Documentação da Implementação de uma Comunicação Serial Periférico-Processador

Vinicius Cerutti e Yuri Bittencourt

25 de setembro de 2017

1 Introdução

A implementação teve como base dois projetos desenvolvidos pelos professores da disciplina de Organização e Arquitetura de Computadores II 2017/2 da Faculdade de informática da PUCRS. Onde um destes projetos era a organização de um processador multiciclo com seu *testbeanch* e uma interface de comunicação serial baseada no padrão RS-232 e também com seu *testbeanch*. Como processador foi utilizado MIPS_S, que possui a seguinte documentação "Processador Multiciclo - MIPS_S".

Para o desenvolvimento da comunicação serial Periférico-Processador foi desenvolvido um hardware (Lógica de cola), um software e um *testbench*, conforme a figura a seguir (Figura 1).

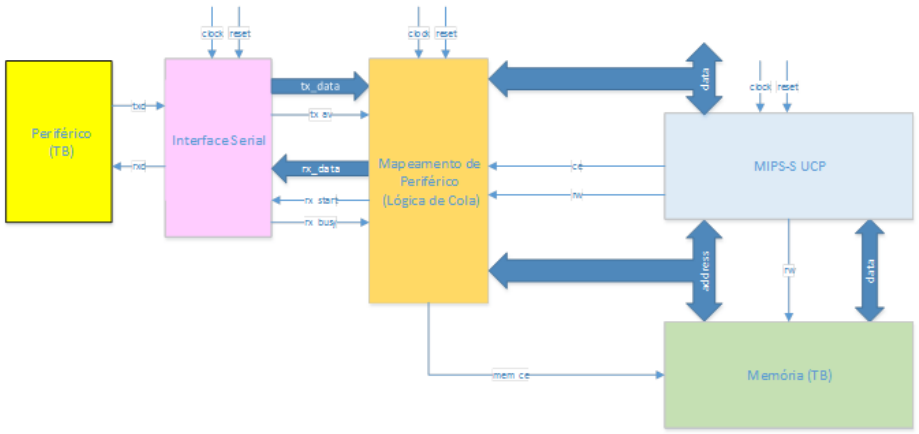

Figura 1: Estrutura geral da organização

2 Lógica de Cola

A lógica de cola realiza o mapeamento dos diversos componentes do ambiente de entrada do periférico em endereços do mapa de Memória do MIPS. Para isto, foi utilizado os seguintes endereços:

```
0x1008000 - tx_data
0x1008001 - tx_av
0x1008002 - rx_data
0x1008003 - rx_start
0x1008004 - rx_busy
```

Estes endereços foram escolhidos, pois são endereços que não relacionam aos endereçamentos do *.data* do MIPS ou das instruções do programa. A lógica de Cola possui o formato de uma máquina de estados onde cada estado é uma sequência ou consequência do programa (software). Assim a lógica de cola é apenas uma máquina de estados, onde poderia ser feita também em duas máquinas já que uma recebe e outra envia dados, porém foi escolhida uma máquina de estados apenas por fins mais didáticos e garantir a ordem do programa.

Desta forma para receber um dado da CPU para interface serial a máquina de estados realiza as seguintes etapas(Figura 2):

1. Espera no estado inicial da máquina o endereço de `rx_busy` e no modo de leitura, desta forma, a máquina de estados realiza a consulta do fio de rx_busy (que vem da interface serial) e coloca como data para a CPU.

2. Se `rx_busy` for igual a 0 então pode-se ir para o próximo estado que é escrita no endereço `rx_data`, caso o `rx_busy` for igual a 1, então a FSM continua no estado inicial.

3. Com a escrita no endereço de `rx_data` ativa-se, o sinal de `rx_start` e obtêm-se o dado transmitido da CPU para lógica de cola e a máquina de estados vai para o próximo estado.

4. Neste estado verifica-se se o dado recebido da CPU foi enviado com sucesso para a interface serial, para isso, necessita que o `rx_busy` esteja em 0, as instruções ficam presas neste estado até que `rx_busy` esteja livre. Estando livre pode-se voltar para o estado inicial da máquina onde pode-se realizar um envido de dado ou recebimento de dado.

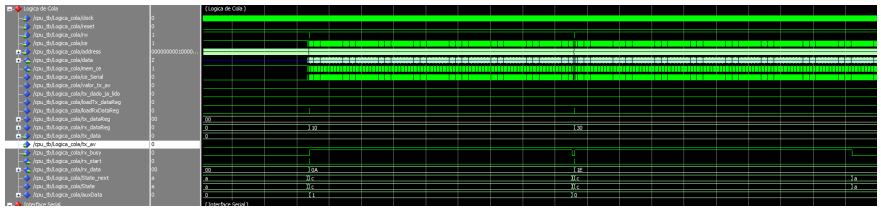

Figura 2: Exemplo de envio de Dados CPU para interface Serial

Assim para o recebimento de dados, ou seja, interface serial para CPU, a máquina de estados realiza as seguintes etapas (Figura 3):

1. Espera no estado inicial da máquina o endereço de `tx_av` e no modo de leitura, desta forma, a máquina de estados realiza a consulta do *signal* interno de tx_av e coloca como data para a CPU.

2. Se o fio interno `tx_av` for igual a 1 então, carregada o dado a ser entregue para a CPU e passa a ir para o próximo estado que é leitura no endereço `tx_data`, caso o *signal* `tx_av` for igual a 0, então a FSM continua no estado inicial.

3. Com a leitura no endereço de `tx_data` transfere-se o dado do registrador auxiliar para a data do CPU e marca como dado já lido, e retorna para o estado inicial.

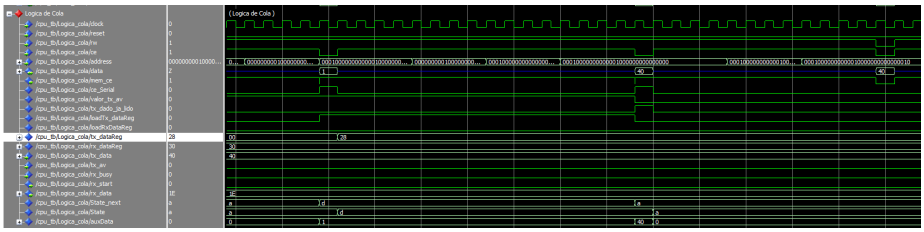

Figura 3: Recebimento de dados da interface Serial para o periférico

Como estrutura auxiliar foi utilizado dois registradores para armazenar as informações de `rx_data` e `tx_data` quando solicitados, pois como o dado da CPU é subscrito a cada instrução estes valores seriam perdidos e assim não sendo possível realizar a envio e recebimento de dados.

Além disso, o *signal* de tx_av é ativado em apenas um *clock*, ou seja necessitava guardar este valor até a instrução for uma leitura no `tx_av` e leitura do `tx_data`, para isso foi utilizado um registador que salva em 1 quando tx_av for um e é desativado quando aquele dado for transmitido para para a CPU, isto é, quando for realizada a leitura de `tx_data`.

Também foi utilizado uma lógica para desativar o a memória do MIPS enquanto o os endereços eram do periférico, isso não foi obrigatório fazer, porém foi feito assim por motivo de seguranças, outra coisa a se mencionar o *chip enable* (ce) da lógica de cola é o inverso do convencional, ou seja, trabalha com valor 0 no lugar de 1, foi escolhido trabalhar assim, pois a RAM da CPU trabalhava assim também.

3 Periférico

Para o periférico foi planejada a ideia de receber dois valores quaisquer e devolver a soma destes valores. Além disso, foi realizado testes de performance para descobrir a melhor velocidade para se utilizar e a escolha foi de se trabalhar com a maior velocidade possível que é 115.200 bits por segundo. Para realizar a sincronização, ou melhor, para a interface serial se adaptar com a velocidade transmissão di periférico é necessário que o periférico mande o valor de 0x55 que equivale (01010101 em binário) na sua velocidade.

Também valores recebidos e enviados do periférico começam com o bit de *start* bit e terminam com o bit de *stop* bit e na ordem binária ao contrária. O periférico funciona também no formato de uma máquina de estado única, além disso, possui o seu próprio *clock* que foi calculado da seguinte maneira:

$$\frac{1s \times 1b}{115200b} = \frac{10^6}{115200} = 8.68055ns = 8.68us$$

Além disso, uma estrutura auxiliar um *array* foi utilizado para armazenar os 8 bits de cada valor recebido, foi utilizado um *array*, pelo seguinte fator: com *array* é possível saber já quantos dados já foram utilizado e não é necessário declarar novos estados ou *signals* para salvar novos valores, basta incrementar o índice do *array* para receber um proximo número.

Assim como o periférico deseja realizar a soma de apenas dois valores, quando o índice do vetor for igual dois então que dizer que os dois valores já estão prontos para serem somados e enviados para a interface serial. Também foi utilizado contadores de bits para enviar e receber dados, assim obteve-se controle dos bits.

Para o envio do valor de sincronização (0x55) foi criado um estado especial para ele, pois são 11 bits de envio. Já para o recebimento de um bit (para formar um valor numérico) teve que seguir os seguintes passos na máquina de estados (Figura 4):

1. Se o valor do índice do vetor (que armazena os números para serem somados) for diferente de dois, então contador de bits de recebimento deve ser alterado para 8 e logo em seguida verificar se o dado que esta sendo recebido é o *start* bit, se for então deve ir para próximo estado, senão permanece no estado inicial.

2. No próximo estado realiza-se o recebimento dos outros bits até o contador de bits for maior do que 1, se o contador do bit for igual a 1 então deve-se salvar este bit e ir para próximo estado que é verificar se o próximo bit é o *stop* bit.

3. se o próximo bit for o *stop* então o índice do vetor é incrementado e volta-se para o estado inicial.

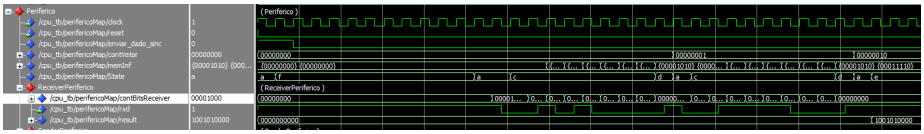

Figura 4: Periférico recebendo dois valores

Para o envio de dados (periférico para interface serial), possui a mesma lógica do recebimento de dados, porém com dois estados a menos, pois não necessita verificar o *start* e *stop* bit já que eles estão inclusos (Figura **??**):

1. No estado inicial se o valor do índices do vetor for igual a dois então significa que já possui dois valores para realizar a soma, assim, neste mesmo estado realiza-se a soma e arruma o *stop* bit. Também como deve ser enviado na ordem invertida o número, o contador de envio começa em zero.

2. No próximo estado o dado (a soma) é enviado bit a bit até o contador for menor ou igual a 8, quando for igual 9 (que é o ultimo bit), envia-se este

ultimo bit e zera o índice do vetor (assim consegue-se dizer que aqueles dados já foram processados) e volta para o estado inicial.

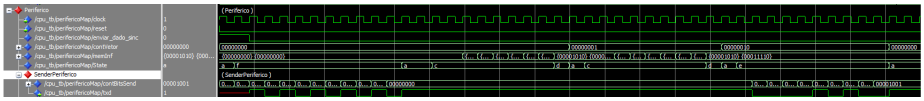

Figura 5: Envio do dado 0x55 e da soma dos outros valores

4 Programa

Este programa possui a finalidade de realizar a comunicação da CPU com o periférico, desta forma ele realiza o envio e obtenção de dados. As primeiras linhas de código do programa é o tempo aproximado para realizar a sincronização entre o periférico e a interface serial. Para descobrir o valor aproximado do laço foi realizado o seguinte cálculo:

$$\frac{8680ns \times 1ciclo}{20ns} = 434ciclos$$

onde $8680ns$ é a velocidade do periférico e $20ns$ é a realização de um ciclo do processador. Com isso se obtêm a quantidade de ciclos para o envio de 1 bit apenas, para enviar 10 bits seria necessário 4340 ciclos. Como cada instrução do primeiro loop dura aproximadamente 4 ciclos conforme a documentação do MIPS explicada na introdução (Capitulo 1) o valor final seria 362 vezes necessárias para fazer o loop, porém executando com este valor percebeu-se que necessitaria 108 repetições a mais para concluir o envio do dado 0x55.

Além disso, para enviar um dado do MIPS para a lógica de cola necessita esperar o sinal `rx_busy` estar em 0 ou seja a interface serial esta desocupada, para isso necessita realizar um laço verificando o endereço de `rx_busy` até for 0. Depois disso é possível salvar o `rx_data`, com uma comando de salvar.

Para ler o dado do periférico no processador através do programa, é da mesma maneira necessita, criar um laço com o valor de `tx_av` se este valor for igual a 1 então o periférico possui um dado. Assim o processador (programa) pode realizar uma leitura no endereço de `tx_data`.