Trabalho 4

Transmissão UART

Este trabalho consiste na criação de um componente chamado UART, que tem como objetivo permitir a receção e o envio de sinais tanto em série como em paralelo.

No caso do envio em série, o componente gera uma sequência de bits correspondente à mensagem. Essa sequência começa com um bit inicial (start bit), que é sempre 0. Em seguida, são transmitidos os 8 bits da mensagem, e, por fim, é adicionado o bit de paridade, encerrando a transmissão de 10 bits no total.

O bit de paridade é calculado com base nos bits da mensagem: se o número de bits a 1 for ímpar, o bit de paridade é definido como 1, garantindo que o número total de bits a 1 seja par, assegurando assim a paridade par.

O sinal de transmissão em série encontra-se, por padrão, a 1 (nível alto). Sempre que ocorre o início de uma transmissão, este sinal é colocado a 0, representando o bit de inicialização (start bit), que indica o começo da transmissão em série. Esta transição de 1 para 0 permite que o componente recetor detete facilmente o início da comunicação e sincronize a leitura dos dados da mensagem.

A respetiva entidade é representada pela imagem abaixo:

```
entity uart is
    Generic (
        data_width : integer := 8
    );
    Port (
        clk : in std_logic;
        rst : in std_logic;
        en : in std_logic;
        start: in std_logic := '0'; -- start signal
        rx : in std_logic;   --reception
        data_in : in std_logic_vector(0 to data_width - 1 ) := (others => '0');
        is_busy: out std_logic := '0';
        data_invalid: out std_logic := '0';
        tx : out std_logic := '0'; --transmission
        data_out : out std_logic_vector(0 to data_width - 1 ) := (others => '0')
    );
end uart;
```

Figura 1 Entidade uart em VHDL

Podemos observar que esta entidade, para além das suas portas, recebe um valor genérico que define o tamanho dos dados a comunicar. Esse valor pode representar, no caso da comunicação paralela, o número de canais de um sinal, ou, no caso da comunicação em série, o número de bits que compõem uma mensagem.
Dessa forma, o componente torna-se mais versátil e facilmente adaptável para utilização em diferentes arquiteturas.

As portas da entidade são as seguintes:

- **Clk** – Sinal de relógio (clock).
- **Rst** – Sinal de reset que reinicia o componente.
- **En** – Ativa ou desativa o funcionamento do componente.
- **Start** – Permite o início do envio dos dados em série, a partir dos dados em paralelo.
- **Rx** – Entrada para receção do sinal em série.
- **Data_in** – Vetor de bits que contém a informação a ser enviada.
- **Is_busy** – Sinal que indica quando o UART está a enviar informação.
- **Data_invalid** – Sinal que indica erro de paridade, ou seja, quando o bit de paridade não corresponde aos dados recebidos, revelando possível corrupção da informação.
- **Tx** – Saída onde é transmitida a informação em série.
- **Data_out** – Vetor de bits que contém a informação recebida em série.

Para testar este componente, foi criado um testbench com duas instâncias do componente UART: uma a funcionar como transmissor e a outra como recetor, designadas respetivamente por uui_1 e uui_2.

O processo é iniciado ao enviar, através do sinal Rx, a informação desejada, já formatada com o bit de início (start bit) e o bit de paridade.
Podemos observar que a mensagem é corretamente recebida pela instância recetora. A imagem abaixo demonstra este processo na simulação.

Figura 2 Receção em série do uart

De seguida, foi realizado o mesmo teste, mas com o bit de paridade invertido, com o objetivo de verificar se o componente deteta o erro e ativa o sinal Data_Invalid ao calcular um bit de paridade diferente do esperado.
A imagem abaixo demonstra esse comportamento na simulação.

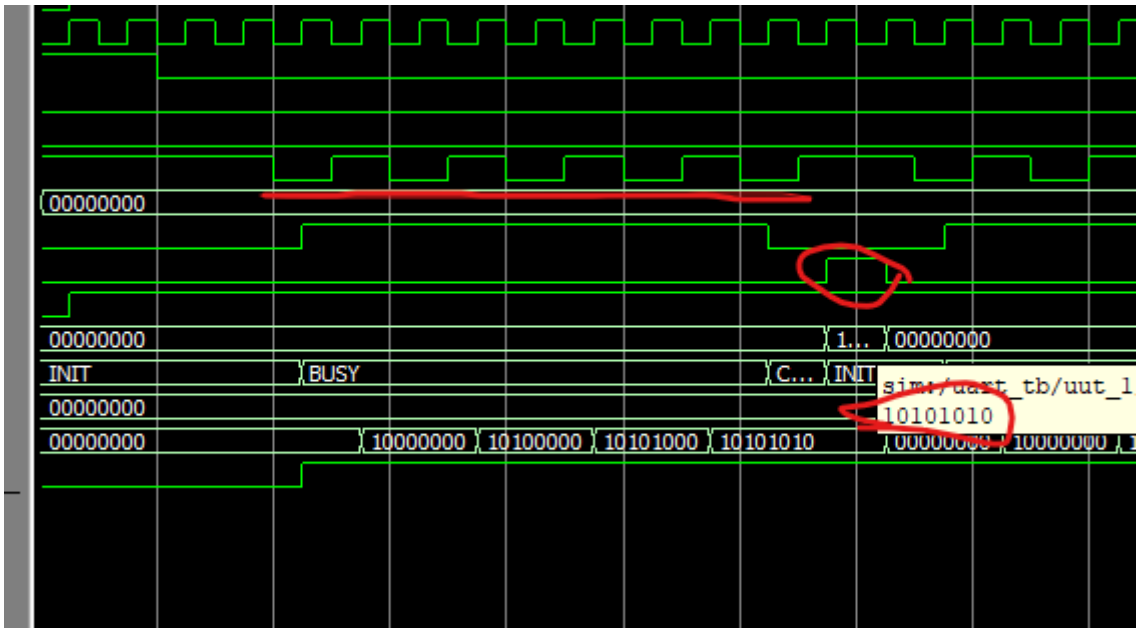

Figura 3 Sinalização de data inválida

Por fim, é simulado o envio de uma mensagem de um UART transmissor para outro UART recetor.

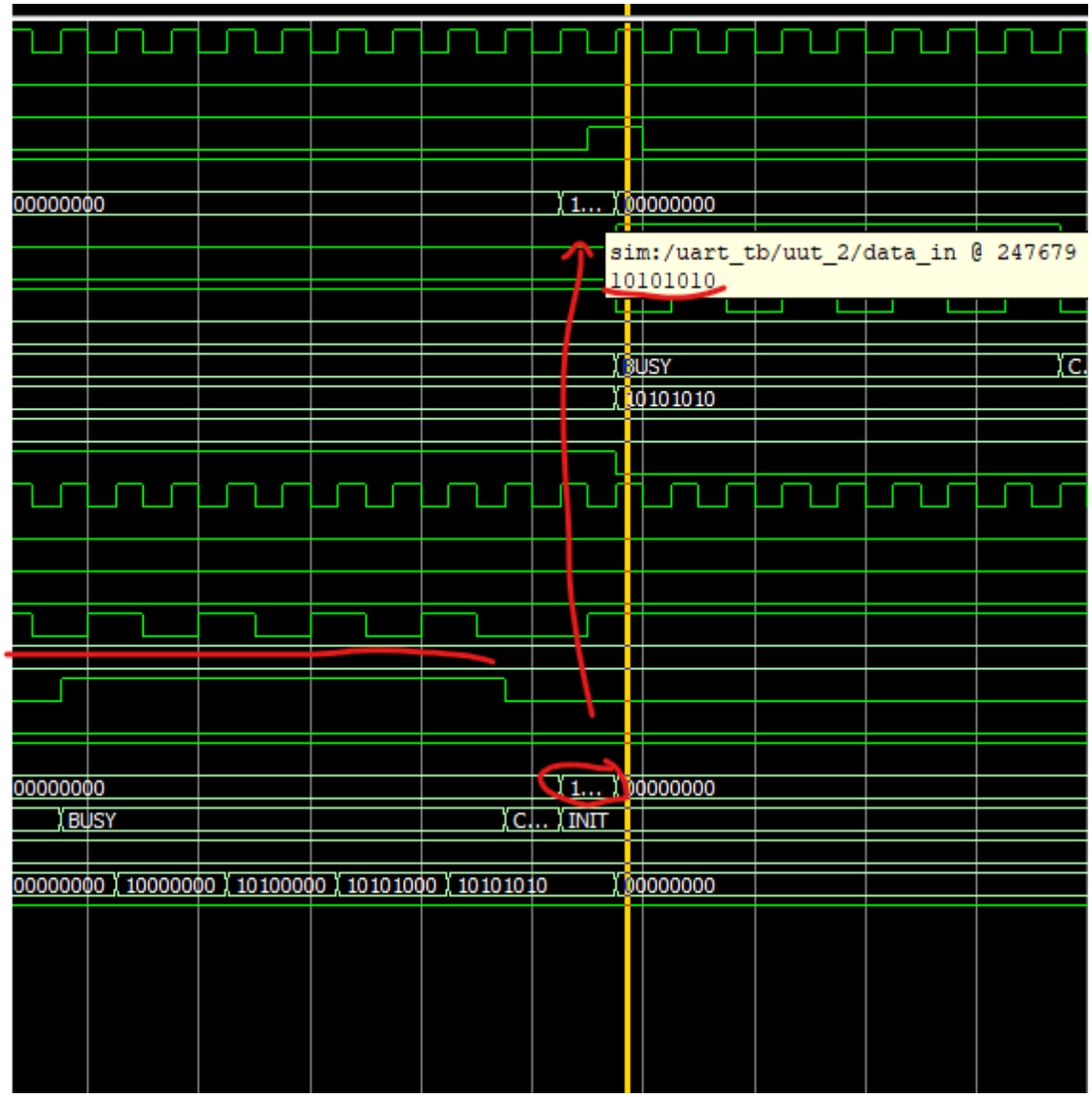

Figura 4 Envio entre componentes

Podemos verificar a entrada do sinal no Rx (primeira linha vermelha sublinhada à esquerda) e, após a receção, a informação é imediatamente enviada para outro componente.
De seguida, esse segundo componente tenta reenviar a informação em série para um terceiro componente, que neste caso não existe na simulação.