Comunicação Serial MIPS e Periférico

Felipe Pfeifer Rubin
Ariel Ril

[1]Faculdade de Informática – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

felipe.rubin;ariel.ril@acad.pucrs.br

Resumo. *O objetivo deste trabalho é de realizar a comunicação de um processador MIPS com um periférico através da interface serial.*

1. Introdução

A proposta do trabalho está em cima da utilização de uma interface serial como meio de comunicação entre processador e periférico. Para que tal comunicação seja possível, é necessário que a interface serial, através de autobaud, detecte o clock do periférico para que seja possível enviar e receber dados deste. Além disso, é necessário uma lógica de cola que mapeie certos endereços de memória do processador para o periférico. O controle da comunicação é realizado através de um software em código de máquina que também deve ser produzido. Neste relatório serão descritos as implementações do hardware e software dessa aplicação.

2. Hardware

A implementação de hardware tem como objetivo a criação de um componente de lógica de cola. A lógica de cola é responsável por mapear certas posições da memória em ações com o periférico. Os endereços escolhidos não são utilizados por nenhuma memória do MIPS (dados ou de instruções) e são descritas a seguir.

1. 0x10008000 : É utilizado para o tx_data, quando houver uma instrução de lbu (load byte unsigned) a lógica de cola deve colocar no barramento de dados o valor correspondente ao dado que o periférico enviou ao processador.
2. 0x10008001 : Indica se existe um dado disponível no barramento tx_data. Quando houver uma instrução lbu, coloca-se no barramento de dados 1 , se há um dado ou 0 se não há.
3. 0x10008002 : Utilizado para enviar um dado ao periférico. Para uso deste é necessário um registrador que armazene tal dado até que a operação de envio seja inciada.
4. 0x10008003 : Indica se existe um dado disponível em rx_data. Quando este recebe 1 , a operação de envio de dados ao periférico inicia.
5. 0x10008004 : Indica se a interface serial está ocupada ou não através de rx_busy realizando um processo de envio de dados ao periférico. Fica em 1 desde rx_start até que termine de enviar.

A implementação da lógica de cola foi feita através de duas máquinas de estados, uma é responsável pela operação de envio de dados ao periférico, enquanto outra é responsável por receber dados do periférico.

3. Software

Inicialmente, como será melhor descrito na seção à respeito do periférico, espera-se um período de tempo necessário para que a interface serial inicie sua comunicação com o periférico.Tal espera é feita através de um laço de repetição. À respeito do software, pode-se dividi-lo em duas partes sendo uma as rotinas de acionamento do periférico (envio e recebimento de dados) e outra a aplicação.

3.1. Rotinas

A comunicação entre periférico e processador possui duas rotinas, que foram implementadas na forma de funções seguindo os padrões de implementação da linguagem de montagem do MIPS, tais como utilização de registradores de argumento e de retorno de função. Para utilizar a rotina de envio de dados, primeiro é necessário armazenar um dado no registrador de argumento a0. Esta rotina possui três passos:

1. Verificar se a interface serial está ocupada através de lbu rx_busy, se estiver tenta novamente, senão pode continuar para o próximo passo.
2. Armazenar o dado provido pelo registrador de argumento através de sb rx_data que deseja-se enviar ao periférico.
3. Armazenar o valor 1 através de sb rx_start para que o envio do dado do passo 2 seja inciado.

A rotina de recebimento de dados do periférico possui dois passos:

1. Verificar se a interface serial já está pronta para mandar o dado através de lbu tx_av, se estiver pode continuar para o próximo passo, senão tenta novamente.
2. Ler o valor enviado do periférico através de lbu tx_data.

Após ler o valor presente em tx_data, este valor é armazenado no registrador de retorno de função e volta-se para a aplicação.

3.2. Aplicação

A aplicação modelada para este trabalho é simples, os passos desta são descritos a seguir.

1. Carregar dado 1 da memória de dados.
2. Enviar dado 1 como argumento para a rotina de envio de dados.
3. Carregar dado 2 da memória de dados.
4. Enviar dado 2 como argumento para a rotina de envio de dados.
5. Realizar a rotina de recebimento de dados.
6. Salvar o valor do registrador de retorno de função na memória de dados.

Em outras palavras, a aplicação é:

$$C = A + B \tag{1}$$

4. Periférico

O periférico implementado no testbench do sistema é uma máquina de estados simples que segue os princípios da aplicação descrita no software. Recebe dois dados, soma-os e devolve o resultado ao processador. Um ponto importante a considerar é que a proposta do periférico era que este tivesse uma frequência significativamente menor que a do processador. Tal frequencia deveria ser entre 1200bps (bits por segundo) até 115200 bps. Optou-se

pela frequência mais alta (115200 bps) e calculou-se que para a transferência de cada bit é necessário um período de $8,64us$. Com propósito de simplificar a implementação do periférico, decidiu-se utilizar tal período como o ciclo (clock) do mesmo. A comunicação entre o periférico e a interface serial requer que o periférico envie o dado $0x55$ para a interface serial de modo que esta consiga calcular a velocidade com que o periférico envia dados. Sobre o envio, há duas peculiaridades que devem ser consideradas. Neste modelo de comunicação, embora os dados sejam de um byte, enviam-se 2 bits extras que representam o inicio de um envio(bit 1) e o fim de um envio(bit 0). Outra ponto importante é que o envio dos dados é do bit menos significativo ao mais significativo.

5. Simulação

Nesta seção serão demonstradas algumas situações relevantes da simulação do sistema através de suas formas de onda.

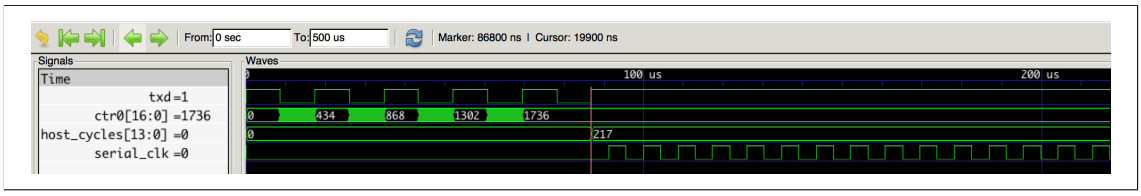

Figura 1. Periférico envia 0x55 para a lógica de cola.

Na situação da Figura 1 o periférico envia o 0x55 para a lógica de cola para que esta possa calcular sua velocidade de transmissão. Observe o valor de host_cycles, o meio ciclo do Periférico. Podemos verificar que o valor está correto.

$$1\ bit\ do\ Periférico = 8,68us = 8680ns \tag{2}$$

$$1\ ciclo\ do\ MIPS = 20ns \tag{3}$$

$$8680ns/20ns = 434\ ciclos\ do\ MIPS \tag{4}$$

$$Meio\ ciclo\ do\ Periférico = 217\ ciclos\ do\ MIPS \tag{5}$$

Logo, conclui-se que a interface serial teve sucesso em calcular a velocidade de transmissão do Periférico.

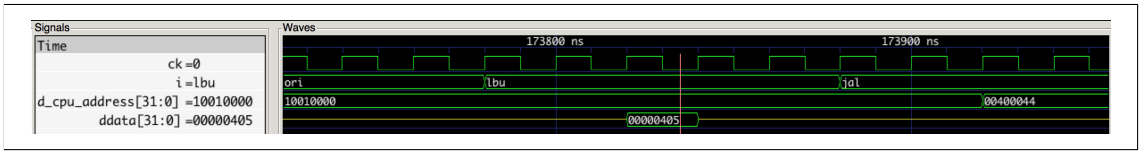

Figura 2. Obtêm-se o primeiro dado da memória de dados.

A aplicação consistem em ler dois dados de 1 byte armazenados na memória de dados, enviá-los ao Periférico, receber o resultado de sua soma e armazená-la na memória de dados. O primeiro passo pode ser visto na Figura 2 em que o primeiro dado é lido.

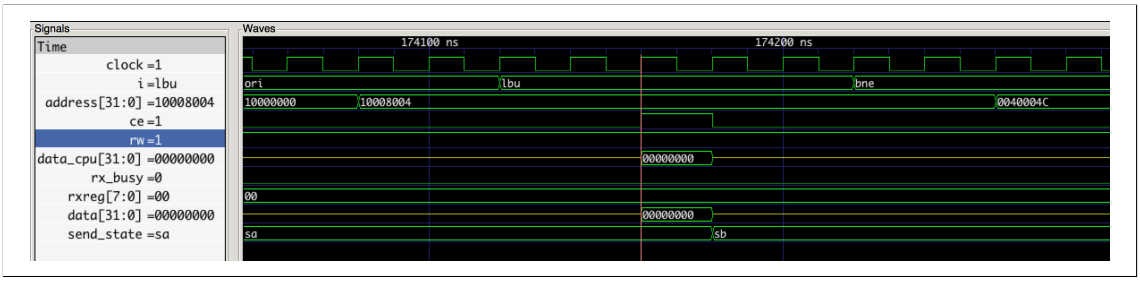

Figura 3. Verificar se a interface serial está ocupada (rx_busy = 1).

Na Figura 3 percebe-se que a interface serial não está ocupada, logo podemos enviar um dado ao periférico.

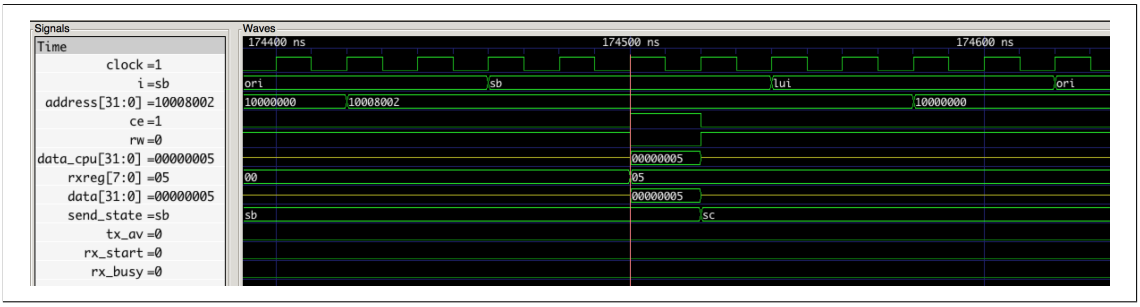

Figura 4. Escreve-se (sb) em rx_data o valor 0x5.

O segundo passo de envio é visto na Figura 4 a lógica de cola salva o dado a ser enviado ao Periférico em um registrador $rxreg$.

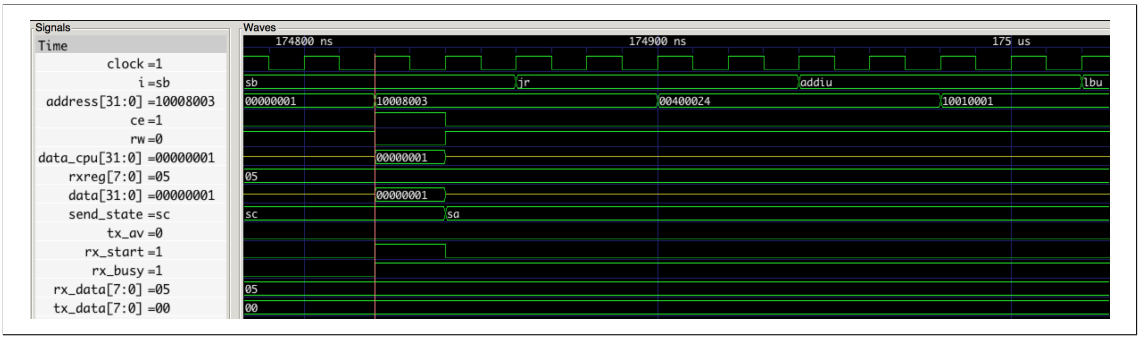

Figura 5. Escreve-se (sb) em rx_start para inciar o envio ao Periférico.

Finalmente, como visto na Figura 5, é possível iniciar a transmissão do dado ao periférico. Perceba que a interface serial muda de estado para ocupada.

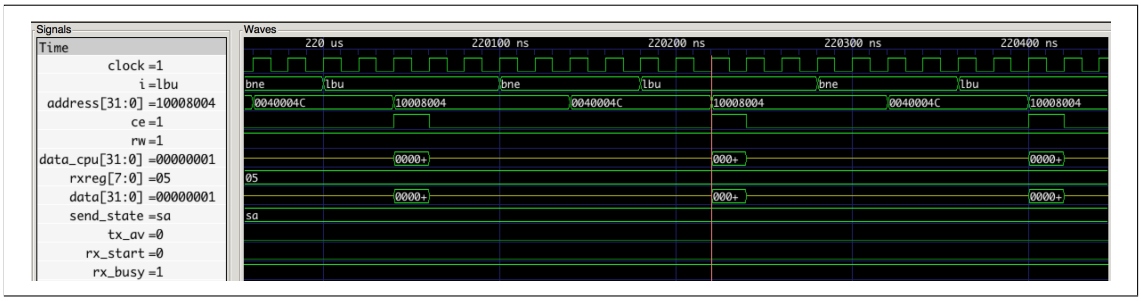

Figura 6. Deseja-se mandar o segundo dado ao Periférico, mas a interface serial está ocupada.

Falta enviar o segundo dado, porém a interface serial na Figura 6 está ocupada enviando o primeiro dado ao periférico.

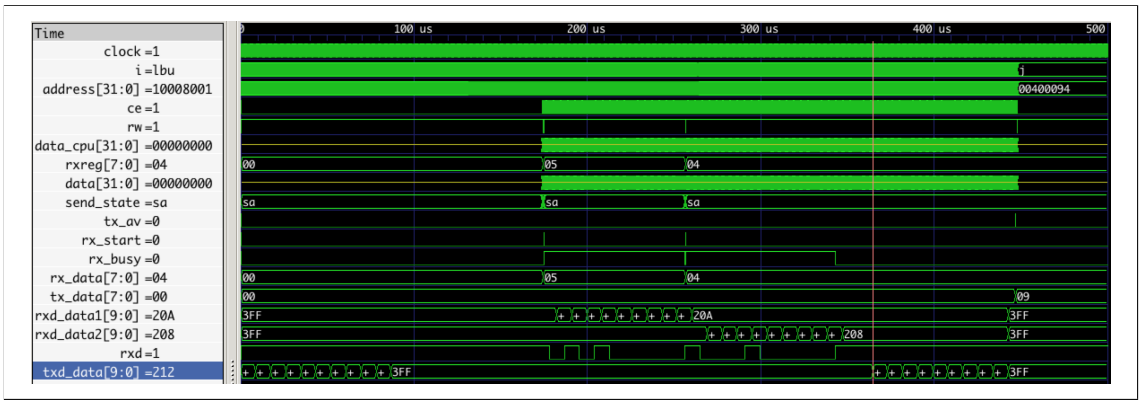

Figura 7. Envia-se o 0x5 e 0x4 para o Periférico que por sua vez os soma.

Na Figura 7, verifica-se que após terminar de enviar o primeiro dado ao Periférico, lê-se o segundo dado (0x4) e repete-se os mesmos passos para o segundo dado. Observe que, embora os dados enviados ao Periférico sejam $0x5$ e $0x4$, o dado 1 e 2, salvos respectivamente nos registradores rxd_data1 e rxd_data2 apresentam valores diferentes. O motivo é que esses registradores armazenam todos os dados que foram recebidos da interface serial, logo eles possuem 2 bits a mais (start e stop bit), portanto usaremos apenas os 8 bits (8 downto 1) que realmente armazenam os dados devidos. Por fim, observa-se que o registrador tx_data recebe a soma dos dados e, do mesmo modo, possui start e stop bit também armazenado.

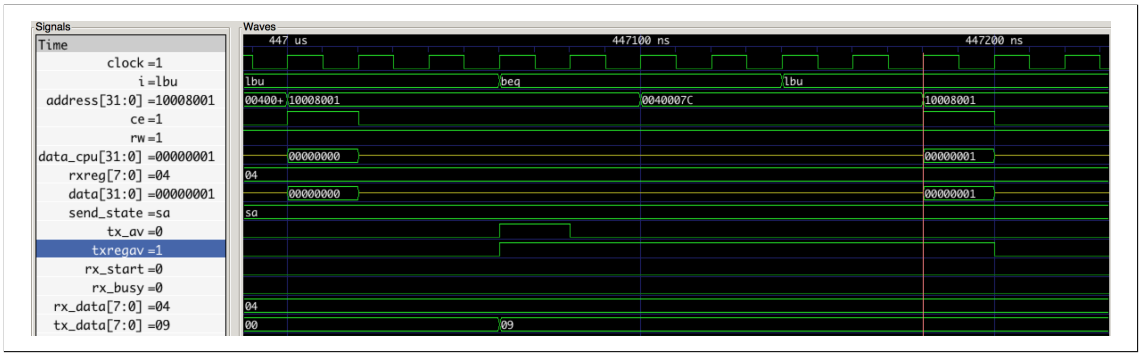

Figura 8. Deseja-se verificar se o Periférico já está pronto para devolver o resultado (tx_av = 1).

Na situação da Figura 8, percebe-se que houve duas leituras de tx_av, a primeira retornou que ainda não há dados enviados do Periférico ao MIPS enquanto a segunda confirmou que há um dado a ser lido como visto em tx_data. Note também que é necessário utilizar um registrador (txregav)que armazena o aviso de tx_av sobre a presença de dados em tx_data, tal necessidade vem do fato que o aviso é recebido em um ciclo e este pode ser um ciclo que não está ocorrendo a sua leitura.

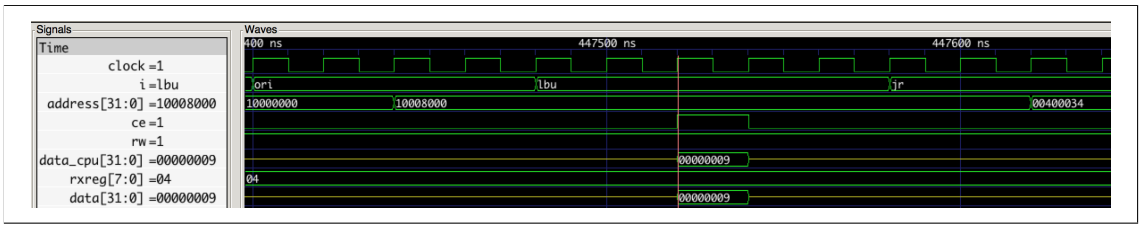

Figura 9. O dado enviado pelo Periférico é lido (lbu tx_data).

Após confirmado que há um dado a ser lido em tx_data, na Figura 9 tal dado é lido.

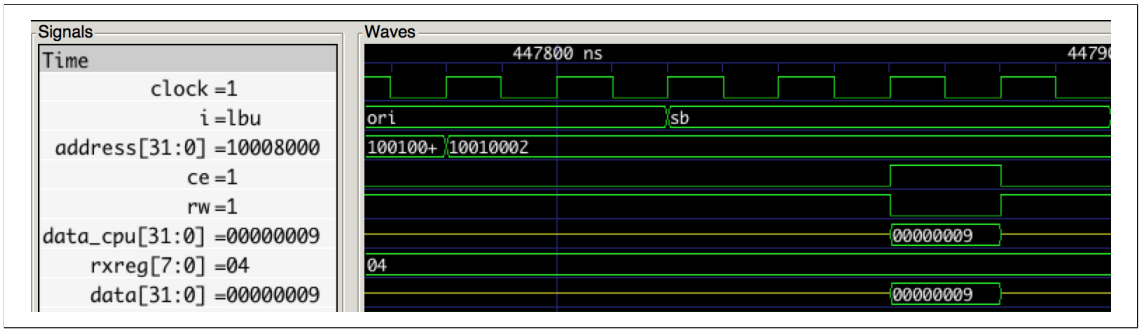

Figura 10. Salva-se o resultado recebido do Periférico na memória de dados.

No fim da aplicação, como visto na Figura 10, o dado resultante da operação de soma realizada pelo Periférico é armazenado na memória de dados do MIPS.

6. Conclusão

O propósito deste trabalho era de realizar uma conexão entre o MIPS e um periférico, o qual foi concluído. Houveram certas dificuldades que geraram atrasos na conclusão, como por exemplo um problema em relação aos ciclos delta, que após depuração observou-se que sua causa estava em sinais presentes em listas de processos da lógica de cola que atualizavam um ao outro indefinitivamente. Conclui-se que o maior desafio deste trabalho não estava implementação, mas no planejamento. Em vhdl, embora existam ambientes como o $Modelsim$ ou ISE, se não houver um planejamento prévio sobre como será realizada a implementação, um pequeno em um sinal do sistema pode levar dias para ser resolvido.

7. Bibliografia

David A. Patterson; John L. Hennessy; Organização e Projeto de Computadores; Elsevier 4a edição