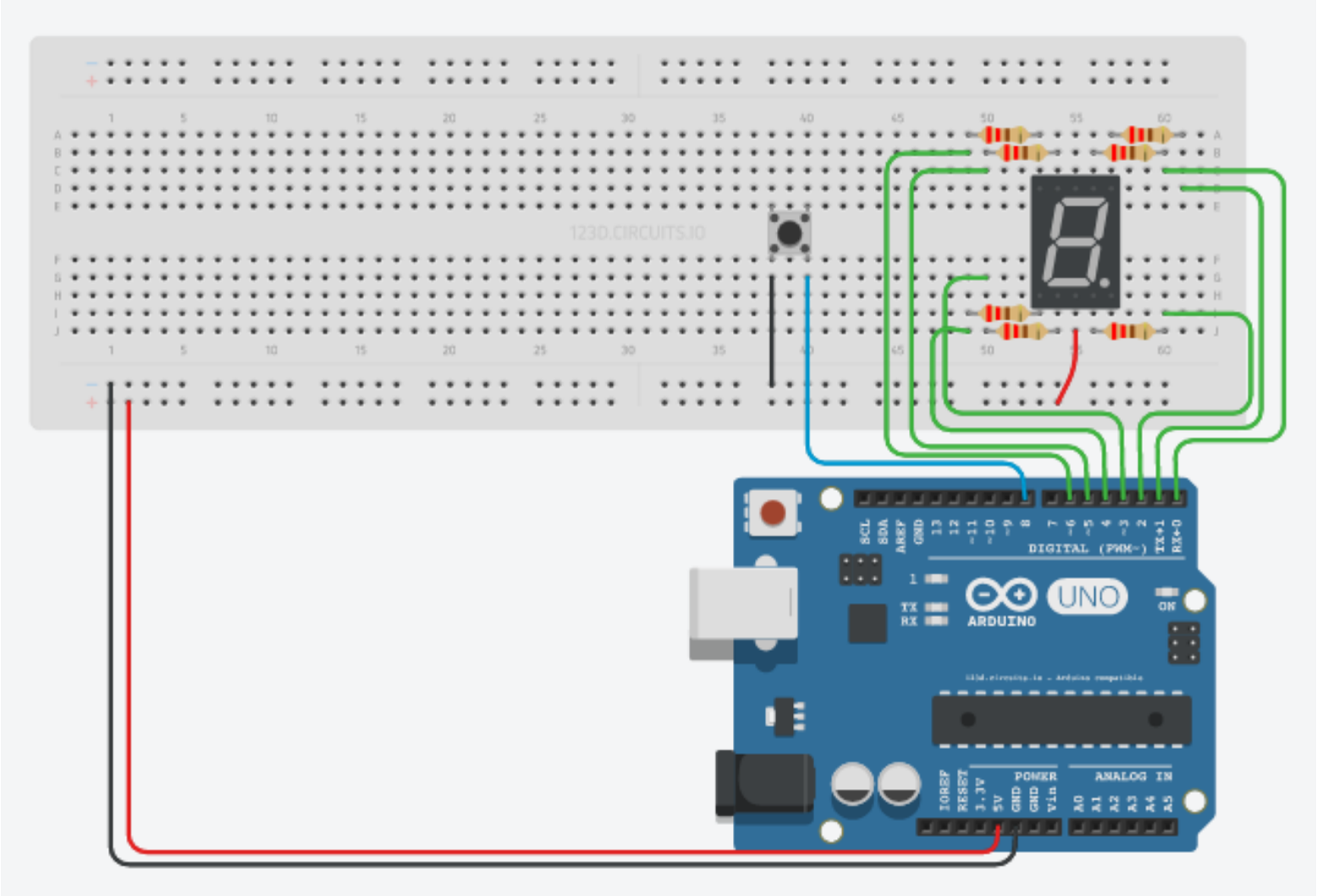


Figura 1: Circuito para acionamento de um display de 7 segmentos anodo comun.

https://www.tinkercad.com/things/fP3uuy44GmX



Interrupções – Arduino

- **Exemplos:**
  - Digital Pins With Interrupts - Arduino
    - » **https://www.arduino.cc/en/Reference/AttachInterrupt**
    - » **https://circuits.io/circuits/2748554-arduino_interrupt**
    - » **http://www.gammon.com.au/interrupts**