MODBUS

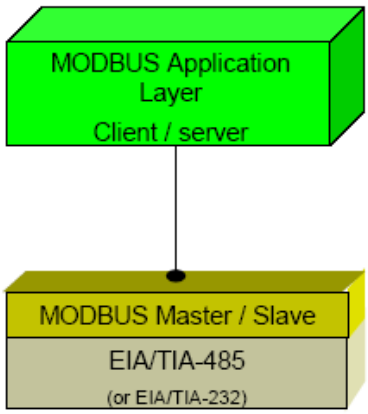
O padrão MODBUS define um Protocolo de mensagens na camada de aplicação, posicionado no 7º nível do Modelo de Referência OSI que provê comunicação "cliente/servidor" entre dispositivos conectados a diferentes tipos de barramentos ou topologias de rede. Este padrão também especifica um protocolo de comunicação serial para requisições MODBUS entre um mestre e um ou vários escravos.

1. Protocolo Modbus em Linha Serial

O protocolo Serial Mobus é um protocolo mestre-escravo. Este protocolo pertence a 2ª camada do Modelo de Referência OSI. Um sistema operando como mestre-escravo possui um nó mestre, que emite comandos explícitos para um dos nós escravos e processa a sua resposta. Tipicamente os escravos não irão transmitir dados sem uma requisição do nó mestre e não se comunicam com outros escravos.

Na camada física os sistemas Modbus em linhas seriais podem usar diferentes interfaces físicas (RS485, RS232, etc.). A interface RS485 de 2 fios é a mais comum. No entanto, a interface RS485 de 4 fios também pode ser implementada. A interface serial RS232 só poderá ser utilizada quando uma comunicação ponto a ponto de curta distância for requerida. A figura abaixo mostra uma representação geral dos protocolos Modbus frente as 7 camadas do modelo de referência OSI.

Layer	ISO/OSI Model	
7	Application	MODBUS Application Protocol
6	Presentation	Empty
5	Session	Empty
4	Transport	Empty
3	Network	Empty
2	Data Link	MODBUS Serial Line Protocol
1	Physical	EIA/TIA-485 (or EIA/TIA-232)



Protocolo Modbus e o Modelo de Referência OSI

O protocolo de mensagens Modbus na camada de aplicação provê comunicação "cliente/servidor" entre dispositivos conectados em diferentes barramentos e topologias de rede. No protocolo Modbus operando em linha serial só existe um cliente, que é o nó mestre da linha serial. Os nós escravos operam como servidores.

Camada de Enlace de Dados Modbus

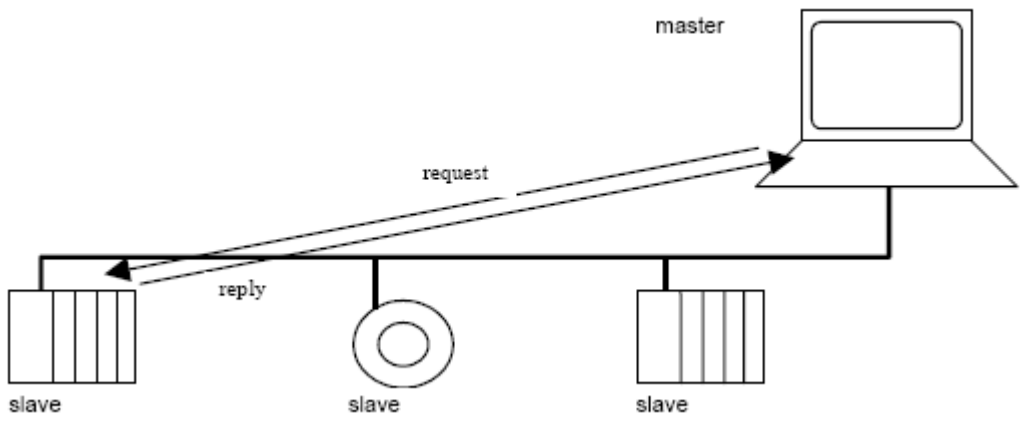
1.1.1 Princípio do Protocolo Modbus Mestre-Escravo

O protocolo Modbus operando em linha serial é um protocolo mestre-escravos. Isso significa que somente um mestre é conectado ao barramento ao mesmo tempo. Quanto aos escravos, um ou mais nós (número máximo de 247) podem ser conectados a este mesmo barramento.

Uma comunicação Modbus é sempre iniciada pelo mestre. O nó escravo nunca irá transmitir dados sem receber uma requisição do nó mestre. Os nós escravos nunca irão se comunicar entre eles. O nó mestre inicia somente uma transação Modbus por vez.

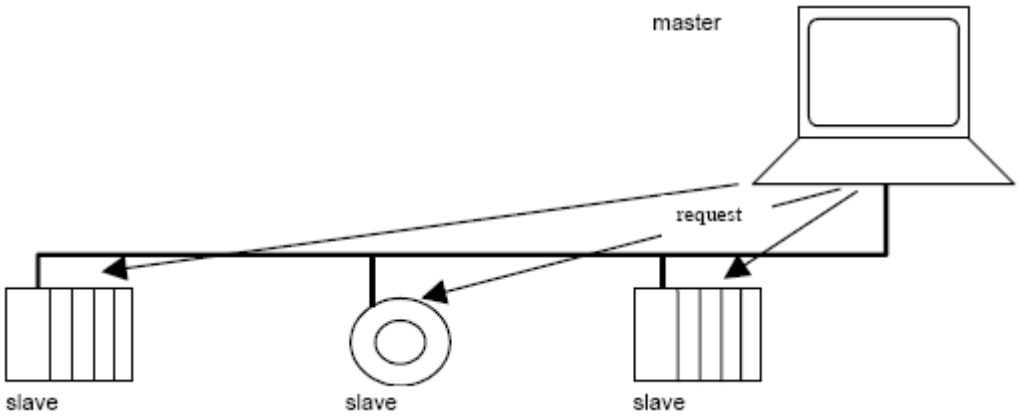
O nó mestre emite uma requisição para um nó escravo em dois modos:

- *Unicast* – o mestre endereça a somente um escravo. Depois de receber e processar a requisição, o escravo retorna uma mensagem de resposta para o mestre. Neste modo, uma transação Modbus consiste de 2 mensagens: uma requisição do mestre e uma resposta do escravo. Cada escravo deve ter um endereço único (de 1 a 247) de forma a poder ser endereçado independentemente de outros nós.



Modo *Unicast*

- *Broadcast* – o nó mestre pode enviar uma mensagem para todos os escravos. Nenhuma resposta deve ser retornada para requisições *broadcast* enviadas pelo mestre. As requisições *broadcast* são necessariamente mensagens de escrita. Todos os dispositivos devem aceitar mensagens broadcast para escrita. O endereço 0 é reservado para identificar uma mensagem *broadcast*.



Modo *Broadcast*

1.1.2 Regras de Endereçamento Modbus

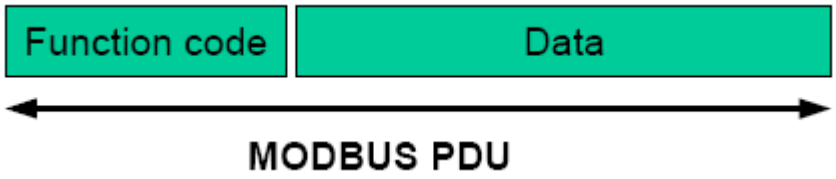
O espaço de endereçamento Modbus compreende 256 diferentes endereços.

0	De 1 a 247	De 248 a 255
Endereço de *Broadcast*	**Endereço individual dos escravos**	**Reservado**

O endereço 0 é reservado para *broadcast*. Todos os escravos devem reconhecer o endereço de *broadcast*. O mestre Modbus não tem endereço específico, somente os nós escravos devem ter um endereço. Os endereços devem ser únicos em um barramento Modbus serial.

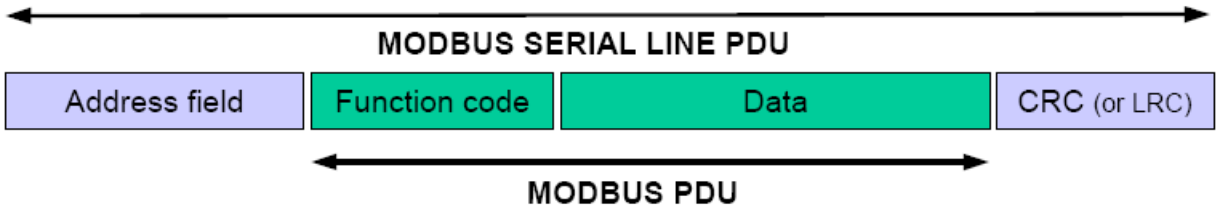
1.1.3 Descrição do Quadro Modbus

O protocolo de aplicação Modbus define uma simples unidade de dados de protocolo (*Protocol Data Unit* – PDU) independentemente das camadas adjacentes.



Unidade de Dados do Protocolo Modbus

As características do protocolo Modbus em um barramento ou topologia de rede específica introduzem alguns campos adicionais ao PDU. O cliente que inicializa uma transação Modbus constrói um Modbus PDU e, então adiciona campos para que construa o PDU apropriado para uma dada comunicação.



Quadro para Modbus em Barramento Serial

Em uma comunicação Modbus sobre barramento serial, o campo endereço contém o endereço de um determinado escravo. Como descrito anteriormente, os endereços válidos para nós escravos estão na faixa de 0 a 247 decimal. Quando utilizado o endereço 0, a mensagem é um *broadcast*, portando todos os nós escravos estarão sendo endereçados. Os escravos são individualmente endereçados na faixa de 1 a 247. Um mestre endereça um escravo colocando o endereço do escravo no campo endereço da mensagem. Quando o escravo retorna sua resposta, ele coloca o seu próprio endereço no campo endereço da mensagem de resposta para permitir que o mestre identifique qual escravo está respondendo.

O código de função indica para o servidor que tipo de ação deve se executada. O código de função pode ser seguido por um campo de dados que contém parâmetros da requisição e da resposta.

O campo de verificação de erro é resultado de um cálculo de verificação de redundância que é adicionado ao conteúdo da mensagem. Dois métodos para cálculo são usados, dependendo do modo de transmissão que está sendo utilizado (RTU ou ASCII).

1.1.4 Diagramas de Estado Mestre / Escravo

A camada de enlace de dados Modbus compreende duas sub-camadas:

- O protocolo Mestre / Escravo;
- O modo de transmissão RTU ou ASCII.

Abaixo temos a descrição dos diagramas de estado de um mestre e um escravo, que são independentes do modo de transmissão utilizado. A recepção e o envio de um quadro Modbus são descritos abaixo.

1.1.4.1 Diagrama de Estado de um Mestre Modbus

O diagrama abaixo descreve o comportamento de um mestre Modbus.

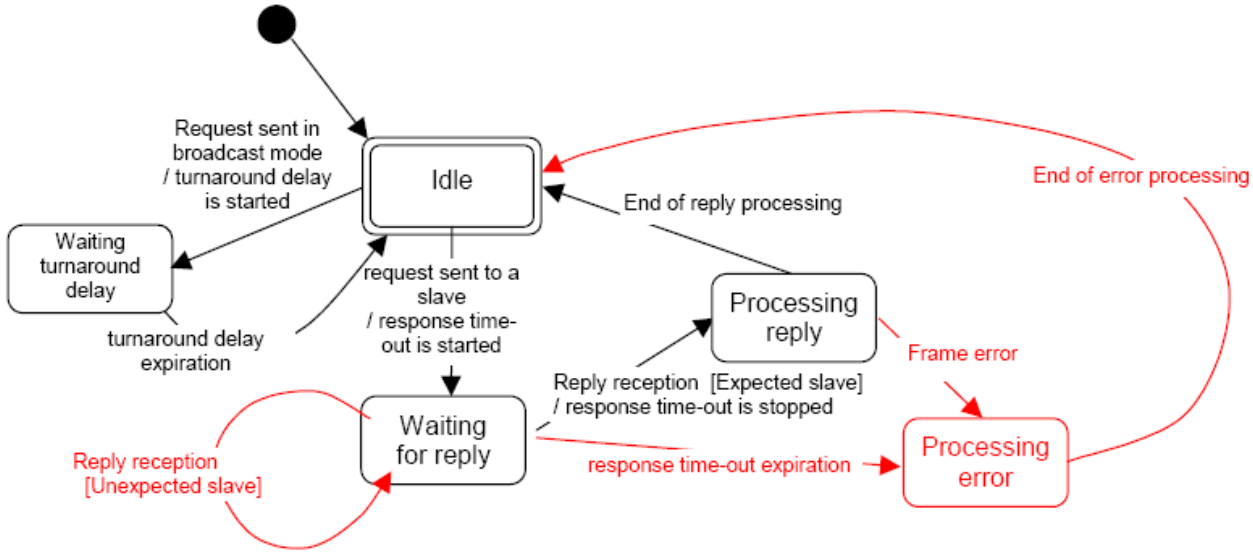


Diagrama de Estados do Mestre

Abaixo temos algumas considerações a respeito do diagrama de estados apresentado:

- Estado “*idle*” (ocioso) - Nenhuma requisição pendente. Este é o estado inicial após a alimentação de um nó mestre. Uma requisição somente pode ser enviada a partir de um estado “*idle*”. Depois de enviar uma requisição, o mestre sai do modo “*idle*” e não pode enviar uma segunda requisição até voltar a este modo;

- Quando uma requisição é enviada para um único escravo, o mestre entra em um estado de espera por resposta (*Waiting for Reply*) e um contador de espera é iniciado. Este contador previne o mestre de ficar indefinidamente no modo de espera de resposta. O valor do tempo de espera por uma resposta é dependente da aplicação;
- Quando uma resposta é recebida, o mestre verifica o resposta antes de iniciar o processamento dos dados. A verificação pode resultar em um erro. Exemplos de possíveis erros são a resposta por um escravo não esperado ou um erro no quadro recebido. No caso de uma resposta recebida por um escravo não esperado, o contador de espera continuara sendo executado. No caso de um erro ser detectado no quadro recebido, uma nova tentativa deve ser executada;
- Se nenhuma resposta é recebida o contador expira e um erro é gerado. Então o mestre entra no modo "*idle*", habilitando uma requisição para nova tentativa. O número máximo de tentativas é definido nas configurações do mestre;
- Quando uma mensagem *broadcast* é enviada no barramento serial, nenhuma resposta é retornada pelos escravos. Contudo, um tempo de espera é respeitado pelo mestre no sentido de permitir que qualquer escravo processe a requisição antes que o mestre envie uma nova mensagem. Portanto, o mestre entra em um estado de espera (*Waiting Turnaround Delay*) antes de voltar para o estado "*idle*" e, portanto, antes de ser capaz de enviar outra requisição;
- Em modo *unicast* o tempo de espera deve ser escolhido de forma que qualquer escravo possa processar a requisição e retornar uma resposta. No modo *broadcast* o tempo de espera deve ser longo o suficiente para que qualquer escravo processe somente a requisição e esteja apto a receber uma nova requisição. Devido a isso, o tempo de espera em modo *broadcast* deve ser menor que em modo *unicast*. Tipicamente o tempo de espera em modo *unicast* é de 1 segundo até vários segundos para uma comunicação em 9600 *bauds* e o tempo de espera em modo *broadcast* entre 100ms e 200ms.
- A verificação de erro no quadro consiste de: 1) Verificação de paridade aplicada a cada caracter; 2) Verificação de redundância aplicada a todo o quadro.

O diagrama de estados é intencionalmente muito simples. Este não leva em consideração o acesso ao meio, enquadramento de mensagem ou retransmissão em caso de erros.

1.1.4.2 Diagrama de Estado de um Escravo Modbus

Abaixo temos o diagrama de estados de um escravo Modbus:

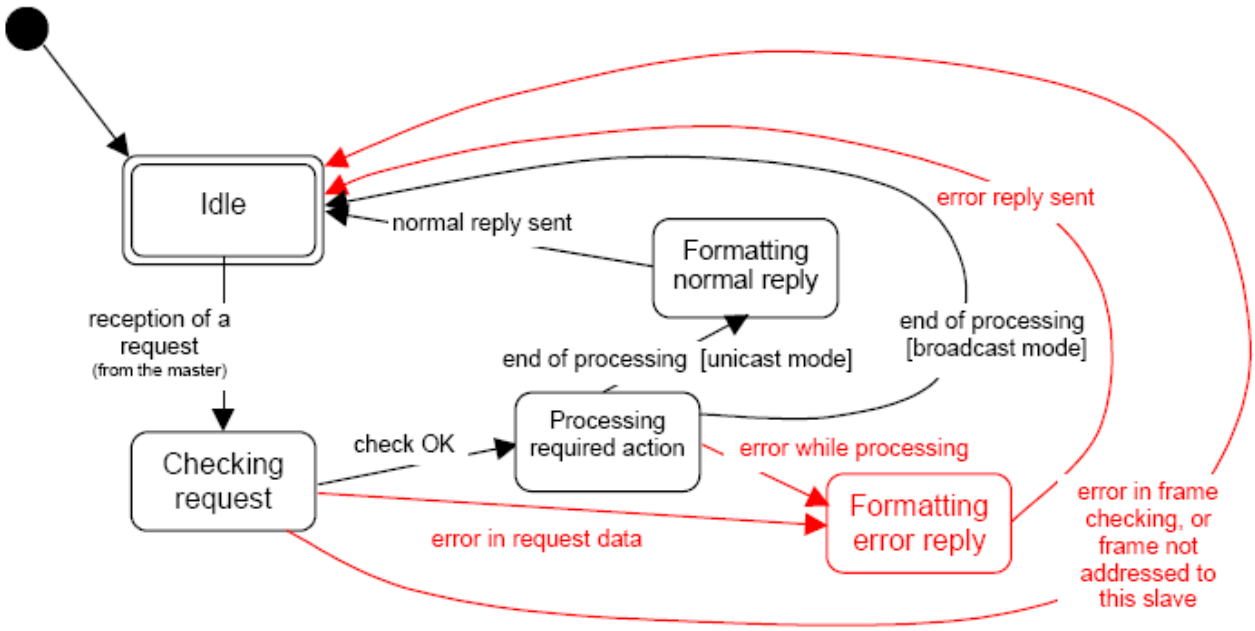


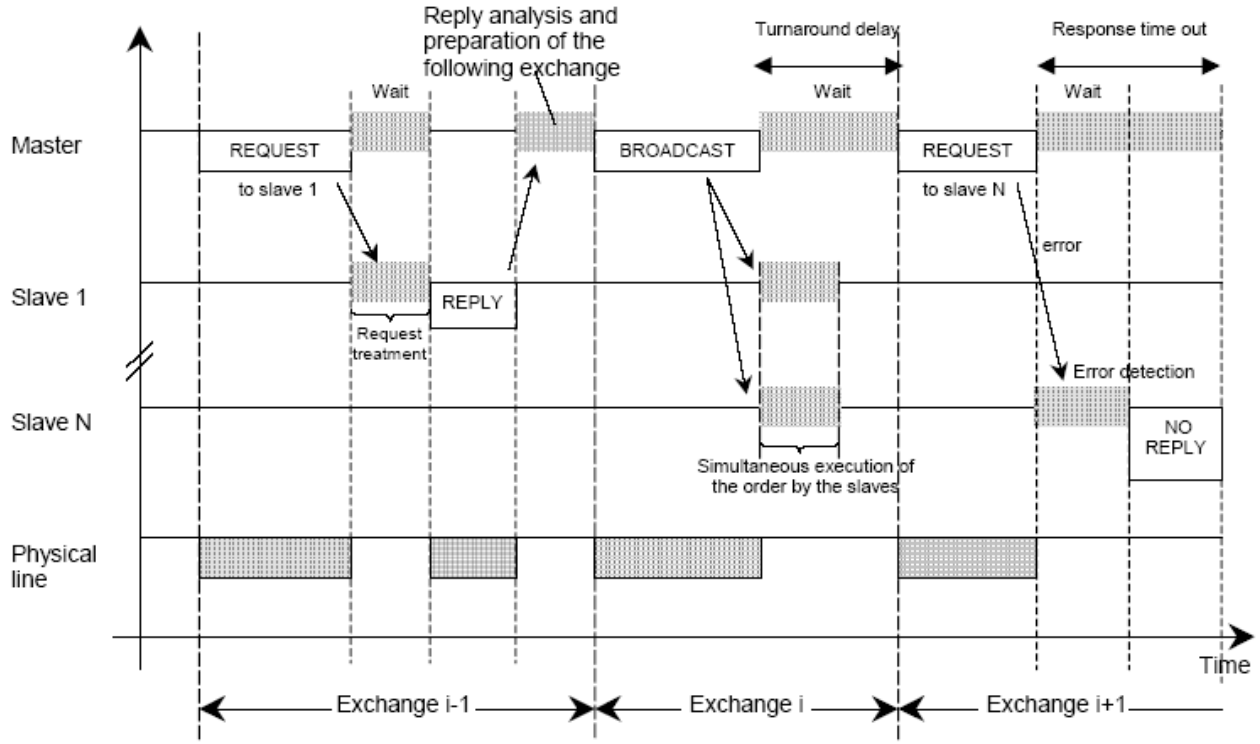
Diagrama de Estados do Escravo

A seguir temos algumas considerações a respeito do diagrama de estados apresentado:

- Estado "*idle*" - Nenhuma requisição pendente. Este é o estado inicial após a alimentação de um nó escravo;
- Quando uma requisição é recebida, o escravo verifica o pacote antes de executar a ação solicitada no pacote. Diferentes erros podem ocorrer: erro do formato da requisição, ação inválida, etc. No caso de um erro, uma resposta deve ser enviada para o mestre;
- Uma mensagem *unicast* requer que uma resposta seja formatada e enviada para o mestre assim que a ação requisitada é completada;
- Se um escravo detecta um erro no quadro recebido, nenhuma resposta é retornada para o mestre;
- Os contadores de diagnóstico Modbus são definidos e devem ser gerenciados por qualquer escravo no sentido de prover informações de diagnóstico. Estes contadores de diagnóstico podem ser adquiridos através da função de diagnóstico Modbus.

1.1.4.3 Diagrama de tempos da comunicação Mestre / Escravo

O diagrama abaixo apresenta 3 cenários específicos de comunicação Modbus Mestre / Escravo:



Cenários de comunicação Mestre / Escravo Modbus

Observações:

- A duração das fases de requisição, resposta e *broadcast* depende de características da comunicação (comprimento do quadro e processamento);
- A duração das fases de espera e tratamento dependem do tempo de processamento da requisição necessário para a aplicação contida nos nós escravos.

1.1.5 Modos de Transmissão Serial

Os dois modos de transmissão serial são definidos como: Modo RTU (*Remote Terminal Unit*) e modo ASCII.

Estes modos definem o conteúdo em bits dos campos das mensagens transmitidas serialmente no barramento. Eles determinam como a informação é empacotada nos campos das mensagens e, posteriormente, como são decodificadas.

O modo de transmissão (e o padrão da porta serial) deve ser o mesmo em todos os dispositivos conectados a linha serial.

Embora o modo ASCII seja requirido em algumas aplicações específicas, interoperabilidade entre dispositivos Modbus pode ser alcançada somente se cada dispositivo tiver o mesmo modo de transmissão. **Todos os dispositivos devem implementar o modo RTU.** A modo de transmissão ASCII é opcional.

Os dispositivos devem ser configurados pelos usuários para o modo de transmissão desejado, RTU ou ASCII. O modo padrão deve ser RTU.

1.1.5.1 Modo de Transmissão RTU

Quando dispositivos se comunicam em um barramento serial Modbus utilizando o modo RTU, cada byte (8 bits) na mensagem irá conter 2 caracteres hexadecimais de 4 bits. A principal vantagem deste modo é que sua maior densidade de caracteres permite um melhor processamento de dados do que o modo ASCII para o mesmo baud rate. Cada mensagem deve ser transmitida em um fluxo contínuo de caracteres.

O formato para cada byte no modo Modbus RTU é:

Sistema de codificação: Binário de 8 bits

Bits or Byte: 1 bit de início

8 bits de dados, sendo o de menor significado enviado primeiro.

1 bit de paridade

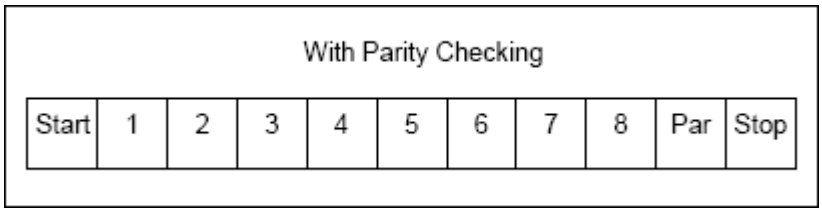
1 bit de parada

Paridade par é requerida, outros modos (paridade impar, sem paridade) podem ser utilizados. No sentido de garantir maior compatibilidade com outros produtos é recomendado suportar o modo sem paridade. O modo de paridade padrão deve ser paridade par.

Como os caracteres são transmitidos serialmente:

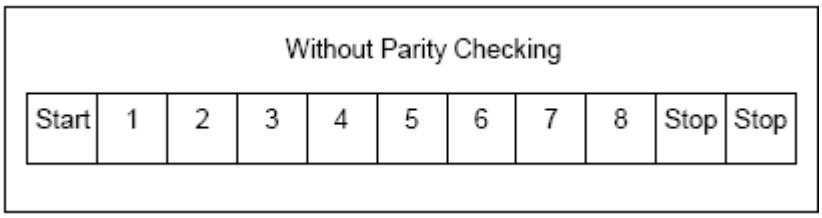
Cada caracter é enviado nesta ordem (da esquerda para a direita).

Bit menos significante ... Bit mais significante.



Sequência de bits no modo RTU com paridade

Os dispositivos podem aceitar configurações para verificação de paridade par, impar e sem paridade. Se o modo sem paridade é implementado, um bit de parada adicional deve ser transmitido para preencher o quadro de caracter mantendo o padrão de comunicação assíncrona com 11 bits.



Sequência de bits no modo RTU sem paridade

Campo de Verificação de Quadro:

Verificação Cíclica de Redundância (CRC)

Descrição do Quadro

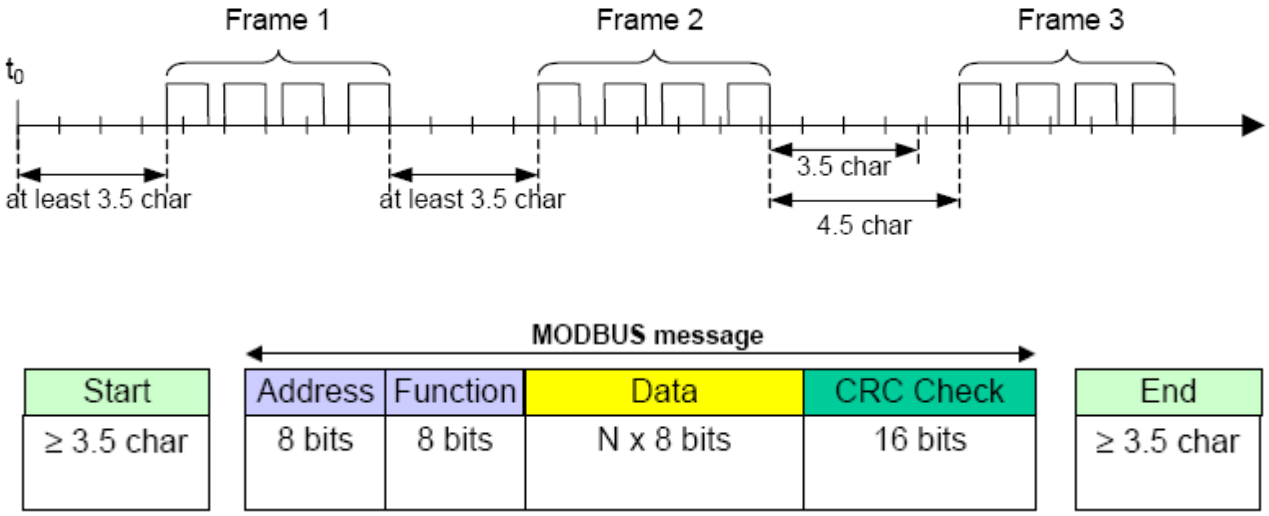
Slave Address	Function Code	Data	CRC
1 byte	1 byte	0 up to 252 byte(s)	2 bytes CRC Low, CRC Hi

Quadro de Mensagem RTU

1.1.5.1.1 Enquadramento de Mensagens Modbus RTU

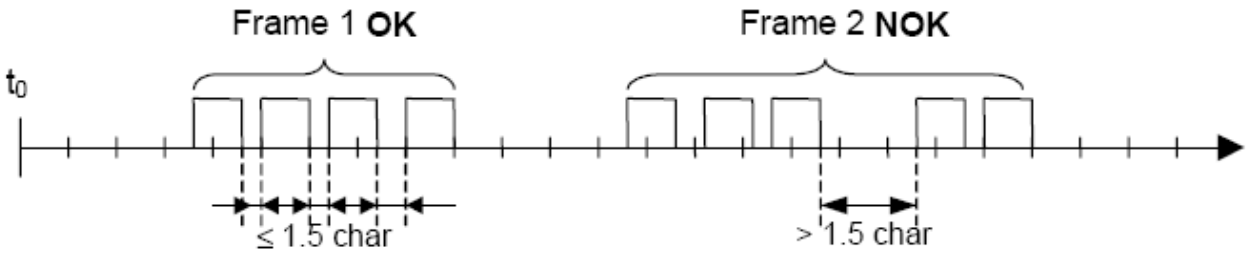
Uma mensagem Modbus é colocada pelo transmissor em um quadro que tem um começo e fim bem definidos. Isto permite que dispositivos que recebam um novo quadro conheçam o início da mensagem e também quando a mesma é completada. Mensagens parciais devem ser detectadas e erros devem ser gerados como resultado desta detecção.

No modo RTU, os quadros de mensagem são separados por um intervalo de silêncio de pelo menos 3.5 tempos de caracter. Nas figuras abaixo, este tempo é apresentado como t3,5.



Quadro de mensagem Modbus

O quadro inteiro da mensagem deve ser transmitido com um fluxo constante de caracteres. Se um tempo de silêncio maior do que o tempo de 1,5 caracter for detectado o quadro da mensagem é declarado incompleto e deve ser descartado pelo receptor.



Observação: A implementação do *driver* de recepção Modbus RTU deve implicar na gerencia de uma série de interrupções devido aos tempos de 1,5 e 3,5 caracteres.

Com altas taxas de comunicação (*baud rate*), este gerenciamento leva a grande carga de CPU. Consequentemente estes 2 temporizadores devem ser respeitados estritamente quando o baud rate é igual ou menor do que 19200 Bps. Para baud rates maiores do que 19200 Bps, um valor fixo de 2 tempos deve ser usado: é recomendado o uso do valor de 750 µs para o tempo entre caracteres (t1,5) e um valor de 1,750 ms para o tempo entre quadros (t3,5).

A figura abaixo apresenta o diagrama de estados do modo de transmissão RTU. Tanto o ponto de vista do mestre quanto do escravo são analisados no mesmo diagrama.

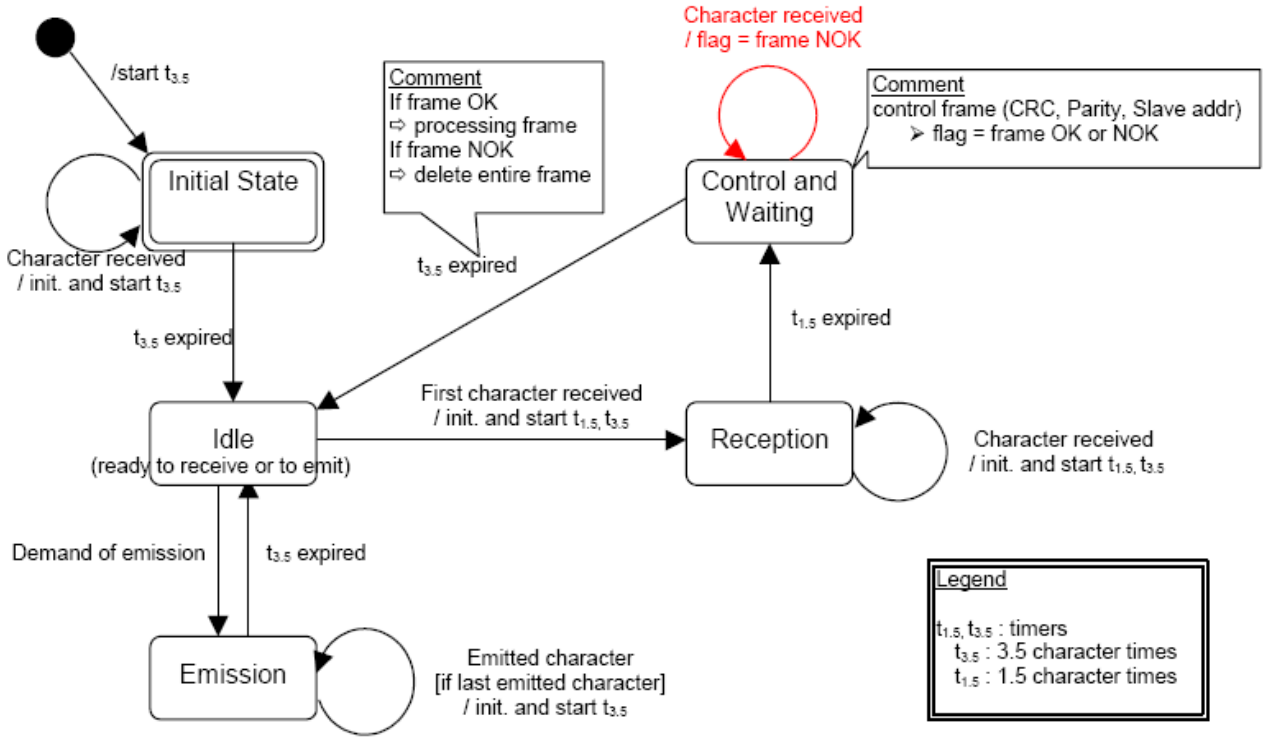


Diagrama de Estados do Modo Modbus RTU

Abaixo temos algumas considerações a respeito do diagrama de estados apresentado:

- A transição do estado inicial (*Initial State*) para o modo ocioso (*Idle*) necessita de um tempo de espera de t3,5. Esta ação garante o tempo de espera entre quadros;
- O estado ocioso é o estado normal quando os estados de transmissão (*emission*) e de recepção (*reception*) não estão ativos;
- No modo RTU, o enlace de comunicação é declarado ocioso quando não existir nenhuma transmissão ativa depois de um intervalo de tempo de pelo menos 3,5 caracteres;
- Quando o enlace está em estado ocioso, cada caracter transmitido detectado no enlace é definido como um início de um quadro. O enlace passa para o estado

ativo. Estando em estado ativo, o fim do quadro é detectado quando a transmissão de caracteres no enlace é interrompida por um intervalo de tempo superior a 3,5 caracteres;

- Após a detecção do fim do quadro, o cálculo de CRC e verificação é realizado. Após o campo de endereço é analisado para determinar se o quadro é destinado a este dispositivo. Se não for, o quadro é descartado. Para reduzir o tempo de processamento da recepção, o campo do endereço pode ser analisado assim que for recebido sem esperar o fim do quadro. Neste caso o CRC será calculado e verificado somente se o quadro é destinado a este escravo (quadros *broadcast* inclusive).

1.1.5.1.2 Verificação do CRC

O modo RTU inclui um campo de verificação de erro que é baseado em um método de Verificação cíclica de Redundância (CRC) incluído no conteúdo da mensagem.

O campo de CRC verifica o conteúdo da mensagem inteira.

O campo de CRC contém um valor de 16-bits implementado como dois bytes de 8 bits.

O campo de CRC é incluído como último campo da mensagem. Quando isto é realizado, o byte de menor ordem do campo é anexado primeiro, seguido pelo byte de maior ordem.

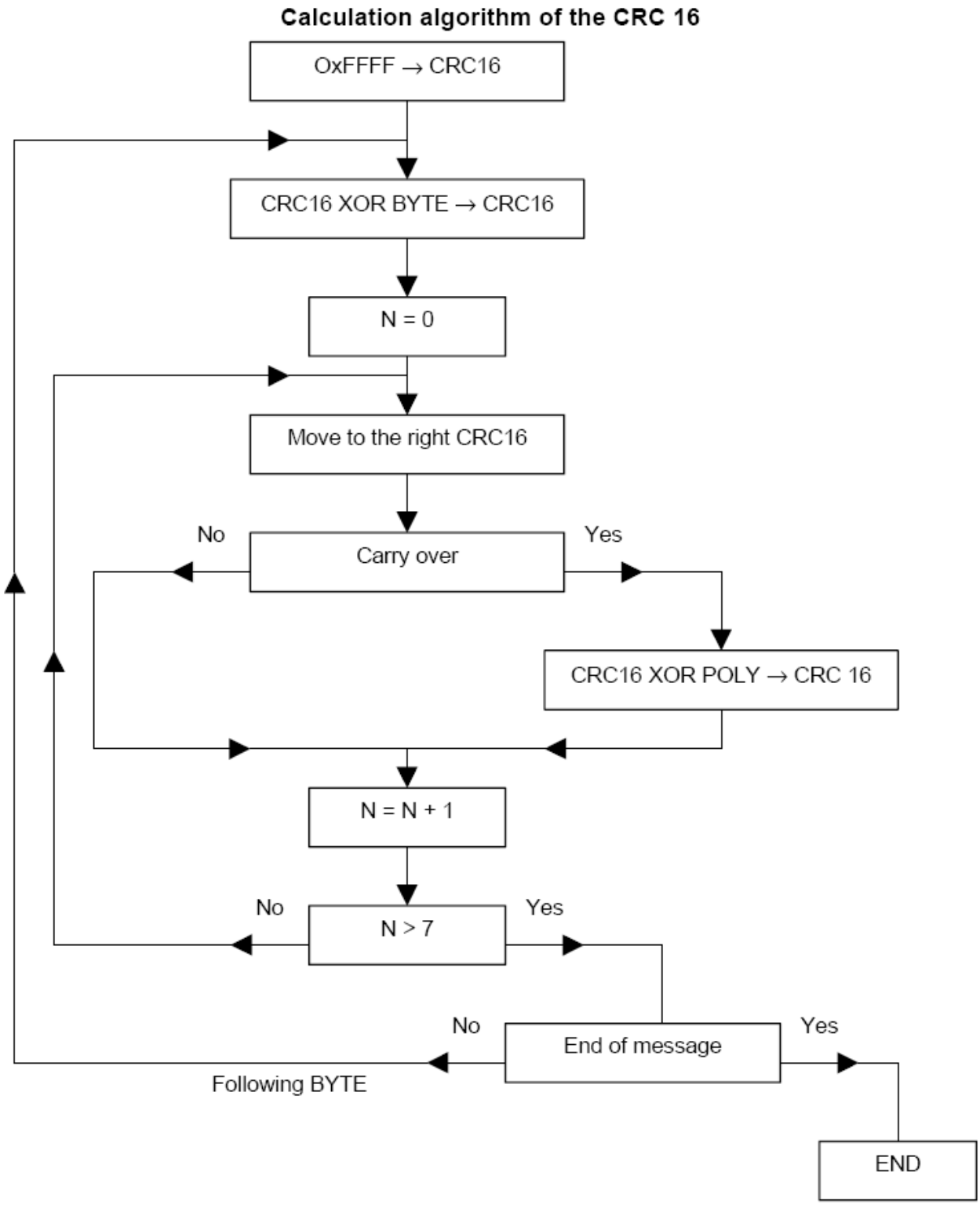
O byte CRC de maior ordem é o último byte enviado na mensagem. O valor do CRC é calculado pelo dispositivo que transmitiu o quadro. O dispositivo que recebe o quadro recalcula o CRC durante a recepção da mensagem, e compara o campo de CRC recebido com o CRC calculado. Se os valores não são iguais, o dispositivo retorna um erro.

O cálculo do CRC é iniciado carregando-se um registrador de 16 bits com todos os bits 1's (65535 decimal). Então um processo se inicia aplicando-se sucessivamente bytes de 8 bits ao conteúdo deste registrador. Somente os 8 bits de dados são utilizados para a geração do CRC. Os bits de início, parada e paridade não participam do cálculo do CRC.

Durante a geração do CRC, cada caracter de 8 bits passa por uma operação de “ou exclusivo” com o conteúdo do registrador de 16 bits. Então o resultado desta operação é deslocado no sentido do bit menos significativo com o bit mais significativo sendo preenchido por um zero. O bit menos significativo é extraído e examinado. Se este bit for 1, o conteúdo do registrador sofre nova operação de “ou exclusivo” com o polinômio

gerador do CRC16. Se o bit for 0, nenhum ação é executada. Este processo é repetido até que 8 deslocamentos tenham sido realizados. Depois do último deslocamento, o próximo byte de 8 bits sofre o mesmo processo de “ou exclusivo” e deslocamentos descrito acima. O conteúdo final do registrador, depois de todos os bytes da mensagem terem passado por este processo, é o valor do CRC. Quando o CRC é anexado a mensagem, o byte de menor ordem será anexado primeiro, seguido pelo byte de maior ordem. Abaixo é apresentado o fluxograma do cálculo do CRC para o quadro Modbus. As seguintes considerações são necessárias para a análise da figura:

- XOR = ou exclusivo
- N = número de bits de informação
- POLY = polinômio de geração do CRC 16 (1010 0000 0000 0001)
- Polinômio de geração utilizado = $1 + X_2+X_{15}+X_{16}$
- No CRC 16 o primeiro byte transmitido é o de menor significado.



Fluxograma do cálculo de geração do CRC

Example of CRC calculation (frame 02 07)

CRC register initialization		1111	1111	1111	1111
XOR 1st character		0000	0000	0000	0000
		1111	1111	1111	1101
	Move 1	0111	1111	1111	1110|1
		1010	0000	0000	0001
Flag to 1, XOR polynomial		1101	1111	1111	1111
	Move 2	0110	1111	1111	1111|1
Flag to 1, XOR polynomial		1010	0000	0000	0001
		1100	1111	1111	1110
	Move 3	0110	0111	1111	1110 | 0
	Move 4	0011	0011	1111	1111 | 1
		1010	0000	0000	0001
		1001	0011	1111	1110
	Move 5	0100	1001	1111	1111 | 0
	Move 6	0010	0100	1111	1111 | 1
		1010	0000	0000	0001
		1000	0100	1111	1110
	Move 7	0100	0010	0111	1111 | 0
	Move 8	0010	0001	0011	1111 | 0
		1010	0000	0000	0001
		1000	0001	0011	1110
		0000	0000	0000	0111
XOR 2nd character		1000	0001	0011	1001
	Move 1	0100	0000	1001	1100 | 1
		1010	0000	0000	0001
		1110	0000	1001	1101
	Move 2	0111	0000	0100	1110 | 1
		1010	0000	0000	0001
		1101	0000	0100	1111
	Move 3	0110	1000	0010	0111 | 1
		1010	0000	0000	0001
		1100	1000	0010	0110
	Move 4	0110	0100	0001	0011 | 0
	Move 5	0011	0010	0000	1001 | 1
		1010	0000	0000	0001
		1001	0010	0000	1000
	Move 6	0100	1001	0000	0100 | 0
	Move 7	0010	0100	1000	0010 | 0
	Move 8	0001	0010	0100	0001 | 0

Most significant | least significant

The CRC 16 of the frame is then: 4112