Tutorial sobre Comunicações Seriais e UART

Resumo

Este artigo fala sobre o uso de hardware serial com o FreeBSD.

Índice

1. A UART: O que é e como funciona

O controlador UART (Universal Asynchronous Receiver / Transmitter) é o componente chave do subsistema de comunicação serial de um computador. O UART pega bytes de dados e transmite os bits individuais de forma seqüencial. No destino, um segundo UART reúne os bits em bytes completos.

A transmissão serial é comumente usada com modems e para comunicação entre computadores, terminais e outros dispositivos sem rede.

Existem duas formas primárias de transmissão serial: Síncrona e Assíncrona. Dependendo dos modos suportados pelo hardware, o nome do subsistema de comunicação geralmente incluirá um `A` se ele suportar comunicações assíncronas, e um `S` se ele suportar Comunicações síncronas. Ambas as formas são descritas abaixo.

Algumas siglas comuns são:

UART Universal Asynchronous Receiver/Transmitter

USART Universal Synchronous-Asynchronous Receiver/Transmitter

1.1. Transmissão Serial Síncrona

A transmissão serial síncrona requer que o emissor e o receptor compartilhem um clock entre si, ou que o remetente forneça um sinal estroboscópico ou outro sinal de tempo para que o receptor saiba

quando deve "ler" o próximo bit dos dados. Na maioria das formas de comunicação serial síncrona, se não houver dados disponíveis em um dado instante para transmitir, um caractere de preenchimento deve ser enviado para que os dados sejam sempre transmitidos. A comunicação síncrona é geralmente mais eficiente, pois somente os bits de dados são transmitidos entre o emissor e o receptor, e a comunicação síncrona pode ser mais cara se fios e circuitos extras forem necessários para compartilhar um sinal de relógio entre o emissor e o receptor.

Uma forma de transmissão síncrona é usada com impressoras e dispositivos de disco fixo em que os dados são enviados em um conjunto de fios enquanto um clock ou strobe é enviado em um fio diferente. Impressoras e dispositivos de disco fixo normalmente não são dispositivos seriais porque a maioria dos padrões de interface de disco fixo envia uma palavra inteira de dados para cada sinal de clock ou de strobe usando um fio separado para cada bit da palavra. Na indústria de PCs, esses são conhecidos como dispositivos paralelos.

O hardware de comunicação serial padrão no PC não suporta operações síncronas. Este modo é descrito aqui apenas para fins de comparação.

1.2. Transmissão Serial Assíncrona

A transmissão assíncrona permite que os dados sejam transmitidos sem que o emissor tenha que enviar um sinal de relógio ao receptor. Em vez disso, o remetente e o receptor devem concordar com os parâmetros de tempo de antecedência e bits especiais são adicionados a cada palavra, os quais são usados para sincronizar as unidades de envio e recebimento.

Quando uma palavra é dada ao UART para uma transmissão assíncrona, um bit chamado "Start Bit" é adicionado ao início de cada palavra que deve ser transmitida. O Start Bit é usado para alertar o receptor de que uma palavra de dados está prestes a ser enviada e para forçar o clock do receptor a sincronizar-se com o clock do transmissor. Estes dois clocks devem ser precisos o suficiente para não ter um desvio de frequência em mais de 10% durante a transmissão dos bits restantes na palavra. (Esse requisito foi definido nos dias das teleimpressoras mecânicas e é facilmente atendido pelos equipamentos eletrônicos modernos.)

Após o Start Bit, os bits individuais da palavra de dados são enviados, com o Bit Menos Significativo (LSB) sendo enviado primeiro. Cada bit na transmissão é transmitido exatamente pelo mesmo período de tempo que todos os outros bits, e o receptor "olha" para o fio aproximadamente na metade do período atribuído a cada bit para determinar se o bit é um `1` ou um `0`. Por exemplo, se forem necessários dois segundos para enviar cada bit, o receptor examinará o sinal para determinar se é um `1` ou um `0` após ter passado um segundo, ele esperará dois segundos e examinará o valor do próximo bit, e assim por diante.

O remetente não sabe quando o receptor "olhou" para o valor do bit. O remetente só sabe quando o clock diz para começar a transmitir o próximo bit da palavra.

Quando toda a palavra de dados foi enviada, o transmissor pode adicionar um Bit de Paridade que o transmissor gera. O Bit de Paridade pode ser usado pelo receptor para executar uma verificação de erros simples. Então pelo menos um Stop Bit é enviado pelo transmissor.

Quando o receptor recebeu todos os bits na palavra de dados, ele pode verificar os bits de paridade (tanto o remetente quanto o receptor devem concordar se um bit de paridade deve ser usado), e

então o receptor procura um Stop Bit. Se o Stop Bit não aparecer quando é suposto aparecer, o UART considera a palavra inteira como ilegível e irá relatar um Framing Error para o processador do host quando a palavra de dados é lida. A causa comum de um Framing Error é que os clocks do emissor e do receptor não estavam sendo executados na mesma velocidade ou que o sinal foi interrompido.

Independentemente de os dados terem sido recebidos corretamente ou não, o UART descarta automaticamente os Bits de Start, Paridade e Stop. Se o emissor e o receptor forem configurados de forma idêntica, esses bits não serão passados para o host.

Se outra palavra estiver pronta para transmissão, o Start Bit da nova palavra pode ser enviado assim que o Stop Bit da palavra anterior for enviado.

Como os dados assíncronos são "auto-sincronizados", se não houver dados para transmitir, a linha de transmissão pode ficar inativa.

1.3. Outras funções UART

Além do trabalho básico de conversão de dados de paralelo para serial para transmissão e de serial para paralelo na recepção, um UART normalmente fornecerá circuitos adicionais para sinais que podem ser usados para indicar o estado da mídia de transmissão, e para regular o fluxo de dados no caso de o dispositivo remoto não estar preparado para aceitar mais dados. Por exemplo, quando o dispositivo conectado à UART é um modem, o modem pode informar a presença de uma operadora na linha telefônica enquanto o computador pode instruir o modem a reinicializar a si mesmo ou a não atender chamadas, aumentando ou diminuindo mais um desses sinais extras. A função de cada um desses sinais adicionais é definida no padrão EIA RS232-C.

1.4. Os padrões RS232-C e V.24

Na maioria dos sistemas de computador, o UART é conectado a um circuito que gera sinais que atendem à especificação EIA RS232-C. Há também um padrão CCITT chamado V.24 que reflete as especificações incluídas no RS232-C.

1.4.1. Atribuições de bit RS232-C (marcas e espaços)

No RS232-C, um valor de `1` é chamado de `Mark` e um valor de `0` é chamado de `Space`. Quando uma linha de comunicação está inativa, a linha é chamada de "Marking", ou seja, está transmitindo o valor `1` continuamente.

O bit de início sempre tem um valor de `0` (um space). O bit de parada sempre tem um valor de `1` (uma mark). Isso significa que sempre haverá uma transição Mark (1) para Space (0) na linha no início de cada palavra, mesmo quando várias palavras forem transmitidas de volta para trás. Isso garante que o remetente e o destinatário possam ressincronizar seus relógios independentemente do conteúdo dos bits de dados que estão sendo transmitidos.

O tempo inativo entre os bits de Stop e Start não precisa ser um múltiplo exato (incluindo zero) da taxa de bits do link de comunicação, mas a maioria dos UARTs é projetada dessa maneira para simplificar.

No RS232-C, o sinal "Marking" (a 1) é representado por uma tensão entre -2 VDC e -12 VDC, e um sinal "Spacing" (um 0) é representado por uma tensão entre 0 e +12 VDC. O transmissor deve enviar +12 VDC ou -12 VCC, e o receptor deve permitir alguma perda de tensão em cabos longos. Alguns transmissores em dispositivos de baixa potência (como computadores portáteis) às vezes usam apenas +5 VCC e -5 VCC, mas esses valores ainda são aceitáveis para um receptor RS232-C, desde que os comprimentos dos cabos sejam curtos.

1.4.2. Sinal de quebra RS232-C

O RS232-C também especifica um sinal chamado de `Break` (quebra), que é causado pelo envio de valores contínuos de espaçamento (sem bits de início ou de parada). Quando não há eletricidade presente no circuito de dados, a linha é considerada como enviando um `Break`.

O sinal `Break` deve ter uma duração maior que o tempo que leva para enviar um byte completo mais os bits Start, Stop e Paridade. A maioria das UARTs pode distinguir entre um Framing Error e um intervalo, mas se a UART não puder fazer isso, a detecção de Framing Error pode ser usada para identificar quebras.

Nos dias das teleimpressoras, quando numerosas impressoras em todo o país eram conectadas em série (como serviços de notícias), qualquer unidade poderia causar um `Break` abrindo temporariamente todo o circuito de modo que nenhuma corrente fluísse. Isso foi usado para permitir que um local com notícias urgentes interrompesse algum outro local que estava enviando informações no momento.

Nos sistemas modernos existem dois tipos de sinais de quebra. Se o Break for maior que 1,6 segundos, será considerado um "Modem Break", e alguns modems podem ser programados para encerrar a conversa e colocar no gancho ou entrar no modo de comando dos modems quando o modem detectar este sinal. Se a quebra for menor que 1,6 segundos, significa uma quebra de dados e cabe ao computador remoto responder a esse sinal. Às vezes essa forma de quebra é usada como um sinal de Atenção ou Interrupção e às vezes é aceita como um substituto para o caractere ASCII CONTROL-C.

Marcas e espaços também são equivalentes a "furos" e "sem furos" em sistemas de fita de papel.

As quebras não podem ser geradas a partir da fita de papel ou de qualquer outro valor de byte, uma vez que os bytes são sempre enviados com bit Start e Stop. A UART geralmente é capaz de gerar o sinal de espaçamento contínuo em resposta a um comando especial do processador host.

1.4.3. Dispositivos RS232-C DTE e DCE

A especificação RS232-C define dois tipos de equipamento: o Data Terminal Equipment (DTE) e o Data Carrier Equipment (DCE). Normalmente, o dispositivo DTE é o terminal (ou computador) e o DCE é um modem. Em toda a linha telefônica, no outro extremo de uma conversa, o modem receptor também é um dispositivo DCE e o computador conectado a esse modem é um dispositivo DTE. O dispositivo DCE recebe sinais nos pinos que o dispositivo DTE transmite e vice-versa.

Quando dois dispositivos DTE ou DCE devem ser conectados sem utilizar um modem ou um tradutor de mídia similar entre eles, um modem NULL deve ser usado. O modem NULL reorganiza

eletricamente o cabeamento para que a saída do transmissor seja conectada à entrada do receptor no outro dispositivo e vice-versa. Traduções semelhantes são executadas em todos os sinais de controle, de modo que cada dispositivo veja o que acha que são sinais de DCE (ou DTE) do outro dispositivo.

O número de sinais gerados pelos dispositivos DTE e DCE não é simétrico. O dispositivo DTE gera menos sinais para o dispositivo DCE do que o dispositivo DTE recebe do DCE.

1.4.4. Atribuições de pinos RS232-C

A especificação EIA RS232-C (e o equivalente ITU, V.24) requer um conector de vinte e cinco pinos (geralmente um DB25) e define a finalidade da maioria dos pinos nesse conector.

No IBM Personal Computer e em sistemas semelhantes, um subconjunto de sinais RS232-C é fornecido por meio de conectores de nove pinos (DB9). Os sinais que não estão incluídos no conector do PC lidam principalmente com a operação síncrona, e esse modo de transmissão não é suportado pelo UART que a IBM selecionou para uso no IBM PC.

Dependendo do fabricante do computador, um DB25, um DB9 ou ambos os tipos de conectores podem ser usados para comunicações RS232-C. (O IBM PC também usa um conector DB25 para a interface de impressora paralela, o que causa alguma confusão.)

Abaixo está uma tabela das atribuições de sinal RS232-C nos conectores DB25 e DB9.

Pinos DB25 RS232-C	Pinos DB9 IBM PC	Símbolo do Circuito EIA	Símbolo do Circuito CCITT	Nome Comum	Fonte de sinal	Descrição
1	-	AA	101	PG/FG	-	Quadro / aterramento de proteção
2	3	BA	103	TD	DTE	Transmit Data
3	2	BB	104	RD	DCE	Receive Data
4	7	CA	105	RTS	DTE	Request to Send
5	8	CB	106	CTS	DCE	Clear to Send
6	6	CC	107	DSR	DCE	Data Set Ready
7	5	AV	102	SG/GND	-	Signal Ground
8	1	CF	109	DCD/CD	DCE	Data Carrier Detect
9	-	-	-	-	-	Reserved for Test

Pinos DB25 RS232-C	Pinos DB9 IBM PC	Símbolo do Circuito EIA	Símbolo do Circuito CCITT	Nome Comum	Fonte de sinal	Descrição
10	-	-	-	-	-	Reserved for Test
11	-	-	-	-	-	Reserved for Test
12	-	CI	122	SRLSD	DCE	Sec. Recv. Line Signal Detector
13	-	SCB	121	SCTS	DCE	Secondary Clear to Send
14	-	SBA	118	DST	DTE	Secondary Transmit Data
15	-	DB	114	TSET	DCE	Trans. Sig. Element Timing
16	-	SBB	119	SRD	DCE	Secondary Received Data
17	-	DD	115	RSET	DCE	Receiver Signal Element Timing
18	-	-	141	LOOP	DTE	Local Loopback
19	-	SCA	120	SRS	DTE	Secondary Request to Send
20	4	CD	108.2	DTR	DTE	Data Terminal Ready
21	-	-	-	RDL	DTE	Remote Digital Loopback
22	9	CE	125	RI	DCE	Ring Indicator
23	-	CH	111	DSRS	DTE	Data Signal Rate Selector

Pinos DB25 RS232-C	Pinos DB9 IBM PC	Símbolo do Circuito EIA	Símbolo do Circuito CCITT	Nome Comum	Fonte de sinal	Descrição
24	-	DA	113	TSET	DTE	Trans. Sig. Element Timing
25	-	-	142	-	DCE	Test Mode

1.5. Bits, Baud e Simbolos

Baud é uma medida de velocidade de transmissão em comunicação assíncrona. Devido aos avanços na tecnologia de comunicação por modem, esse termo é frequentemente mal utilizado na descrição das taxas de dados em dispositivos mais recentes.

Tradicionalmente, uma taxa de transmissão representa o número de bits que estão realmente sendo enviados pela mídia, não a quantidade de dados que é realmente movida de um dispositivo DTE para outro. A contagem de Baud inclui os bits de Start, Stop e Paridade que são gerados pelo UART de envio e removidos pelo UART de recebimento. Isso significa que palavras de dados de sete bits na verdade levam 10 bits para serem completamente transmitidas. Portanto, um modem capaz de mover 300 bits por segundo de um lugar para outro normalmente só pode mover 30 palavras de 7 bits se a Paridade for usada e um bit de Start e Stop estiver presente.

Se palavras de dados de 8 bits são usadas e bits de paridade também são usados, a taxa de dados cai para 27,27 palavras por segundo, porque agora leva 11 bits para enviar as palavras de oito bits, e o modem ainda envia apenas 300 bits por segundo.

A fórmula para converter bytes por segundo em uma taxa de transmissão e vice-versa era simples até que os modems de correção de erros apareceram. Esses modems recebem o fluxo serial de bits da UART no computador host (mesmo quando os modems internos são usados, os dados ainda são frequentemente serializados) e convertem os bits de volta em bytes. Esses bytes são então combinados em pacotes e enviados pela linha telefônica usando um método de transmissão síncrona. Isso significa que os bits de Stop, Start e Paridade adicionados pelo UART no DTE (o computador) foram removidos pelo modem antes da transmissão pelo modem de envio. Quando esses bytes são recebidos pelo modem remoto, o modem remoto adiciona bits de Start, Stop e paridade às palavras, converte-os em um formato serial e envia-os para o UART receptor no computador remoto, que retira o Start, Stop e bits de paridade.

A razão pela qual todas essas conversões extras são feitas é para que os dois modems possam executar a correção de erros, o que significa que o modem receptor pode solicitar ao modem de envio para reenviar um bloco de dados que não foi recebido com a soma de verificação correta. Essa verificação é feita pelos modems, e os dispositivos DTE geralmente não sabem que o processo está ocorrendo.

Ao separar os bits de Start, Stop e Paridade, os bits adicionais de dados que os dois modems devem compartilhar entre si para executar a correção de erros são praticamente ocultados da taxa de transmissão efetiva vista pelo equipamento DTE de envio e recebimento. Por exemplo, se um modem enviar dez palavras de 7 bits para outro modem sem incluir os bits Start, Stop e Paridade, o

modem de envio poderá adicionar 30 bits de suas próprias informações que o modem receptor pode usar para corrigir erros. sem afetar a velocidade de transmissão dos dados reais.

O uso do termo Baud é ainda mais confuso pelos modems que executam compressão. Uma única palavra de 8 bits transmitida pela linha telefônica pode representar uma dúzia de palavras que foram transmitidas para o modem de envio. O modem de recebimento irá expandir os dados de volta ao seu conteúdo original e passar esses dados para o DTE de recebimento.

Os modems modernos também incluem buffers que permitem que a taxa na qual os bits se movem pela linha telefônica (do DCE para o DCE) seja uma velocidade diferente da velocidade que os bits se movem entre o DTE e o DCE em ambas as extremidades da conversação. Normalmente, a velocidade entre o DTE e o DCE é maior que a velocidade do DCE para o DCE devido ao uso de compactação pelos modems.

Como o número de bits necessários para descrever um byte variou durante a viagem entre as duas máquinas, mais as diferentes velocidades de bits por segundo usadas nos links DTE-DCE e DCE-DCE, o uso do termo Baud para descrever a velocidade geral de comunicação causa problemas e pode deturpar a velocidade real de transmissão. Então Bits Por Segundo (bps) é o termo correto a ser usado para descrever a taxa de transmissão na interface DCE para DCE e Baud ou Bits Por Segundo são termos aceitáveis para uso quando uma conexão é feita entre dois sistemas com uma conexão com fio ou se estiver em uso um modem que não esteja executando correção de erros ou compactação.

Os modernos modems de alta velocidade (2400, 9600, 14.400 e 19.200bps) na realidade ainda operam a 2400 ou abaixo de 2400 baud, ou mais precisamente, 2400 símbolos por segundo. O modem de alta velocidade é capaz de codificar mais bits de dados em cada Symbol usando uma técnica chamada Constellation Stuffing, é por isso que a taxa efetiva de bits por segundo do modem é maior, mas o modem continua a operar dentro da largura de banda limitada de áudio que o sistema telefônico fornece. Modems operando a 28.800 e velocidades mais altas têm taxas variáveis de Symbol, mas a técnica é a mesma.

1.6. O computador pessoal IBM e o UART

Começando com o IBM Personal Computer original, a IBM selecionou o National Semiconductor INS8250 UART para uso no adaptador IBM PC Paralelo/Serial. Gerações subsequentes de computadores compatíveis da IBM e de outros fornecedores continuaram a usar o INS8250 ou versões aprimoradas da família UART da National Semiconductor.

1.6.1. Árvore Genealógica da National Semiconductor UART

Houve várias versões e gerações subseqüentes do INSART50 UART. Cada versão principal está descrita abaixo.

```
INS8250  -> INS8250B
  \
   \
    \-> INS8250A -> INS82C50A
              \
```

```
              \
               \-> NS16450 -> NS16C450
                        \
                         \
                          \-> NS16550 -> NS16550A -> PC16550D
```

INS8250

Esta parte foi usada no IBM PC original e no IBM PC/XT. O nome original para esta parte era o INS8250 ACE (Elemento de Comunicação Assíncrona) e ele era feito com tecnologia NMOS.

O 8250 usa oito portas de I/O e tem um envio de um byte e um buffer de recebimento de um byte. Esta UART original tem várias "race conditions" e outras falhas. O IBM BIOS original incluia código para contornar essas falhas, mas isso tornava o BIOS dependente das falhas estarem presentes, portanto, componentes subsequentes como o 8250A, 16450 ou 16550 não podiam ser usados no IBM PC original ou no IBM PC/XT.

INS8250-B

Esta é a velocidade mais lenta do INS8250 feito a partir da tecnologia NMOS. Ele contém os mesmos problemas que o INS8250 original.

INS8250A

Uma versão melhorada do INS8250 usando a tecnologia XMOS com várias falhas funcionais corrigidas. O INS8250A foi usado inicialmente em computadores clones de PC por fornecedores que usavam projetos de BIOS "limpos". Devido às correções no chip, este componente não pode ser usado com um BIOS compatível com o INS8250 ou o INS8250B.

INS82C50A

Esta é uma versão CMOS (baixo consumo de energia) do INS8250A e possui características funcionais semelhantes.

NS16450

O mesmo que o NS8250A com melhorias para que possa ser usado com projetos de barramento de CPU mais rápidos. A IBM usou esse componente no IBM AT e atualizou o IBM BIOS para não depender mais dos erros no INS8250.

NS16C450

Esta é uma versão CMOS (baixo consumo de energia) do NS16450.

NS16550

O mesmo que NS16450 com um buffer de envio e recebimento de 16 bytes, mas o design do buffer era falho e não podia ser usado com segurança.

NS16550A

O mesmo que NS16550 com as falhas de buffer corrigidas. O 16550A e seus sucessores se tornaram o projeto UART mais popular na indústria de PCs, principalmente devido à sua capacidade de lidar de forma confiável com taxas de dados mais altas em sistemas operacionais com tempos de resposta de interrupção lentos.

NS16C552

Este componente consiste em dois UARTs CMOS NS16C550A em um único chip.

PC16550D

O mesmo que NS16550A com falhas sutis corrigidas. Esta é a revisão D da família 16550 e é o mais recente projeto disponível da National Semiconductor.

1.6.2. O NS16550AF e o PC16550D são a mesma coisa

A National reorganizou seu sistema de numeração de peças há alguns anos e o NS16550AFN não existe mais com esse nome. (Se você tiver um NS16550AFN, observe o código de data da peça, que é um número de quatro dígitos que geralmente começa com nove. Os dois primeiros dígitos do número são o ano, e os dois últimos dígitos são a semana do ano em que a peça foi fabricada. Se você tem um NS16550AFN, ele provavelmente tem alguns anos.)

Os novos números são como PC16550DV, com pequenas diferenças nas letras de sufixo, dependendo do material da embalagem e sua forma. (Uma descrição do sistema de numeração pode ser encontrada abaixo.)

É importante entender que em algumas lojas, você pode pagar US$ 15 por um NS16550AFN fabricado em 1990 e no próximo bin encontrar as novas peças PC16550DN com pequenas correções que o National fez desde que a peça AFN estava em produção, o PC16550DN foi provavelmente feito nos últimos seis meses e custa metade (tão baixo quanto US$ 5 se comprado em volume) do NS16550AFN porque ele está prontamente disponível.

Como o fornecimento de chips NS16550AFN continua encolhendo, o preço provavelmente continuará aumentando até que mais pessoas descubram e aceitem que o PC16550DN realmente tem a mesma função que o número de peça antigo.

1.6.3. Sistema de Numeração de Peças da National Semiconductor

Os números de peça mais antigos NS*nnnnnrqp* agora são do formato PC*nnnnnrgp*.

O *r* é o campo de revisão. A revisão atual do 16550 da National Semiconductor é `D`.

O *p* é o campo que define o tipo de encapsulamento. Os tipos são:

"F"	QFP	(quad flat pack) L lead type
"N"	DIP	(dual inline package) through hole straight lead type
"V"	LPCC	(lead plastic chip carrier) J lead type

O *g* é o campo de classificação do produto. Se um `I` precede a letra do tipo de encapsulamento, ele indica uma parte de classe "industrial", que possui especificações mais altas que uma parte padrão, mas não tão alta quanto o componente Especificação Militar (Milspec) . Este é um campo opcional.

Então, o que costumávamos chamar de NS16550AFN (Pacote DIP) agora é chamado de PC16550DN

ou PC16550DIN.

1.7. Outros fornecedores e UARTs semelhantes

Ao longo dos anos, o 8250, o 8250A, o 16450 e o 16550 foram licenciados ou copiados por outros fornecedores de chips. No caso do 8250, 8250A e 16450, o circuito exato (o "megacell") foi licenciado para muitos fornecedores, incluindo a Western Digital e a Intel. Outros fornecedores realizaram engenharia reversa da peça ou produziram emulações que tiveram comportamento semelhante.

Nos modems internos, o projetista de modem freqüentemente emula o 8250A/16450 com o microprocessador de modem, e o UART emulado frequentemente terá um buffer oculto que consiste em várias centenas de bytes. Por causa do tamanho do buffer, essas emulações podem ser tão confiáveis quanto uma 16550A em sua capacidade de lidar com dados de alta velocidade. No entanto, a maioria dos sistemas operacionais ainda relatará que o UART é apenas um 8250A ou 16450, e pode não fazer uso efetivo do buffer extra presente no UART emulado, a menos que drivers especiais sejam usados.

Alguns fabricantes de modem são motivados pelas forças do mercado a abandonar um design que possui centenas de bytes de buffer e, em vez disso, usam uma UART 16550A para que o produto compare favoravelmente nas comparações de mercado, embora o desempenho efetivo possa ser reduzido por essa ação.

Um equívoco comum é que todas as partes com "16550A" escritas nelas são idênticas no desempenho. Existem diferenças e, em alguns casos, falhas definitivas na maioria desses clones 16550A.

Quando o NS16550 foi desenvolvido, a National Semiconductor obteve várias patentes sobre o projeto e também limitou o licenciamento, tornando mais difícil para outros fornecedores fornecer um chip com características semelhantes. Por causa das patentes, projetos de engenharia reversa e emulações tiveram que evitar infringir as reivindicações cobertas pelas patentes. Posteriormente, essas cópias quase nunca funcionam exatamente da mesma forma que a NS16550A ou a PC16550D, que são as peças que a maioria dos fabricantes de computadores e modems deseja comprar, mas às vezes não estão dispostas a pagar o preço necessário para obter a peça genuína.

Algumas das diferenças nas peças do clone 16550A não são importantes, enquanto outras podem impedir que o dispositivo seja usado com um determinado sistema operacional ou driver. Essas diferenças podem aparecer ao usar outros drivers ou quando ocorrem determinadas combinações de eventos que não foram bem testadas ou consideradas no driver Windows®. Isso ocorre porque a maioria dos fornecedores de modem e de fabricantes de clones do 16550 usam os drivers da Microsoft do Windows® para Workgroups 3.11 e Microsoft®MS-DOS® como o principal teste de compatibilidade com o NS16550A. Esse critério excessivamente simplista e significa que se um sistema operacional diferente for usado, poderão surgir problemas devido a diferenças sutis entre os clones e os componentes genuínos.

A National Semiconductor disponibilizou um programa chamado COMTEST que realiza testes de compatibilidade independentemente de qualquer driver do sistema operacional. Deve ser lembrado que o propósito deste tipo de programa é demonstrar as falhas nos produtos dos concorrentes, de modo que o programa reportará diferenças importantes e extremamente sutis no comportamento da peça que está sendo testada.

Em uma série de testes realizados pelo autor deste documento em 1994, componentes fabricados pela National Semiconductor, TI, StarTech e CMD, bem como megacells e emulações incorporadas em modems internos foram testados com o COMTEST. Uma contagem de diferença para alguns desses componentes está listada abaixo. Como esses testes foram realizados em 1994, eles podem não refletir o desempenho atual do produto de um determinado fornecedor.

Deve-se notar que o COMTEST normalmente aborta quando um número excessivo ou certos tipos de problemas são detectados. Como parte desse teste, o COMTEST foi modificado para não abortar, independentemente de quantas diferenças fossem encontradas.

Fornecedor	**Número da peça**	**Erros (também conhecidos como "diferenças")**
National	(PC16550DV)	0
National	(NS16550AFN)	0
National	(NS16C552V)	0
TI	(TL16550AFN)	3
CMD	(16C550PE)	19
StarTech	(ST16C550J)	23
Rockwell	Modem de referência com 16550 interno ou uma emulação (RC144DPi / C3000-25)	117
Sierra	Modem com um 16550 interno (SC11951/SC11351)	91

Até o momento, o autor deste documento não encontrou nenhuma peça não-National que relate diferença zero usando o programa COMTEST. Também deve ser notado que a National teve cinco versões do 16550 ao longo dos anos e as partes mais novas se comportam de maneira um pouco diferente do NS16550AFN clássico que é considerado o benchmark para funcionalidade. O COMTEST parece fechar os olhos para as diferenças dentro da linha de produto da National e não relata nenhum erro nas peças da National (exceto para o original 16550) mesmo quando há erratas oficiais que descrevem erros nas revisões A, B e C das partes, então este viés no COMTEST deve ser levado em consideração.

É importante entender que uma simples contagem de diferenças do COMTEST não revela muito sobre quais diferenças são importantes e quais não são. Por exemplo, cerca de metade das diferenças relatadas nos dois modems listados acima que têm UARTs internas foram causadas pelos clones UARTs que não suportam modos de caractere de cinco e seis bits. Todos os UARTs 16550, 16450 e 8250 reais suportam esses modos e o COMTEST verifica a funcionalidade desses modos, de modo que mais de cinquenta diferenças são relatadas. No entanto, quase nenhum modem moderno suporta caracteres de cinco ou seis bits, particularmente aqueles com recursos de correção de erros e compressão. Isso significa que as diferenças relacionadas aos modos de caractere de cinco e seis bits podem ser desconsideradas.

Muitas das diferenças que o COMTEST reporta têm a ver com o tempo. Em muitos projetos de

clones, quando o host lê de uma porta, os bits de status em alguma outra porta podem não ser atualizados na mesma quantidade de tempo (alguns mais rápidos, alguns mais lentos) que um NS16550AFN *real* e o COMTEST procura por essas diferenças. Isso significa que o número de diferenças pode ser enganoso, pois um dispositivo pode ter apenas uma ou duas diferenças, mas elas serem extremamente sérias, e algum outro dispositivo que atualiza o status de registro mais rápido ou mais devagar que a peça de referência (que provavelmente nunca afetaria o operação de um driver devidamente escrito) poderia ter dezenas de diferenças relatadas.

O COMTEST pode ser usado como uma ferramenta de triagem para alertar o administrador sobre a presença de componentes potencialmente incompatíveis que podem causar problemas ou que precisam ser tratados como um caso especial.

Se você executar o COMTEST em um 16550 que esteja em um modem ou se um modem estiver conectado à porta serial, será necessário primeiro emitir um comando ATE0&W para o modem para que o modem não faça eco de nenhum dos caracteres de teste. Se você esquecer de fazer isso, o COMTEST informará pelo menos essa diferença:

```
Error (6)...Timeout interrupt failed: IIR = c1  LSR = 61
```

1.8. Registradores 8250/16450/16550

O UART 8250/16450/16550 ocupa oito endereços contíguos de porta de I/O. No IBM PC, há dois locais definidos para essas oito portas e eles são conhecidos coletivamente como COM1 e COM2. Os fabricantes de PC-clones e placas adicionais criaram duas áreas adicionais conhecidas como COM3 e COM4, mas essas portas COM extras entram em conflito com outro hardware em alguns sistemas. O conflito mais comum é com adaptadores de vídeo que fornecem emulação IBM 8514.

A COM1 está localizada de 0x3f8 a 0x3ff e normalmente usa o IRQ 4. A COM2 está localizada de 0x2f8 a 0x2ff e normalmente usa IRQ 3. A COM3 está localizada de 0x3e8 a 0x3ef e não tem IRQ padronizado. A COM4 está localizada de 0x2e8 a 0x2ef e não tem IRQ padronizado.

Uma descrição das portas I/O da UART 8250/16450/16550 é fornecida abaixo.

Porta I/O	**Acesso permitido**	**Descrição**
+0x00	write (DLAB==0)	Transmit Holding Register (THR). As informações gravadas nessa porta são tratadas como palavras de dados e serão transmitidas pela UART.
+0x00	read (DLAB==0)	Receive Buffer Register (RBR). Quaisquer palavras de dados recebidas pelo UART a partir do link serial são acessadas pelo host lendo esta porta.

Porta I/O	Acesso permitido	Descrição
+0x00	write/read (DLAB==1)	Divisor Latch LSB (DLL) Este valor será dividido a partir do clock de entrada principal (no IBM PC, o clock principal é 1.8432MHz) e o clock resultante determinará a taxa de transmissão do UART. Este registrador contém os bits de 0 a 7 do divisor.
+0x01	write/read (DLAB==1)	Divisor Latch MSB (DLH) Este valor será dividido a partir do clock de entrada principal (no IBM PC, o clock principal é 1.8432MHz) e o clock resultante determinará a taxa de transmissão do UART. Este registrador contém os bits 8 a 15 do divisor.
+0x01	write/read (DLAB==0)	Interrupt Enable Register (IER) A UART 8250/16450/16550 classifica os eventos em uma de quatro categorias. Cada categoria pode ser configurada para gerar uma interrupção quando qualquer um dos eventos ocorrer. A UART 8250/16450/16550 gera um único sinal de interrupção externa, independentemente de quantos eventos nas categorias ativadas ocorreram. Cabe ao processador host responder à interrupção e depois pesquisar as categorias de interrupção ativadas (geralmente todas as categorias têm interrupções ativadas) para determinar a(s) causa(s) verdadeira(s) da interrupção. Bit 7 → Reserved, always 0. Bit 6 → Reserved, always 0. Bit 5 → Reserved, always 0. Bit 4 → Reserved, always 0. Bit 3 → Ativa o Modem Status Interrupt (EDSSI). Definir esse bit como "1" permite que o UART gere uma interrupção quando ocorrer uma alteração em uma ou mais das linhas de status. Bit 2 → Ativa a interrupção de status da linha receptora (ELSI) Configurar este bit como "1" faz com que o UART gere uma interrupção quando um erro (ou um sinal BREAK) for detectado nos dados de entrada. Bit 1 → Ativa a Interrupção Vazia do Registro de Holding do Transmissor (ETBEI) Configurar este bit como "1" faz com que o UART gere uma interrupção quando o UART tiver espaço para um ou mais caracteres adicionais que serão transmitidos. Bit 0 → Ativar a Interrupção Disponível de Dados Recebidos (ERBFI) Configurar este bit para "1" faz com que o UART gere uma interrupção quando o UART tiver recebido caracteres suficientes para exceder o nível de disparo do FIFO, ou o temporizador FIFO tiver expirado (dados antigos) ou um único caractere tiver sido recebido quando o FIFO está desativado.

Porta I/O	Acesso permitido	Descrição
+0x02	write	Registro de Controle FIFO (FCR) (Esta porta não existe no UART 8250 e 16450). Bit 7 → Receiver Trigger Bit #1 Bit 6 → Trigger do Receptor Bit #0 Esses dois bits controlam em que ponto o receptor deve gerar uma interrupção quando o FIFO está ativo. 7 6 Quantas palavras são recebidas antes que uma interrupção seja gerada 0 0 1 0 1 4 1 0 8 1 1 14 Bit 5 → Reserved, always 0. Bit 4 → Reserved, always 0. Bit 3 → DMA Mode Select. Se o Bit 0 for ajustado para "1" (FIFOs habilitado), a configuração deste bit altera a operação dos sinais -RXRDY e -TXRDY do Modo 0 para o Modo 1. Bit 2 → Transmit FIFO Reset. Quando um "1" é gravado neste bit, o conteúdo do FIFO é descartado. Qualquer palavra atualmente sendo transmitida será enviada intacta. Esta função é útil para anular transferências. Bit 1 → Receiver FIFO Reset. Quando um "1" é gravado neste bit, o conteúdo do FIFO é descartado. Qualquer palavra atualmente montada no registrador de turno será recebida intacta. Bit 0 → 16550 FIFO Enable. Quando configurado, os FIFOs de transmissão e recepção estão ativados. Qualquer conteúdo no registro de espera, registradores de deslocamento ou FIFOs são perdidos quando as FIFOs são ativadas ou desativadas.

Porta I/O	Acesso permitido	Descrição
+0x02	read	Registro de identificação de interrupção Bit 7 → FIFOs habilitado. No 8250/16450 UART, esse bit é zero. Bit 6 → FIFOs habilitado. No 8250/16450 UART, esse bit é zero. Bit 5 → Reserved, always 0. Bit 4 → Reserved, always 0. Bit 3 → ID de Interrupção Bit #2. No 8250/16450 UART, esse bit é zero. Bit 2 → ID de Interrupção Bit #1 Bit 1 → ID de Interrupção Bit #0.Esses três bits se combinam para relatar a categoria de evento que causou a interrupção que está em andamento. Essas categorias têm prioridades, portanto, se várias categorias de eventos ocorrerem ao mesmo tempo, a UART relatará os eventos mais importantes primeiro e o host precisará resolver os eventos na ordem em que forem relatados. Todos os eventos que causaram a interrupção atual devem ser resolvidos antes que novas interrupções sejam geradas. (Esta é uma limitação da arquitetura do PC.) 2 1 0 Prioridade Descrição 0 1 1 Primeiro Received Error (OE, PE, BI, or FE) 0 1 0 Segundo Dados Recebidos Disponíveis 1 1 0 Segundo Identificação do nível de gatilho (dados obsoletos no buffer de recebimento) 0 0 1 Terceiro Transmissor tem espaço para mais palavras (THRE) 0 0 0 Quarto Alteração de status do modem (-CTS, -DSR, -RI ou -DCD) Bit 0 → Interromper Bit Pendente. Se este bit estiver definido como "0", pelo menos uma interrupção está pendente.

Porta I/O	Acesso permitido	Descrição
+0x03	write/read	Registro de Controle de Linha (LCR) Bit 7 → Divisor Latch Access Bit (DLAB). Quando configurado, o acesso ao registro de transmissão / recepção de dados (THR / RBR) e ao Registro de Ativação de Interrupção (ITA) é desabilitado. Qualquer acesso a essas portas é agora redirecionado para os Registradores de Latch do Divisor. Definir esse bit, carregar os Registradores do Divisor e limpar o DLAB deve ser feito com as interrupções desativadas. Bit 6 → Set Break. Quando definido para "1", o transmissor começa a transmitir espaçamento contínuo até que este bit seja definido como "0". Isso substitui todos os bits de caracteres que estão sendo transmitidos. Bit 5 → Paridade da varStick Parity. Quando a paridade está ativada, a configuração desse bit faz com que a paridade seja sempre "1" ou "0", com base no valor do Bit 4. Bit 4 → Even Parity Select (EPS). Quando a paridade é ativada e o bit 5 é "0", a configuração desse bit faz com que a paridade par seja transmitida e esperada. Caso contrário, a paridade ímpar é usada. Bit 3 → Parity Enable (PEN). Quando definido para "1", um bit de paridade é inserido entre o último bit dos dados e o bit de Stop. A UART também espera que a paridade esteja presente nos dados recebidos. Bit 2 → Number of Stop Bits (STB). Se definido como "1" e usando palavras de dados de 5 bits, 1.5 bits de parada são transmitidos e esperados em cada palavra de dados. Para palavras de dados de 6, 7 e 8 bits, 2 Stop Bits são transmitidos e esperados. Quando este bit é definido como "0", um bit de parada é usado em cada palavra de dados. Bit 1 → Comprimento de palavra selecione bit #1 (WLSB1) Bit 0 → Comprimento de palavra selecione bit #0 (WLSB0) Juntos, esses bits especificam o número de bits em cada palavra de dados. 1 0 Comprimento da palavra 0 0 5 bits de dados 0 1 6 bits de dados 1 0 7 bits de dados 1 1 8 bits de dados

Porta I/O	Acesso permitido	Descrição
+0x04	write/read	Registro de Controle de Modem (MCR) Bit 7 → Reserved, always 0. Bit 6 → Reserved, always 0. Bit 5 → Reserved, always 0. Bit 4 → Loop-Back Enable. Quando definido para "1", o transmissor e o receptor UART são conectados internamente para permitir operações de diagnóstico. Além disso, as saídas de controle do modem UART são conectadas às entradas de controle do modem UART. O CTS está conectado a RTS, o DTR está conectado a DSR, o OUT1 está conectado a RI e o OUT 2 está conectado a DCD. Bit 3 → OUT 2. É uma saída auxiliar que o processador host pode definir como alta ou baixa. No adaptador serial IBM PC (e na maioria dos clones), OUT 2 é usado para triestar (desabilitar) o sinal de interrupção do UART 8250/16450/16550. Bit 2 → OUT 1. É uma saída auxiliar que o processador host pode definir como alta ou baixa. Essa saída não é usada no adaptador serial IBM PC. Bit 1 → Solicitação para enviar (RTS). Quando definido para "1", a saída da linha UART-RTS é Baixa (Ativo). Bit 0 → Data Terminal Ready (DTR). Quando definido para "1", a saída da linha UART-DTR é baixa (ativa).

Porta I/O	Acesso permitido	Descrição
+0x05	write/read	Registro de status de linha (LSR) Bit 7 → Error in Receiver FIFO. No UART 8250/16450, esse bit é zero. Esse bit é definido como "1" quando qualquer um dos bytes no FIFO tem uma ou mais das seguintes condições de erro: PE, FE ou BI. Bit 6 → Transmitter Empty (TEMT). Quando definido para "1", não há palavras restantes no FIFO de transmissão ou no registrador de deslocamento de transmissão. O transmissor está completamente ocioso. Bit 5 → Transmissor Holding Register Empty (THRE). Quando definido para "1", o FIFO (ou registrador de retenção) agora tem espaço para pelo menos uma palavra adicional para transmitir. O transmissor ainda pode estar transmitindo quando este bit está definido para "1". Bit 4 → Break Interrupt (BI). O receptor detectou um sinal de Break. Bit 3 → Framing Error (FE). Um Bit de Início foi detectado, mas o Bit de Stop não apareceu no horário esperado. A palavra recebida é provavelmente truncada. Bit 2 → Parity Error (PE). O bit de paridade estava incorreto para a palavra recebida. Bit 1 → Overrun Error (OE). Uma nova palavra foi recebida e não havia espaço no buffer de recebimento. A palavra recém-chegada no registrador de deslocamento é descartada. Nos UARTs 8250/16450, a palavra no registro de retenção é descartada e a palavra recém-chegada é colocada no registro de retenção. Bit 0 → Data Ready (DR). Uma ou mais palavras estão no FIFO de recepção que o host pode ler. Uma palavra deve ser completamente recebida e movida do registrador de deslocamento para o FIFO (ou registrador de sustentação para desenhos do 8250/16450) antes que este bit seja definido.
+0x06	write/read	Registro de status de modem (MSR) Bit 7 → Data Carrier Detect (DCD). Reflete o estado da linha DCD no UART. Bit 6 → Indicador de anel (RI). Reflete o estado da linha RI no UART. Bit 5 → Conjunto de dados pronto (DSR). Reflete o estado da linha DSR no UART. Bit 4 → Limpar para enviar (CTS). Reflete o estado da linha CTS no UART. Bit 3 → Delta Data Carrier Detect (DDCD). Defina para "1" se a linha -DCD mudou de estado mais uma vez desde a última vez em que o MSR foi lido pelo host. Bit 2 → Trailing Edge Ring Indicator (TERI). Defina para "1" se a linha -RI teve uma transição baixa para alta desde a última vez em que o MSR foi lido pelo host. Bit 1 → Delta Data Set Ready (DDSR). Defina para "1" se a linha -DSR mudou de estado mais uma vez desde a última vez em que o MSR foi lido pelo host. Bit 0 → Delta Clear To Send (DCTS). Defina para "1" se a linha -CTS mudou de estado mais uma vez desde a última vez em que o MSR foi lido pelo host.

Porta I/O	Acesso permitido	Descrição
+0x07	write/read	Scratch Register (SCR). Este registro não executa nenhuma função no UART. Qualquer valor pode ser gravado pelo host para este local e lido pelo host mais tarde.

1.9. Além do UART 16550A

Embora a National Semiconductor não tenha oferecido nenhum componente compatível com o 16550 que forneça recursos adicionais, vários outros fornecedores oferecem. Alguns desses componentes são descritos abaixo. Deve ser entendido que para utilizar efetivamente essas melhorias, os drivers podem ter que ser fornecidos pelo fornecedor do chip, já que a maioria dos sistemas operacionais populares não suportam recursos além daqueles fornecidos pelo 16550.

ST16650

Por padrão, essa peça é semelhante ao NS16550A, mas um buffer de envio e recebimento de 32 bytes estendido pode ser opcionalmente ativado. Fabricado pela StarTech.

TIL16660

Por padrão, essa peça se comporta de maneira semelhante ao NS16550A, mas um buffer de envio e recebimento de 64 bytes estendido pode ser opcionalmente ativado. Fabricado pela Texas Instruments.

Hayes ESP

Esta placa plug-in proprietária contém um buffer de envio e recebimento de 2048 bytes e suporta taxas de dados de até 230,4 Kbit/s. Fabricada pela Hayes.

Além desses UARTs "dumb", muitos fornecedores produzem placas de comunicação serial inteligentes. Esse tipo de design geralmente fornece um microprocessador que faz interface com vários UARTs, processa e armazena os dados em buffer e, em seguida, alerta o processador principal do PC quando necessário. Como os UARTs não são acessados diretamente pelo processador do PC nesse tipo de sistema de comunicação, não é necessário que o fornecedor use UARTs compatíveis com o UART 8250, 16450 ou 16550. Isso deixa o designer livre para usar componentes que tenham melhores características de desempenho.

2. Configurando o driver sio

O driver sio fornece suporte para interfaces de comunicação EIA RS-232C (CCITT V.24) baseadas em NS8250-, NS16450-, NS16550 e NS16550A. Várias placas multiportas também são suportadas. Consulte a página de manual sio(4) para obter documentação técnica detalhada.

2.1. Digi International (DigiBoard) PC/8

Contribuição de Andrew Webster awebster@pubnix.net. 26 de agosto de 1995.

Aqui está um trecho de configuração de uma máquina com uma placa Digi International PC/8 com 16550. Ela tem 8 modems conectados a essas 8 linhas e eles funcionam muito bem. Não se esqueça de adicionar `options COM_MULTIPORT` ao seu kernel ou ela não funcionará muito bem!

```
device          sio4    at isa? port 0x100 flags 0xb05
device          sio5    at isa? port 0x108 flags 0xb05
device          sio6    at isa? port 0x110 flags 0xb05
device          sio7    at isa? port 0x118 flags 0xb05
device          sio8    at isa? port 0x120 flags 0xb05
device          sio9    at isa? port 0x128 flags 0xb05
device          sio10   at isa? port 0x130 flags 0xb05
device          sio11   at isa? port 0x138 flags 0xb05 irq 9
```

O truque para configurá-la é que o MSB dos flags representa a última porta SIO, neste caso 11, então as flags são 0xb05.

2.2. Boca 16

Contribuição de Don Whiteside whiteside@acm.org. 26 de agosto de 1995.

Os procedimentos para fazer uma placa multiporta Boca 16 funcionar com o FreeBSD são bastante diretos, mas você precisará de algumas coisas para fazê-la funcionar:

1. Você precisa do código fonte do kernel instalado para poder recompilar as opções necessárias ou precisará de alguém para compilá-las para você. O kernel padrão 2.0.5 *não* vem com suporte a múltiplas portas ativado e você precisará adicionar uma entrada de dispositivo para cada porta de qualquer maneira.
2. Dois, você precisará saber a configuração de interrupção e de I/O da sua placa Boca para que você possa definir essas opções corretamente no kernel.

Uma nota importante - os chips UART reais para a Boca 16 estão nos conectores, não na própria placa interna. Então, se você os tiver desconectado, os probes destas portas falharão. Eu nunca testei a inicialização com a caixa desconectada e conectando-a novamente, e sugiro que você também não o faça.

Se você ainda não tiver um arquivo de configuração de kernel personalizado, consulte o capitulo Configuração do Kernel no Handbook do FreeBSD para os procedimentos gerais. A seguir estão as especificações para a placa Boca 16 e supõe-se que você esteja usando o nome do kernel MYKERNEL e editando com o vi.

1. Adicione a linha

   ```
   options COM_MULTIPORT
   ```

   ao arquivo de configuração.

2. Onde as linhas atuais do dispositivo `device sion` estão, você precisará adicionar mais 16

dispositivos. O exemplo a seguir é para uma placa Boca com uma interrupção de 3 e um endereço de IO base de 100h. O endereço IO para cada porta é +8 hexadecimal da porta anterior, portanto, os endereços são 100h, 108h, 110h ....

```
device sio1 at isa? port 0x100 flags 0x1005
device sio2 at isa? port 0x108 flags 0x1005
device sio3 at isa? port 0x110 flags 0x1005
device sio4 at isa? port 0x118 flags 0x1005
...
device sio15 at isa? port 0x170 flags 0x1005
device sio16 at isa? port 0x178 flags 0x1005 irq 3
```

A entrada de flags *deve* ser alterada deste exemplo, a menos que você esteja usando exatamente as mesmas atribuições de sio. As sinalizações são definidas de acordo com 0x`MYY` onde *M* indica o número menor da porta principal (a última porta em uma Boca 16) e *YY* indica se o FIFO está ativado ou desativado (ativado), o compartilhamento de IRQ é usado (sim) e se há um registro de controle de IRQ compatível com AST/4 (não). Neste exemplo,

```
 flags
         0x1005
```

indica que a porta principal é a sio16. Se eu adicionasse outra placa e atribuísse do sio17 até sio28, os sinalizadores para todas as 16 portas *nesta* placa seriam 0x1C05, onde 1C indica o menor número da porta principal. Não altere a configuração 05.

3. Salve e complete a configuração do kernel, recompile, instale e reinicialize. Presumindo que você tenha instalado com sucesso o kernel recompilado e configurado para o endereço e IRQ correto, sua mensagem de boot deve indicar o teste bem-sucedido das portas Boca da seguinte forma: (obviamente os números sio, IO e IRQ podem ser diferentes)

```
sio1 at 0x100-0x107 flags 0x1005 on isa
sio1: type 16550A (multiport)
sio2 at 0x108-0x10f flags 0x1005 on isa
sio2: type 16550A (multiport)
sio3 at 0x110-0x117 flags 0x1005 on isa
sio3: type 16550A (multiport)
sio4 at 0x118-0x11f flags 0x1005 on isa
sio4: type 16550A (multiport)
sio5 at 0x120-0x127 flags 0x1005 on isa
sio5: type 16550A (multiport)
sio6 at 0x128-0x12f flags 0x1005 on isa
sio6: type 16550A (multiport)
sio7 at 0x130-0x137 flags 0x1005 on isa
sio7: type 16550A (multiport)
sio8 at 0x138-0x13f flags 0x1005 on isa
sio8: type 16550A (multiport)
sio9 at 0x140-0x147 flags 0x1005 on isa
```

```
sio9: type 16550A (multiport)
sio10 at 0x148-0x14f flags 0x1005 on isa
sio10: type 16550A (multiport)
sio11 at 0x150-0x157 flags 0x1005 on isa
sio11: type 16550A (multiport)
sio12 at 0x158-0x15f flags 0x1005 on isa
sio12: type 16550A (multiport)
sio13 at 0x160-0x167 flags 0x1005 on isa
sio13: type 16550A (multiport)
sio14 at 0x168-0x16f flags 0x1005 on isa
sio14: type 16550A (multiport)
sio15 at 0x170-0x177 flags 0x1005 on isa
sio15: type 16550A (multiport)
sio16 at 0x178-0x17f irq 3 flags 0x1005 on isa
sio16: type 16550A (multiport master)
```

Se as mensagens forem muito rápidas para serem visualizadas,

```
# dmesg | more
```

mostrará as mensagens de inicialização.

4. Em seguida, as entradas apropriadas em /dev para os dispositivos devem ser criadas usando o script /dev/MAKEDEV. Esta etapa pode ser omitida se você estiver executando o FreeBSD 5.X com um kernel que tenha sido compilado com o suporte ao devfs(5).

   Se você precisar criar as entradas /dev, execute o seguinte como `root`:

   ```
   # cd /dev
   # ./MAKEDEV tty1
   # ./MAKEDEV cua1
   (everything in between)
   # ./MAKEDEV ttyg
   # ./MAKEDEV cuag
   ```

   Se você não quiser ou precisar de dispositivos de chamada por algum motivo, você pode dispensar o uso dos dispositivos cua*.

5. Se você quiser uma maneira rápida e desleixada de se certificar de que os dispositivos estão funcionando, você pode simplesmente conectar um modem em cada porta e (como root)

   ```
   # echo at > ttyd*
   ```

   para cada dispositivo que você fez. Você *deve* ver as luzes RX piscando para cada porta em funcionamento.

2.3. Suporte para cartões Multi-UART baratos

Contribuição de Helge Oldach hmo@sep.hamburg.com, setembro de 1999

Já se perguntou se o FreeBSD suporta a sua placa multi-I/O de US$ 20 com duas (ou mais) portas COM, compartilhando IRQs? Aqui está como:

Normalmente, a única opção para suportar esse tipo de placa é usar um IRQ distinto para cada porta. Por exemplo, se a placa da CPU tiver uma porta COM1 integrada (também conhecida como sio0 - endereço de I/O 0x3F8 e IRQ 4) e você tiver uma placa de extensão com dois UARTs , você normalmente precisará configurá-los como COM2 (também conhecido como sio1 - endereço de I/O 0x2F8 e IRQ 3) e a terceira porta (também conhecida como sio2) como I/O 0x3E8 e IRQ 5. Obviamente, isso é um desperdício de recursos de IRQ, já que deve ser basicamente possível executar ambas as portas da placa de extensão usando um único IRQ com a configuração `COM_MULTIPORT` descrita nas seções anteriores.

Essas placas de I/O baratas geralmente têm uma matriz de jumpers de 4 por 3 para as portas COM, semelhante à seguinte:

```
          o  o  o  *
Port A                |
          o  *  o  *
Port B      |
          o  *  o  o
IRQ       2  3  4  5
```

É mostrada aqui a porta A com fiação para IRQ 5 e a porta B com fiação para IRQ 3. As colunas de IRQ em sua placa específica podem variar - outras placas podem fornecer jumpers para IRQs 3, 4, 5 e 7.

Pode-se concluir que a fiação de ambas as portas para o IRQ 3 usando um jumper feito a mão e feito à mão cobrindo todos os três pontos de conexão na coluna IRQ 3 resolveria o problema, mas não. Você não pode duplicar o IRQ 3 porque os drivers de saída de cada UART estão conectados de forma "totem pole", portanto, se um dos UARTs ativar o IRQ 3, o sinal de saída não será o esperado. Dependendo da implementação da placa de extensão ou da placa-mãe, a linha IRQ 3 permanecerá sempre ativa ou sempre baixa.

Você precisa separar os drivers de IRQ para as duas UARTs, de modo que a linha IRQ da placa só suba se (e somente se) uma das UARTs ativar uma IRQ e permanecendo abaixo de outra forma. A solução foi proposta por Joerg Wunsch j@ida.interface-business.de: Soldar um cabo - ou consistindo de dois diodos (de Germânio ou do tipo-Schottky são fortemente preferidos) e um resistor de 1 kOhm. Aqui está o esquema, a partir do campo de jumper 4 por 3 acima:

```
                        Diode
               +---------->|-------+
              /                    |
          o  *  o  o               |     1 kOhm
Port A                             +----|######|-------+
```

```
           o  *  o  o             |                  |
Port B          `-----------------+                ==+==
           o  *  o  o             |                Ground
               \                  |
                +--------->|-------+
IRQ        2  3  4  5    Diode
```

Os cátodos dos diodos estão conectados a um ponto comum, junto com um resistor de 1 kOhm. É essencial conectar o resistor ao terra para evitar a flutuação da linha IRQ no barramento.

Agora estamos prontos para configurar um kernel. Ficando com este exemplo, nós configuraríamos:

```
# standard on-board COM1 port
device          sio0    at isa? port "IO_COM1" flags 0x10
# patched-up multi-I/O extension board
options         COM_MULTIPORT
device          sio1    at isa? port "IO_COM2" flags 0x205
device          sio2    at isa? port "IO_COM3" flags 0x205 irq 3
```

Note que a configuração das `flags` para sio1 e sio2 é realmente essencial; consulte sio(4) para detalhes. (Geralmente, o `2` no atributo "flags" refere-se ao sio2 que contém o IRQ, e você certamente deseja um "nibble" abaixo de `5`.) Com o modo verboso do kernel ativado, isso deve render algo semelhante a isto:

```
sio0: irq maps: 0x1 0x11 0x1 0x1
sio0 at 0x3f8-0x3ff irq 4 flags 0x10 on isa
sio0: type 16550A
sio1: irq maps: 0x1 0x9 0x1 0x1
sio1 at 0x2f8-0x2ff flags 0x205 on isa
sio1: type 16550A (multiport)
sio2: irq maps: 0x1 0x9 0x1 0x1
sio2 at 0x3e8-0x3ef irq 3 flags 0x205 on isa
sio2: type 16550A (multiport master)
```

Embora o /sys/i386/isa/sio.c seja um pouco enigmático com o uso do array "irq maps" acima, a ideia básica é que você observe `0x1` no primeiro, terceiro e quarto lugar. Isso significa que o IRQ correspondente foi definido na saída e limpo depois, o que é exatamente o que esperaríamos. Se o seu kernel não exibir esse comportamento, provavelmente há algo errado com a sua fiação.

3. Configurando o driver cy

Contribuição de Alex Nash. 6 de Junho de 1996.

As placas multiseriais da Cyclades são baseadas no driver cy em vez do driver usual sio usado por outras placas multiseriais. Configuração é uma simples questão de:

1. Adicione o dispositivo cy à sua configuração do kernel (observe que suas configurações irq e iomem podem ser diferentes).

```
device cy0 at isa? irq 10 iomem 0xd4000 iosiz 0x2000
```

2. Recompile e instale o novo kernel.
3. Crie os device nodes digitando (o exemplo a seguir assume uma placa de 8 portas) [1]:

```
# cd /dev
# for i in 0 1 2 3 4 5 6 7;do ./MAKEDEV cuac$i ttyc$i;done
```

4. Se apropriado, adicione entradas de discagem ao /etc/ttys duplicando as entradas dos dispositivos seriais (`ttyd`) e usando `ttyc` no lugar de `ttyd`. Por exemplo:

```
ttyc0   "/usr/libexec/getty std.38400"  unknown on insecure
ttyc1   "/usr/libexec/getty std.38400"  unknown on insecure
ttyc2   "/usr/libexec/getty std.38400"  unknown on insecure
...
ttyc7   "/usr/libexec/getty std.38400"  unknown on insecure
```

5. Reinicie com o novo kernel.

4. Configurando o driver si

Contribuição de Nick Sayer nsayer@FreeBSD.org. 25 de Março de 1998.

As placas multiportas Specialix SI/XIO e SX usam o driver si. Uma única máquina pode ter até 4 placas host. As seguintes placas host são suportadas:

- ISA SI/XIO host card (2 versions)
- EISA SI/XIO host card
- PCI SI/XIO host card
- ISA SX host card
- PCI SX host card

Embora as placas host SX e SI/XIO pareçam marcadamente diferentes, sua funcionalidade é basicamente a mesma. Os cartões de host não usam locais de I/O, mas exigem um bloco de memória de 32K. A configuração de fábrica para cartões ISA coloca isso em `0xd0000-0xd7fff`. Elas também exigem um IRQ. As placas PCI, é claro, se configuram automaticamente.

Você pode anexar até 4 módulos externos a cada placa de host. Os módulos externos contêm 4 ou 8 portas seriais. Eles vêm nas seguintes variedades:

- Módulos de portas SI 4 ou 8. Até 57600 bps em cada porta suportada.

- Módulos de porta XIO 8. Até 115.200 bps em cada porta suportada. Um tipo de módulo XIO possui 7 portas seriais e 1 porta paralela.
- Módulos de porta SXDC 8. Até 921.600 bps em cada porta suportada. Tal como no XIO, um módulo está disponível com uma porta paralela também.

Para configurar uma placa de host ISA, adicione a seguinte linha ao seu arquivo de configuração do kernel, alterando os números conforme apropriado:

```
device si0 at isa? iomem 0xd0000 irq 11
```

Números de IRQ válidos são 9, 10, 11, 12 e 15 para placas host SX ISA e 11, 12 e 15 para placas host ISA/XIO ISA.

Para configurar uma placa de host EISA ou PCI, use esta linha:

```
device si0
```

Depois de adicionar a entrada de configuração, recompile e instale seu novo kernel.

A etapa seguinte, não é necessária se você estiver usando o devfs(5) no FreeBSD 5.*X*.

Após a reinicialização com o novo kernel, você precisa criar os device nodes no /dev. O script MAKEDEV cuidará disso para você. Conte quantas portas totais você tem e digite:

```
# cd /dev
# ./MAKEDEV ttyAnn cuaAnn
```

(no qual *nn* é o número de portas)

Se você quiser que as solicitações de login apareçam nessas portas, você precisará adicionar linhas como esta para /etc/ttys:

```
ttyA01  "/usr/libexec/getty std.9600"   vt100   on insecure
```

Altere o tipo de terminal conforme apropriado. Para modems, `dialup` ou `unknown` está bem.

[1] Você pode omitir esta parte se você estiver executando o FreeBSD 5.X com devfs5.