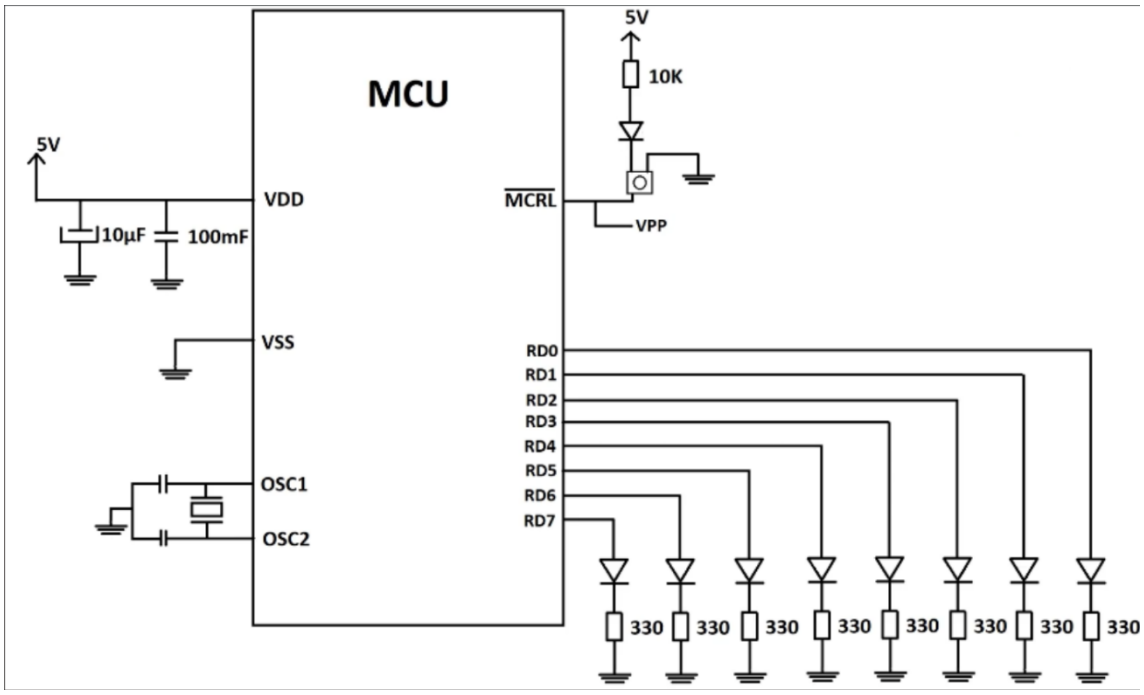
Módulo 2 - GPIO’s

Considerações gerais a respeito dos pinos

- Quase todos os pinos podem ser utilizados como Input/output, apenas os dedicados ao reset, clock e alimentação podem ser desconsiderados.
- Os pinos serem entradas ou saídas significa que você pode fazer acionamentos ou leituras de sinais.
- Fornecem até 25mA por pino
- Vem por padrão configurados como entradas analógicas no PIC18F45k22, você deve configurá-los como digitais a todo o momento

Circuito inicial

Você precisará de uma alimentação de 5V, um circuito de reset e um circuito oscilador. Também colocaremos LEDS no PORTD para realizar nosso primeiro acionamento. Monte a MCU com estes componentes:



Bits de configuração

A primeira coisa que fazemos quando abrimos um primeiro projeto é definir os bits de configuração. O assunto de bits de configuração é complexo e difere muito de nosso objetivo neste momento, por isso ele ficará para aulas mais a frente. A IDE MplabX nos ajuda com uma interface que define estes bits para nós. Ao finalizar de configurá-los como fizemos na aula, crie um arquivo chamado `config.h` e salve este código lá.

Criando sua biblioteca

O arquivo `config.h` pode ser colocado em uma biblioteca para facilitar a utilização futura. Para isso, crie uma pasta chamada “Bibliotecas” no mesmo local onde você mantém seus projetos do MPLAB. Depois, basta incluir estes bits de configuração com a seguinte diretiva:

```
#include "Bibliotecas/config.h"
```

Seu primeiro código

Depois de criar o cabeçalho com os bits de configuração, podemos escrever o código para acender os leds no PORTB. Inicialmente faremos o acionamento de todo o PORTB no formato de um pisca pisca:

```
#include <xc.h>
#include "config.h"

// Esta definição serve para a função __delay_ms
// Coloque a frequência do oscilador que você está utilizando
#define _XTAL_FREQ 8000000

void main(void) {
      // Define todo o PORTB como saída
      TRISB = 0;

      // Loop infinto
      while(1){
            // Coloca todo o PORTB em nível lógico alto
            PORTB = 0xFF; // é o mesmo que 0b11111111
            __delay_ms(1000); // Aguarda 1 segundo
            // Todo o PORTB em nível lógico baixo
            PORTB = 0;
            __delay_ms(1000);
      }
}
```

O código do microcontrolador não tem fim. Ele não pode dar um return na main pois não há onde retornar. O que acontecerá é que o MCU vai continuar lendo endereços de memória até que ele sofra um reset e retorne ao primeiro endereço. Por isso a importância do laço while.

A compilação do código vai gerar um hexadecimal no formato intel HEX. Existe bastante material pela internet sobre como funciona este formato, mas neste

momento não é necessário entender. É justamente este arquivo que será carregado no microcontrolador.

Função delay

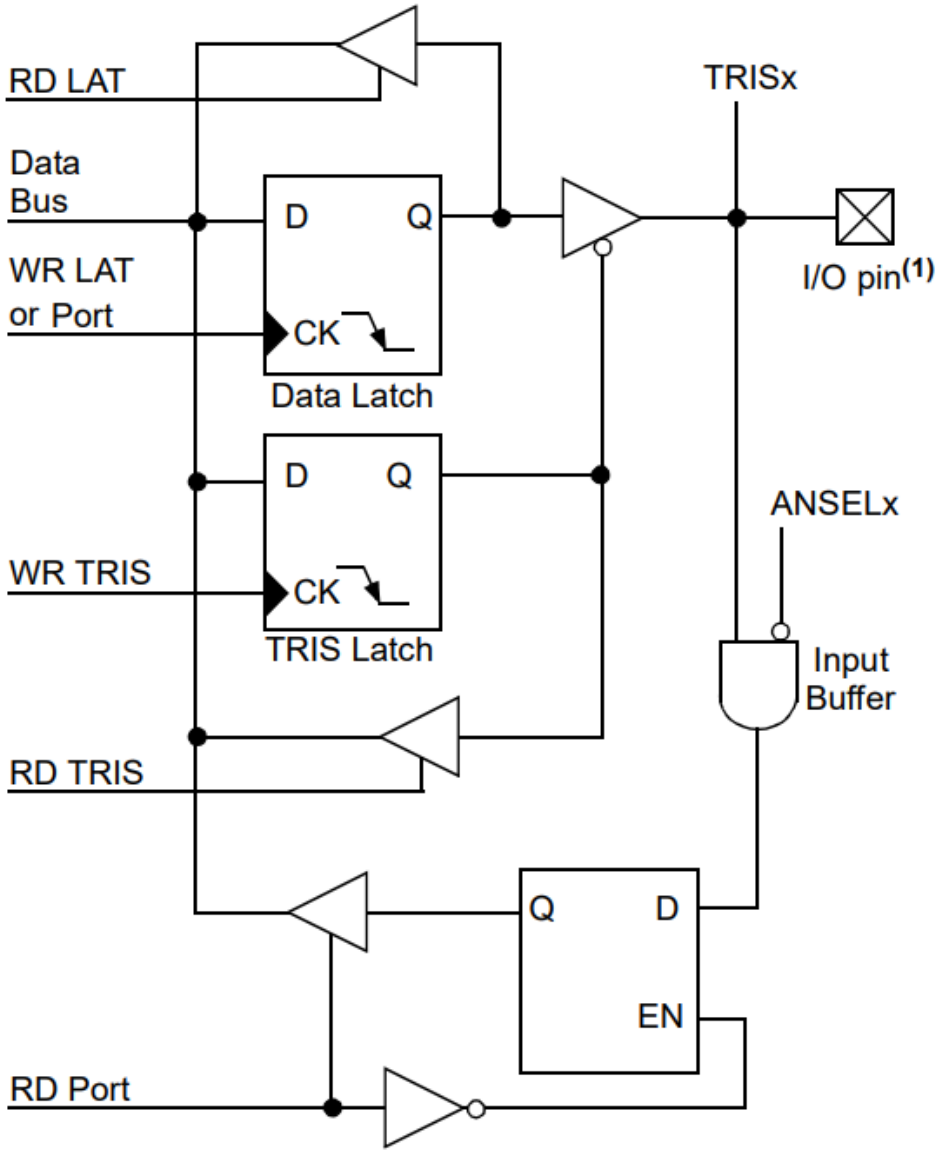
A função "delay" em microcontroladores é uma função que causa uma pausa ou atraso temporário na execução do código. Ela é usada principalmente para controlar o tempo de execução de tarefas ou para criar atrasos propositais em um programa. A importância da função "delay" em microcontroladores é variável e depende do aplicativo específico e dos requisitos de tempo do projeto. Aqui estão algumas razões pelas quais a função "delay" é importante:

1. Temporização precisa: Em muitos sistemas embarcados, é necessário controlar o tempo com precisão. Isso pode ser necessário para gerar sinais de temporização, sincronizar dispositivos externos, ou para garantir que uma ação específica ocorra após um determinado intervalo de tempo. A função "delay" ajuda a atingir essa precisão, permitindo que os desenvolvedores programem intervalos de tempo específicos.
2. Sincronização de periféricos: Em sistemas que envolvem múltiplos periféricos ou componentes, a função "delay" pode ser usada para garantir que os eventos ocorram em uma ordem específica ou para sincronizar a comunicação entre eles. Por exemplo, pode ser necessário esperar um tempo específico após enviar um comando para um dispositivo antes de ler os dados de resposta.
3. Debounce de entradas: Em muitos casos, as entradas digitais (por exemplo, botões) podem gerar ruído elétrico que causa flutuações nos valores lidos. A função "delay" pode ser usada para implementar uma técnica de debounce, onde o programa aguarda um curto período após uma mudança de estado antes de reconhecer a entrada como válida.
4. Simulação de processos em tempo real: Em sistemas de controle em tempo real, é importante garantir que as ações ocorram dentro de intervalos de tempo específicos. A função "delay" pode ser usada para criar atrasos necessários para simular esses intervalos de tempo durante o desenvolvimento e teste do sistema.
5. Economia de energia: Em sistemas alimentados por bateria ou energia limitada, a função "delay" pode ser usada para colocar o microcontrolador em um modo de baixo consumo de energia durante períodos de inatividade. Isso ajuda a prolongar a vida útil da bateria ou reduzir o consumo de energia.

É importante notar que a implementação da função "delay" em microcontroladores pode variar de acordo com a arquitetura do microcontrolador e o ambiente de desenvolvimento. Alguns microcontroladores têm timers integrados que podem ser usados para criar atrasos precisos, enquanto outros podem depender de loops de contagem de ciclos de clock para criar atrasos. A escolha da técnica de atraso depende dos requisitos específicos do projeto e das capacidades do microcontrolador.

Esquemático do TRIS e PORT

Muitos profissionais de eletrônica se perguntam como é possível um pino poder ser tanto entrada como saída. Acontece que, para cada pino, existe uma sequência de portas lógicas que são alteradas justamente pelo registrador TRIS. Ainda, quando configurada como entrada, uma porta pode ser definida como entrada digital ou analógica pelo registrador ANSEL (quando disponível).

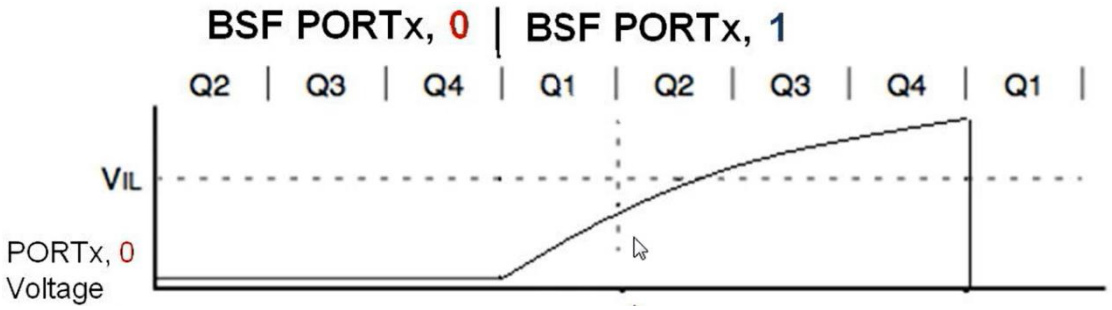


Note 1: I/O pins have diode protection to VDD and VSS.

Registrador LAT

O registrador PORT armazena exatamente o nível lógico presente no pino. Há um problema nisso, as vezes, o nível lógico do pino não está no valor esperado, mesmo que tenha sido setado antes.

Isso acontece pois uma placa de circuito pode apresentar uma capacitância que resulta em um tempo de carregamento até a tensão no pino atingir um mínimo legível (VIL).



Se o PORT for lido antes desta tensão mínima ter sido atingido, ele retornará zero mesmo que tenha sido setado para 1.

O LAT resolve este problema salvando o nível lógico do pino em um buffer (espécie de memória). Esta memória é setada imediatamente após a alteração no LAT, isso garante que você esteja lendo o nível lógico real do pino.

Na maioria dos casos, utilizar LAT ou PORT não fará a menor diferença. Só é necessário quando você cria uma rotina onde uma parte do programa seta determinado pino para 1 ou 0 e outra parte pode querer consultar este valor muito rapidamente.

Pinos setados como entrada

Para setar os pinos como entrada, e ler o nível lógico que chega para ele, basta setar o registrador TRIS do respectivo pino para 1.

```
// Define o pino 4 do PORT A como entrada
TRISAbits.TRISA4 = 1;
```

Agora, ler este pino, utiliza-se o PORT respectivo do pino:

```
if(PORTAbits.RA4 == 0){ // Se o nivel logico do pino RA4 for 0
      // Faz alguma coisa
}
```

No caso de pressionamento de chaves (ou botões), existe o chamado debouncing. O "debouncing" é um problema de ruído elétrico e instabilidade nas entradas digitais, como botões ou chaves, em circuitos de microcontroladores. Quando uma chave ou botão é pressionado ou liberado, o sinal elétrico pode oscilar rapidamente entre os estados lógicos 0 e 1 devido a fenômenos elétricos, resultando em leituras errôneas pelo microcontrolador. Tratamos o debouncing para garantir que o microcontrolador reconheça apenas um único evento de pressionar ou liberar a chave, evitando múltiplas detecções devido a flutuações no sinal.

Existem várias maneiras de tratar o debouncing em circuitos de microcontroladores:

1. **Hardware**: O tratamento pode ser realizado com componentes externos, como resistores e capacitores, para filtrar o ruído e estabilizar o sinal. Um circuito RC (resistor-capacitor) é frequentemente usado para criar uma transição suave do sinal quando a chave é pressionada ou liberada. O tempo de constante de carga e descarga do capacitor determina a velocidade do debouncing.
2. **Software**: O tratamento também pode ser implementado por software. Nesse caso, o microcontrolador aguarda um período após a detecção da mudança de estado da entrada antes de considerar a entrada como válida. Esse período é chamado de "tempo de debounce" e é geralmente implementado usando um temporizador ou uma função de delay. Durante esse tempo, o microcontrolador verifica constantemente o estado da entrada e aceita apenas um único evento de pressionamento ou liberação, ignorando as flutuações.

Abaixo, há um exemplo onde tratamos o debouncing via software com uma função delay. Está é a forma mais simples. O recomendado é combinar soluções de hardware e software para obter o melhor resultado.

```
#define BOTAO PORTAbits.RA4
#define PRESSIONADO 0

if(BOTAO == PRESSIONADO){
      // Faz alguma coisa
      __delay_ms(300); // Aguarda o debouncing
}
```

Outra forma é travando o processamento até que o botão seja solto:

```
#define BOTAO PORTAbits.RA4
#define PRESSIONADO 0

if(BOTAO == PRESSIONADO){
      // Aguarda soltar o botão
      while(BOTAO == PRESSIONADO) delay_ms(20);
      // Faz alguma coisa
}
```