Pontifícia Universidade Católica do
Rio Grande do Sul

Arquitetura de Computadores II

Trabalho II

Alunos	Diego Henrique Oliveira
	Ícaro Stumpf
	Leonardo Barbosa
Professor	Dr. Cesar Augusto Missio Marcon

Porto Alegre, Maio de 2021

Conteúdo

1 Introdução

Neste trabalho é requerido que seja feito um sistema computacional composto por dois conjuntos computacionais independentes. Cada conjunto contém uma CPU, memória e UART.

A funcionalidade final solicitada é que um sistema, denominado SisA, consiga enviar uma mensagem a outro sistema, SisB, e que após SisB confirmar o recebimento da mensagem, envie uma resposta ao sistema SisA.

Para que isso seja possível, foi modificada a descrição de hardware da CPU, provida pelo professor, e da UART (finalizada no trabalho 1). Além disso, foi também realizada a construção de programas em assembly afim de gerenciar a CPU para o envio e recebimento de mensagens.

2 Desenvolvimento

2.1 Sistemas

Para cada sistema, foram inseridas e modificadas uma série de itens, desde as ligações realizadas através de port map, até parâmetros recebidos através do testbench.

As modificações realizadas serão descritas nos tópicos abaixo.

2.1.1 Manipulação do sinal ack

O sinal ackTX é posto em estado lógico alto 15 nanossegundos após ser detectada a mudança do sinal irTX para nível lógico alto. Foi realizado desta maneira para que houvesse maior facilidade para a CPU detectar a modificação do sinal ackTX.

A descrição supracitada pode ser visualizada através do trecho de código abaixo:

```
ack: process(ck, rst)
        begin
                if rst = '1' then
                        ackTX <= '0';
                        ackRX <= '0';
                elsif ck'event and ck = '1' then
                        if irTX = '1' then
                                ackTX <= '1' after 15ns;
                        else ackTX <= '0';
                        end if;
                        if irRX = '1' then
                                ackRX <= '1' after 15ns;
                        else ackRX <= '0';
                        end if;
                end if;
        end process;
```

2.1.2 Parâmetros do Sistema

Para cada Sistema, foi inserida a necessidade de receber como parâmetro os sinais TX e RX. Com isso, é possível, no testbench, conectar o RX do Sistema A no TX do Sistema B, e o RX do Sistema B no TX do Sistema A.

O recebimento destes parâmetros em cada sistema pode ser visualizado no trecho de código abaixo:

```
entity sisA is
    port (
        RX: in std_logic;
        TX: out std_logic
    );
end SisA;
```

No exemplo acima foi utilizado o trecho de código do Sistema A, no entanto, é válido ressaltar que o trecho de código no Sistema B é equivalente.

Já no testbench, arquivo Tb.vhd, contém a seguinte descrição:

```
entity tb is
end tb;

architecture tb of tb is
        signal AparaB, BparaA: std_logic;
begin
        SA: entity work.sisA port map(TX => AparaB, RX => BparaA);
        SB: entity work.sisB port map(TX => BparaA, RX => AparaB);
end tb;
```

Essa descrição resulta na criação de dois fios, sendo estes chamados de "AparaB"e "BparaA", cuja função é realizar as ligações destes sistemas. Como dito em seu próprio nome, "AparaB"conecta o TX do Sistema A no RX do Sistema B, e, ao contrário deste, o sinal "BparaA"conecta o TX do Sistema B com o RX do Sistema A. A partir disso, será possível realizar a comunicação entre esses sistemas.

2.1.3 Execução dos programas

Cada um dos sistemas é vinculado ao seu devido programa. O programa "progSisASisB.txt"é responsável por realizar o envio da mensagem do Sistema A para o Sistema B. A instanciação do nome do arquivo para ser lido pelo Sistema A pode ser visualizado em VHDL através deste trecho de código:

```
-- Programa que manda a mensagem de sisA para sisB
    file ARQ: TEXT open READ_MODE is "progSisASisB.txt";
```

Concomitantemente a este, no arquivo VHDL do Sistema B é possível visualizar este trecho de código:

```
-- Programa que manda a mensagem de sisB para sisA
    file ARQ: TEXT open READ_MODE is "progSisBSisA.txt";
```

Posteriormente, será descrito o funcionamento destes programas.

2.1.4 Vínculo do Sistema com a CPU

Para cada sistema, inicialmente foram modificadas as ligações do port map para que fosse possível realizar a comunicação entre a CPU e a UART. As modificações podem ser visualizadas no trecho de código abaixo:

```
-- Port map dos subsistemas ----
------------------------------
CPU: Entity work.MR2 port map
    (clock => ck, reset => rstCPU, i_address => i_cpu_address,
    instruction => Idata, ce => ce, rw => rw, bw => bw,
    d_address => d_cpu_address, data => data_cpu,
        irTX => irTX, irRX => irRX);
```

2.1.5 Tratamento de interrupções

Cada um dos sistemas possui o mesmo tratamento de interrupções. De maneira pratica, são descritos sinais cuja responsabilidade é salvar o estado atual destes tratamentos, para assim, ser possível verificar se a interrupção solicitada pode ser tratada. Caso seja aceito o tratamento de uma interrupção, é salvo o endereço da instrução atual sendo executada (caso a instrução atual seja um salto incondicional) ou o endereço da próxima instrução a ser executada (caso a instrução atual não seja de salto incondicional). Esse algoritmo pode ser visto no seguinte trecho de código VHDL abaixo:

```
elsif (irTX = '1' or irRX = '1') and em_int = '0' and
    flagEndereco = '0' then -- Caso haja solicitacao de interrupcao
                            -- e nao ha uma ativa (em_int = 0)
    if uins.i=J or uins.i=JALR or uins.i=JR then
        pc_salvo <= pc; -- salva o endereco do pc no pc_salvo
    else
        pc_salvo <= incpc; -- salva o endereco do pc+4 no pc_salvo
    end if;
    flagEndereco <= '1'; -- ativa a flag de endereco para que o
                         -- valor de pc_salvo nao seja perdido
    flagEndereco <= '0' after 35ns; -- desativa a flag depois de 35ns
end if;
```

Como pode ser visto acima, foram criados os sinais em_int, pc_salvo e flagEndereco. Os dois primeiros, conforme especificação do trabalho, são usados para monitorar se uma interrupção qualquer está sendo tratada (evitando, assim, que haja aninhamento de interrupções) e para salvar o contexto atual

do program counter (PC), a depender da instrução executada. O terceiro sinal, flagEndereco, foi criado para auxiliar na atribuição do endereço correto no sinal pc_salvo nos seus devidos ciclos de relógio.

No trecho de código VHDL a seguir, é descrita a atribuição dos sinais em_int e de outros dois sinais, intEscrita e intLeitura. Os dois últimos sinais foram criados para suportar o empilhamento de interrupções a serem tratadas, caso já haja uma em tratamento.

```
if (irTX = '1' or irRX = '1') and em_int = '0' then
     em_int <= '1' after 25ns;
elsif (irTX = '1' and em_int = '1') then
     intEscrita <= '1';
elsif (irRX = '1' and em_int = '1') then
        intLeitura <= '1';
elsif uins.i = ERET then
       em_int <= '0';
end if;

if dtpc = x"00400004" and intEscrita = '1' then
   intEscrita <= '0' after 25ns;
end if;
if dtpc = x"00400008" and intLeitura = '1' then
     intLeitura <= '0' after 25ns;
end if;
```

O algoritmo funciona da seguinte forma:

É verificado se os sinais irTX ou irRX estão em nível lógico alto, ou seja, solicitando o tratamento de uma interrupção. Caso o sinal em_int esteja em '0', significa que a interrupção pode ser livremente tratada. Dessa forma, o sinal em_int é colocado em '1' e, através disso, o sistema é informado que um tratamento de interrupção está em andamento.

Caso a condição acima não seja satisfeita, é verificado se há tratamento de interrupção em andamento e se há solicitação de tratamento de interrupção de escrita na UART. Caso positivo, o sinal que empilha interrupções de escrita a serem tratadas posteriormente é colocado em nível lógico alto. É possível observar o mesmo comportamento na condicional seguinte, com a diferença desta ser para empilhamento de interrupções de leitura da UART.

A próxima condicional verifica se o tratamento de interrupção já foi finalizado (através da instrução ERET, entregue já implementada). Caso positivo, o sinal em_int volta ao seu estado original (nível lógico baixo, '0').

Por fim, nas últimas duas condicionais é verificado se o program counter (PC) está no endereço de tratamento de interrupções de entrada ou saída

e se os sinais que indicam empilhamento de interrupções a serem tratadas estão ativas (em '1'). Caso positivo, é sabido que as interrupções empilhadas já foram tratadas, e os sinais são colocados de volta ao estado original (nível lógico baixo, '0').

As atribuições no program counter (PC) foram feitas de forma a reutilizar o código-fonte oferecido, apenas adicionando as condicionais necessárias relativas ao tratamento de interrupções de entrada e saída da UART. O código escrito em VHDL pode ser visto a seguir:

```
dtpc <= pc_salvo when uins.i=ERET and intEscrita = '0' and intLeitura = '0'
    else x"00400004" when uins.i=ERET and intEscrita = '1'
    else x"00400008" when uins.i=ERET and intLeitura = '1'
    else result when (inst_branch='1' and salta='1') or uins.i=J or
                                          uins.i=JALR or uins.i=JR
    else npc;
```

As quatro condicionais exibidas no código acima englobam todos os possíveis cenários de atribuição do program counter (PC) no contexto de tratamento de interrupções. Na primeira condicional, o PC recebe o sinal pc_salvo quando uma interrupção terminou de ser tratada e não há outras interrupções empilhadas a serem tratadas. As duas condicionais seguintes atribuem o PC com o endereço de tratamento de interrupções de entrada ou saída, dado que o tratamento de uma interrupção tenha terminado e haja outra interrupção empilhada para tratamento. A última condicional, aparentemente, não parece tanger nenhum dos cenários de tratamento de interrupções, mas, na verdade, o sinal result pode possuir o endereço de tratamento de interrupções de entrada ou saída, a depender do estado lógico dos sinais inst_branch e salta.

Essas condicionais podem ser vistas no código VHDL abaixo:

```
inst_branch <= '1' when ... or ((irTX = '1' or irRX = '1')
                                          and em_int = '0')
        else '0';

salta <= '1' when ... or ((irTX = '1' or irRX = '1')
                                          and em_int = '0')
        else '0';

result <= ...
          else x"00400004" when irTX = '1' and em_int = '0'
          else x"00400008" when irRX = '1' and em_int = '0'
          else ...
```

Ou seja, essas condicionais tangem o caso primário: caso haja uma solicitação de tratamento de interrupção de entrada ou saída e não haja interrupção sendo tratada (sinal em_int igual a '0'), os sinais salta e inst_branch são setados para '1' (além de também haver outras condicionais já existentes que tangem instruções de salto do MIPS, ocultadas no trecho de código exibido acima). Além disso, o sinal result recebe o endereço referente ao tratamento de interrupção requisitado (entrada ou saída), dentre outras possibilidades que tangem outras instruções do MIPS (também ocultadas no trecho de código acima).

2.1.6 Vínculo do sistema com a UART

Semelhante ao item anterior, cada sistema contém, também, uma UART. O barramento de endereçamento de dados da UART é mapeado nos últimos 4 bits do barramento de endereçamento da memória de dados. A consequência disto é que a UART conseguirá detectar quando ela for requerida pelo sistema, função esta gerenciada pela execução do assembly.

Além disso, também efetua-se o mapeamento do barramento de dados da UART nos últimos 8 bits do barramento de dados da memória de dados, em função do protocolo P82. Essa alteração ocorre em função da necessidade de ler e escrever na memória de dados do sistema.

Este port map pode ser visualizado no trecho de código VHDL abaixo:

```
UART: entity work.UART port map
   (rst => rst, ck => ck, TX => TX, RX => RX,
    add => d_cpu_address(3 downto 0), data => data_cpu(7 downto 0), rw => rw,
    ce => ce(0), irTX => irTX, irRX => irRX,
    ackTX => ackTX, ackRX => ackRX);
```

2.2 UART

2.2.1 Área de dados

Dada a UART descrita no trabalho 1 desta disciplina, foi alterada a maneira com que esta lida com a memória. Foi incluso na lista de sensibilidade o sinal "ce"a fim de conseguir detectar o momento em que ocorre a modificação do sinal "ce"e a partir disso, salvar o valor do registrador na memória de dados.

O código cuja funcionalidade está descrita acima, pode ser visualizado no trecho descrito abaixo:

```
-- Inclusão do sinal CE
save_data: process(rst, ce, ck)
    begin
            if rst = '1' then
                    data <= (others=>'Z');
                    irRX <= '0';
            -- Verificação do estado lógico do sinal CE
            elsif ce = '1' and rw = '1' and add = x"8" then
                            data <= register_r;
                            data <= (others=>'Z') after 10ns;
            elsif ck'event and ck = '1' then
                    if receive_state = stop then
                            irRX <= '1';
                    end if;
                    if ackRX = '1' then
                            irRX <= '0';
                    end if;
            end if;
    end process;
```

Essas modificações foram efetuadas no arquivo UART.vhd, além disso, as modificações foram realizadas especificamente nas linhas abaixo dos comentários realizados.

2.3 Programas

Cada sistema efetua ações de acordo com a programação de um assembly especifico. Estes programas tem como intuito realizar a comunicação entre os sistemas.

Conforme especificação do trabalho, o Programa A envia uma mensagem ao Sistema B, que por sua vez, salva na memória a mensagem recebida e envia uma resposta ao Sistema A, através do Programa B. O Programa A salvará na memória a mensagem recebida, assim como o Programa B.

Memória de dados do Programa A:

```
.data
# Mensagem transmitida do 'SisA' pro 'SisB'
mensagemTransmitida:            .asciiz "Isto eh um teste!"
# Mensagem recebida pelo 'SisA' do 'SisB'
mensagemRecebida:               .asciiz "                    "

EnderecoUART:                           .word 0xFFE00000
```

Memória de dados do Programa B:

```
.data
# Mensagem transmitida do 'SisB' pro 'SisA'
mensagemTransmitida:            .asciiz "Teste compreendido!"
# Mensagem recebida pelo 'SisB' do 'SisA'
mensagemRecebida:               .asciiz "                  "

EnderecoUART:                           .word 0xFFE00000
```

Em ambos assembly é referenciada a posição da memória em que se encontra a mensagem a ser enviada, bem como a posição em que será salva a mensagem recebida. Isso é possível visualizar através do seguinte trecho de código:

```
la $s0, mensagemTransmitida  #s0 é o ponteiro de mensagemTransmitida
la $s1, mensagemRecebida     #s1 é o ponteiro de mensagemRecebida
```

2.3.1 Programa A

O programa que denominamos como "A"é o assembly responsável por enviar a mensagem do Sistema A para o Sistema B. Para que isso seja possível, antes de iniciar a transmissão, carrega-se no registrador $s2 o endereço da UART deste sistema, conforme pode ser visto abaixo:

```
la $s2, EnderecoUART # Carrega o endereço da UART na memória no registrador
lw $s2, 0($s2) # Carrega o endereço da UART no registrador
```

Feito isso, tem-se, então, as informações necessárias para iniciar a transmissão dos dados. No trecho de código logo abaixo ao supracitado, referenciado pelo título "SaltoMyMain", é efetuado o carregamento da letra cujo endereço está contido no registrador $s0, apontando para a primeira letra da mensagem a ser transmitida. Para cada caractere obtido, verifica-se se este está valorado com 0, pois isso significará que a transmissão fora encerrada.

```
SaltoMyMain:
    lbu $s3, 0($s0)  # Carrega a letra da mensagem em £s3
    # Caso o caractere lido seja igual a 0 (fim da frase),
    #   desvia para o fim do programa
    beq $s3, $zero, UltimoCaracter
    sb $s3, 4($s2)  # Armazena a letra na UART A
    nop
    lw $t1, 4($sp) # Recupera o valor de t1 salvo na pilha
    lw $t0, 0($sp) # Recupera o valor de t2 salvo na pilha
    addiu $sp, $sp, 8 # Libera o espaÁo anteriormente armazenado na pilha
    j SaltoMyMain
```

Em caso de finalização do envio da mensagem, ficará em loop na parte do código "j Fim". Caso contrário, é carregado no registrador $s2 o caractere a ser transmitido. Como dito anteriormente, o registrador $s2 contém o endereço da UART, ou seja, assim que carregado um valor para a transmissão, é efetuado um salto, ministrado pela descrição contida no VHDL, para realizar o tratamento da rotina de interrupção.

Essa rotina salva o contexto atual do código, neste caso apenas os registradores t0 e t1, por esse motivo foi utilizado apenas 2 bytes de espaço na pilha. Após isso, incrementa-se o ponteiro da mensagem que está sendo transmitida. Ao final desse trecho de código, é executada a instrução "ERET", finalizando o tratamento de uma interrupção.

Ao retornar para o trecho de "SaltoMyMain", são recuperados os valores do contexto salvo na pilha, bem como liberado o espaço utilizado para salvar estas informações. Após isso, retorna-se ao inicio deste algoritmo para realizar o envio do próximo caractere ou finalizar o envio.

Grande parte da aplicação citada acima é facilmente verificada no trecho de código antecessor ao visto abaixo. Neste trecho, portanto, é apenas demonstrado como foi realizado o tratamento da interrupção de envio de mensagens.

```
CPU_to_UART: # Tratamento de interrupção de ESCRITA na UART
    addiu $sp, $sp, -8 # Aloca espaço na pilha
    sw $t0, 0($sp) # Salva o conte_do do registrador na pilha
    sw $t1, 4($sp) # Salva o conte_do do registrador na pilha
    addiu $s0, $s0, 1 # Incrementa o ponteiro de mensagemTransmitida
    eret # Fim da rotina de tratamento de interrupção
```

2.3.2 Programa B

O Programa B, em sua maioria, é semelhante ao funcionamento do Programa A, citado no tópico anterior, o que muda é que o Programa B só inicia o envio de sua mensagem após a finalização do recebimento da mensagem do Sistema A. Isso ocorre para que seja possível enviar uma resposta ao Sistema A.

De forma prática, o trecho de código referenciado pelo título "MyMain"de ambos os programas são iguais, porém, o Programa A, ao finalizar este trecho de código, inicia o envio dos caracteres de sua mensagem, o Programa B, no entanto, possui um funcionamento diferente. Ao finalizar o trecho de código que salva os endereços da memória de dados, inicia-se um loop no trecho de código "AguardaRecepcao"que espera o recebimento total da mensagem enviada pelo Sistema A, para então iniciar a transmissão, que, assim como no Programa A, é realizado no trecho nomeado como "SaltoMyMain". O trecho "AguardaRecepcao"pode ser visto abaixo:

```
AguardaRecepcao:
# Caso o caractere lido seja igual a 0 (fim da frase),
#   desvia para o envio da mensagem de sisB para sisA
        beq $s4, $zero, SaltoMyMain
        nop
        j AguardaRecepcao
```

2.3.3 Recebimento de caractere

Em ambos sistemas o assembly programado para tratar essa ação é equivalente. No momento em que é recebido um caractere e detectado pela UART este recebimento, ocorre uma interrupção e o assembly é direcionado para o trecho de código nomeado como "UART_to_CPU". Neste trecho de código, é lido o caractere recebido, armazenado no endereço referenciado da memória de dados "mensagemRecebida"e incrementado o ponteiro para a posição do próximo caractere a ser salvo. Ao finalizar, executa a instrução ERET para retornar à execução do programa.

Este algoritmo pode ser visualizado através do trecho de código abaixo:

```
UART_to_CPU: # Tratamento de interrupção de LEITURA da UART
    # Leitura do caractere escrito na UART B e passa para o registrador s4
        lbu $s4, 8($s2)
        sb $s4, 0($s1)    # Armazena caractere no mensagemRecebida
        addiu $s1, $s1, 1 # Incrementa ponteiro de mensagemRecebida
        eret # Fim da rotina de tratamento de interrupção
```

3 Resultados

3.1 Envio e recebimento de mensagens

Na Figura abaixo, é possível visualizar um trecho da mensagem "Isto eh um teste!", sendo enviada pelo Sistema A, através do "register_wo".

Figura 1: Mensagem enviada pelo Sistema A

Logo em seguida, é possível visualizar os caracteres sendo recebidos pelo Sistema B através do "register_ri", conforme imagem abaixo:

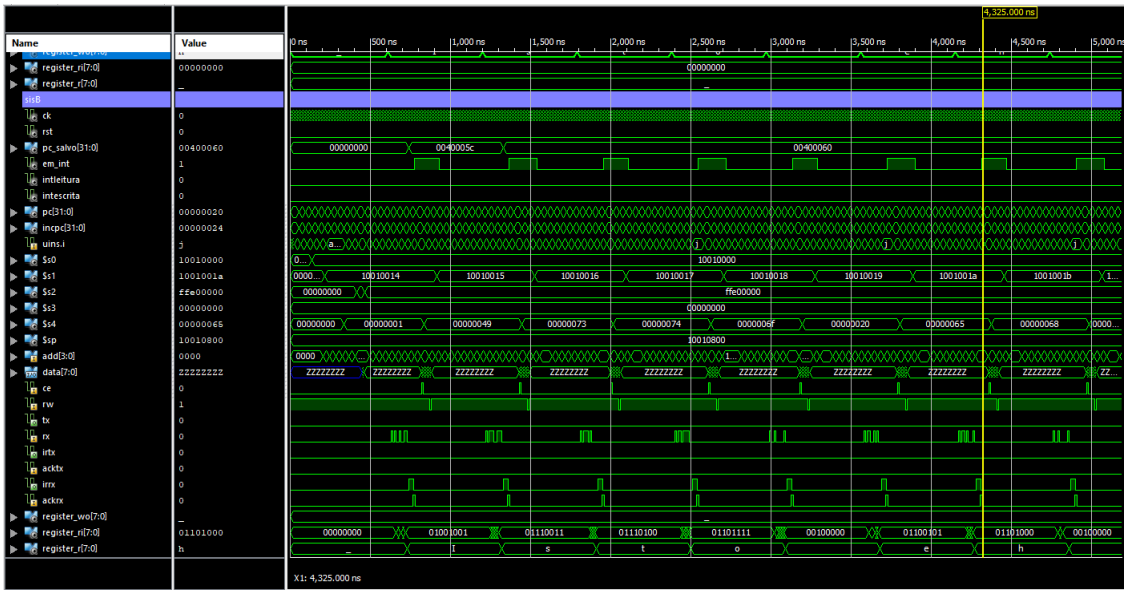


Figura 2: Mensagem recebida pelo Sistema B

Nas duas imagens acima é possível visualizar que no Sistema A (Figura 1) o registrador de entrada, "register_ri"não está recebendo nenhuma mensagem, isso em função do Sistema B não estar, ainda, enviando nenhum dado. Ao contrário deste, no Sistema B (Figura 2) o registrador de saída de dados, "register_wo"ainda não está sendo utilizado, pois este só começará a enviar dados após a finalização do envio do Sistema A.

Na Figura 3, é possível verificar a mensagem "Teste compreendido!"sendo enviada pelo Sistema B.

Já na próxima imagem, Figura 4, será possível visualizar o recebimento da mensagem enviada pelo Sistema B, no Sistema A.

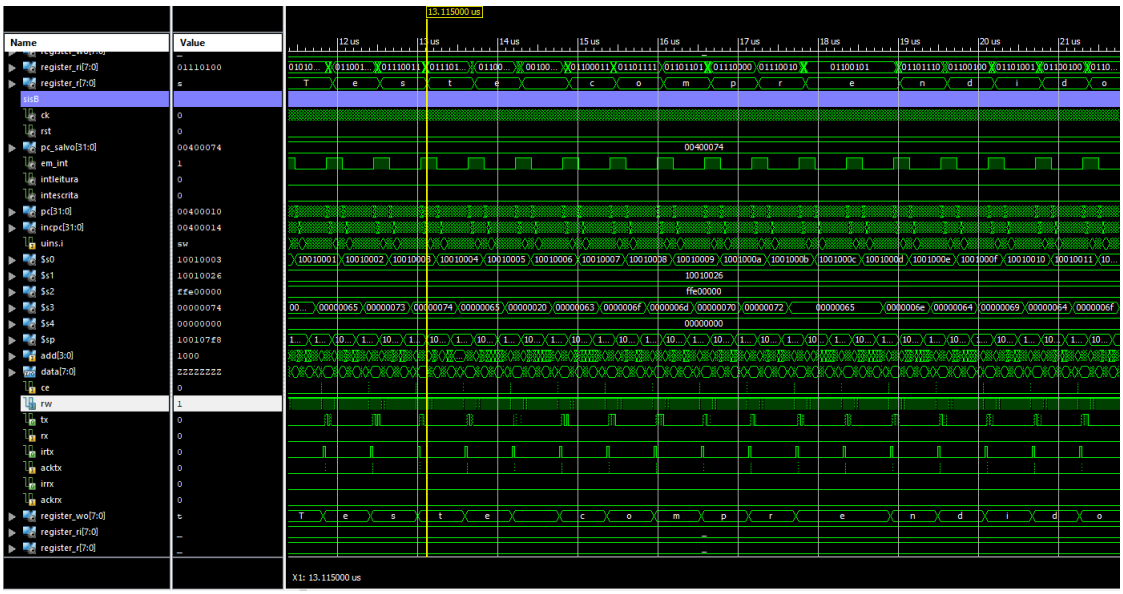


Figura 3: Mensagem enviada pelo Sistema B

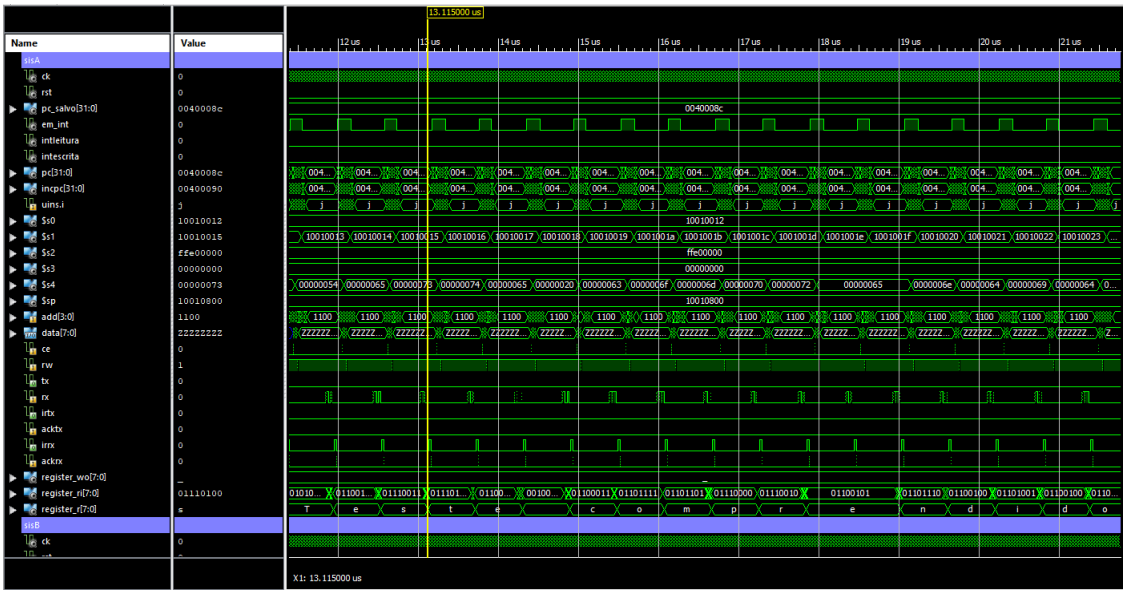


Figura 4: Mensagem recebida pelo Sistema A

Nessas duas últimas figuras, é possível verificar que a utilização dos registradores fora invertida, isto ocorre em função de que a ação de cada sistema fora invertida. Após o recebimento pelo Sistema B da mensagem completa enviada pelo Sistema A, o Sistema B passa a enviar a sua mensagem avisando que o teste foi concluído com sucesso e o Sistema A apenas recebe esta mensagem e a salva na sua memória de dados.

3.2 Memória de dados

Nas imagens abaixo, é possível visualizar a memória de dados dos sistemas contendo, inicialmente, apenas a mensagem a ser transmitida.

Figura 5: Memória de dados inicial

(a) Sistema A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	I	S	T	O		E	H		U	M		T	E	S	T	E
0x10	!															
0x20															À	Ÿ
0x30																
0x40	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x50	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x60	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x70	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x80	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x90	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xA0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xB0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xC0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xD0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xE0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xF0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ

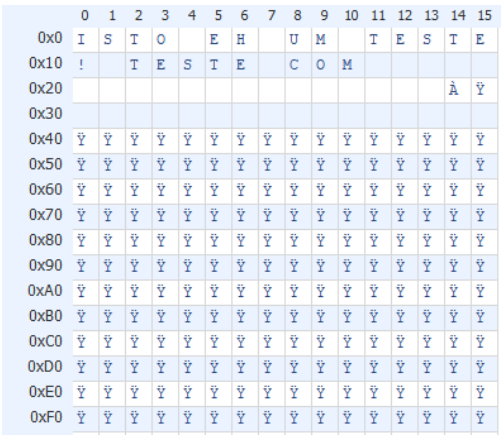
(b) Sistema B

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	T	E	S	T	E		C	O	M	P	R	E	E	N	D	I
0x10	D	O	!													
0x20											À	Ÿ				
0x30																
0x40	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x50	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x60	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x70	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x80	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x90	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xA0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xB0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xC0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xD0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xE0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xF0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ

Já em uma próxima etapa, é possível verificar que a memória de dados está sendo preenchida, ainda não totalmente, isso porque os envios não foram finalizados.

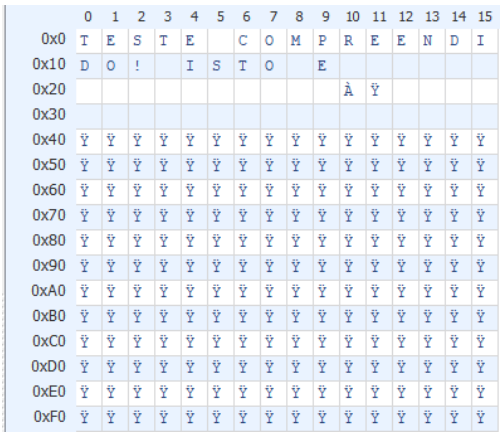
Figura 6: Memória de dados intermediária

(a) Sistema A



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	I	S	T	O		E	H		U	M		T	E	S	T	E
0x10	!		T	E	S	T	E		C	O	M					
0x20															À	Ÿ
0x30																
0x40	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x50	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x60	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x70	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x80	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x90	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xA0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xB0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xC0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xD0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xE0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xF0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ

(b) Sistema B



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	T	E	S	T	E		C	O	M	P	R	E	E	N	D	I
0x10	D	O	!		I	S	T	O		E						
0x20											À	Ÿ				
0x30																
0x40	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x50	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x60	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x70	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x80	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x90	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xA0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xB0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xC0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xD0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xE0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xF0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ

O Sistema A encontra-se nesta etapa, parcialmente preenchido, após a finalização do envio da sua mensagem, pois o Sistema B inicia o envio de sua mensagem apenas após a conclusão do recebimento da mensagem do Sistema A. O Sistema B, no entanto, encontra-se neste estado durante o envio da mensagem do Sistema A, antes do envio da sua mensagem para o outro sistema.

Após a finalização do envio do Sistema A, é possível verificar a mensagem "Isto eh um teste!"salvo na memória de dados do Sistema B. Assim como, ao finalizar a execução do envio da mensagem do Sistema B para o Sistema A, será possível visualizar a mensagem "Teste compreendido!"na memória de dados do Sistema A.

Figura 7: Memória de dados final

(a) Sistema A

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	I	S	T	O		E	H		U	M		T	E	S	T	E
0x10	!		T	E	S	T	E		C	O	M	P	R	E	E	N
0x20	D	I	D	O	!										À	Ÿ
0x30																
0x40	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x50	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x60	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x70	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x80	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x90	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xA0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xB0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xC0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xD0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xE0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xF0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ

(b) Sistema B

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	T	E	S	T	E		C	O	M	P	R	E	E	N	D	I
0x10	D	O	!		I	S	T	O		E	H		U	M		T
0x20	E	S	T	E	!						À	Ÿ				
0x30																
0x40	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x50	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x60	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x70	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x80	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0x90	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xA0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xB0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xC0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xD0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xE0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
0xF0	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ

3.3 Conclusão

A partir de todo o conteúdo dissertado, é possível concluir o funcionamento pleno deste trabalho, no qual o Sistema A envia a mensagem "Isto eh um teste!"para o Sistema B, este sistema armazena esta mensagem recebida e retorna uma reposta "Teste compreendido!"para o Sistema A, que por sua vez, também armazena a mensagem.

Este trabalho teve como consequência a melhoria destas abstrações para os alunos responsáveis por ele, enfatizando alguns tópicos desconhecidos, até então, por nós. Como, por exemplo, como era realizada a comunicação da UART com um sistema e como era tratada uma interrupção em VHDL, entre outros tópicos. No geral, foi possível adquirir um conhecimento excelente, não só pela execução deste trabalho, mas também pelos debates gerados através dele.