

Introdução às Interfaces de Comunicação Serial com RP2040 UART, SPI e I2C

Unidade 4 | Capítulo 6 - Aula síncrona (03/02/2024)

Prof. Wilton Lacerda Silva

Executores:



Coordenação:



Iniciativa:



Jornada até aqui...

Executores:



Coordenação:



Iniciativa:

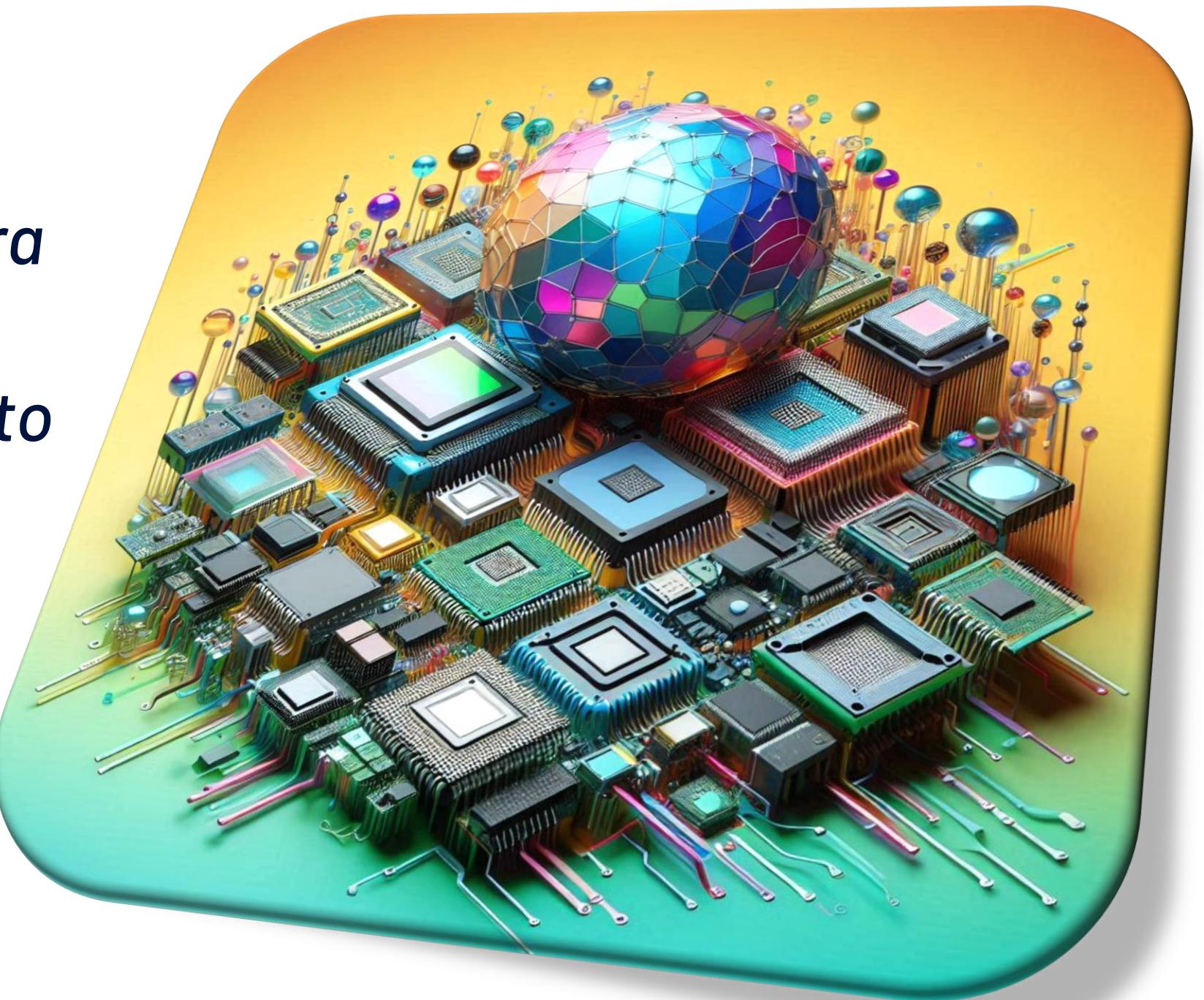


Sumário

- Objetivos
- Revisão
- Contexto de Interfaces Seriais
- UART
- SPI
- I2C
- Exemplos de Código e aplicações
- Principais Pontos
- Conclusão

Pré-requisitos

- Familiaridade com programação em C/C++ para microcontroladores..
- Noções de uso do ambiente de desenvolvimento VS Code.
- Sistemas de numeração.
- Compreensão do exercício S.O.S. em morse.



Objetivos

- Compreender o funcionamento dos diferentes protocolos de interfaces de comunicação serial.
- Implementar e configurar UART, SPI e I2C no RP2040.
- Aplicar essas interfaces em projetos reais com exemplos de código e esquemas.



Revisão conversão de números binários

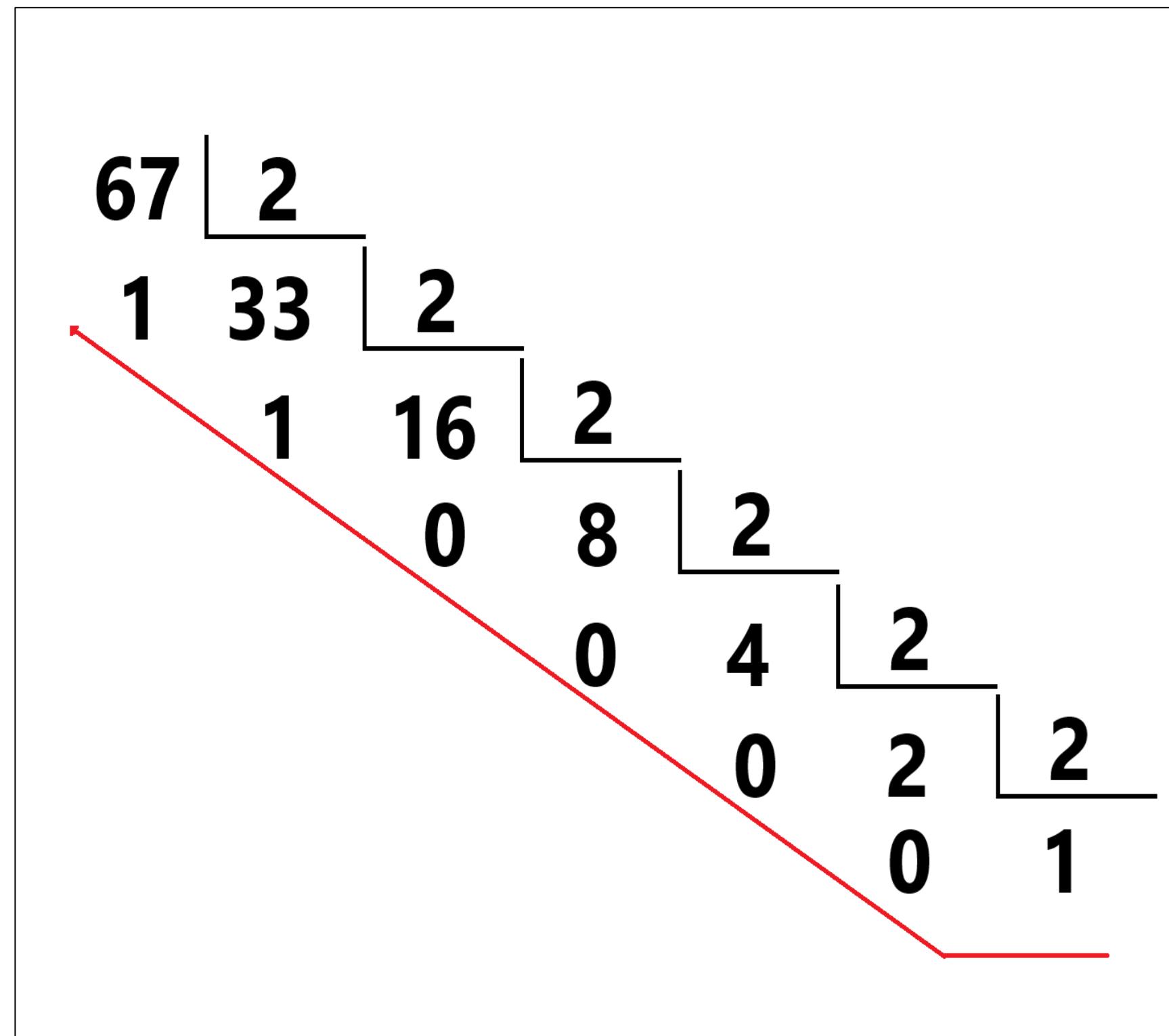
Como converter um número de decimal para binário?

67 escrito em decimal tem qual representação em binário?

000000????

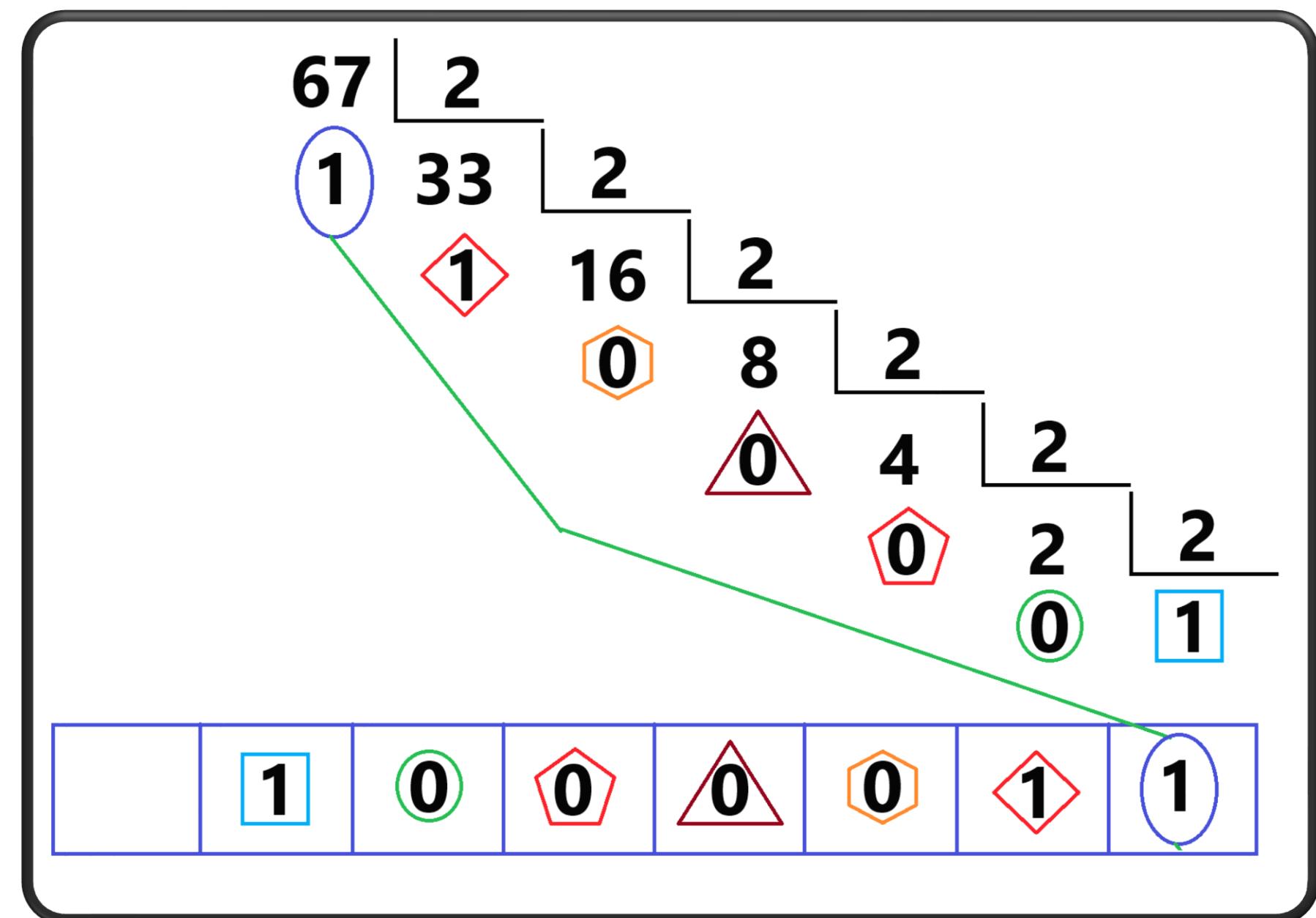
Revisão conversão de números binários

Conversão de Decimal para binário



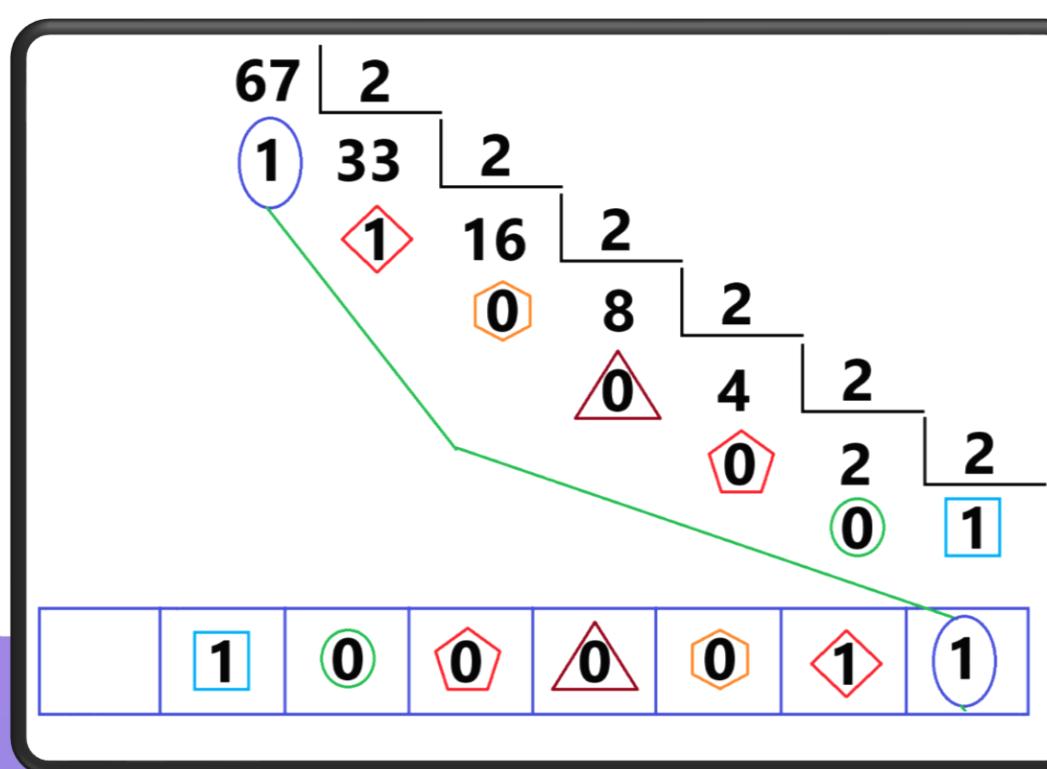
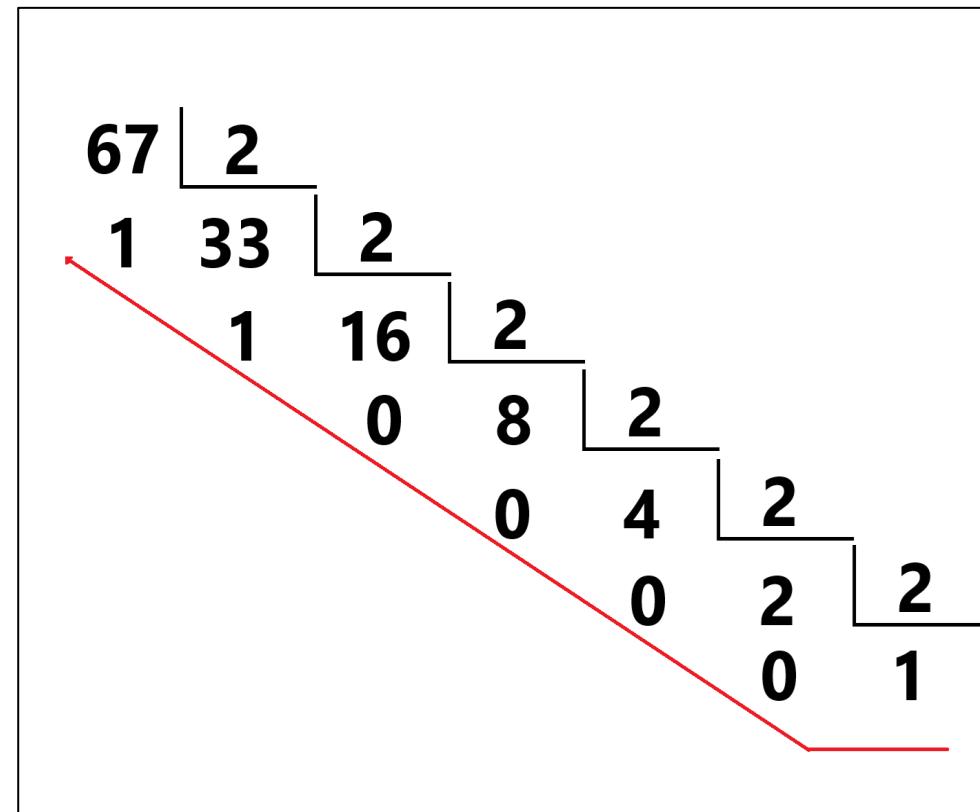
Revisão conversão de números binários

Representação do número em binário



Revisão conversão de números binários

Conversão de Decimal para binário



Formação do Byte (8 bits)

Sistema decimal

100	10	1
10^2	10^1	10^0

100	10	1
0	6	7

Sistema binário

128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

128	64	32	16	8	4	2	1
0	1	0	0	0	0	1	1

msb

lsb

Revisão conversão de números binários

Transformações binário para decimal.

128	64	32	16	8	4	2	1	
0	0	0	0	1	1	1	1	15
128	64	32	16	8	4	2	1	
0	0	0	0	1	1	1	0	14
128	64	32	16	8	4	2	1	
0	1	1	1	1	1	1	1	127
128	64	32	16	8	4	2	1	
0	0	0	0	0	0	1	1	3
128	64	32	16	8	4	2	1	
0	0	0	0	0	1	1	0	6
128	64	32	16	8	4	2	1	
0	0	0	0	1	1	0	0	12
128	64	32	16	8	4	2	1	
0	0	0	1	1	0	0	0	24

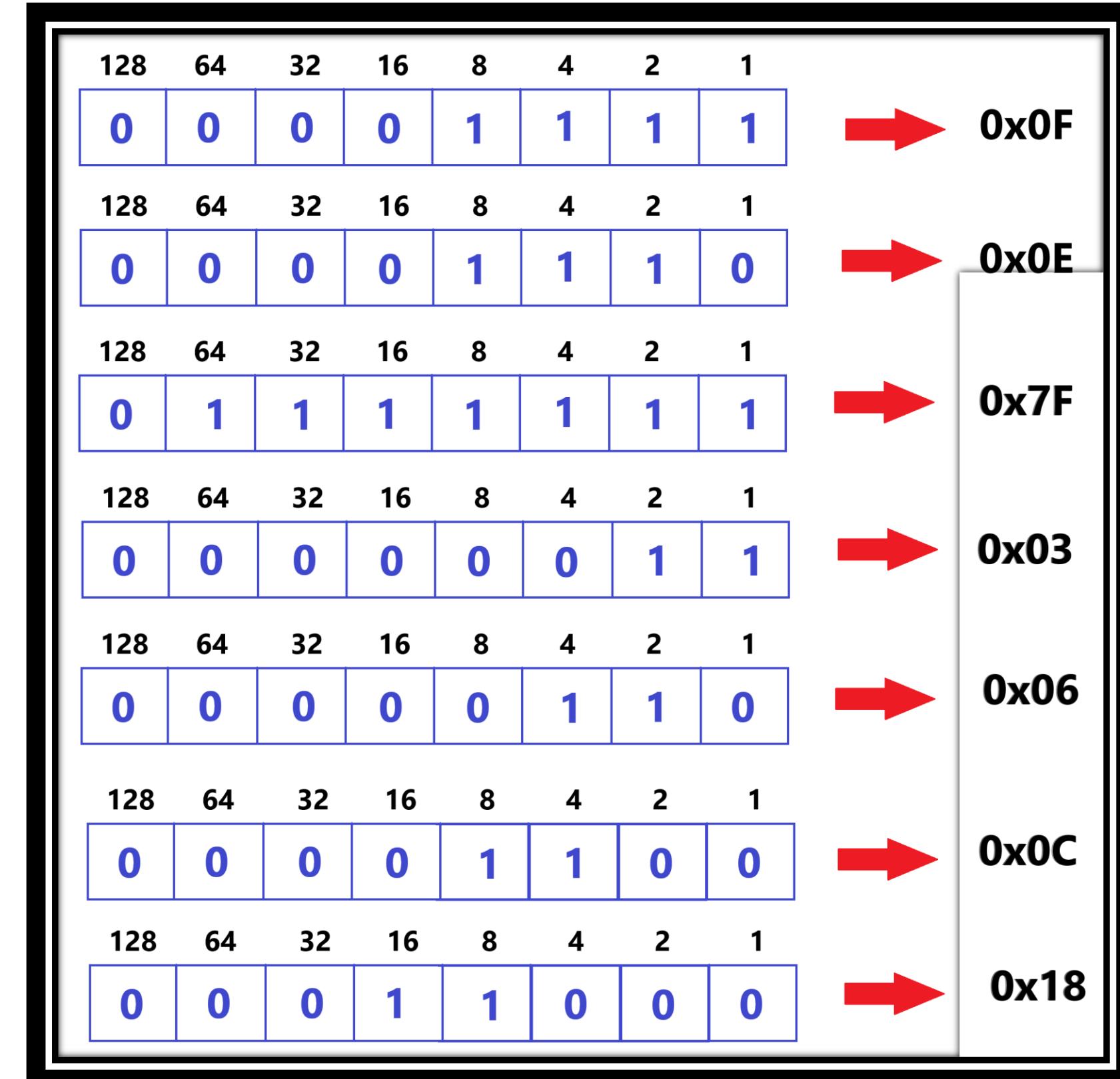
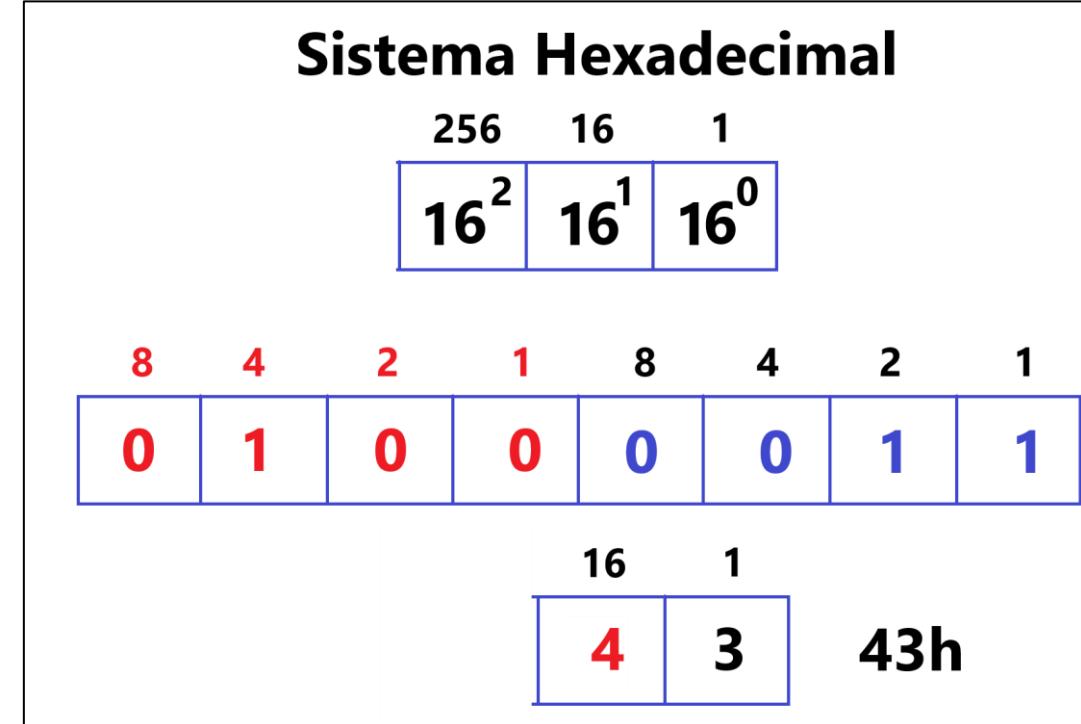
Formação do Byte (8 bits)

Sistema binário								
128	64	32	16	8	4	2	1	
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
0	1	0	0	0	0	1	1	
msb				lsb				

Revisão: Binários e Hexadecimal

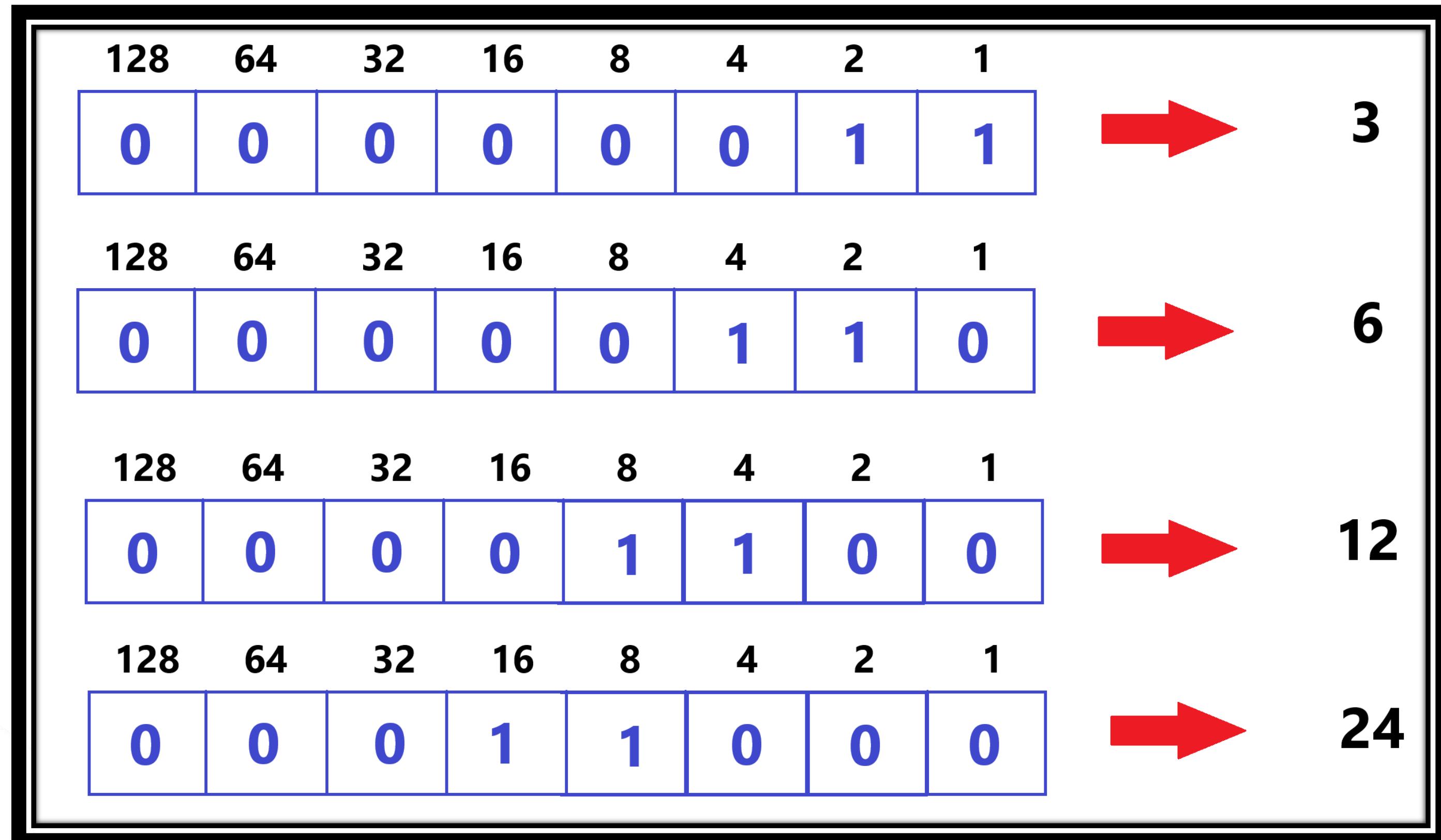
Sistema Hexadecimal.

Decimal	Binário	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



Revisão Sistema digital

Rotação (multiplicação e divisão)
(Deslocamento para esquerda<< ou direita>>)

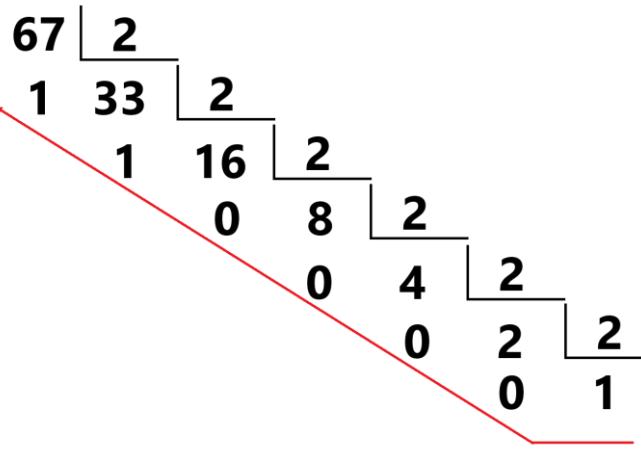


Revisão Sistema digital

Transformações e rotações.

Obs: rotações implicam em multiplicação ou divisão.

Conversão de Decimal para binário



Sistema decimal			Sistema binário							
100	10	1	128	64	32	16	8	4	2	1
10^2	10^1	10^0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
100	10	1	128	64	32	16	8	4	2	1
0	6	7	0	1	0	0	0	0	1	1
			msb						lsb	

Decimal	Binário	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Sistema Hexadecimal		
256	16	1
16^2	16^1	16^0
8	4	2
0	1	0
0	0	1
0	1	1
1	1	0
4	3	43h

128	64	32	16	8	4	2	1
0	0	0	0	1	1	1	1
0	0	0	0	0	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	1	0
0	0	0	0	1	1	0	0
0	0	0	1	1	0	0	0

Revisão: Tabela ASCII

ASCII control characters			ASCII printable characters																				
Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr										
00	NULL	(Null character)	32	space	64	@	96	`	0 0	000	NULL	32 20	040	 	Space	64 40	100	@	@	96 60	140	`	`
01	SOH	(Start of Header)	33	!	65	A	97	a	1 1	001	Start of Header	33 21	041	!	!	65 41	101	A	A	97 61	141	a	a
02	STX	(Start of Text)	34	"	66	B	98	b	2 2	002	Start of Text	34 22	042	"	"	66 42	102	B	B	98 62	142	b	b
03	ETX	(End of Text)	35	#	67	C	99	c	3 3	003	End of Text	35 23	043	#	#	67 43	103	C	C	99 63	143	c	c
04	EOT	(End of Trans.)	36	\$	68	D	100	d	4 4	004	End of Transmission	36 24	044	$	\$	68 44	104	D	D	100 64	144	d	d
05	ENQ	(Enquiry)	37	%	69	E	101	e	5 5	005	Enquiry	37 25	045	%	%	69 45	105	E	E	101 65	145	e	e
06	ACK	(Acknowledgement)	38	&	70	F	102	f	6 6	006	Acknowledgment	38 26	046	&	&	70 46	106	F	F	102 66	146	f	f
07	BEL	(Bell)	39	'	71	G	103	g	7 7	007	Bell	39 27	047	'	'	71 47	107	G	G	103 67	147	g	g
08	BS	(Backspace)	40	(72	H	104	h	8 8	010	Backspace	40 28	050	((72 48	110	H	H	104 68	150	h	h
09	HT	(Horizontal Tab)	41)	73	I	105	i	9 9	011	Horizontal Tab	41 29	051))	73 49	111	I	I	105 69	151	i	i
10	LF	(Line feed)	42	*	74	J	106	j	10 A	012	Line feed	42 2A	052	*	*	74 4A	112	J	J	106 6A	152	j	j
11	VT	(Vertical Tab)	43	+	75	K	107	k	11 B	013	Vertical Tab	43 2B	053	+	+	75 4B	113	K	K	107 6B	153	k	k
12	FF	(Form feed)	44	,	76	L	108	l	12 C	014	Form feed	44 2C	054	,	,	76 4C	114	L	L	108 6C	154	l	l
13	CR	(Carriage return)	45	-	77	M	109	m	13 D	015	Carriage return	45 2D	055	-	-	77 4D	115	M	M	109 6D	155	m	m
14	SO	(Shift Out)	46	.	78	N	110	n	14 E	016	Shift Out	46 2E	056	.	.	78 4E	116	N	N	110 6E	156	n	n
15	SI	(Shift In)	47	/	79	O	111	o	15 F	017	Shift In	47 2F	057	/	/	79 4F	117	O	O	111 6F	157	o	o
16	DLE	(Data link escape)	48	0	80	P	112	p	16 10	020	Data Link Escape	48 30	060	0	0	80 50	120	P	P	112 70	160	p	p
17	DC1	(Device control 1)	49	1	81	Q	113	q	17 11	021	Device Control 1	49 31	061	1	1	81 51	121	Q	Q	113 71	161	q	q
18	DC2	(Device control 2)	50	2	82	R	114	r	18 12	022	Device Control 2	50 32	062	2	2	82 52	122	R	R	114 72	162	r	r
19	DC3	(Device control 3)	51	3	83	S	115	s	19 13	023	Device Control 3	51 33	063	3	3	83 53	123	S	S	115 73	163	s	s
20	DC4	(Device control 4)	52	4	84	T	116	t	20 14	024	Device Control 4	52 34	064	4	4	84 54	124	T	T	116 74	164	t	t
21	NAK	(Negative acknowl.)	53	5	85	U	117	u	21 15	025	Negative Ack.	53 35	065	5	5	85 55	125	U	U	117 75	165	u	u
22	SYN	(Synchronous idle)	54	6	86	V	118	v	22 16	026	Synchronous idle	54 36	066	6	6	86 56	126	V	V	118 76	166	v	v
23	ETB	(End of trans. block)	55	7	87	W	119	w	23 17	027	End of Trans. Block	55 37	067	7	7	87 57	127	W	W	119 77	167	w	w
24	CAN	(Cancel)	56	8	88	X	120	x	24 18	030	Cancel	56 38	070	8	8	88 58	130	X	X	120 78	170	x	x
25	EM	(End of medium)	57	9	89	Y	121	y	25 19	031	End of Medium	57 39	071	9	9	89 59	131	Y	Y	121 79	171	y	y
26	SUB	(Substitute)	58	:	90	Z	122	z	26 1A	032	Substitute	58 3A	072	:	:	90 5A	132	Z	Z	122 7A	172	z	z
27	ESC	(Escape)	59	;	91	[123	{	27 1B	033	Escape	59 3B	073	;	;	91 5B	133	[[123 7B	173	{	{
28	FS	(File separator)	60	<	92	\	124		28 1C	034	File Separator	60 3C	074	<	<	92 5C	134	\	\	124 7C	174	|	
29	GS	(Group separator)	61	=	93]	125	}	29 1D	035	Group Separator	61 3D	075	=	=	93 5D	135]]	125 7D	175	}	}
30	RS	(Record separator)	62	>	94	^	126	_	30 1E	036	Record Separator	62 3E	076	>	>	94 5E	136	^	^	126 7E	176	~	_
31	US	(Unit separator)	63	?	95	-			31 1F	037	Unit Separator	63 3F	077	?	?	95 5F	137	_	Del	127 7F	177		Del
127	DEL	(Delete)																					

0 0	000	NULL	32 20	040	 	Space	64 40	100	@	@	96 60	140	`	`
1 1	001	Start of Header	33 21	041	!	!	65 41	101	A	A	97 61	141	a	a
2 2	002	Start of Text	34 22	042	"	"	66 42	102	B	B	98 62	142	b	b
3 3	003	End of Text	35 23	043	#	#	67 43	103	C	C	99 63	143	c	c
4 4	004	End of Transmission	36 24	044	$	\$	68 44							

Tipos de Comunicação:

Comunicação Serial

Direção da Transferência dos Dados

a) SIMPLEX



b) HALF-DUPLEX



c) FULL-DUPLEX



Tipos de Comunicação:

Tipos de Comunicação Serial

Síncrona: um sinal de clock em separado é associado com o dado.

Assíncrona: não existe sincronismo entre transmissor e receptor - a re-sincronização é feita caractere por caractere.

Obs. USART (Universal Synchronous/Asynchronous Receiver/Transmitter)

Taxas de Comunicação serial:

Taxa	Tempo por Bit	Taxa	Tempo por Bit
110 Baud	9.1 ms	19200 Baud	52 µs
150 Baud	6.66 ms	38400 Baud	26 µs
300 Baud	3.33 ms	57600 Baud	17 µs
600 Baud	1.66 ms	115200 Baud	8.68 µs
1200 Baud	833 µs	230400 Baud	4.34 µs
2400 Baud	416 µs	460800 Baud	2.17 µs
4800 Baud	208 µs	921600 Baud	1.08 µs
9600 Baud	104 µs	1 Mbps Baud	1 µs

Comunicação Serial

Entendendo as Interfaces de Comunicação Serial

As interfaces de comunicação serial são utilizadas para transmitir dados entre dispositivos de forma eficiente e com uso mínimo de cabos. Em contraste com a comunicação paralela, onde vários bits são transmitidos simultaneamente, a comunicação serial envia os dados bit a bit.

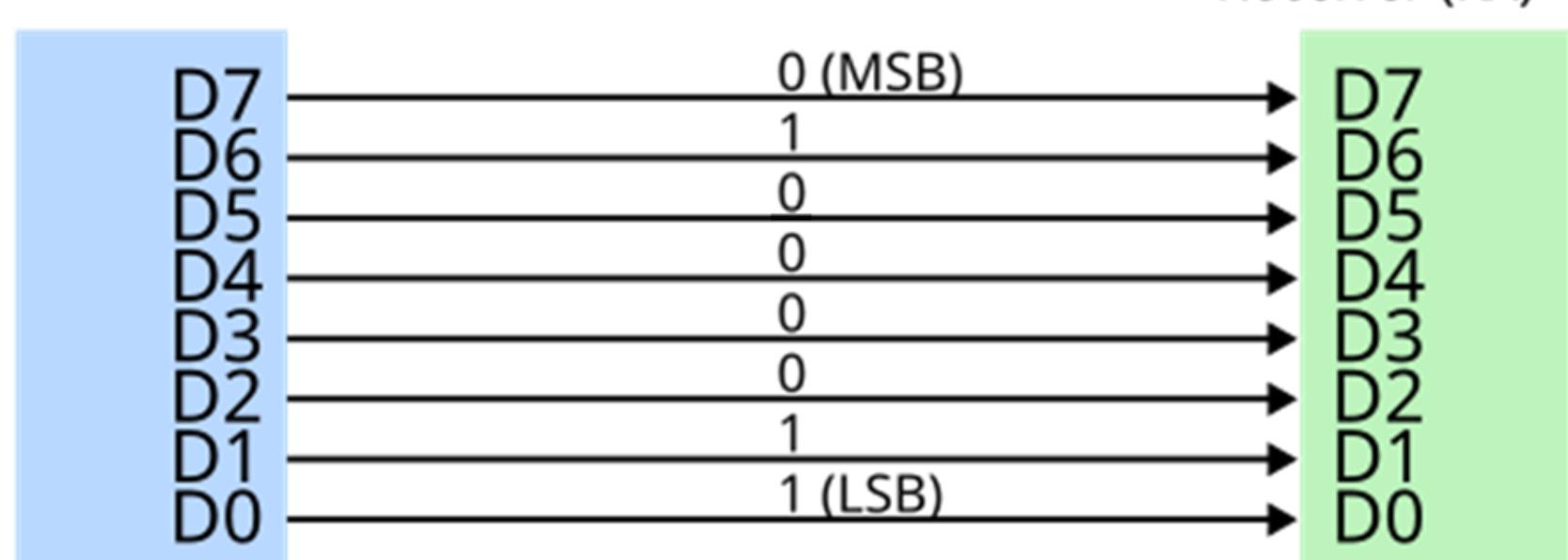
Por Que Usar Comunicação Serial?

- **Eficiência:** Menor quantidade de cabos e conexões.
- **Flexibilidade:** Pode ser usada em longas distâncias e em ambientes com alta interferência.
- **Simplicidade:** Configuração e implementação diretas.

Bin	Oct	Dec	Hex	Sinal
0100	103	67	43	C
0011				

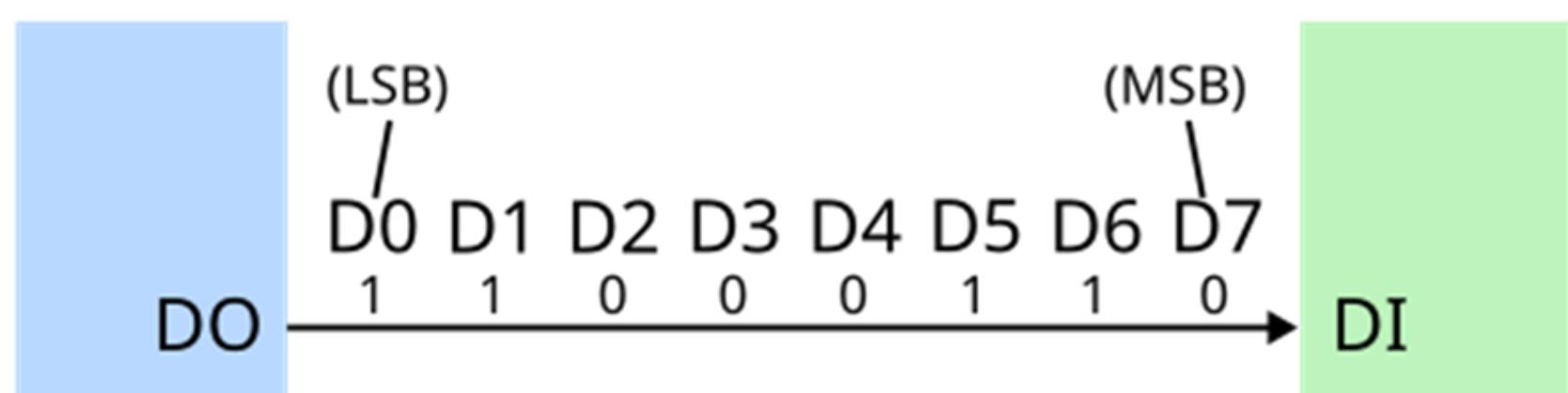
Parallel interface example

Transmitter (TX)



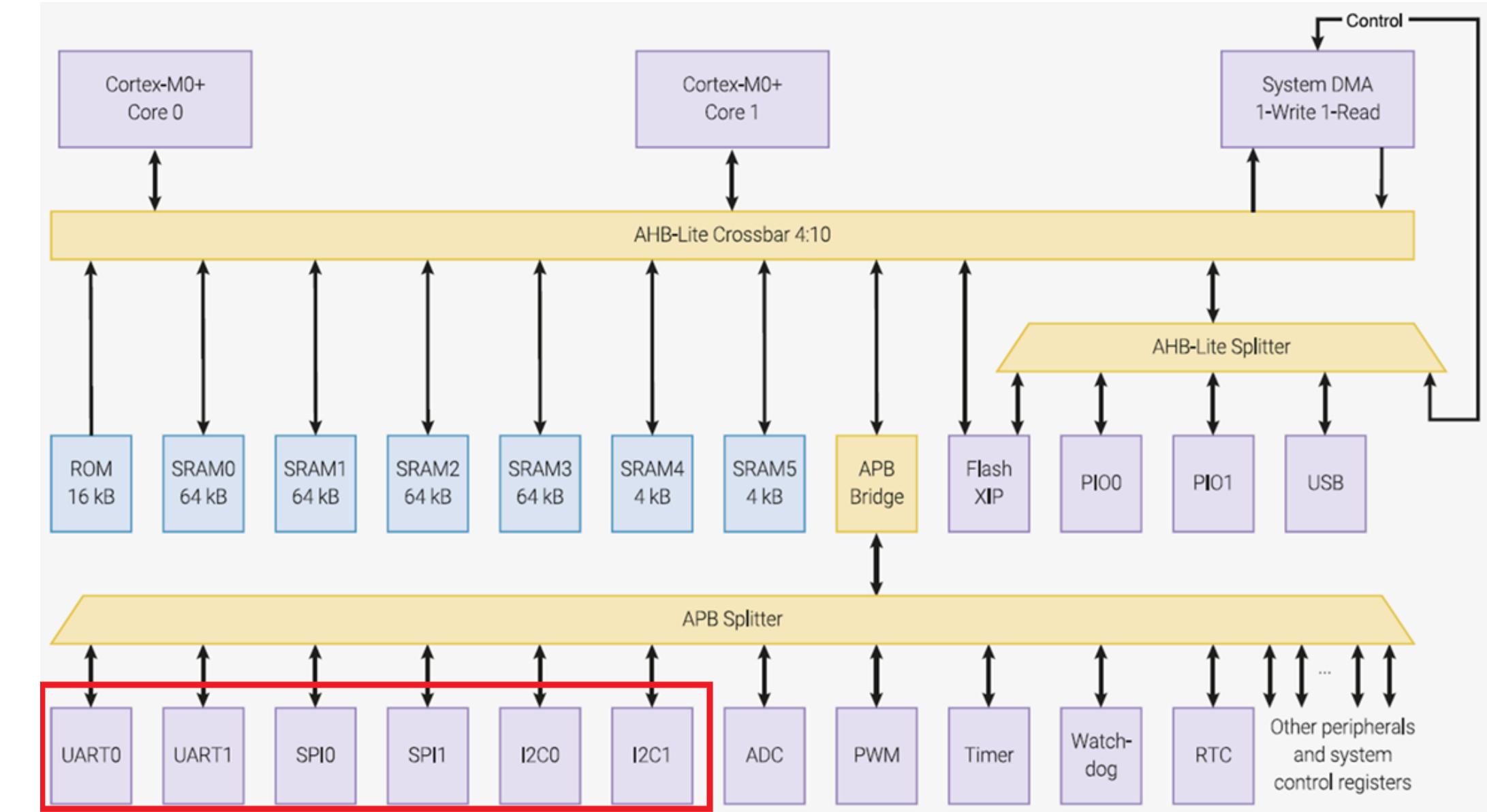
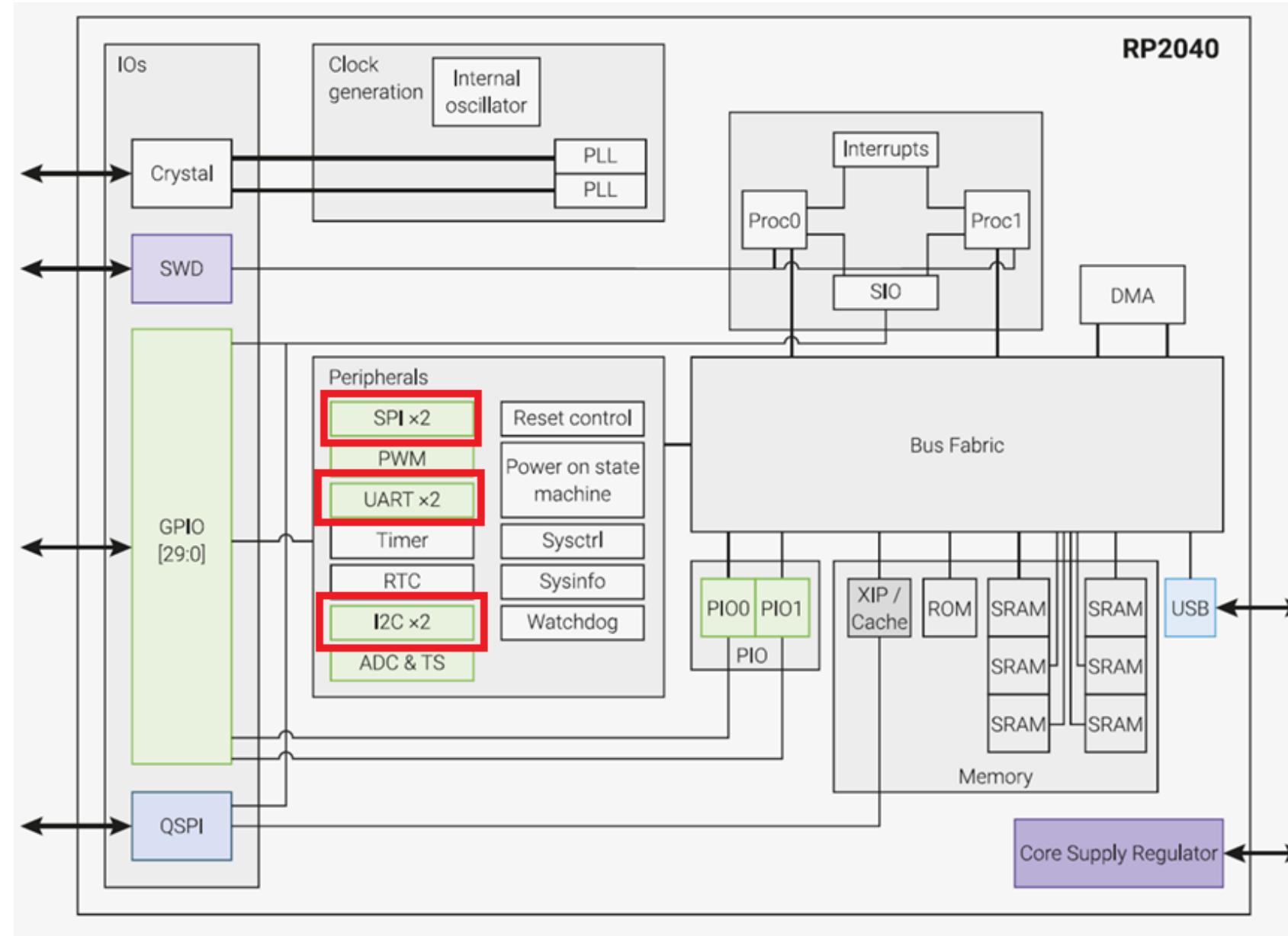
Serial interface example

Transmitter (TX)



Interfaces de Comunicação Serial

UART, SPI e I2C no RP2040.



Comunicação Serial Assíncrona

O protocolo serial assíncrono possui diversas regras embutidas — mecanismos que garantem transferências de dados confiáveis e sem erros. Esses mecanismos, que substituem a necessidade de um sinal de clock externo, incluem:

- **Bits de dados,**
- **Bits de sincronização,**
- **Bits de paridade,**
- **Taxa de saída “Baud”.**

Devido à flexibilidade desses mecanismos, não há um único padrão para enviar dados de forma serial. O protocolo é altamente configurável, e o aspecto mais importante é garantir que **ambos os dispositivos conectados ao barramento serial estejam configurados para usar exatamente os mesmos parâmetros de protocolo.**

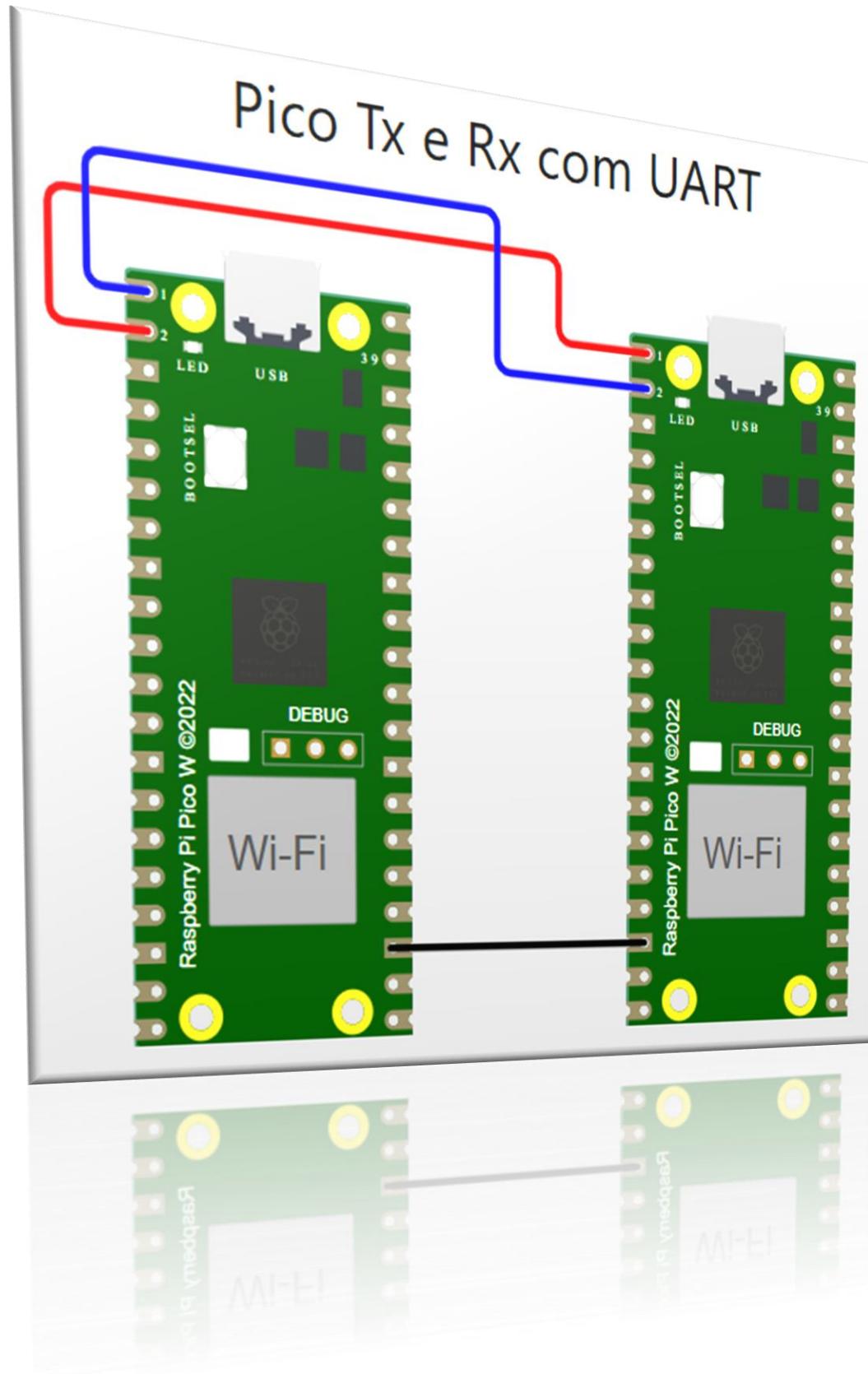
Comunicação Serial UART

UART - Universal Asynchronous Receiver/Transmitter

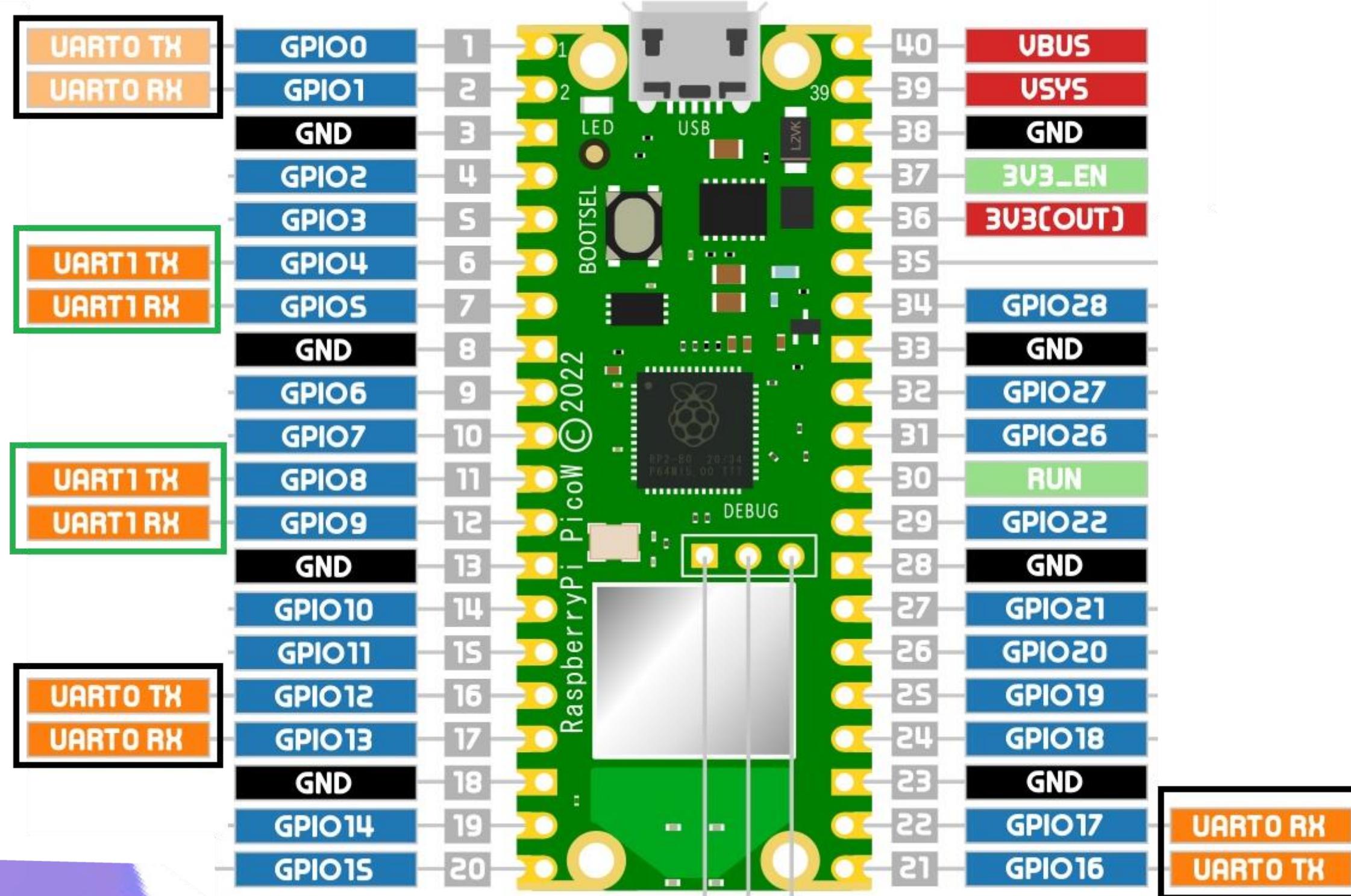
A UART é uma interface de comunicação assíncrona que permite a troca de dados entre dois dispositivos. Ela é frequentemente utilizada em módulos Bluetooth, GPS e para depuração de código.

Funcionamento da UART

UART utiliza apenas dois fios principais: **TX** (transmissor) e **RX** (receptor), além do fio de **GND** (terra). A **comunicação é assíncrona**, o que significa que não utiliza um sinal de clock compartilhado. Isso simplifica a implementação, mas requer que os dispositivos estejam configurados com a mesma taxa de transmissão, conhecida como **baud rate**.



Comunicação serial UART

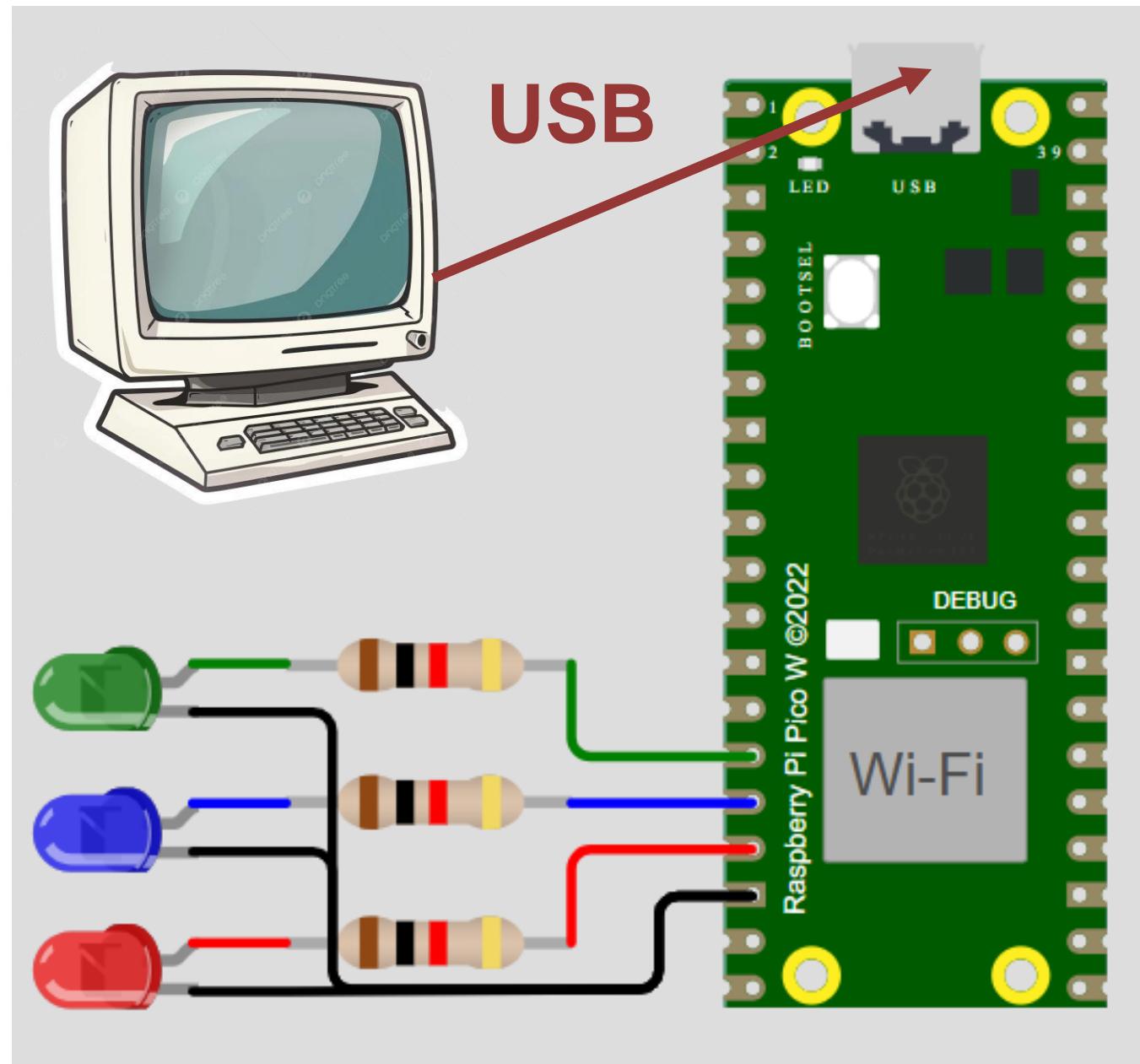
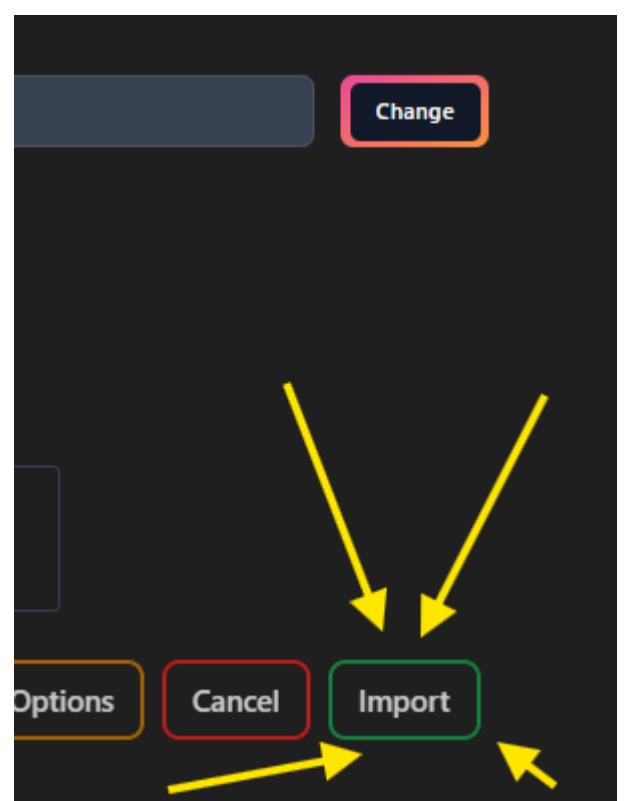
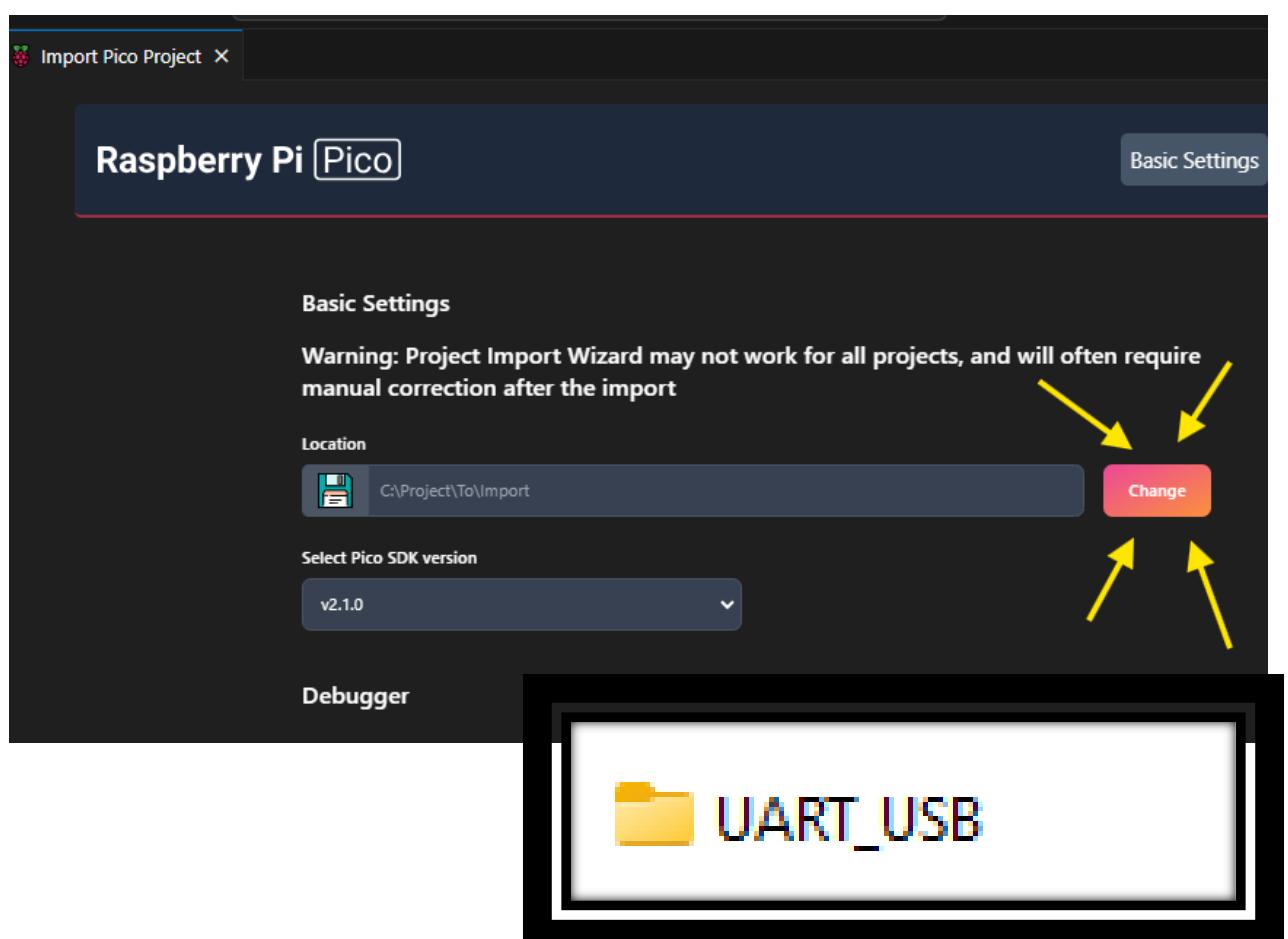
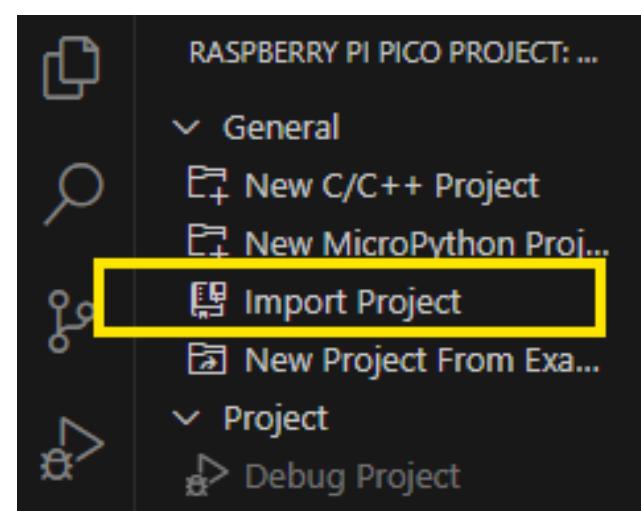


Linhas UART no RP2

- **TX** (Transmissor)
- **RX** (Receptor)

Exemplo de Comunicação UART.

**Na extensão do RP2 vá em “Import Project”,
depois selecione a pasta e finalmente clicar em “Import”**



**Neste exemplo os LEDs são controlados
pelo envio de mensagens (“r”, “g” e “b”)
pela comunicação serial USB CDC.**

Vá no: github.com/wiltonlacerda
git clone <https://github.com/wiltonlacerda/EmbarcaTechU4C6.git>

Exemplo: LED RGB na UART USB

```
#include <stdio.h>
#include "pico/stdc.h"
#define led_pin_g 11
#define led_pin_b 12
#define led_pin_r 13

int main()
{
    stdio_init_all(); // Inicializa comunicação USB CDC
    // Configura os pinos dos LEDs como saída
    gpio_init(led_pin_r);
    gpio_set_dir(led_pin_r, GPIO_OUT);
    gpio_put(led_pin_r, 0); // Inicialmente desligado

    gpio_init(led_pin_g);
    gpio_set_dir(led_pin_g, GPIO_OUT);
    gpio_put(led_pin_g, 0); // Inicialmente desligado

    gpio_init(led_pin_b);
    gpio_set_dir(led_pin_b, GPIO_OUT);
    gpio_put(led_pin_b, 0); // Inicialmente desligado

    printf("RP2040 inicializado. Envie 'r', 'g' ou 'b'\n");
    para alternar os LEDs.\n");
}
```

Pasta: UART_USB

```
while (true)
{ if (stdio_usb_connected())
{ char c;
    if (scanf("%c", &c) == 1)
    { // Lê caractere da entrada padrão
        printf("Recebido: '%c'\n", c);
        switch (c)
        {
            case 'r':
                gpio_put(led_pin_r, !gpio_get(led_pin_r));
                printf("LED vermelho alternado!\n");
                break;
            case 'g':
                gpio_put(led_pin_g, !gpio_get(led_pin_g));
                printf("LED verde alternado!\n");
                break;
            case 'b':
                gpio_put(led_pin_b, !gpio_get(led_pin_b));
                printf("LED azul alternado!\n");
                break;
            default:
                printf("Comando inválido: '%c'\n", c);
        }
    } } sleep_ms(40); } return 0; }
```

Exemplo: LED RGB na UART USB

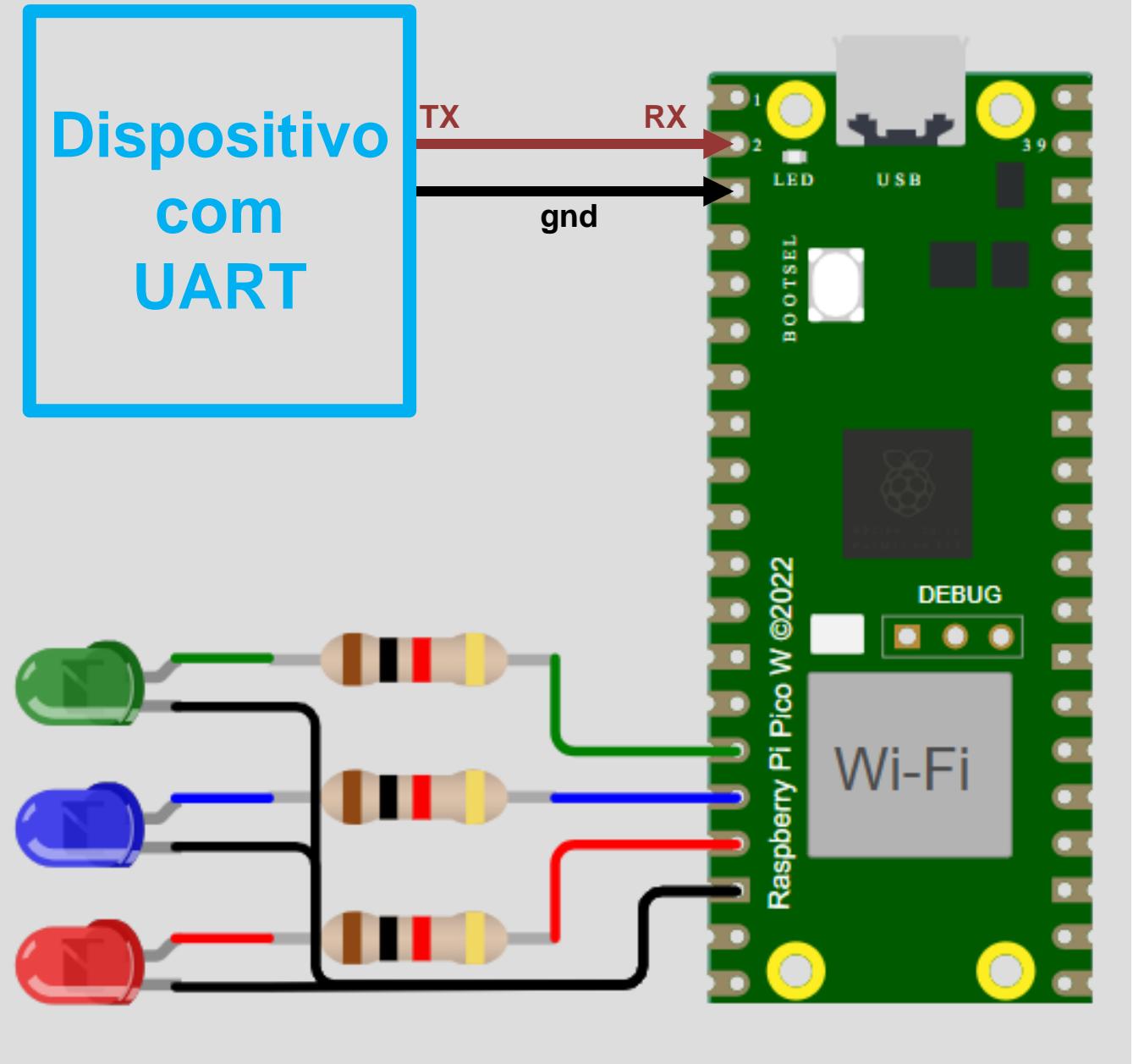
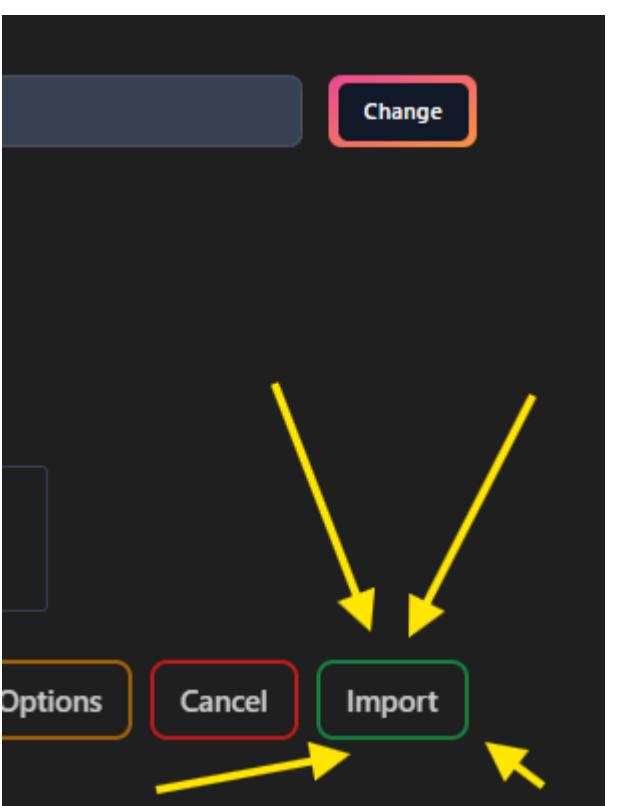
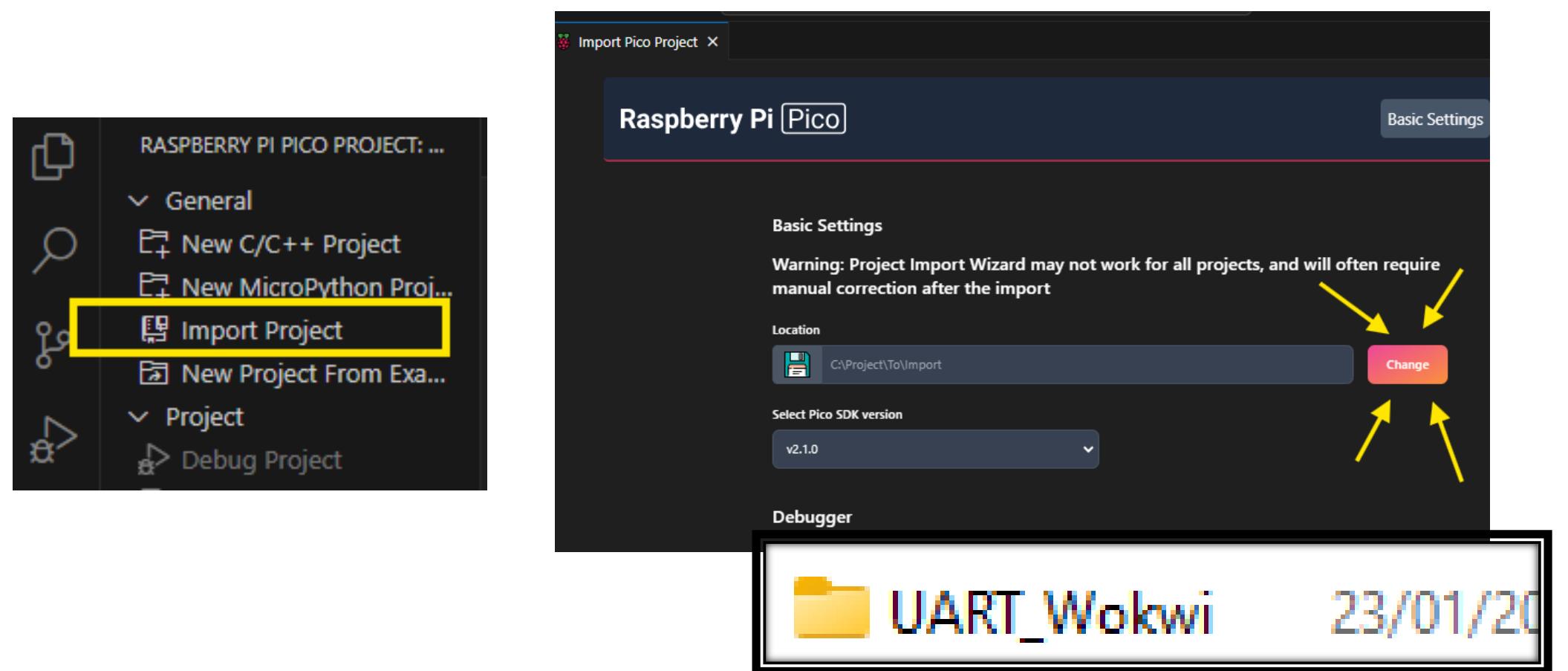
```
switch (c)
{
case 'r':
    gpio_put(led_pin_r, !gpio_get(led_pin_r));
    printf("LED vermelho alternado! \n");
break;
```

**Pasta:
UART_USB**

Caso o caractere recebido seja 'r', será lido o estado do led vermelho `gpio_get(led_pin_r)`. O seu valor será invertido “!”. O resultado será aplicado ao `gpio_put()`. Logo, se o led estiver aceso ele será apagado, e se estiver apagado ele será aceso.

Exemplo de Comunicação UART.

**Na extensão do RP2 vá em “Import Project”,
depois selecione a pasta e finalmente clicar em “Import”**



Neste exemplo os LEDs são controlados pelo envio de mensagens (“r”, “g” e “b”) pela comunicação serial **UART**.

Vá no: github.com/wiltonlacerda
git clone <https://github.com/wiltonlacerda/EmbarcaTechU4C6.git>

Exemplo: LED RGB na UARTO

```
#include <stdio.h>
#include "pico/stdlib.h"

#define UART_ID uart0 // Seleciona a UART0
#define BAUD_RATE 115200 // Define a taxa de transmissão
#define UART_TX_PIN 0 // Pino GPIO usado para TX
#define UART_RX_PIN 1 // Pino GPIO usado para RX
#define led_pin_g 11
#define led_pin_b 12
#define led_pin_r 13

int main() {
    // Inicializa a biblioteca padrão
    stdio_init_all();

    // Inicializa a UART
    uart_init(UART_ID, BAUD_RATE);

    // Configura os pinos GPIO para a UART
    gpio_set_function(UART_TX_PIN, GPIO_FUNC_UART);
    gpio_set_function(UART_RX_PIN, GPIO_FUNC_UART);

    const char *init_message = "UART Demo - RP2\r\n"
                               "Digite algo e veja o eco:\r\n";
    uart_puts(UART_ID, init_message);

    gpio_init(led_pin_g);
    gpio_set_dir(led_pin_g, GPIO_OUT);
    gpio_init(led_pin_b);
    gpio_set_dir(led_pin_b, GPIO_OUT);
    gpio_init(led_pin_r);
    gpio_set_dir(led_pin_r, GPIO_OUT);
```

Pasta:
UART_WOKWI

```
while (1) {
    // Verifica se há dados disponíveis para leitura
    if (uart_is_readable(UART_ID)) {
        // Lê um caractere da UART
        char c = uart_getc(UART_ID);
        if (c == 'r'){
            gpio_put(led_pin_r, !gpio_get(led_pin_r));
        }
        if (c == 'g'){
            gpio_put(led_pin_g, !gpio_get(led_pin_g));
        }
        if (c == 'b'){
            gpio_put(led_pin_b, !gpio_get(led_pin_b));
        }
        // Envia de volta o caractere lido (eco)
        uart_putc(UART_ID, c);
        // Envia uma mensagem adicional para cada caractere recebido
        uart_puts(UART_ID, " <- Eco do RP2\r\n");
    }
    sleep_ms(40);
}
return 0;
```

Exemplo: LED RGB na UART USB

```
...  
uart_init(UART_ID, BAUD_RATE);  
...  
if (uart_is_readable(UART_ID)) {  
    // Lê um caractere da UART  
    char c = uart_getc(UART_ID);  
    if (c == 'r'){  
        gpio_put(led_pin_r, !gpio_get(led_pin_r));  
    }  
}
```

Pasta:
UART_WOKWI

`uart_init(UART_ID, BAUD_RATE);`. Aqui a UART é inicializada pelo padrão (115200 dada pela variável BAUD_RATE, 8 bits de dados, 1 bit de parada e SEM paridade).

Caso tenha caracteres disponíveis, `uart_is_readable(UART_ID)` torna-se verdadeiro. `c = uart_getc(UART_ID);` recebe o caractere lido e passa para os comandos de decisão ifs. Logo, se o led estiver aceso ele será apagado, e se estiver apagado ele será aceso.

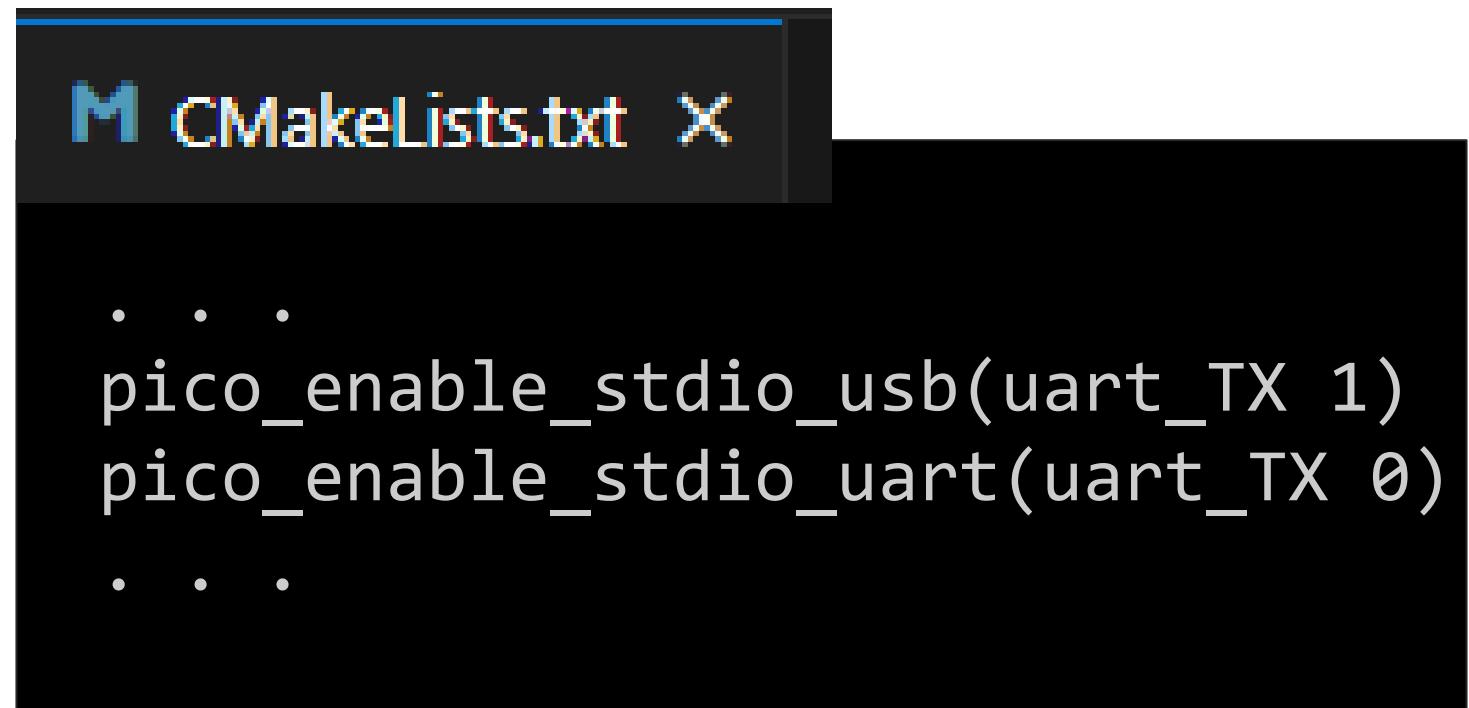
Observações do CMakeLists.txt

pico_enable_stdio_usb(uart_usb 1)

- Esta linha habilita a saída padrão (**STDIO**) para a interface **USB CDC** (também conhecida como "USB Virtual COM Port") no projeto.
- Quando você define o valor como 1 (ex.: pico_enable_stdio_usb(1)), a entrada e saída padrão (como **printf** e **scanf**) são redirecionadas para o USB.
- A função **pico_enable_stdio_usb()** ativa a comunicação USB para os comandos como **printf**, permitindo que você veja saídas no **Serial Monitor** via cabo USB.
- Essa configuração é necessária para que a saída padrão (e entrada padrão, se usada) seja roteada para o USB CDC no RP2040.

pico_enable_stdio_uart(uart_usb 1)

- Esta linha habilita a saída padrão (**STDIO**) para a interface **UART** no projeto.
- A função **pico_enable_stdio_uart()** define se o STDIO será roteado pela interface UART0 do RP2040 (padrão nos GPIO 0 e 1).
- É útil quando o Pico está conectado a outro dispositivo por UART



```
M CMakeLists.txt X
...
pico_enable_stdio_usb(uart_TX 1)
pico_enable_stdio_uart(uart_TX 0)
...
```

Quando Usar USB ou UART?

USB (CDC):

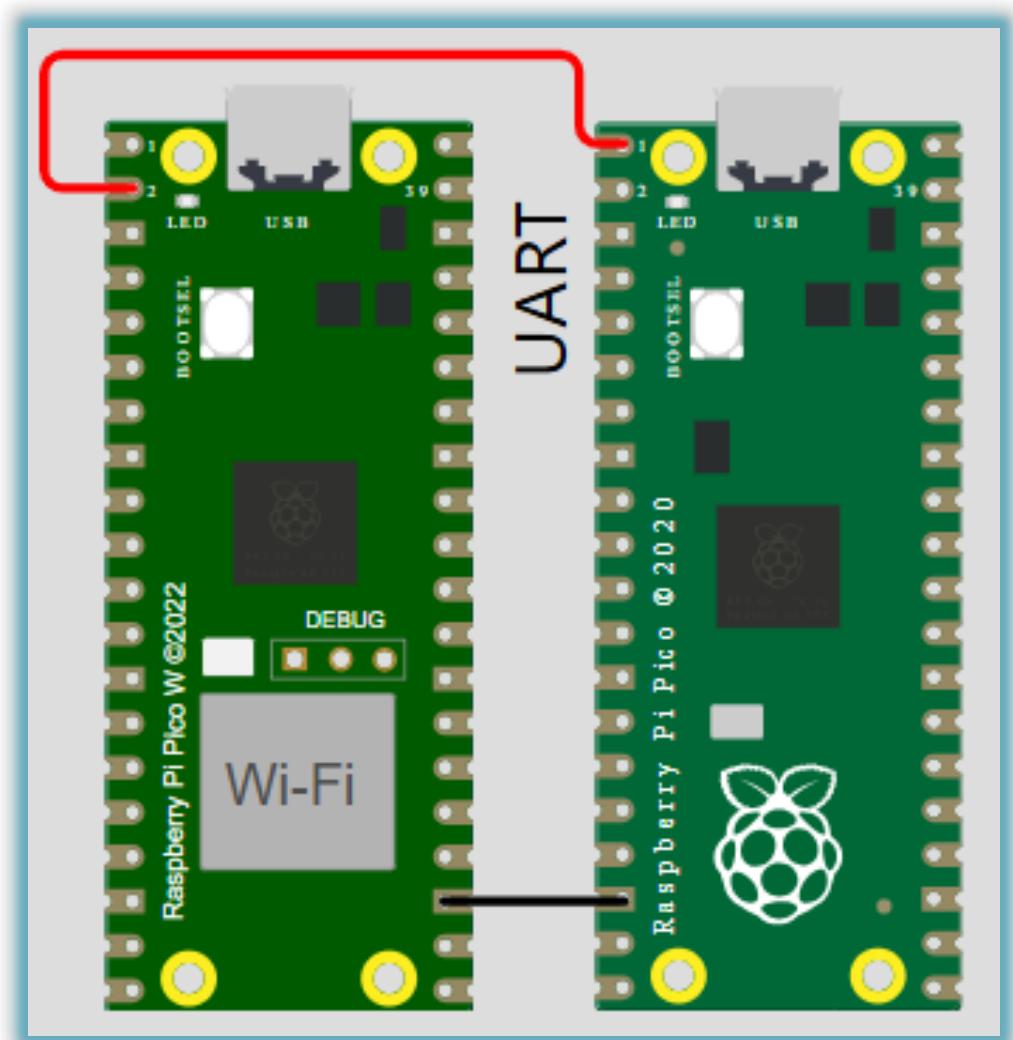
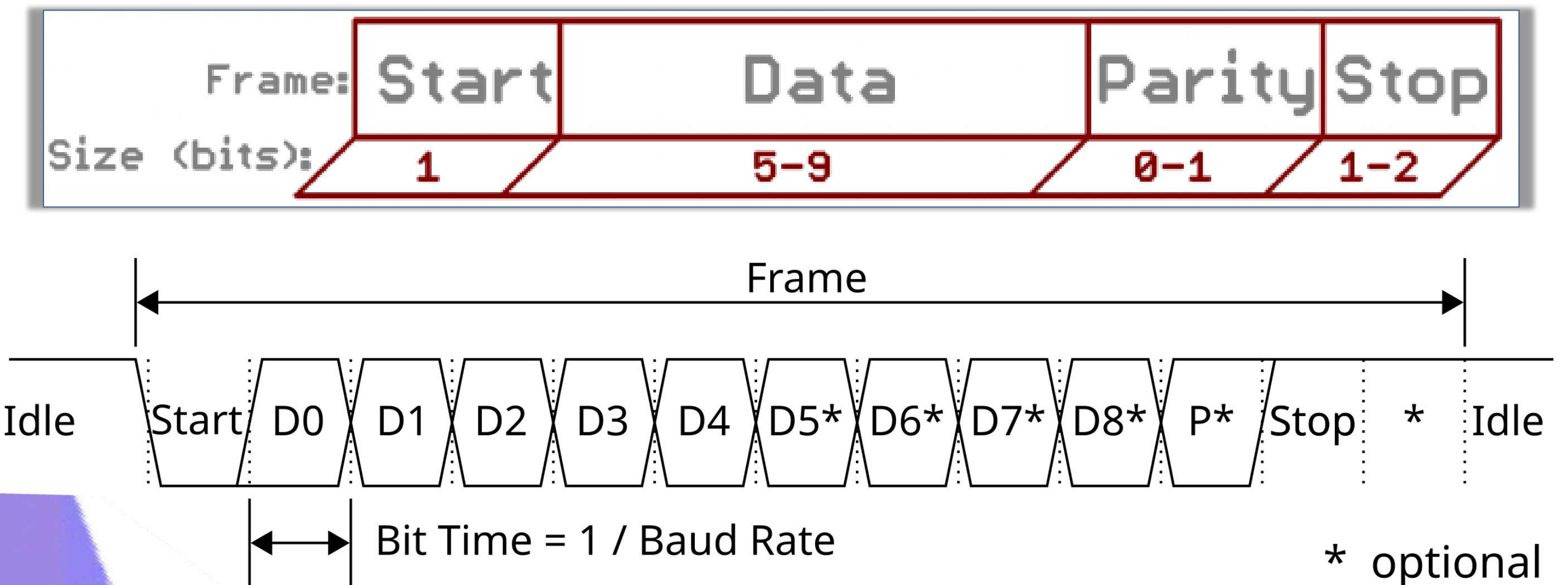
- Ideal para desenvolvimento, pois você só precisa de um cabo USB.
- O STDIO funciona diretamente no Serial Monitor sem pinos adicionais.

UART:

- Útil em casos onde o RP2040 precisa se comunicar diretamente com outro dispositivo serial (outro microcontrolador, sensores ou computadores via adaptador USB-Serial).
- Útil quando o **USB do RP2040** está sendo usado para outras funções, como armazenamento em massa ou dispositivo HID.

Formatação do Frame UART

Cada bloco (geralmente um byte) de dados transmitido é, na verdade, enviado em um pacote ou quadro de bits. Os quadros são criados adicionando **bits de sincronização** e **bits de paridade** aos dados. A quantidade de **dados** em cada pacote pode variar **entre 5 a 9 bits**. O bit de início (**start bit**) é sempre indicado por uma linha de dados em repouso (**IDLE**) que muda **de 1 para 0**, enquanto o bit(s) de parada (stop bit(s)) retornará ao estado de repouso mantendo a linha em **1**. A paridade (**parity**) é uma forma muito simples e de baixo nível de verificação de erros. Ela pode ser de dois tipos: **ímpar** (odd) ou **par** (even). Finalmente, basta escolher a ordem de envio: (**MSB**, most-significant bit) é enviado primeiro, ou vice-versa.



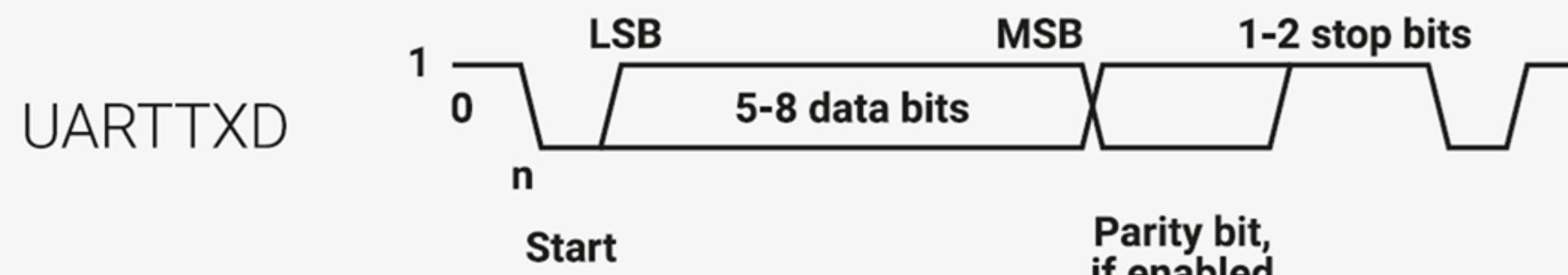
Formatação do Frame UART

Informação retirada do “rp2040-datasheet.pdf”.

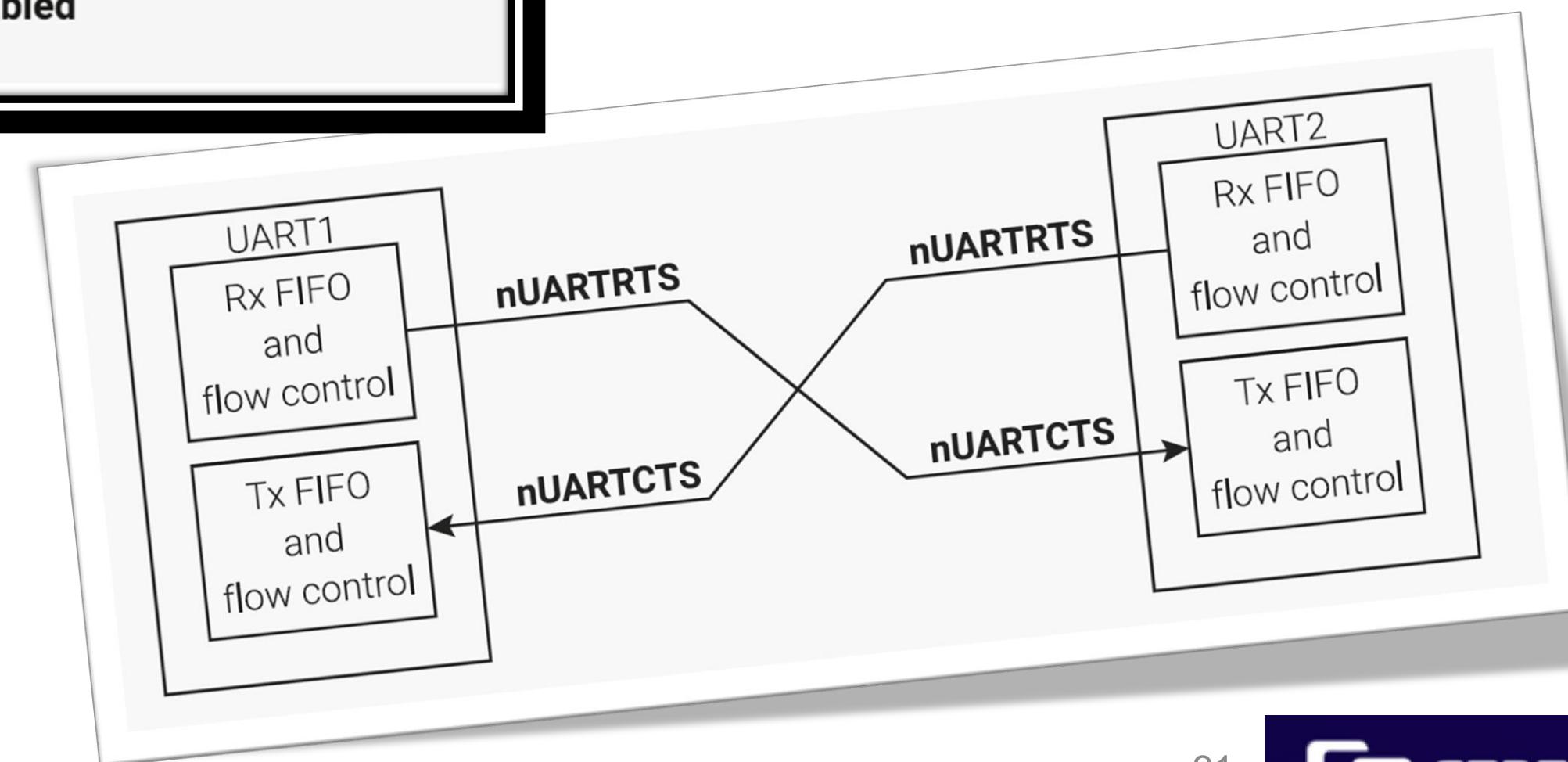
4.2.3.3. UART character frame

Página 421

Figure 61. **UART** character frame.

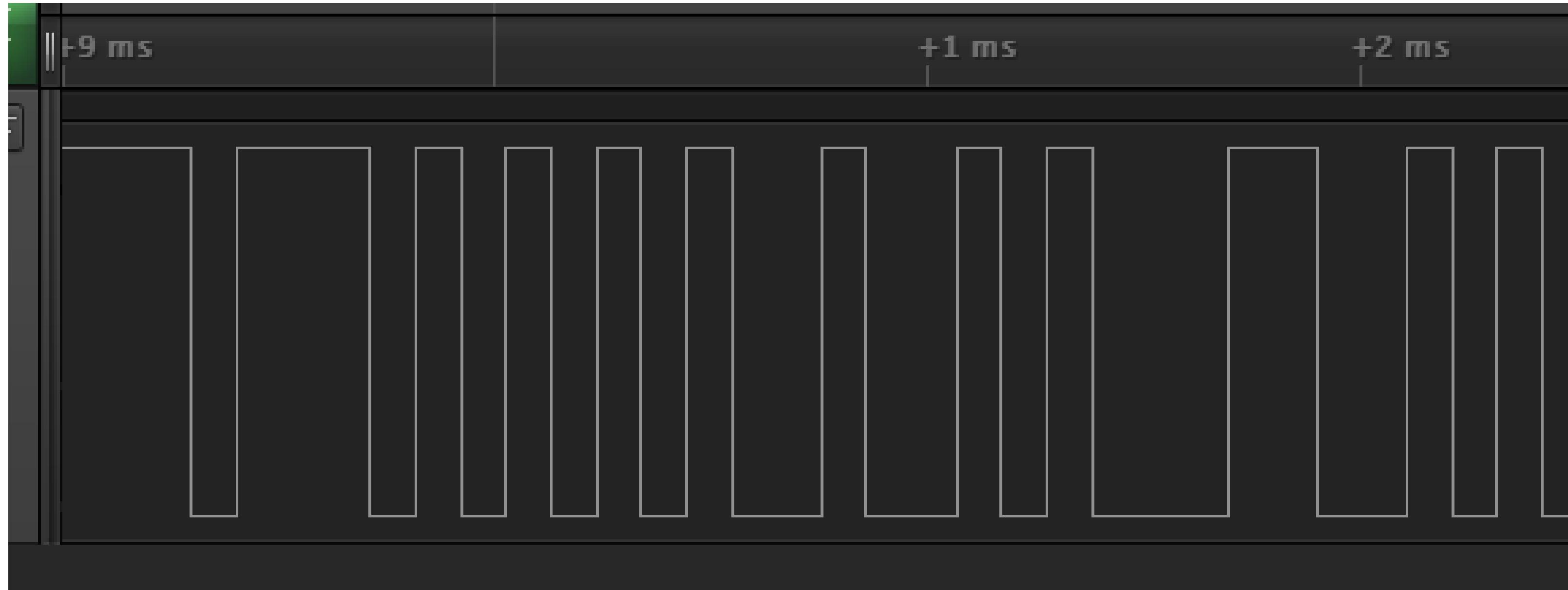


Supports a **maximum baud rate** of $\text{UARTCLK} / 16$ in **UART mode** (**7.8 Mbaud** at 125MHz) pg.416



Exemplo: O que foi enviado aqui:

8N1(8bits, Sem Paridade e 1 Stop Bit)



Analisador Lógico

docs.wokwi.com/pt-BR/guides/logic-analyzer

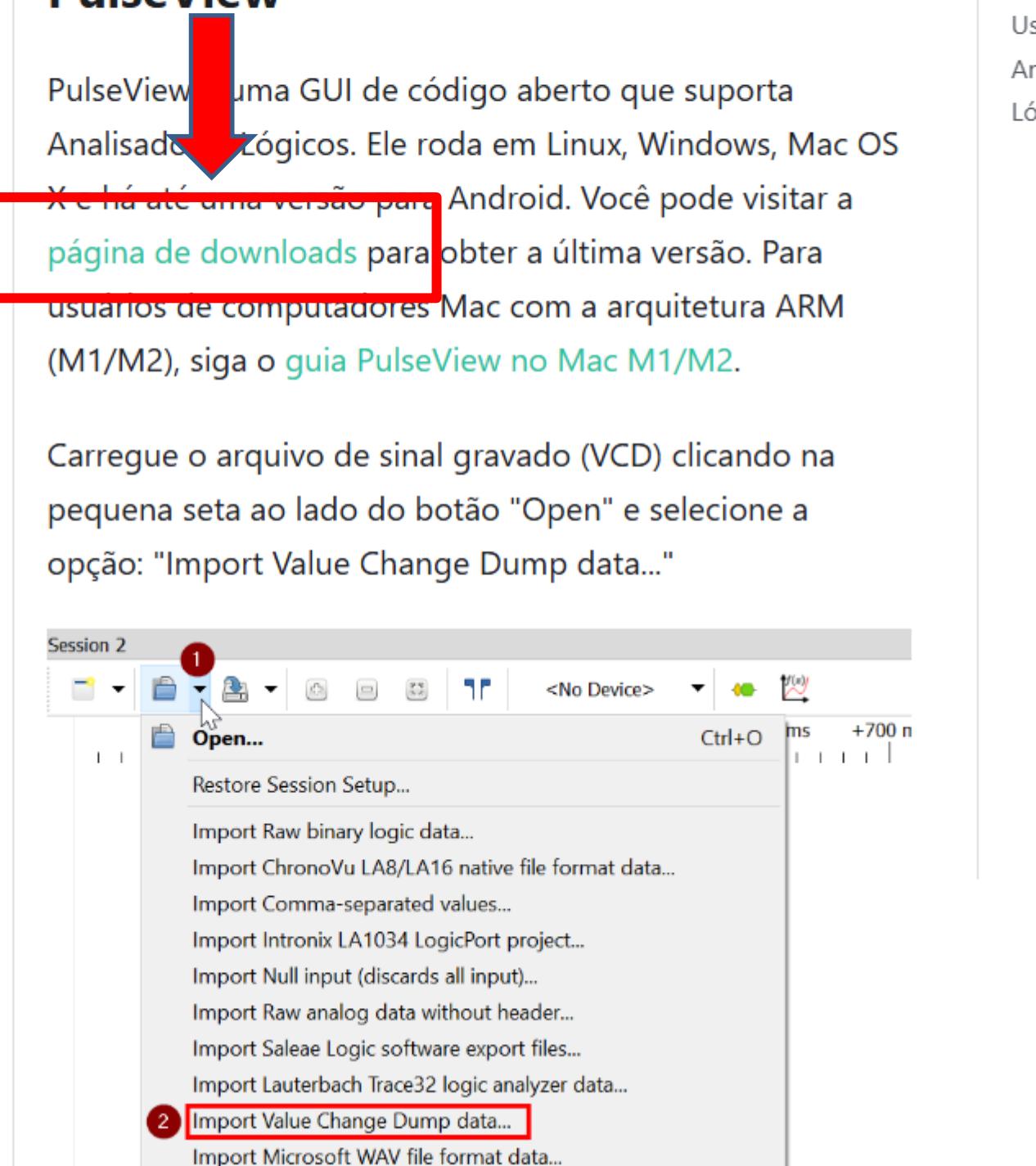
Wokwi Docs Português (Brasil) Blog Simulador Procurar

Começando >
Guias >
Editor de Diagrama
Teclas de Atalho
Interactive Debugger
Depurador
Monitor Serial
Analisador Lógico
Bibliotecas
MicroPython
CircuitPython
Simulador ESP32
WiFi ESP32
Traduções
Referência do Diagrama >
Chips API >
VS Code Extension >
Wokwi CI >

Visualizando os dados no PulseView

PulseView é uma GUI de código aberto que suporta Analisadores Lógicos. Ele roda em Linux, Windows, Mac OS X e há até uma versão para Android. Você pode visitar a [página de downloads](#) para obter a última versão. Para usuários de computadores Mac com a arquitetura ARM (M1/M2), siga o [guia PulseView no Mac M1/M2](#).

Carregue o arquivo de sinal gravado (VCD) clicando na pequena seta ao lado do botão "Open" e selecione a opção: "Import Value Change Dump data..."



Session 2

- 1 Open...
- 2 Import Value Change Dump data...

Um **analisador lógico** é um instrumento utilizado para capturar, visualizar e analisar sinais digitais em circuitos eletrônicos. Ele é amplamente usado no desenvolvimento, depuração e diagnóstico de sistemas digitais, como microcontroladores, FPGAs, processadores e outros dispositivos digitais.



sigrok.org/wiki/Downloads

Pulse View

Log in

Page Discussion Read View source View history Search sigrok

Downloads

Linux Windows Mac OS X Other

Nightly builds (recommended, always up-to-date)

PulseView (32bit)	PulseView (32bit)	PulseView (64bit)	See below
PulseView (64bit)	PulseView (64bit)	sigrok-cli (64bit)	
sigrok-cli (32bit)	sigrok-cli (32bit)	sigrok-cli (64bit)	
sigrok-cli (64bit)	sigrok-cli (64bit)		

1. Open the "Downloads" page on sigrok.org.

2. Under the "Nightly builds (recommended, always up-to-date)" section, click on the "PulseView (64bit)" button.

3. A yellow circle highlights the "PulseView (64bit)" button, and a yellow arrow points to the "sigrok-cli (64bit)" button.

Exercício: TX UART contador

```
#include <stdio.h>
#include "pico/stdlib.h"

#define UART_ID uart0      // Seleciona a UART
#define BAUD_RATE 115200   // Define a taxa de transmissão
#define UART_TX_PIN 0      // Pino GPIO usado para TX
#define UART_RX_PIN 1      // Pino GPIO usado para RX
#define LED_PIN 13         // Pino GPIO usado para o LED
#define BUTTON_PIN 5        // Pino GPIO usado para o botão

static volatile uint16_t counter = 0;    // Variável inteira para contagem
static volatile uint32_t last_time = 0; // Armazena o tempo do último evento
static volatile bool flag_var = 0;      // Variável de flag para debouncing

static void gpio_irq_handler(uint gpio, uint32_t events);

int main()
{
    stdio_init_all();
    uart_init(UART_ID, BAUD_RATE);
    gpio_set_function(UART_TX_PIN, GPIO_FUNC_UART); // Configura o pino 0
para TX
    gpio_set_function(UART_RX_PIN, GPIO_FUNC_UART); // Configura o pino 1
para RX
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);
    gpio_init(BUTTON_PIN);
    gpio_set_dir(BUTTON_PIN, GPIO_IN);
    gpio_pull_up(BUTTON_PIN);

    gpio_set_irq_enabled_with_callback(BUTTON_PIN, GPIO_IRQ_EDGE_FALL, true,
&gpio_irq_handler);
}
```

Pasta: UART_PicoWL_TX

```
while (1)
{
    if (flag_var)          // Verifica se a flag está ativa
    {
        // Cria a mensagem com o valor atual do contador
        char message[50];
        sprintf(message, "Contagem: %d\n", counter);

        // Envia a mensagem pela UART
        uart_puts(UART_ID, message);
        // Pisca o LED
        gpio_put(LED_PIN, 1); sleep_ms(50);
        gpio_put(LED_PIN, 0); sleep_ms(50);
        flag_var = 0; // Limpa a flag de interrupção
    }
    sleep_ms(10);
}

// Função de interrupção com debouncing
void gpio_irq_handler(uint gpio, uint32_t events)
{
    // Obtém o tempo atual em microssegundos
    uint32_t current_time = to_us_since_boot(get_absolute_time());
    // Verifica se passou tempo suficiente desde o último evento
    if (current_time - last_time > 200000) // 50 ms de debouncing
    {
        last_time = current_time; // Atualiza o tempo do último evento
        counter++;                // Incrementa o contador
        flag_var = 1;
    }
}
```

Exercício: TX UART contador

Neste exemplo enviaremos dois caracteres. O "C" e o "o".

Em ASCII:

C => (67d) (43h) (0100.0011 b)

o => (111d) (6Fh) (0110.1111 b)

Name: **UART**

Color: **green**

UART

RX (UART receive line): **logic.D0**

TX (UART transmit line): **-**

Baud rate: **115200**

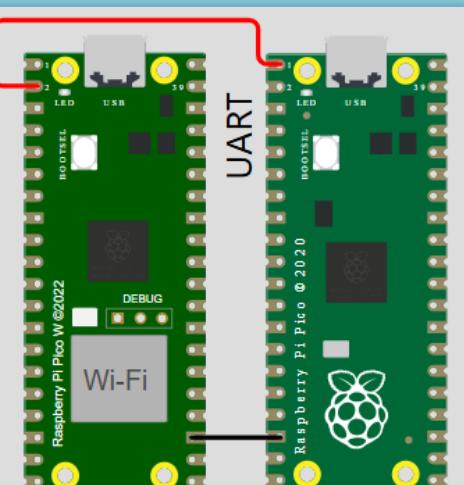
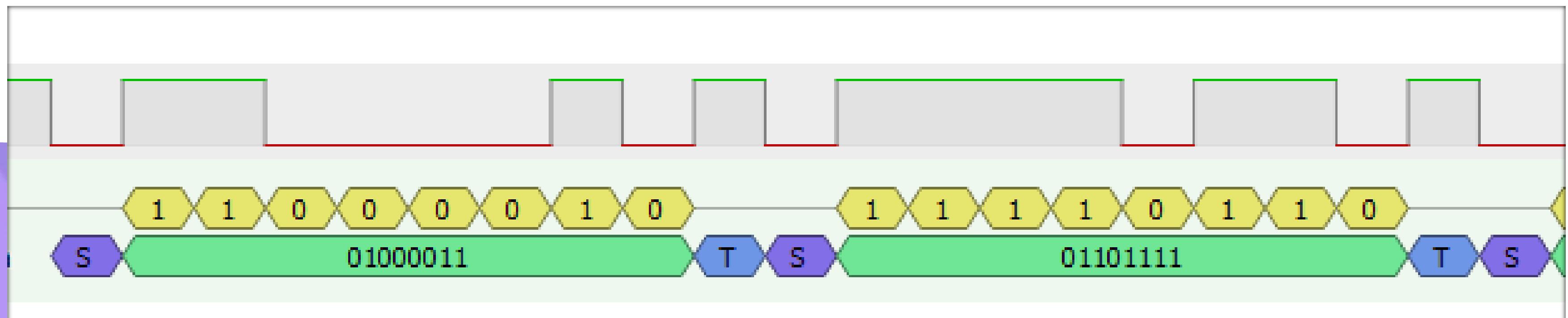
Data bits: **8**

Parity: **none**

Stop bits: **1.0**

Bit order: **lsb-first**

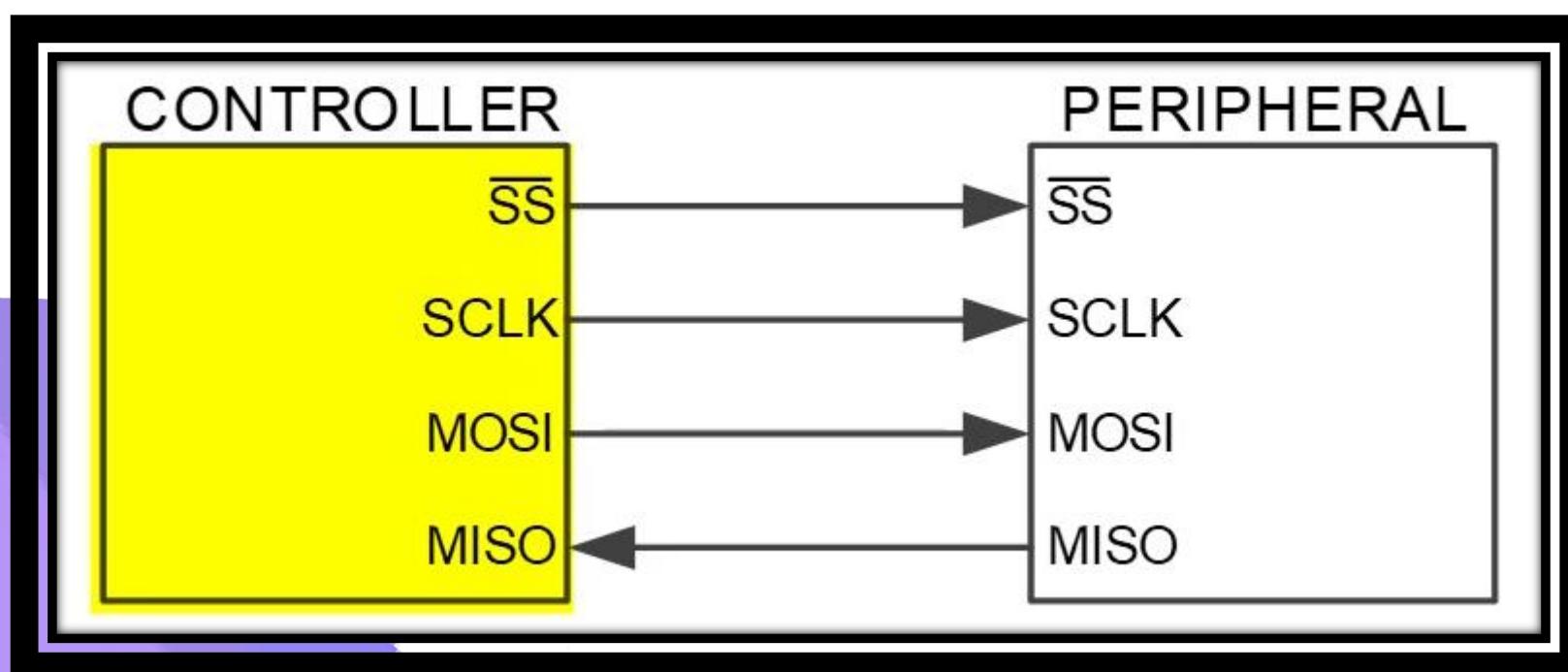
Data format: **bin**



Comunicação serial SPI

SPI (Serial Peripheral Interface)

O SPI é um protocolo de comunicação serial amplamente utilizado para conectar dispositivos eletrônicos, como microcontroladores, sensores, displays, memórias, conversores analógicos/digitais e outros periféricos. Ele é rápido, simples e eficiente, sendo ideal para trocas de dados de alta velocidade em curtas distâncias.



Características em destaque do SPI:

Comunicação mestre-escravo:

- Um dispositivo atua como mestre (geralmente um microcontrolador) e controla a comunicação com um ou mais escravos.

Síncrono:

- A comunicação é sincronizada por um relógio compartilhado (SCK).

Full-duplex:

- Permite enviar e receber dados simultaneamente.

Linhas de comunicação:

O SPI usa um conjunto mínimo de quatro fios:

- SCK** (Serial Clock): Gerado pelo mestre, sincroniza a transferência de dados.
- MOSI** (Master Out, Slave In): Linha de dados do mestre para o escravo.
- MISO** (Master In, Slave Out): Linha de dados do escravo para o mestre.
- SS** (Slave Select): **CS** (Chip Select) **CE** (Chip Enable)

Usada para selecionar o escravo ativo. Em sistemas com múltiplos escravos, cada um tem seu próprio pino SS.

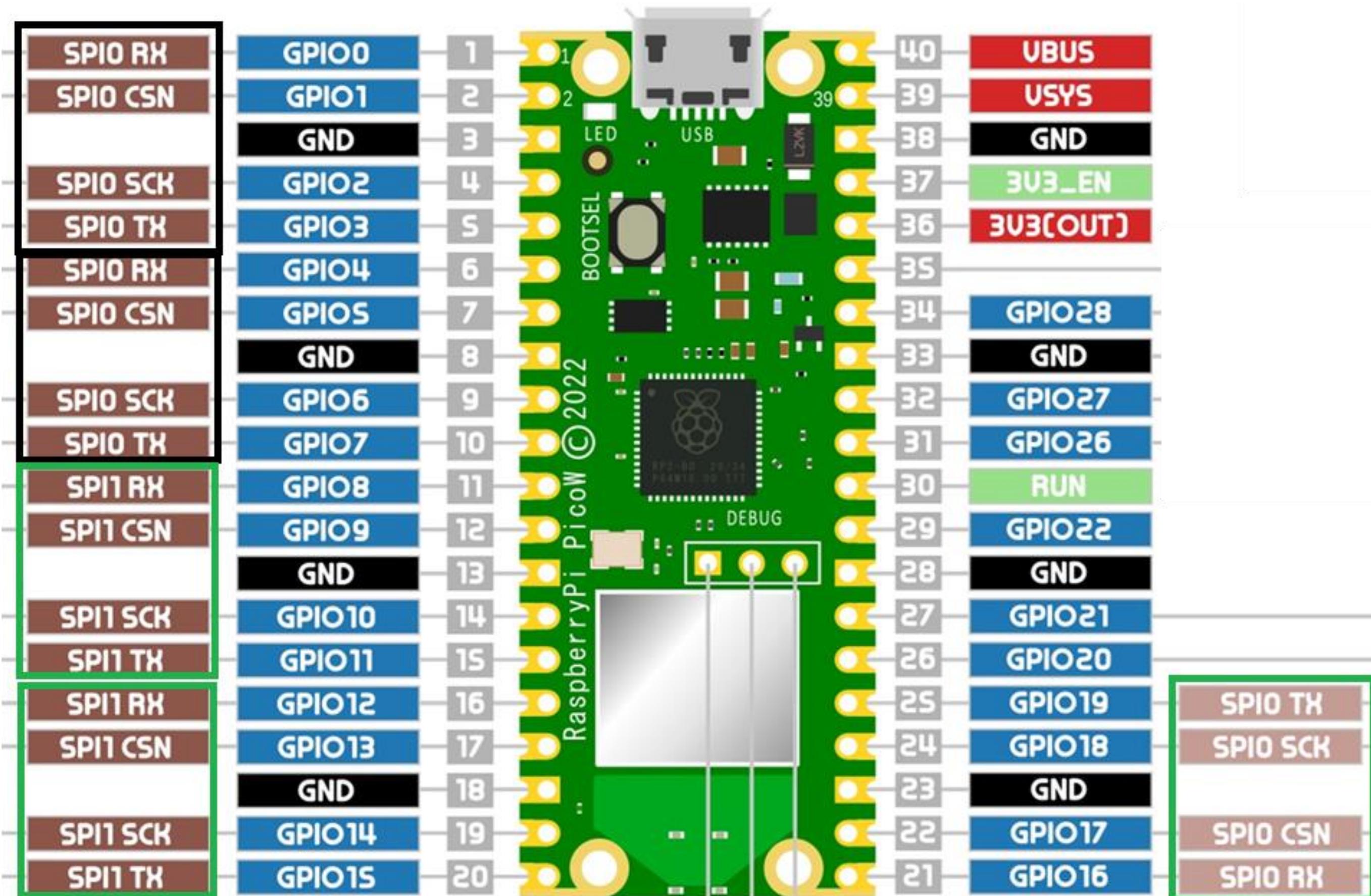
Velocidade configurável:

- A taxa de transferência de dados é determinada pelo mestre, com frequências que podem atingir dezenas de megahertz, dependendo do hardware.

Comunicação serial SPI

Linhas de SPI no RP2

- **SCK** (Serial Clock) = **SCK**
- **MOSI** (Master Out, Slave In) = **TX**
- **MISO** (Master In, Slave Out) = **RX**
- **SS** (Slave Select): **CSN**



Comunicação serial SPI

No SPI, apenas um lado gera o sinal de clock (geralmente chamado de CLK ou **SCK** para Serial Clock).

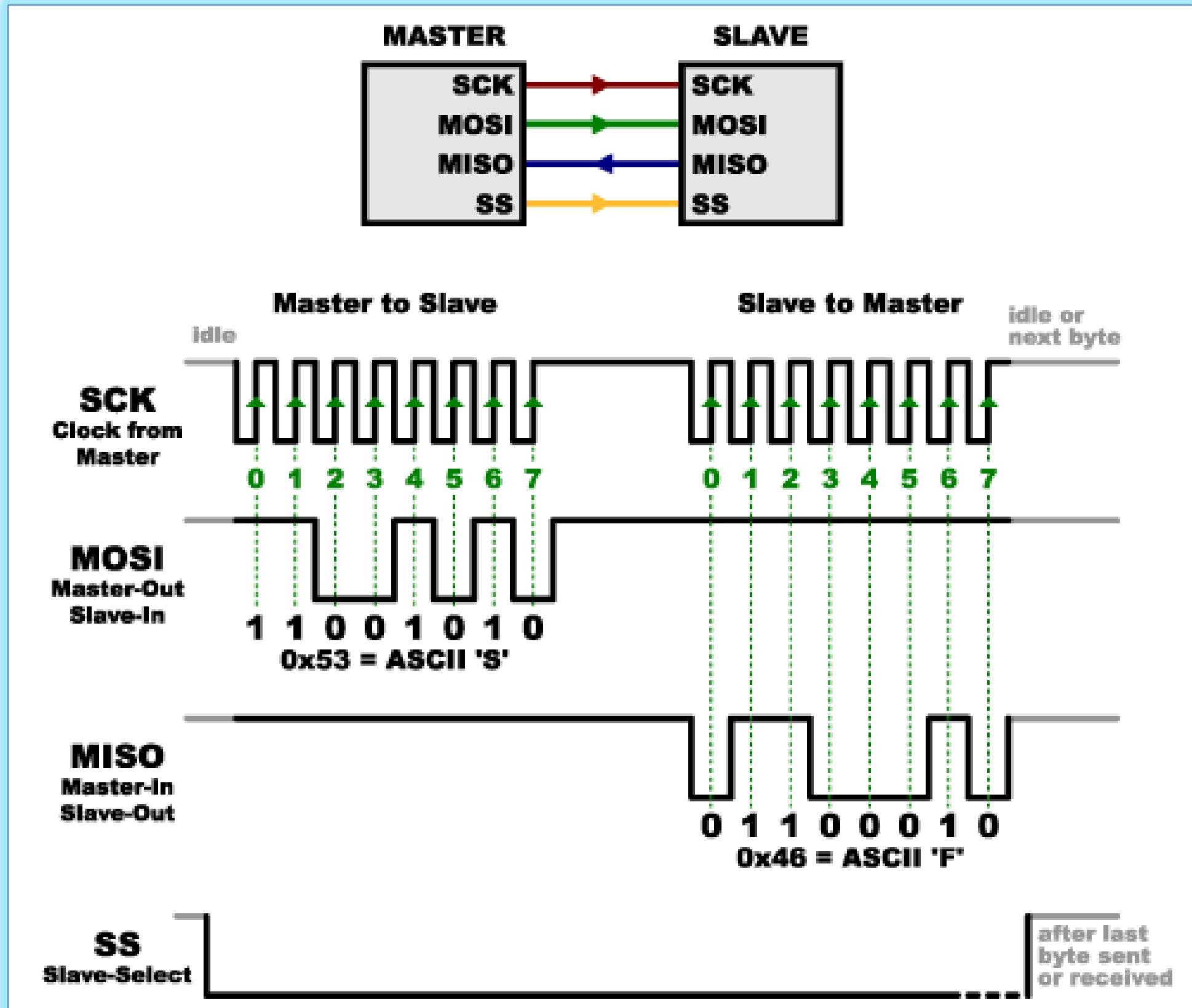
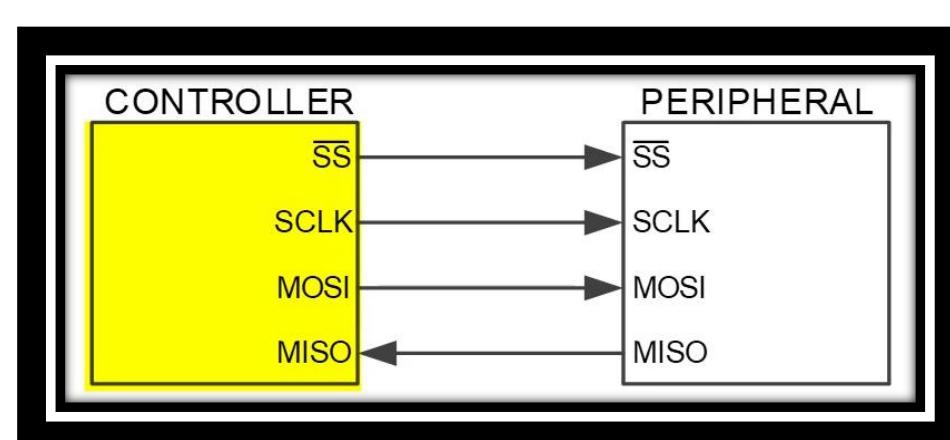
O lado que gera o clock é chamado de **controlador** (anteriormente "master"), e o outro lado é chamado de **periférico** (anteriormente "slave").

Há sempre apenas um controlador (geralmente o microcontrolador), mas podem existir **múltiplos periféricos**.

Quando dados são enviados do **controlador** para um **periférico**, eles são transmitidos por uma linha de dados chamada **MOSI** (Controller Out / Peripheral In ou também **Master Out e Slave In**).

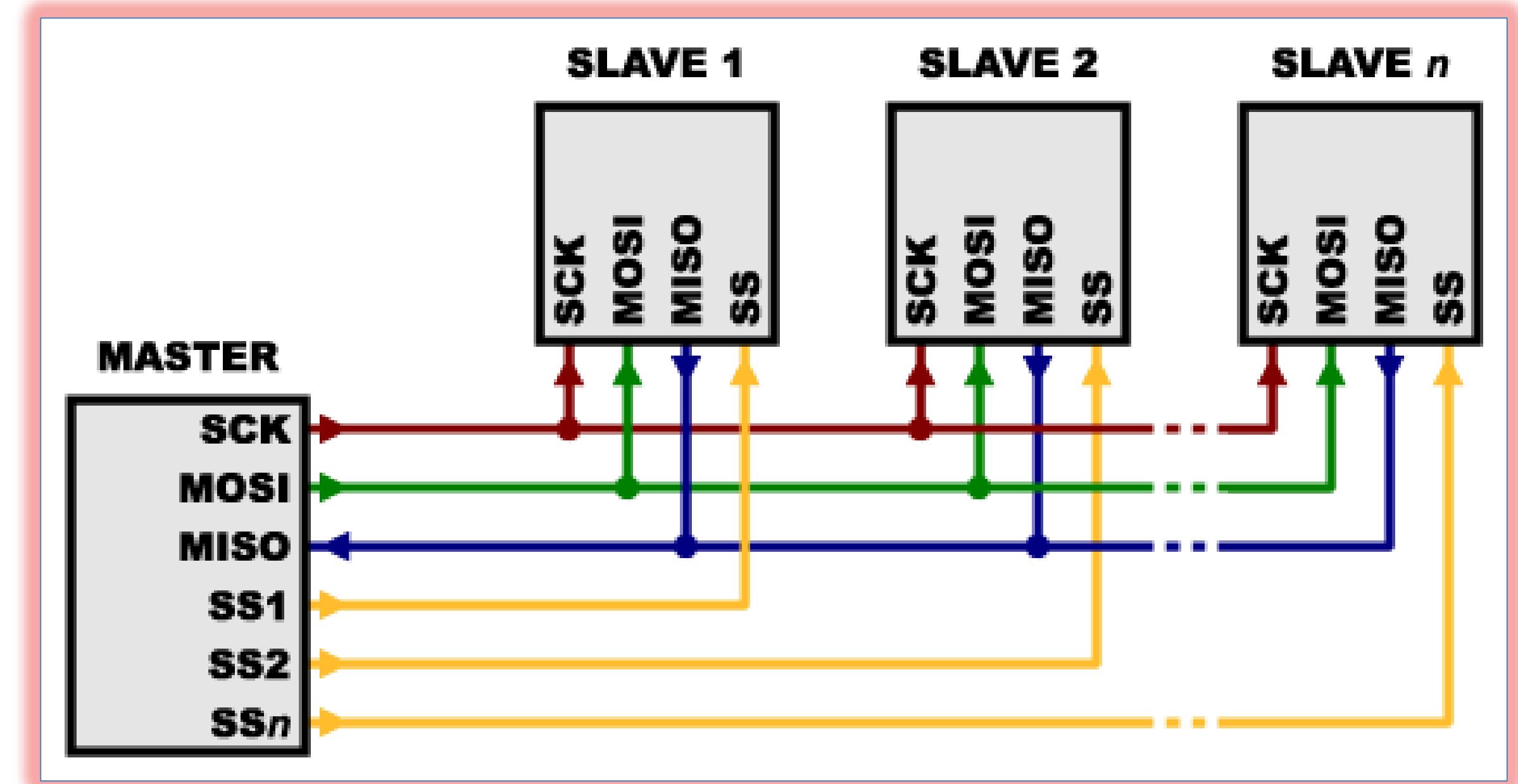
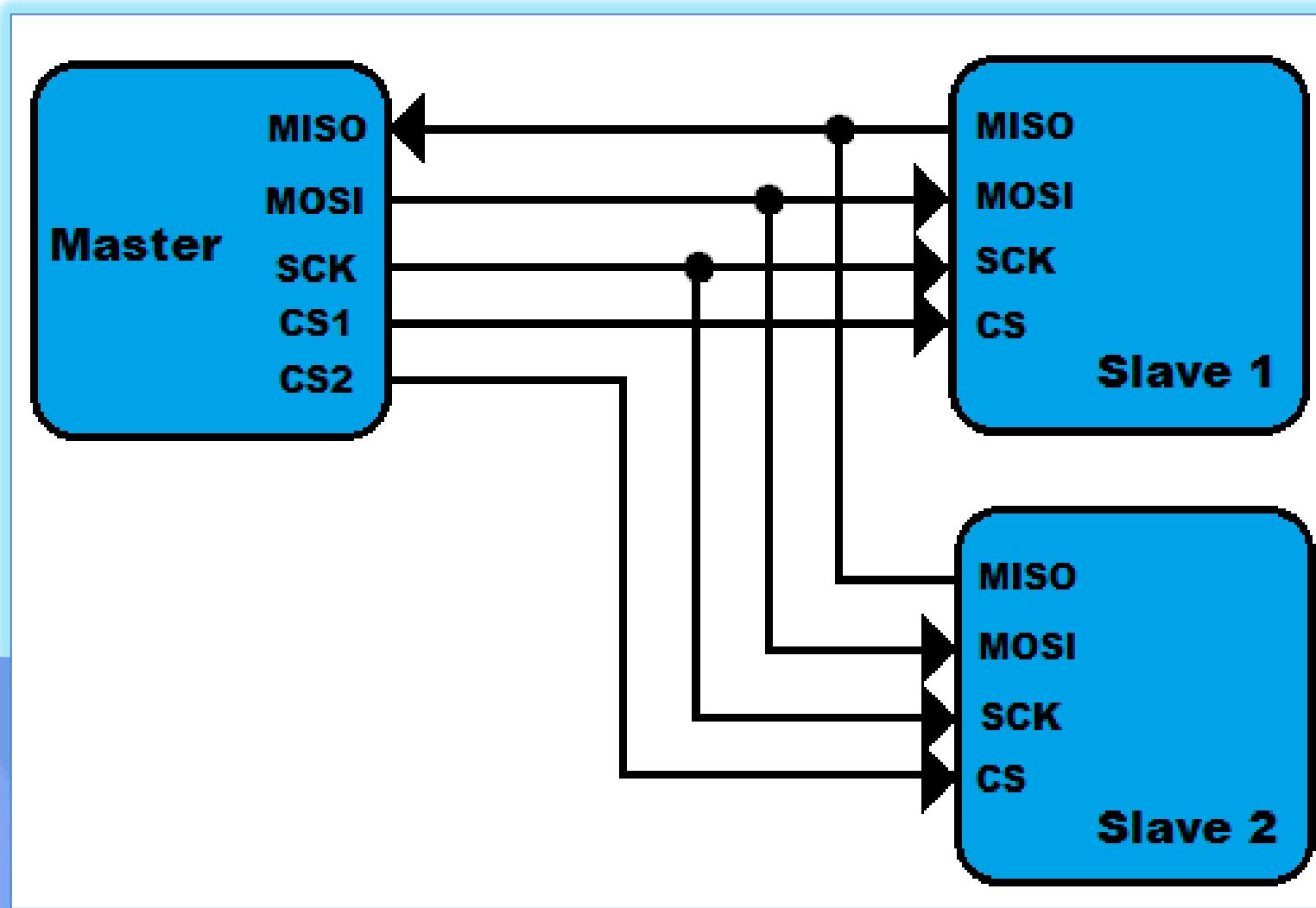
Se o **periférico** precisar enviar uma resposta ao **controlador**, este continuará a gerar o número pré-determinado de ciclos de clock, e o **periférico** enviará os dados por meio de uma terceira linha de dados chamada **MISO** (Peripheral Out / Controller In ou **Master In / Slave Out**).

A linha **SS** (**Seleção**) é normalmente mantida em estado alto, o que desconecta o dispositivo periférico do barramento SPI.



Comunicação serial SPI

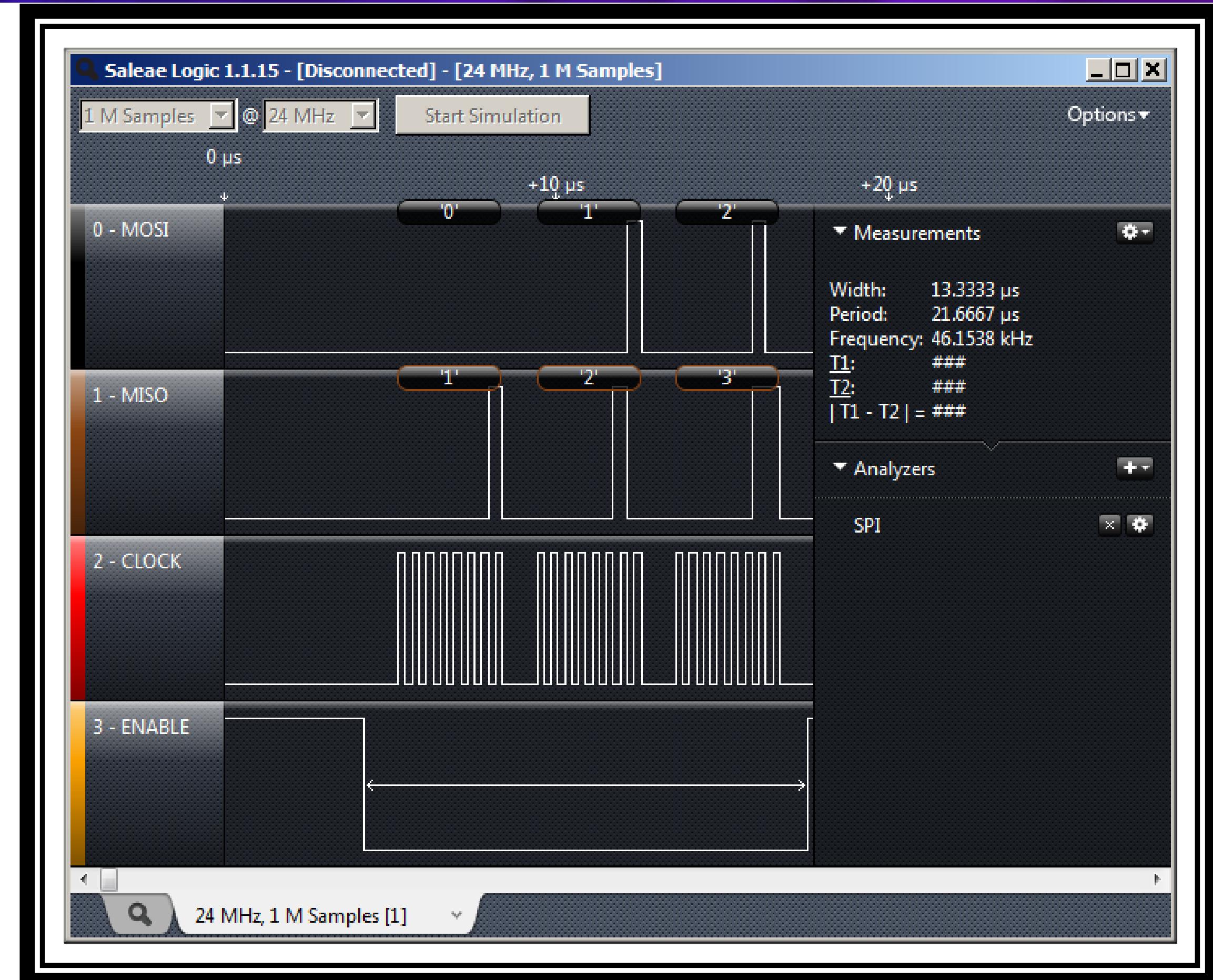
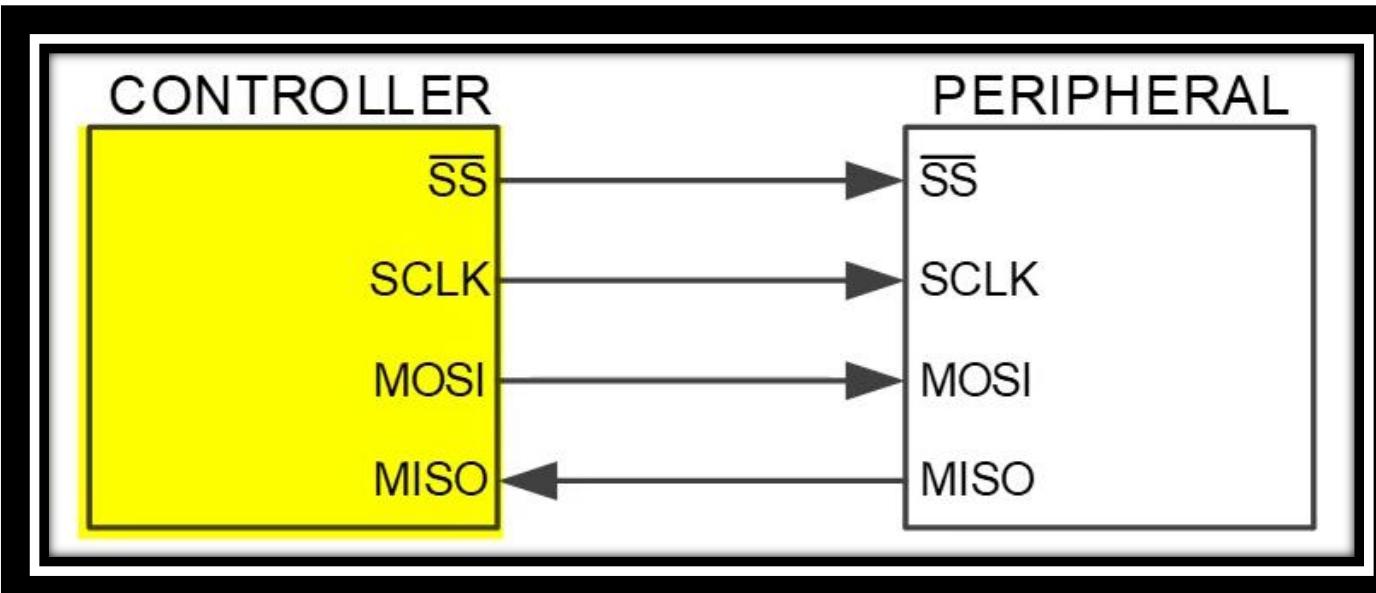
Representação
esquemática de
comunicação SPI com 2 ou 3
dispositivos periféricos.



Comunicação serial SPI

Linhas de comunicação RP2

- **SCK** (Serial Clock) = **SCK**
- **MOSI** (Master Out, Slave In) = **TX**
- **MISO** (Master In, Slave Out) = **RX**
- **SS** (Slave Select): **CSN**



Comunicação serial I²C

I²C (Inter-Integrated Circuit)

O I²C é um **protocolo serial síncrono** usado para a troca de dados entre MCUs e periféricos, tais como: sensores e displays.

Criado pela Philips Semiconductors em 1982. O protocolo suporta **múltiplos dispositivos** alvo em um barramento de comunicação e também pode suportar **múltiplos controladores** que enviam e recebem comandos e dados. A comunicação é **enviada em pacotes** de bytes com um **endereço exclusivo** para cada dispositivo alvo.

Características em destaque do I²C :

Barramento de dois fios:

- **SDA** (Serial Data Line): linha de dados bidirecional.
- **SCL** (Serial Clock Line): linha de clock gerada pelo dispositivo mestre.

Topologia mestre-escravo:

- O dispositivo mestre controla o clock e inicia as comunicações.
- Escravos respondem a comandos do mestre.

Endereçamento:

- Cada dispositivo escravo tem um endereço único de 7 bits.

Velocidade:

Standard Mode: até 100 kbps.

Fast Mode Plus: até 1 Mbps.

Ultra-Fast Mode: até 5 Mbps.

Fast Mode: até 400 kbps.

High-Speed Mode: até 3.4 Mbps.

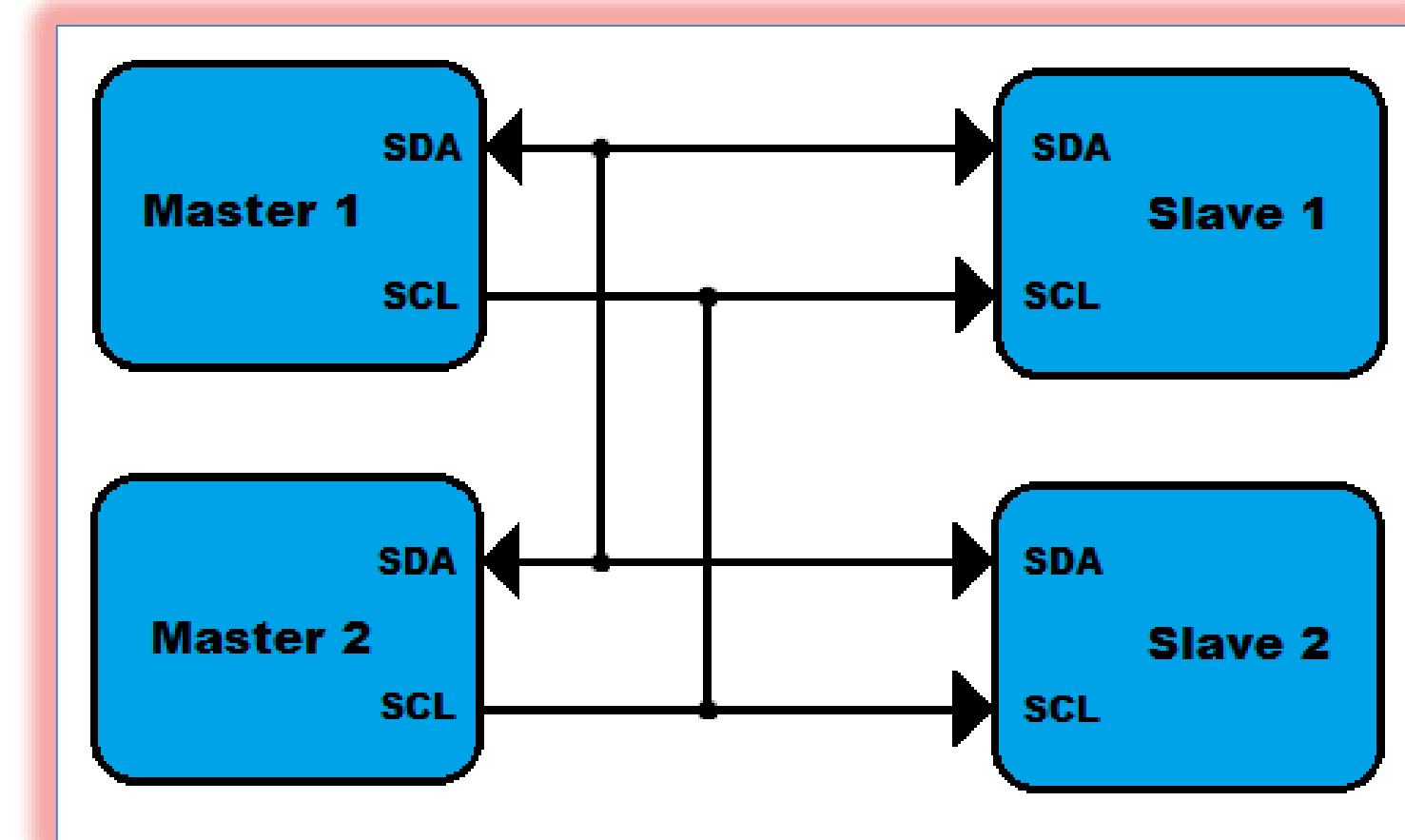
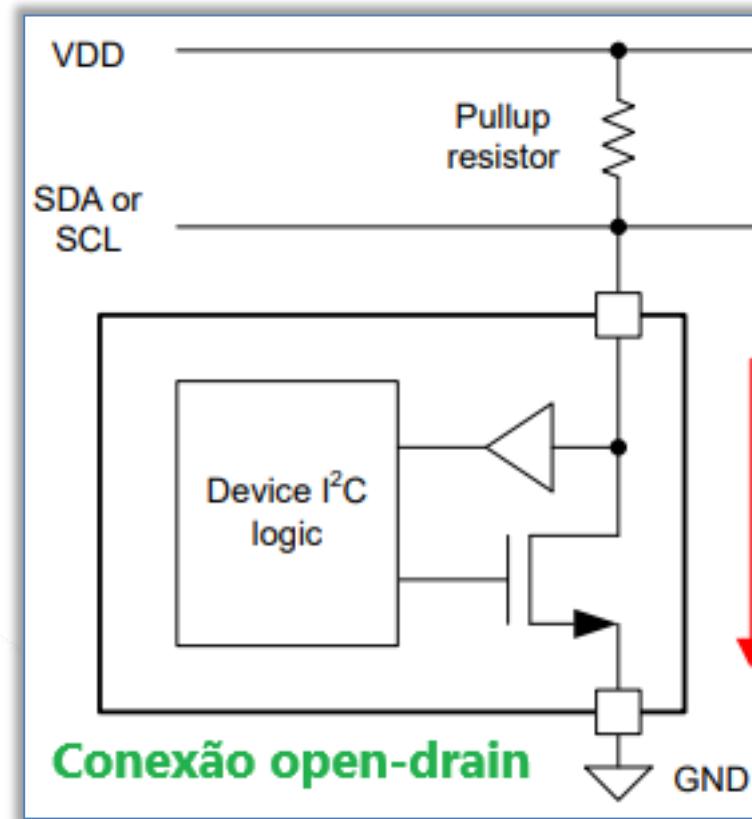
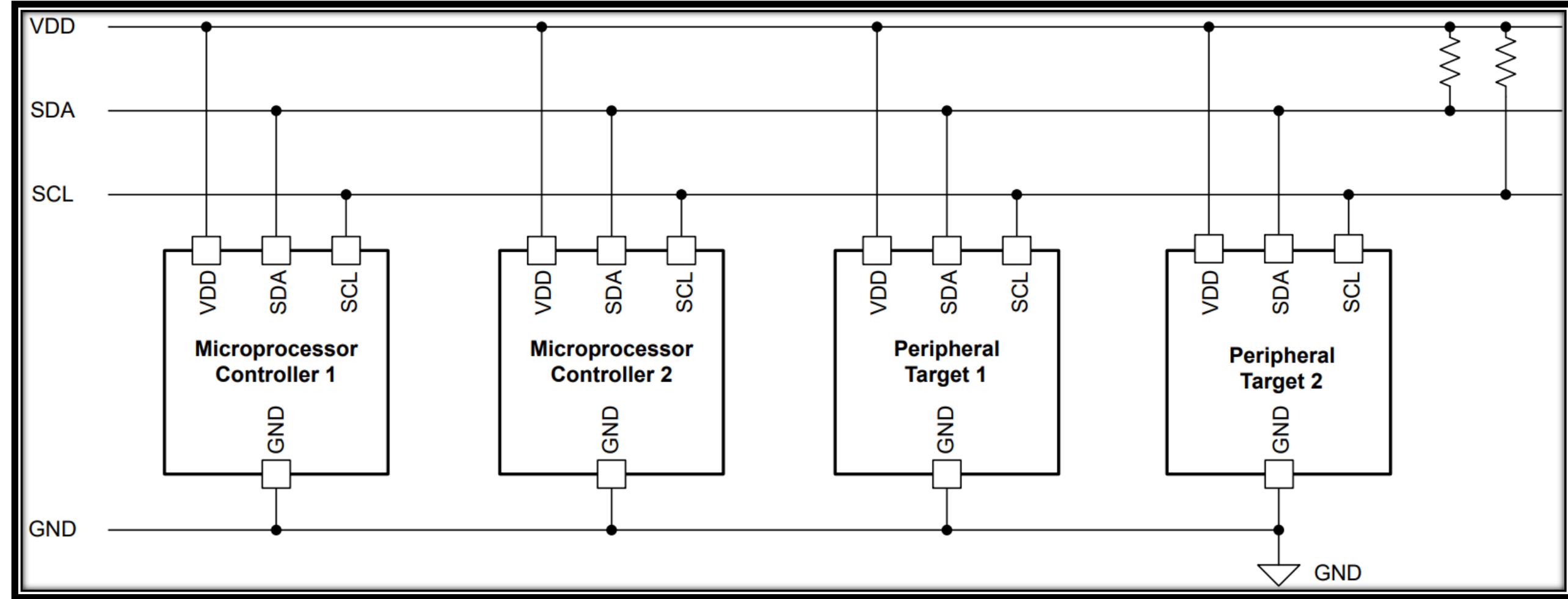
Pull-up resistors:

As linhas SDA e SCL são conectadas a resistores pull-up para manter o estado alto quando não estão em uso.

Comunicação serial I²C

Linhas de comunicação RP2

- **SDA** (Serial Data)
- **SCL** (Serial Clock)



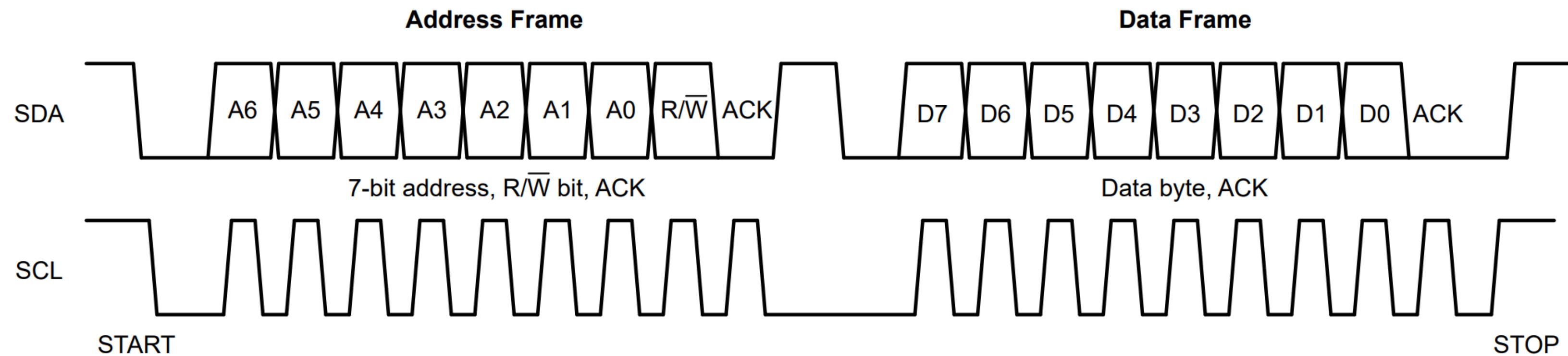
Comunicação serial I²C

Quadro de endereços e Dados do protocolo I²C

As mensagens são divididas em dois tipos de quadro (frame):

- Um **quadro de endereço**, onde o mestre indica o escravo para o qual a mensagem está sendo enviada. (7 bits o que resulta em no máximo **127 dispositivos** endereçáveis).
- Um ou mais **quadros de dados**, que são mensagens de dados de 8 bits passadas de mestre para escravo ou vice-versa .

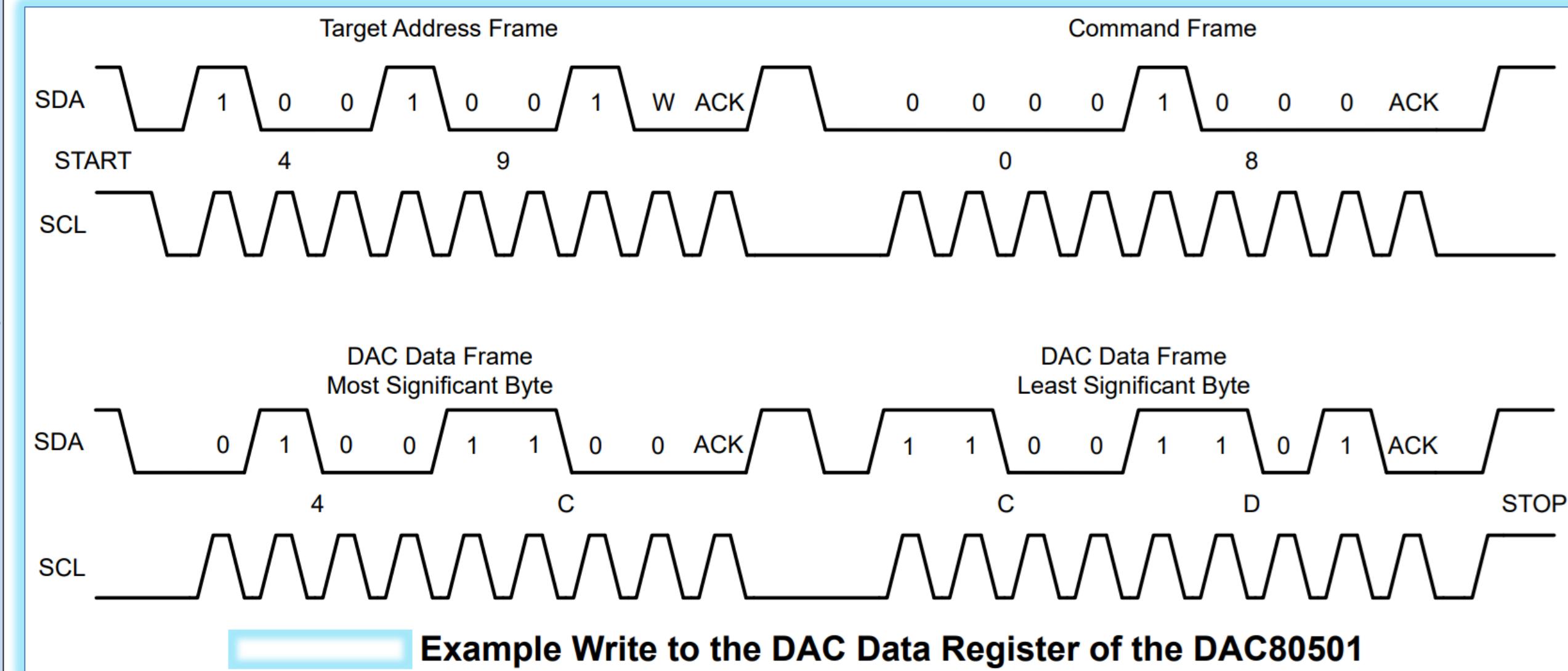
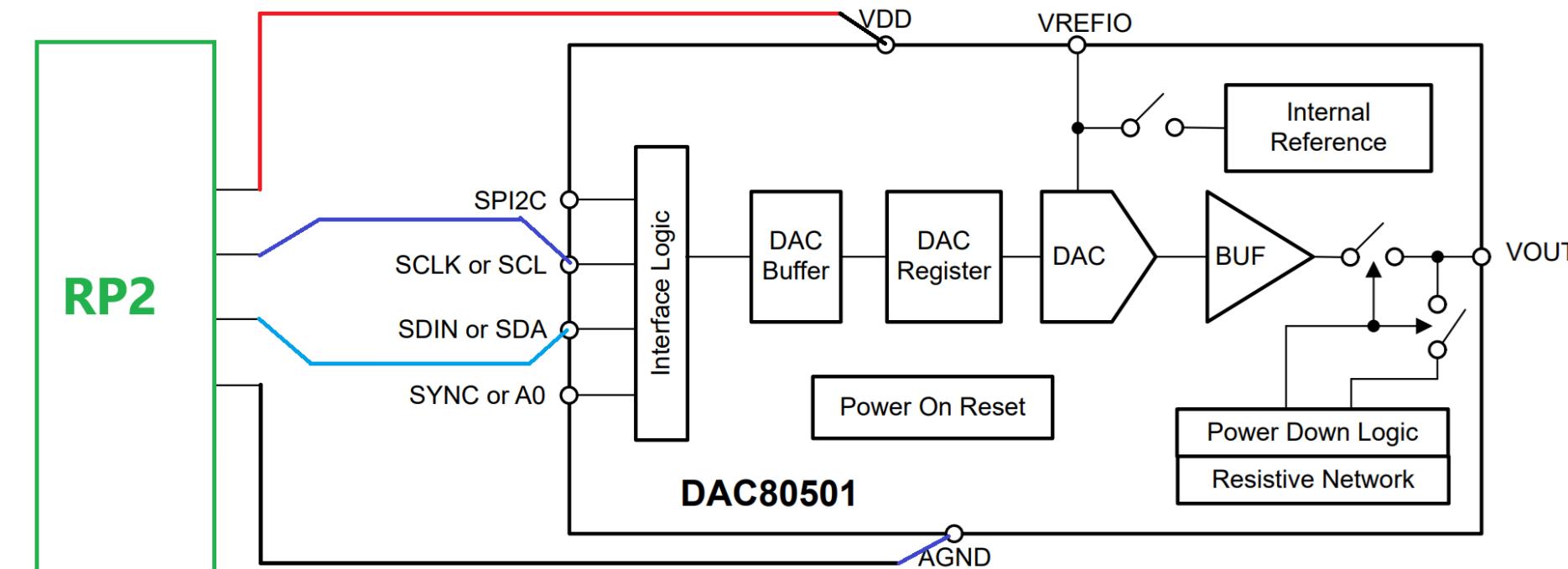
Os dados são colocados na linha SDA depois que o SCL fica baixo e são amostrados depois que a linha SCL fica alta. O tempo entre a transição do clock e a leitura/gravação dos dados é definido pelos dispositivos no barramento e varia de chip para chip.



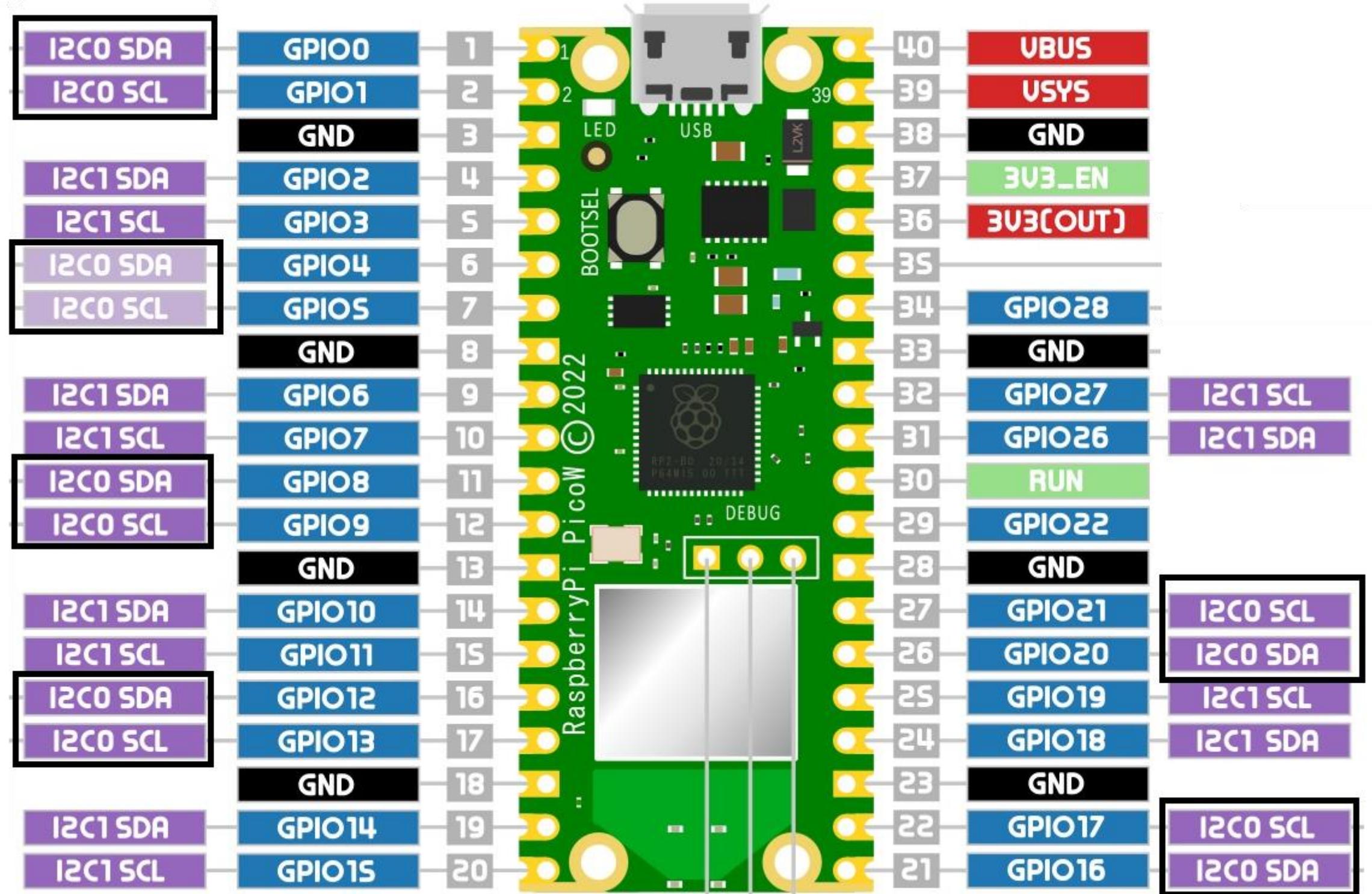
Comunicação serial I²C

Funcionamento Básico:

- Início (Start Condition):** O mestre envia um pulso de início puxando SDA para baixo enquanto SCL está alto.
- Endereçamento:** O mestre envia o endereço do escravo desejado, seguido de um bit indicando se a operação será de leitura ou escrita.
- Transferência de Dados:**
Escrita: O mestre envia dados ao escravo.
Leitura: O escravo envia dados ao mestre.
- Confirmação (ACK/NACK):** Após cada byte transferido, o receptor (mestre ou escravo) envia um sinal de confirmação (ACK) ou negação (NACK).
- Término (Stop Condition):** O mestre sinaliza o fim da comunicação liberando SDA enquanto SCL está alto.



Comunicação serial I²C no RP2040



Linhas de I2C no RP2

- **SDA** (Serial Data Line)
- **SCL** (Serial Clock Line)

Display I²C SSD1306

Características:

Consumo de energia ultra baixo: tela acesa 0.08W

O brilho e o contraste altos são ajustáveis

Com controlador incorporado.

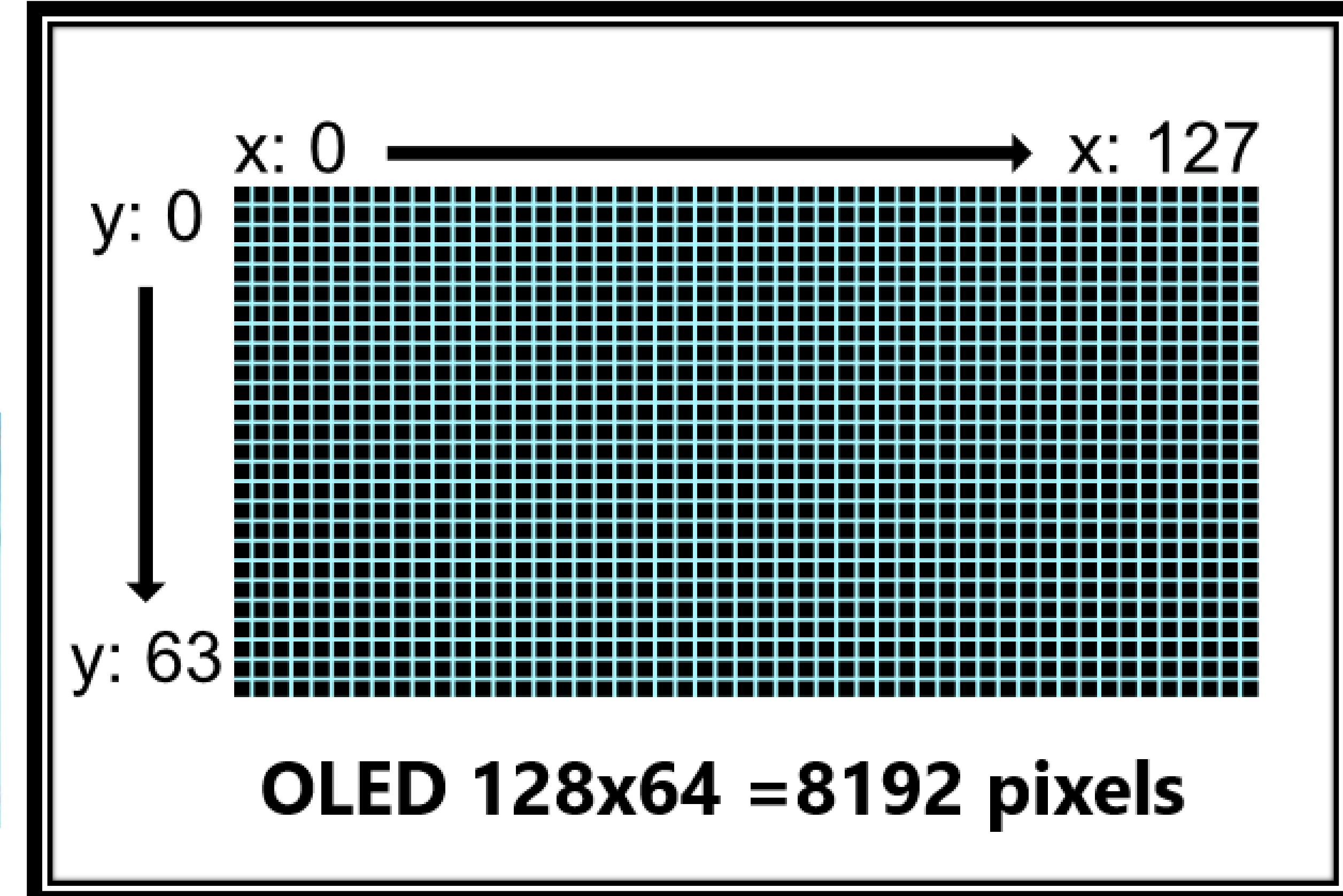
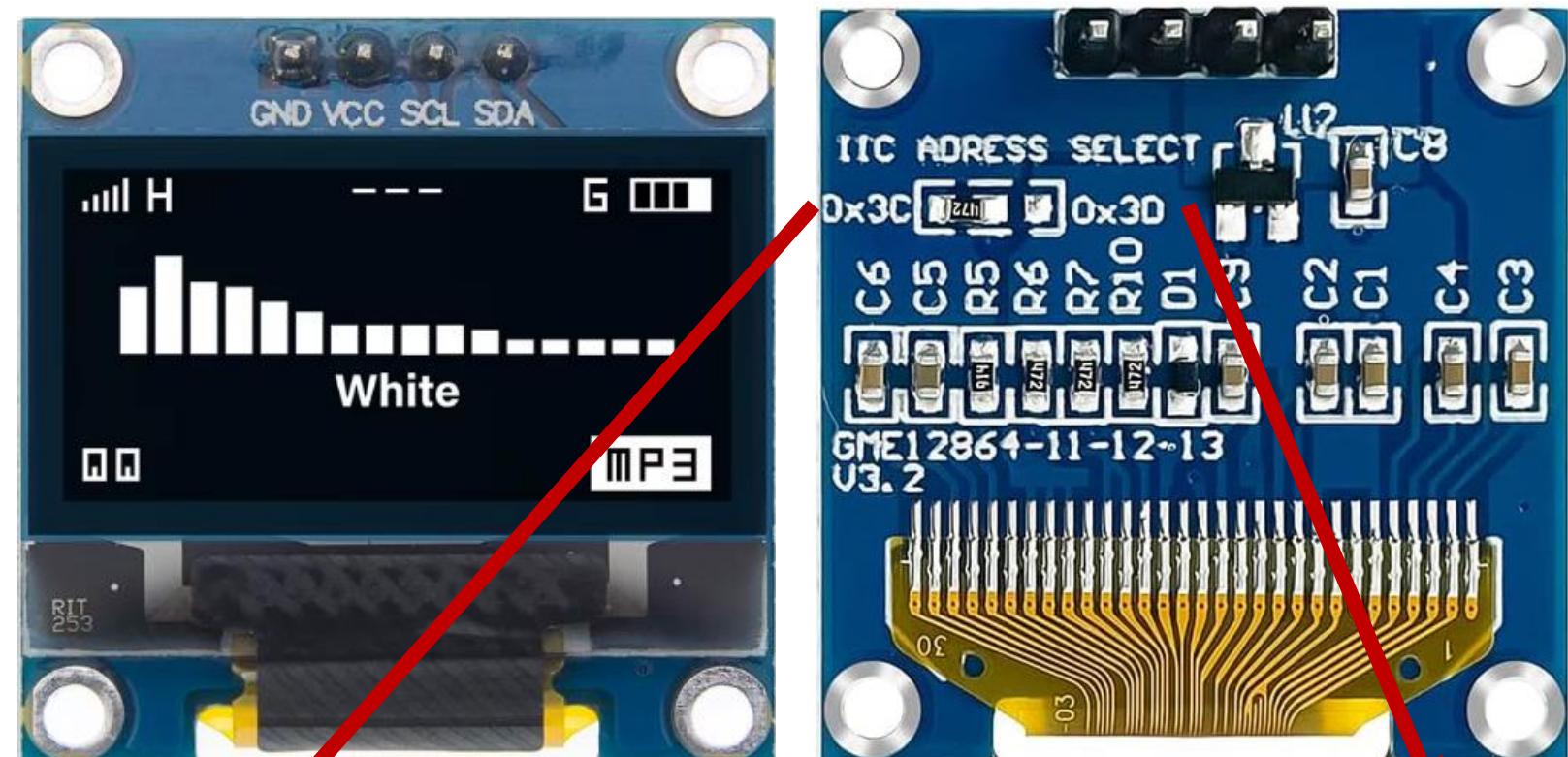
Tipo de interface I²C

Definição do Pin: **GND, VCC ,SCL, SDA**

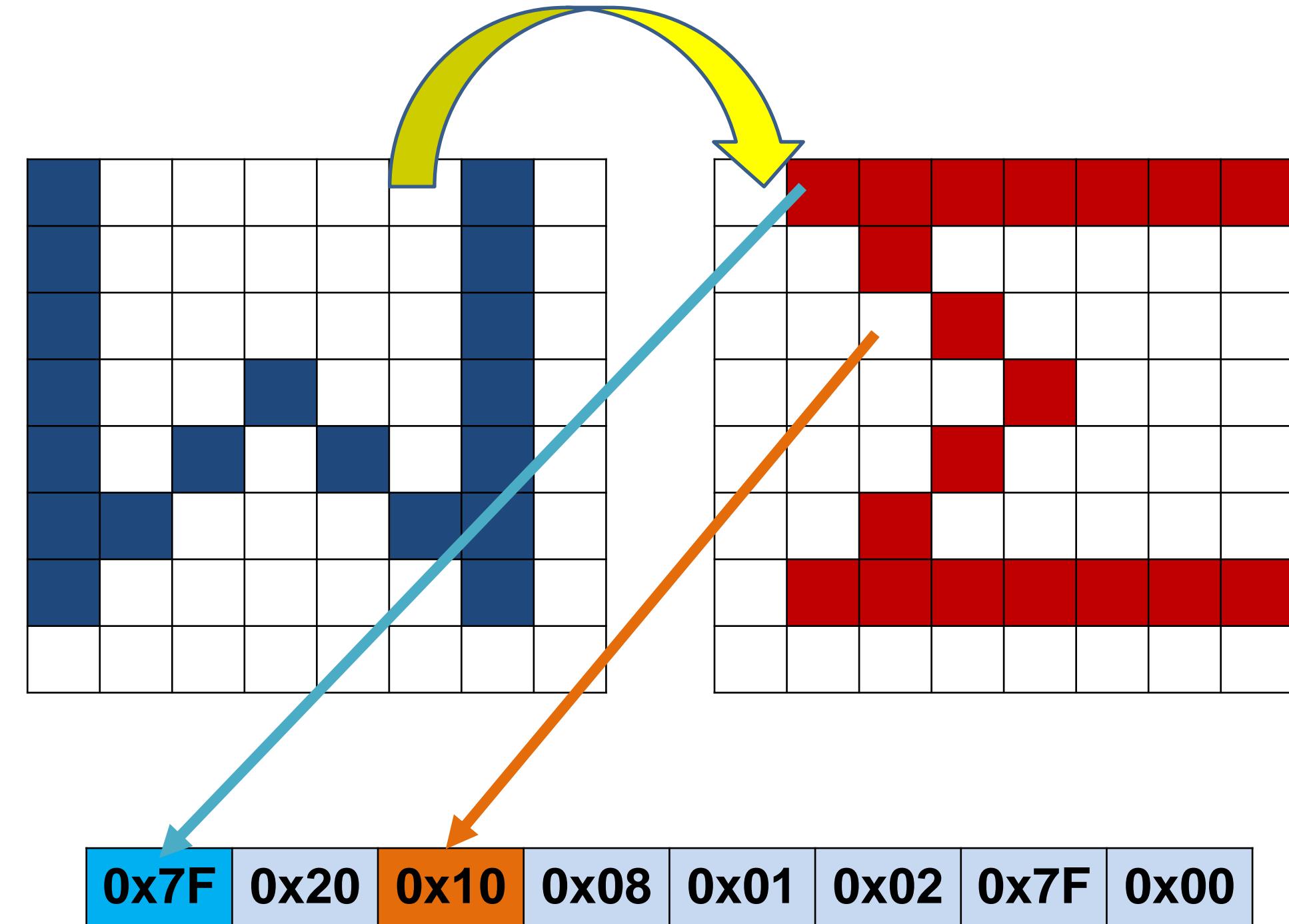
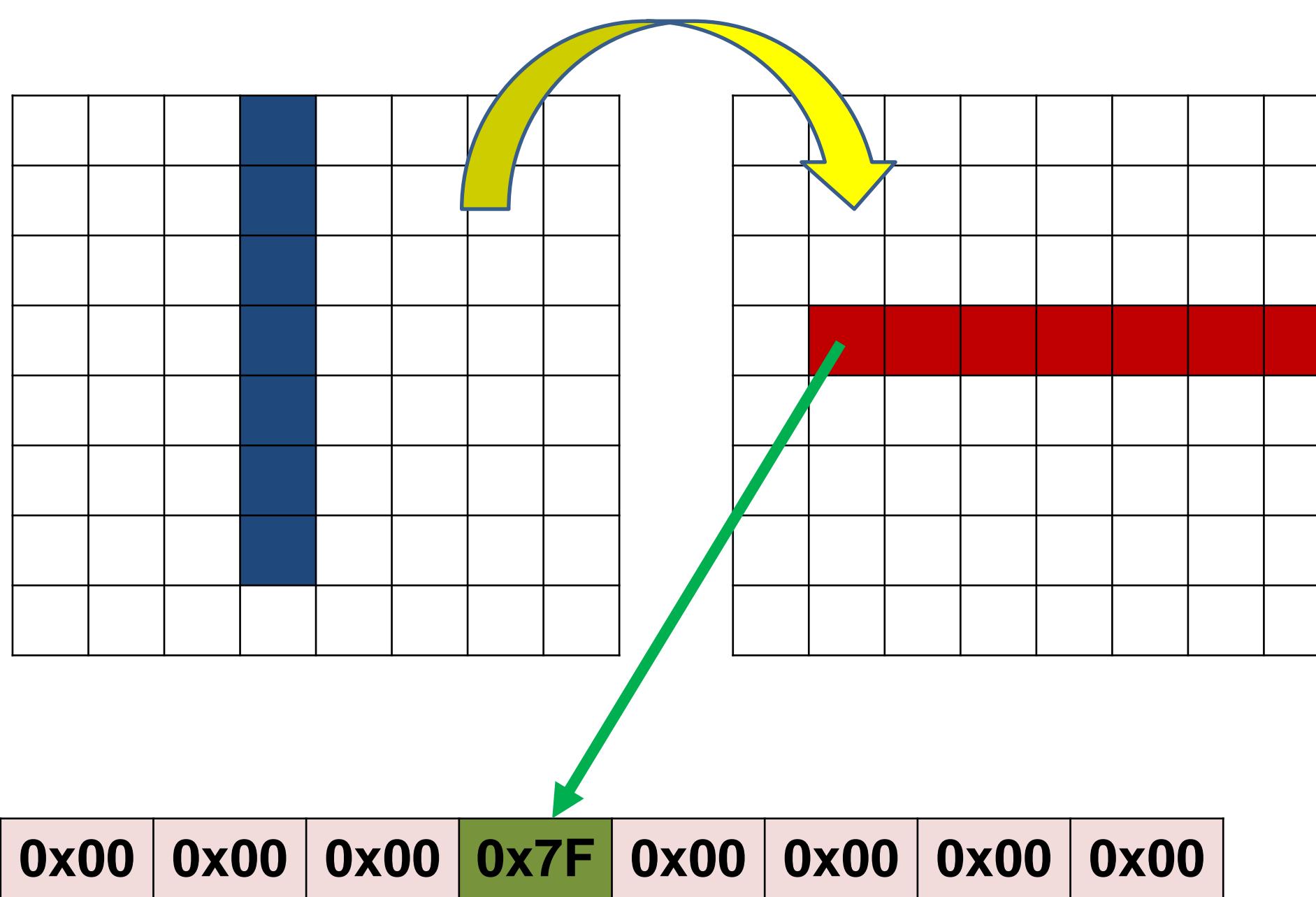
Endereço: **0x3C** ou **0x3D**

Tensão: 3V ~ 5V DC

Temperatura de operação: -30 ° ~ 70 °

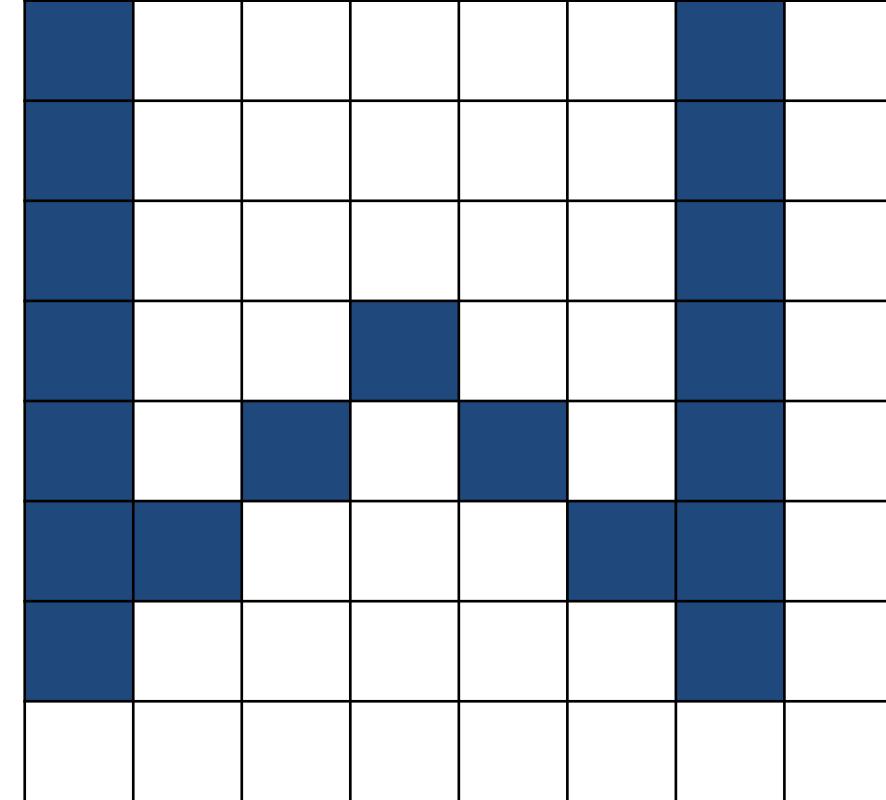


Desenho de caracteres 8x8 no display

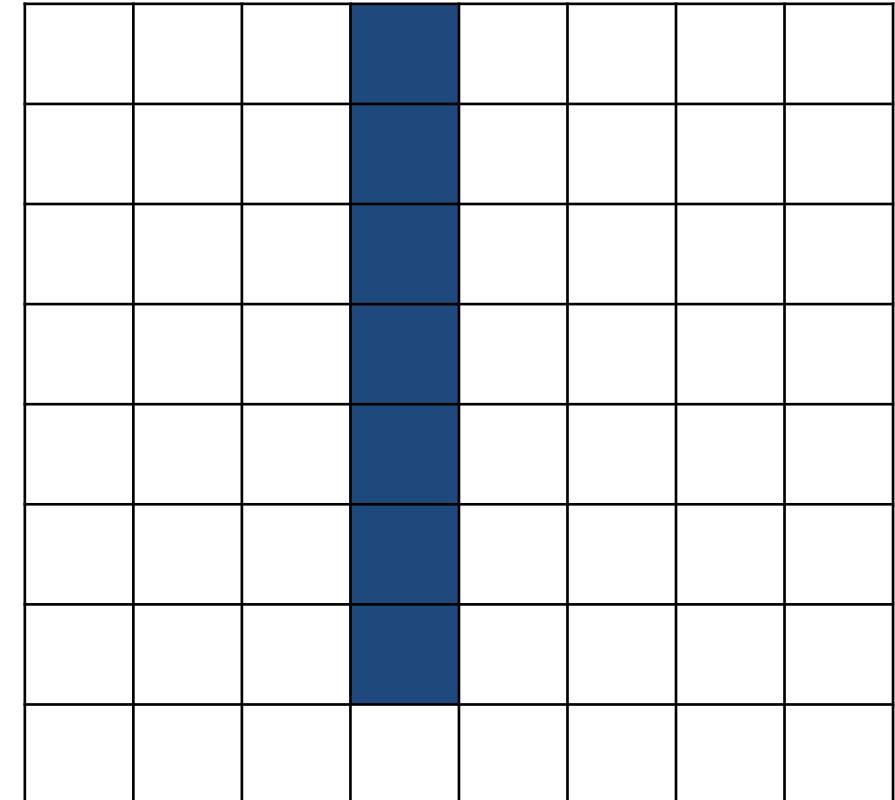


Desenho de caracteres 8x8 no display

Posição (7,3)

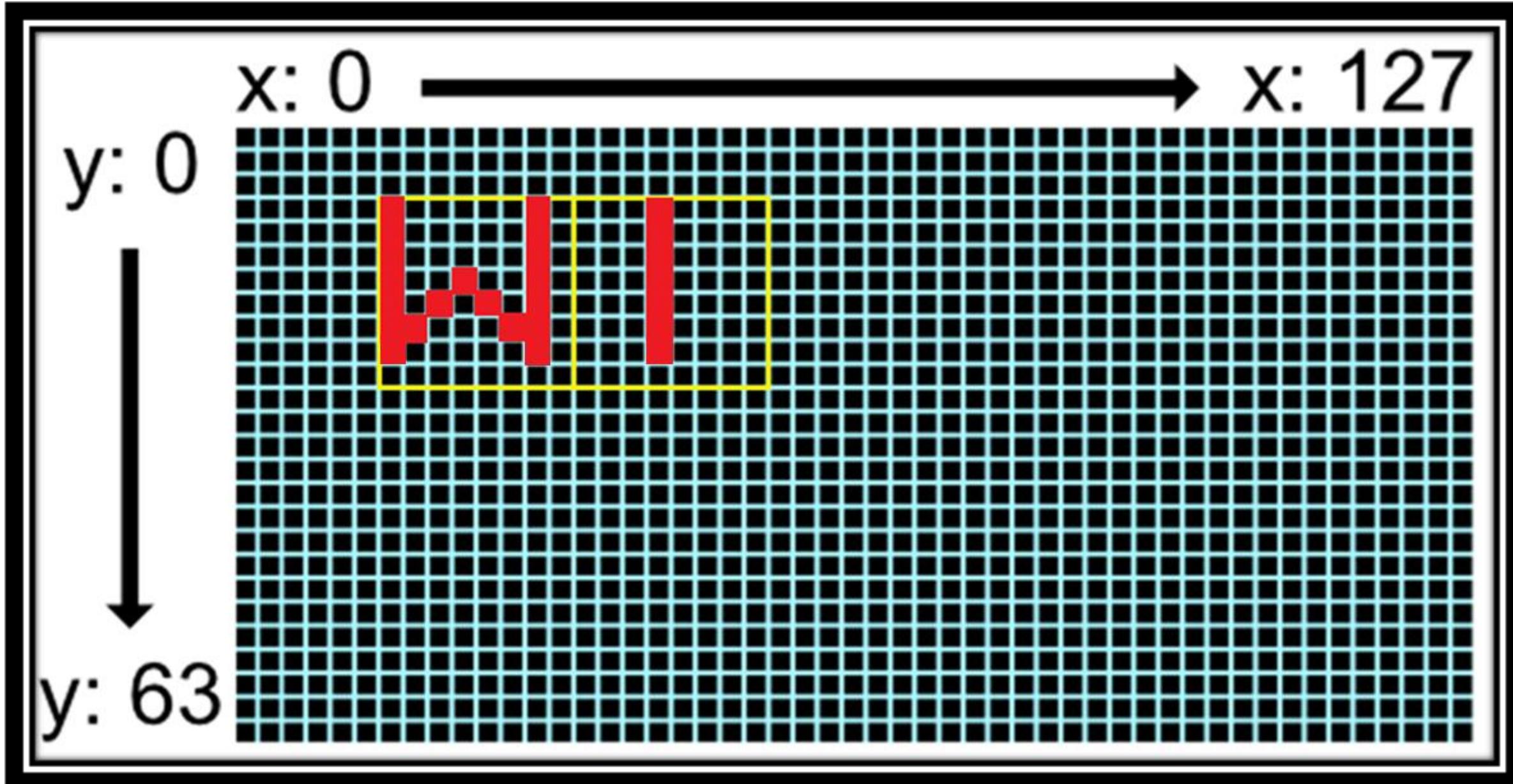


Posição (15,3)



0x7F 0x20 0x10 0x08 0x01 0x02 0x7F 0x00

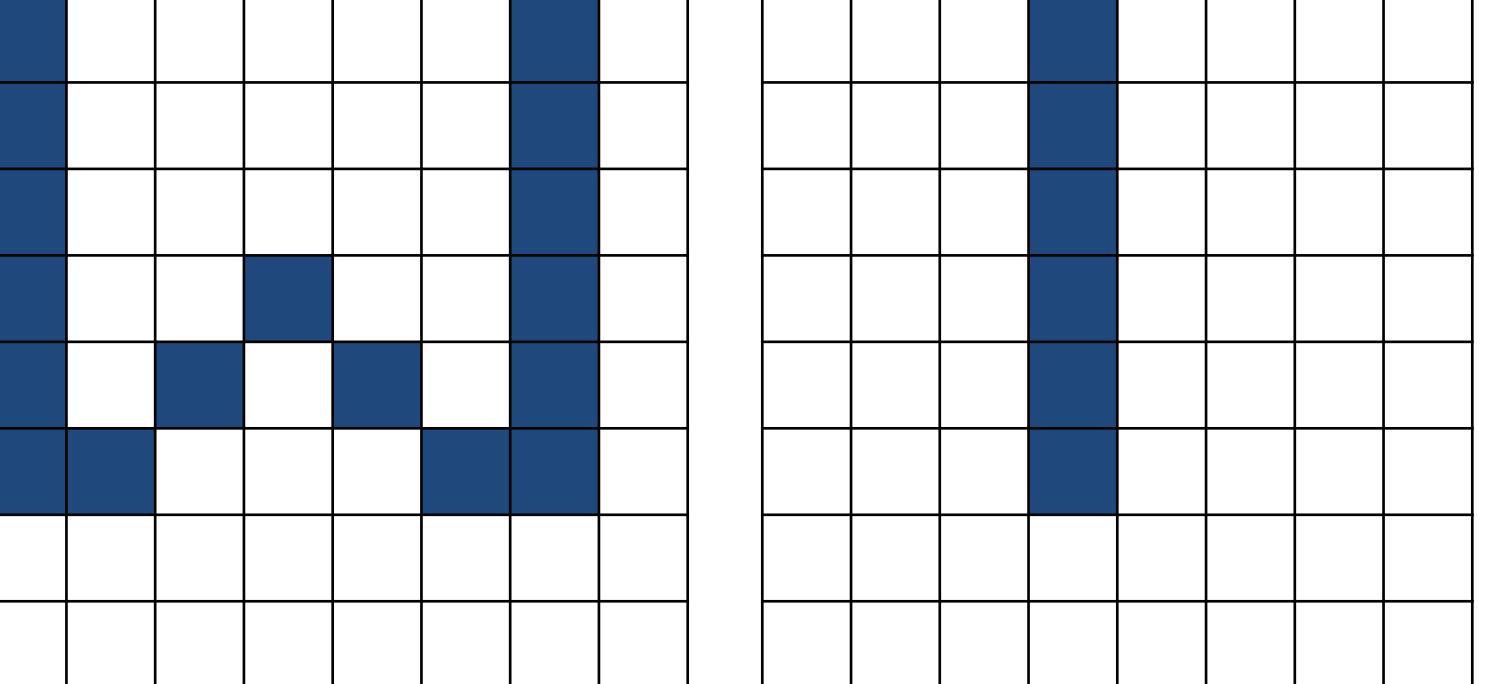
0x00 0x00 0x00 0x7F 0x00 0x00 0x00 0x00



Quantos bytes devemos enviar para o preenchimento de toda a tela do display?
Lembre-se 128 x 64 (pixels)

Exercício: Display I²C SSD1306

```
static uint8_t font[] = {  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Nothing  
    0x3e, 0x41, 0x41, 0x49, 0x41, 0x41, 0x3e, 0x00, //0  
    0x00, 0x00, 0x42, 0x7f, 0x40, 0x00, 0x00, 0x00, //1  
    0x30, 0x49, 0x49, 0x49, 0x49, 0x46, 0x00, 0x00, //2  
    0x49, 0x49, 0x49, 0x49, 0x49, 0x36, 0x00, 0x00, //3  
    0x3f, 0x20, 0x20, 0x78, 0x20, 0x20, 0x00, 0x00, //4  
    ...  
    0x7f, 0x41, 0x41, 0x41, 0x51, 0x51, 0x73, 0x00, //G  
    0x7f, 0x08, 0x08, 0x08, 0x08, 0x08, 0x7f, 0x00, //H  
    0x00, 0x00, 0x00, 0x7f, 0x00, 0x00, 0x00, 0x00, //I  
    0x21, 0x41, 0x41, 0x3f, 0x01, 0x01, 0x01, 0x00, //J  
    0x00, 0x7f, 0x08, 0x08, 0x14, 0x22, 0x41, 0x00, //K  
    ...  
    0x3f, 0x40, 0x40, 0x40, 0x40, 0x40, 0x3f, 0x00, //U  
    0x0f, 0x10, 0x20, 0x40, 0x20, 0x10, 0x0f, 0x00, //V  
    0x7f, 0x20, 0x10, 0x08, 0x10, 0x20, 0x7f, 0x00, //W  
    0x00, 0x41, 0x22, 0x14, 0x14, 0x22, 0x41, 0x00, //X  
    0x01, 0x02, 0x04, 0x78, 0x04, 0x02, 0x01, 0x00, //Y  
    0x41, 0x61, 0x59, 0x45, 0x43, 0x41, 0x00, 0x00, //Z  
};
```



0x00 0x00 0x00 0x7F 0x00 0x00 0x00 0x00

0x7F 0x20 0x10 0x08 0x01 0x02 0x7F 0x00

Exercício: Display I²C SSD1306

```
#include <stdlib.h>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include "inc/ssd1306.h"
#include "inc/font.h"
#define I2C_PORT i2c1
#define I2C_SDA 14
#define I2C_SCL 15

int main()
{
    // I2C Initialisation. Using it at 400Khz.
    i2c_init(I2C_PORT, 400 * 1000);
    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
    gpio_pull_up(I2C_SDA);
    gpio_pull_up(I2C_SCL);
    ssd1306_t ssd;
    ssd1306_init(&ssd, WIDTH, HEIGHT, false, 0x3C, I2C_PORT);
    ssd1306_config(&ssd);
    ssd1306_send_data(&ssd);
```

Pasta:
I2C_DisplayEmC

```
// Limpa o display
ssd1306_fill(&ssd, false);
ssd1306_send_data(&ssd);

bool cor = true;
while (true)
{
    cor = !cor;

    // Atualiza o conteúdo do display com animações
    ssd1306_fill(&ssd, !cor);
    ssd1306_rect(&ssd, 3, 3, 122, 58, cor, !cor);
    ssd1306_draw_string(&ssd, "CEPEDI TIC37", 8, 10);
    ssd1306_draw_string(&ssd, "EMBARCATECH", 20, 30);
    ssd1306_draw_string(&ssd, "PROF WILTON", 15, 48);
    ssd1306_send_data(&ssd);

    sleep_ms(1000);
}
```

Comunicação serial comparação

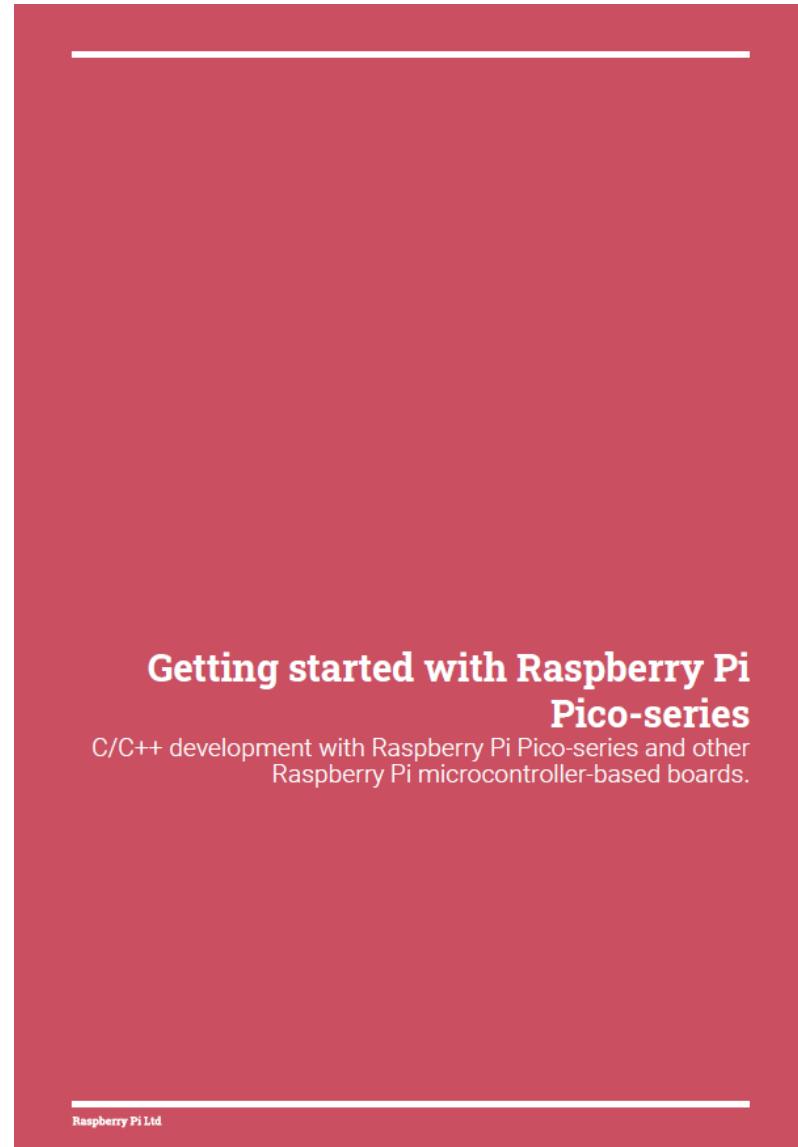
Interface	UART	I2C	SPI
Complexidade	Simples	Fácil de comandar vários dispositivos	Complexidade aumenta com o número de dispositivos.
Velocidade	Slowest	Mais rápida que UART	Mais rápida das três
Bits por segundo (bps)	Até 115.2k	Até 400k	Até 10M
Número de dispositivos	Up to 2 devices	Up to 127	Alguns
Número de fios	3	3	4+1 por dispositivo
Nº de Masters e Slaves	Um para um	Múltiplos Masters e Slaves	1 Master, múltiplos Slaves

Introdução às Interfaces de Comunicação Serial com RP2040

Finalizando

Introdução às Interfaces de Comunicação Serial com RP2040

Referências



<https://github.com/BitDogLab/BitDogLab/tree/main/doc>

<https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html>

Introdução às Interfaces de Comunicação Serial com RP2040

Pontos Principais

- Conversões de números
 - » Binário, Hexadecimal e Decimal.
 - » Tabela ASCII
- Comunicação UART, SPI e I2C no RP2

Obrigado!

Executores:



Coordenação:



Iniciativa:

