EA871

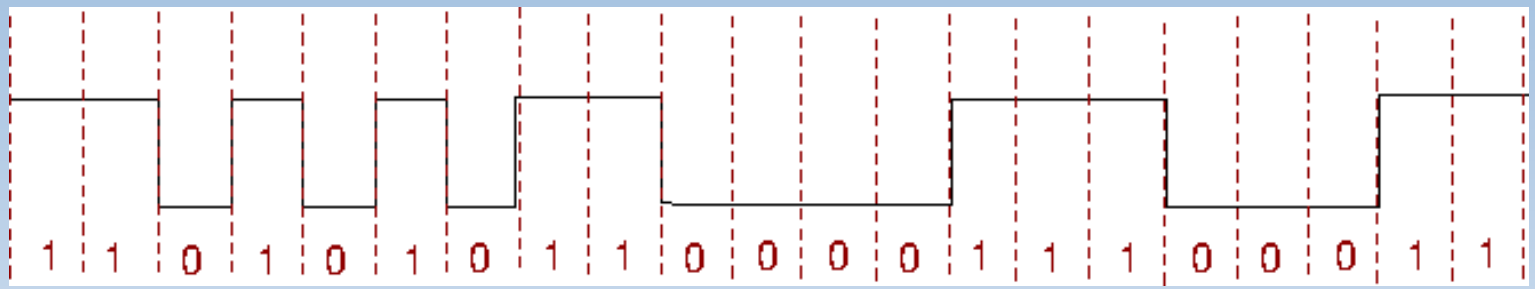
Comunicação Serial Assíncrona

UART

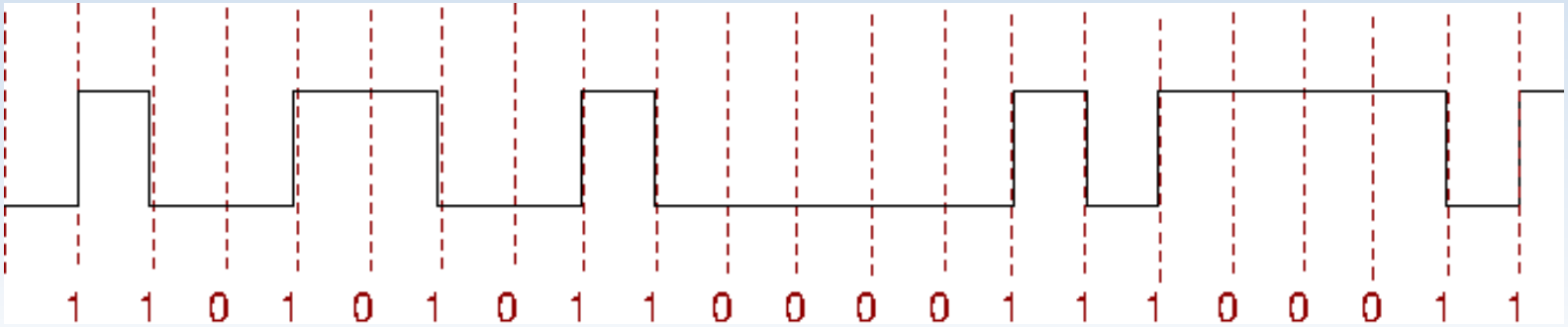
Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

Conceitos

- **Representação de dados binários por sinais digitais**:
  - **Não-Retorno-ao-Zero** (*Non-Return-to-Zero, NRTZ*): representa-se ‘0’ por um nível de tensão e ‘1’ por um outro nível de tensão.



Não-Retorno-ao-Zero Invertido (*Non-Return-to-Zero Inverted, NRTZI*): representa-se ‘1’ por uma transição de níveis de tensão e ‘0’ por não-transição.

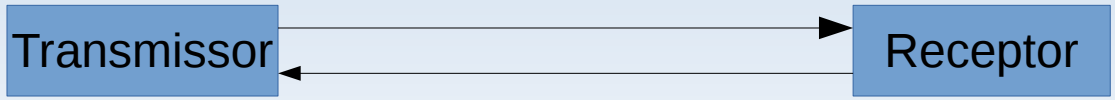


Conceitos

- **Modos de transmissão quanto aos sentidos de fluxos de dados entre dois dispositivos:**
  - **Simplex**: sentido unidirecional. Por exemplo, teclado e monitor.

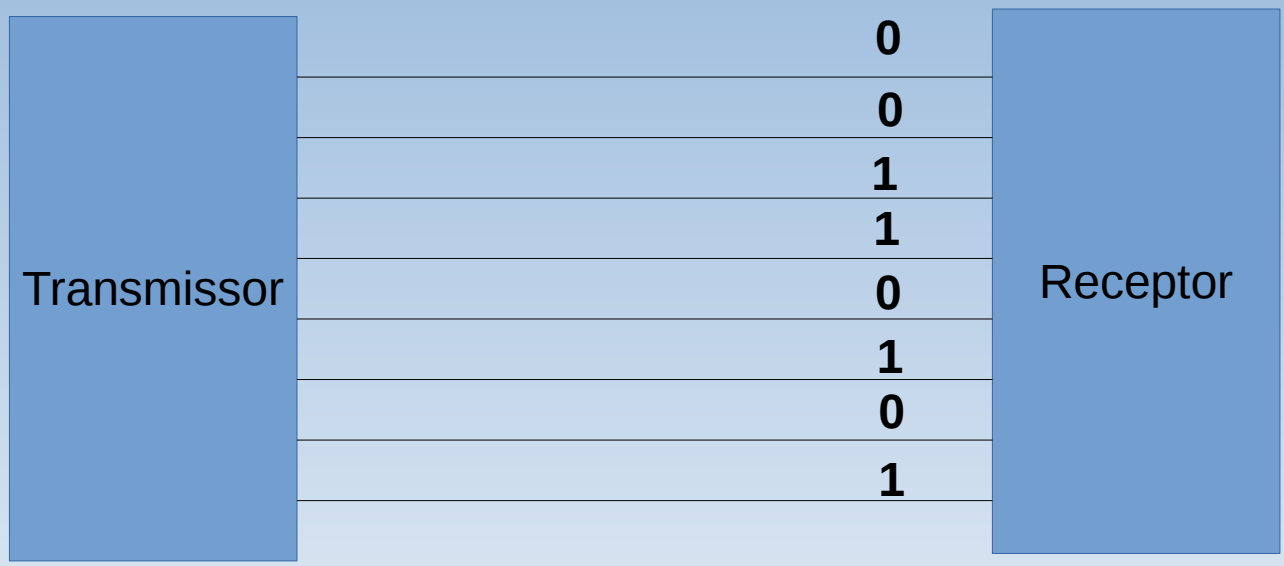
  - **Half-duplex**: sentido bi-direcional alternado. Por exemplo, walkie-talkie.

  - **Full-duplex**: sentido bidirecional. Por exemplo, telefone.

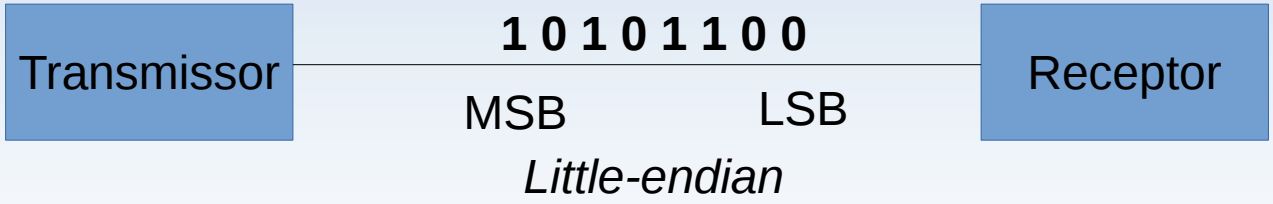


Conceitos

- **Modo de transmissão quanto à quantidade de *bits*** envolvidos numa transmissão:
  - **Transmissão paralela**: mais de um *bit* transmitidos ao mesmo tempo.

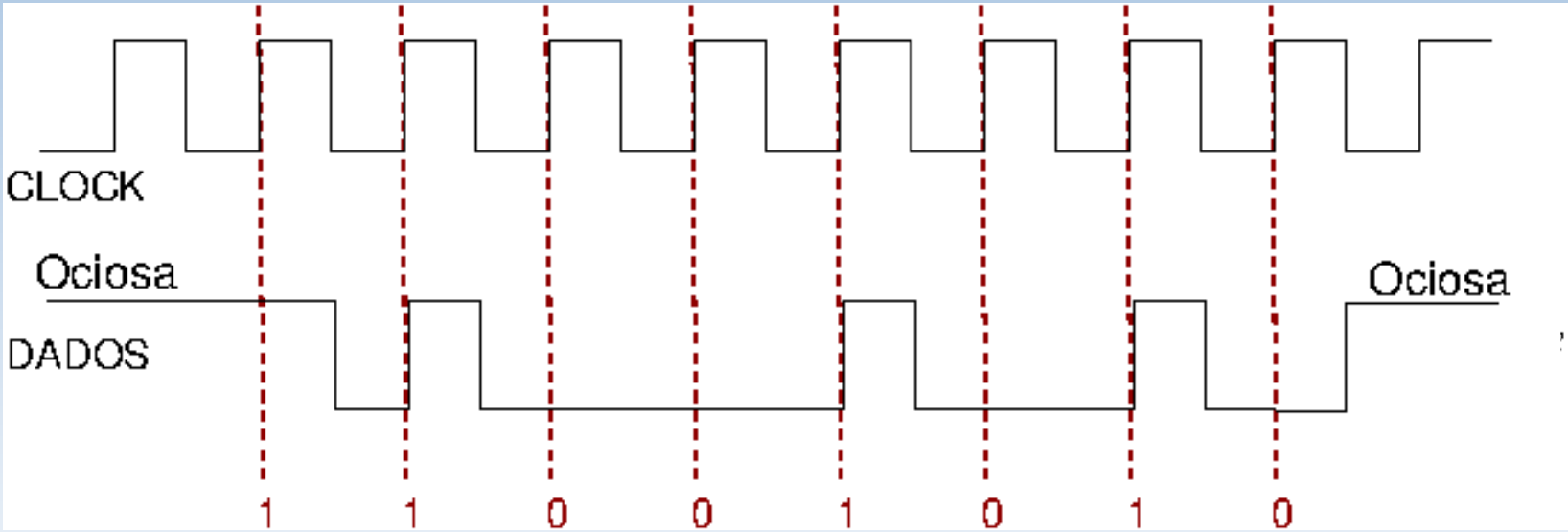


  - **Transmissão serial**: os sinais são transmitidos *bit* a *bit*.



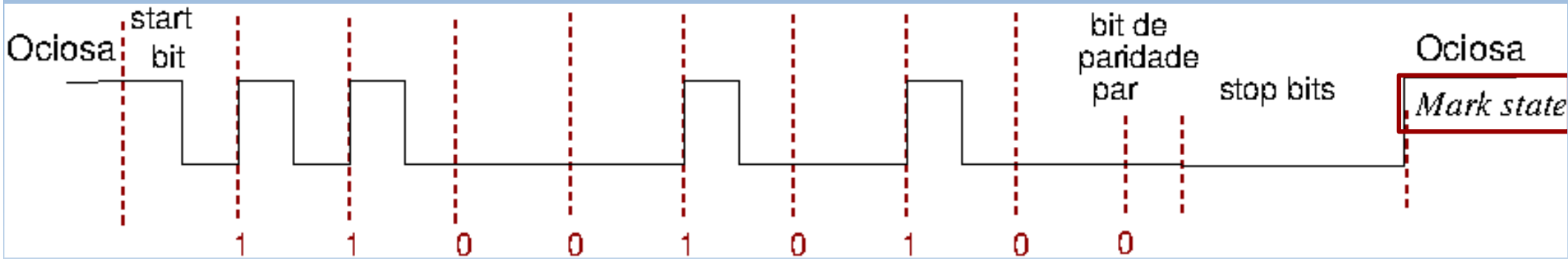
Conceitos

- **Modos de transmissão serial quanto ao compartilhamento do sinal de *clock*:**
  - **Síncrona**: quando o transmissor e o receptor compartilham o mesmo sinal de *clock*.



Conceitos

- **Assíncrona**: quando o sinconismo entre o transmissor e o receptor, em velocidades bem próximas, se dá por meio de uma série de regras para assegurar uma transferência robusta.



- ***Framing***: quantidade de *bits* de dados.
- ***Bits* de sincronização**: sinaliza o início de o fim dos *bits* de dados.
  - *Start bits*: indicam o início de uma transmissão e forçam a sincronização dos *clocks* de receptor e transmissor.
  - *Stop bits*: indicam o fim da transmissão dos *bits* de dados.

Conceitos

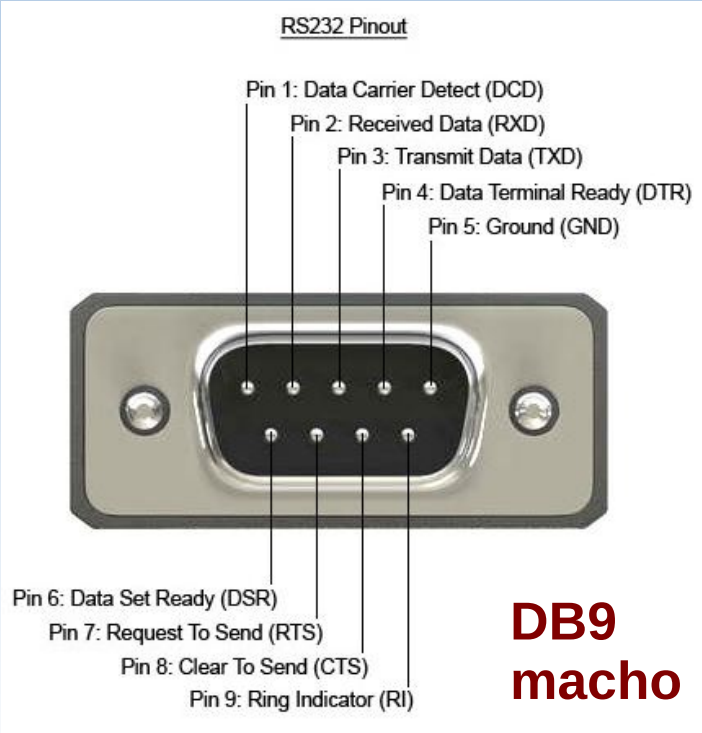
- ***Bit* de paridade:** *bit* adicional para checagem da consistência dos *bits* de dados. O seu valor é tal que a quantidade total de 1's seja par (paridade par) ou ímpar (paridade ímpar)**.**
- ***Baud rate* (taxa de *bits* por segundo):** velocidade de transferência de dados, usualmente medida por bps (*bits* por segundo). Por exemplo, 9600 bps.
- **Técnicas de controle de fluxo de dados**: estabelecem uma forma de sinalizar o início e o fim de uma transmissão dos dados, conhecida por ***handshaking***
  - por *hardware*: um circuito dedicado para geração de sinais de *aknowledgment* (ACK) entre RTS (*Request To Send*) e CTS (*Clear To Send*).
  - por *software*: caracteres de controle especiais para começar (XON, 0x11 em ASCII) e terminar (XOFF, 0x13 em ASCII) uma transmissão.

Conceitos

- **Ordem de *bits*** (*Endianness*) numa transmissão: ordem dos *bits* que compõem os *bits* de dados.
  - ***Little endian***: de *bit* menos significativo (LSB) para o mais (MSB).
  - ***Big endian***: de *bit* mais significativo (MSB) para o menos (LSB).
- **Protocolos de comunicação serial** (em microcontroladores): regras estabelecidas entre os dispositivos envolvidos numa comunicação serial para representar e interpretar os sinais digitais envolvidos.
  - **Síncrona**: SPI (*Serial Peripheral Interface*) e I2C (*Inter-Integrated Circuit*).
  - **Assíncrona**: RS-232 e RS-485

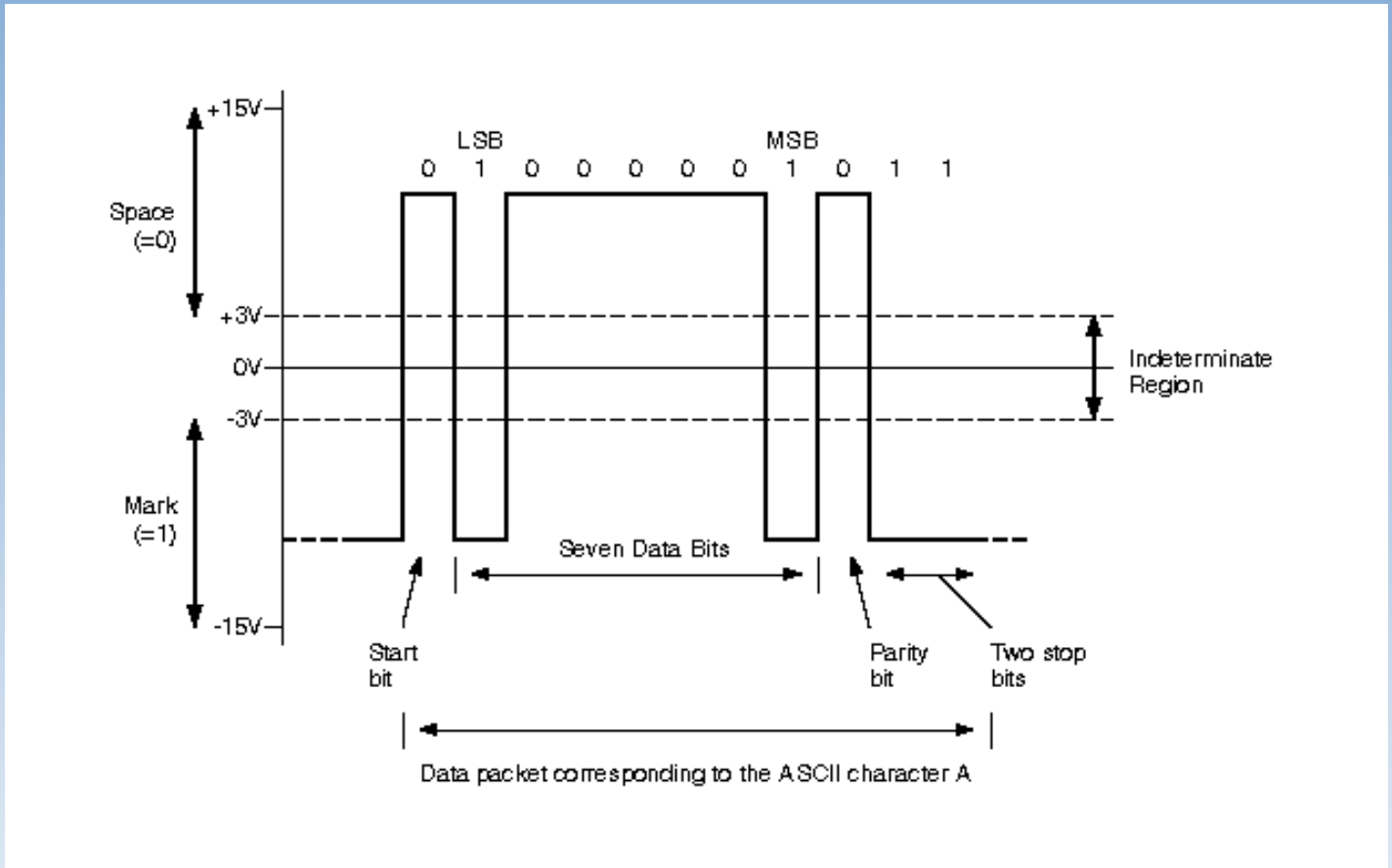
Protocolo RS-232

- Padrão de protocolo definido pela *Electronic Industries Association* (EIA) para transmissão síncrona (DB25) e assíncrona (DB9).
- Inclui especificações das características funcional, elétrica e temporal dos sinais numa transferência, inclusive a sua interface mecânica (conectores) com os equipamentos.



Fonte: https://www.arcelect.com/rs232.htm

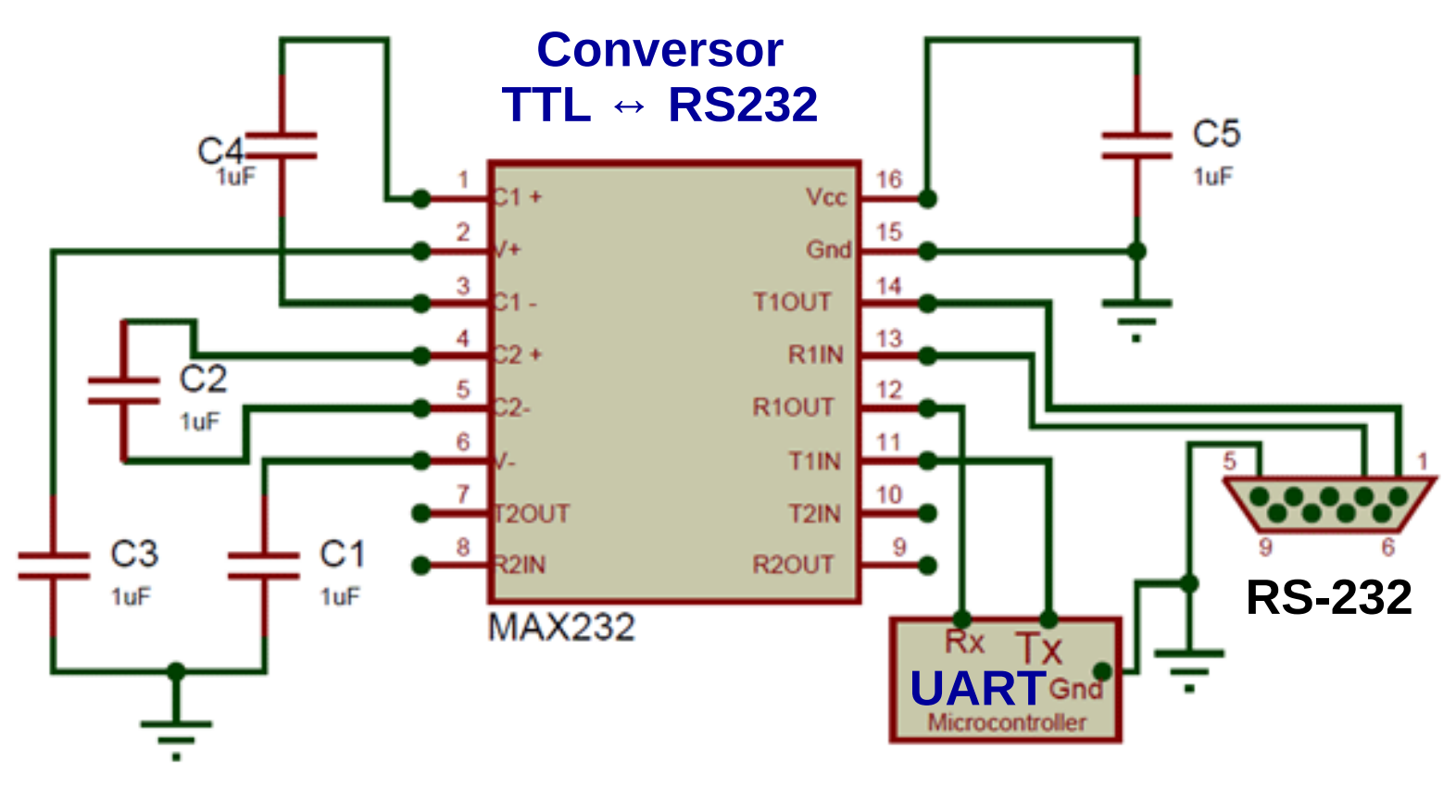
Protocolo RS-232



- **Linha Ociosa** (*Idle Line*)**:** o estado da linha de transmissão quando não está transmitindo (estado *Mark*, nível lógico 1)**.**
- **Sinal de Quebra** (*Break)*: recepção de valores contínuos de Espaços (estado *Space*, nível lógico 0) sem *start* e *stop bits.*

UART

- É um circuito que implementa todas as funções lógicas essenciais para geração dos sinais em conformidade com o protocolo assíncrono de RS-232, a menos dos níveis de tensão.

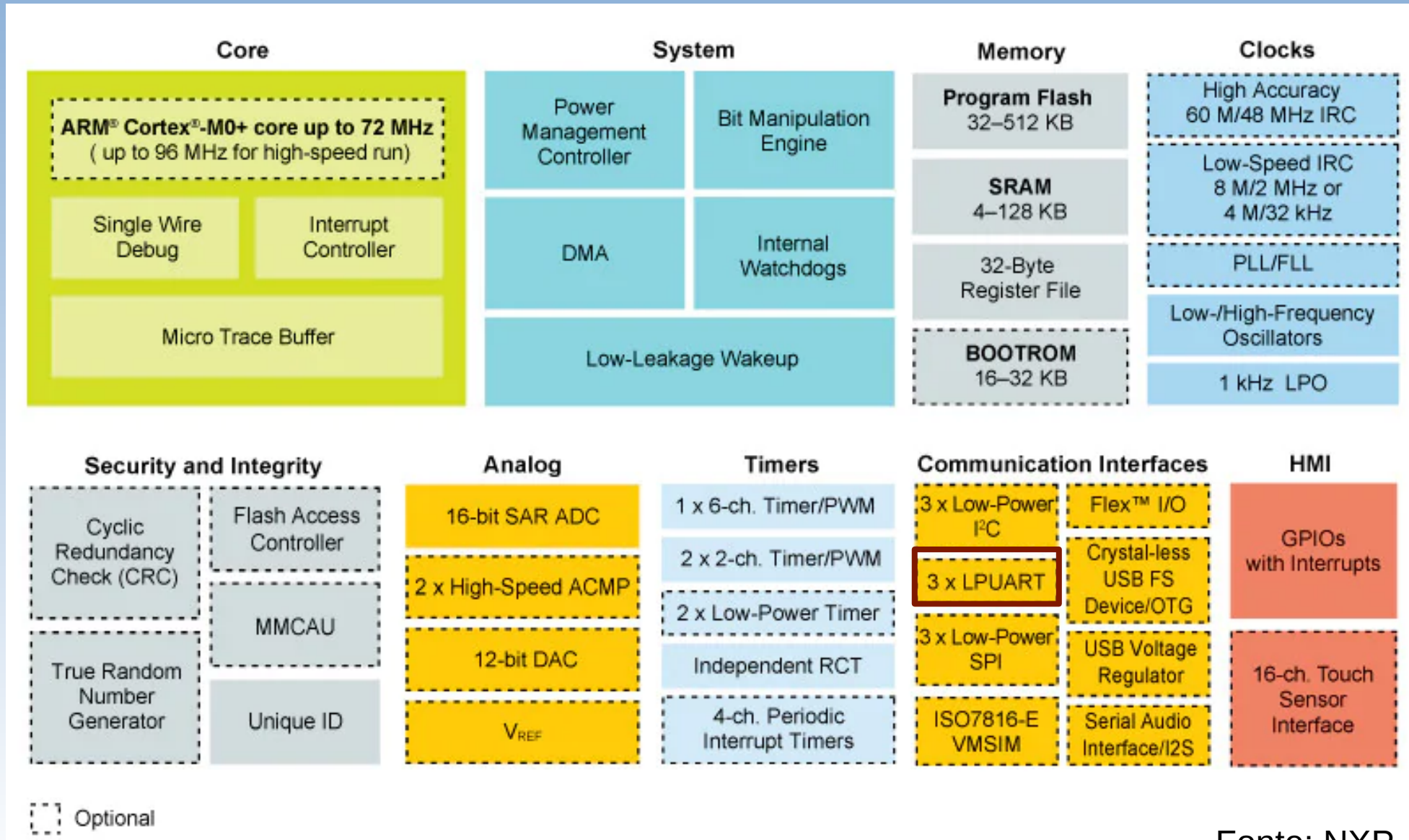


Fonte: https://components101.com/max232-pinout-features-datasheet

Conceitos

- **Taxa de amostragem por *bit***: quantidade de amostras por cada *bit* de um caractere de *bits* recebido.
  - Superamostragem: quando o número de amostras por *bit* é maior que 1.
- **Detecção de erros na transmissão**
  - **Erro de sobeposição** (*Overrun error*): indica a recepção de um novo caractere antes da conclusão do acesso de leitura do caractere anterior.
  - **Erro de paridade** (*Parity error*): indica que o *bit* de paridade recebido não corresponde ao valor esperado.
  - **Erro de enquadramento** (*Framing error*): indica que o *stop bit* não foi recebido conforme esperado.
  - **Erro de ruído** (*Noise error*): quando não há concordância entre as amostras de um *bit* no modo de superamostragem.

Microcontrolador Kinetis KL25Z



Fonte: NXP

UARTx

- UART1 e UART2
  - Full-duplex, formato NRTZ, com *buffer* de dados separado para transmissor e receptor.
  - Fonte de *clock*: *bus clock*.
  - Velocidade de transmissão, quantidade de *bits* de dados, quantidade de *stop bits* e polaridade do transmissor programáveis.
  - Monitoramento de diferentes estados por *polling* e por interrupção (recepção ocupada, transmissão ociosa, transmissão concluída, erros na transmissão).
  - Geração e checagem de *bits* de paridade
  - Geração de sinais de *break* cujo tamanho é programável
  - Interface com 5 canais do módulo DMA (acesso direto à memória)

UARTx

- UART0
  - Funções de UART1 e UART2.
  - Fonte de *clock* configurável.
  - Superamostragem dos sinais digitais dos *bits* (4x até 32x).
  - Suporta operação no modo Stop.
  - Polaridade do receptor programável.

UART0 – Fontes de *Clock*

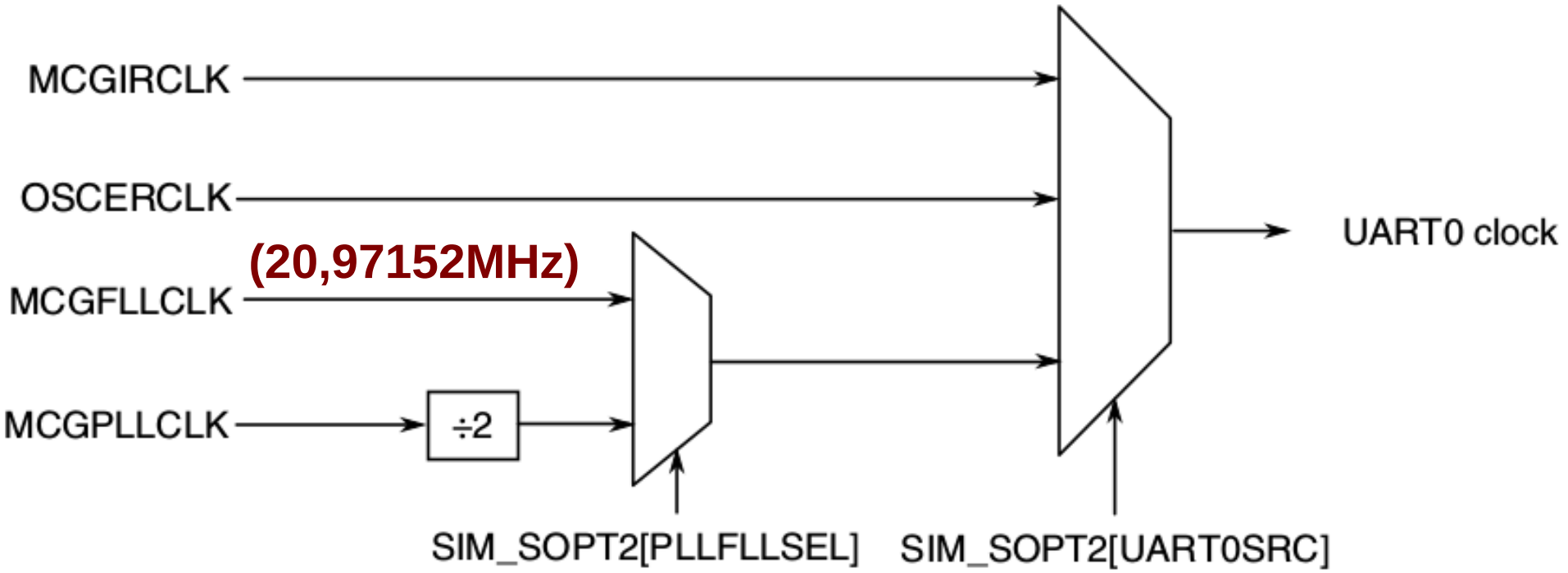


Figure 5-7. UART0 clock generation

UART0 - Transmissor

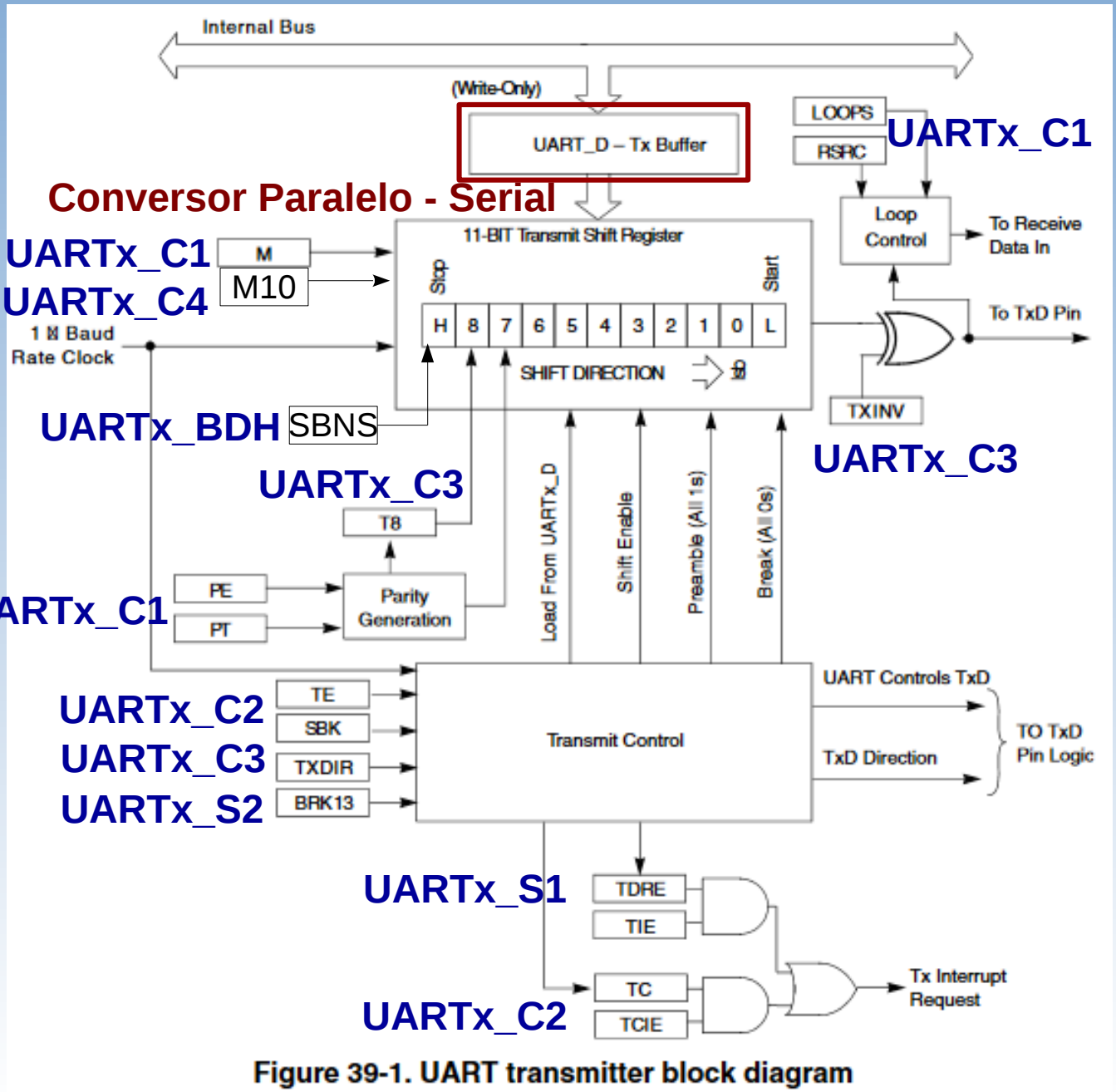


Figure 39-1. UART transmitter block diagram

Geração de *Baud Rate*

- O sinal da fonte de *clock* é dividido por um divisor de frequência de 13 *bits* configurados em 2 registradores (5 *bits* mais significativos em UART0_BDH [4:0] e 8 *bits* menos significativos em UART0_BDL[7:0]).

$$UARTx\,baud\,rate = \frac{Frequência\,da\,Fonte\,de\,Clock}{UART\,0:BDx[SBR]}$$

- No caso de superamostragem de cada *bit* (> 3x),

$$UART\,0\,baud\,rate = \frac{Frequência\,da\,Fonte\,de\,Clock}{UART\,0:BDx[SBR]*(UART\,0:C\,4[OSR]+1)}$$

Quando um valor inválido, fora do intervalo inteiro [3,31], é setado no campo OSR, assume-se o valor 15.
Quando a superamostragem é entre 4 (3+1) a 7 (6+1), a amostragem DEVE ser em ambas as bordas.

LOOP, RSRC, TXDIR

Full-Duplex (operação normal)

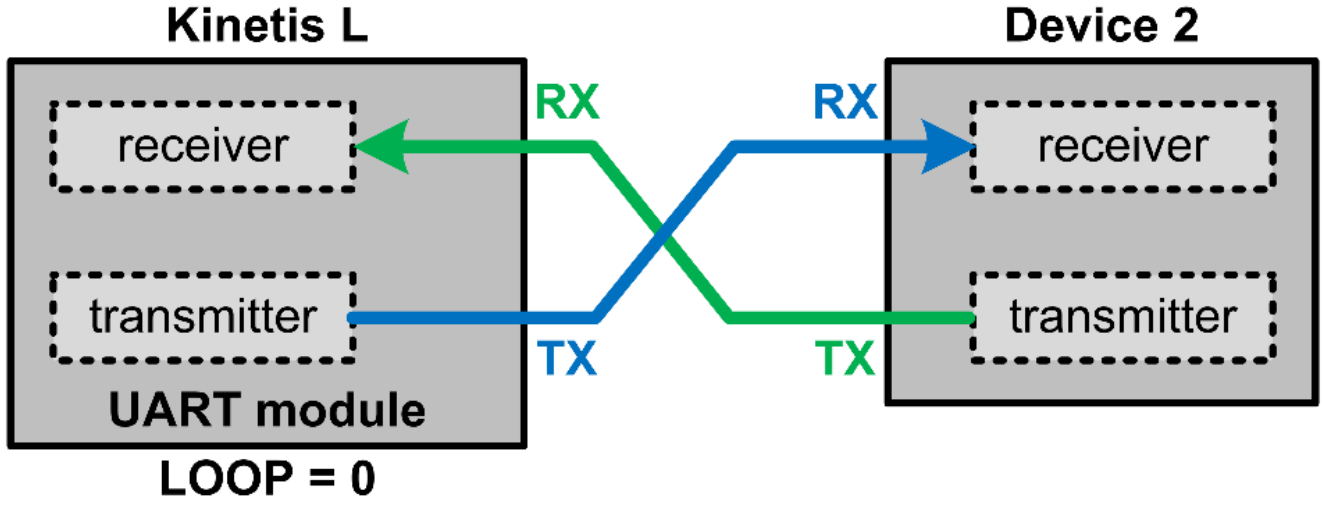


Figure 8-1. Block diagram of loop mode (disabled)

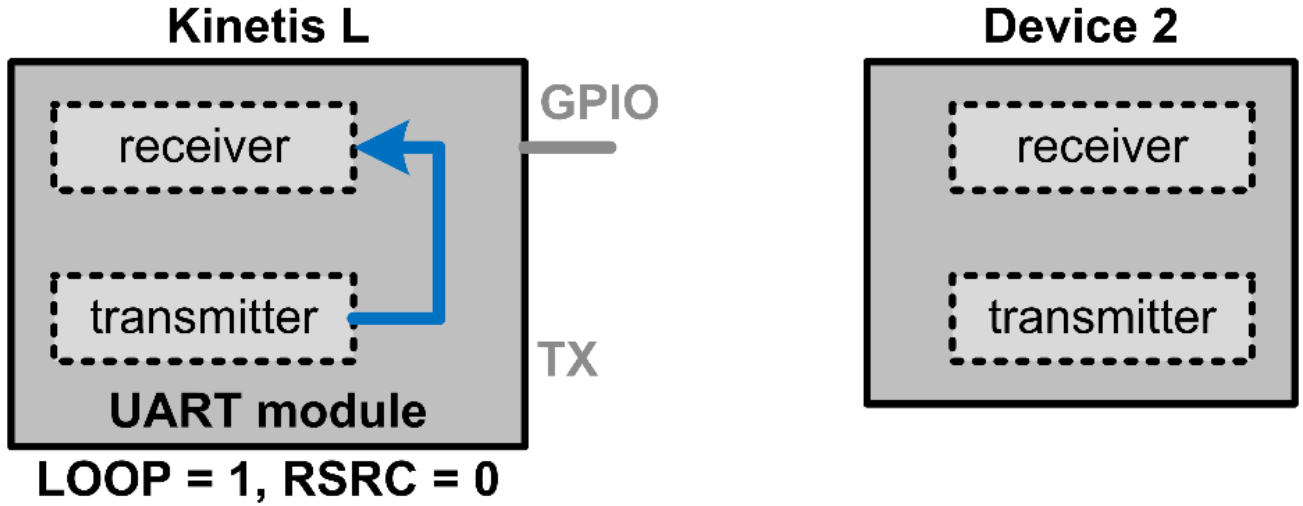


Figure 8-2. Block diagram of loop mode (enabled)

LOOP, RSRC, TXDIR

Half-Duplex

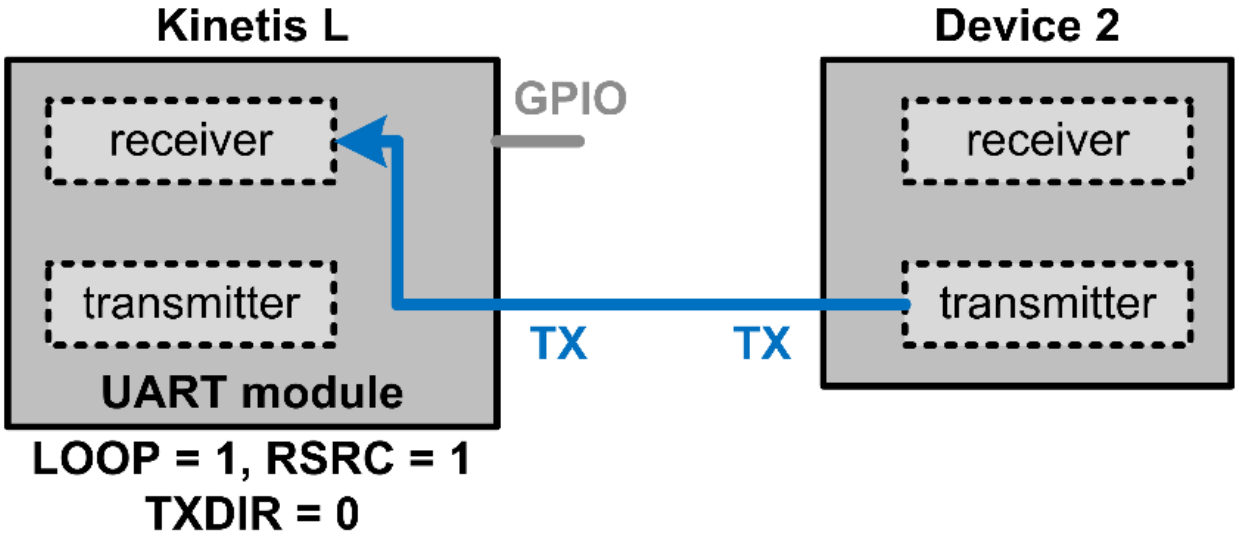


Figure 8-3. Block diagram of single wire mode (TXDIR disabled)

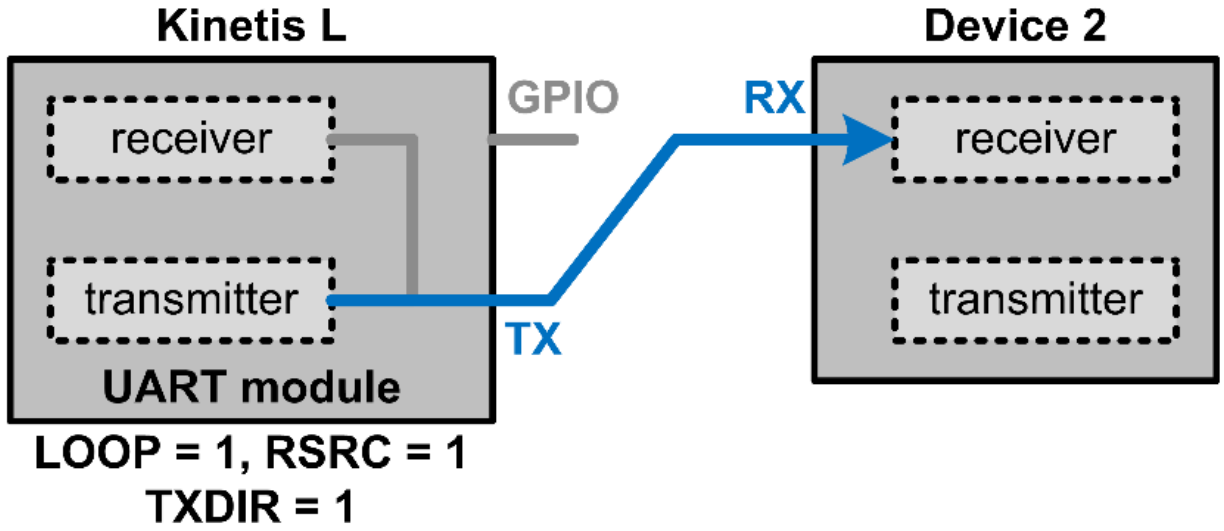


Figure 8-4. Block diagram of single wire mode (TXDIR enabled)

UART0 - Receptor

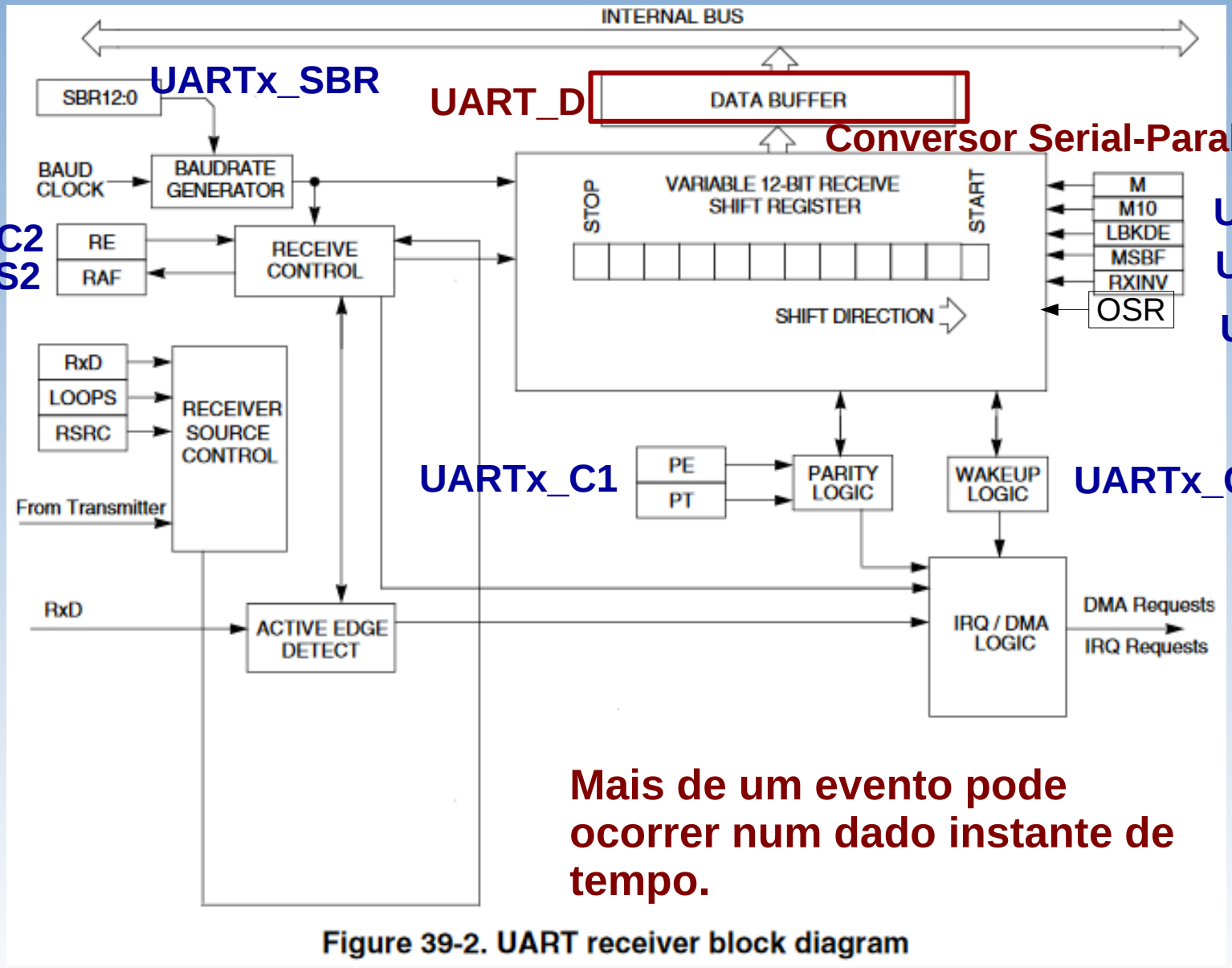


Figure 39-2. UART receiver block diagram

Estados Detectáveis

Address: Base address + 4h offset

Overrun error

Bit	7	6	5	4	3	2	1	0
Read	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Write				w1c	w1c	w1c	w1c	w1c
Reset	1	1	0	0	0	0	0	0

UARTx_S1 field descriptions

Noise error

Framing error

Parity error

Address: Base address + 5h offset

Bit	7	6	5	4	3	2	1	0
Read	LBKDIF	RXEDGIF	MSBF	RXINV	RWUID	BRK13	LBKDE	RAF
Write								
Reset	0	0	0	0	0	0	0	0

UARTx_S2 field descriptions

UART_RX Pin Active Edge

LIN Break Detect

Registradores

- SIM

  SIM_SOPT2*: selação de fonte de clock*

  SIM_SCGC4: habilitação da porta de clock

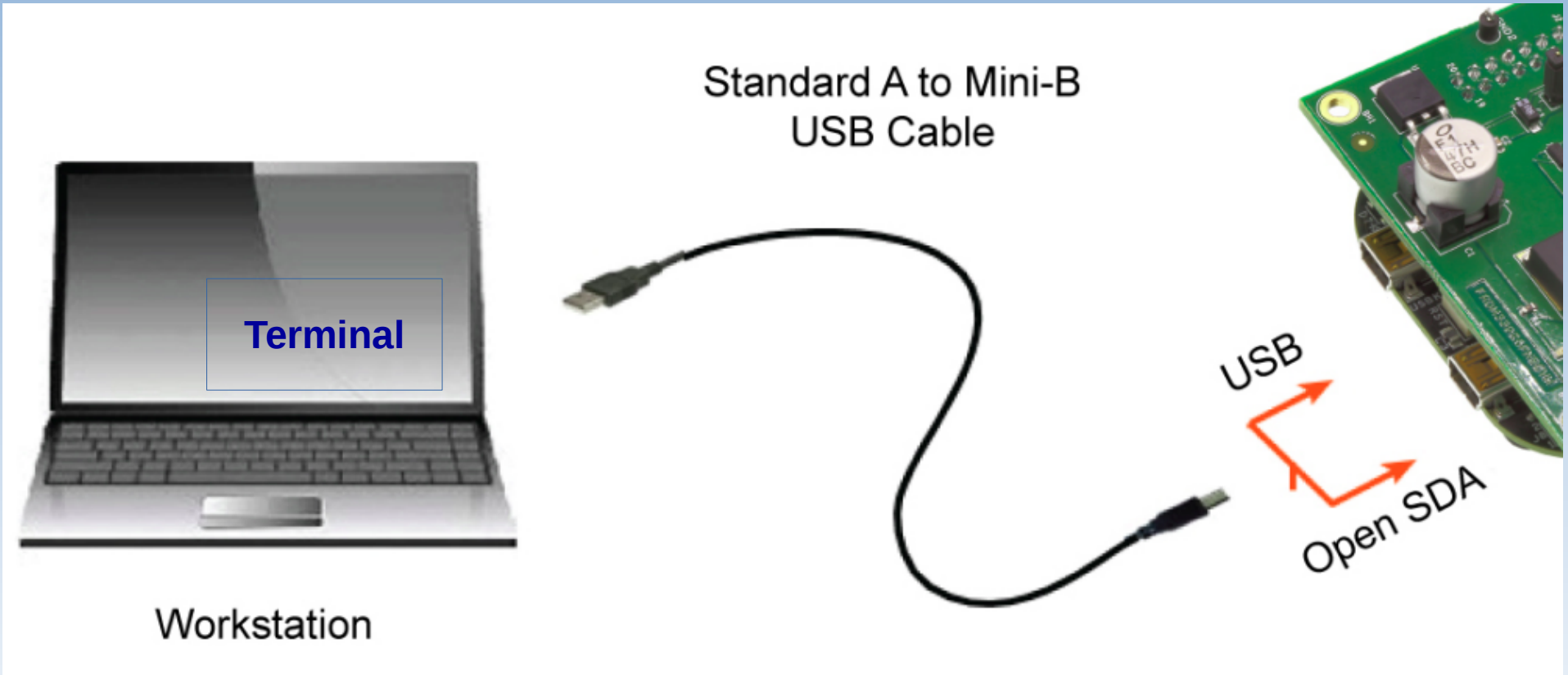
- PORTx
  - PORTx_PCRn: configuração da função de dois pinos para as funções RX e TX do modo *full-duplex* de uma comunicação serial assíncrona.
- NVIC: Gerenciamento da solicitação de interrupção IRQ 12
  - NVIC_ISER: habilitação de IRQs.
  - NVIC_ICER: limpeza das IRQs habilitadas.
  - NVIC_ISPR: habilitação de pendências.
  - NVIC_ICPR: limpeza das pendências.
  - NVIC_IPR3: atribuição de prioridade de atendimento.

Registradores

- UART0
  - UART0_BDH: configuração stop *bits*, *baud rate.*
  - UART0_BDL: configuração de *baud rate*.
  - UART0_C1: controle do modo de transmissão, *bit* de paridade e quantidade de *bits* de dados.
  - UART0_C2: controle da habilitação do transmissor, do receptor e das diversas interrupções.
  - UART0_S1: registrador de esatdo de transmissão, recepção e erros.
  - UART0_S2: registrador de estado da linha, controle da polaridade do receptor e da ordem dos *bits*.
  - UART0_C3: controle do modo de transferência, da polaridade do transmissor, da habilitação das interrupções por erros.
  - UART0_D: registrador de dados (de transmissão/de recepção).
  - UART0_M1, UART0_M2:endereço de “correspondência”.
  - UART0_C4: controle da habilitação de certificação de endereço, da quantidade de *bits* de dados e da superamostragem.
  - UART0_C5: controle de acesso direto à memória, resincronização e amostragem.’

Comunicação UART0 – Terminal

- **UART0 – mini-USB** e **USB – Terminal**



Fonte: https://www.nxp.com/docs/en/user-guide/KTFRDM33926PNBEVMUG.pdf

Comunicação UART0 - miniUSB

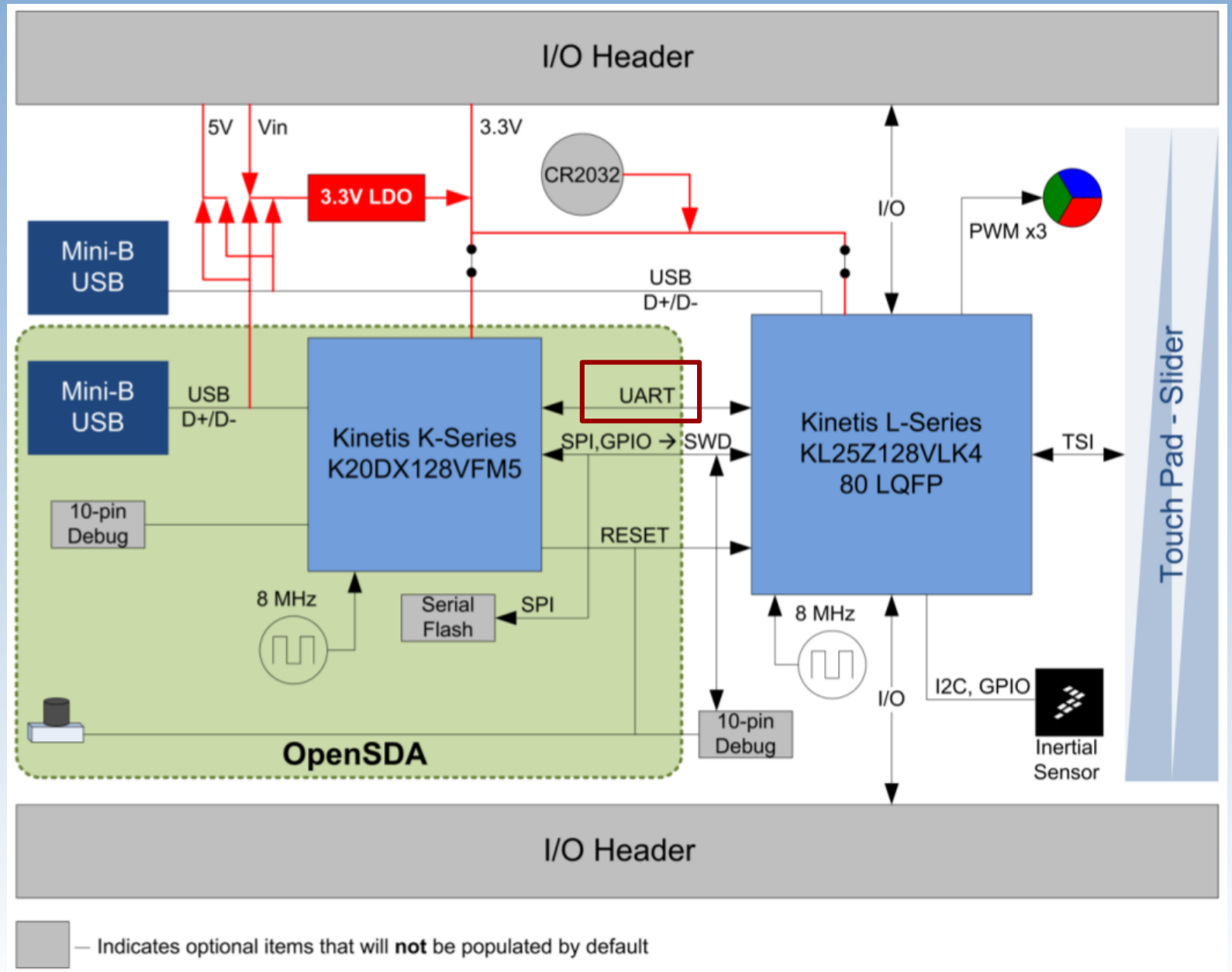
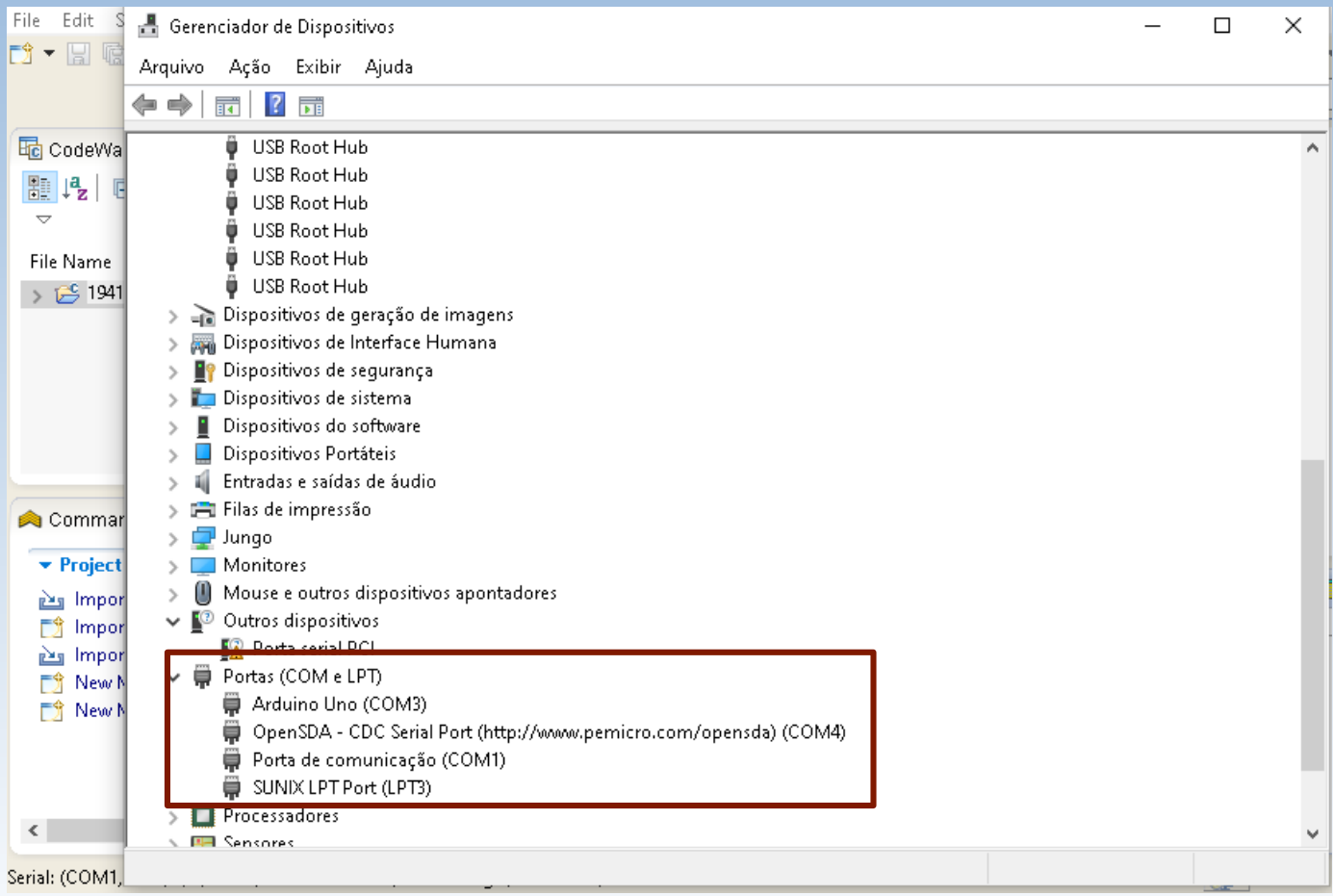


Figure 1. FRDM-KL25Z Block Diagram

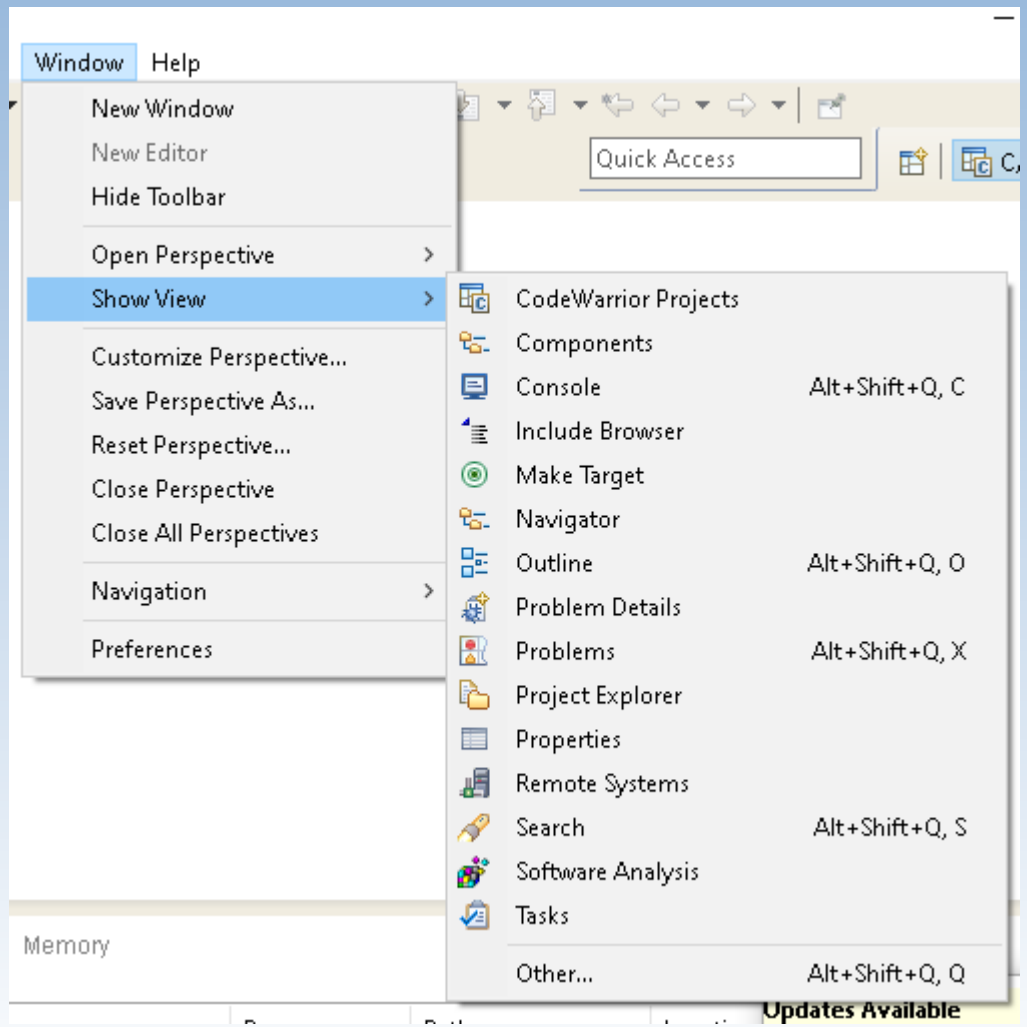
USB - Terminal

- **Emuladores de Terminais**: possibilitam acessos diretos às portas seriais e às conexões de rede (ssh, telnet).
  - PuTTy
  - TeraTerm
  - KiTTy
  - SmarTTy
  - RealTerm
  - RxTx em Eclipse

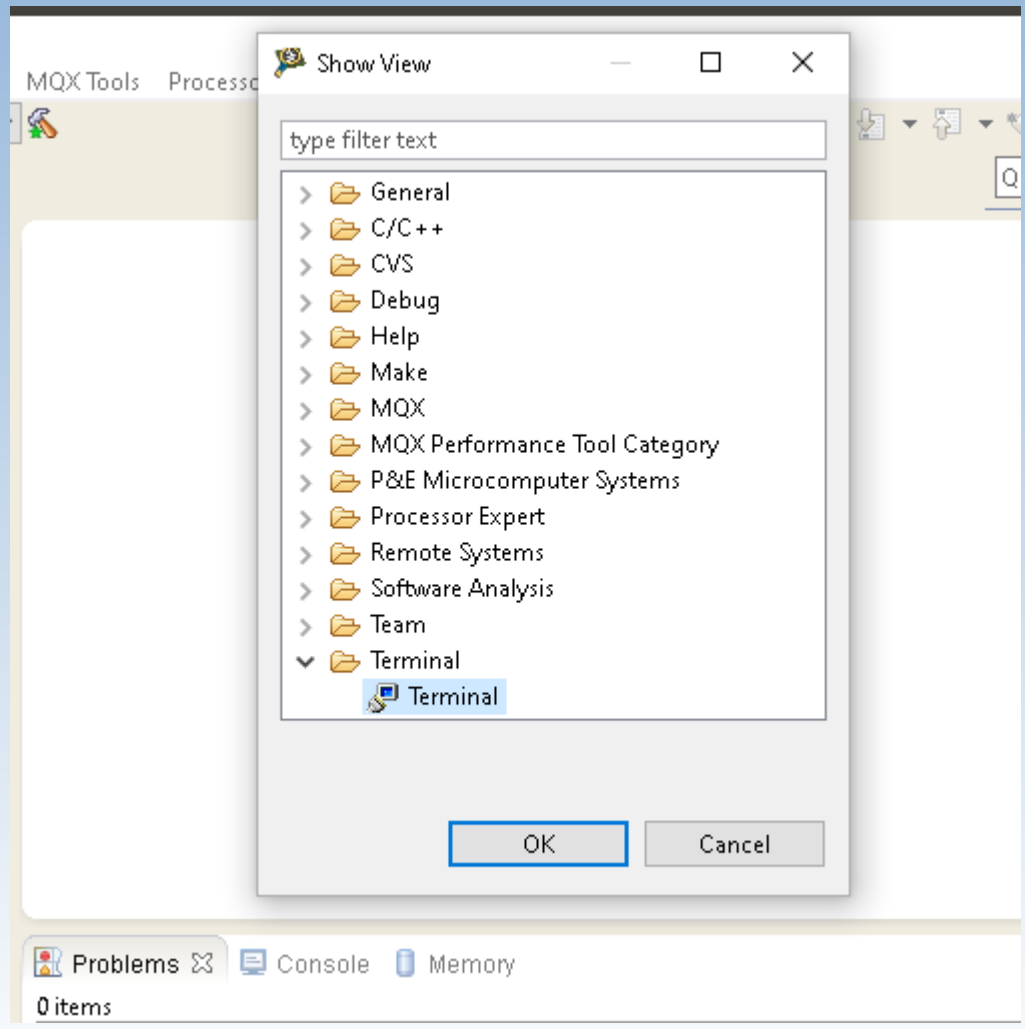


Terminal em IDE CodeWarrior

- Clique em “Others ...”



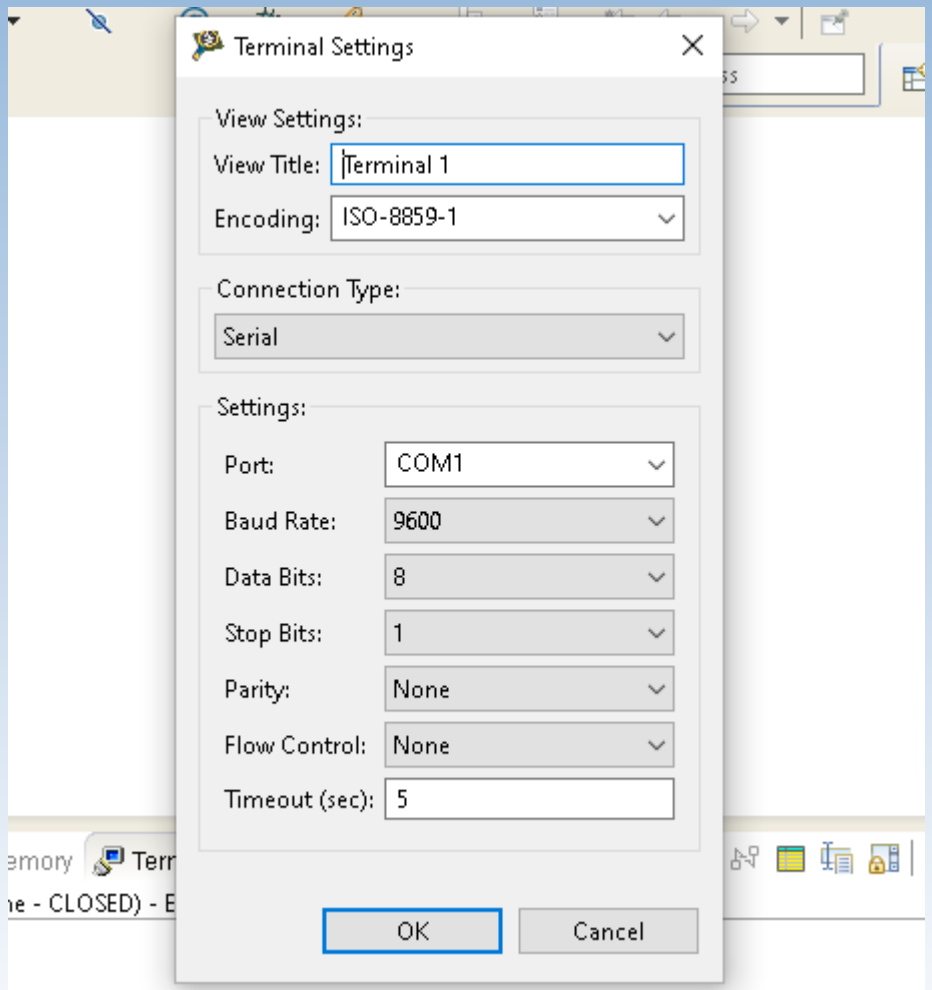
- Selecionar “Terminal”



Terminal em IDE CodeWarrior

- Ative o pop-up menu clicando no ícone ...

- Atribua os parâmetros especificados



Baud rate suportado:
300,1200,2400,4800,9600,19200,38400,
57600, 115200

Projeto-Exemplo

- Ecoar no “Terminal”, por *polling*, os caracteres digitados através do monitoramento dos estados das *flags* RDRF e TDRE (Exemplo 1 Cap. 8, *Kinetis Quick Reference*).

IDE Codewarrior
Terminal View

Superamostragem 16

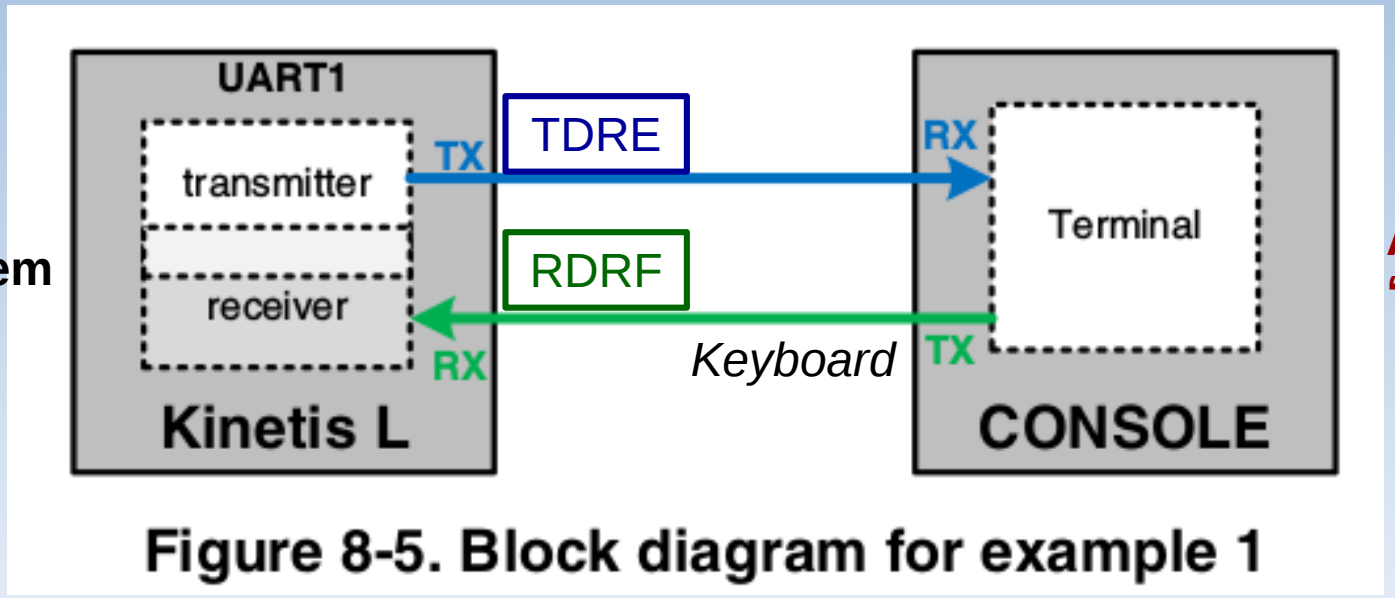
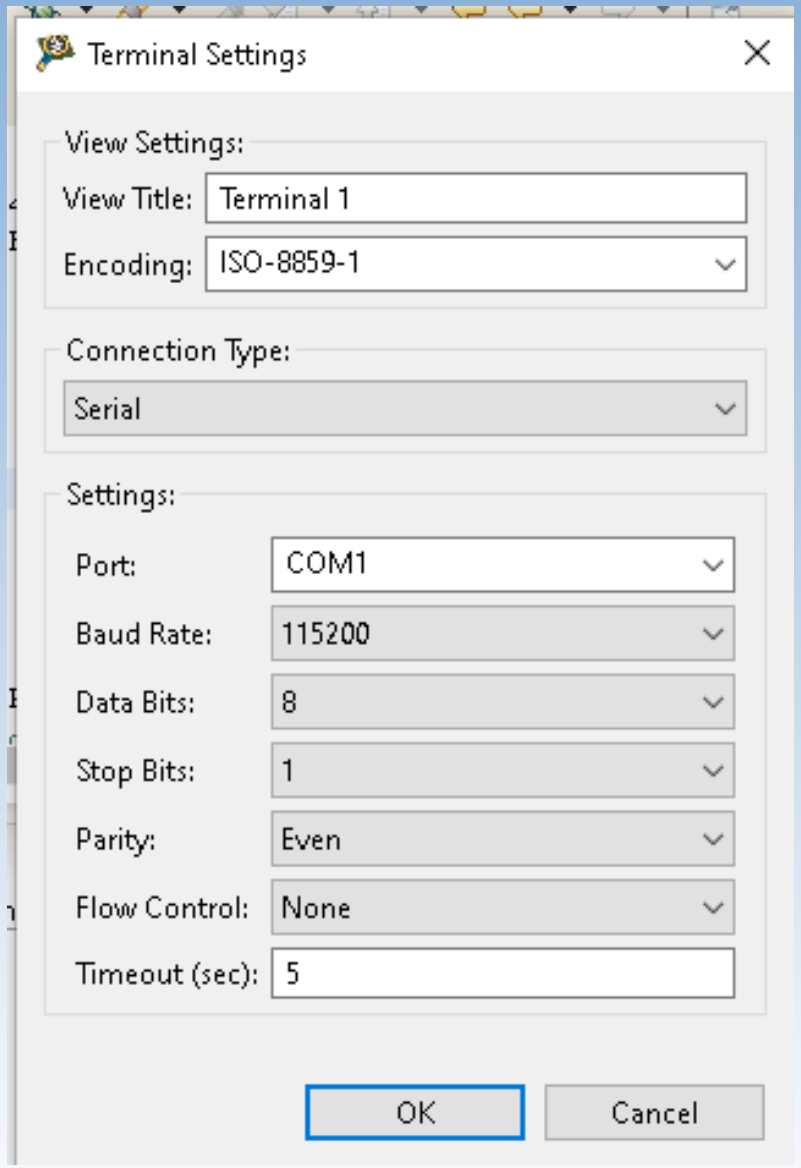


Figure 8-5. Block diagram for example 1

Aplicativo “Terminal”

Configuração do protocolo de comunicação: *baud rate (115200)*, quantidade de *bits* de dados (8 *bits*), paridade par, 1 *stop bit,* LSB primeiro, fluxo de controle NENHUM (não é suportado por Kinetis).

Configuração do Terminal



Configuração do Kinetis

- *Baud rate*

$$115200 = \frac{20971520}{UART0:BDx[SBR] * (15+1)} \Rightarrow UART0:BDx[SBR] = 11,3778 \rightarrow 12$$

  - UART0_BDH = 0x00
  - UART0_BDL = 0x0C
  - UART0_C4[OSR] = 0x0F
- Quantidade de *bits* de dados: 8 *bits*
  - UART0_C1[M] = 0;
- Quantidade de *stop bits*: 1 *bit*
  - UART0_BDH[SBNS] = 0;
- Paridade:
  - UART0_C1[PE] = 1;
  - UART0_C1[PT] = 0;
  - UART0_C1[M] = 1;

Técnicas de Programação

Uso do tipo de dados struct para facilitar a configuração do módulo:

```
typedef struct  UART0Configuration {
    uint8_t bdh_sbns;      // selecionar quantidade de stop bits (0 = 1; 1 = 2)
    uint16_t sbr;          // baud rate
    uint8_t c1_loops;      // operacao em loop (normal = 0)
    uint8_t c1_dozeen;     // habilitar espera (doze)
    uint8_t c1_rsrc;       // habilitar a saida do TX em operação de loop
    uint8_t c1_m;          // 8-bit (0) ou 9-bit de dados
    uint8_t c1_wake;       // forma de wakeup do RX (0-idle line; 1-address-mark)
    uint8_t c1_ilt;        // selecionar a forma de deteccao de "idle line"
    uint8_t c1_pe;         // habilitar paridade
    uint8_t c1_pt;         // tipo de paridade (0-par; 1-impar)
    uint8_t c2_rwu;        // setar o RX no estado de standby aguardando pelo wakeup
    uint8_t c2_sbk;        // habilitar o enfileiramento de caracteres break
    uint8_t s2_msbf;       // setar endianess para MSB (0 = LSB;1 = MSB)
    uint8_t s2_rxinv;      // habilitar a inversao da polaridade dos bits do RX
```

Técnicas de Programação

```
    uint8_t s2_rwuid;  // habilitar o set do IDLE bit durante standby do RX
    uint8_t s2_brk13; // selecionar o comprimento de caracter break (0=10 bits; 1=13 bits)
    uint8_t s2_lbkde;  // habilitar deteccao de caractere break longo
    uint8_t c3_r8t9;    // bit 8 (RX)/bit 9 (TX)
    uint8_t c3_r9t8;    // bit 9 (RX)/bit 8 (TX)
    uint8_t c3_txdir;    // se RSRC=1, configurar o sentido de TX (0=entrada; 1=saida)
    uint8_t c3_txinv;   // habilitar a inversao da polaridade dos bits de TX
    uint8_t c4_maen1;          // habilitar controle de "match address" 1
    uint8_t c4_maen2;          // habilitar controle de "match address" 2
    uint8_t c4_m10;    // selecionar o modo de bits (0=8/9 bits;1=10 bits)
    uint8_t c4_osr;     // taxa de super-amostragem (default=16(0b01111)
    uint8_t c5_tdmae;          // habilitar transmissao por DMA
    uint8_t c5_rdmae;          // habilitar recepcao por DMA
    uint8_t c5_bothedge;       // amostrar os dados em ambas as bordas do clock de baud rate
    uint8_t c5_resyncdis; // desabilitar o resincronismo nas transições de 1 para 0
} UART0Config;
```

Técnica de Programação

- Quantidade de amostras por *bit:* [4,32]
  - Se escrever 0 em UART0_C4[OSR] → atribui-se 16 no campo!
  - **Como setar valores >= 3 sem limpar os *bits* do campo?**

    UART0_C4 |= (0b11111 << UART0_C4_OSR_SHIFT);
  - UART0_C4 &= ((config->c4_osr << UART0_C4_OSR_SHIFT) | ~UART0_C4_OSR_MASK);
- **Divisor de frequência em dois registradores**

  config->sbr |= (config->sbr & 0x1FFF);

  UART0_BDH &= ~UART0_BDH_SBR(0x1F);

  UART0_BDL &= ~UART0_BDL_SBR(0xFF);

  UART0_BDH |= UART0_BDH_SBR(config->sbr >> 8);

  UART0_BDL |= UART0_BDL_SBR(config->sbr & 0x00FF);

Técnicas de Programação

- Modularização de códigos

  SIM_selPLLFLL (uint8_t base); // selecionar base de tempo

  SIM_initBusClock (uint8_t OUTDIV4); // inicializar fonte de clock

  PORTx_muxUART0RxTxPins(void); // configurar pinos para função Rx e Tx

  UART0_init (UART0Config *config); // inicializar e habilitar UART0

  UART0_reconfig(UART0Config *config); // reconfigurar os parâmetros de UART0

  UART0_enableRE (void); // habilitar canal RX

  UART0_enableTE (void); // habilitar canal TX

  UART0_disableRE (void); // desabilitar canal RX

  UART0_disableTE (void); // desabilitar canal TX

Pseudocódigo

SIM_initBusClock (0b000); // Divisor de frequencia em 1

SIM_selPLLFLL (0); // selecionar MCGFLLCLK (20971520Hz)

PORTx_muxUART0RxTxPins();

config.c4_osr = 15;

config.sbr = (uint16_t)(20971520./(115200 * (config.c4_osr+1)));

UART0_init(&config);

Laço:

Se há dado no receptor, d = UART0_D;

Se o transmissor está ocioso, UART0_D = d;

CodeWarrior IDE Development Suite

Informações Adicionais

- Serial Terminal Basics

  https://learn.sparkfun.com/tutorials/terminal-basics/all
- A UART: O que é e como funciona.

  https://www.freebsd.org/doc/pt_BR.ISO8859-1/articles/serial-uart/
- RS-232 Data Interface: a Tutorial on Data Interface and cables

  https://www.arcelect.com/rs232.htm
- KL25 Sub-Family Reference Manual

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf
  - Clock Distribution: Capítulo 5 (página 123)
  - SIM: Capítulo 12 (páginas 200, 207)
  - UART0: Capítulo 39 (página 721)

Informações Adicionais

- Kinetis L Peripheral Module Quick Reference (Rev. 0.09/2012) – Cap. 8

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KLQRUG.pdf

- How to Analyze UART?

  https://support.saleae.com/tutorials/example-projects/how-to-analyze-uart

- 4 HyperTerminal Alternatives for Windows 10

  https://helpdeskgeek.com/free-tools-review/windows-7-8-10-hyperterminal/

-

EA871

Comunicação Serial Assíncrona

Interrupção

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

Estados Detectáveis

Address: Base address + 4h offset

Overrun error

Bit	7	6	5	4	3	2	1	0
Read	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Write				w1c	w1c	w1c	w1c	w1c
Reset	1	1	0	0	0	0	0	0

UARTx_S1 field descriptions

Noise error

Framing error

Parity error

Address: Base address + 5h offset

Bit	7	6	5	4	3	2	1	0
Read	LBKDIF	RXEDGIF	MSBF	RXINV	RWUID	BRK13	LBKDE	RAF
Write								
Reset	0	0	0	0	0	0	0	0

UARTx_S2 field descriptions

UART_RX Pin Active Edge

LIN Break Detect

Habilitação de Interrupção

Address: Base address + 3h offset

Bit	7	6	5	4	3	2	1	0
Read Write	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

UARTx_C2 field descriptions

Address: Base address + 6h offset

Bit	7	6	5	4	3	2	1	0
Read Write	R8T9	R9T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
Reset	0	0	0	0	0	0	0	0

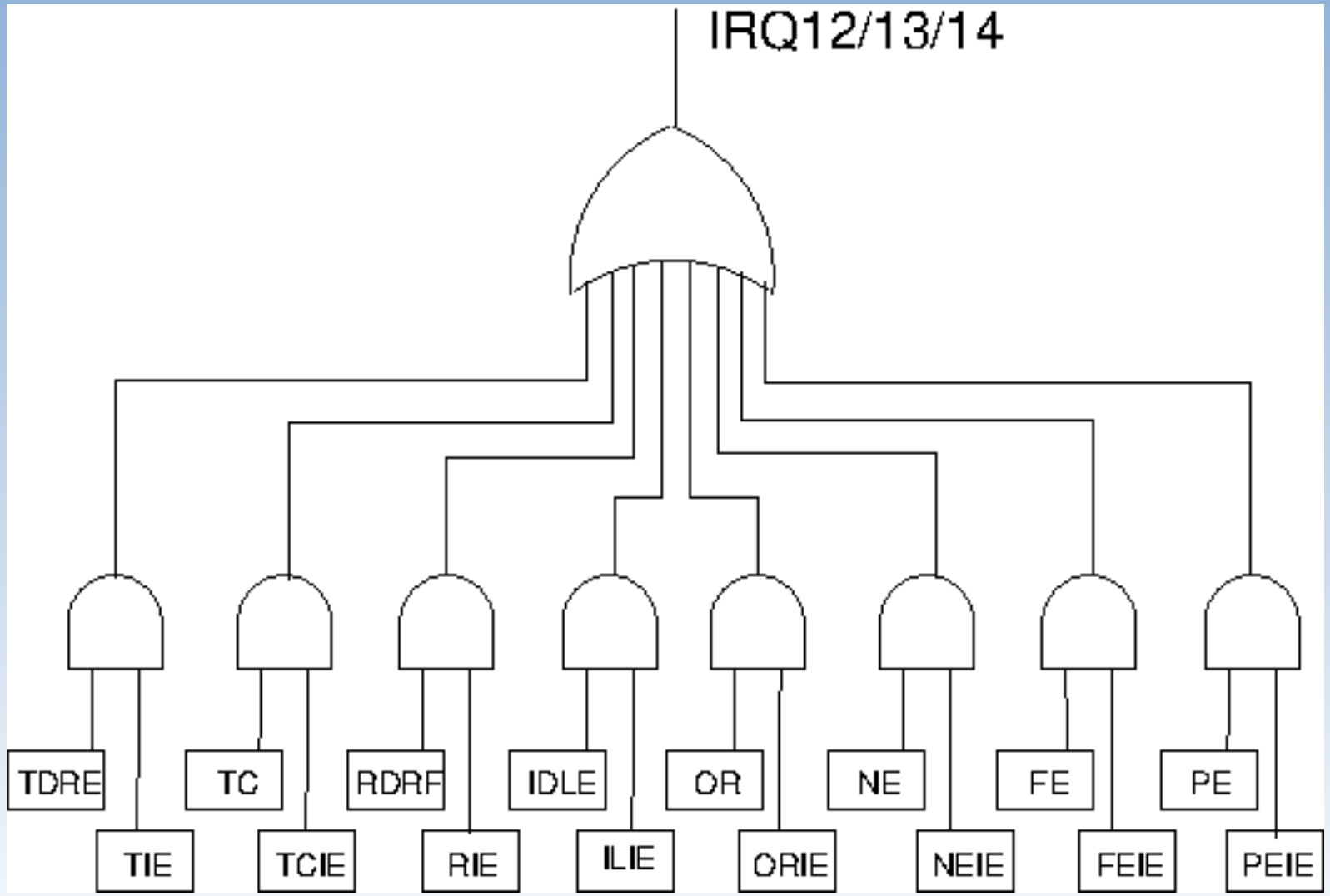
UARTx_C3 field descriptions

UARTx e NVIC

- É associado um IRQ a cada módulo UARTx.

Address	Vector	IRQ[1]	NVIC IPR register number[2]	Source module	Source description
ARM Core System Handler Vectors					
Non-Core Vectors					
0x0000_0040	16	0	0	DMA	DMA channel 0 transfer complete and error
0x0000_0044	17	1	0	DMA	DMA channel 1 transfer complete and error
0x0000_0048	18	2	0	DMA	DMA channel 2 transfer complete and error
0x0000_004C	19	3	0	DMA	DMA channel 3 transfer complete and error
0x0000_0050	20	4	1	—	—
0x0000_0054	21	5	1	FTFA	Command complete and read collision
0x0000_0058	22	6	1	PMC	Low-voltage detect, low-voltage warning
0x0000_005C	23	7	1	LLWU	Low Leakage Wakeup
0x0000_0060	24	8	2	I²C0	
0x0000_0064	25	9	2	I²C1	
0x0000_0068	26	10	2	SPI0	Single interrupt vector for all sources
0x0000_006C	27	11	2	SPI1	Single interrupt vector for all sources
0x0000_0070	28	12	3	UART0	Status and error
0x0000_0074	29	13	3	UART1	Status and error
0x0000_0078	30	14	3	UART2	Status and error

Ativação de IRQ



Processamento de IRQs

```
void NVIC_enableUART0/1/2(char priority)
{
    NVIC_ISER |= 1 << 12/13/14 ;              // Habilitar IRQ12/13/14
    NVIC_ICPR |= 1 << 12/13/14 ;          // Limpar as pendencias em solicitacao de interrupcoes
    NVIC_IPR3 |= NVIC_IP_PRI_12/13/14(priority << 6); ///< seta prioridade

}
```

- Definição das rotinas de serviço

```
void UART0_IRQHandler() __attribute__ ((weak, alias("Default_Handler")));
void UART1_IRQHandler() __attribute__ ((weak, alias("Default_Handler")));
void UART2_IRQHandler() __attribute__ ((weak, alias("Default_Handler")));
```

Limpar os estados dos eventos

Bit	7	6	5	4	3	2	1	0
Read	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Write				w1c	w1c	w1c	w1c	w1c
Reset	1	1	0	0	0	0	0	0

UARTx_S1 field descriptions

Field	Description
7 TDRE	Transmit Data Register Empty Flag TDRE is set out of reset and whenever there is room to write data to the transmit data buffer. To clear TDRE, write to the UART data register (UART _D). 0 Transmit data buffer full. 1 Transmit data buffer empty.
6 TC	Transmission Complete Flag TC is set out of reset and when TDRE is set and no data, preamble, or break character is being transmitted. TC is cleared automatically by one of the following: • Write to the UART data register (UART _D) to transmit new data • Queue a preamble by changing TE from 0 to 1 • Queue a break character by writing 1 to UART _C2[SBK] 0 Transmitter active (sending data, a preamble, or a break). 1 Transmitter idle (transmission activity complete).

RDRF To clear RDRF, read UART_D.

Limpar os estados dos eventos

```
UART0_IRQHandler() {static char d;
    if (UART0_S1 & UART0_S1_OR_MASK) {
        UART0_S1 |= UART0_S1_OR_MASK;
    } else if (UART0_S1 & UART0_S1_NF_MASK) {
        UART0_S1 |= UART0_S1_NF_MASK;
    } else if (UART0_S1 & UART0_S1_FE_MASK) {
        UART0_S1 |= UART0_S1_FE_MASK;
    } else if (UART0_S1 & UART0_S1_PF_MASK) {
        UART0_S1 |= UART0_S1_PF_MASK;
    } else if (UART0_S1 & UART0_S1_RDRF_MASK) {
        d = UART0_D;
    } else if (UART0_S1 & UART0_S1_IDLE_MASK) {
        UART0_S1 |= UART0_S1_IDLE_MASK;
    } else if (UART0_S1 & UART0_S1_TDRE_MASK) {
        UART0_D = d;
    }
```

Transmissor

Address: Base address + 4h offset

Bit	7	6	5	4	3	2	1	0
Read	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Write				w1c	w1c	w1c	w1c	w1c
Reset	1	1	0	0	0	0	0	0

UARTx_S1 field descriptions

- *Reset*
  - *Buffer* de dados de saída vazio.
    - TDRE e TC em 1.
  - Se TIE e/ou TCIE habilitados,
    - geram-se eventos de interrupção
    - como não se faz nenhum acesso de escrita, a solicitação não é limpa → programa fica preso no atendimento de TIE/TCIE.
- **Só habilitar TIE e/ou TCIE, quando os dados estiverem prontos para envio**.

Projeto-Exemplo

- Ecoar no “Terminal”, por interrupção, os caracteres digitados (Exemplo 1 Cap. 8, *Kinetis Quick Reference*), e controlar a piscada da cor amarela do ledRGB.

IDE Codewarrior
Terminal View

Superamostragem 8

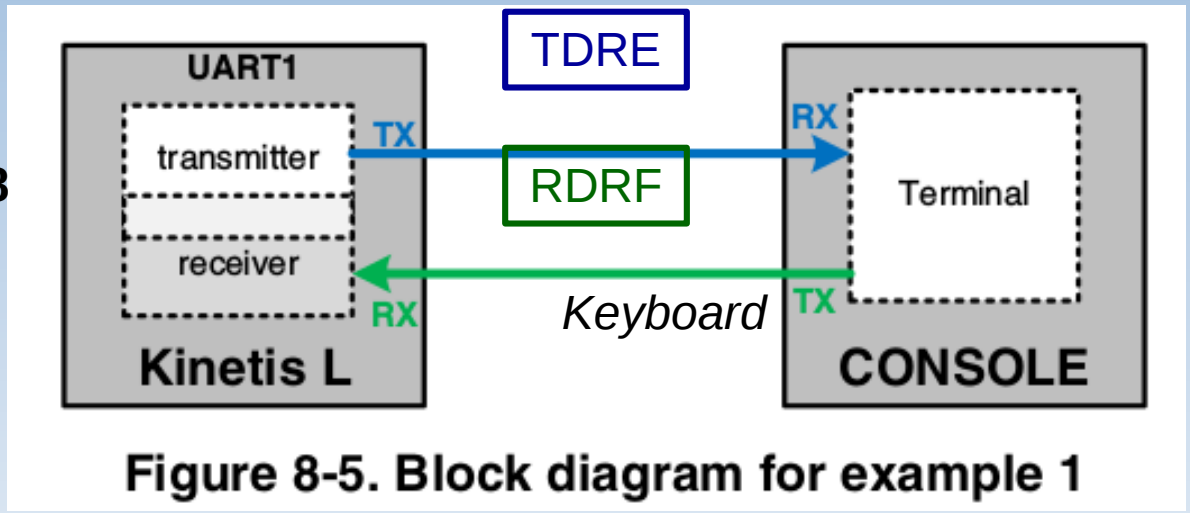


Figure 8-5. Block diagram for example 1

Aplicativo “Terminal”

Configuração do protocolo de comunicação: *baud rate (19200)*, quantidade de *bits* de dados (8 *bits*), sem paridade, 2 *stop bits,* LSB primeiro, fluxo de controle NENHUM (não é suportado por Kinetis).

Técnicas de Programação

- Uso do tipo de dados struct para facilitar a configuração do módulo:

```
typedef struct UART0ConfigInterrupts_tag {
    uint8_t c2_tie;      // habilitar interrupcao TX
    uint8_t c2_tcie;     // habilitar interrupcao completa TX
    uint8_t c2_rie;      // habilitar interrupcao do receptor
    uint8_t c2_ilie;// habilitar interrupcao por linha ociosa
    uint8_t c3_orie;     // habilitar interrupcao por overrun
    uint8_t c3_neie;     // habilitar interrupcao por noise error
    uint8_t c3_feie;     // habilitar interrupcao por framing error
    uint8_t c3_peie;     // habilitar interrupcao por parity error
} UART0ConfigInterrupts_type;
```

Técnicas de Programação

- Modularização de códigos

  SIM_selPLLFLL (uint8_t base); // selecionar base de tempo

  SIM_initBusClock (uint8_t OUTDIV4); // inicializar fonte de clock

  PORTx_muxUART0RxTxPins(void); // configurar pinos para função Rx e Tx

  UART0_init (UART0Config *config); // inicializar e habilitar UART0

  UART0_reconfig(UART0Config_type *config); // reconfigurar os parâmetros de UART0

  UART0_configInterrupts (UART0ConfigInterrupt_type &configInterrupt); // configurar os parâmetros de interrupção

  UART0_enableTEInterrup (void) // habilitar a interrupção TIE

  UART0_disableTEInterrup (void) // desabilitar a interrupção TIE

Pseudocódigo

```
SIM_initBusClock (0b000);     // Divisor de frequencia em 1
SIM_selPLLFLL (0);     // selecionar MCGFLLCLK (20971520Hz)
PORTx_muxUART0RxTxPins();
NVIC_enableUART0(2);
config.c4_osr = 7;
config.sbr = (uint16_t)(20971520./(19200 * (config.c4_osr+1)));
UART0_init(&config);
configInterrupt.c2_rie = 1; configInterrupt.c2_ilie = 1;configInterrupt.c3_orie = 1;...
UART0_configInterrupts (&configInterrupt);
Laço :
    Piscar o ledRGB em cor amarela.
```

CodeWarrior IDE Development Suite

Informações Adicionais

- KL25 Sub-Family Reference Manual

  ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf

  - NVIC: Capítulo 3 (páginas 51, 77)
  - Clock Distribution: Capítulo 5 (página 123)
  - SIM: Capítulo 12 (páginas 200, 207)
  - UART0: Capítulo 39 (página 721)

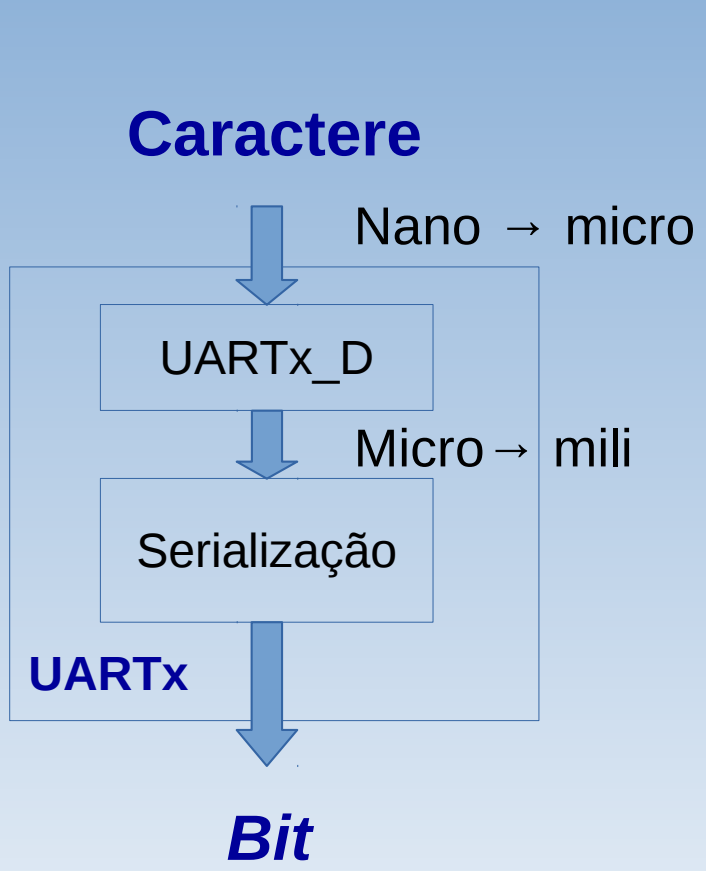
EA871

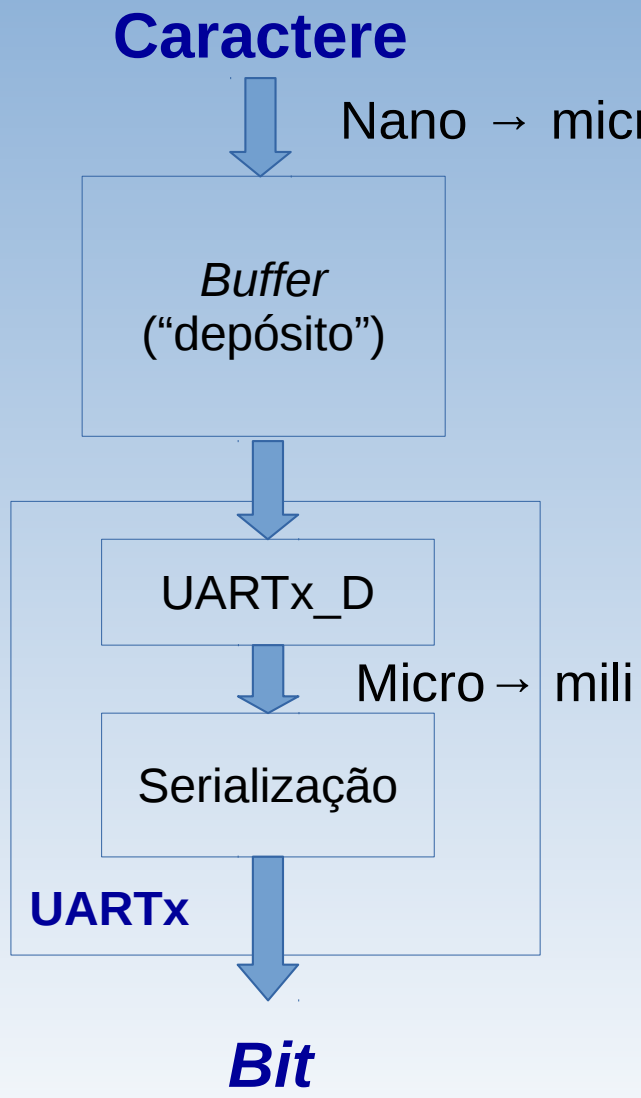
Comunicação Serial Assíncrona

Buffer Circular

Wu Shin – Ting
DCA – FEEC - Unicamp
Segundo Semestre de 2020

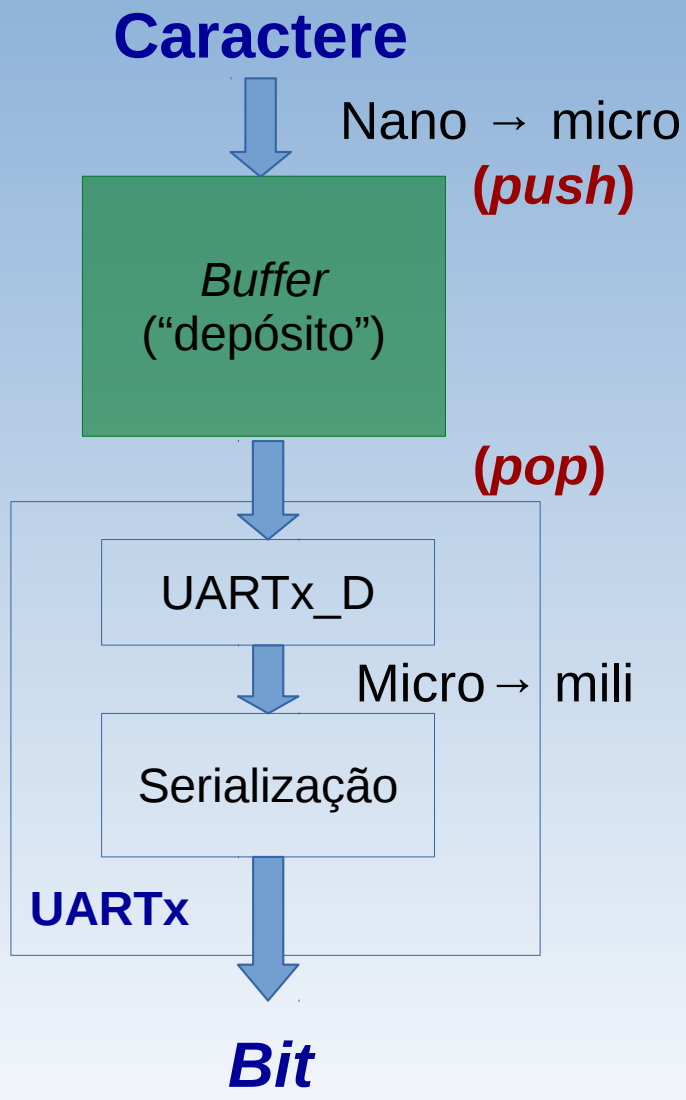
Processamento de uma rajada de caracteres





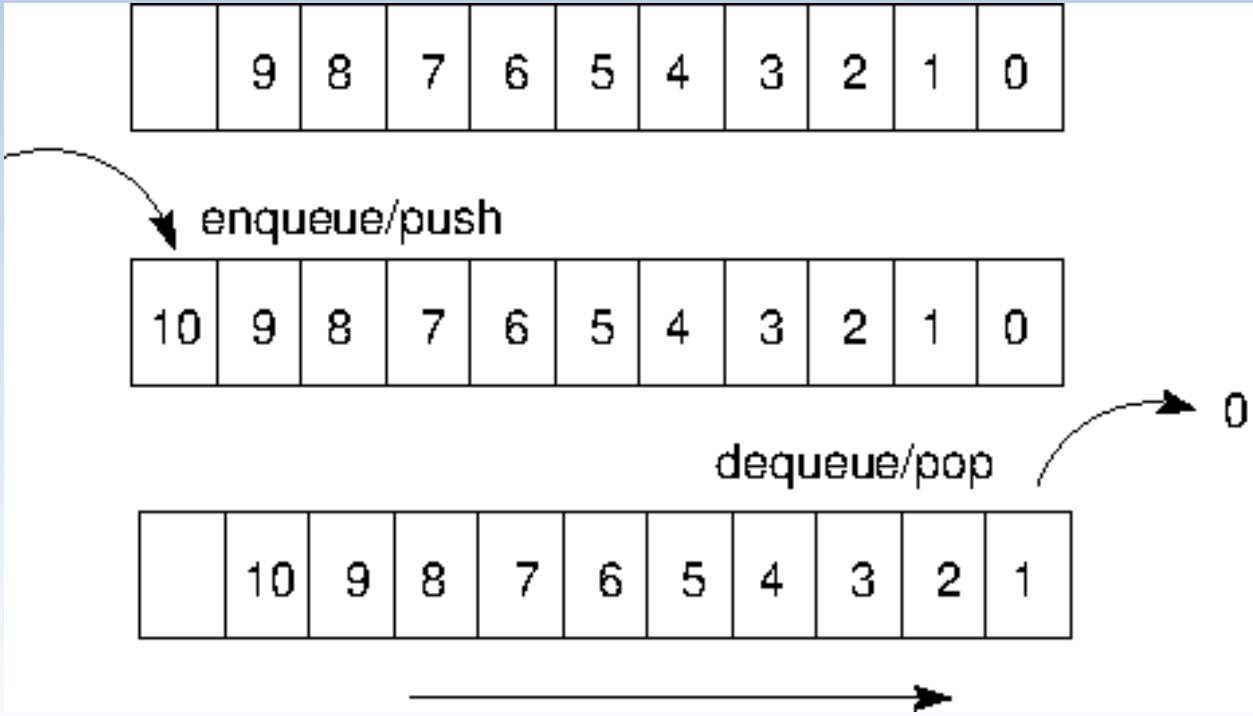
Conceitos

- **Acessos Concorrentes:** quando dois ou mais trechos de códigos de programa disputam o acesso a recursos compartilhados.
- **Regiões críticas:** são trechos de códigos de programa que fazem acessos aos recursos compartilhados.
  - **Execução atômica**: execução dos trechos sem interrupção.



Buffer de Dados

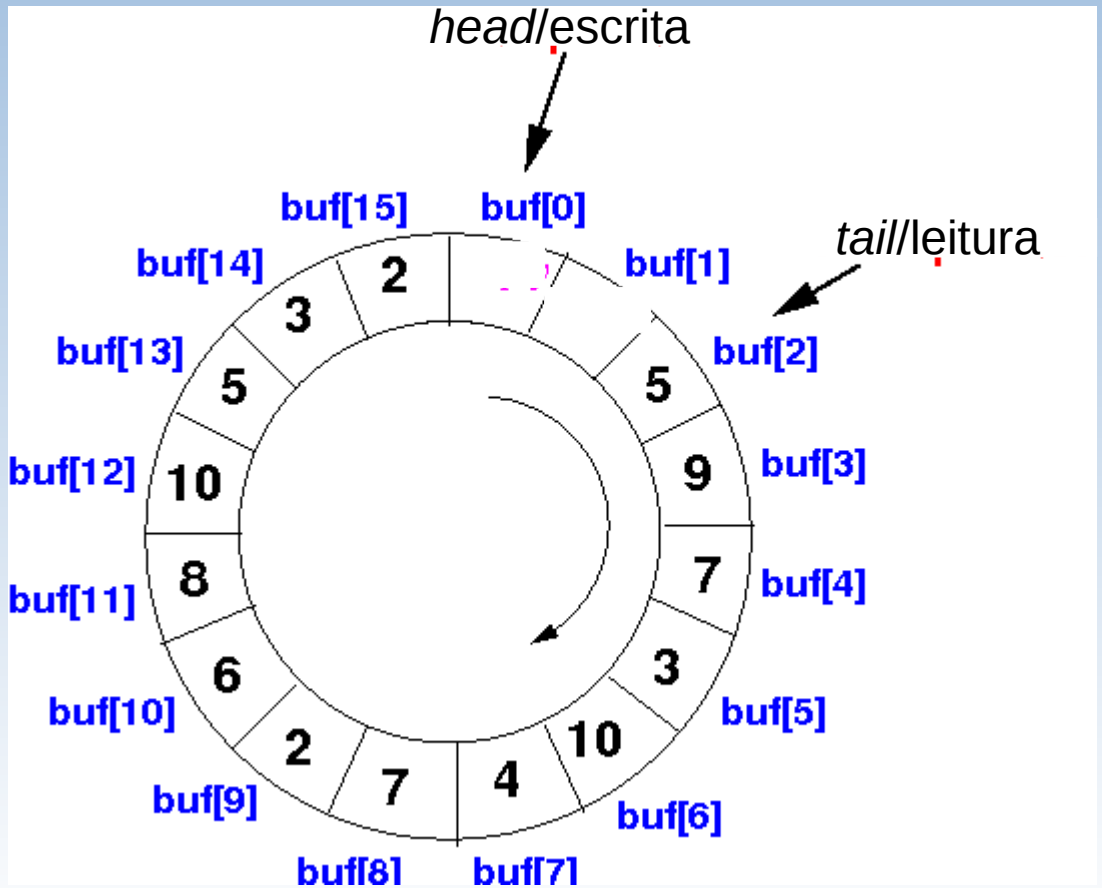
- **Fila**: uma estrutura linear de dados em que o primeiro elemento inserido (*First-In*) é o primeiro elemento a ser extraído (*First-Out*).
  - ***Enqueue/Push***: Insere um novo elemento no final da sequência.
  - ***Dequeue/Pop:*** Retire o primeiro elemento da sequência.



Manipulação dinâmica dos dados na memória

Buffer de Dados

- ***Buffer circular***: é uma fila circular, de tamanho fixo N, cujas pontas são conectadas.
  - Operações sobre dois ponteiros: *head*/escrita e *tail*/leitura.



Buffer Circular

- Detalhes de implementação
  - Diferenciação de um *buffer* cheio de um *buffer* vazio?
    - *Buffer* vazio: escrita (*head*) == leitura (*tail*).
    - *Buffer* cheio: escrita(head) + 1 == leitura (tail).
  - Transição de item (N-1) para item 0
    - escrita + 1 == N → escrita = 0;
    - leitura + 1 == N → leitura = 0;
  - O que fazer quando o *buffer* estiver cheio?
    - Descartar o novo item.
    - Sobreescrever os itens antigos.
  - Atualizar os itens no *buffer* antes de atualizar os ponteiros.

Uma implementação em C

- Declaração de um tipo de dado adequado

```
typedef struct BufferCirc_tag
{
char * const dados;        // buffer de dados
    unsigned int tamanho;    // quantidade total de elementos
    unsigned int leitura;        // indice de leitura (tail)
    unsigned int escrita;        // indice de escrita (head)
} BufferCirc_type;
```

Uma implementação em C

- Inicialização de um *buffer* circular: uso de alocação dinâmica para alocar o espaço de memória desejado.

```
void BC_init(BufferCirc_type *buffer, unsigned int tamanho)
{
    buffer->dados = malloc(tamanho * sizeof(char));
    buffer->tamanho = tamanho;
    buffer->escrita = 0;
    buffer->leitura = 0;
}
```

Uma implementação em C

- Inseerção de um novo item no *buffer* circular

```
int BC_push (BufferCirc_type *buffer, char item)
{
    unsigned int next;
    next = buffer->escrita+1;
    if (next == buffer->tamanho) next = 0;
    if (next == buffer->leitura) { //sem espaco no buffer
        return -1;
    }
     buffer->dados[buffer->escrita] = item;
    buffer->escrita = next;
     return 0;
}
```

Uma implementação em C

- Remoção de um item do *buffer* circular

```
int BC_pop (BufferCirc_type *buffer, char *item)
{
unsigned int next;
if (buffer->escrita == buffer->leitura) { //buffer vazio
      buffer->escrita = buffer->leitura = 0; //reset
      return -1;
}
next = buffer->leitura + 1;
if (next == buffer->tamanho) next = 0;
 *item = buffer->dados[buffer->leitura];
 buffer->leitura = next;
 return 0;
}
```

Projeto-Exemplo

- Escrever no Terminal repetitivamente os caracteres ASCII de 0x30 a 0xA5 **por linhas**, pelo mecanismo de interrupção.
  - Os caracteres são escritos num *buffer* circular de 128 elementos enquanto este não estiver cheio.
  - Os caracteres são transferidos do *buffer* circular para o canal Tx assim que este estiver disponível.
  - Configuração do protocolo de comunicação: *baud rate (9600)*, quantidade de *bits* de dados (8 *bits*), paridade ímpar, 1 *stop bit,* LSB primeiro, fluxo de controle NENHUM (não é suportado por Kinetis).
  - Superamostragem (amostras por *bit*): 13.

Técnica de Programação

- Espaço de memória para os elementos de um *buffer* é dependente da aplicação.
  - Uso de alocação dinâmica de memória para alocar o espaço para um *buffer* de dados.

    #include <stdlib.h>

    malloc (<quantidade de *bytes*>)
    - Não zera o espaço alocado

    calloc (<quantidade de *bytes*>)
    - Zera o espaço alocado

    free (<endereço do espaço alocado>)

Técnica de Programação

- Separação por linhas as sequências de caracteres 0x30 a 0xA5
  - Uso de caracteres de controle para o Terminal
    - ‘\r’ (carriage return): retorno de carro (da máquina de escrever)
    - ‘\n’ (newline): uma nova linha
  - Entre as duas sequências, adiciona-se os dois caracteres de controle:

    … 0xA2 0xA3 0xA4 0xA5 \r \n 0x30 0x31 0x32 0x33 …
- No canal receptor: tecla “Return” é representada pela sequência de caracteres de controle “\n\r”.
- No canal transmissor: sequência “\n\r” é visualizado como mudança de linha no aplicativo Terminal.

Pseudocódigo

- Fluco de controle principal

  SIM_initBusClock (0b000); // Divisor de frequencia em 1

  SIM_selPLLFLL (0); // selecionar MCGFLLCLK (20971520Hz)

  PORTx_muxUART0RxTxPins();

  NVIC_enableUART0(2);

  config.c4_osr = 7;

  config.sbr = (uint16_t)(20971520./(19200 * (config.c4_osr+1)));

  UART0_init(&config);

  BC_push(&buffer, 0x30);

  configInterrupt.c2_tie = 1;

  UART0_configInterrupts (&configInterrupt);

  Laço :

    Inserir no buffer circular a sequência de caracteres, uma por um.

Pseudocódigo

- Rotina de Serviço

```
if (UART0_S1 & UART0_S1_TDRE_MASK) {
    BC_pop (&buffer, &item);
    UART0_D = item;
}
```

CodeWarrior IDE Development Suite

Informações Adicionais

- Implementing circular buffer in C

  https://embedjournal.com/implementing-circular-buffer-embedded-c/