TRABALHO: **PROTOCOLO DE COMUNICAÇÃO**

1. OBJETIVO

O **objetivo** desse trabalho é de simular um ambiente de verificação de um sistema digital. Com esse propósito, é dado uma especificação de um protocolo de comunicação, e uma primeira implementação da equipe do "*design*". Sua tarefa é utilizar **code coverage** e **PSL** para apontar os erros no "*design*" e criar um monitor correspondente a especificação do protocolo de comunicação. Esse trabalho deverá ser realizado **em duplas**.

2. ESPECIFICAÇÃO DO PROTOCOLO DE COMUNICAÇÃO

2.1. Definição

O protocolo de comunicação que deverá ser implementado tem o nome de **comm_protocol**. Ele recebe dados de 32 bits (tamanho do dado de entrada) e deve se comunicar com um módulo que trabalha com 8 bits (tamanho do dado de saída).

Os sinais de entrada e saída do módulo **comm_protocol** são mostrados abaixo:

A definição dos sinais são a seguinte:

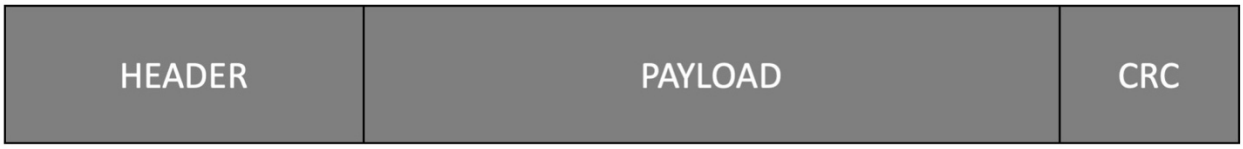
- **rst_n**: é o sinal de reset. Ele é um reset "***negado***", logo esse sinal é ativado quando seu valor for '**0**', devendo resetar os registradores internos do modulo;
- **clk**: este é o sinal do relógio que trabalha a uma frequência de 100 MHz;
- **send_i**: é o sinal de entrada que indica que querem se comunicar com o módulo ***comm_protocol***. Quando esse sinal esta ativo ('**1**'), o valor no sinal ***data_32_i*** é um dado válido;
- **data_32_i**: é o dado de 32 bits correspondente a entrada da comunicação. O primeiro dado válido de uma comunicação é o "***serviço***", indicado pelo ***header*** do pacote. Se for um serviço de envio de mensagem, ele receberá em seguida, a mensagem a ser transmitida (***payload***);

- **busy_o**: sinal que indica a transmissão de parte do pacote, informando que nesse intervalo em que esta ativo (‘**1**’), não pode receber nenhuma requisição de nova mensagem;
- **msg_o**: sinal que quando ativo (‘**1**’) indica que esta no processo de envio de uma mensagem. Sendo assim, nesse período, o módulo só passará adiante a mensagem (***payload***) sem considerar nenhum serviço neste momento;
- **valid_o**: sinal que quando ativo (‘**1**’) indica que o valor no sinal ***data_8_o*** é um dado válido;
- **data_8_o**: é o dado de 8 bits correspondente a saída da comunicação. É por ele que é transmitido o ***header***, ***payload*** e o ***CRC***.

2.2. Serviços Suportados

Este protocolo de comunicação tem 2 serviços: (*1*) ***serviço de mensagem***, onde uma mensagem é transmitida, sendo convertida de 32 bits para 8 bits. Além disso, é acrescido um **CRC** (em inglês, ***Cyclic Redundancy Check***) que corresponde a verificação cíclica de redundância. Esse é um método de detecção de erros normalmente usada em protocolos de comunicação; (2) ***serviço de contagem de mensagens***, onde é informado a quantidade de mensagens que já foram transmitidas enquanto o módulo estiver em funcionamento (isto é, o valor é zerado se o reset for ativado).

O formato do pacote da mensagem no “**serviço de mensagem**” é mostrado a seguir:



O ***header*** tem 32 bits, onde os *bits* correspondem a:

- **De 31 a 24**: endereço de quem esta enviando a mensagem;
- **De 23 a 16**: endereço de quem irá receber a mensagem;
- **De 15 a 8**: valor do serviço. Onde o valor **x“01”** corresponde ao ***serviço de mensagem*** e o valor **x“02”** corresponde ao ***serviço de contagem de mensagens***, outros valores são reservados para futuros serviços;
- **De 7 a 0**: se o header corresponder ao ***serviço de mensagem***, estes bits representam a ***quantidade de mensagens*** (“**qnt_msg**”)desse pacote.

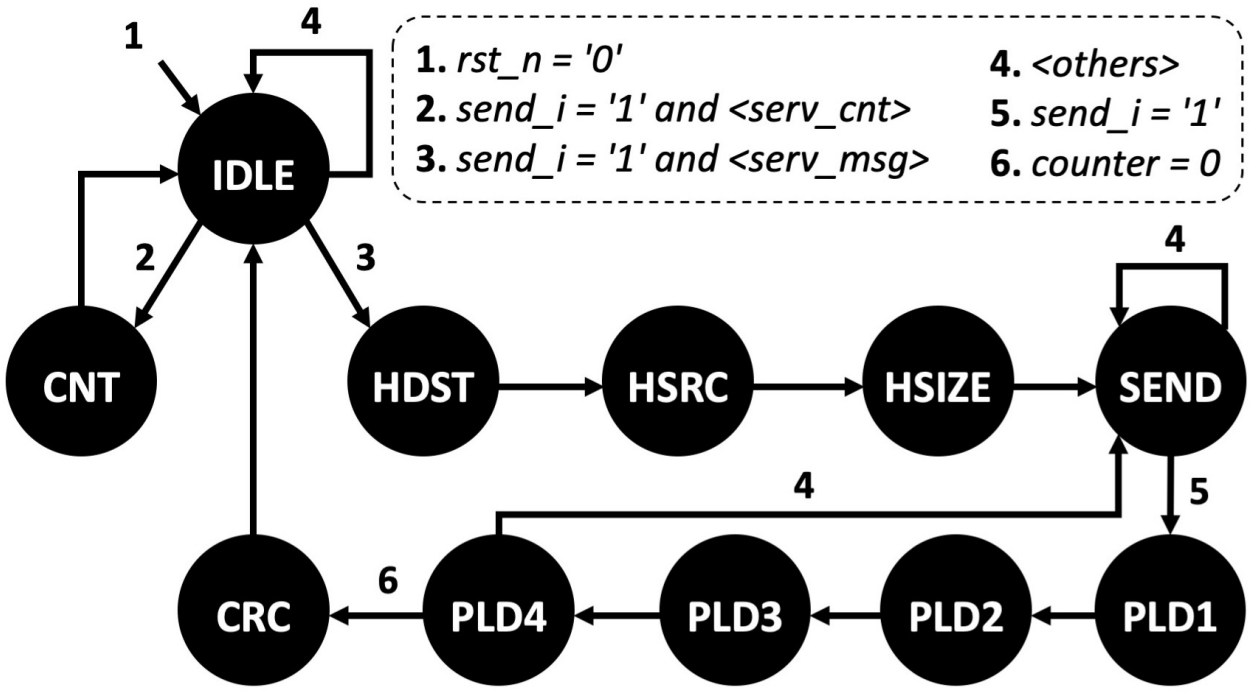
A mensagem de saída correspondente ao ***header*** tem “**3 x 8 bits**” (***destinatário***, ***remetente*** e ***tamanho do pacote***).

O ***payload*** tem 32 bits a cada mensagem de entrada. Logo, note que o tamanho do pacote corresponde a “**4 x qnt_msg x 8 bits + 8 bits**”, sendo que os últimos 8 bits correspondem ao valor do CRC que é adicionado.

O valor do **CRC** é calculado pelo **XOR** dos valores das mensagens de todo o ***payload***, e incluído no final do pacote enviado.

2.3. Máquina de Estados (FSM)

Para que esse módulo funcione da maneira esperada, a **máquina de estados** desse módulo é definida como segue:

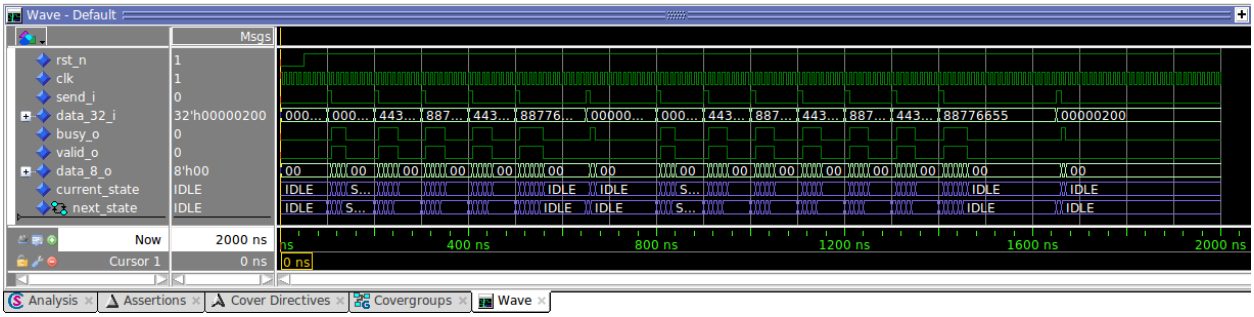


Onde cada estado tem o seguinte significado:

- **IDLE**: estado inicial da máquina de estados. É esse o estado "seguro" onde a máquina vai quando o sinal reset é ativado;
- **CNT**: estado que deves indicar a quantidade de mensagens já foram enviadas por esse módulo;
- **HDST**: estado que deves enviar o destinatário desse pacote de mensagens;
- **HSRC**: estado que deves enviar o remetente desse pacote de mensagens;
- **HSIZE**: estado que deves enviar a quantidade de mensagens de 8 bits restantes ao pacote (***payload*** + **CRC**);
- **SEND**: estado que espera o recebimento da mensagem de 32 bit para encaminha-la em partes menores de 8 bits;
- **PLD1**: estado que envia os dados entre os bits **7** e **0**;
- **PLD2**: estado que envia os dados entre os bits **15** e **8**;
- **PLD3**: estado que envia os dados entre os bits **23** e **16**;
- **PLD4**: estado que envia os dados entre os bits **31** e **24**;
- **CRC**: estado que envia o valor do **CRC** calculado.

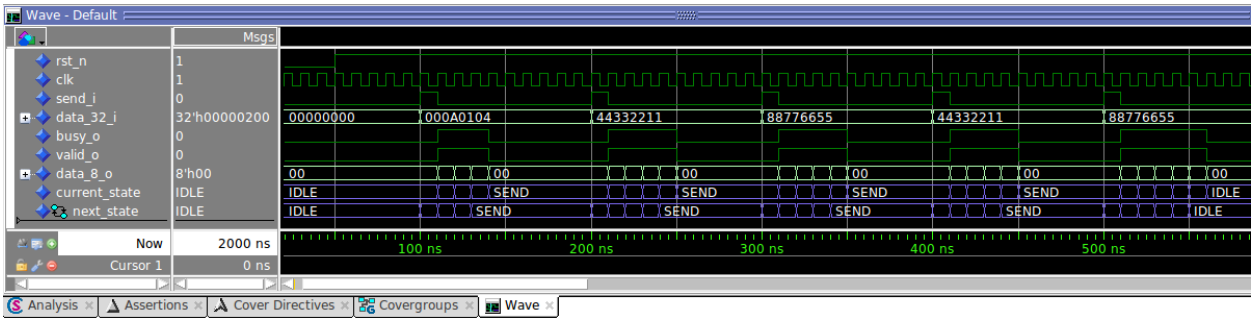
2.4. Funcionamento básico através de formas de ondas

O **funcionamento básico** desse protocolo de comunicação é mostrado na ***Waveform 1***. Nele, o sinal de reset leva o módulo a um estado seguro. Em seguida, são simulados 4 serviços, na seguinte ordem: (1) serviço de mensagem; (2) serviço de contagem de mensagens; (3) serviço de mensagem e (4) serviço de contagem de mensagens.



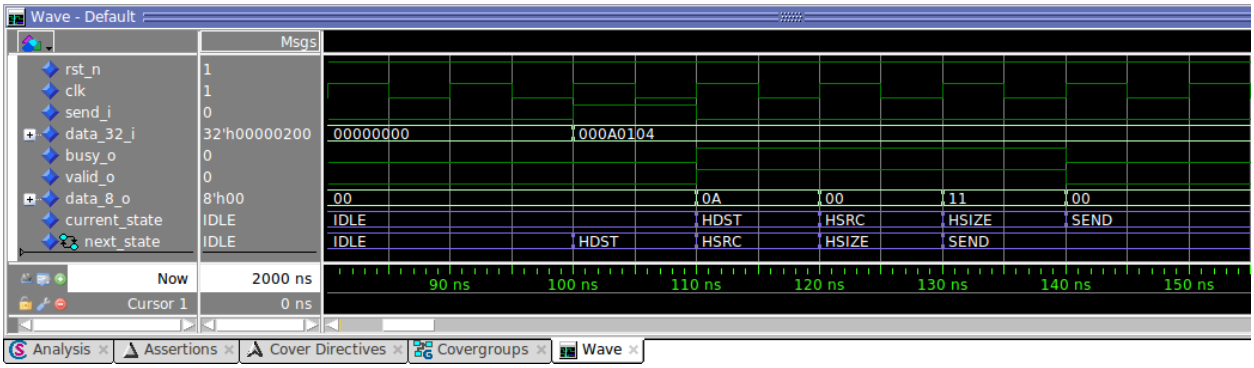
Waveform 1: Funcionamento básico do protocolo de comunicação.

A ***Waveform 2*** mostra um zoom do início de um ***serviço de mensagem***. Nele, vimos que a primeira mensagem de 32 bits é o ***header***. Composto pelo remetente (**x"00"**), destinatário (**x"A0"**), serviço (**x"01"**) e quantidade de mensagens restantes (**x"04"**). A partir disso, vemos a sequência das 4 mensagens (**x"44332211"**, **x"88776655"**, **x"44332211"**, **x"88776655"**).



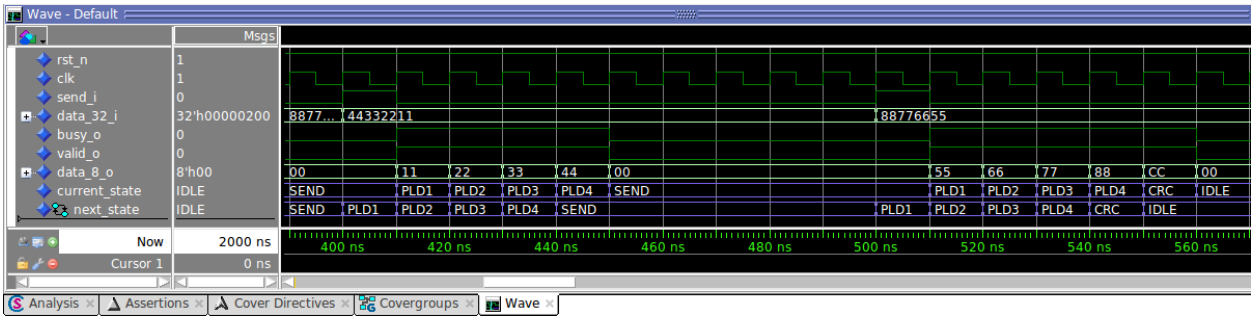
Waveform 2: Funcionamento do serviço de mensagem.

A ***Waveform 3*** mostra a troca dos estados iniciais de um ***serviço de mensagem***, que corresponde aos estados ***HDST***, ***HSRC*** e ***HSIZE***.



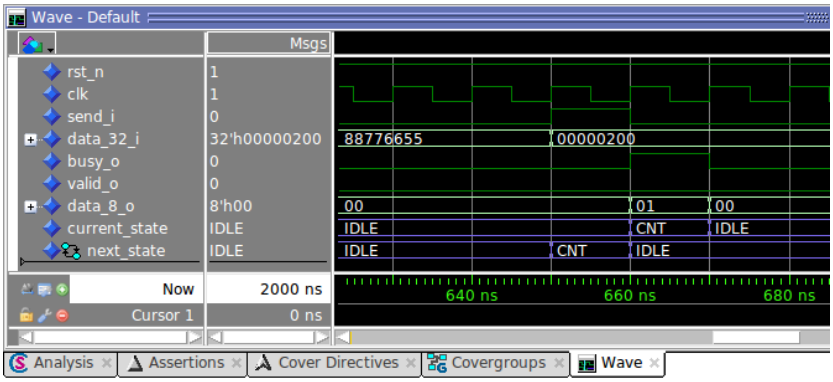
Waveform 3: Zoom mostrando os estados iniciais de um serviço de mensagem.

A ***Waveform 4*** mostra a troca dos estados quando recebemos uma mensagem que faz parte do ***payload***. Ao recebermos a mensagem através do sinal ***send_i***, a troca dos estados é a seguinte: ***PLD1***, ***PLD2***, ***PLD3*** e ***PLD4***. Se tivermos mais mensagens, ao final dessa transmissão voltaremos ao estado ***SEND***, caso esta seja a última parte da mensagem, enviaremos o valor calculado do **CRC** através do estado correspondente (***CRC***) e voltares ao estado seguro (***IDLE***).



Waveform 4: Zoom mostrando os estados finais de um serviço de mensagem.

A ***Waveform 5*** mostra o funcionamento do serviço de contagem de mensagens. Nele, quando o sinal ***send_i*** estiver ativo e recebermos o serviço correspondente (**x"02"**), iremos ficar 1 ciclo de relógio no estado ***CNT***.



Waveform 5: Funcionamento do serviço de contagem de mensagens.

2.5. Casos de Testes

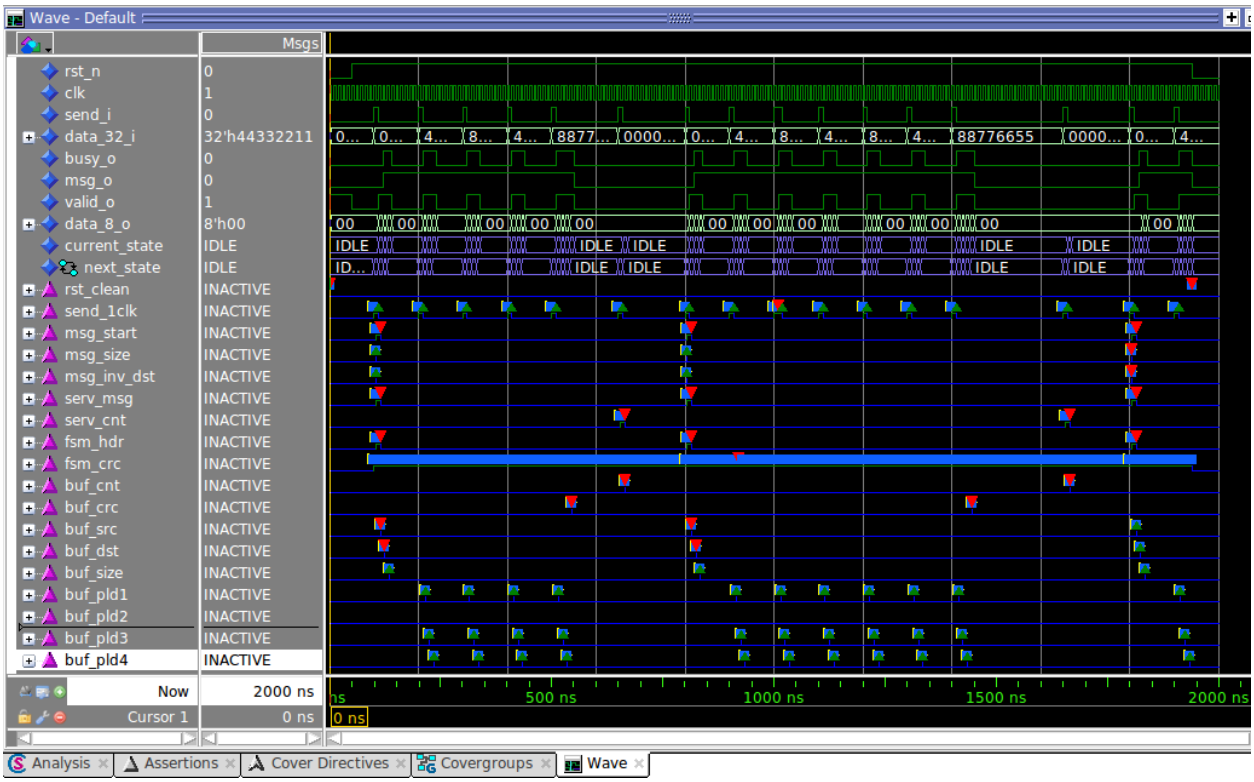
Os casos de testes (em formato de **asserções em PSL**) que devem ser criados para cobrirmos essa especificação são os seguintes:

- **rst_clean**: esta asserção deve controlar para que nunca que o reset esteja ativo, esteja ativo ao também os sinais de: (1) dado válido, (2) ocupado ou (3) o que indica que estamos enviando uma mensagem;
- **send_1clk**: esta asserção deve controlar que o sinal ***send_i*** fique ativo somente 1 ciclo de relógio. Essa asserção só pode ser ativada nas bordas de subida do relógio;
- **msg_start**: esta asserção deve controlar que o sinal que indica o envio de uma mensagem fique ativo no ciclo de relógio seguinte ao recebimento do ***serviço de mensagem***. Essa asserção só pode ser ativada nas bordas de descida do relógio;

- **msg_size**: esta asserção deve controlar a quantidade de mensagens que serão enviadas no ***payload***. Como temos 8 bits de tamanho e perdemos 2 bits na transição de 32 bits para 8 bits, ***os 2 bits mais significativos do tamanho de entrada devem estar sempre zerados***. Esse controle deve acontecer no recebimento do ***serviço de mensagem***. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **msg_inv_dst**: esta asserção deve controlar se o destinatário não é o mesmo que o remetente. Esse controle deve acontecer no recebimento do ***serviço de mensagem***. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **serv_msg**: esta asserção deve controlar se os sinais correspondentes estão ativos no ciclo de relógio seguinte ao recebimento do ***serviço de mensagem***. Essa asserção só pode ser ativada nas bordas de descida do relógio e esta asserção deve ser abortada se o reset estiver ativo;
- **serv_cnt**: esta asserção deve controlar se os sinais correspondentes estão ativos no ciclo de relógio seguinte ao recebimento do ***serviço de contagem de mensagem***. Essa asserção só pode ser ativada nas bordas de descida do relógio e deve ser abortada se o reset estiver ativo;
- **fsm_hdr**: esta asserção deve controlar se os 3 estados correspondentes ao ***header*** foram ativados na sequência no ciclo de relógio seguinte ao recebimento do ***serviço de mensagem***. Essa asserção só pode ser ativada nas bordas de descida do relógio e deve ser abortada se o reset estiver ativo;
- **fsm_crc**: esta asserção deve controlar se o estado CRC esta sendo chamado no momento correto. Esse controle deve ser feito ***N ciclos*** depois do recebimento do ***serviço de mensagem***. Essa asserção só pode ser ativada nas bordas de subida do relógio e deve ser abortada se o reset estiver ativo;
- **buf_cnt**: esta asserção deve controlar se no estado CNT, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **buf_crc**: esta asserção deve controlar se no estado CRC, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **buf_src**: esta asserção deve controlar se no estado HSRC, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **buf_dst**: esta asserção deve controlar se no estado HDST, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **buf_size**: esta asserção deve controlar se no estado HSIZE, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **buf_pld1**: esta asserção deve controlar se no estado PLD1, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio;

- **buf_pld2**: esta asserção deve controlar se no estado PLD2, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **buf_pld3**: esta asserção deve controlar se no estado PLD3, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio;
- **buf_pld4**: esta asserção deve controlar se no estado PLD4, o dado de saída esta pegando o dado do buffer correspondente. Essa asserção só pode ser ativada nas bordas de descida do relógio.

A ***Waveform 8*** mostra os casos de testes aplicadas a versão entregue pela equipe do "design".



Waveform 8: Exemplo mostrando o controle das asserções aplicadas a versão inicial do *design*.

3. MATERIAL DE APOIO

O material de apoio corresponde a pasta ***Comm_Protocol***, o qual contém os seguintes arquivos e definições:

- **comm_protocol.vhd**: esta é a versão entregue pela equipe do "design" correspondente ao módulo do protocolo de comunicação escrito em linguagem VHDL. Note que essa versão contém erros, ao qual você deverá corrigir;

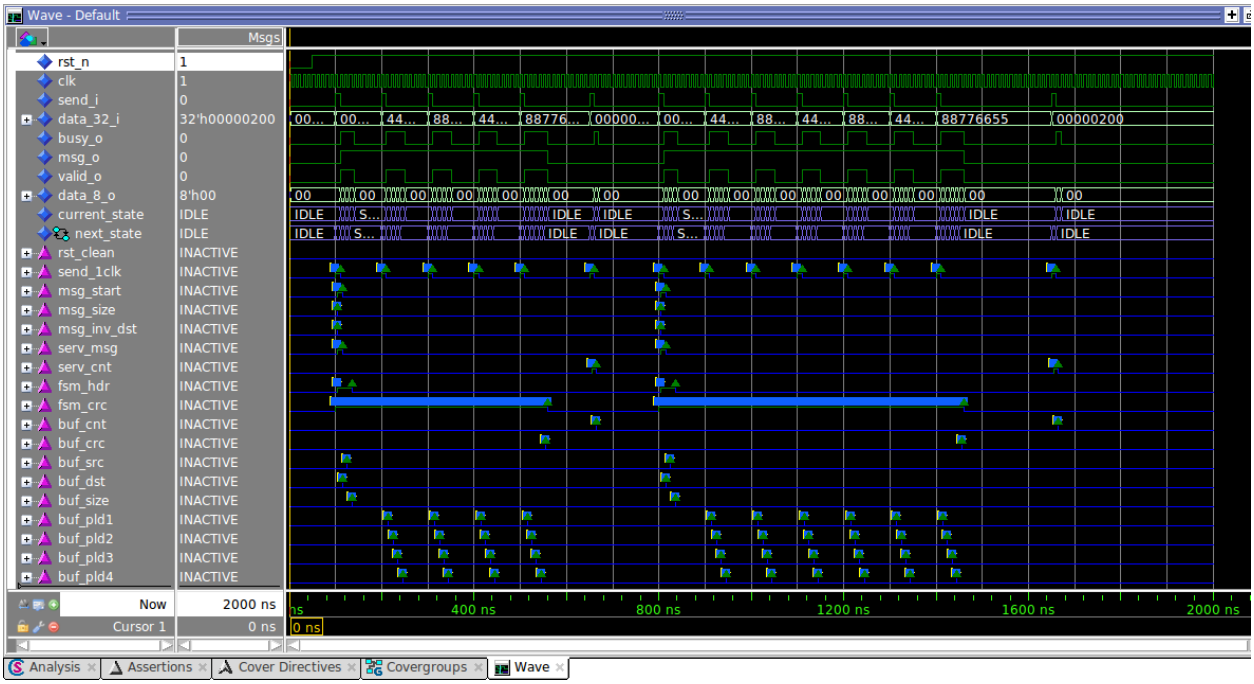
- **tb_comm_protocol.vhd**: esta é a versão inicial do *testbench* feito pelos teus colegas da equipe de verificação. Note que essa versão pode ter alguns problemas e, caso tenha, você deve corrigi-los;
- **load.do**: este é o script de simulação. A cada asserção criada, você deve descomentar as linhas correspondentes nesse arquivo;
- **wave.do**: arquivo que cria as formas de onda padrões do módulo do protocolo de comunicação;
- **coverage_rep**: este arquivo é o ***report*** extraído do **code coverage**.

No material de apoio, o arquivo **coverage_rep**, mostra o ***report*** feito a partir da implementação enviada pela equipe do "*design*". Nela, vemos que não esta coberta integralmente as coberturas de declaração (**97.2%**), ramos (**96.6%**) e os estados da máquina de estados (**90.9%**). Esse são bons indicativos que o "*design*" entregue pode conter falhas, e o seu trabalho como parte da equipe de verificação é descobri-las!!!

4. ATIVIDADES

1. Olhar o arquivo de **code coverage** da implementação oferecida e modificar o *testbench* e o *design* para que além da correção do *design*, seja coberta 100% das coberturas de declaração (statement), ramos (branch) e máquina de estados (FSM - state);

2. Criar as **asserções** necessárias para que passem todos os casos de testes. A ***Waveform 9*** mostra um exemplo de forma de onda da solução.



Waveform 9: Exemplo de solução com todas as asserções em correto funcionamento.