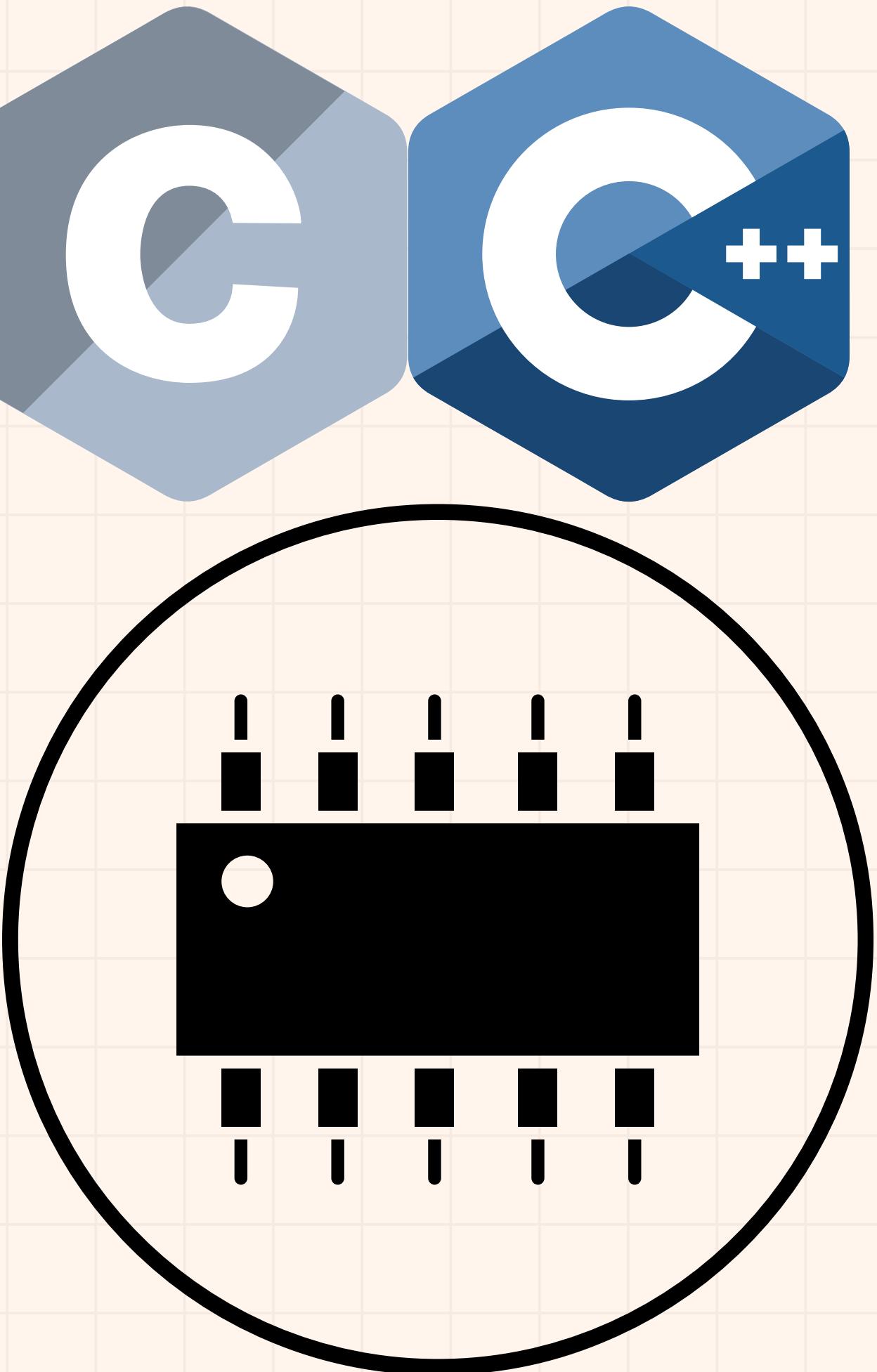


PROGRAMANDO EM C/C++ PARA PIC



INTRODUÇÃO

OBJETIVOS:

O objetivo deste minicurso é explicar e exemplificar o básico da programação em C para utilizar microcontroladores para as diversas funções para as quais serve.

LINK DO GITHUB DO CURSO:



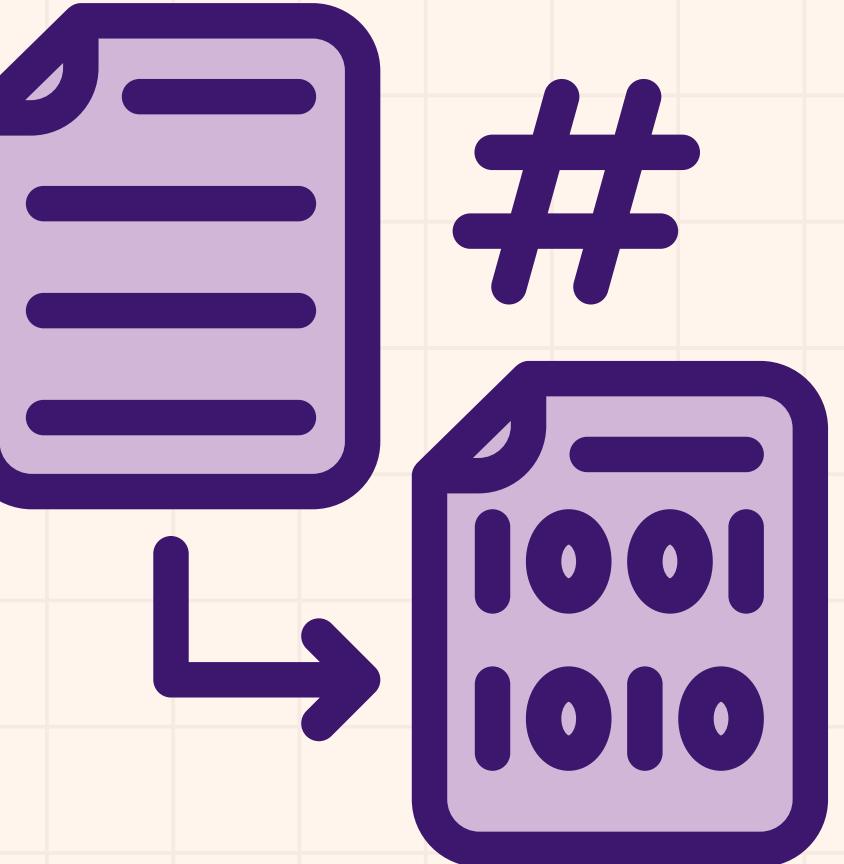
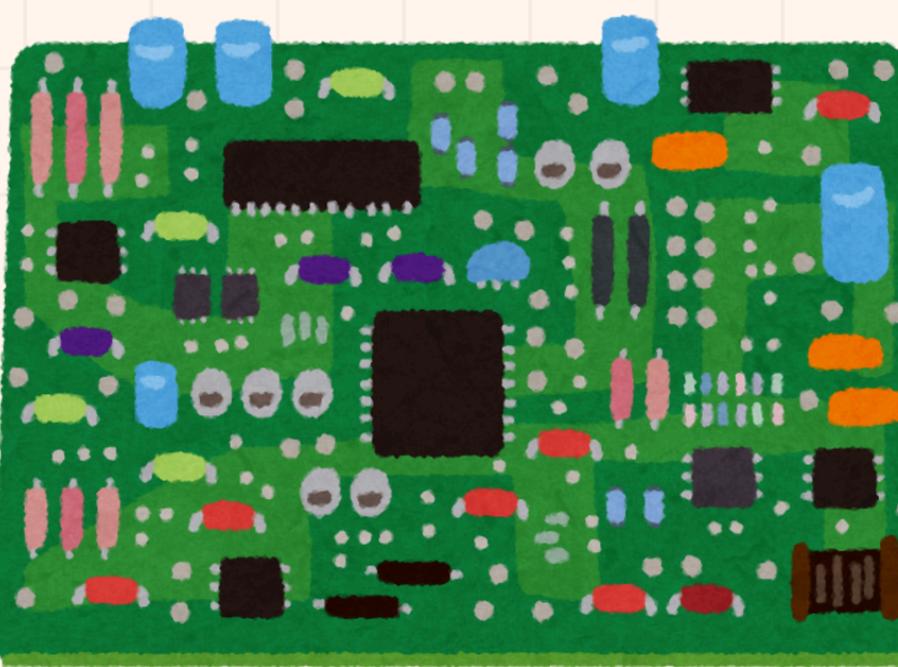
[HTTPS://GITHUB.COM/ERICKSEBBA/MINICURCOPIC](https://github.com/ericksebba/minicurcopic)

O QUE É UM MICROCONTROLADOR

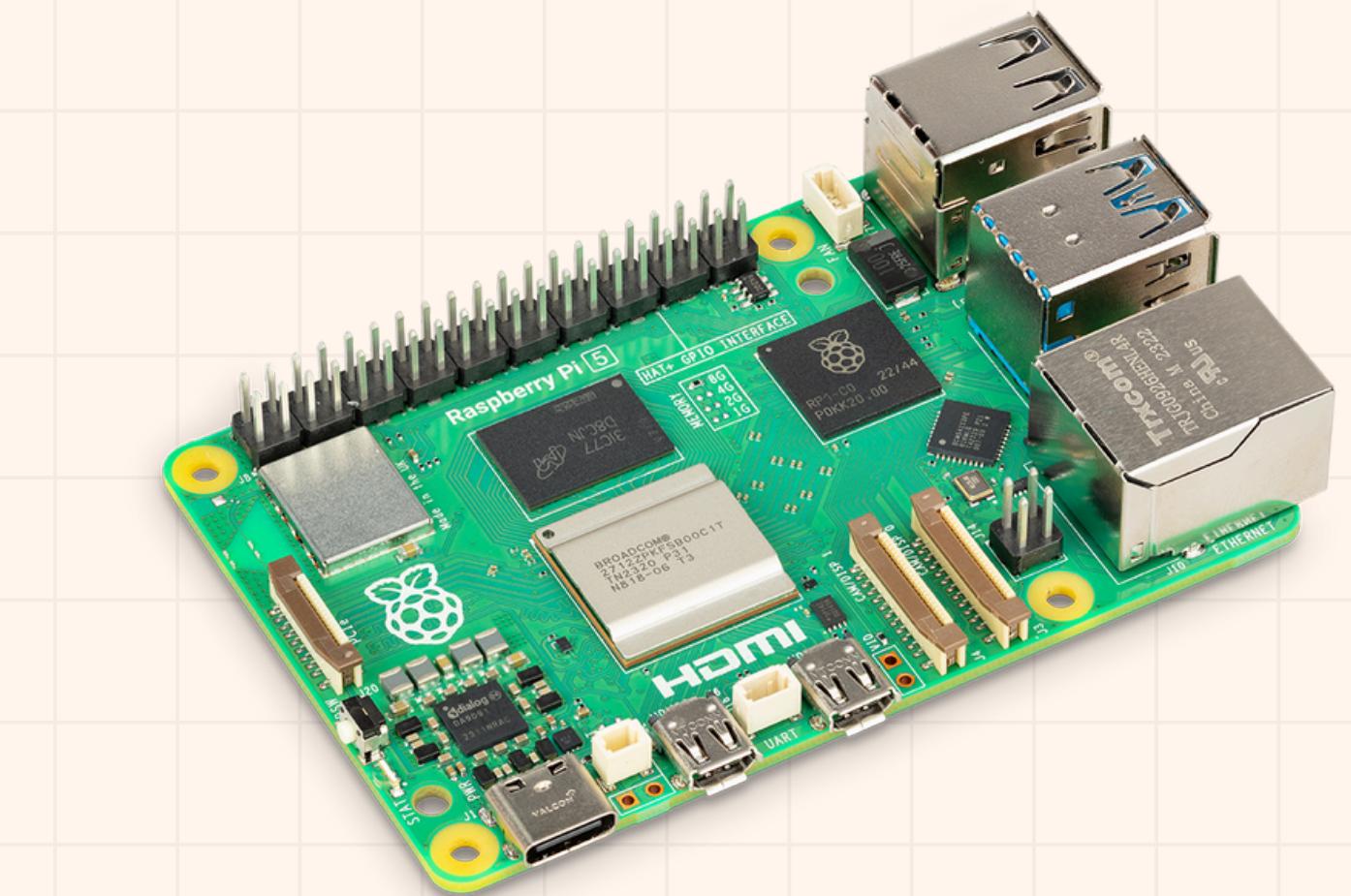
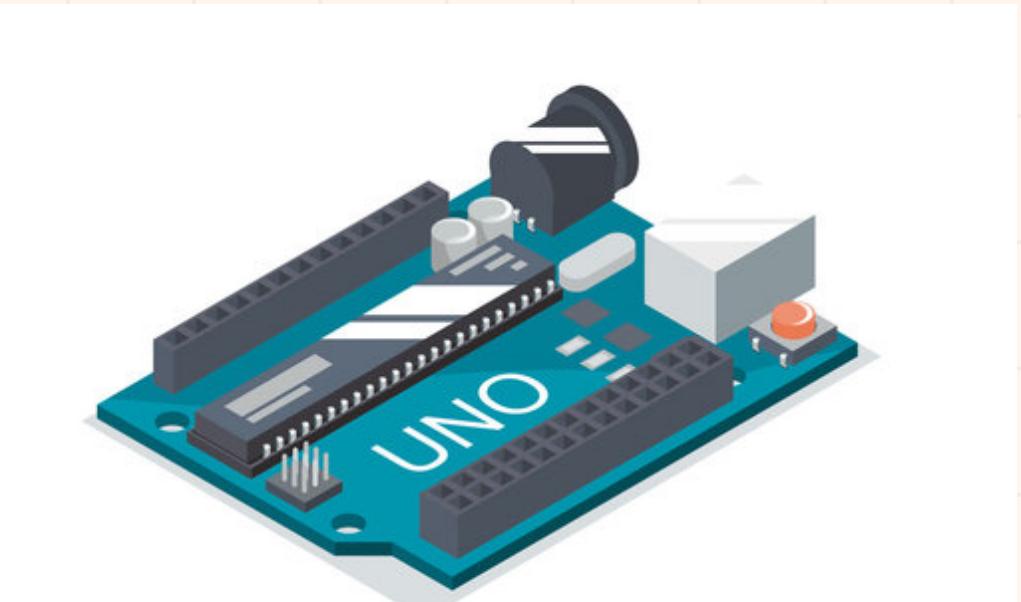
Um microcontrolador é um chip que contém um computador em miniatura, com um processador, memória, periféricos de entrada e saída, entre outros componentes. Ele é responsável por controlar circuitos eletrônicos por meio de algoritmos programáveis.

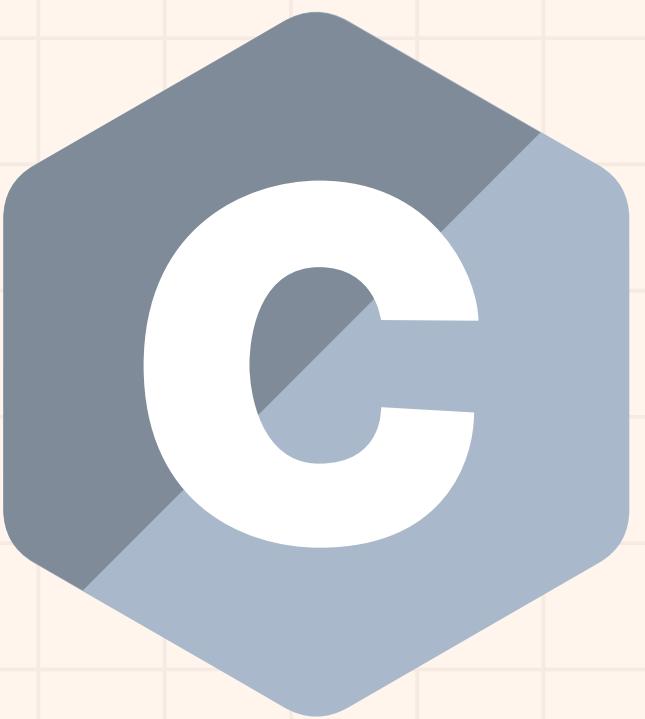
Os microcontroladores são usados em uma grande variedade de produtos e dispositivos, como:

- Controles remotos
- Eletrodomésticos
- Controles de carros
- Brinquedos
- Dispositivos médicos implantáveis
- Máquinas de escritório
- Ferramentas elétricas



O QUE É UM MICROCONTROLADOR



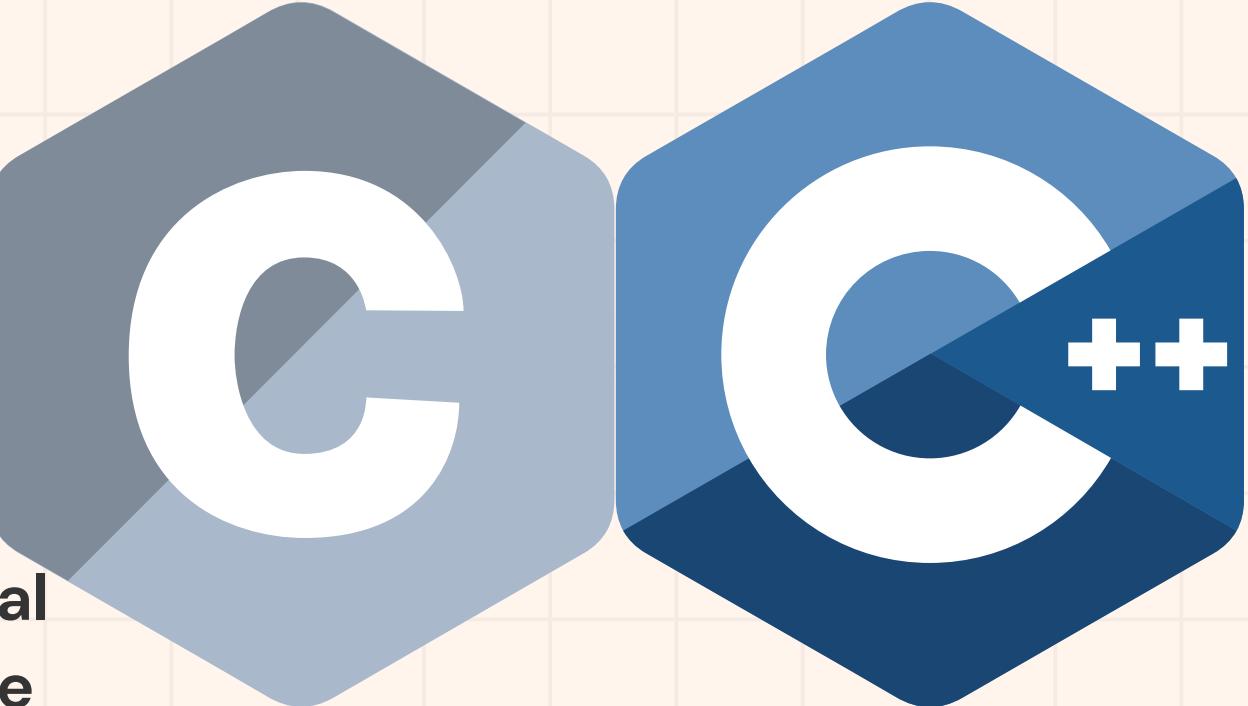


POR QUE PROGAMAR EM C NO MICROCONTROLADOR

Neste Minicurso iremos apresentar as vantagens e como programar microcontroladores em geral utilizando a linguagem C.

Mas primeiro, qual é a vantagem de se programar em C puro, ante as outras linguagens: A resposta é simples, o compilador da linguagem C é tão rápida quanto quanto o Assembly do chip!

POR QUE PROGAMAR EM C NO MICROCONTROLADOR



- **Propósito Geral:** C é uma linguagem de programação de propósito geral usada para uma ampla gama de desenvolvimentos de software, desde sistemas operacionais até sistemas embarcados.
- **Sintaxe:** C possui uma sintaxe mais complexa e exige o manuseio explícito de operações de baixo nível, como gerenciamento de memória e interação com hardware.
- **Compilador:** O código em C é compilado usando um compilador C, que converte o código em código de máquina que um computador ou microcontrolador pode executar.
- **Bibliotecas Padrão:** C possui um conjunto rico de bibliotecas padrão, mas frequentemente é necessário incluir bibliotecas específicas para interação com hardware

ESTRUTURA

The diagram illustrates the sequential execution of a C program. It features a dark rectangular box containing the code, with three colored dots (red, yellow, green) positioned above it. A purple arrow points from the word "INICIO" (beginning) at the top right towards the first dot. Another purple arrow points from the last dot towards the word "FIM" (end) at the bottom right.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     printf("Hello World\n");
6     system("pause");
7     return 0;
8 }
```

A screenshot of a terminal window on a dark background. It displays the same C program code as the diagram. The output shows the program's execution: it includes the standard libraries, defines the main function, prints "Hello World", and pauses before exiting with a status of 0.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     int a = 10;
6     a=2;
7     printf("O valor de a é: %d\n", a);
8     system("pause");
9     return 0;
10 }
```

LINGUAGEM ESTRUTURADA
A linguagem C é estruturada, sua principal característica é a sequência de execução do programa

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     printf("Hello World\n");
6     system("pause");
7     return 0;
8 }
```

```
PS C:\Users\erick\OneDrive\Documentos\GitHub\MinicursoPIC\CodigosC\EstruturaDaLinguagem\output> & .\estrutura.exe
Hello World
Pressione qualquer tecla para continuar. . .
```

```
● ● ●  
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4 int main(){  
5     int a = 10;  
6     a=2;  
7     printf("O valor de a é: %d\n", a);  
8     system("pause");  
9     return 0;  
10 }
```

```
PS C:\Users\erick\OneDrive\Documentos\GitHub\MinicursoPIC\CodigosC\EstruturaDaLinguagem\output> & .\estrutura2.exe'  
O valor de a eh: 2  
Pressione qualquer tecla para continuar. . .
```

ENDENTAÇÃO

Endentação é o nome que se dá à técnica de separar graficamente cada bloco de instruções de acordo com seu grau de pertinência.

Os compiladores da linguagem C não necessitam de endentação para funcionar corretamente, entretanto ela melhora muito o entendimento do Código.



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     for(int i = 0; i < 10; i++){
6         if(i % 2 == 0){
7             printf("i eh par e eh: %d\n", i);
8         }
9     else{
10         printf("i eh impar e eh: %d\n", i);
11     }
12 }
13 }
```



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     for(int i = 0; i < 10; i++){
6         if(i % 2 == 0){
7             printf("i eh par e eh: %d\n", i);
8         }
9     else{
10         printf("i eh impar e eh: %d\n", i);
11     }
12 }
13 }
```



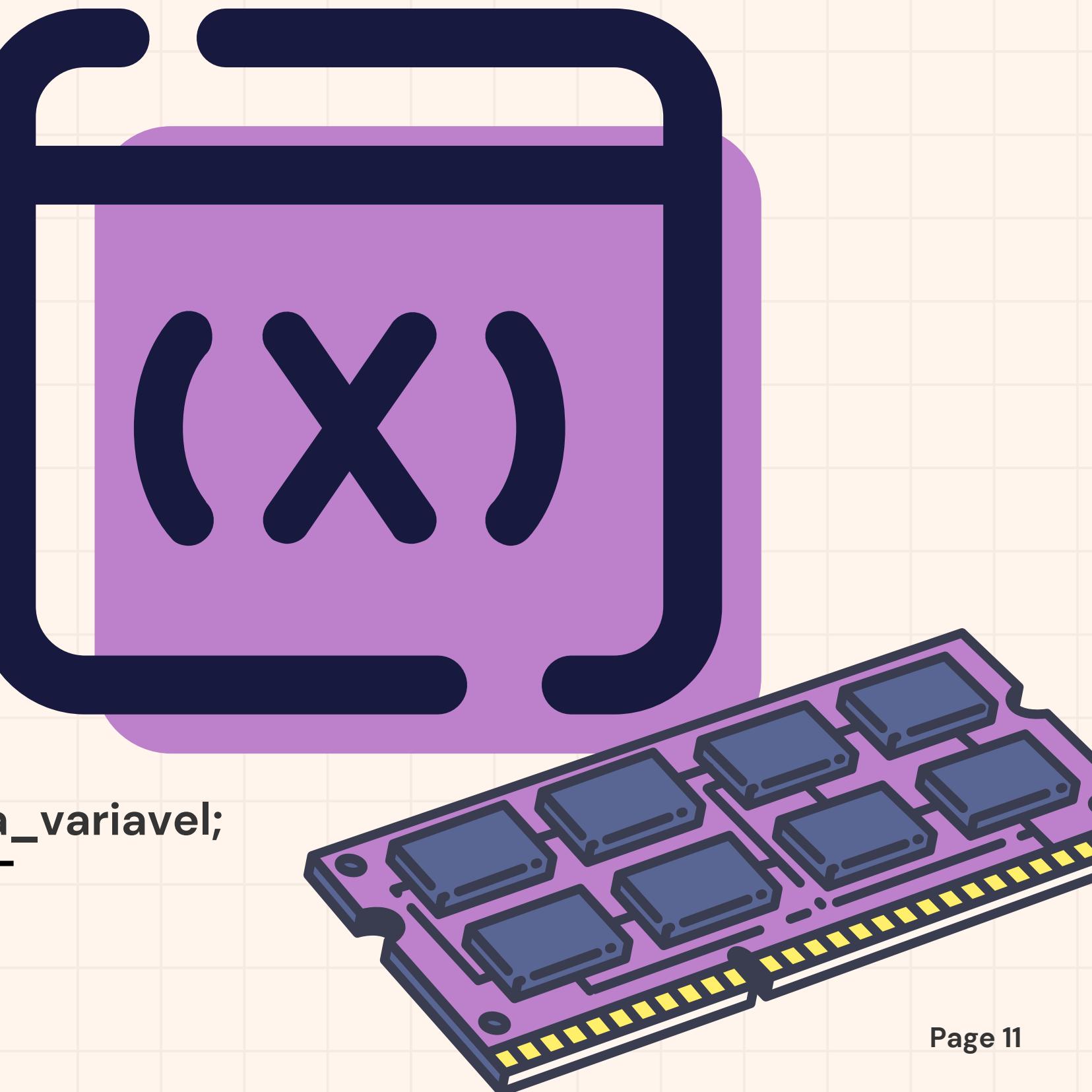
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     for(int i = 0; i < 10; i++){
6         if(i % 2 == 0){
7             printf("i eh par e eh: %d\n", i);
8         }
9     else{
10         printf("i eh impar e eh: %d\n", i);
11     }
12 }
13 }
```

VARIÁVEIS E TIPOS DE DADOS

variável é um espaço na memória RAM, alocado para armazenar um determinado valor para uso futuro. Esse valor será modificado e utilizado para efetuar operações, comparações e outras instruções

Uma variável sempre deve ser declarada, ou seja, todas as variáveis utilizadas em uma função precisam ser relacionadas no início desta função. A declaração de variáveis é feita conforme exemplo a seguir:

<modificador de escopo> <modificador de tipo> <tipo> nome_da_variavel;





```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     char a = 'a';
6     int b = 10;
7     float c = 10.5;
8     double d = 10.55901;
9
10    signed char e = -10;
11    short int f = 10;
12    long int g = 10;
13    long long int h = 10;
14    unsigned char i = 10;
15
16    register char k = 10;
17    static int l = 10;
18    const unsigned char m = 10;
19    volatile int n = 10;
20    extern float o;//error an initializer is not allowed on a local declaration of an extern variable
21 }
```

Tab. 4.2. Tipos de variáveis.

Declaração	Bytes	Variação
char	1	-128 a 127
int*	2	-32.768 a 32.767
float	4	$1,2 \times 10^{-38}$ a $3,4 \times 10^{+38}$
double	8	$2,2 \times 10^{-308}$ a $1,8 \times 10^{+308}$

* Depende da arquitetura do processador, sistema operacional e compilador utilizado.

Tab. 4.3 Modificadores de tipo.

Declaração	Bytes	Variação
signed char	1	-128 a 127
unsigned char	1	0 a 255
short	2	-32.768 a 32.767
short int	2	-32.768 a 32.767
signed int	2	-32.768 a 32.767
signed short	2	-32.768 a 32.767
signed short int	2	-32.768 a 32.767
unsigned int	2	0 a 65.535
unsigned short	2	0 a 65.535
unsigned short int	2	0 a 65.535
long	4	-2.147.438.648 a 2.147.438.647
long int	4	-2.147.438.648 a 2.147.438.647
signed long	4	-2.147.438.648 a 2.147.438.647
signed long int	4	-2.147.438.648 a 2.147.438.647
unsigned long	4	0 a 4.294.967.295
unsigned long int	4	0 a 4.294.967.295
long double*	8	$2,2 \times 10^{-308}$ a $1,8 \times 10^{+308}$

* Dependendo do compilador pode ser atribuído 8, 10 ou 12 bytes.

Tab. 4.2. Tipos de variáveis.

Declaração	Bytes	Variação
char	1	-128 a 127
int*	2	-32.768 a 32.767
float	4	$1,2 \times 10^{-38}$ a $3,4 \times 10^{+38}$
double	8	$2,2 \times 10^{-308}$ a $1,8 \times 10^{+308}$

* Depende da arquitetura do processador, sistema operacional e compilador utilizado.

NOME DE VARIÁVEIS E PALAVRAS RESERVADAS

A linguagem C17 possui 32 comandos e palavras reservadas, ou seja, palavras que executam funções específicas e que não podem ser utilizadas como nomes de funções ou variáveis, conforme tabela abaixo:

Tab. 4.1. Palavras reservadas da linguagem C.

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char auto = 'a';
    int break = 10;
    float continue = 10.5;
    double default = 10.55901;
}
```

NOME DE VARIÁVEIS

Sendo o nome escolhido para a variável diferente de alguma palavra reservada é importante lembrar que das mais importantes características da linguagem C é a sensibilidade à caixa, isto é, nomes de funções, variáveis e constantes declaradas são tratados como entidades diferentes se escritas com letras maiúsculas ou minúscula. Além disso o ÚNICO caráter especial que pode ser usado é o "_"



```
1
2 int media; //uma variável
3 int Media; //outra variável
4 int mEdIa; //uma terceira variável
5 int MEDIA; //uma quarta variável
6
```

NOME DE VARIÁVEIS



```
1 int variavel;  
2 int _variavel;  
3 int vari_avel;  
4 int /variavel; //invalido
```

MODIFICADORES DE TIPOS

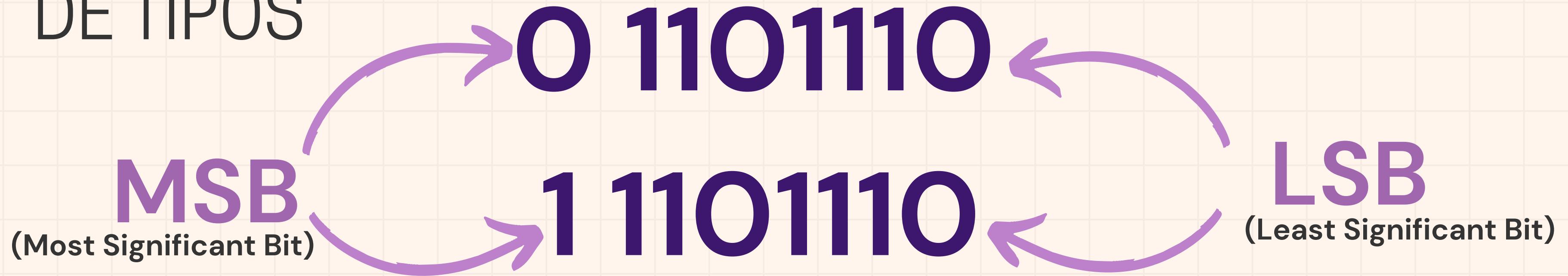
Os quatro tipos de dados podem receber os modificadores opcionais de modo a criar um tipo de dado mais adequado para cada aplicação. A instrução **short** informa ao compilador que o tamanho da variável necessária é menor que o padrão do tipo de dado; de forma análoga, a instrução **long** informa que o tamanho necessário é maior que o tamanho padrão do tipo de dado, e as instruções **unsigned** e **signed** informam que os valores armazenados serão apenas positivos ou positivos e negativos, respectivamente

Tab. 4.3 Modificadores de tipo.

Declaração	Bytes	Variação
signed char	1	-128 a 127
unsigned char	1	0 a 255
short	2	-32.768 a 32.767
short int	2	-32.768 a 32.767
signed int	2	-32.768 a 32.767
signed short	2	-32.768 a 32.767
signed short int	2	-32.768 a 32.767
unsigned int	2	0 a 65.535
unsigned short	2	0 a 65.535
unsigned short int	2	0 a 65.535
long	4	-2.147.438.648 a 2.147.438.647
long int	4	-2.147.438.648 a 2.147.438.647
signed long	4	-2.147.438.648 a 2.147.438.647
signed long int	4	-2.147.438.648 a 2.147.438.647
unsigned long	4	0 a 4.294.967.295
unsigned long int	4	0 a 4.294.967.295
long double*	8	$2,2 \times 10^{-308}$ a $1,8 \times 10^{308}$

* Dependendo do compilador pode ser atribuído 8, 10 ou 12 bytes.

MODIFICADORES DE TIPOS



`unsigned char a = Ob01101110 // a é igual à 110`

`unsigned char b = Ob11101110 // b é igual à 238`

`signed char c = Ob01101110 // a é igual à 110`

`signed char b = Ob11101110 // b é igual à -110`

MODIFICADORES DE TIPOS

Valor	Base numérica
99	Decimal
099	Octal
0x99	Hexadecimal
Ob10011001	Binário

unsigned char a = Ob01101110 // a é igual à 110

unsigned char b = Ob11101110 // b é igual à 238

signed char c = Ob01101110 // a é igual à 110

signed char b = Ob11101110 // b é igual à -110

CONVERSÃO DE TIPOS

Deve ser utilizada para garantir o funcionamento correto quando se realiza operações com variáveis de diferentes tipos. Isto é, impedir que o resultado estoure a capacidade da variável que armazena o resultado, ou que a conversão automática seja incorreta resultando em uma conversão forçada (casting), devolvendo valores incorretos.



```
1
2 float conversao = 10.5;
3 int inteiro = (int)conversao; // inteiro = 10
```

MODIFICADORES DE ESCOPO

Um modificador de escopo trata-se de um parâmetro opcional na declaração de uma variável que permite alterar a área de existência da variável dentro do programa ou também definir seu local de criação:

auto, register, const, volatile, static, extern

auto: modificador de escopo automático para variáveis locais (declaradas dentro de funções). Pode ser omitido.

register: modificador que solicita ao microcontrolador o armazenamento da variável dentro de um registrador. Não pode ser utilizado na declaração de variáveis globais.

const: modificador que concede a variável a inalterabilidade de seu valor, isto é, qualquer tentativa de mudar o valor de uma variável declarada com “const” resultará em erro de compilação.

MODIFICADORES DE ESCOPO

Um modificador de escopo trata-se de um parâmetro opcional na declaração de uma variável que permite alterar a área de existência da variável dentro do programa ou também definir seu local de criação:

auto, register, const, volatile, static, extern

volatile: modificador que permite que uma variável seja alterável sem o conhecimento do programa principal, ainda que seja declarada dentro do seu escopo (utilizada para impedir otimizações errôneas do compilador).

static: modificador utilizado para tornar uma variável global conhecida apenas no arquivo no qual ela foi declarada e para manter inalterado o valor de uma variável local entre as chamadas de função.

extern: modificador utilizado para indicar ao compilador que a variável ou função foi declarada em outro arquivo, indicando a necessidade de um “linker” (possibilita incorporar funções que não possuem correspondência na “linguagem C”, como algumas em “assembly”).

ESCREVENDO OPERAÇÕES

Page 25

1- Existem pelo menos 3 opções quanto a sintaxe para escrita de operações em C (exemplo na imagem).

2- A posição do operador (“pré” ou “pós” fixado) interfere diretamente no comportamento da variável envolvida na operação (exemplo na imagem).

```
1 #include <stdio.h>
2
3 int main() {
4     int a = 5, b = 10;
5
6     int soma = a + b; //Escrita completa
7
8     a += b; //Escrita compacta equivalente a operação escrita acima
9
10    // Escrita utilizando operador ternário (alternativa ao if/else):
11    int maior = (a > soma) ? a : soma; /*se a condição entre parenteses for verdadeira
recebe o primeiro parâmetro após (?), senão o segundo*/
12
13    printf("Maior valor: %d\n", maior); //será impresso 15
14
15    // Operador pré-fixado
16    int x = ++a; // Incrementa primeiro, depois atribui
17
18    // Operador pós-fixado
19    int y = b++; // Atribui primeiro, depois incrementa
20
21
22    printf("a: %d, x: %d\n", a, x); // Imprime a: 6, x: 6
23    printf("b: %d, y: %d\n", b, y); // Imprime b: 11, y: 10
24
25    return 0;
26 }
```

OPERADORES ARITIMÉTICOS

Os operadores aritméticos da linguagem C são utilizados para realizar operações matemáticas básicas em variáveis e valores. Eles incluem:

- Adição (+): Soma de dois operandos.
- Exemplo: $a + b$
- Subtração (-): Subtração de um operando pelo outro.
- Exemplo: $a - b$
- Multiplicação (*): Multiplica dois operandos.
- Exemplo: $a * b$
- Divisão (/): Divide um operando pelo outro.
- Exemplo: a / b (se ambos forem inteiros, o resultado será truncado).
- Módulo (%): Retorna o resto da divisão entre dois operandos inteiros.
- Exemplo: $a \% b$

A = 10

B=20

Page 26

Operador	Descrição	Exemplo	Resultado
+	soma	A + B	30
-	subtração	A - B	-10
*	multiplicação	A * B	200
/	divisão inteira	B / A	2
%	módulo	B % A	0
++	incremento	A++	11
--	decremento	A--	9

```
● ● ●  
1 #include <stdio.h>  
2 int main() {  
3     int a = 10, b = 3;  
4     printf("Soma: %d\n", a + b);           // Resultado: 13  
5     printf("Subtração: %d\n", a - b);       // Resultado: 7  
6     printf("Multiplicação: %d\n", a * b);   // Resultado: 30  
7     printf("Divisão: %d\n", a / b);         // Resultado: 3  
8     printf("Módulo: %d\n", a % b);          // Resultado: 1  
9  
10 }
```

ESTRUTURAS CONDICIONAIS

Estruturas condicionais em C são utilizadas para executar um comando ou bloco de comandos, dependendo de uma condição estabelecida.

Para executar uma estrutura condicional em C, é necessário utilizar o comando IF, que exige uma condição que pode ser verdadeira ou falsa.

Existem dois tipos de estruturas condicionais: simples e composta. A estrutura condicional simples executa um comando ou vários comandos se a condição for verdadeira. A estrutura condicional composta executa um bloco de instruções se a condição for verdadeira e outro se a condição for falsa.

```
IF (CONDIÇÃO1){  
    // BLOCO DE CÓDIGO PARA CONDIÇÃO1  
}  
ELSE IF (CONDIÇÃO2){  
    // BLOCO DE CÓDIGO PARA CONDIÇÃO2  
}  
ELSE{  
    // BLOCO DE CÓDIGO SE NENHUMA CONDIÇÃO FOR VERDADEIRA  
}
```

EXISTE OUTRO TIPO:

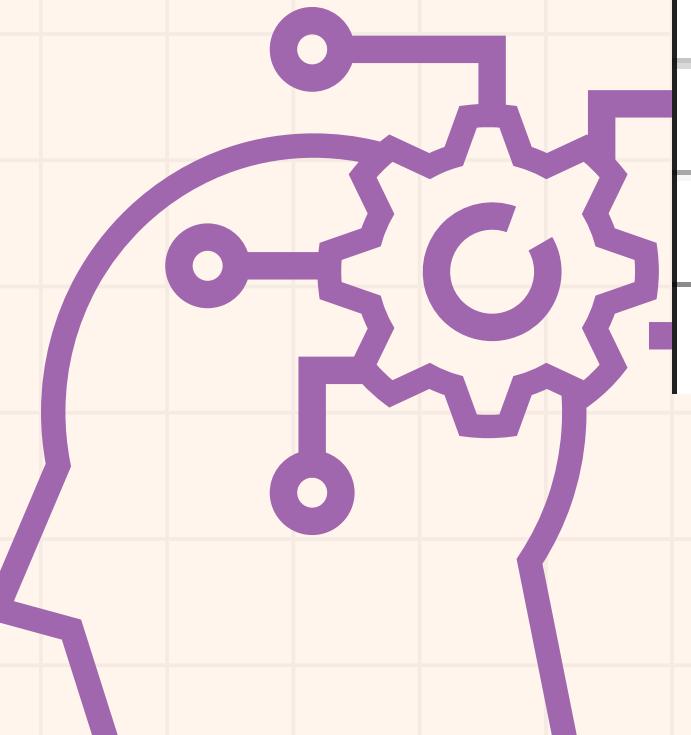
CONDIÇÃO ? EXPRESSÃO1:EXPRESSÃO2;



OPERADORES CONDICIONAIS & LÓGICOS

Os operadores lógicos em C são utilizados para combinar ou inverter condições em expressões booleanas. Eles retornam valores verdadeiro (1) ou falso (0).

Os operadores condicionais são usados para comparar valores. Eles retornam um valor booleano (0 ou 1).



Operadores Relacionais e Lógicos

Tipo	Operador	Sintaxe
Menor que	<	a < b
Maior que	>	a > b
Menor ou igual que	<=	a <= b
Maior ou igual que	>=	a >= b
Igual a	==	a == b
Diferente de	!=	a != b
Não (Lógico)	!	!a
E (Lógico)	&&	a && b
Ou (Lógico)		a b

OPERADORES CONDICIONAIS & LÓGICOS

```
1 #include <stdio.h>
2 int main(){
3     int a=10, b=3;
4
5     if(a > b){
6         printf("A é maior que B\n");
7     }
8
9     if(a<b){
10        printf("A é menor que B\n");
11    }
12
13    if(a<=b){
14        printf("A é menor ou igual a B\n");
15    }
16
17    if(a>=b){
18        printf("A é maior ou igual a B\n");
19    }
20
21    if(a==b){
22        printf("A é igual a B\n");
23    }
24
25    if(a!=b){
26        printf("A é diferente de B\n");
27    }
28
29 //não, e, ou logico
30 if(a>b && a<b){
31     printf("A é maior que B e menor que B\n");
32 }
33
34 if(a>b || a<b){
35     printf("A é maior que B ou menor que B\n");
36 }
37
38
```

OPERADORES CONDICIONAIS & LÓGICOS

```
#include <stdio.h>

int main() {
    int a = 5, b = 3; // Valores iniciais de exemplo

    // Operador OR bit a bit (Forma compacta)
    a |= b; // Equivale a: a = a | b;
    printf("Resultado de a |= b: %d\n", a);

    // Redefinindo o valor de 'a' para o proximo exemplo
    a = 5;

    // Operador XOR bit a bit (Forma compacta)
    a ^= b; // Equivale a: a = a ^ b;
    printf("Resultado de a ^= b: %d\n", a);

    // Redefinindo o valor de 'a' para o proximo exemplo
    a = 5;

    // Operador shift a esquerda (Forma compacta)
    a <<= b; // Equivale a: a = a << b;
    printf("Resultado de a <<= b: %d\n", a);

    // Redefinindo o valor de 'a' para o proximo exemplo
    a = 5;

    // Operador shift a direita (Forma compacta)
    a >>= b; // Equivale a: a = a >> b;
    printf("Resultado de a >>= b: %d\n", a);

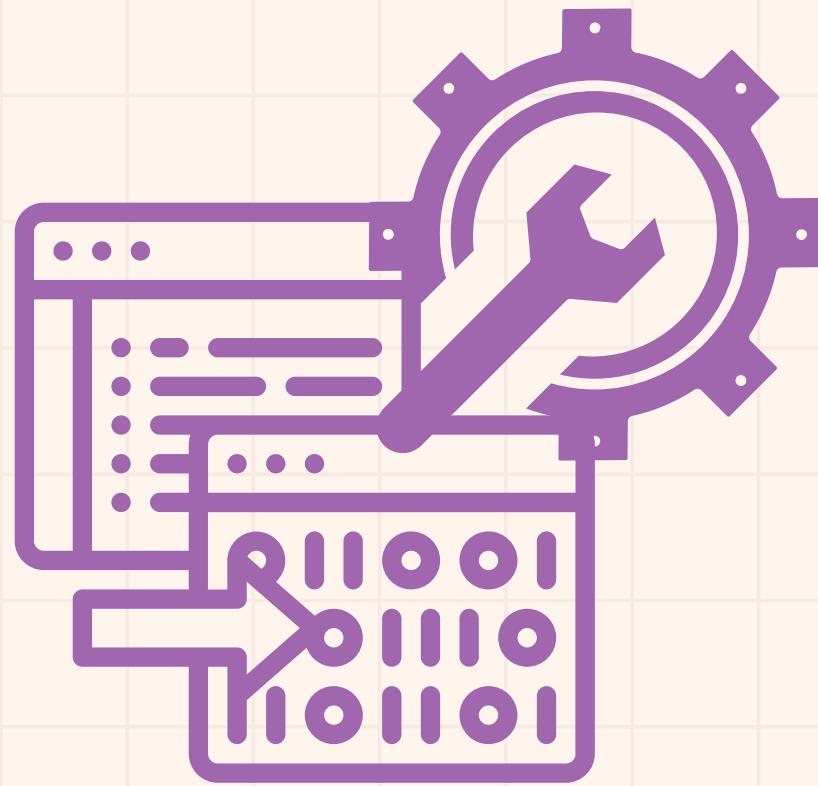
    return 0;
}
```

DIRETIVAS DE PRÉ COMPILAÇÃO

Conceitualmente, a linguagem C é baseada em blocos de construção. Assim sendo, um programa em C nada mais é que um conjunto de funções básicas ordenadas pelo programador. Algumas instruções não fazem parte do conjunto de palavras padrão da linguagem C. Essas palavras desconhecidas são blocos de instruções ordenados de forma a realizar um determinado processamento.

A diretiva **#include** é certamente a diretiva de pré-compilação mais usada na linguagem C. Essa diretiva copia o conteúdo de um arquivo àquele ponto do código-fonte.

```
#include  
#include "nome_do_arquivo.extensão"
```



```
#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
    float numero1, numero2, resultado;  
    int escolha;
```

DIRETIVAS DE PRÉ COMPILAÇÃO

Pré-processamento

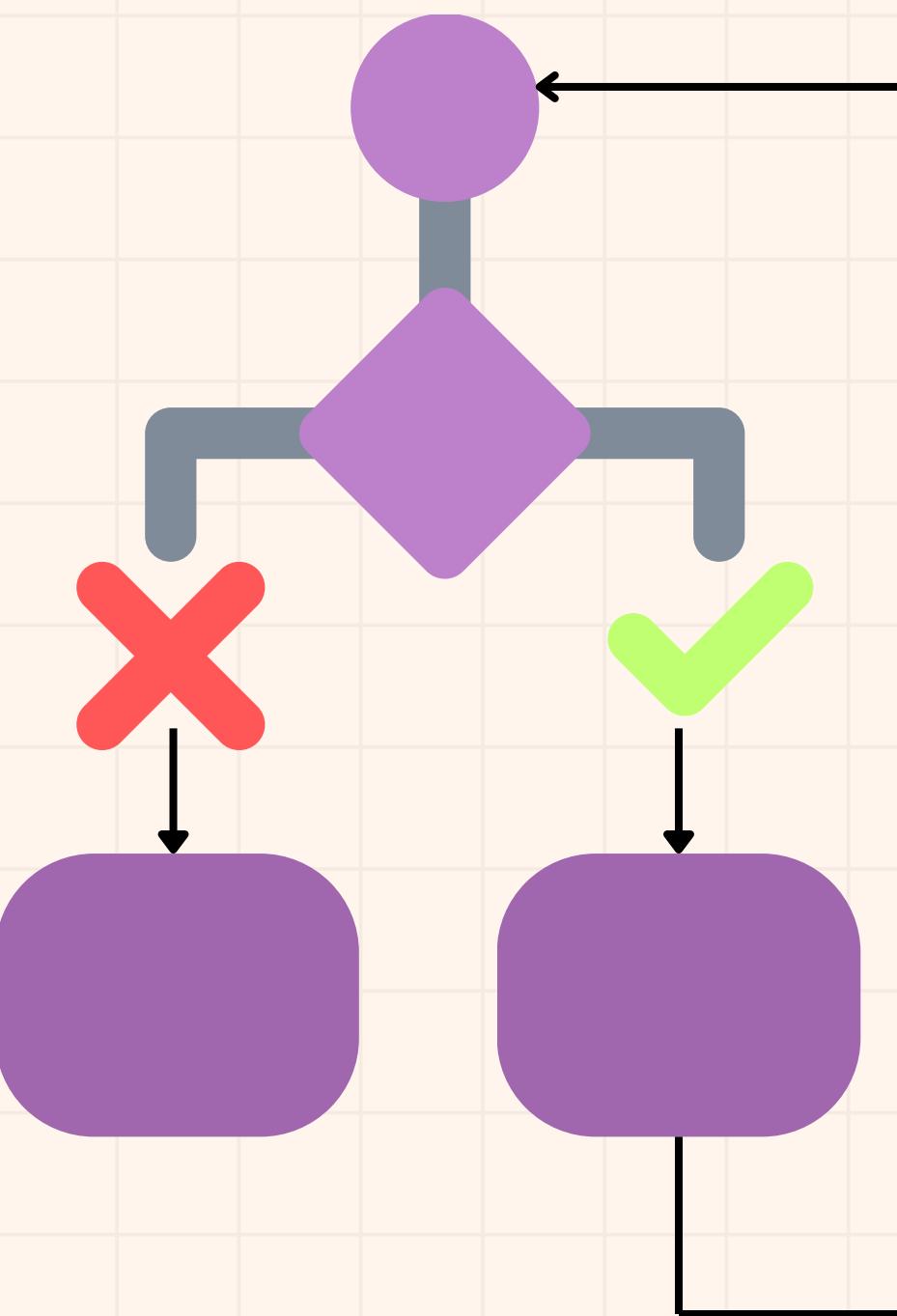
- O pré-processador executa as diretivas de pré-compilação, como **#include**, **#define**, **#ifdef**, entre outras.
- Substitui macros, processa inclusões de arquivos de cabeçalho, e realiza exclusões/inclusões condicionais de código.
- O resultado é um arquivo de código fonte expandido, sem diretivas de pré-processamento.

```
#include <nome_do_arquivo.extensão>
#include "nome_do_arquivo.extensão"
#include <iostream>
#define nome_do_simbolo
#define nome_da_constante seu_valor
#define nome_da_macro(parâmetros) expressao_equivalente
#undef nome_do_simbolo
#undef nome_da_constante
#undef nome_da_macro
#if expressão
    código 1
#else
    código 2
#endif
#endif
#ifdef símbolo
    código
#endif
#endif
#ifndef símbolo
    código
#endif
#endif
```

ESTRUTURA DE REPETIÇÃO

As estruturas de repetição na linguagem C permitem executar blocos de código múltiplas vezes com base em uma condição, otimizando tarefas repetitivas.

- O **for** é usado quando o número de iterações é conhecido, com inicialização, condição e incremento declarados em uma única linha.
- O **while** executa o bloco enquanto a condição especificada for verdadeira, sendo útil para repetições com condições dinâmicas.
- **Do-while** garante que o bloco de código seja executado pelo menos uma vez, pois a condição é avaliada após a execução. Essas estruturas são essenciais para automação de processos e manipulação de coleções, permitindo maior eficiência e legibilidade no desenvolvimento.



ESTRUTURA DE REPETIÇÃO

```
// Configuração do pino do LED  
int ledPin = 13; // Pino digital ao qual o LED está conectado
```

```
void setup() {  
    // Define o pino como saída  
    pinMode(ledPin, OUTPUT);  
}
```

```
void loop() {  
    // Liga o LED  
    digitalWrite(ledPin, HIGH);  
    delay(1000); // Aguarda 1 segundo (1000 ms)  
  
    // Desliga o LED  
    digitalWrite(ledPin, LOW);  
    delay(1000); // Aguarda 1 segundo (1000 ms)  
}
```

```
/*  
 * File: main.c  
 * Author: Djmz2  
 * Created on 14 de Novembro de 2024, 18:04  
 */  
  
#include <xc.h>  
  
// Configuração do oscilador (defina a frequência do cristal)  
#define _XTAL_FREQ 4000000 // 4 MHz (ajuste se necessário)  
  
void main(void) {  
    // Limpar a porta C  
    PORTC = 0x00;  
    CMCON = 0x07;  
  
    // Configurar RC0 como saída  
    TRISCbits.TRISCO = 0;  
    PORTCbits.RC0 = 1;  
    while (1) {  
        // Acender o LED (RC0 = 1)  
        PORTCbits.RC0 = 1;  
        __delay_ms(1000); // Aguardar 1 segundo  
  
        // Apagar o LED (RC0 = 0)  
        PORTCbits.RC0 = 0;  
        __delay_ms(1000); // Aguardar 1 segundo  
    }  
}
```

OPERADORES DE ATRIBUIÇÃO

Os operadores de atribuição em C são usados para atribuir o valor do operando do lado direito a uma variável, propriedade ou elemento indexador fornecido pelo operando do lado esquerdo. O operador de atribuição simples é o (=).

Operador	Finalidade	Exemplo	Resultado	O mesmo que:
=	Atribuição	a = b	A é 5	a = b
+=	Adição	a += b	10	a = a+b
-=	Subtração	a -= b	5	a = a-b
*=	Multiplicação	a *= b	25	a = a*b
/=	Divisão (Quociente)	a /= b	5	a = a/b
%=	Divisão (Resto)	a %= b	0	a = a%b

```

● ● ●

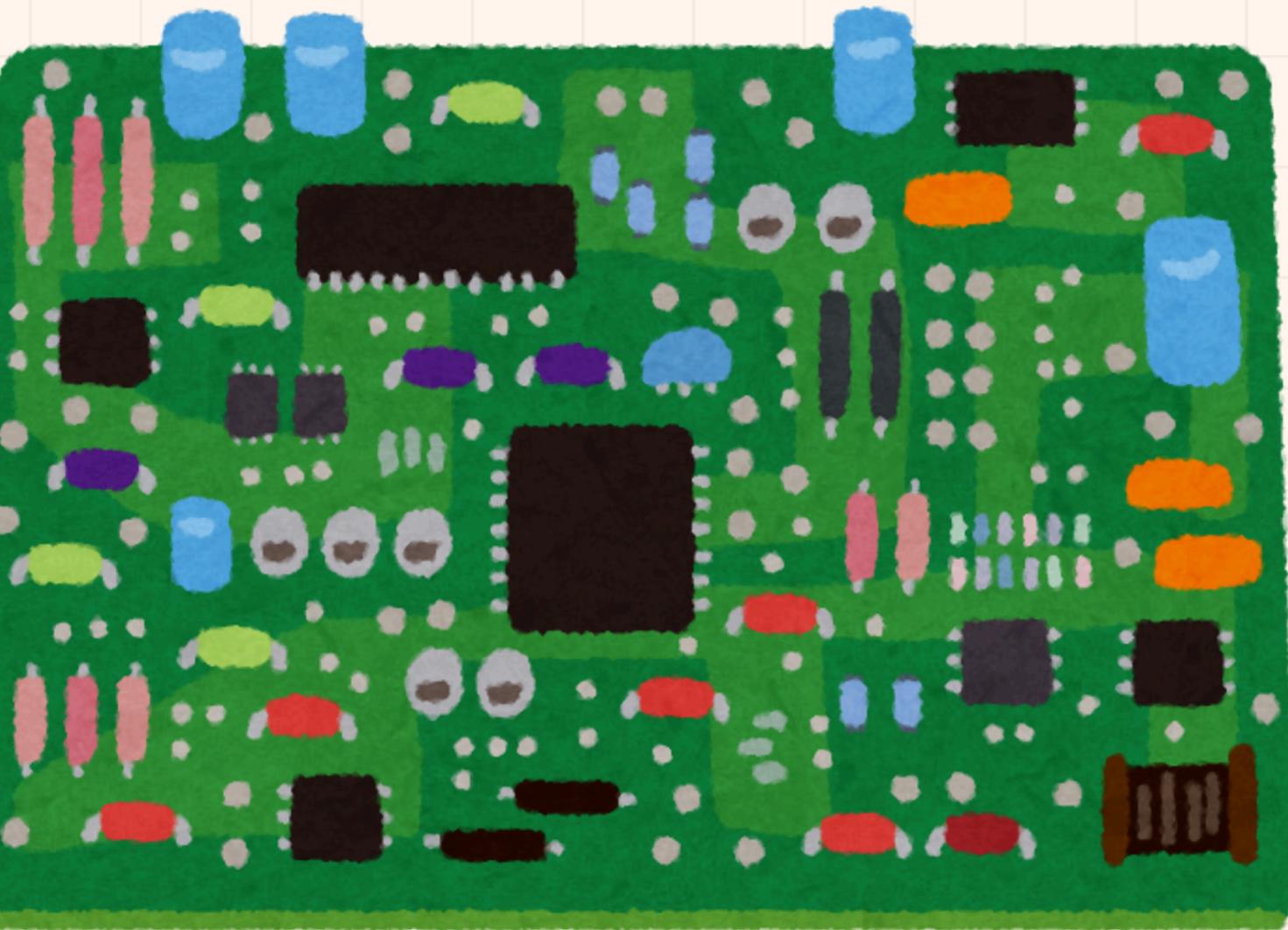
1 int a = 10, b = 3;
2 a += b; // a = a + b
3 printf("Atribuição de soma: %d\n", a); // Resultado: 13
4
5 a = 10;
6 a -= b; // a = a - b
7 printf("Atribuição de subtração: %d\n", a); // Resultado: 7
8
9 a = 10;
10 a *= b; // a = a * b
11 printf("Atribuição de multiplicação: %d\n", a); // Resultado: 30
12
13 a = 10;
14 a /= b; // a = a / b
15 printf("Atribuição de divisão: %d\n", a); // Resultado: 3
16
17 a = 10;
18 a %= b; // a = a % b
19 printf("Atribuição de módulo: %d\n", a); // Resultado: 1
20

```

DATASHEETS & CIRCUITOS

Quando se trata de sistemas embarcados e microcontroladores é muito importante ter a experiência com datasheets e os circuitos mais simples.

Nos exemplos de datasheets do PIC16f630 e do DHT11 podemos entender aspectos fundamentais de seu funcionamento. E no que se trata de programação, podemos ver, de forma antecipada, o que o Micro escolhido tem a nos oferecer, além de ver como um sensor qualquer faz a comunicação.

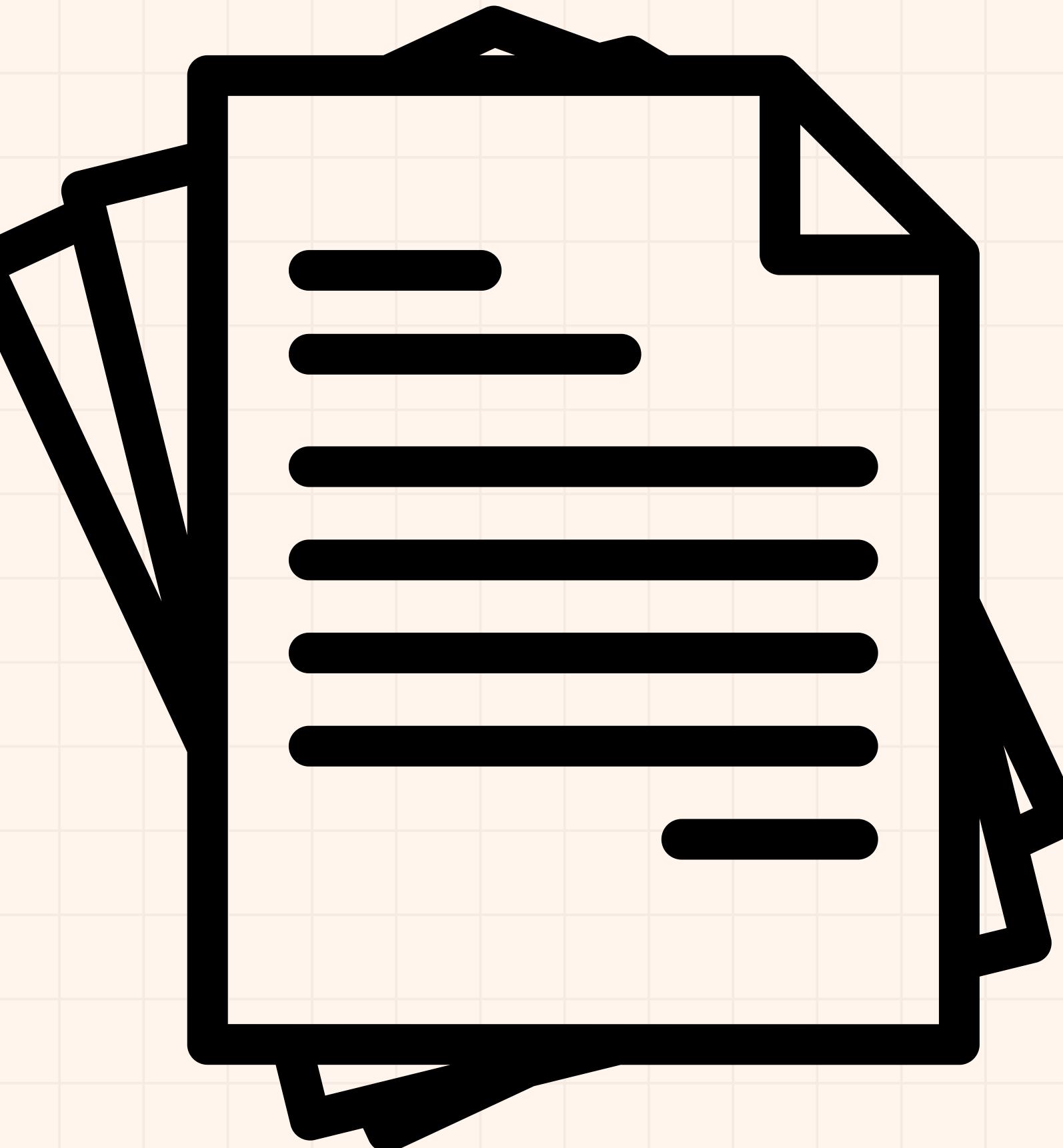


DATASHEETS & CIRCUITOS

O que é um Datasheet?

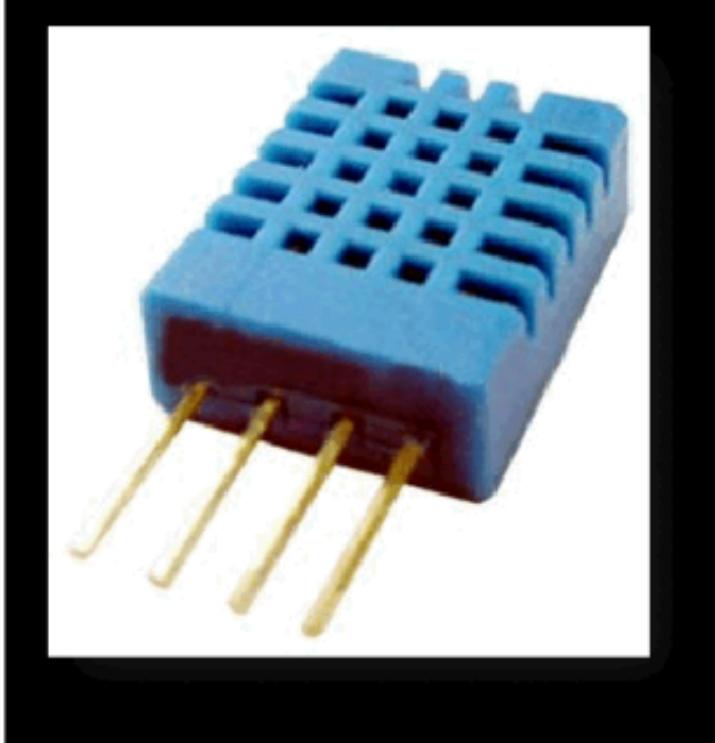
Um datasheet, folha de dados ou folha de especificações é um documento que resume o desempenho e outras características técnicas de um produto, máquina, componente (por exemplo, um componente eletrônico), material, subsistema (por exemplo, fonte de alimentação) ou software em detalhe suficiente para que possa ser usado por um engenheiro de projeto para integrar o componente em um sistema.

wikipédia



É um documento que reúne informações sobre as características, especificações, funcionamento e aplicação de um produto,

DATASHEETS & CIRCUITOS



Each DHT11 element is strictly calibrated in the laboratory that is extremely accurate on humidity calibration. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. The single-wire serial interface makes system integration quick and easy. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 4-pin single row pin package. It is convenient to connect and special packages can be provided according to users' request.

2. Technical Specifications:

Overview:

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 °C	±5%RH	±2°C	1	4 Pin Single Row

3. Typical Application (Figure 1)

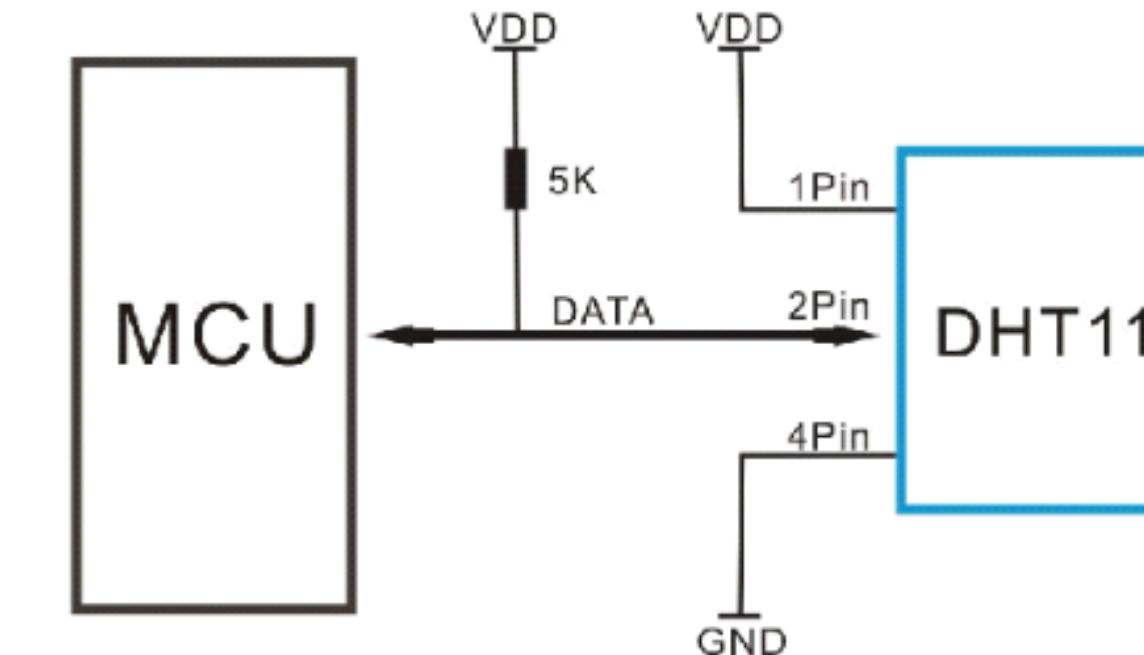


Figure 1 Typical Application

Note: 3Pin – Null; MCU = Micro-computer Unite or single chip Computer

When the connecting cable is shorter than 20 metres, a 5K pull-up resistor is recommended; when the connecting cable is longer than 20 metres, choose a appropriate pull-up resistor as needed.

4. Power and Pin

DHT11's power supply is 3-5.5V DC. When power is supplied to the sensor, do not send any instruction to the sensor in within one second in order to pass the unstable status. One capacitor valued 100nF can be added between VDD and GND for power filtering.

5. Communication Process: Serial Interface (Single-Wire Two-Way)

Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms.

Data consists of decimal and integral parts. A complete data transmission is 40bit, and the sensor sends higher data bit first.

Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

DATASHEETS & CIRCUITOS

5.1 Overall Communication Process (Figure 2, below)

When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.

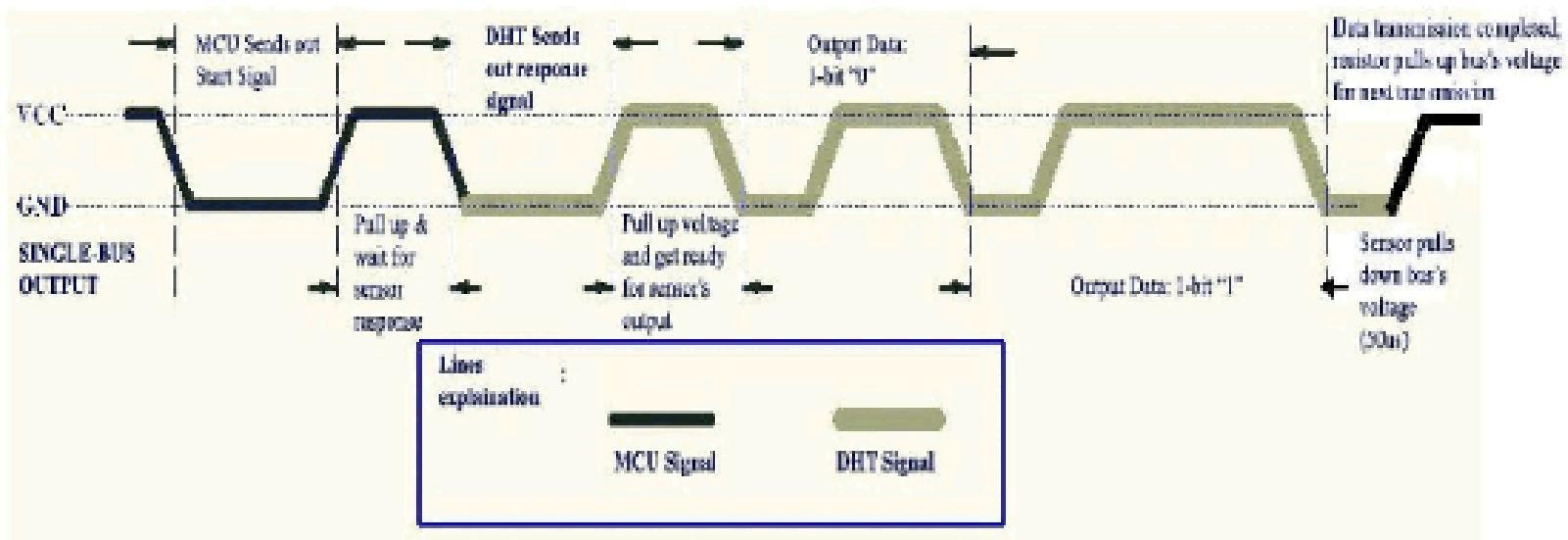


Figure 2 Overall Communication Process

5.2 MCU Sends out Start Signal to DHT (Figure 3, below)

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.

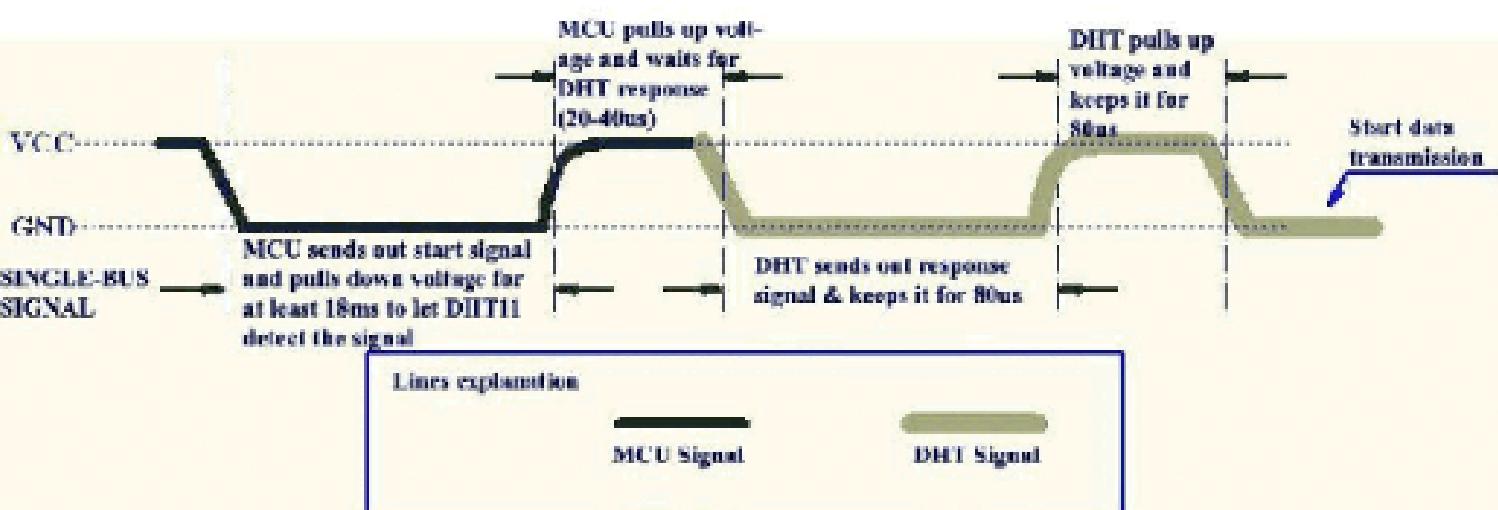


Figure 3 MCU Sends out Start Signal & DHT Responses

DATASHEETS & CIRCUITOS

5.1 Overall Communication Process (Figure 2, below)

When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.

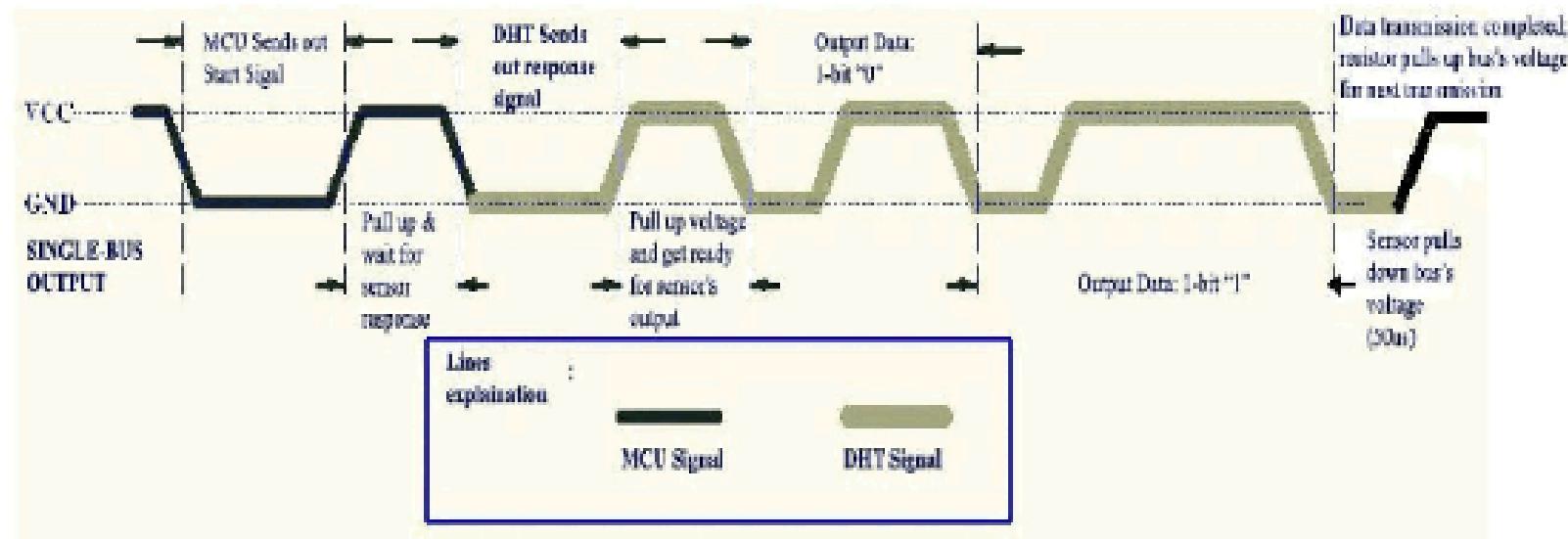


Figure 2 Overall Communication Process

5.2 MCU Sends out Start Signal to DHT (Figure 3, below)

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40μs for DHT's response.

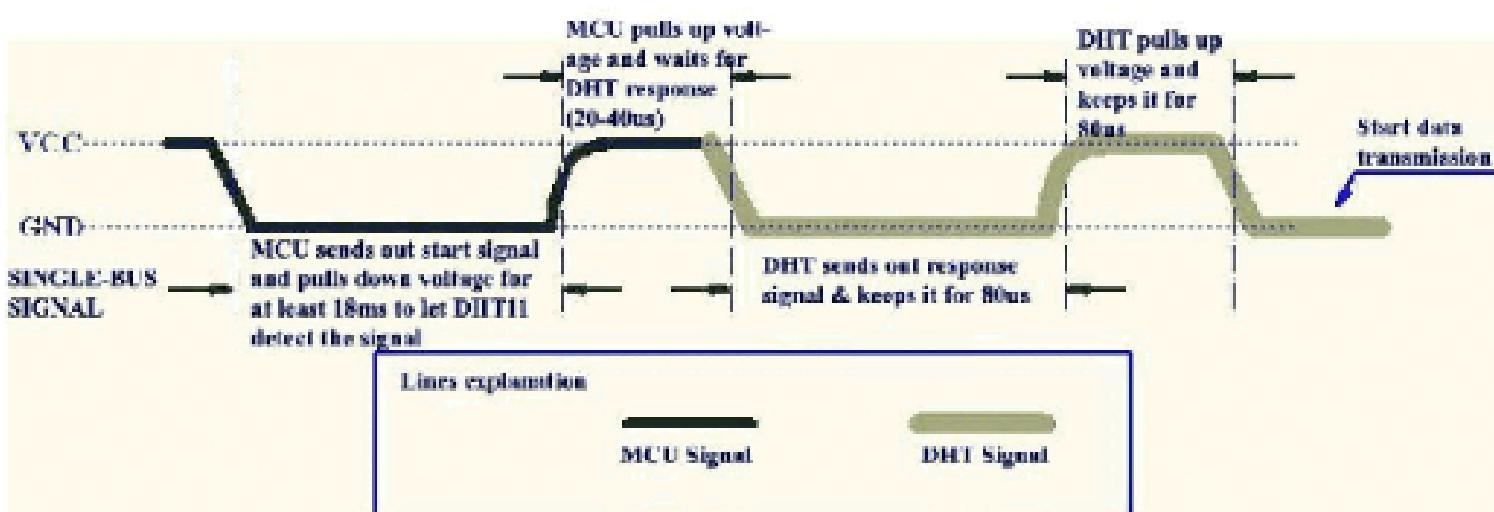


Figure 3 MCU Sends out Start Signal & DHT Responses

DATASHEETS & CIRCUITOS



PIC16F630/676

14-Pin FLASH-Based 8-Bit CMOS Microcontroller

High Performance RISC CPU:

- Only 35 instructions to learn
 - All single cycle instructions except branches
- Operating speed:
 - DC - 20 MHz oscillator/clock input
 - DC - 200 ns instruction cycle
- Interrupt capability
- 8-level deep hardware stack
- Direct, Indirect, and Relative Addressing modes

Low Power Features:

- Standby Current:
 - 1 nA @ 2.0V, typical
- Operating Current:
 - 8.5 μ A @ 32 kHz, 2.0V, typical
 - 100 μ A @ 1 MHz, 2.0V, typical
- Watchdog Timer Current:
 - 300 nA @ 2.0V, typical
- Timer1 oscillator current:
 - 4 μ A @ 32 kHz, 2.0V, typical

Special Microcontroller Features:

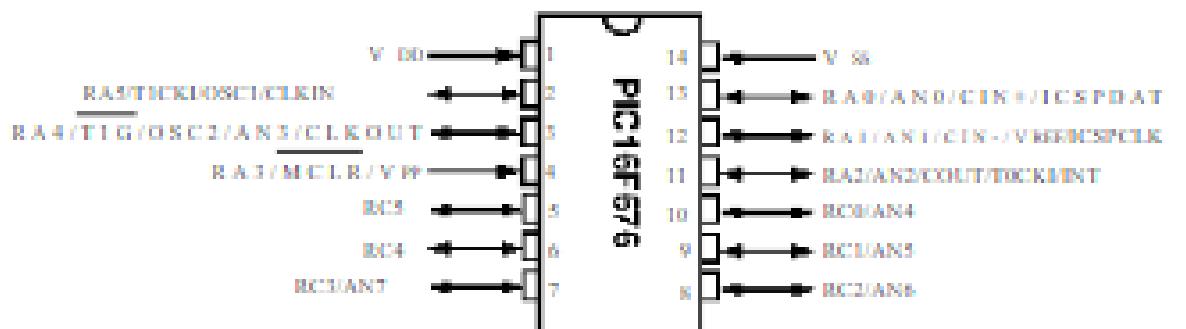
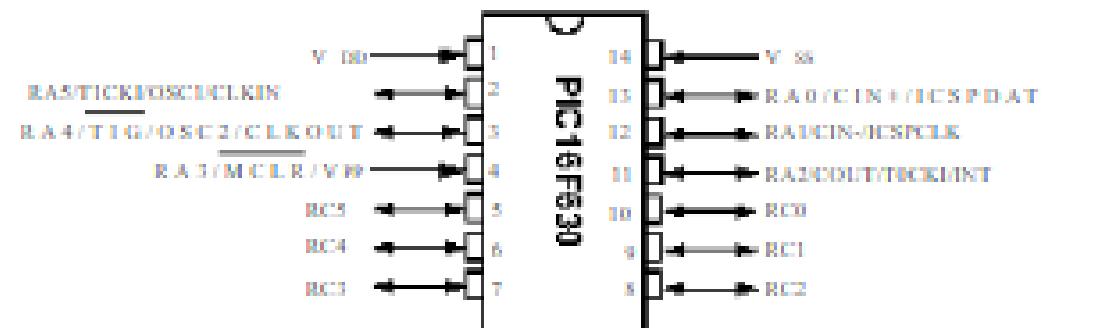
- Internal and external oscillator options
 - Precision Internal 4 MHz oscillator factory calibrated to $\pm 1\%$
 - External Oscillator support for crystals and resonators
 - 5 μ s wake-up from SLEEP, 3.0V, typical
- Power saving SLEEP mode
- Wide operating voltage range - 2.0V to 5.5V
- Industrial and Extended temperature range
- Low power Power-on Reset (POR)
- Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Brown-out Detect (BOD)
- Watchdog Timer (WDT) with independent oscillator for reliable operation
- Multiplexed MCLR/Input-pin
- Interrupt-on-pin change
- Individual programmable weak pull-ups
- Programmable code protection
- High Endurance FLASH/EEPROM Cell
 - 100,000 write FLASH endurance
 - 1,000,000 write EEPROM endurance
 - FLASH/Data EEPROM Retention: > 40 years

Peripheral Features:

- 12 I/O pins with individual direction control
- High current sink/source for direct LED drive
- Analog comparator module with:
 - One analog comparator
 - Programmable on-chip comparator voltage reference (CV REF) module
 - Programmable input multiplexing from device inputs
 - Comparator output is externally accessible
- Analog-to-Digital Converter module (PIC16F676):
 - 10-bit resolution
 - Programmable 8-channel input
 - Voltage reference input
- Timer0: 8-bit timer/counter with 8-bit programmable prescaler
- Enhanced Timer1:
 - 16-bit timer/counter with prescaler
 - External Gate Input mode
 - Option to use OSC1 and OSC2 in LP mode as Timer1 oscillator, if INTOSC mode selected
- In-Circuit Serial ProgrammingTM(ICSPTM) via two pins

Pin Diagrams

14-pin PDIP, SOIC, TSSOP



Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	Comparators	Timers 8/16-bit
	FLASH (words)	S R A M (bytes)	EEPROM (bytes)				
PIC16F630	1024	64	128	12	-	1	1/1
PIC16F676	1024	64	128	12	8	1	1/1

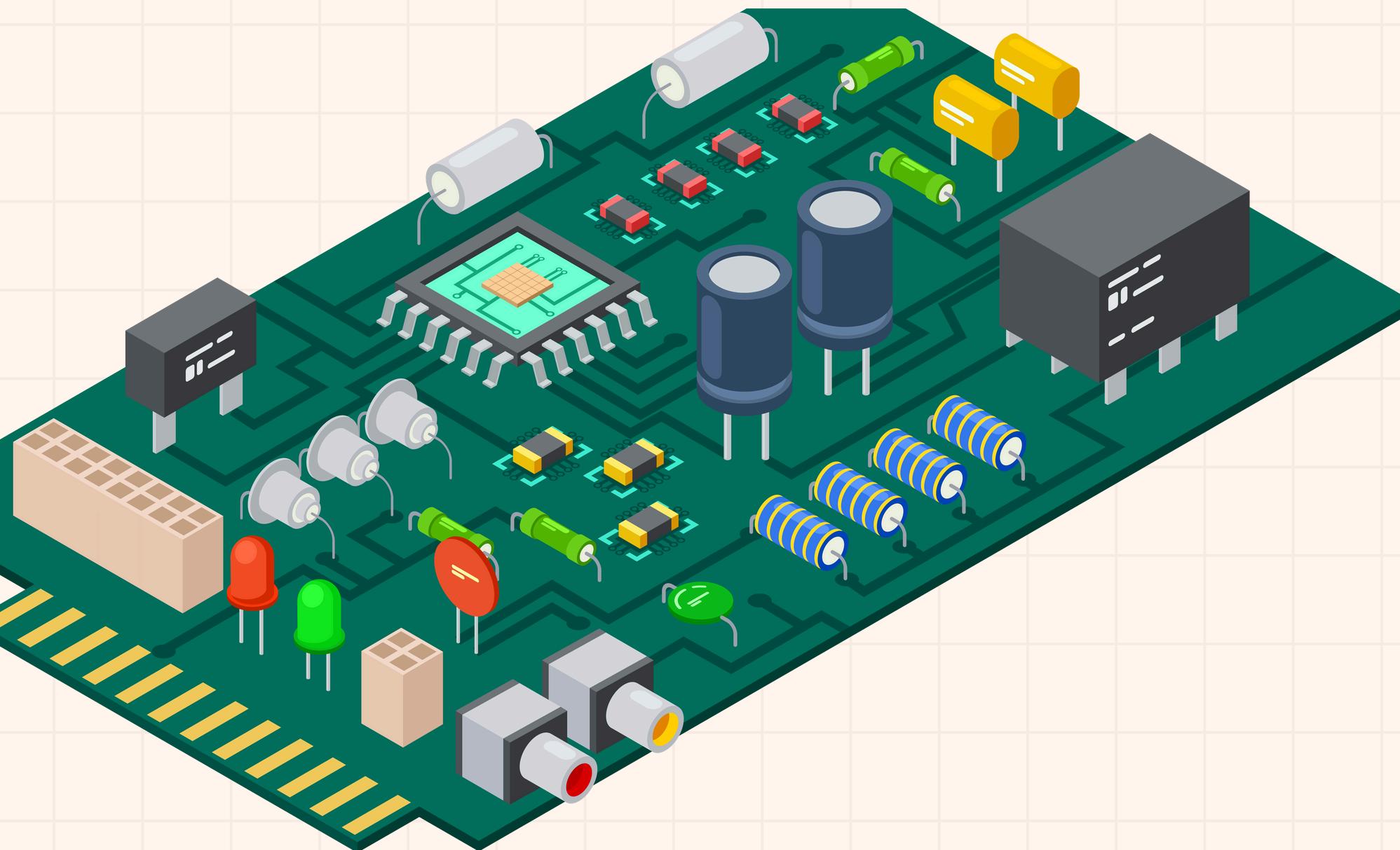
DATASHEETS & CIRCUITOS

RLF	Rotate Left f through Carry	SLEEP
Syntax:	[label] RLF f,d	Syntax: [label] SLEEP
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$	Operands: None
Operation:	See description below	Operation: $00h \rightarrow WDT,$ $0 \rightarrow WDT \text{ prescaler},$ $1 \rightarrow \overline{TO},$ $0 \rightarrow \overline{PD}$
Status Affected:	C	Status Affected: $\overline{TO}, \overline{PD}$
Description:	The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is stored back in register 'f'.	Description: The power-down STATUS bit, PD is cleared. Time-out STATUS bit, TO is set. Watchdog Timer and its prescaler are cleared. The processor is put into SLEEP mode with the oscillator stopped.
		
RETURN	Return from Subroutine	SUBLW
Syntax:	[label] RETURN	Syntax: [label] SUBLW k
Operands:	None	Operands: $0 \leq k \leq 255$
Operation:	$TOS \rightarrow PC$	Operation: $k - (W) \rightarrow (W)$
Status Affected:	None	Status Affected: C, DC, Z
Description:	Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two-cycle instruction.	Description: The W register is subtracted (2's complement method) from the eight-bit literal 'k'. The result is placed in the W register.
RRF	Rotate Right f through Carry	SUBWF
Syntax:	[label] RRF f,d	Syntax: [label] SUBWF f,d
Operands:	$0 \leq f \leq 127$ $d \in [0,1]$	Operands: $0 \leq f \leq 127$ $d \in [0,1]$
Operation:	See description below	Operation: $(f) - (W) \rightarrow (destination)$
Status Affected:	C	Status Affected: C, DC, Z
Description:	The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.	Description: Subtract (2's complement method) W register from register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.
		

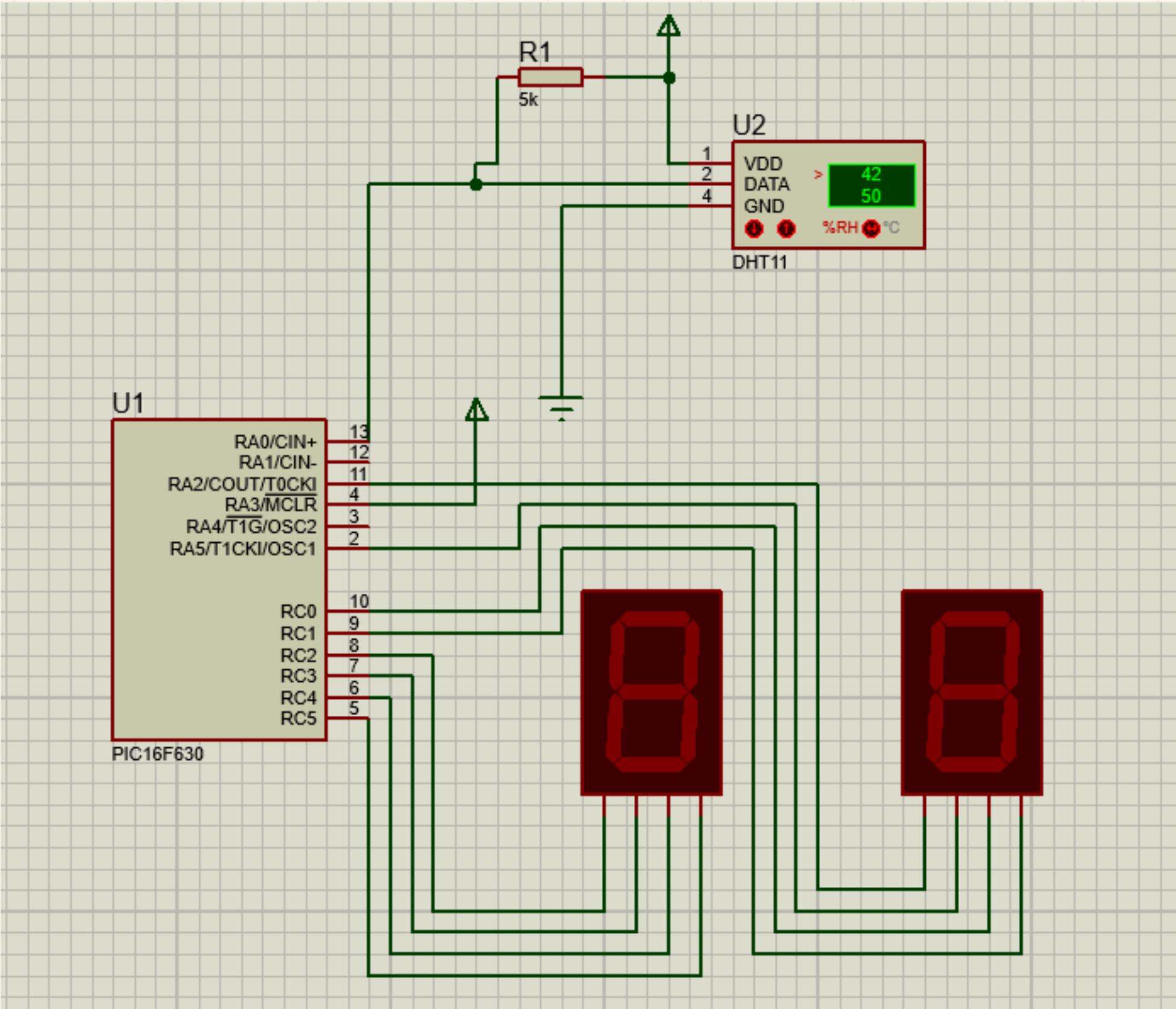
DATASHEETS & CIRCUITOS

O que é um circuito?

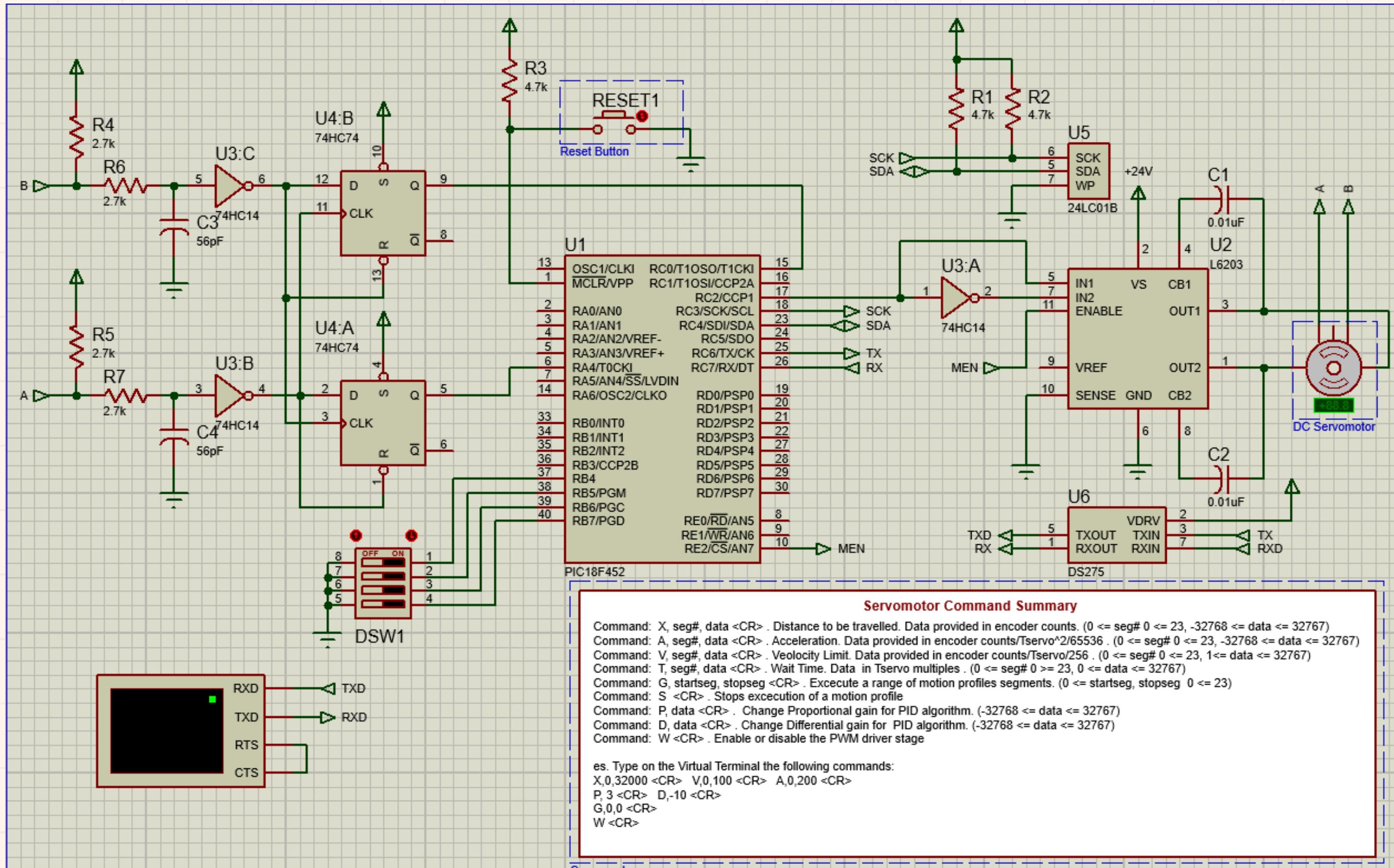
Um circuito elétrico é um caminho fechado por onde a corrente elétrica circula, formado por componentes elétricos interligados, como fios, resistores, capacitores, indutores, diodos, transistores e fontes de energia



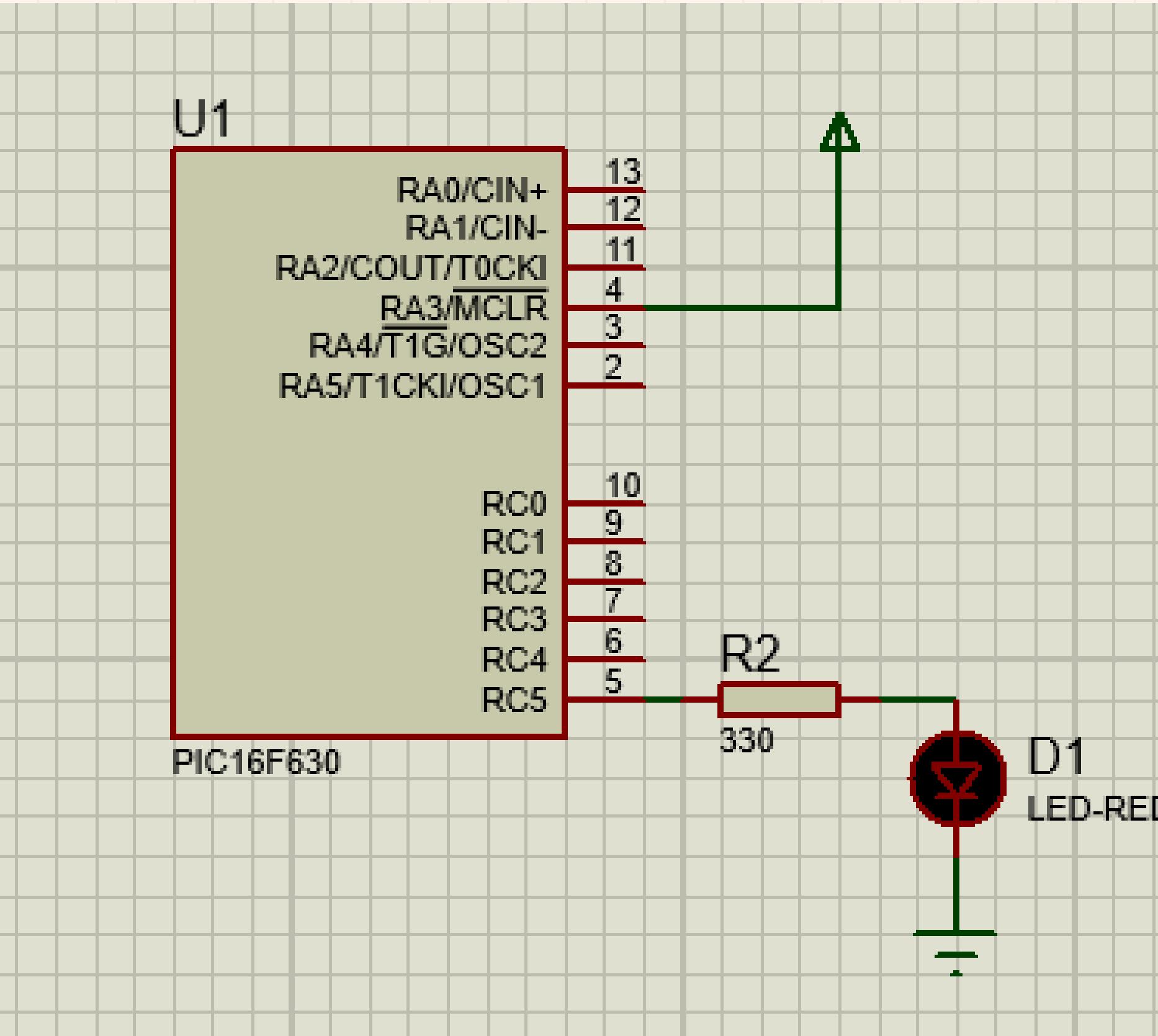
DATASHEETS & CIRCUITOS



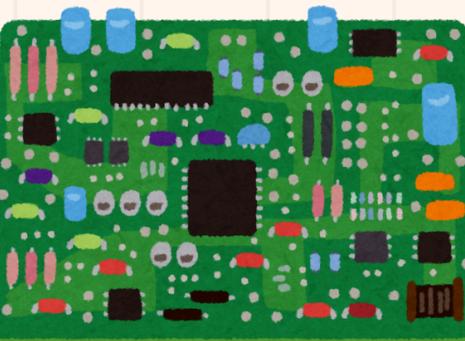
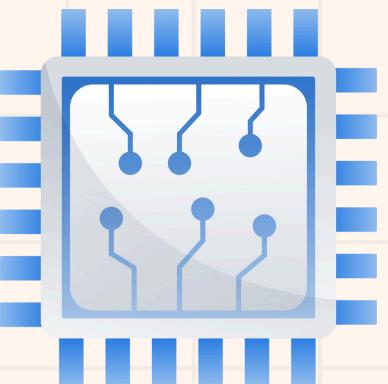
DATASHEETS & CIRCUITOS



DATASHEETS & CIRCUITOS



REGISTRADORES



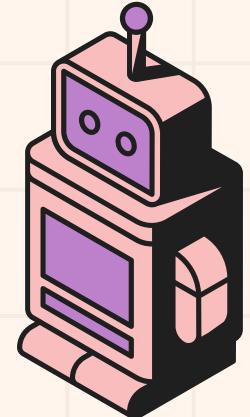
Exemplos de registradores:

1. **PCL (Program Counter Low)**: Armazena os 8 bits menos significativos do contador de programa, que indica a próxima instrução a ser executada.
2. **PCH (Program Counter High)**: Armazena os 8 bits mais significativos do contador de programa.
3. **STATUS**: Contém flags de controle, incluindo o Carry bit (C), o Zero bit (Z), e outros que indicam o estado do microcontrolador durante operações aritméticas e de controle.
4. **PORTA** e **PORTC**: Registradores de entrada/saída digitais, que permitem a leitura e escrita nos pinos de I/O.
5. **TRISA** e **TRISC**: Configuram os pinos de PORTA e PORTC como entradas ou saídas, controlando a direção do fluxo de dados.
6. **TMRO** e **TMR1**: Registradores de temporizador ou Timers, responsáveis por gerar atrasos temporais no microcontrolador.
7. **INTCON**: Controla o gerenciamento de interrupções, habilitando ou desabilitando as interrupções globais e de periféricos.
8. **OPTION_REG**: Configura o controle do temporizador, como o prescaler, e outros parâmetros de operação relacionados a temporização e interrupções.

Esses registradores são essenciais para o controle e operação do PIC16F630, permitindo o gerenciamento de fluxos de dados, temporização e interrupções.

File Address	File Address
Indirect addr. ⁽¹⁾	Indirect addr. ⁽¹⁾
00h	80h
01h	81h
PCL	PCL
02h	82h
STATUS	STATUS
03h	83h
FSR	FSR
04h	84h
PORTA	TRISA
05h	85h
PORTC	86h
06h	87h
07h	88h
08h	89h
09h	8Ah
PCLATH	PCLATH
INTCON	INTCON
PIR1	PIE1
0Ch	8Ch
0Dh	8Dh
TMR1L	PCON
TMR1H	8Eh
T1CON	8Fh
10h	90h
11h	ANSEL ⁽²⁾
12h	91h
13h	92h
14h	93h
15h	94h
WPUA	95h
16h	96h
IOCA	97h
17h	98h
CMCON	VRCON
18h	EEDAT
19h	99h
1Ah	EEADR
1Bh	9Ah
1Ch	EECON1
1Dh	9Bh
EECON2 ⁽¹⁾	9Ch
1Eh	9Dh
ADRESH ⁽²⁾	9Eh
1Fh	9Fh
ADCON0 ⁽²⁾	A0h
ADCON1 ⁽²⁾	
General Purpose Registers	accesses 20h-5Fh
64 Bytes	
5Fh	DFh
60h	E0h
7Fh	FFh

■ Unimplemented data memory locations, read as '0'.
 1: Not a physical register.
 2: PIC16F676 only.



REGISTRADORES

No seguinte trecho de código pode-se observar o uso de alguns dos registradores citados anteriormente:

```
#include <xc.h>

// Configurações do PIC16F877A
#pragma config FOSC = HS           // Oscilador externo de alta velocidade
#pragma config WDTE = OFF          // Watchdog Timer desativado
#pragma config PWRTE = ON          // Power-up Timer ativado
#pragma config BOREN = ON          // Brown-out Reset ativado
#pragma config LVP = OFF           // Programação em baixa tensão desativada
#pragma config CPD = OFF           // Código de proteção para EEPROM desativado
#pragma config WRT = OFF           // Código de proteção de gravação desativado
#pragma config CP = OFF            // Código de proteção desativado

#define _XTAL_FREQ 20000000          // Frequência do oscilador de 20 MHz
#define PWM_FREQ 50                 // Frequência do PWM desejada (50 Hz)

void ADC_Init() {
    ADCON0 = 0x41;   // Canal AN0 selecionado, ADC ligado
    ADCON1 = 0x80;   // Configuração do ADC para VDD e GND como referências
}

unsigned int ADC_Read(unsigned char channel) {
    ADCON0 &= 0xC5;           // Limpa o canal atual
    ADCON0 |= (channel << 3); // Seleciona o canal desejado
    __delay_ms(2);            // Tempo para estabilização do capacitor de
    GO_nDONE = 1;              // Inicia a conversão
    while (GO_nDONE);         // Aguarda o término da conversão
    return ((ADRESH << 8) + ADRESL); // Retorna o valor de 10 bits
}
```

```
/*
 * File: main.c
 * Author: Djmz2
 * Created on 14 de Novembro de 2024, 18:04
 */

#include <xc.h>

// Configuração do oscilador (defina a frequência do cristal)
#define _XTAL_FREQ 4000000 // 4 MHz (ajuste se necessário)

void main(void) {
    // Limpar a porta C
    PORTC = 0x00;
    CMCON = 0x07;

    // Configurar RC0 como saída
    TRISCbits.TRISCO = 0;
    PORTCbits.RC0 = 1;

    while (1) {
        // Acender o LED (RC0 = 1)
        PORTCbits.RC0 = 1;
        __delay_ms(1000); // Aguardar 1 segundo

        // Apagar o LED (RC0 = 0)
        PORTCbits.RC0 = 0;
        __delay_ms(1000); // Aguardar 1 segundo
    }
}
```

REGISTRADORES

Observa-se a importância de uma boa análise no Datasheet do dispositivo para o bom uso dos registradores

PIC16F630/676

TABLE 2-2: PIC16F630/676 SPECIAL FUNCTION REGISTERS SUMMARY BANK 1

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOD	Page		
Bank 1													
80h	INDIF	Addressing this location uses contents of FSR to address data memory (not a physical register)											
81h	OPTION_REG	RAPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	14, 32		
82h	PCL	Program Counter's (PC) Least Significant Byte											
83h	STATUS	IRP ⁽²⁾	RP ₁ ⁽²⁾	RP0	TO	PD	Z	DC	C	0001 1xxx	13		
84h	FSR	Indirect data memory Address Pointer											
85h	TRISA	—	—	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	--11 1111	21		
86h	—	Unimplemented											
87h	TRISC	—	—	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	--11 1111	—		
88h	—	Unimplemented											
89h	—	Unimplemented											
8Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of program counter					---0 0000	19		
8Bh	INTCON	GIE	PEIE	TOIE	INTE	RAIE	TOIF	INTF	RAIF	0000 0000	15		
8Ch	PIE1	EEIE	ADIE	—	—	CMIE	—	—	TMR1IE	00-- 0--0	16		
8Dh	—	Unimplemented											
8Eh	PCON	—	—	—	—	—	—	POR	BOD	---- --gg	18		
8Fh	—	—											
90h	OSCCAL	CAL5	CA_4	CAL3	CAL2	CAL1	CAL0	—	—	1000 03--	18		
91h	ANSEL ⁽³⁾	ANS7	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0	1111 1111	48		
92h	—	Unimplemented											
93h	—	Unimplemented											
94h	—	Unimplemented											
95h	WPUA	—	—	WPUA5	WPUA4	—	WPUA2	WPUA1	WPUA0	--11 -111	22		
96h	IOCA	—	—	IOCA5	IOCA4	IOCA3	IOCA2	IOCA1	IOCA0	--03 0300	23		
97h	—	Unimplemented											
98h	—	Unimplemented											
99h	VRCON	VREN	—	VRR	—	VR3	VR2	VR1	VR0	0-0- 0000	44		
9Ah	EEDAT	EEPROM data register											
9Bh	EEADR	—	EEPROM address register										
9Ch	EECON1	—	—	—	WRERR	WREN	WR	RD	----	x000	52		
9Dh	EECON2	EEPROM control register 2 (not a physical register)											
9Eh	ADRESL ⁽³⁾	Least Significant 2 bits of the left shifted result or 8 bits of the right shifted result											
9Fh	ADCON1 ⁽³⁾	—	ADC-S2	ADC-S1	ADC-S0	—	—	—	—	-000 ----	47, 63		

Legend: — = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented

Note 1: Other (non Power-up) Resets include MCLR Reset, Brown-out Detect and Watchdog Timer Reset during normal operation.

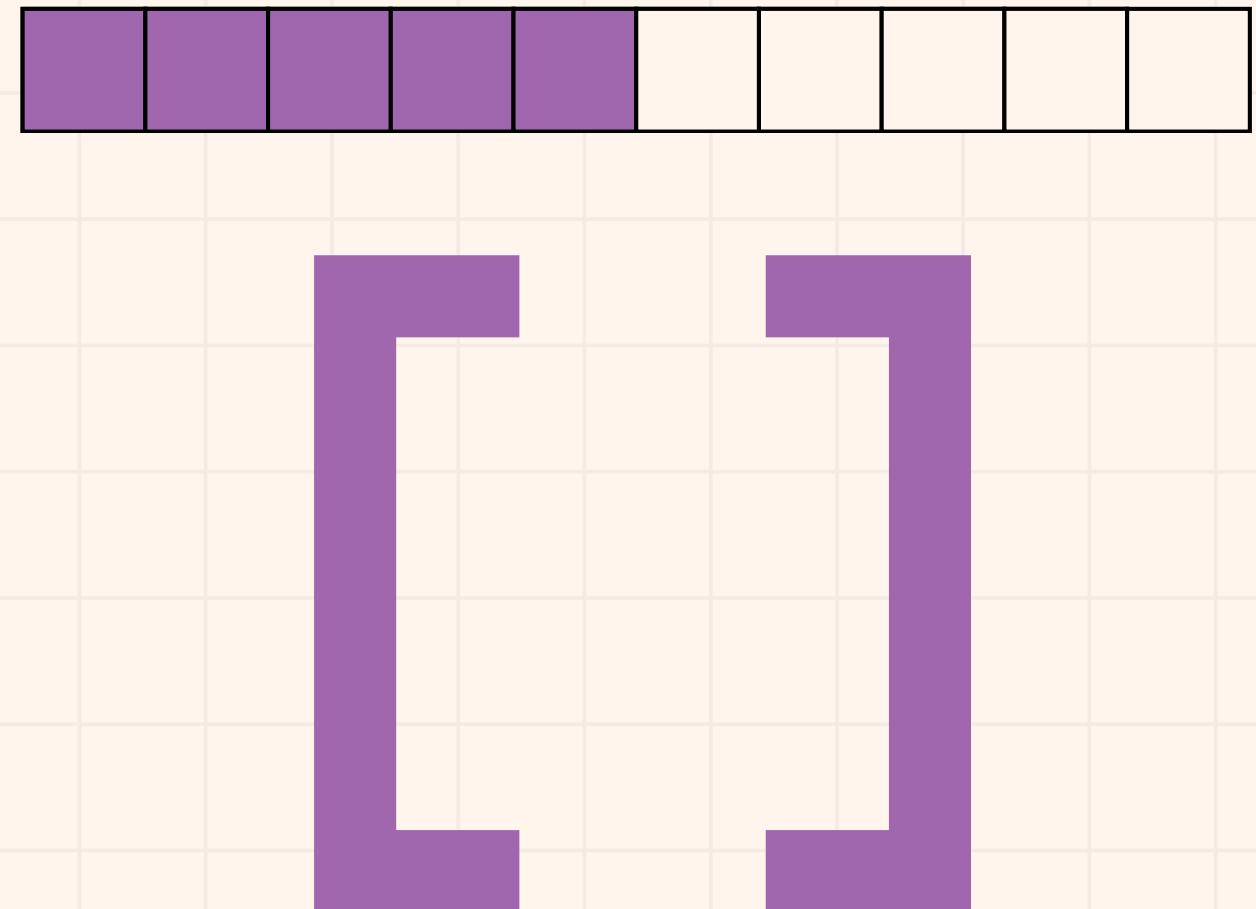
2: IRP and RP1 bits are reserved, always maintain these bits clear.

3: PIC16F676 only.

ARRAYS / VETORES

Na linguagem C, um array trata-se de uma estrutura de dados homogênea e de tamanho fixo, que permite armazenar dados do mesmo tipo em posições sucessivas da memória. Abaixo comentaremos algumas características desse tipo de estrutura.

- Permite dados de apenas um tipo (especificado no momento de sua declaração).
- O acesso as posições do array é feito utilizando-se índices que vão de 0 a n-1, onde n corresponde ao tamanho do array (indicado também no momento da declaração).
- A manipulação dos arrays é feita associando a eles laços de repetição (for/while), permitindo imprimir, buscar, sobrescrever valores etc...
- Possuem declaração de sintaxe muito simples e fácil de utilizar (observe imagem abaixo).



```
1 #include <stdio.h>
2
3 int main() {
4
5     //Declaração de arrays:
6
7     int array1[10]; // Array de 10 inteiros
8     float array2[15]; //Array de 15 floats
9     char array3[20]; /*Array de 20 caracteres normalmente conhecido como string;
10    possui uma biblioteca com funções específicas para manipulá-lo (string.h)*/
11
12     return 0;
13 }
```

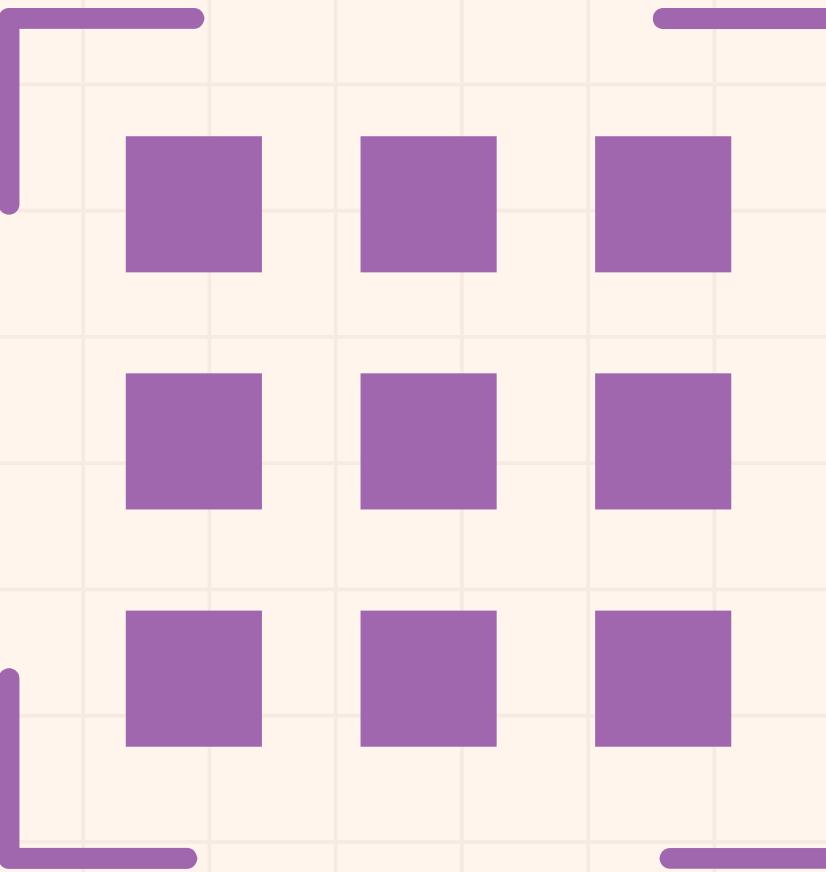
MATRIZES

Page 51

Em linguagem C, uma matriz é uma estrutura de dados multidimensional usada para armazenar elementos do mesmo tipo em uma grade bidimensional (ou superior). Ela pode ser vista como um vetor de vetores, organizado em linhas e colunas. Cada elemento é acessado usando dois índices: um para a linha e outro para a coluna.

Para a declaração de uma matriz, assim como visto em vetores usam-se as chaves, porém com uma mudança, agora são 4 chaves declaradas.

Ex: matriz1[][][], matriz2[][][]

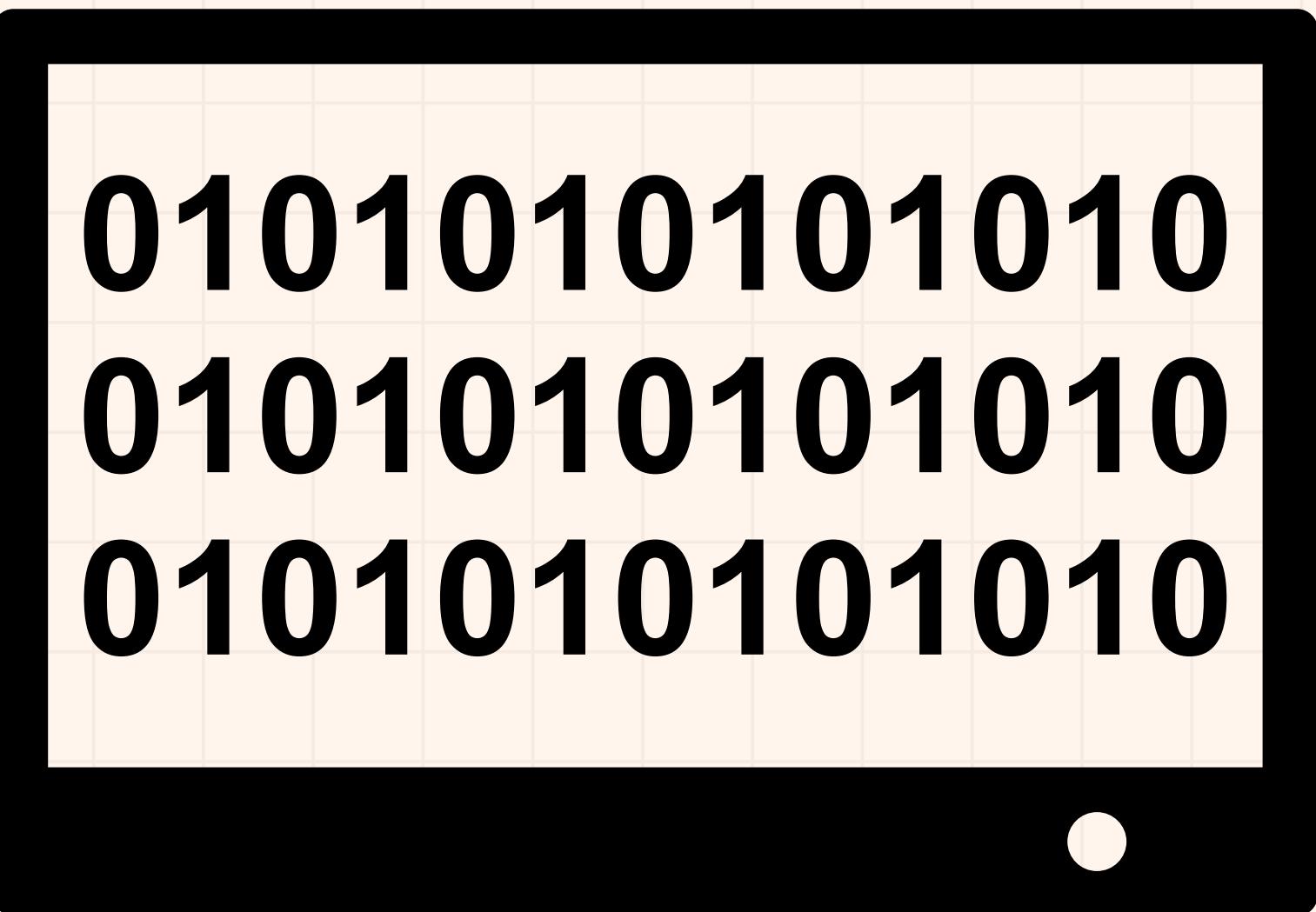


```
● ● ●  
1 #include <stdio.h>  
2  
3 int main(){  
4     int matriz[2][2] = {{1, 2}, {3, 4}};  
5     char matrizChar[2][2] = {{'a', 'b'}, {'c', 'd'}};  
6     float matrizFloat[2][2] = {{1.1, 2.2}, {3.3, 4.4}};  
7  
8     return 0;  
9 }
```

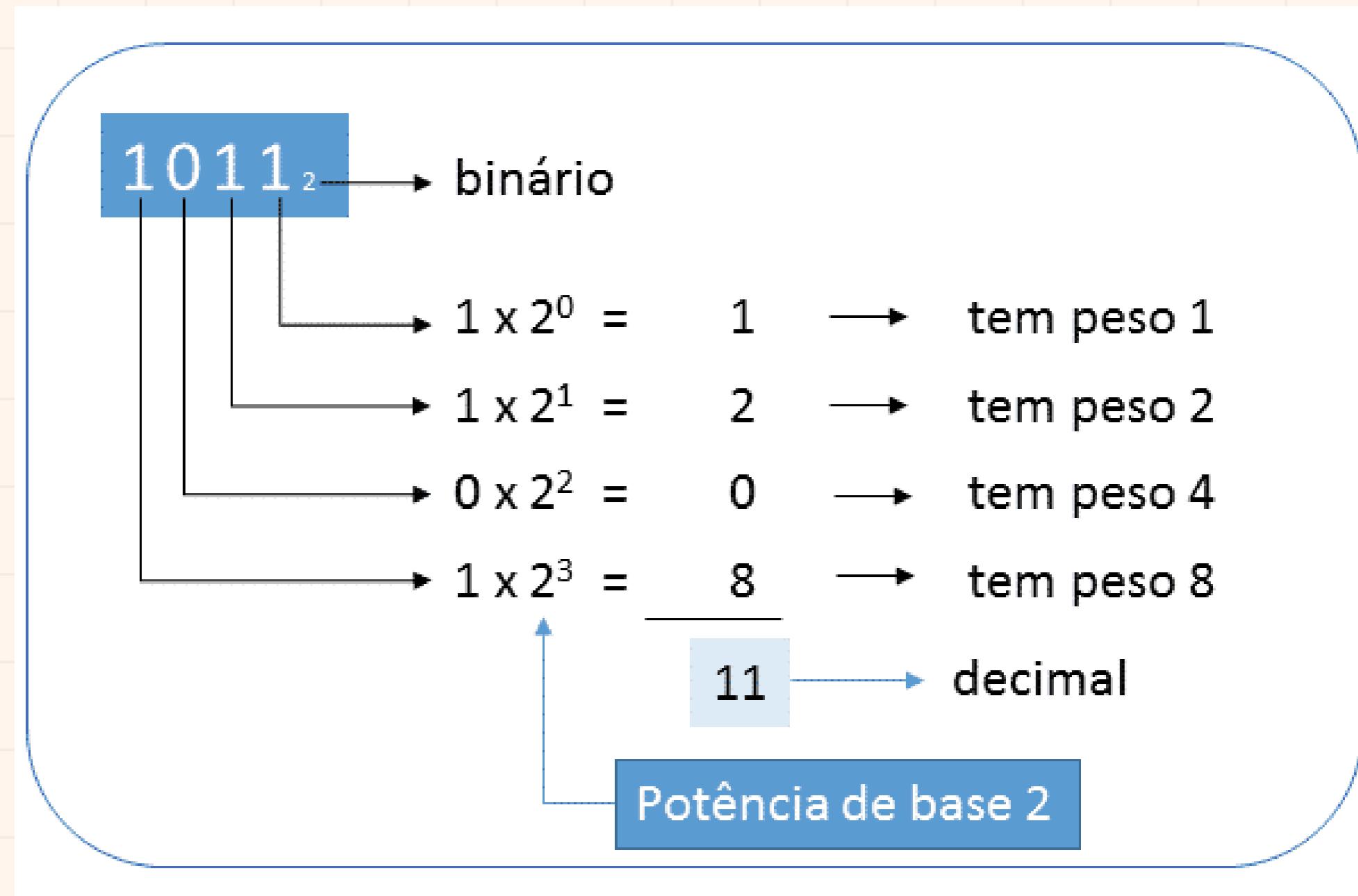
SISTEMA DE NUMERAÇÃO BINÁRIA

O sistema binário ou de base 2 é um sistema de numeração posicional em que todas as quantidades se representam com base em dois números, ou seja, zero e um (0 e 1).

Os computadores digitais trabalham internamente com dois níveis de tensão, pelo que o seu sistema de numeração natural é o sistema binário. Com efeito, num sistema simples como este é possível simplificar o cálculo, com o auxílio da lógica booleana. Em computação, chama-se um dígito binário (0 ou 1) de bit, que vem do inglês Binary Digit. Um agrupamento de 8 bits corresponde a um byte (Binary Term). Um agrupamento de 4 bits, ainda, é chamado de nibble.



SISTEMA DE NUMERAÇÃO BINÁRIA



0010 1001

$$1 \times 2^0 = 1$$

$$0 \times 2^1 = 0$$

$$0 \times 2^2 = 0$$

$$1 \times 2^3 = 16$$

$$0 \times 2^4 = 0$$

$$1 \times 2^5 = 64$$

$$0 \times 2^6 = 0$$

$$0 \times 2^7 = 0$$

$$= 81$$

OPERADORES DE BITWISE

Operadores bitwise são operações lógicas feitas "bit-a-bit" em números. Essas operações só podem ser feitas nos tipos **char**, **int** e **long int**

Operadores bitwise são utilizados quando precisamos realizar operações em nível de bits com números inteiros, ou seja, trabalhar com sua representação binário.

Não confunda os operadores bitwise com operadores lógicos, algumas representações são parecidas, mas suas funcionalidades são diferentes

Operator	Description
&	bitwise AND
	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	one's complement

OPERADORES DE BITWISE



```
1 #include <stdio.h>
2
3 int main(){
4     int a = 0b1010, b = 0b0011;// 10, 3
5     printf("AND: %d\n", a & b); // Resultado: 0010(2)
6     printf("OR: %d\n", a | b); // Resultado: 1011(11)
7     printf("XOR: %d\n", a ^ b); // Resultado: 1001(9)
8     printf("NOT: %d\n", ~a); // Resultado: 0101(-11)
9     printf("Deslocamento para a esquerda: %d\n", a << 1); // Resultado: 10100(20)
10    printf("Deslocamento para a direita: %d\n", a >> 1); // Resultado: 0101(5)
11
```

OPERADORES DE BITWISE

00000000 00001010
 $\sim A = 111111111\ 11110101$



```
1 unsigned char c = 0b1010; //2bytes
2 printf("NOT 2: %u\n", ~c); // Resultado (4294967285)
```

FUNCÕES E COMO CRIA-LAS EM C

Uma função nada mais é do que uma subrotina usada em um programa.

Na linguagem C, denominamos função a um conjunto de comandos que realiza uma tarefa específica em um módulo dependente de código.

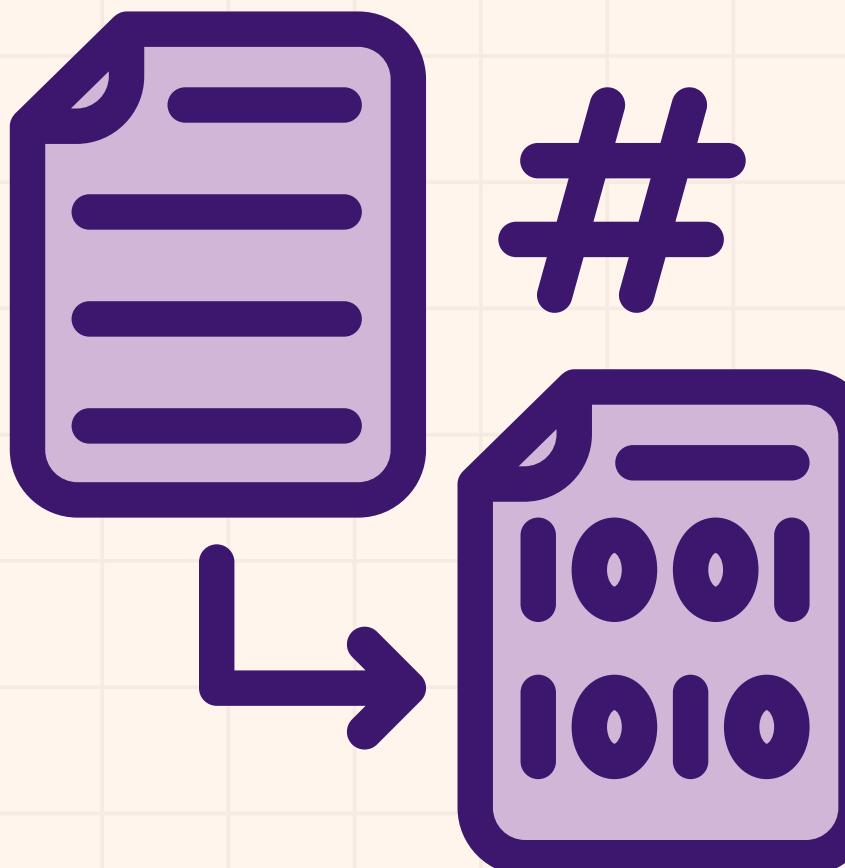
A função é referenciada pelo programa principal através do nome atribuído a ela.

A utilização de funções visa modularizar um programa, o que é muito comum em programação estruturada.

Desta forma podemos dividir um programa em várias partes, no qual cada função realiza uma tarefa bem definida.

`tipo_de_retorno nome_da_função (listagem de parâmetros)`

FUNCTION()



FUNÇÕES E COMO CRIA-LAS EM C



```
1 #include <stdio.h>
2
3 int somaPorValor(int a, int b){
4     return a + b;
5 }
6
7 int somaPorReferencia(int *a, int *b){
8     return *a + *b;
9 }
10
11 int main(){
12     int a = 10, b = 3;
13     printf("Soma: %d\n", soma(a, b)); // Resultado: 13
14     printf("Soma por referência: %d\n", somaPorReferencia(&a, &b)); // Resultado: 13
15     return 0;
16 }
```

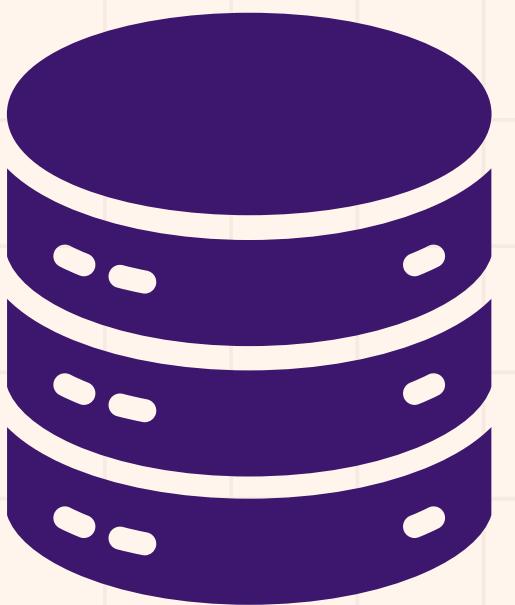
FUNÇÕES E COMO CRIA-LAS EM C

```
23 // Inicialização do microcontrolador
24 void init(){
25     TRIS_C = 0x00; // Configura todos os pinos de PORTC como saída
26     TRIS_A = 0xC1; // RA0 como entrada (DHT11), RA2 e RA5 como saídas
27     PORTC = 0x00; // Inicializa PORTC com 0
28     PORTA &= 0xF3; // Garante que RA2 e RA5 estão zerados
29 }
30
31 unsigned char decimal_to_bcd(unsigned char decimal){
32     unsigned char bcd = 0;
33     for (int i = 0; i < 4; i++) {
34         bcd = (bcd << 1) | (decimal & 1);
35         decimal >>= 1;
36     }
37     return bcd;
38 }
39
40 // Função para enviar um valor BCD para um display específico
41 void sendToDisplay1(unsigned char value){
42     PORTC &= 0b00000011; // Limpa os bits RC5 a RC2
43     PORTC |= (value << 2); // Envia os bits para RC5 a RC2
44 }
45
46 void sendToDisplay2(unsigned char value){
47     PORTC &= 0b1111100; // Limpa RC1 e RC0
48     PORTC |= (value & 0x03); // Bits menos significativos para RC1 e RC0
49     PORTA &= 0b11001111; // Limpa RA5 e RA2
50     PORTA |= ((value & 0x0C) << 2); // Bits mais significativos para RA5 e RA2
51 }
52
53 void main(){
54     unsigned char temp_int, temp_dec; // Parte inteira e decimal da temperatura
55
56     init(); // Configuração inicial
57
58 while (1){
59     // Solicita dados do DHT11
60     DHT11_Start();
61
62     DHT11_ReadByte(); // Ignora umidade inteira
63     DHT11_ReadByte(); // Ignora umidade decimal
64     temp_int = DHT11_ReadByte(); // Parte inteira da temperatura
65     temp_dec = DHT11_ReadByte(); // Parte decimal da temperatura
66     DHT11_ReadByte(); // Ignora checksum
67
68     // Envia os valores lidos para os displays
69     sendToDisplay1(9); // Parte decimal no primeiro display
70     sendToDisplay2(8); // Parte inteira no segundo display
71
72     delay_ms(1000); // Acesso de 1 segundo
73 }
```

```
1  char read_dht11(){
2
3     char data, for_count;
4
5     for(for_count = 0; for_count < 8; for_count++){
6
7         while(!DHT11_Data_Pin);
8
9         __delay_us(30);
10
11        if(DHT11_Data_Pin == 0){
12
13            data&= ~(1<<(7 - for_count)); //Clear bit (7-b)
14
15        }
16
17        else{
18
19            data|= (1 << (7 - for_count)); //Set bit (7-b)
20
21        while(DHT11_Data_Pin);
22
23    } //Wait until PORTD.F0 goes LOW
24
25 }
26
27 return data;
28
29 }
```

PONTEIROS E ENDEREÇOS

Os conceitos de endereço e ponteiro são fundamentais em qualquer linguagem de programação. Em C, esses conceitos são explícitos; em algumas outras linguagens eles são ocultos (e representados pelo conceito mais abstrato de referência). Dominar o conceito de ponteiro exige algum esforço e uma boa dose de prática.



Um ponteiro pode ter o valor NULL que é um endereço inválido. A macro NULL está definida na interfacestdlib.h e seu valor é 0 (zero) na maioria dos computadores. Se um ponteiro p armazena o endereço de uma variável i, podemos dizer p aponta para i ou p é o endereço de i. (Em termos um pouco mais abstratos, diz-se que p é uma referência à variável i.) Se um ponteiro p tem valor diferente de NULL então de bytes de uma variável é dado pelo operador sizeof. A expressão sizeof (char), por exemplo, vale 1 em todos os computadores e a expressão sizeof (int) vale 4 em muitos computadores.

PONTEIROS E ENDEREÇOS



```
1 #include <stdio.h>
2
3 void somaponteiros(int *a, int *b){
4     *a = *a + *b;
5 }
6
7 int soma(int a, int b){
8     a = a + b;
9     return a;
10}
11
12 int main(){
13     int a = 5;
14     int b = 3;
15     soma(&a, &b);
16     printf("%d", a);
17     return 0;
18}
19
```

Suponha que a, b e c são variáveis inteiras e veja um jeito bobo de fazer $c = a+b$:

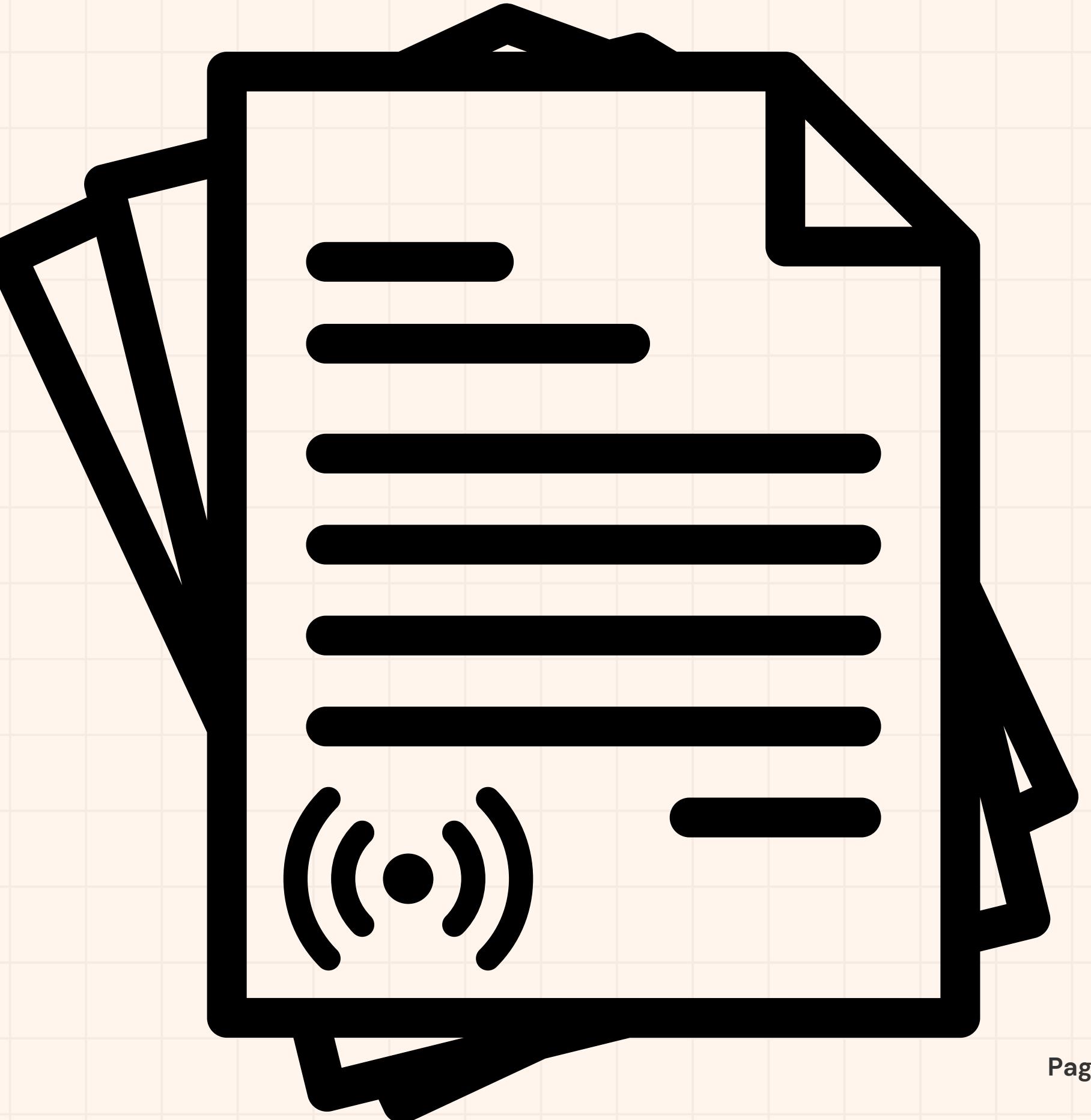


```
1 int *p; // p é um ponteiro para um inteiro
2 int *q;
3 p = &a; // o valor de p é o endereço de a
4 q = &b; // q aponta para b
5 c = *p + *q;
```

DECODIFICAÇÃO PELO DATASHEET

Agora que já entendemos o que é um datasheet, na questão de sensores e outros periféricos, é necessário entender o seu funcionamento corretamente para poder decodificá-lo da melhor maneira.

Para isso vamos ver quais as principais informações que devemos tirar ao se ler um datasheet de algum Periférico/Sensor



DECODIFICAÇÃO

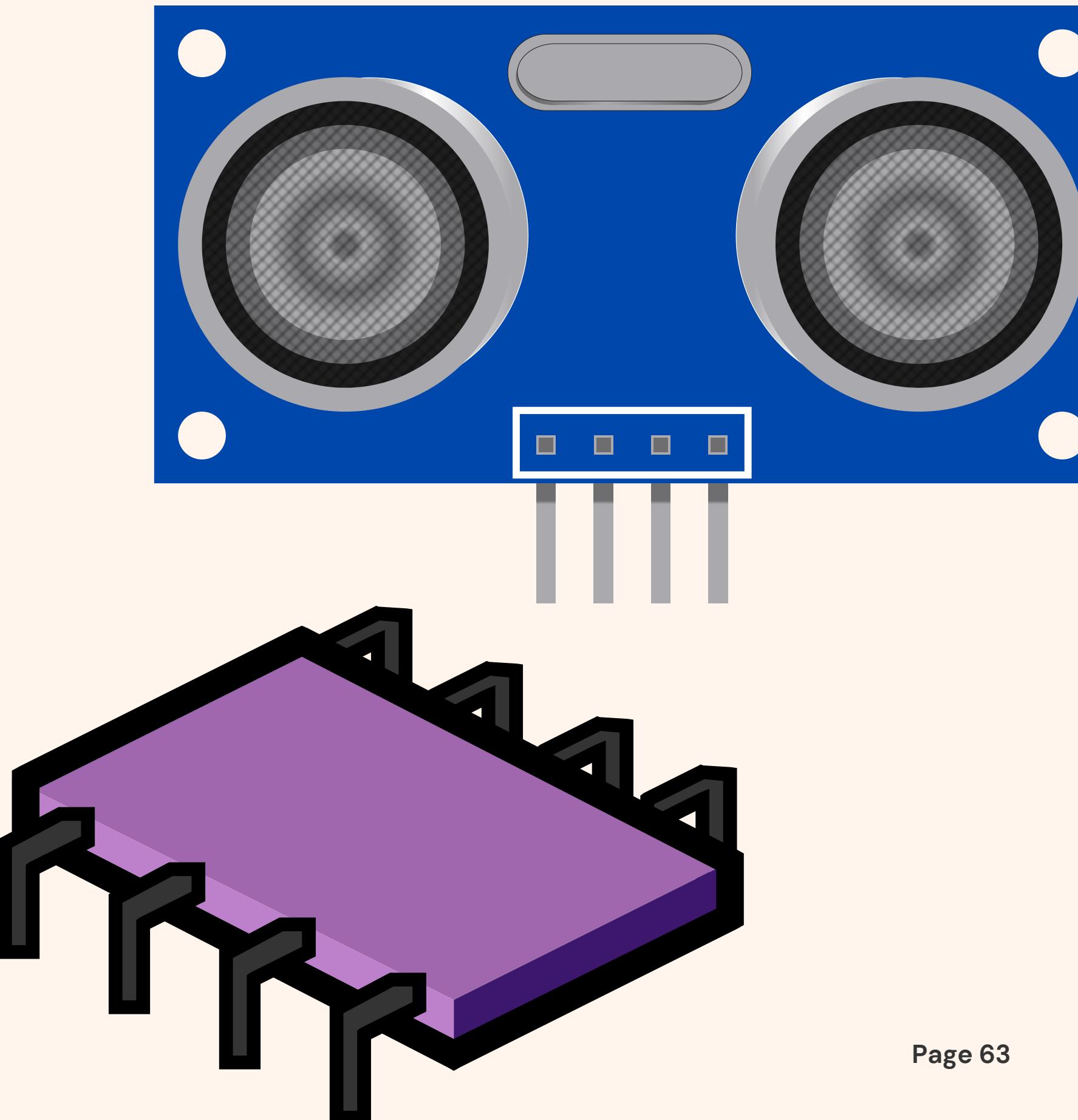
HC-SR04

O sensor ultrasônico HC-SR04 funciona como um detector de objetos e permite medir distâncias mínimas de 2 centímetros podendo chegar a distâncias máximas de até 4 metros com uma precisão de 3 milímetros. Estes sensores emitem um sinal ultrassônico (40 kHz) que ao atingir um objeto, retorna ao sensor. O sinal de retorno é captado, permitindo-se calcular a distância do objeto ao sensor, medindo o tempo de ida e volta do sinal emitido.

Usando a seguinte fórmula, é possível calcular a distância, sendo que v (velocidade do som) é ~350m/s.

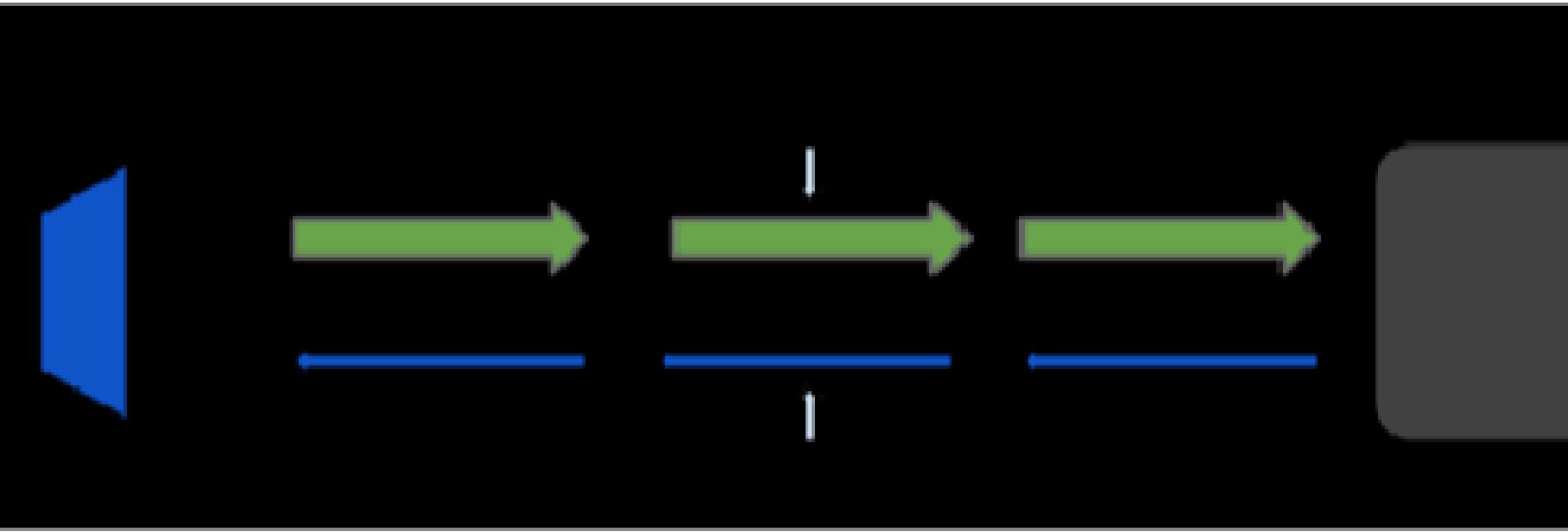
$$d[m] = (v[m/s] * t[s]) / 2$$

Para a contagem do tempo, foi utilizado o Timer1, configurado de modo que incremente a cada 1us.



1. How Ultrasonic Sensors Work

Ultrasonic sensors use sound to determine the distance between the sensor and the closest object in its path. How do ultrasonic sensors do this? Ultrasonic sensors are essentially sound sensors, but they operate at a frequency above human hearing.



The sensor sends out a sound wave at a specific frequency. It then listens for that specific sound wave to bounce off of an object and come back (Figure 1). The sensor keeps track of the time between sending the sound wave and the sound wave returning. If you know how fast something is going and how long it is traveling you can find the distance traveled with equation 1.

$$\text{Equation 1. } d = v \times t$$

The speed of sound can be calculated based on a variety of atmospheric conditions, including temperature, humidity and pressure. Actually calculating the distance will be shown later on in this document.

It should be noted that ultrasonic sensors have a cone of detection, the angle of this cone varies with distance, Figure 2 shows this relation. The ability of a sensor to

3. Timing Chart and Pin Explanations

The HCSR04 has four pins, VCC, GND, TRIG and ECHO; these pins all have different functions. The VCC and GND pins are the simplest they power the HCSR04. These pins need to be attached to a +5 volt source and ground respectively. There is a single control pin: the TRIG pin. The TRIG pin is responsible for sending the ultrasonic burst. This pin should be set to HIGH for 10 μ s, at which point the HCSR04 will send out an eight cycle sonic burst at 40 kHz. After a sonic burst has been sent the ECHO pin will go HIGH. The ECHO pin is the data pin it is used in taking distance measurements. After an ultrasonic burst is sent the pin will go HIGH, it will stay high until an ultrasonic burst is detected back, at which point it will go LOW.

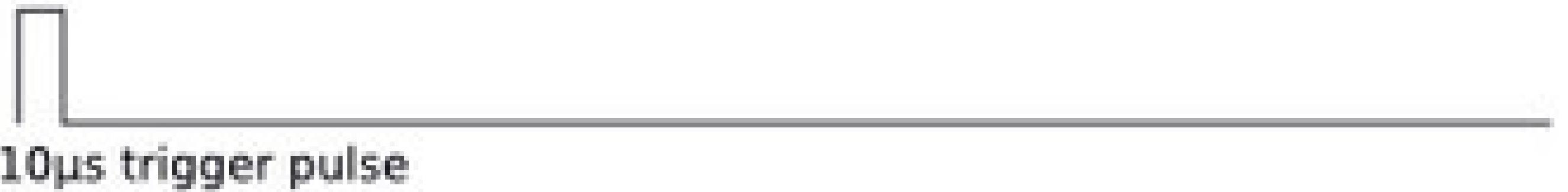
Taking Distance Measurements

The HCSR04 can be triggered to send out an ultrasonic burst by setting the TRIG pin to HIGH. Once the burst is sent the ECHO pin will automatically go HIGH. This pin will remain HIGH until the the burst hits the sensor again. You can calculate the distance to the object by keeping track of how long the ECHO pin stays HIGH. The time ECHO stays HIGH is the time the burst spent traveling. Using this measurement in equation 1 along with the speed of sound will yield the distance travelled. A summary of this is listed below, along with a visual representation in Figure 2.

1. Set TRIG to HIGH
2. Set a timer when ECHO goes to HIGH
3. Keep the timer running until ECHO goes to LOW
4. Save that time
5. Use equation 1 to determine the distance travelled

HC-SR04 Timing Diagram

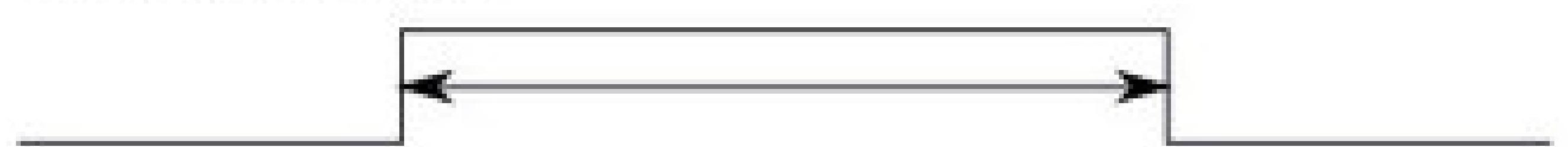
Trigger Pin



Transmit Wave



Echo Pin



Time it takes pulse to leave and return to sensor

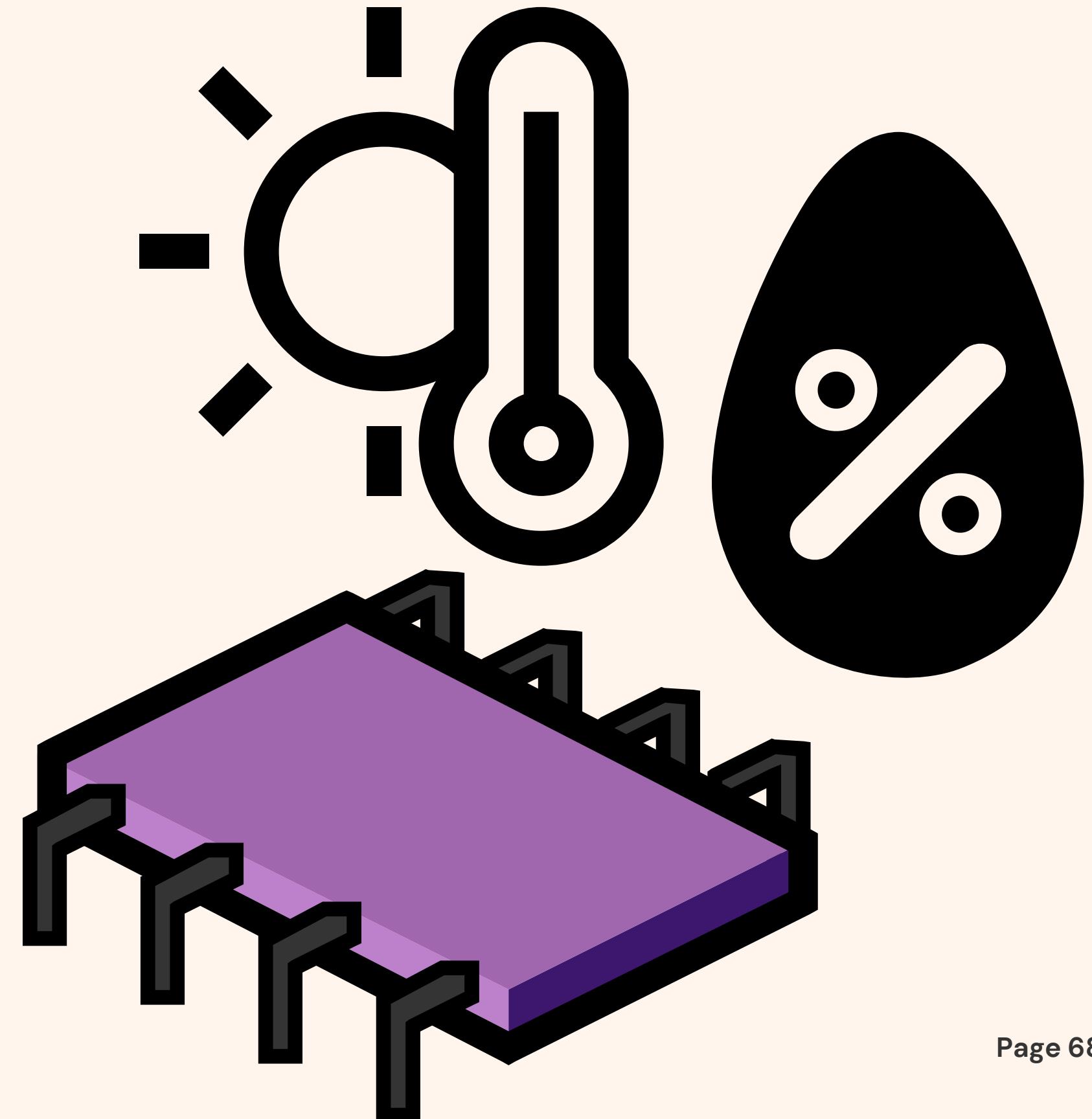


```
1
2     unsigned int measure_distance() {
3         unsigned int time = 0;
4
5         // Envia o pulso TRIG
6         TRIG_PIN = 1;
7         _delay_us(10);
8         TRIG_PIN = 0;
9
10        // Aguarda o ECHO ficar alto
11        while (!ECHO_PIN);
12
13        // Mede o tempo que o ECHO permanece alto
14        while (ECHO_PIN) {
15            _delay_us(1);
16            time++;
17            if (time > 30000) { // Timeout para distâncias muito grandes
18                return 300; // Retorna uma distância máxima (~300 cm)
19            }
20        }
21
22        // Calcula a distância em cm
23        // Velocidade do som = 340 m/s, ou 29 us/cm ida e volta
24        return time / 58;
25    }
26
```

DECODIFICAÇÃO

DHT11

O sensor DHT11 vem com um invólucro de cor azul ou branco. Dentro deste invólucro, temos dois componentes importantes que nos ajudam a sentir a umidade relativa e a temperatura. O primeiro componente é um par de eletrodos; a resistência elétrica entre esses dois eletrodos é decidida por um substrato de retenção de umidade. Portanto, a resistência medida é inversamente proporcional à umidade relativa do ambiente. Maior a umidade relativa menor será o valor de resistência e vice-versa. Além disso, note que a umidade relativa é diferente da umidade real. A humidade relativa mede o teor de água no ar em relação à temperatura no ar.



DECODIFICAÇÃO

DHT11

3. Typical Application (Figure 1)

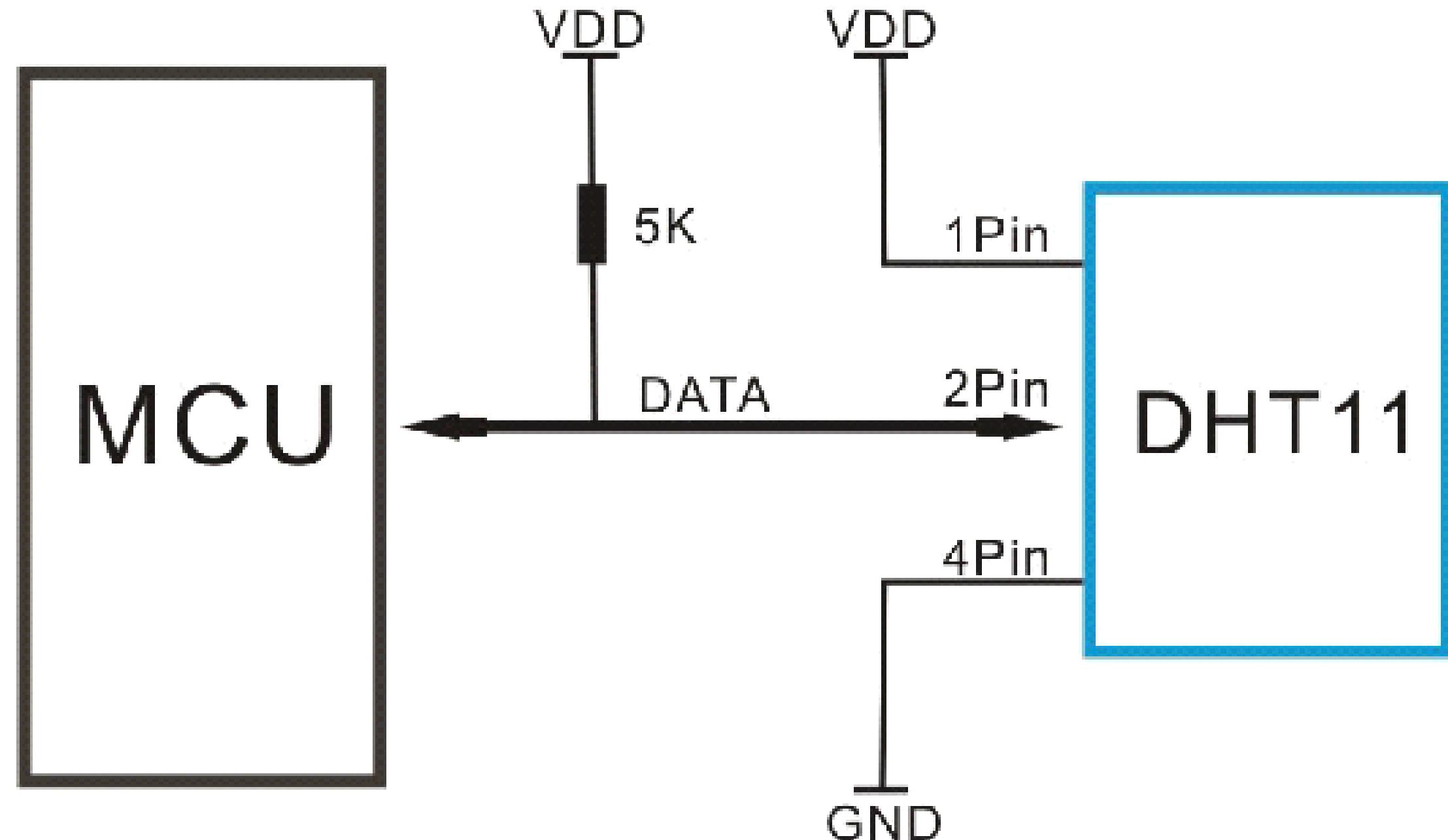


Figure 1 Typical Application

DECODIFICAÇÃO

DHT11

4. Power and Pin

DHT11's power supply is 3-5.5V DC. When power is supplied to the sensor, do not send any instruction to the sensor in within one second in order to pass the unstable status. One capacitor valued 100nF can be added between VDD and GND for power filtering.

5. Communication Process: Serial Interface (Single-Wire Two-Way)

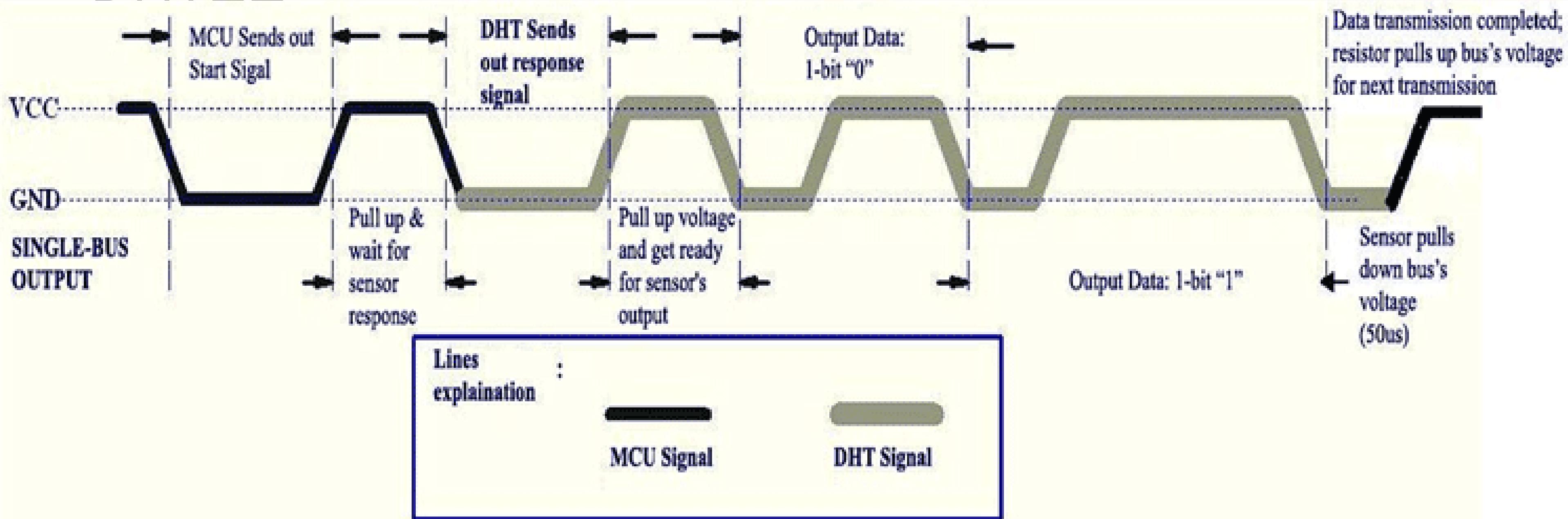
Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms.

Data consists of decimal and integral parts. A complete data transmission is 40bit, and the sensor sends higher data bit first.

Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

DECODIFICAÇÃO

DHT11



DECODIFICAÇÃO

DHT11



```
1 void dht11_init(){
2     DHT11_Data_Pin_Direction= 0; //Configura RD0 como output
3     DHT11_Data_Pin = 0; //RD0 envia 0 para o sensor
4     __delay_ms(18);
5     DHT11_Data_Pin = 1; //RD0 envia 1 para o sensor
6     __delay_us(30);
7     DHT11_Data_Pin_Direction = 1; //configura RD0 como input
8 }
9 void find_response(){
10     Check_bit = 0;
11     __delay_us(40);
12     if (DHT11_Data_Pin == 0){
13         __delay_us(80);
14         if (DHT11_Data_Pin == 1){
15             Check_bit = 1;
16         }
17         __delay_us(50);}
```

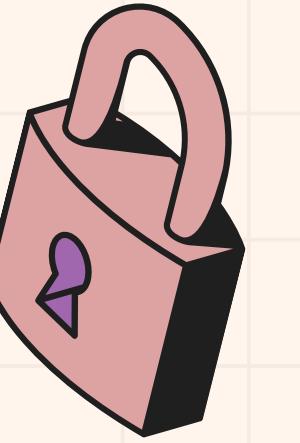
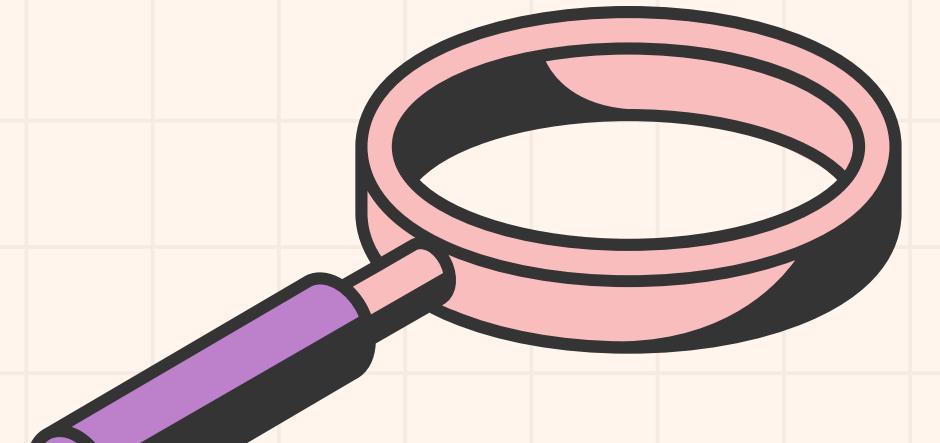
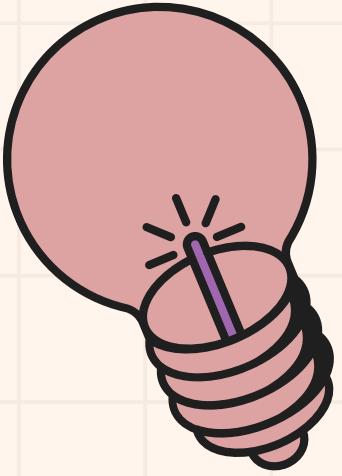
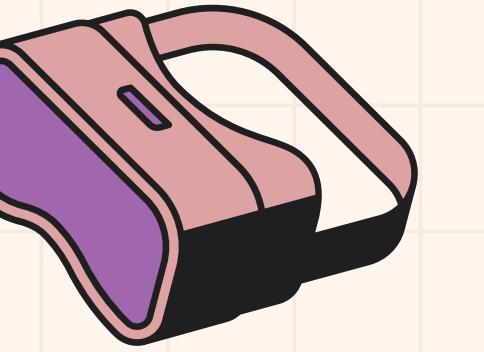
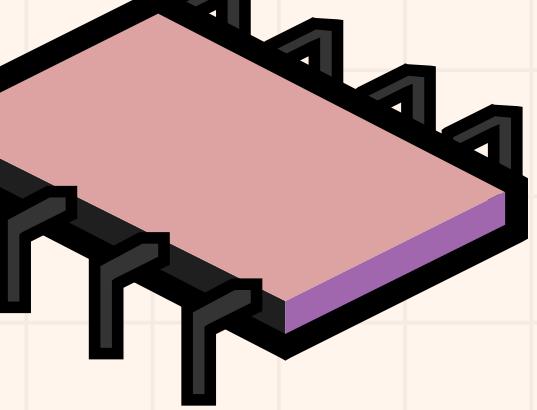
DECODIFICAÇÃO

DHT11



```
1
2  char read_dht11(){
3  char data, for_count;
4  for(for_count = 0; for_count < 8; for_count++){
5      while(!DHT11_Data_Pin);
6      __delay_us(30);
7      if(DHT11_Data_Pin == 0){
8          data&= ~(1<<(7 - for_count)); //Clear bit (7-b)
9      }
10     else{
11         data|= (1 << (7 - for_count)); //Set bit (7-b)
12         while(DHT11_Data_Pin);
13     }
14 }
15 return data;
16 }
```

OBRIGADO



BIBLIOGRAFIA:

ESSAS CABEÇAS



CADO

FININ

RESEBBA

NIEL

BIBLIOGRAFIA: