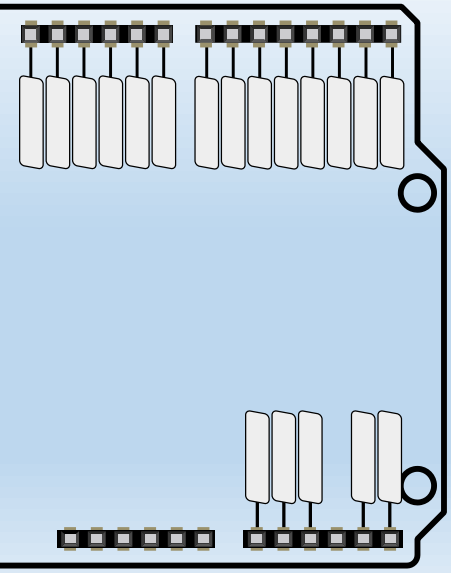
# Entradas digitais

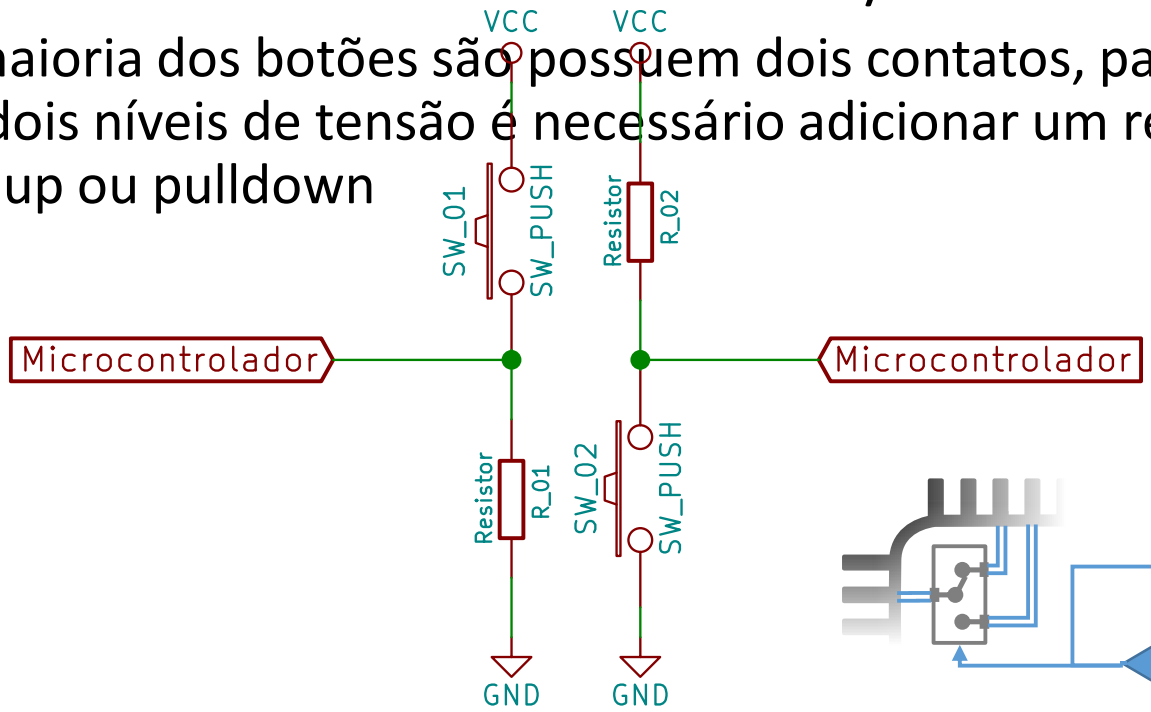


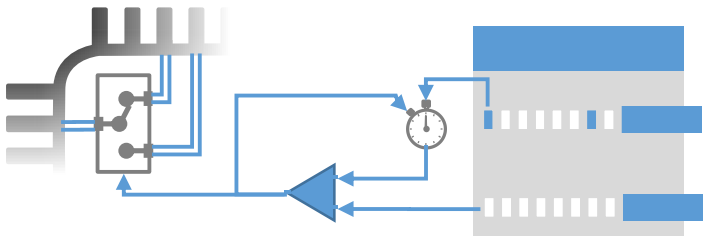
*Supostamente deveria ser automático, mas, na verdade, você tem que apertar este botão.*

*John Brunner*

# *Entrada digitais*

- Criação de um circuito para entrada de informação digital
  - A saída deve possuir apenas dois estados
  - O consumo de energia deve ser o mínimo possível
  - As tensões devem ser compatíveis com o circuito (TTL, CMOS, etc...)
- O circuito mais comum são os botões/teclas
  - A maioria dos botões são possuem dois contatos, para formar os dois níveis de tensão é necessário adicionar um resistor: pullup ou pulldown

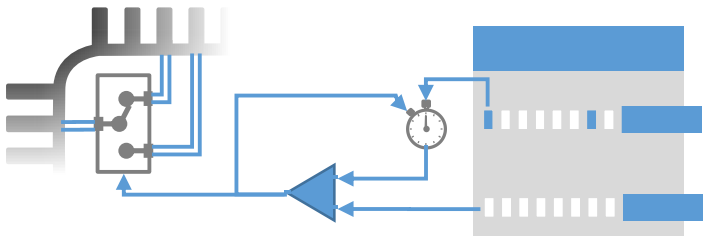


# *Leitura de teclas*

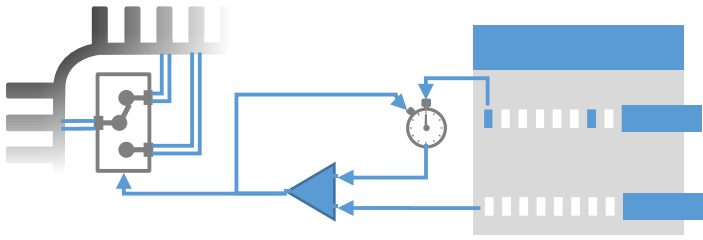
- Para fazer a leitura direto de uma porta
  - Inicializar o terminal como entrada
  - Testar o bit desejado

```
void main (void){
        //inicialização como entrada
        BitClr(TRISD,0);
        for(;;){
                //teste do bit
                if(BitTst(PORTD,0)){
                        //A chave está pressionada
                }else{
                        //A chave está solta
                }
        }
}
```

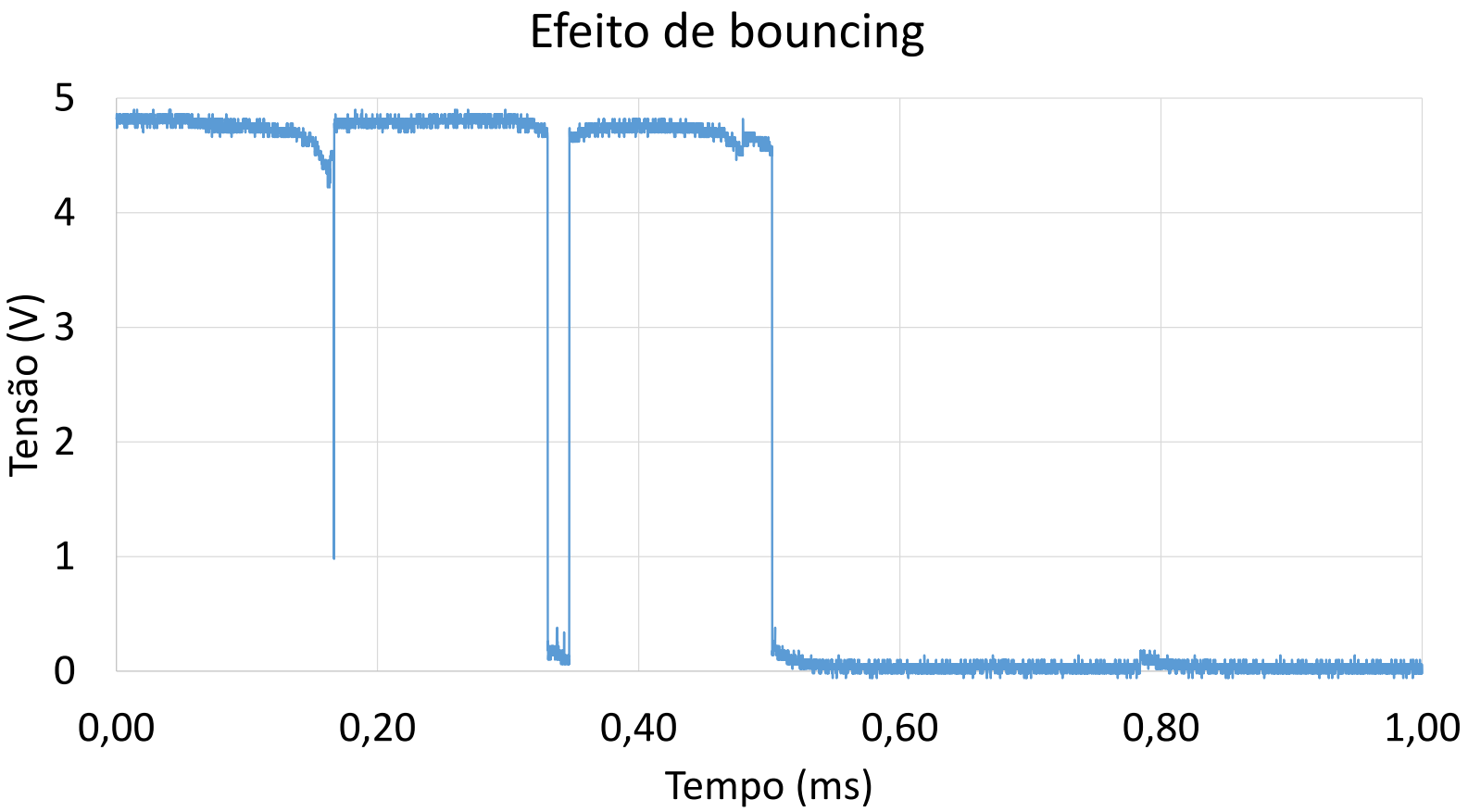


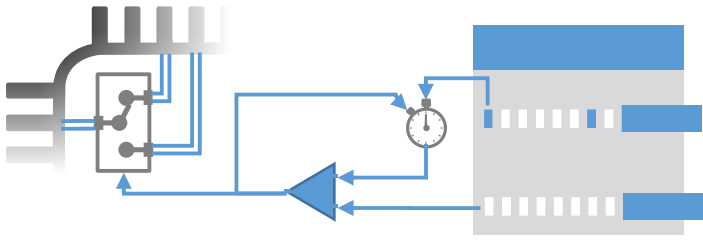
# *Bouncing*

- Problema de bouncing
  - Causado pela oscilação mecânica da chave
  - Pode gerar acionamentos indevidos no sistema
- Soluções
  - Via hardware através de circuito dedicado
  - Via software através de confirmação temporal



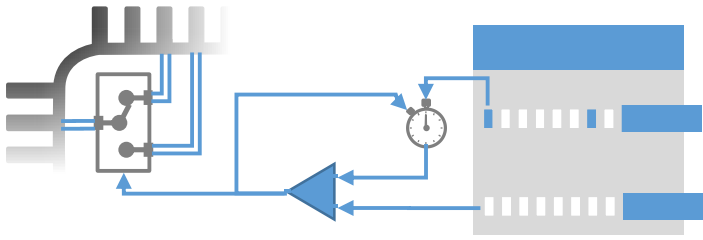
# Bouncing de uma tecla com pullup



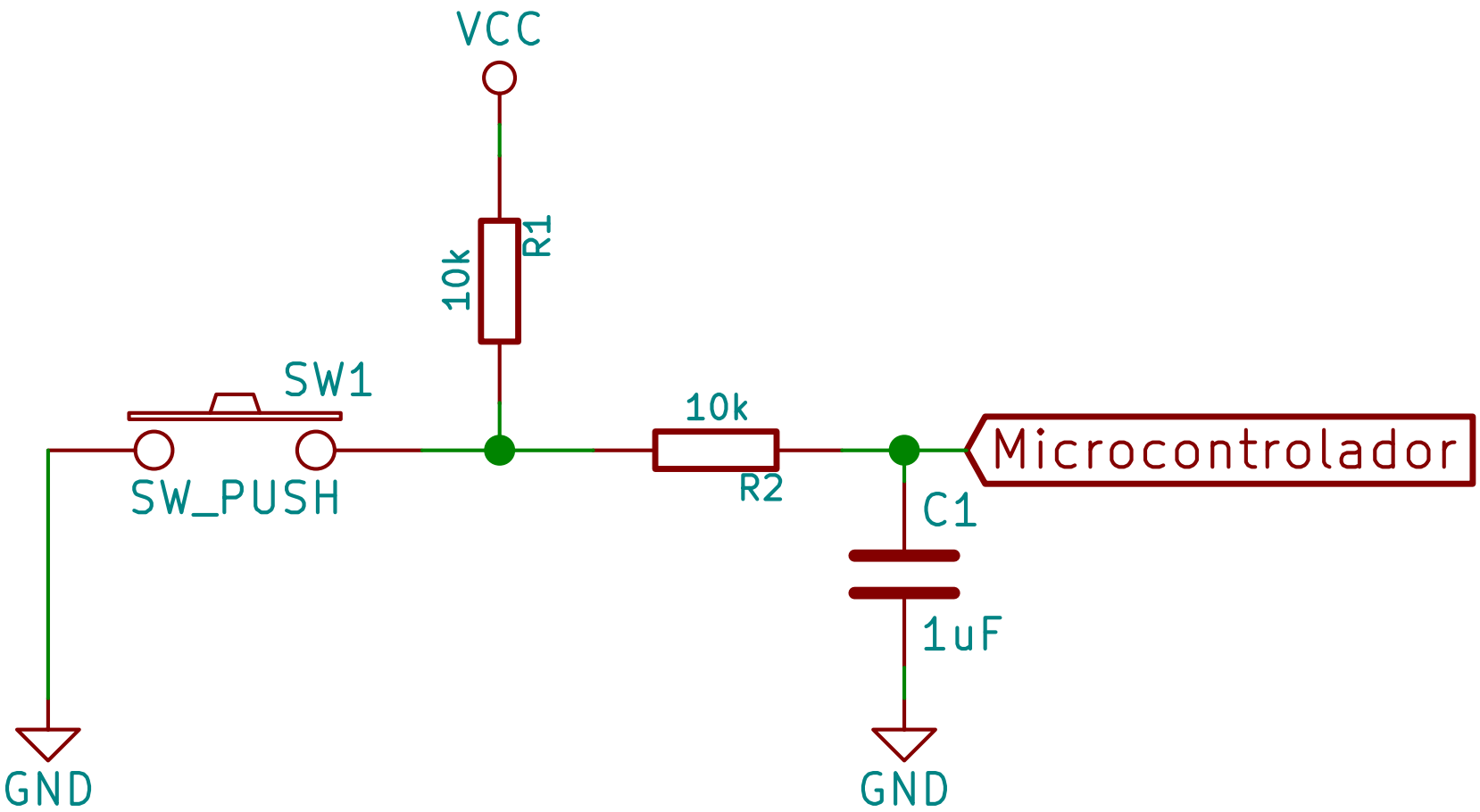



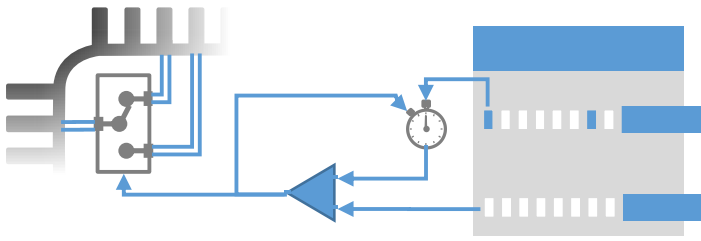
# *Debounce por hardware*

- Não gasta processamento
- Protege o circuito contra surtos
- Auxilia na estabilidade do sistema pois funciona como filtro
- Gera delay de RxC na resposta

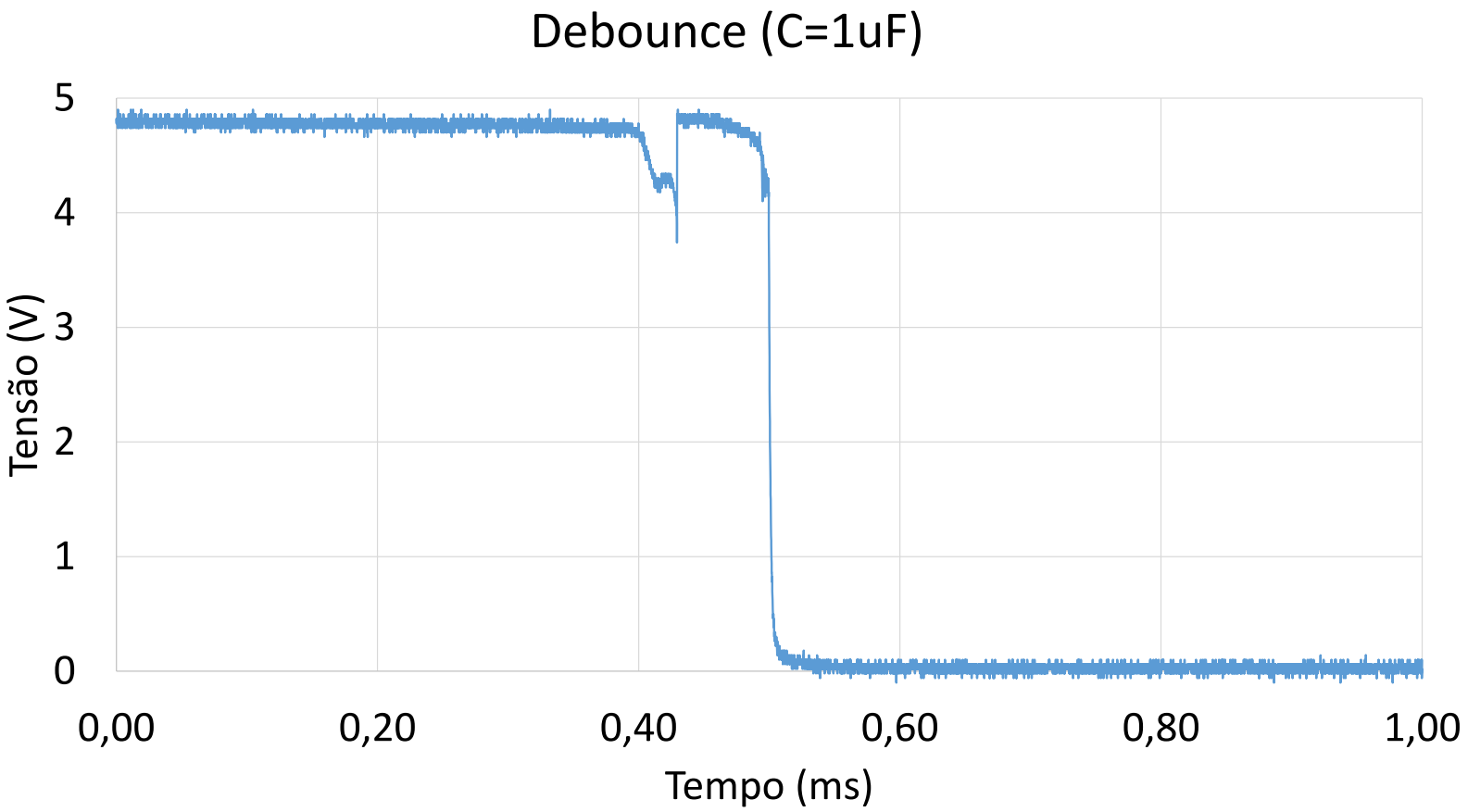


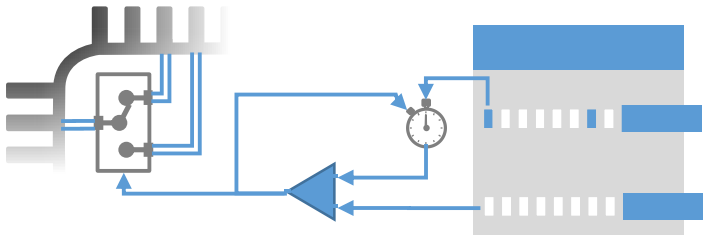
# Circuito de debounce com Capacitor



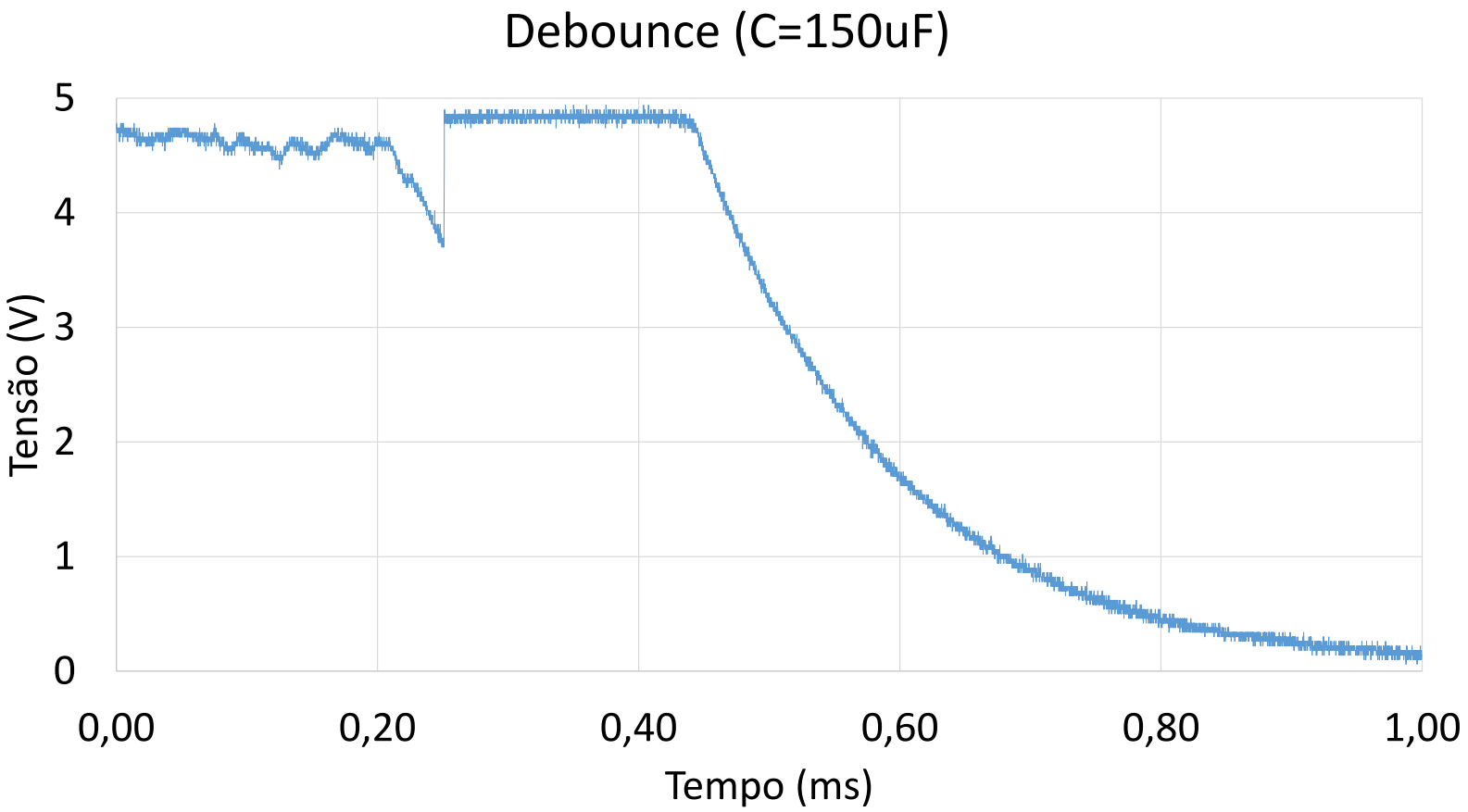



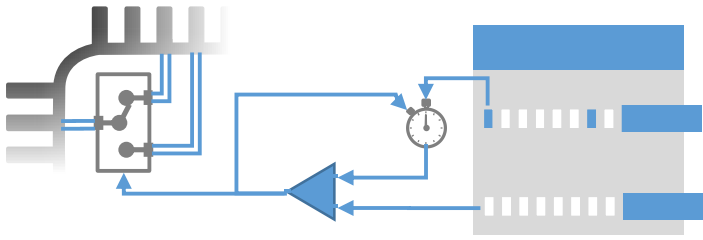
# Capacitor pequeno (mais rápido)



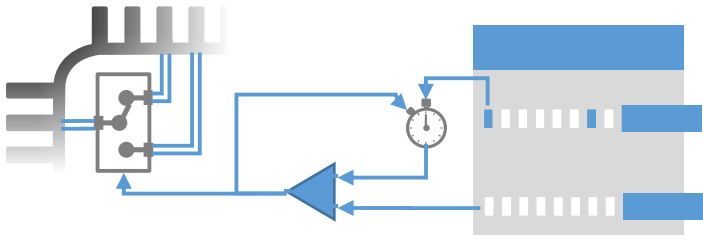
# Capacitor grande (menos ruídos)



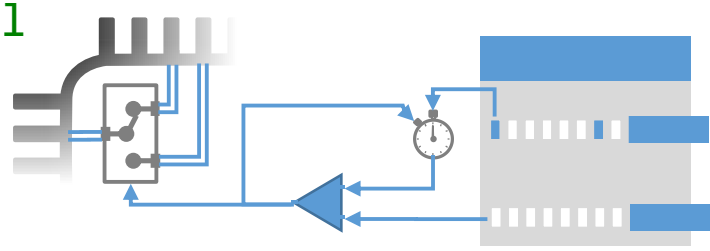



# *Debounce por software*

- Consome tempo e recurso do processador
- Não necessita de hardware ($$$) extra
- Gera delay de acordo com a rotina desenvolvida
- É preciso conhecer aproximadamente o tempo de estabilização da chave



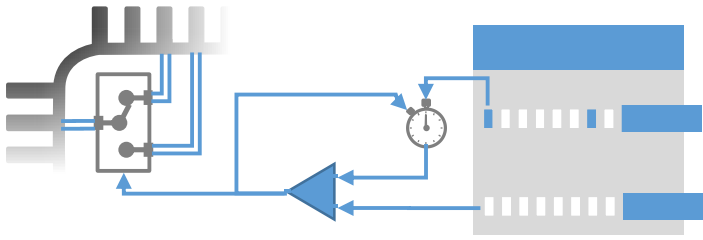
```
void main (void){
    int oldValue;
    int t;
    //inicialização como entrada
    pinMode(13,INPUT);
    //incializa o valor antigo
    oldValue = digitalRead(13);
    for(;;){
        //verifica se houve mudança
        if (oldValue != digitalRead(13)){
            //atualiza o sinal;
            oldValue == digitalRead(13);
            //se passar por 100 testes o sinal está estável
            for(t=0; t<100; t++){
                //se o valor mudar, reinicia a contagem
                if(oldValue != digitalRead(13)){
                    t = 0;
                }
            }
        }
        //aqui o valor da chave é estável
    }
}
```



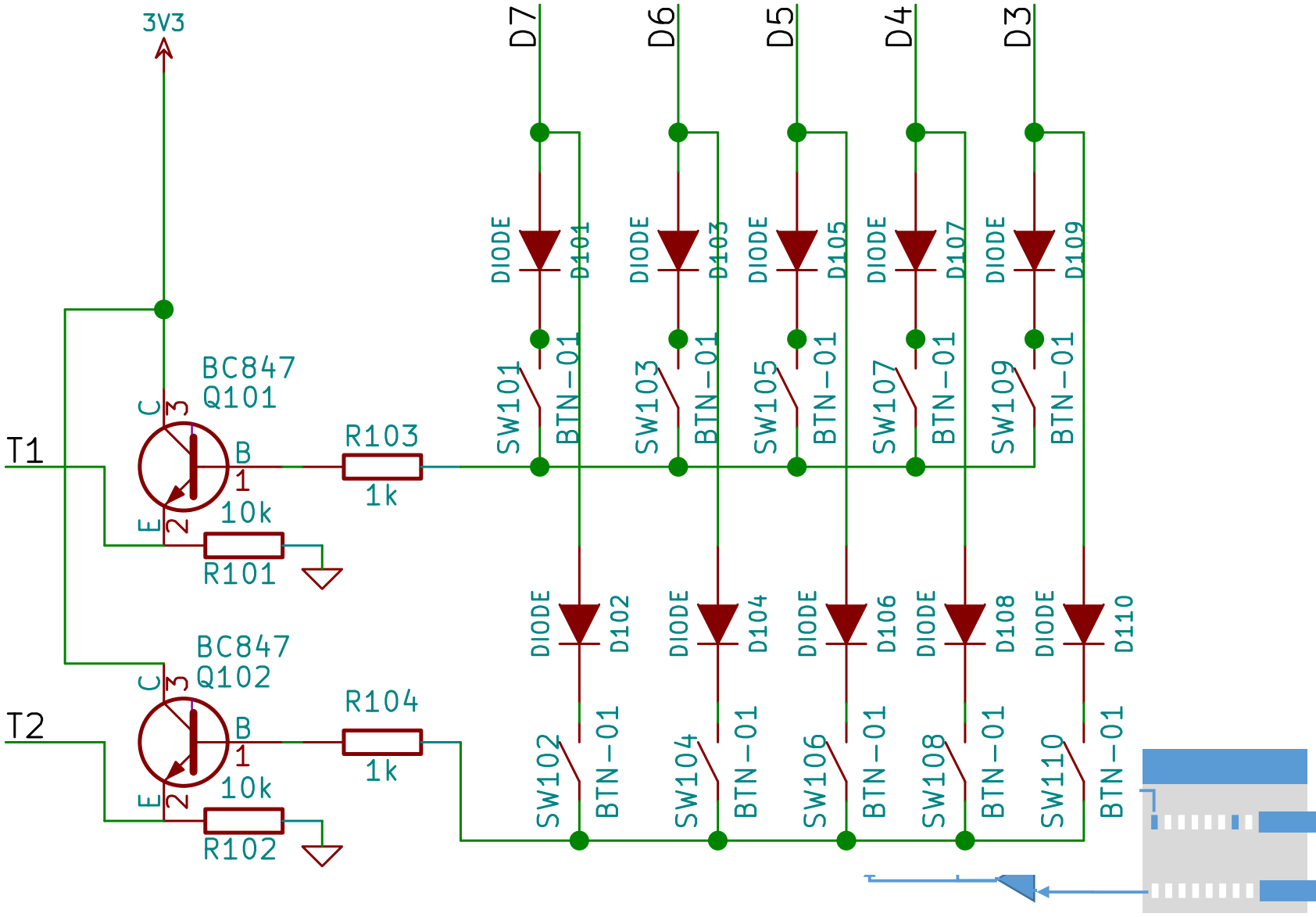
# *Leitura matricial*

# *Leitura matricial*

- Para cada tecla inserida no projeto, do modo apresentado até agora, é necessário um terminal de I/O no microcontrolador
- Projetos que exigem muitos terminais são mais caros por vários motivos
  - O encapsulamento pode gerar entre 5 e 30% de diferença no preço
  - O processo de inserção e soldagem de componentes é mais caro
  - O tamanho e complexidade da placa aumenta (quantidade da camadas, roteamento, etc...)
- Para entradas digitais do tipo teclas existe uma alternativa
  - Leitura por varredura matricial
- O teclado da placa utiliza um conversor transistorizado para 3.3v para não queimar as entradas das placas

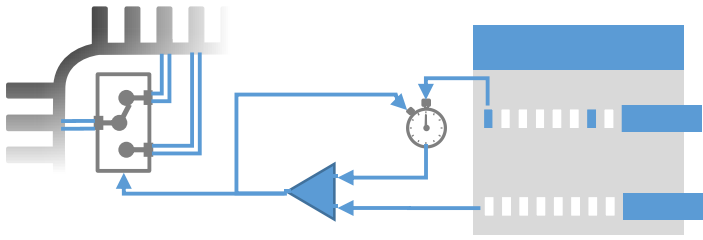


# Leitura matricial



# *Leitura matricial*

- Na leitura por varredura matricial temos:
  - Aumento da quantidade de chaves que podem ser lidas,
    - Para N terminais disponíveis no microcontrolador, temos um aumento de N chaves para $N^2$ chaves
  - Aumento da complexidade do software
  - Atraso na detecção de eventos (devido à varredura)
- O processo pode ser descrito como:
  1. Desligar todas as colunas
  2. Ligar apenas a coluna X
  3. Aguardar um tempo para estabilização dos sinais
  4. Realizar leitura nos terminais de entrada
  5. Passar para a próxima coluna X = X + 1

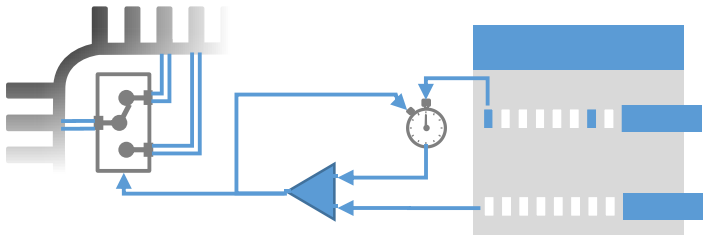


# Leitura matricial

```
#include "so.h"
void main(void){
    unsigned char coluna;
    unsigned char chave[5][2];
    soInit();
    pinMode(INPUT,12); pinMode(INPUT,13);
    for(;;){
        for(coluna = 0; coluna < 5; coluna++){
            //ligar apenas a coluna indicada
            soWrite(1<<coluna);
            //teste da 1a linha
            if (digitalRead(12)){ chave[coluna][0] = 1; }
            else                { chave[coluna][0] = 0; }
        //teste da 2a linha
            if (digitalRead(13)){ chave[coluna][1] = 1; }
            else                { chave[coluna][1] = 0; }
        }
    }
}
```

# *Leitura matricial*

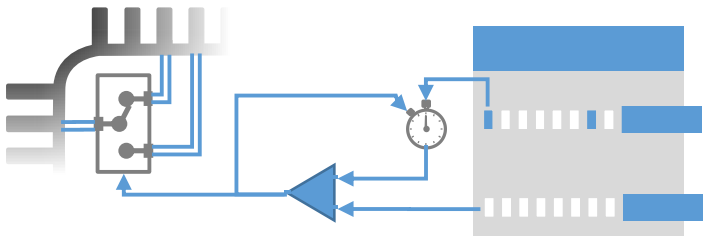
- O código não apresenta debounce em software para as teclas.
- A operação com uma matriz gasta muito processamento e memória.
- Para minimizar estes gastos podemos representar cada chave como um bit numa variável.
- A operação de debounce junto com a varredura será apresentada nas funções da biblioteca.



# *Criação da biblioteca keypad*

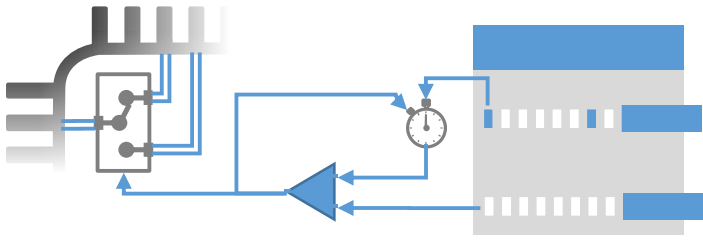
# *Criação da biblioteca keypad*

- Funções necessárias para a criação de uma biblioteca que realiza a leitura de um teclado matricial e disponibiliza as teclas pressionadas:
  - kpInit()
    - Configura os terminais de acionamento e de leitura
  - kpDebounce()
    - Realiza a leitura das teclas (varredura)
    - Realiza o debounce dos valores
    - Armazena internamente as teclas pressionadas
  - kpRead()
    - Retorna uma variável unsigned int
    - Cada bit desta variável representa uma tecla
    - 0 = desligado, 1 = pressionado
  - kpReadKey()
    - Retorna o “nome” da tecla pressionada
    - O “nome” é um caracter ASCII



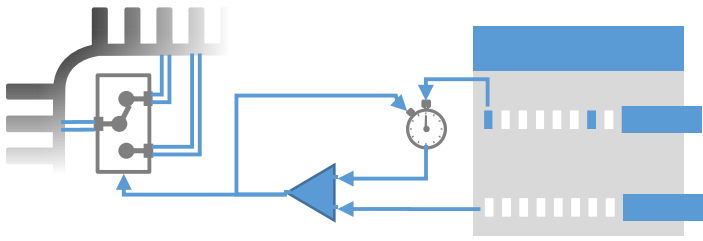
# *Criação da biblioteca keypad*

```
#ifndef KEYPAD_H
#define KEYPAD_H
        unsigned int kpRead(void);
        void kpDebounce(void);
        void kpInit(void);
#endif
```



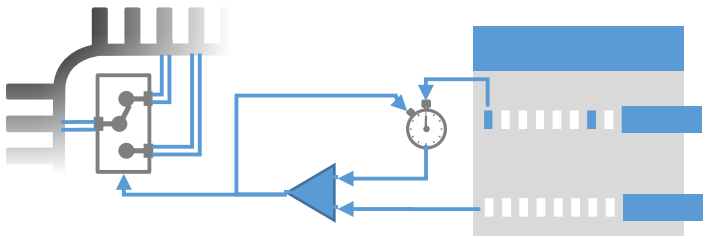
# *Criação da biblioteca keypad*

```
#include "keypad.h"
#include "so.h"
#include "io.h"

static unsigned int keys;

//vetor com o "nome" dos botões
//U -> up, L -> left, D -> down, R -> right
//S -> start, s -> select
//a ordem é referente a posição dos botões
static const char charKey[] =
{'U','L','D','R','S','s','Y','B','A','X'};

unsigned int kpRead(void) {
    return keys;
}
```

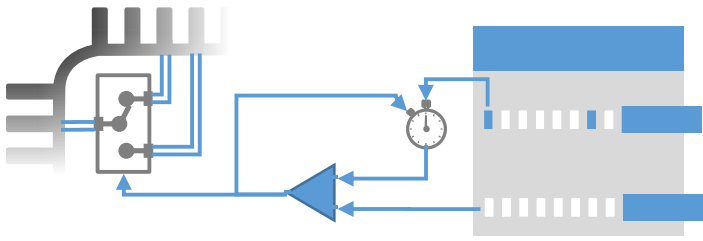


# *Criação da biblioteca keypad*

```
void kpInit(void) {
  soInit();
  pinMode(12, INPUT);
  pinMode(13, INPUT);
}

char kpReadKey(void){
      int i;
      for(i=0;i<10;i++){
            if (bitTst(keys,0)){
                  return charKey[i];
            }
      }
      //nenhuma tecla pressionada
      return 0;
}
```



```
void kpDebounce(void) {
        int i;
        static unsigned char tempo;
        static unsigned int newRead;
        static unsigned int oldRead;
        newRead = 0;
        for(i = 0; i<5; i++){
                soWrite(1<<(i+3));
                if(digitalRead(13)){
                        bitSet(newRead,i);
                }
                if(digitalRead(12)){
                        bitSet(newRead,(i+5));
                }
        }
        if (oldRead == newRead) {
                tempo--;
        } else {
                tempo = 4;
                oldRead = newRead;
        }
        if (tempo == 0) {
                keys = oldRead;
        }
}
```



# *Exemplo de utilização*

```
for(;;){
        kpDebounce();
        actualValue = kpRead();
        //houve algum evento?
        if (actualValue != lastValue){
                //processa os eventos
                if (bitTst(actualValue,0)){
                        //executa atividade
                        cont++;
                        if (cont>0xf){ cont = 0; }
                        ssdDigit(0,cont);
                }
                //armazena a mudança para a próxima rodada
                lastValue = actualValue;
        }
        ssdUpdate();
        //tempo pra evitar flicker
        for(t=0;t<100;t++);
}
```