Comunicação serial (USART)

Receptor/transmissor universal assíncrono, é um circuito físico inserido em um microcontrolador que permite troca de dados entre dispositivos em sistemas embarcados. Suporta transmissão bidirecional e é considerada assíncrona, devido a falta de um canal de clock. A ausência do canal de clock é compensada pelo BaudRate, o que permite o sistema saber quando esses bit foram clocados. Ainda, a transmissão UART adiciona um start e um stop bit no dado transmitido.

UART não utiliza o paradigma mestre/escravo, mas sim transmissor/receptor. Dentro do microcontrolador, os dados estão em formato paralelo, o conversor UART converte esses dados para serial, transmite, e o receptor faz o processo inverso.

EUSART - ENHANCED UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER

- Full-duplex (modo assíncrono): é possível transmitir e receber dados ao mesmo tempo;
- Half-duplex (modo síncrono): não suporta transmissão e recepção de dados ao mesmo tempo, ou você fala ou você escuta
- 8 ou 9 bits para comunicação: o nono bit está disponível em alguns dispositivos e serve para enviar dados sobre o endereçamento;
- 1 buffer de transmissão;
- 2 buffer de recepção;
- Calibração automática de baudRate. Através da própria informação que chega o microcontrolador ajusta automaticamente seu BaudRate
- Pode ativar a interrupção na recepção ou transmissão de dados;
- UART simplificada contendo apenas pinos RX e TX

O PIC18F45K22 possui dois canais de comunicação TX e RX, sendo um em RB6 e RB7 e outro em RC6 e RC7

Rotina de configuração

- Configura o PORT relacionado a comunicação como circuito digital
- Habilita a chave geral pelo bit SPEN
- Habilita a transmissão (se necessário) pelo bit TXEN
- Habilita a recepção (se necessário) pelo bit CREN
- Configura o modo de operação pelos registradores TXREG e RCREG
- Configura BaudRate através dos registradores BAUDCON e SPBRG e SPBRGH
- Configura interrupções (se necessário)

Registradores:

- TXREG1 → Registrador de envio. É o registrador a ser escrito, seu valor será enviado pela serial
- RCREG1 → Registrador de recebimento. É o registrador a ser lido. É preenchido automaticamente durante a recepção de um dado
- TXSTA1 → Registrador de configurações do transmissor
- RCSTA1 → Registrador de configurações do receptor
- BAUDCON1 → Registrador de controle do BaudRate
- SPBRG e SPBRGH → Registradores que definem a velocidade do BaudRate

TXSTA1

REGISTER 16-1: TxSTAx: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **CSRC:** Clock Source Select bit
Asynchronous mode:
Don't care
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9:** 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN:** Transmit Enable bit(1)
1 = Transmit enabled
0 = Transmit disabled

bit 4 **SYNC:** EUSART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **SENDB:** Send Break Character bit
Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed
Synchronous mode:
Don't care

bit 2 **BRGH:** High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode

bit 1 **TRMT:** Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** Ninth bit of Transmit Data
Can be address/data bit or a parity bit.

- CSRC → Somente modo síncrono. Define se este dispositivo vai gerar ou vai receber o clock
- TX9 → Transmissão em 9 bits
- TXEN → Habilita transmissão serial
- SYNC → Define Modo síncrono ou assíncrono
- SENDB → Este bit habilita a comunicação pelo protocolo LIN
- BRGH → Define modo High Speed ou Low Speed da comunicação

- TRMT → Indica se o buffer de transmissão está vazio ou cheio
- TX9D → Estado do nono bit

RCSTA1

REGISTER 16-2: RCSTAx: RECEIVE STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RXx/DTx and TXx/CKx pins as serial port pins)
0 = Serial port disabled (held in Reset)

bit 6 **RX9:** 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit
Asynchronous mode:
Don't care
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave
Don't care

bit 4 **CREN:** Continuous Receive Enable bit
Asynchronous mode:
1 = Enables receiver
0 = Disables receiver
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enable interrupt and load the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit
Asynchronous mode 8-bit (RX9 = 0):
Don't care

bit 2 **FERR:** Framing Error bit
1 = Framing error (can be updated by reading RCREGx register and receive next valid byte)
0 = No framing error

bit 1 **OERR:** Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** Ninth bit of Received Data
This can be address/data bit or a parity bit and must be calculated by user firmware.

- SPEN → Chave geral que habilita a comunicação serial
- RX9 → Habilita o modo 9 bits na recepção

- SREN → Habilita a recepção, válido apenas para o modo síncrono
- CREN (asynch) → habilita recepção modo assíncrono
- CREN (sync) → Se estiver no modo síncrono, este sinaliza quando acontece um erro
- ADDEN → Se RX9 estiver ligado, ativa detecção de endereço
- FERR → Erro de framing. Quando o start bit ou o stop bit não chegam como o esperado, este flag é setado.
- OEFF → Erro de overflow. Se chegar mais um byte no buffer RCREG1 sem ele ter sido lido antes, ocorre um erro e este flag é setado. A comunicação para neste momento.
- RXD9 → Estado do bit 9 (se habilitado). Checa se a informação é de endereçamento ou de dado, chama a interrupção

BAUDCON1

REGISTER 16-3: BAUDCONx: BAUD RATE CONTROL REGISTER

R/W-0	R-1	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	DTRXP	CKTXP	BRG16	—	WUE	ABDEN
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **ABDOVF:** Auto-Baud Detect Overflow bit
Asynchronous mode:
1 = Auto-baud timer overflowed
0 = Auto-baud timer did not overflow
Synchronous mode:
Don't care

bit 6 **RCIDL**: Receive Idle Flag bit
Asynchronous mode:
1 = Receiver is Idle
0 = Start bit has been detected and the receiver is active
Synchronous mode:
Don't care

bit 5 **DTRXP**: Data/Receive Polarity Select bit
Asynchronous mode:
1 = Receive data (RXx) is inverted (active-low)
0 = Receive data (RXx) is not inverted (active-high)
Synchronous mode:
1 = Data (DTx) is inverted (active-low)
0 = Data (DTx) is not inverted (active-high)

bit 4 **CKTXP**: Clock/Transmit Polarity Select bit
Asynchronous mode:
1 = Idle state for transmit (TXx) is low
0 = Idle state for transmit (TXx) is high
Synchronous mode:
1 = Data changes on the falling edge of the clock and is sampled on the rising edge of the clock
0 = Data changes on the rising edge of the clock and is sampled on the falling edge of the clock

bit 3 **BRG16:** 16-bit Baud Rate Generator bit
1 = 16-bit Baud Rate Generator is used (SPBRGHx:SPBRGx)
0 = 8-bit Baud Rate Generator is used (SPBRGx)

bit 2 **Unimplemented:** Read as '0'

bit 1 **WUE:** Wake-up Enable bit
Asynchronous mode:
1 = Receiver is waiting for a falling edge. No character will be received but RCxIF will be set on the falling edge. WUE will automatically clear on the rising edge.
0 = Receiver is operating normally
Synchronous mode:
Don't care

bit 0 **ABDEN**: Auto-Baud Detect Enable bit
Asynchronous mode:
1 = Auto-Baud Detect mode is enabled (clears when auto-baud is complete)
0 = Auto-Baud Detect mode is disabled
Synchronous mode:
Don't care

- ABDOVF → Auto baudrate status

- RCIDL → Modo Idle
- SCKP → Polaridade do clock
- BRG16 → BaudRate 16 bits
- WUE → Sai do modo sleep quando chega algum dado pela serial
- ABDEN → Habilita o auto BaudRate. Quando você tem uma fonte de clock que você não conhece a frequência da mesma

Bits de interrupção

PIE1bits.RC1IE = Habilita a interrupção por **recepção** serial

PIR1bits.RC1IF = Flag da interrupção. A interrupção acontece no momento que o buffer de recepção estiver cheio.

IPR1bits.RC1IP = Prioridade

PIE1bits.TX1IE = Habilita a interrupção por **transmissão** serial

PIR1bits.TX1IF = vale 1 se o buffer de escrita estiver vazio. Quando escrito, vale 0

IPR1bits.TX1IP = prioridade da transmissão (para interrupções)

Trabalhando com BaudRate

O BaudRate é uma medida de velocidade de transmissão de dados em comunicação serial. Ele representa a quantidade de símbolos (ou bits) de dados que podem ser transmitidos por segundo em uma interface serial de comunicação. O BaudRate é geralmente expresso em unidades de "bauds", que representam a quantidade de símbolos de dados transmitidos por segundo.

Por exemplo, se o BaudRate de uma interface serial é de 9600 bauds, isso significa que a interface é capaz de transmitir 9600 símbolos de dados por segundo. É importante ressaltar que o BaudRate precisa ser configurado corretamente em ambas as extremidades da comunicação para que a transmissão de dados ocorra de forma correta e sem erros.

A tabela abaixo mostra as equações utilizadas para encontrar o valor de SPBRG e SPBRGH a partir de um BaudRate (BR) predefinido:

Sync	BRG16	BRGH = 0	BRGH = 1

0	0	$(Fosc/(64 * BR)) - 1$	$(Fosc/(16 * BR)) - 1$
0	1	$(Fosc/(16 * BR)) - 1$	$(Fosc/(4 * BR)) - 1$
1	0	$(Fosc/(4 * BR)) - 1$	N/A
1	1	$(Fosc/(4 * BR)) - 1$	N/A

Já com a tabela abaixo, encontra-se o BaudRate a partir de um valor de SPBRG (n) predefinido:

TABLE 16-3: BAUD RATE FORMULAS

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	FOSC/[64 (n+1)]
0	0	1	8-bit/Asynchronous	FOSC/[16 (n+1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	FOSC/[4 (n+1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGHx, SPBRGx register pair.

Taxa de erro do BaudRate

$$Erro = (baudEncontrado - baudDesejado)/baudDesejado$$

BaudRates recomendados

O datasheet do PIC18F45k22 apresenta uma tabela completa com as configurações recomendadas dos registradores para cada BaudRate. Nela, você verá que algumas linhas estão riscadas, pois se tratam de valores que não estão disponíveis para aquela configuração. Esta tabela está a partir da página 281 do datasheet.

TABLE 16-5: BAUD RATES FOR ASYNCHRONOUS MODES

BAUD RATE	SYNC = 0, BRGH = 0, BRG16 = 0											
	Fosc = 64.000 MHz			Fosc = 18.432 MHz			Fosc = 16.000 MHz			Fosc = 11.0592 MHz		
	Actual Rate	% Error	SPBRxG value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)
300	—	—	—	—	—	—	—	—	—	—	—	—
1200	—	—	—	1200	0.00	239	1202	0.16	207	1200	0.00	143
2400	—	—	—	2400	0.00	119	2404	0.16	103	2400	0.00	71
9600	9615	0.16	103	9600	0.00	29	9615	0.16	25	9600	0.00	17
10417	10417	0.00	95	10286	-1.26	27	10417	0.00	23	10165	-2.42	16
19.2k	19.23k	0.16	51	19.20k	0.00	14	19.23k	0.16	12	19.20k	0.00	8
57.6k	58.82k	2.12	16	57.60k	0.00	7	—	—	—	57.60k	0.00	2
115.2k	111.11k	-3.55	8	—	—	—	—	—	—	—	—	—

BAUD RATE	SYNC = 0, BRGH = 0, BRG16 = 0											
	Fosc = 8.000 MHz			Fosc = 4.000 MHz			Fosc = 3.6864 MHz			Fosc = 1.000 MHz		
	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)	Actual Rate	% Error	SPBRGx value (decimal)
300	—	—	—	300	0.16	207	300	0.00	191	300	0.16	51
1200	1202	0.16	103	1202	0.16	51	1200	0.00	47	1202	0.16	12
2400	2404	0.16	51	2404	0.16	25	2400	0.00	23	—	—	—
9600	9615	0.16	12	—	—	—	9600	0.00	5	—	—	—
10417	10417	0.00	11	10417	0.00	5	—	—	—	—	—	—
19.2k	—	—	—	—	—	—	19.20k	0.00	2	—	—	—
57.6k	—	—	—	—	—	—	57.60k	0.00	0	—	—	—
115.2k	—	—	—	—	—	—	—	—	—	—	—	—

Construindo protocolos

Um protocolo de comunicação é uma série de regras e procedimentos que permitem que dois dispositivos se comuniquem entre si. Esses protocolos especificam como os dados são transferidos de um dispositivo para outro, como os dispositivos estabelecem a conexão e como as informações são interpretadas. Até agora, com o modo USART, vimos somente como os dados são transferidos de um dispositivo para o outro. Agora, vamos trabalhar na interpretação destes dados criando um pacote padronizado que ditará funções dentro do microcontrolador.

Um protocolo de comunicação geralmente possui uma estrutura padronizada, geralmente chamada de grame, que garante a transmissão correta e confiável dos dados entre dispositivos. Essa estrutura pode variar dependendo do protocolo em questão, mas geralmente inclui os seguintes elementos:

1. Sincronização (Sync): é um sinal utilizado para sincronizar a comunicação entre dispositivos. Ele indica o início e o fim de cada mensagem.

2. Endereço (Address): é um valor utilizado para identificar o destinatário da mensagem. Se houver mais de um dispositivo conectado a uma rede, cada um deve ter um endereço único para que as mensagens sejam direcionadas corretamente.
3. Controle (Control): são bits utilizados para controlar a transmissão e recepção de dados. Eles podem indicar, por exemplo, se a mensagem é uma requisição ou uma resposta.
4. Dados (Data): é a informação que está sendo transmitida entre os dispositivos.
5. Verificação de erros (Error checking): é um mecanismo utilizado para detectar erros de transmissão de dados. Geralmente, isso envolve a adição de bits extras aos dados transmitidos que podem ser utilizados para verificar se houve erros durante a transmissão.

No entanto, nem todos os protocolos de comunicação possuem todos esses elementos. Alguns protocolos podem ter apenas alguns deles, enquanto outros podem ter elementos adicionais. O importante é que cada protocolo de comunicação defina uma estrutura clara e consistente para garantir a comunicação confiável entre dispositivos.

Para nosso protocolo, utilizaremos um byte de **sincronização** para indicar o início e o fim da comunicação, esse byte será o “{”. Depois virá o **endereço**, que determinará onde o comando será executado, isso vai ser definido pelo conjunto de caracteres “PIN” e depois pelo **subendereço**, que serão valores de 0 a 7. Por fim, teremos 3 tipos de **data** = S (estado), ON (ligar), OFF(desligar). Não trabalharemos com verificação de erros neste momento. Por fim, finalia

- {PINxS} → Retorna o estado do LED
- {PINxC1} → Liga o LED
- {PINxC0} → Desliga o LED

O “x” sinaliza qual led será ligado.

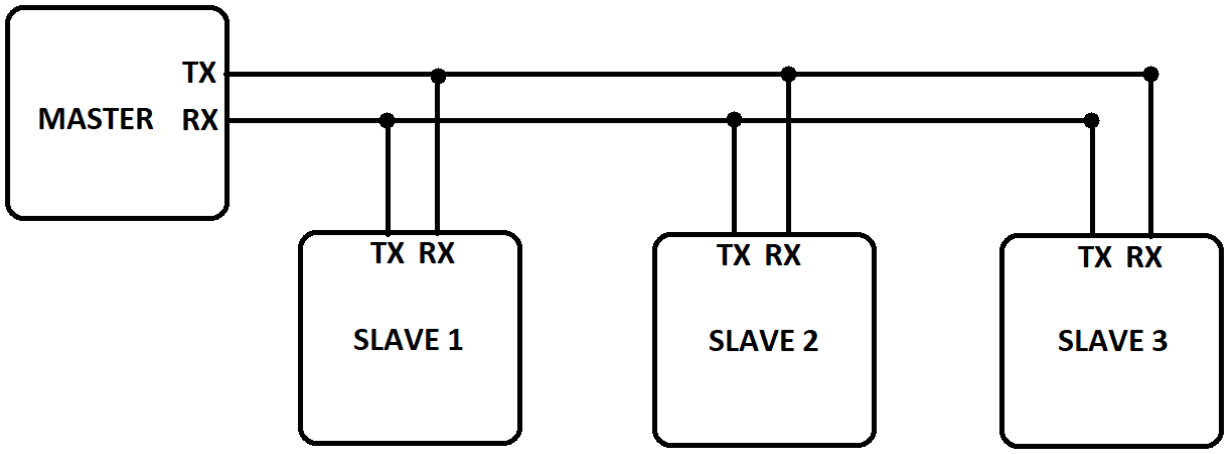
Definir um pacote de dados garante que a comunicação ocorra mesmo que ocorra interferências durante o processo. É claro que se a comunicação for irreconhecível, não haverá nenhuma ação.

Modo 9 bits

9 bits é uma configuração especial de UART que permite trabalhar com endereçamento em uma rede de comunicação, quando há mais de dois microcontroladores se comunicando. Geralmente, o primeiro byte da comunicação é para endereçamento. Este endereçamento é atribuído via software.

Durante a comunicação, os bytes são enviados para todos os microcontroladores ao mesmo tempo. Para evitar que todos os microcontroladores tenham interrupções ativadas para verificar estes bytes (mesmo ele não pertencendo à eles), o endereçamento diz a um microcontrolador que somente ele deverá analisar os bytes que estão chegando.

Sempre quando o nono bit da comunicação estiver setado, os microcontroladores que recebem este dado saberão que se trata de um endereço. O microcontrolador com o respectivo endereço saberá que a comunicação está sendo realizada com ele e assim ele habilitará as interrupções. Isso preserva a rotina de processamento de outros dispositivos da rede.



Processo de funcionamento

- Para transmitir um bloco de dados, o MASTER deverá primeiramente enviar um byte inicialmente com endereço do SLAVE.
- Esse byte deverá ser enviado com o 9-bit em 1. Para isso deve ser configurado:

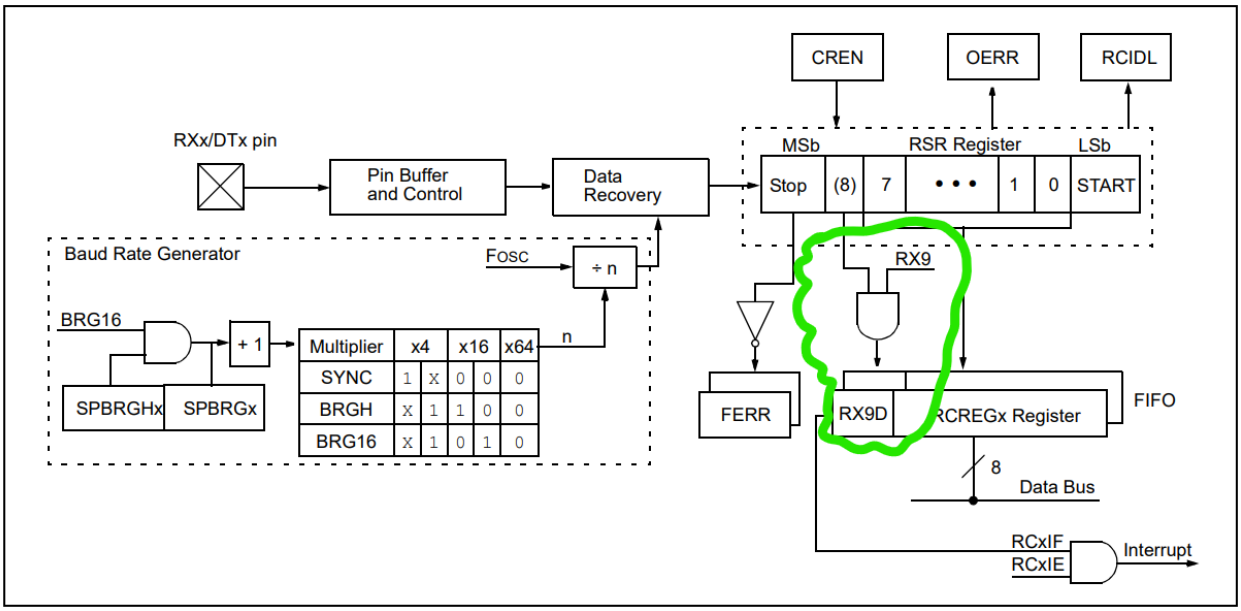
```
TX9 = 1;  // Transmissão do nono bit habilitada
TX9D = 1; // Nível do nono bit
```

- Os SLAVES deverão estar com suas interrupções na recepção serial habilitadas e configuradas como:

```
ADDEN = 1; // Endereçamento habilitado
RX9 = 1;   // Recepção do nono bit habilitada
RX9D = 0;  // Valor do nono bit recebido
```

- A interrupção do slave só será gerada se o transmissor enviar este novo bit em nível lógico 1. Quando isso ocorrer, o bit RX9D será setado.

FIGURE 16-2: EUSART RECEIVE BLOCK DIAGRAM



- A primeira coisa que deve ser feita na recepção é ler RX9D,
- Todos os SLAVES receberão esse endereço, porém somente o SLAVE que possui o endereço enviado, deverá resetar o bit ADDEN = 0;
- O MASTER agora está pronto para enviar o pacote de dados (o byte de endereço ele já enviou primeiro), porém, antes de enviar deverá resetar TX9D = 0, ou seja, deverá enviar o 9-bit em nível zero, pois assim somente o SLAVE que tiver ADDEN = 0 é que vai receber esses dados.
- Após o término da comunicação, este SLAVE deverá setar novamente ADDEN = 1 e o MASTER deverá configurar TX9D = 1 para restabelecer a comunicação com o

9-bit em nível 1.

O envio do nono bit com o byte de endereçamento vai gerar interrupção em todos os microcontroladores