



Universidade Estadual de Campinas  
Faculdade de Engenharia Elétrica  
e de Computação



---

# **EA871 - Laboratório de Programação Básica de Sistemas Digitais**

**Roteiro 11 - Interface Serial Assíncrona UART**

---

Autores: Fernando Teodoro de Cillo e Rafael Silva Cirino

RA: 197029    223730

Campinas  
Junho de 2022

# Introdução:

O intuito deste experimento é compreender o funcionamento da interface serial assíncrona UART e da porta serial COM. Outros tópicos abordados são o buffer circular e manipulação de strings em C.

# Experimento:

## 1

**1.a** Configurando os parâmetros de comunicação serial do terminal e do canal 2 do analisador lógico para *baud rate* 4800, 1 *stop bit*, sem bit de paridade e executando o projeto `rot11.aula` obtemos o print da figura 1. Podemos ver que após o start bit temos os bits de dados, começando pelo LSB, e cada valor hexadecimal corresponde a um caractere pelo padrão ASCII, até o *stop bit*.

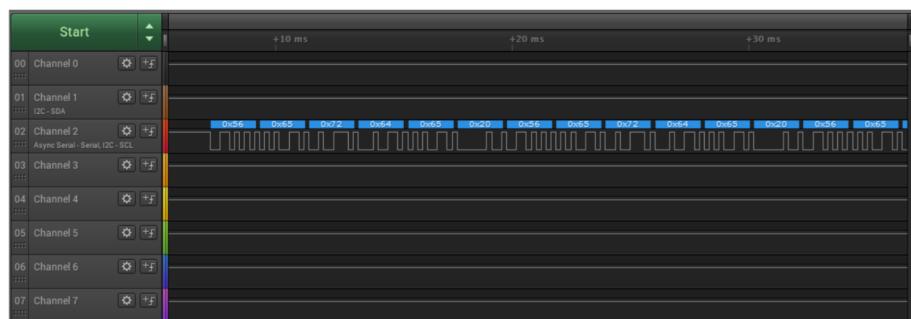


Figura 1: Sinais capturados pelo analisador lógico

**1.b** A largura de pulso de um *bit* está coerente com o *baud rate* configurado para o módulo UART2/UART0, como mostra a figura 2. Vemos que a largura de 1 bit condiz com os 0.2ms esperados.

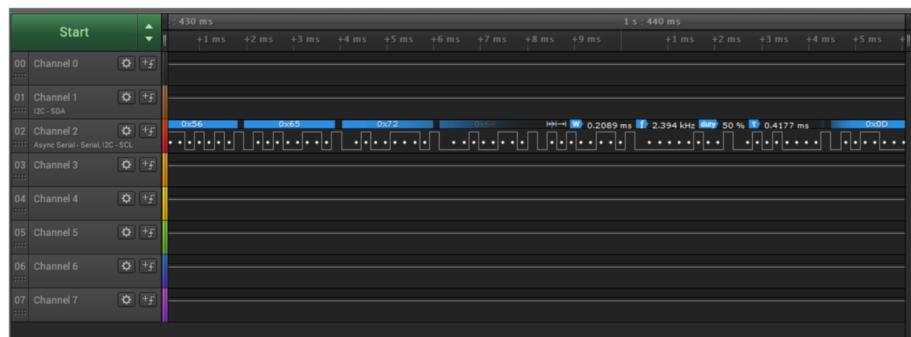


Figura 2: períodos

**1.c** Alterando os parâmetros de comunicação serial obtemos a figura 3, cuja forma de onda condiz com o esperado, pois vemos que agora há dois *stop bits*.

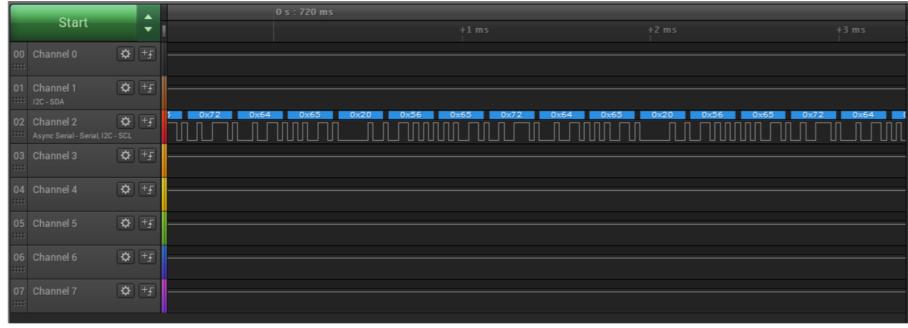


Figura 3: *baud rate* 38400, 2 *stop bits* e sem *bit* de paridade

**1.d** Inserindo a função `UART0_ativaInterruptTxTerminal` não esperamos que o analisador lógico mostre alterações, porque não há escrita no barramento pelo analisador, somente leitura.

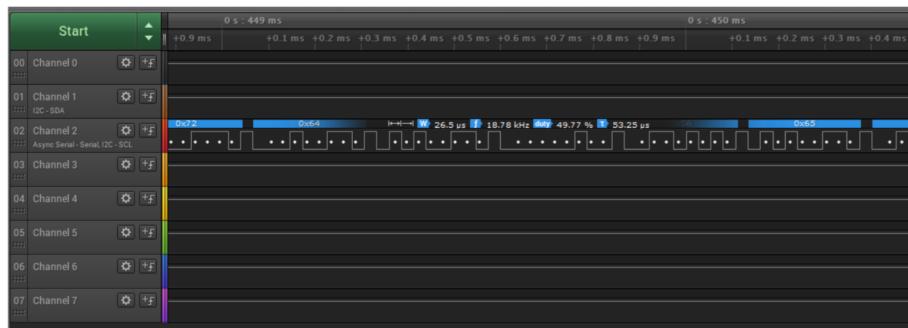


Figura 4: q1d

## 7

A figura 5 mostra a aba *Variables* para diversas operações lógico-aritméticas com *floats* e com inteiros. Os resultados das operações de soma e divisão com *floats* são mostradas em *fvalor* e com inteiros em *ivalor*. Vemos que todas as operações estão corretas, porque retornam os valores que se espera.

| Name     | Value      | Location   |
|----------|------------|------------|
| > op     | 0x4761032d | 0x20002f4  |
| string   | 0x20002fe8 | 0x20002fe8 |
| ↳ [0]    | '4'        | 0x20002fe8 |
| ↳ [1]    | ' '        | 0x20002fe9 |
| ↳ [2]    | '.'        | 0x20002fea |
| ↳ [3]    | ' '        | 0x20002feb |
| ↳ [4]    | '5'        | 0x20002fec |
| ↳ [5]    | ' '        | 0x20002fed |
| ↳ fvalor | 9.0        | 0x20002fe4 |
| ↳ ivalor | 0          | 0x20002fe0 |
| ↳ check  | 1          | 0x20002ff0 |

(a) Soma:  $4 + 5 = 9$

| Name     | Value      | Location   |
|----------|------------|------------|
| > op     | 0x4761032d | 0x20002f4  |
| string   | 0x20002fe8 | 0x20002fe8 |
| ↳ [0]    | '4'        | 0x20002fe8 |
| ↳ [1]    | ' '        | 0x20002fe9 |
| ↳ [2]    | '/'        | 0x20002fea |
| ↳ [3]    | ' '        | 0x20002feb |
| ↳ [4]    | '5'        | 0x20002fec |
| ↳ [5]    | ' '        | 0x20002fed |
| ↳ fvalor | 0.8        | 0x20002fe4 |
| ↳ ivalor | 4          | 0x20002fe0 |
| ↳ check  | 1          | 0x20002ff0 |

(b) Divisão:  $4 / 5 = 0.8$

| Name     | Value      | Location   |
|----------|------------|------------|
| > op     | 0x4761032d | 0x20002f4  |
| string   | 0x20002fe8 | 0x20002fe8 |
| ↳ [0]    | '4'        | 0x20002fe8 |
| ↳ [1]    | ' '        | 0x20002fe9 |
| ↳ [2]    | ' '        | 0x20002fea |
| ↳ [3]    | '.'        | 0x20002feb |
| ↳ [4]    | '5'        | 0x20002fec |
| ↳ [5]    | ' '        | 0x20002fed |
| ↳ fvalor | 0.8        | 0x20002fe4 |
| ↳ ivalor | 5          | 0x20002fe0 |
| ↳ check  | 2          | 0x20002ff0 |

(c) OR bit-a-bit:  $4(100) | 5(101) = 5(101)$

| Name     | Value      | Location   |
|----------|------------|------------|
| > op     | 0x4761032d | 0x20002f4  |
| string   | 0x20002fe8 | 0x20002fe8 |
| ↳ [0]    | '4'        | 0x20002fe8 |
| ↳ [1]    | ' '        | 0x20002fe9 |
| ↳ [2]    | '&'        | 0x20002fea |
| ↳ [3]    | '.'        | 0x20002feb |
| ↳ [4]    | '5'        | 0x20002fec |
| ↳ [5]    | ' '        | 0x20002fed |
| ↳ fvalor | 9.0        | 0x20002fe4 |
| ↳ ivalor | 4          | 0x20002fe0 |
| ↳ check  | 1          | 0x20002ff0 |

(d) AND bit-a-bit:  $4(100) \& 5(101) = 4(100)$

Figura 5: Operações

9

INICIO

MAIN:

Configuração UART0

Inicialização das variáveis

Configuração das mensagens que aparecem para o usuário

Inicialização da máquina de estados

LOOP:

Se houver mudança de estado:

atualiza o estado atual

switch case:

caso INICIO  
     altera o estado atual para MENSAGEM  
 caso MENSAGEM  
     exibe a frase ”Entre  $\langle op1 \rangle \langle op \rangle \langle op2 \rangle$  ( $\langle op \rangle : +, -, *, /, \%, \#, \&, |$ )\n \ r”  
     espera o usuário inserir a expressão  
 caso EXPRESSAO  
     espera  
 caso CALCULO  
     obtém os *tokens* dos operandos e do operador  
     realiza a operação correspondente  
     altera o estado atual para RESULTADO ou para ERRO  
 caso RESULTADO  
     exibe o resultado  
     altera o estado atual para INICIO  
 caso ERRO  
     exibe as mensagens de erro correspondentes ao erro  
 FIM

## 11

As figuras de 6 até 13 mostram o fluxo de execução do programa **calculadora**. Nota-se que as variáveis *estado\_atual* e *estado\_anterior* fazem o esperado para nossa máquina de estados segundo o pseudo-código. A figura 10 mostra o que ocorre quando há um erro, e na figura 11 é retomado o fluxo padrão (sem erro) da rotina.

| Name               | Value      | Location   |
|--------------------|------------|------------|
| > * op             | 0xffff054  | 0x20002fe4 |
| <> s_inicio        | 0.0        | 0x20002fd4 |
| <> s_erro          | 0          | 0x20002fd0 |
| <> s_error_op      | 0          | 0x20002fec |
| > s_inicio         | 0x20002f9c | 0x20002f9c |
| > s_erro           | 0x20002f60 | 0x20002f60 |
| > s_error_op       | 0x20002f20 | 0x20002f20 |
| <> estado_atual    | INICIO     | 0x20002fea |
| <> estado_anterior | RESULTADO  | 0x20002feb |

Figura 6: Estado atual: INICIO

| Variables          |            |            |
|--------------------|------------|------------|
| Name               | Value      | Location   |
| > * op             | 0xffff054  | 0x20002fe4 |
| <> s_inicio        | 0.0        | 0x20002fd4 |
| <> s_erro          | 0          | 0x20002fd0 |
| <> s_error_op      | 0          | 0x20002fec |
| > s_inicio         | 0x20002f9c | 0x20002f9c |
| > s_erro           | 0x20002f60 | 0x20002f60 |
| > s_error_op       | 0x20002f20 | 0x20002f20 |
| <> estado_atual    | MENSAGEM   | 0x20002fea |
| <> estado_anterior | INICIO     | 0x20002feb |

Figura 7: Estado atual: MENSAGEM

| Name            | Value     | Location   |
|-----------------|-----------|------------|
| op              | 0xffff054 | 0x20002fd4 |
| fvalor          | 0.0       | 0x20002fd0 |
| ivalor          | 0         | 0x20002fd0 |
| check           | 0         | 0x2000fec  |
| s_inicio        | 0x200029c | 0x200029c  |
| s_erro          | 0x2000260 | 0x2000260  |
| s_erro_op       | 0x20002f0 | 0x20002f0  |
| estado_atual    | EXPRESSAO | 0x2000fea  |
| estado_anterior | MENSAGEM  | 0x2000feb  |

Figura 8: Estado atual: EXPRESSAO

| Name             | Value      | Location   |
|------------------|------------|------------|
| fvalor           | 0.0        | 0x20002fd4 |
| ivalor           | 0          | 0x20002fd0 |
| check            | 0          | 0x2000fec  |
| s_inicio         | 0x200029c  | 0x200029c  |
| s_erro           | 0x2000260  | 0x2000260  |
| s_erro_op        | 0x20002f0  | 0x20002f0  |
| estado_atual     | CALCULO    | 0x2000fea  |
| estado_anterior  | EXPRESSAO  | 0x2000feb  |
| string_resultado | 0x1ffff020 | 0x1ffff020 |

Figura 9: Estado atual: CALCULO

| Name             | Value      | Location   |
|------------------|------------|------------|
| fvalor           | 0.0        | 0x20002fd4 |
| ivalor           | 0          | 0x20002fd0 |
| check            | -1         | 0x2000fec  |
| s_inicio         | 0x200029c  | 0x200029c  |
| s_erro           | 0x2000260  | 0x2000260  |
| s_erro_op        | 0x20002f0  | 0x20002f0  |
| estado_atual     | ERRO       | 0x2000fea  |
| estado_anterior  | CALCULO    | 0x2000feb  |
| string_resultado | 0x1ffff020 | 0x1ffff020 |

Figura 10: Estado atual: ERRO

| Name             | Value      | Location   |
|------------------|------------|------------|
| fvalor           | 0.0        | 0x20002fd4 |
| ivalor           | 0          | 0x20002fd0 |
| check            | -1         | 0x2000fec  |
| s_inicio         | 0x200029c  | 0x200029c  |
| s_erro           | 0x2000260  | 0x2000260  |
| s_erro_op        | 0x20002f0  | 0x20002f0  |
| estado_atual     | CALCULO    | 0x2000fea  |
| estado_anterior  | EXPRESSAO  | 0x2000feb  |
| string_resultado | 0x1ffff020 | 0x1ffff020 |

Figura 11: Estado atual: CALCULO

| Name             | Value      | Location   |
|------------------|------------|------------|
| fvalor           | 3.0        | 0x20002fd4 |
| ivalor           | 0          | 0x20002fd0 |
| check            | 1          | 0x2000fec  |
| s_inicio         | 0x200029c  | 0x200029c  |
| s_erro           | 0x2000260  | 0x2000260  |
| s_erro_op        | 0x20002f0  | 0x20002f0  |
| estado_atual     | RESULTADO  | 0x2000fea  |
| estado_anterior  | CALCULO    | 0x2000feb  |
| string_resultado | 0x1ffff020 | 0x1ffff020 |

Figura 12: Estado atual: RESULTADO

| Name             | Value      | Location   |
|------------------|------------|------------|
| fvalor           | 3.0        | 0x20002fd4 |
| ivalor           | 0          | 0x20002fd0 |
| check            | 1          | 0x2000fec  |
| s_inicio         | 0x200029c  | 0x200029c  |
| s_erro           | 0x2000260  | 0x2000260  |
| s_erro_op        | 0x20002f0  | 0x20002f0  |
| estado_atual     | INICIO     | 0x2000fea  |
| estado_anterior  | RESULTADO  | 0x2000feb  |
| string_resultado | 0x1ffff020 | 0x1ffff020 |

Figura 13: Estado atual: INICIO

## 13

A tabela 1 mostra o teste funcional de uma operação de soma. A mensagem mostrada é "Digite a operação", a operação digitada foi "2 + 2" e o resultado obtido é "=4".

Tabela 1: Teste funcional

| Estado    | Terminal          |
|-----------|-------------------|
| INICIO    |                   |
| MENSAGEM  | Digite a operação |
| EXPRESSAO | 2+2               |
| CALCULO   |                   |
| RESULTADO | =4                |
| INICIO    |                   |

## 14

Na figura 15 temos o *print size* obtido no experimento 11, comparando com a figura 14, referente ao experimento 10, podemos perceber que há um aumento nos três valores: **.txt** corresponde ao tamanho das instruções e dados constantes armazenados na memória flash, **.data** se refere a valores com inicialização armazenada na memória RAM e **.bss** variáveis sem inicialização armazenadas na memória RAM. Os 3 aumentos são esperados, visto que foram adicionadas variáveis globais ao programa e aumentou o número de funções que deve executar. Em detalhe, podemos comentar o grande aumento no valor para **.txt**.

| text  | data | bss  | dec   | hex  | filename                    |
|-------|------|------|-------|------|-----------------------------|
| 15256 | 60   | 2100 | 17416 | 4408 | cooler_multi_velocidade.elf |

Figura 14: *Print size* do projeto **cooler\_multi\_velocidade**

| text  | data | bss  | dec   | hex  | filename        |
|-------|------|------|-------|------|-----------------|
| 24696 | 88   | 2124 | 26908 | 691c | calculadora.elf |

Figura 15: *Print size* do projeto **calculadora**