ESCOLA POLITÉCNICA DA USP

# Relatório Final do Projeto barramento Amba


| | |
|---|---|
| Thábata Pontes | 9868300 |
| Guilherme Vecchi | 9838115 |
| João Lins | 9347880 |
| Giovanni Abeni | 9837121 |
| Lucas Pereira | 9853335 |


São Paulo

2019

# Sumário

# 1. Introdução

O presente trabalho é sobre a implementação do barramento AMBA 3 AHB-Lite, mais especificamente sobre as etapas de realização para alcançar o barramento completo, além de sua descrição e de outros barramentos possíveis. O objetivo deste barramento é permitir que sejam interligados módulos externos de mestres e escravos - descritos ao longo do relatório - de forma a possibilitar que os mesmos façam uso correto do barramento.

Este relatório está organizado em cinco partes, na primeira, sobre a descrição do barramento, será abordada a definição de um barramento, seu significado dentro do contexto de computadores, além de um aprofundamento sobre o protocolo AMBA, os casos de uso com barramento - quais mestres e escravos são comuns, dentre outros - e, finalizando, as escolhas de projeto.

Na segunda parte, serão abordadas descrições dos módulos que compõe o barramento e seus respectivos diagramas em RTL, como: mestres, escravos, multiplexador do barramento, decodificador do barramento e o árbitro, este último subdividido pelos seus componentes (multiplexador, decodificador, registrador e seletor de mestre). Já na terceira parte será abordada, brevemente, a questão de dependências, que devido ao fato de terem sido contornadas, não prolongaram o assunto.

Na quarta parte serão apresentados os testes unitários, ou seja, testes feitos para cada um dos componentes do barramento, demonstrando seu correto funcionamento através de formas de onda comprobatórias. Na quinta e última parte, trata-se das simulações de mestres, escravos, casos de leitura e escrita, de concorrência entre mestres ou conflitos temporais de acesso.

Como é sabido, a parte teórica ou prática do projeto não foram estudadas antes pelos integrantes da equipe ou da turma, por esse motivo, a metodologia utilizada para implementar o barramento foi através de pesquisas em documentações e outras informações dispersas, além da consulta ao professor para validação da teoria.

# 2. Descrição do barramento

## a. O que é um barramento?

Um barramento é uma interface de conexão entre mestres e escravos, que possibilita transferências de dados entre estes, ou seja, é um conjunto de linhas que permite a ligação entre tais dispositivos. Os mestres são alguns dos componentes conectados ao barramento, que fornecem um sinal de endereço e controlam a informação para iniciar operações de leitura e escrita, como a CPU, por exemplo. Já os escravos, também são ligados ao barramento e podem ser periféricos e memória externa ou interna, e eles são responsáveis por responder às transferências iniciadas pelos mestres. Essas transferências podem ser do tipo básicas, como leitura e escrita, além de operações como *burst*, *locked*, *waited*, que serão menos abordadas neste relatório.

Um barramento pode ter interligações com diversos mestres e escravos, sendo que o número máximo fica a cargo da escolha de projeto, que será melhor detalhada a frente. Desta forma, um ilustrativo de como é a estrutura macro do barramento, supondo 2 mestres e 2 escravos, pode ser visto na figura abaixo:

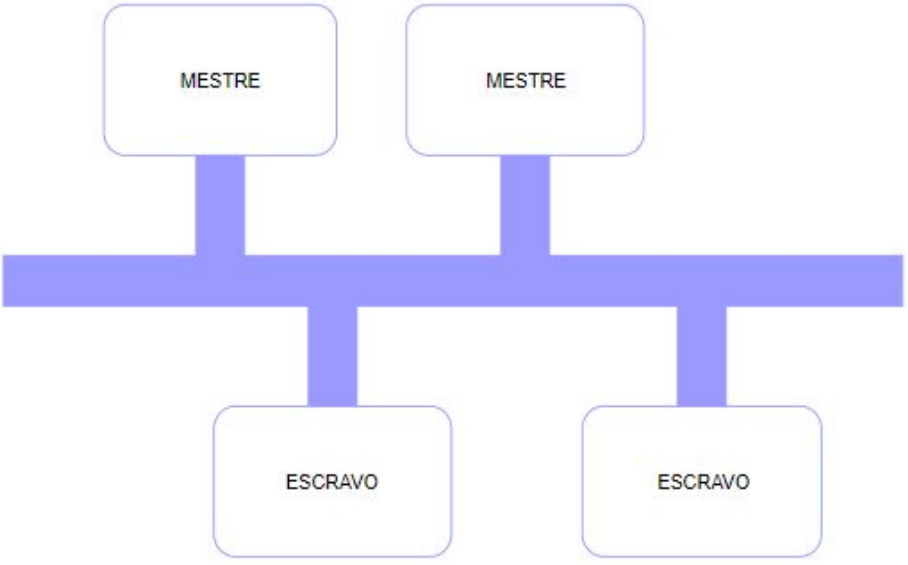



**Figura 1 -** Exemplo de estrutura macro de um barramento.

## b. O protocolo AMBA

Este protocolo, chamado de "*Advanced Microcontroller Bus Architecture"* , é um tipo de interconexão para gerenciar blocos funcionais, ou seja, ele permite o desenvolvimento de projetos com multiprocessadores porque permite um número alto de controladores e periféricos interligados em uma arquitetura de barramento como dito na seção anterior.

O protocolo foi criado pela ARM e os barramentos AMBA podem ser o ASB (*Advanced System Bus*), APB (*Advanced Peripheral Bus*), AHB (*Advanced High-Performance Bus*) sendo o último um barramento de alto desempenho, dentre outros mais recentes. Um dos grandes objetivos do protocolo é promover o desenvolvimento de projetos de microcontroladores com uma ou mais CPUs, GPUs ou processadores de sinal. Além de minimizar a infra-estrutura de silício ainda aumentando o desempenho.

## c. Casos de uso

O AMBA AHB é uma interface destinada para fornecer grandes taxas de transferências de dados (largura de banda alta), utilizando frequências elevadas de clock. Os escravos mais comuns utilizados nesses barramentos são dispositivos de memória internos, interfaces de memória externas, e periféricos de largura de banda alta.

## d. Escolhas de projeto

A primeira escolha do grupo para a realização do presente projeto foi com relação ao tipo de barramento que seria implementado, o AMBA 3 AHB-Lite um subconjunto do AHB, escolhido por alguns motivos como: maior documentação disponível e de fácil acesso, é um barramento de alta velocidade, com usos para processadores e memórias que se encaixam bem no contexto dos projetos da turma, tem maior facilidade de implementação e recursos de teste automático, além de permitir múltiplos mestres (mesmo que dois não possam utilizar o barramento simultaneamente), transferências na subida de clock, e outros. Já um barramento do tipo APB (*Advanced Peripheral Bus)* possui baixa velocidade e é um barramento

secundário, usado normalmente como um escravo em barramentos como o AHB e ASB.

Outra escolha do projeto foi com relação ao número de mestres e escravos, o grupo tentou utilizar o *generic* para todos os componentes, mas isto dificultou a compilação do projeto e gerou diversos erros que não conseguimos resolver. Por isso, decidiu-se que todo o projeto seria adaptado para comportar até 4 módulos de mestres e até 16 módulos de escravos.

# 3. Módulos

O que será neste relatório chamado de “módulos” são os componentes associados ao barramento AMBA. Para o caso de o barramento possuir somente um mestre e vários escravos (que serão melhores detalhados nos itens a seguir), faz-se necessário somente os componentes multiplexador, decodificador e o encapsulamento de mestre e escravo. Como dito anteriormente em escolhas de projeto, preferiu-se possibilitar a interconexão de mais de um mestre ao barramento, e, desta forma, acrescentou-se o módulo do árbitro, em que sua definição e implementação serão descritas a frente. Este consiste de, basicamente, um módulo para gerir o acesso de mestres aos escravos. Tal árbitro, possui internamente outros componentes, como multiplexador, registrador, decodificador e o que chamamos de “MasterSelector” que possui a função de selecionar o mestre que acessará o barramento. De forma macro, o projeto do barramento AMBA ficaria assim:

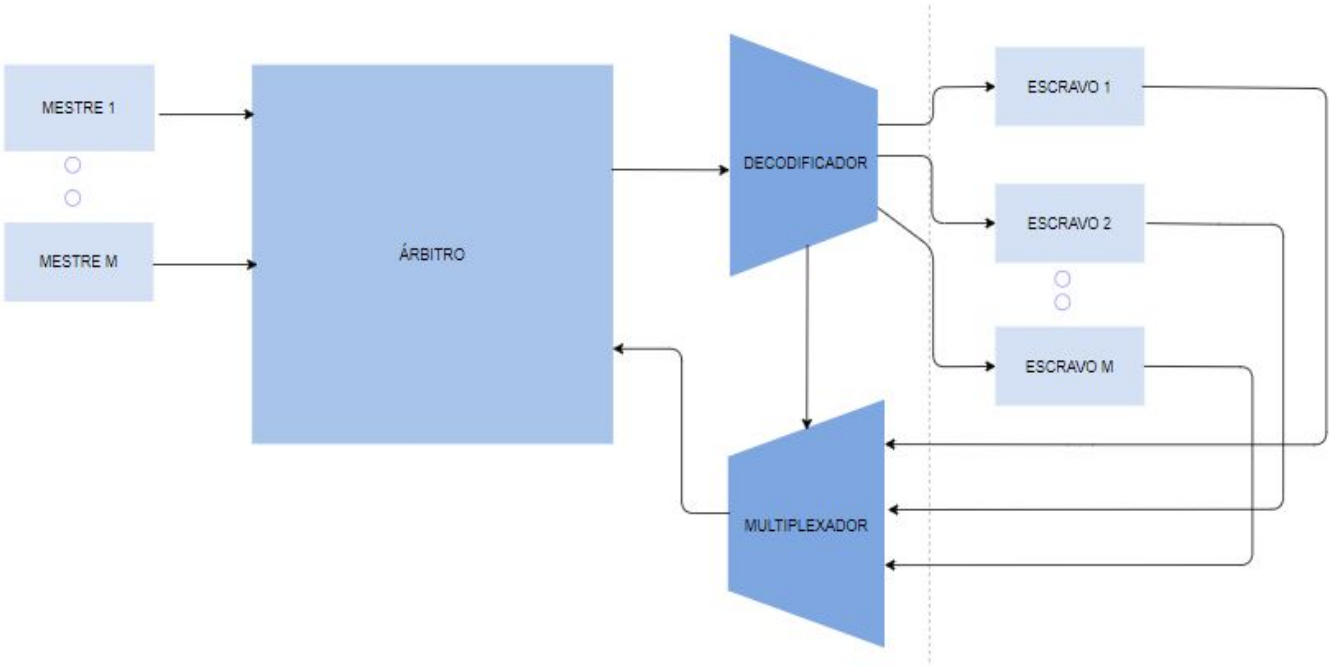



**Figura 2a -** Exemplo de estrutura do barramento com os módulos.

Aumentando a especificidade, após as devidas implementações terem sido realizadas, o RTL do projeto resultou na figura a seguir:

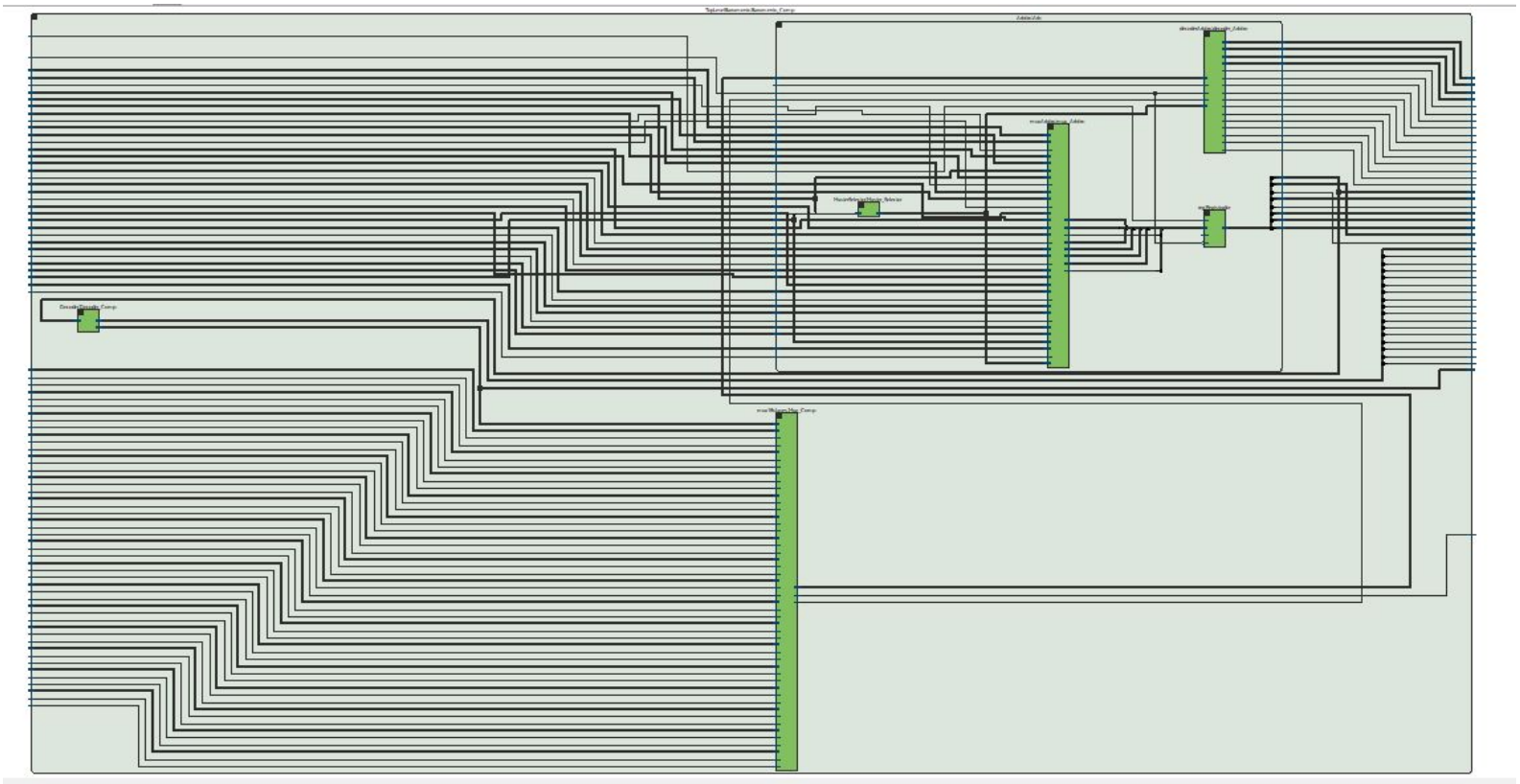


**Figura 2b -** Exemplo de estrutura do barramento com os módulo árbitro, decodificador e multiplexador.

### a. Escravos

Como dito na seção de barramento, a função do escravo é responder as transferências iniciadas pelo mestre. Utiliza do sinal HSELx (o x é substituído por um valor que identifica este sinal para cada escravo) mostrado no diagrama abaixo, vindo da saída do decodificador, para controlar quando ele responde a uma transferência do barramento.

Alguns dos sinais relevantes dos escravos são o HRDATA, que durante operações de leitura, transfere dados de leitura do escravo ao multiplexador do barramento, e ao árbitro que destina ao respectivo mestre. Outro sinal é o HREADYOUT, que em HIGH indica transferência finalizada no barramento. Por último, o sinal HRESP, quando *LOW* indica transferência com status "*OK*" e *HIGH* significa "*ERROR*".

Para ilustrar este módulo, a seguinte figura representa os sinais de um escravo:

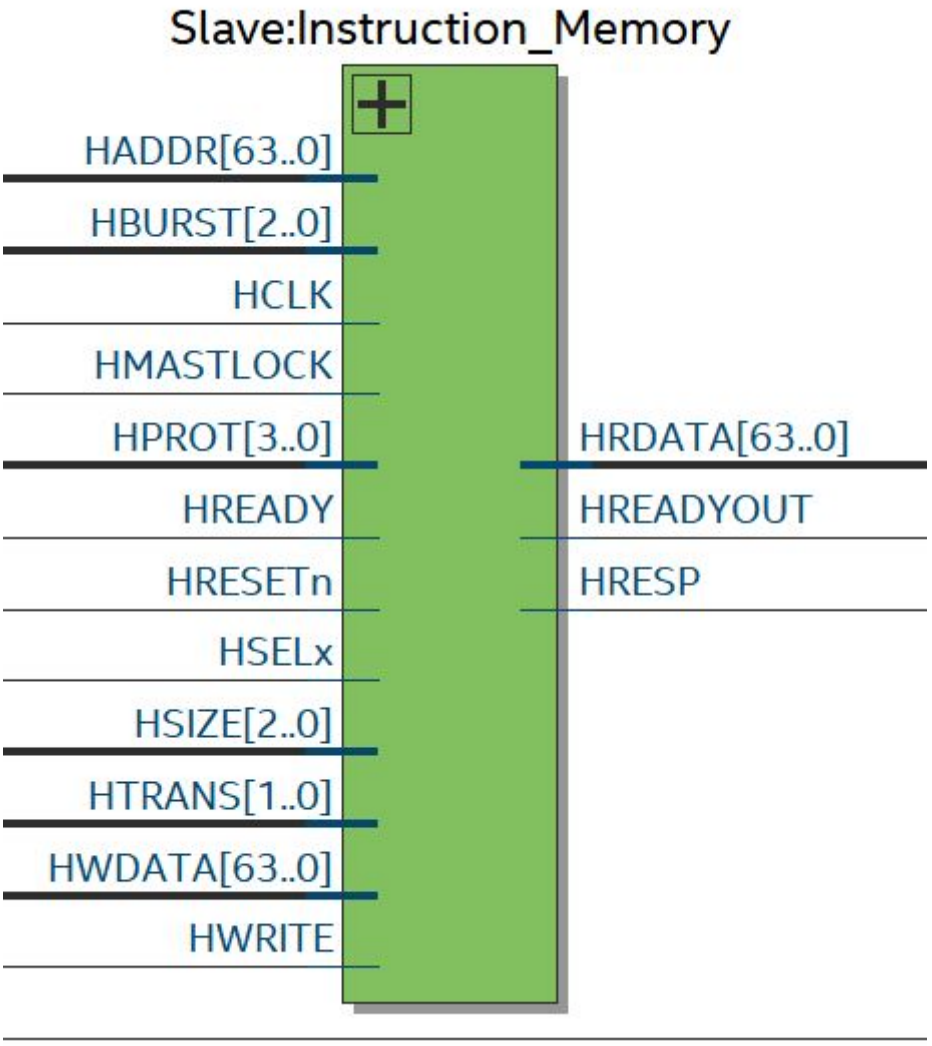



**Figura 3 -** RTL final de um encapsulamento de escravo, nesse caso instanciado como Memória de Instrução (mas o formato é padrão).

## b. Mestres

Já os mestres, como antecipado na seção de barramentos, é responsável por fornecer endereço e controlar informação para iniciar operações de leitura e escrita. Esse endereço é dado através do sinal HADDR, outros sinais como HBURST, HMASTLOCK e HPROT não serão melhor descritos aqui, porque apesar de estarem contemplados no encapsulamento dos mestres, não são implementados ou trabalhados efetivamente como os outros sinais.

O sinal HSIZE indica o tamanho de uma transferência e ,apesar do protocolo AMBA permitir que este tamanho chegue a 1024 bits, deixou-se fixo no projeto em 64 bits. HTRANS é um sinal vital para que seja possível viabilizar o funcionamento

do componente árbitro, que será detalhado no item a seguir, pois tal sinal indica o tipo da transferência atual, podendo ser IDLE, BUSY, NONSEQUENTIAL ou SEQUENTIAL, e dado este tipo, consegue-se selecionar no multiplexador e decodificador do árbitro qual o mestre deve utilizar o barramento, a partir da prioridade. Por fim, o sinal HWDATA transfere dado do mestre ao escravo durante uma operação de escrita, enquanto que HWRITE indica a direção da transferência, ou seja, quando *HIGH* é de escrita, e *LOW* de leitura. Para ilustrar esse módulo, a figura a seguinte contempla o RTL da implementação do mesmo, com os devidos sinais:

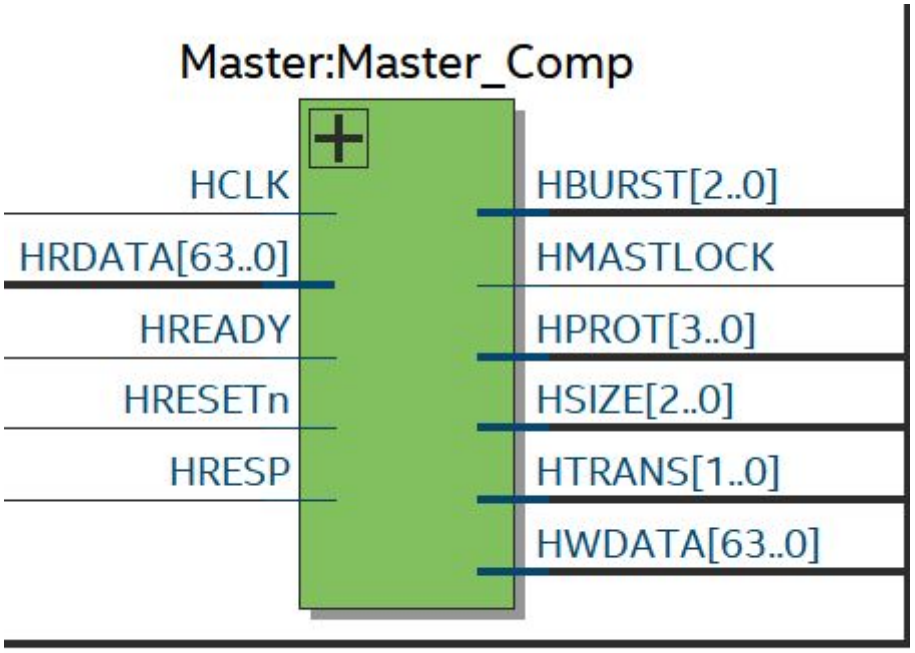



**Figura 4 -** RTL final do encapsulamento de um mestre.

### c. Árbitro

O árbitro é um componente necessário ao barramento por conta do projeto permitir a entrada de mais de um mestre. Com isso, deve-se decidir, com base em uma prioridade fixa - ou seja, em que o grupo escolheu a respectiva prioridade e fixou-a - qual o mestre poderá acessar o barramento. Desta forma, o árbitro é composto por mais 4 módulos, um multiplexador, um decodificador, um registrador e o MasterSelector, este último, criado pelo grupo para solucionar a questão da

prioridade. De maneira macro, o seguinte diagrama pode ilustrar os módulos que compõe o árbitro:

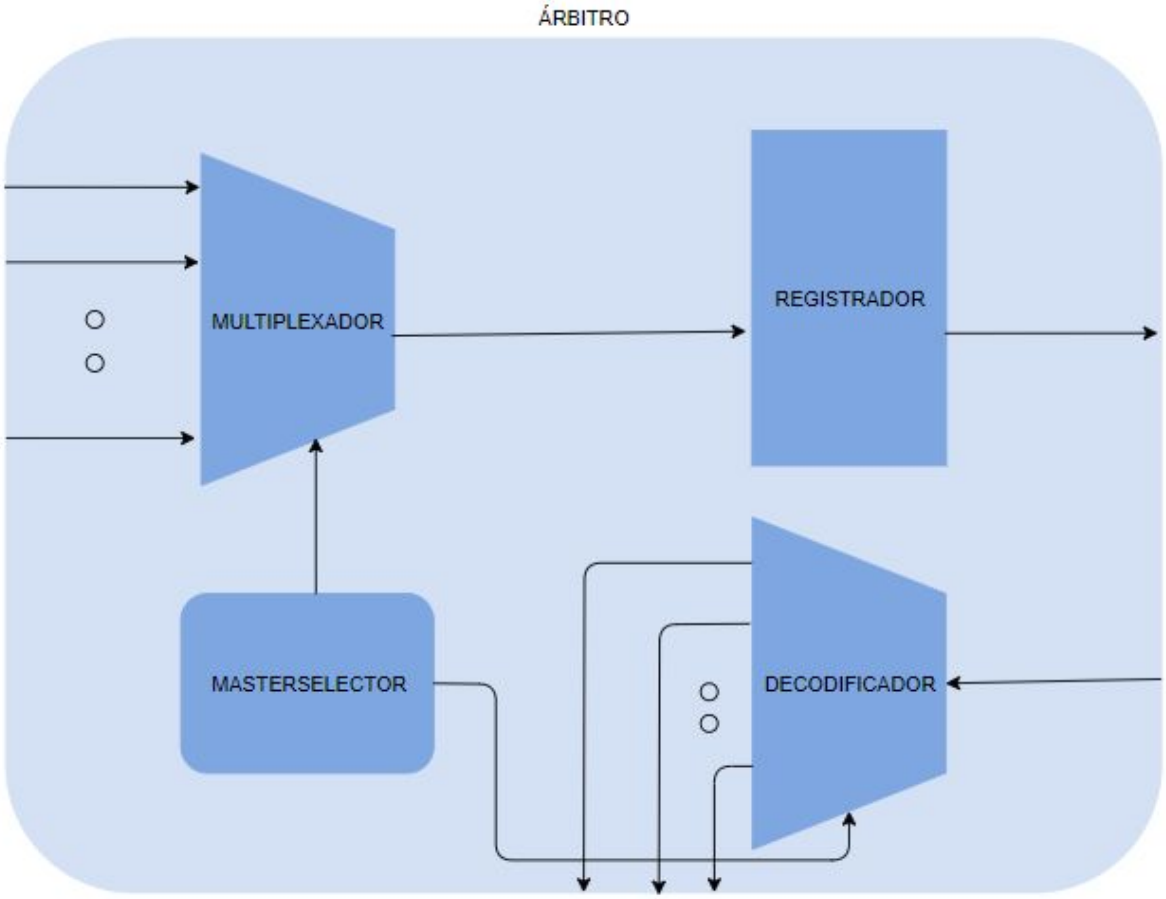



**Figura 5 -** Exemplo de estrutura macro do árbitro.

Mais especificamente, trazendo as interconexões e componentes internos do Master Selector e os outros módulos do árbitro, o RTL implementado resultou na seguinte imagem:

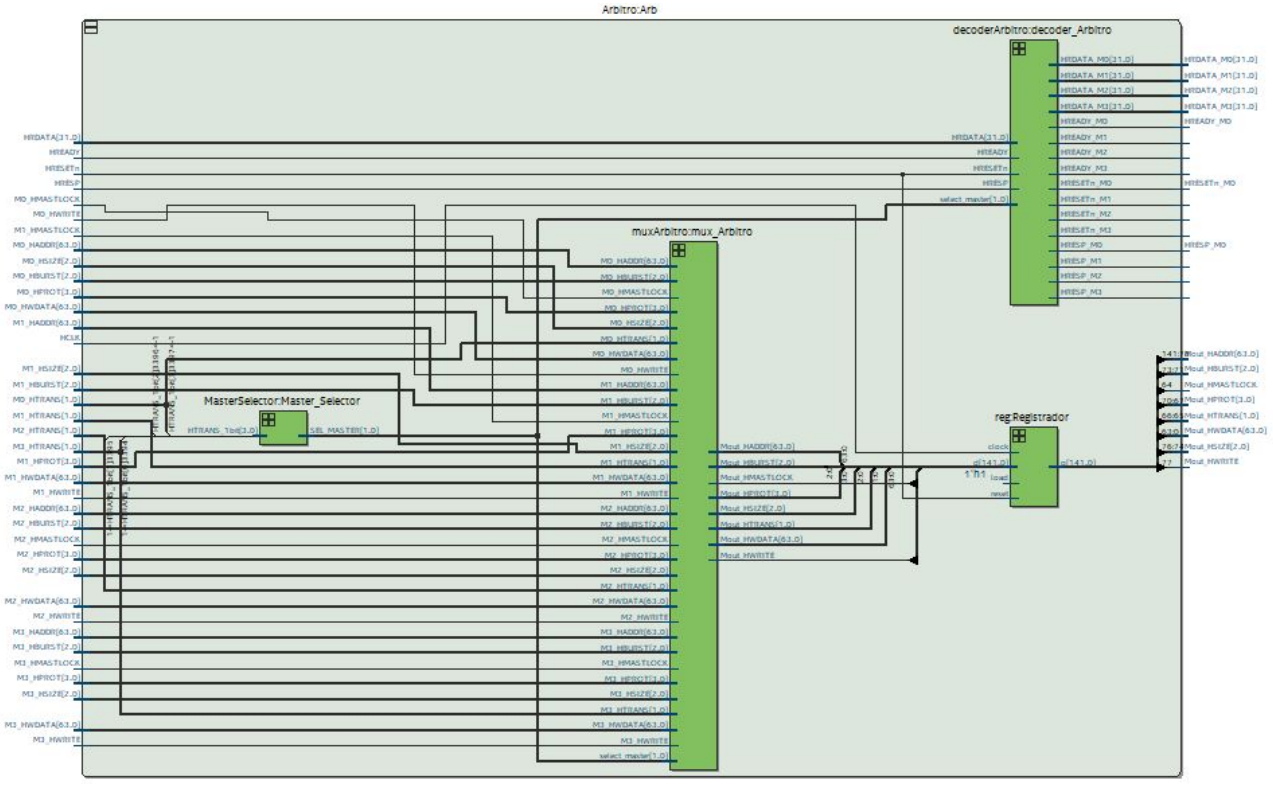



**Figura 6 -** RTL final do árbitro - consultar código para detalhes.

### i. Multiplexador de seleção do Mestre

Este multiplexador tem a função de decidir qual dos mestres pode acessar o barramento. Sendo assim, a lógica por trás de tal escolha é fornecida pela entrada do MasterSelector, que será detalhada no tópico correspondente. O multiplexador do árbitro tem como entradas todas as saídas de um mestre, além do sinal seletor. E a lógica implementada dentro deste componente é a seguinte, para cada sinal do mestre que poderá acessar o barramento, ele recebe o sinal:

- do Mestre 0, caso o seletor seja “00”;
- do Mestre 1, caso o seletor seja “01”;
- do Mestre 2, caso o seletor seja “10”;
- do Mestre 3, caso o seletor seja “11”

É importante frisar que a saída desse multiplexador é um sinal concatenado de todos os sinais do mestre selecionado. Sendo assim, o RTL final desta implementação é apresentado no diagrama a seguir:

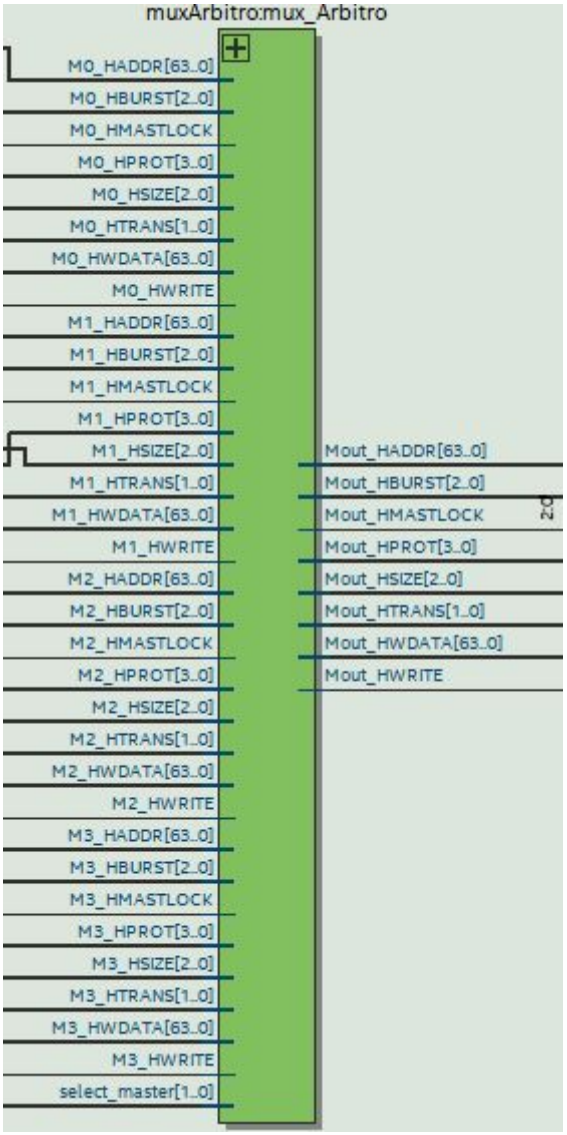



**Figura 7 -** RTL final do multiplexador do árbitro - consultar código para detalhes.

### ii. Decodificador de seleção do Mestre

O decodificador de seleção do mestre recebe como entradas os sinais vindos do barramento: HREADYOUT, HRESET, HRESP e HRDATA, que são as saídas do escravo. Além disso, a entrada seletora do decodificador é, semelhante ao multiplexador, a saída de seleção do componente Seletor do Mestre ou “MasterSelector” como representado no código. A lógica para mandar os sinais vindos do escravo para o mestre correto também é feita através do sinal seletor:

- Mestre 0 recebe os sinais, caso o seletor seja “00”;
- Mestre 1 recebe os sinais, caso o seletor seja “01”;
- Mestre 2 recebe os sinais, caso o seletor seja “10”;

- Mestre 3 recebe os sinais, caso o seletor seja “11”.

  O RTL do decodificador de seleção do Mestre pode ser visto na figura abaixo:

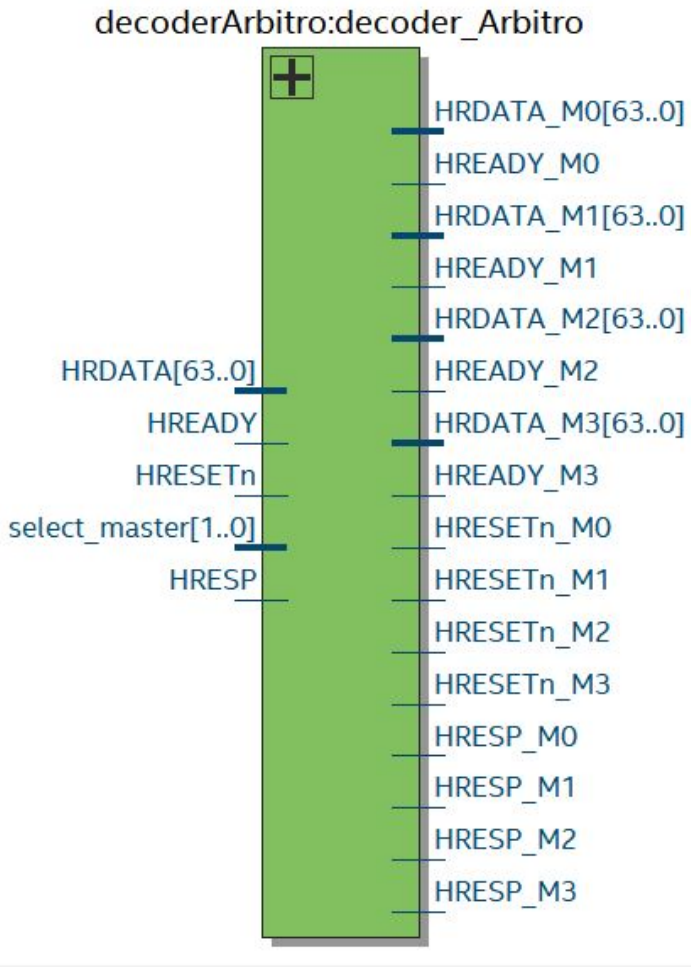



**Figura 8 -** RTL final do decodificador do árbitro - consultar código para detalhes.

### iii. Seletor do Mestre

O Seletor do Mestre é o componente responsável pela lógica de prioridade fixa que escolherá qual mestre tem acesso ao barramento. Ele recebe como entrada o sinal HTRANS, descrito na seção do Mestre, como sua saída que indica o tipo de transferência da atual, tendo quatro possibilidades: IDLE sendo representado por “00”, BUSY como “01”, NONSEQUENTIAL como “10” e SEQUENTIAL “11”. Foi considerado que quando o primeiro bit do sinal HTRANS fosse “1”, ou seja, tipo NONSEQUENTIAL ou SEQUENTIAL, o mestre poderia acessar ao barramento.

Para definir a prioridade no caso de mais de um mestre ter o primeiro bit do sinal HTRANS como “1” optou-se por fazê-la “sequencial crescente”, ou seja, caso o Mestre 0 esteja com o bit em “1” e outros também, a prioridade é dada a ele, ou se,

por exemplo, o Mestre 2 estiver com o HTRANS[1] em “1” e nem o Mestre 1 ou 0 estiver com tal bit em “1”, a prioridade então será dada a ele.

É importante ressaltar também que a entrada HTRANS é a concatenação do primeiro bit do sinal HTRANS de cada mestre, tendo assim, 4 bits. A saída do componente é o selecionador de mestre ou “SEL_MASTER” como no código, e segue a lógica descrita acima:

- Para HTRANS “1- - - “, selecionador = “00” (Mestre 0);
- Para HTRANS “01- -”, selecionador = “01” (Mestre 1);
- Para HTRANS “001-”, selecionador = ”10” (Mestre 2);
- Para HTRANS “0001”, selecionador = “11” (Mestre 3).

Nesse caso “-” representa *don’t care*, ou seja, independente do valor dos mestres seguintes, caso aquele tenha o primeiro bit de HTRANS = ‘1’ a prioridade é do mesmo.

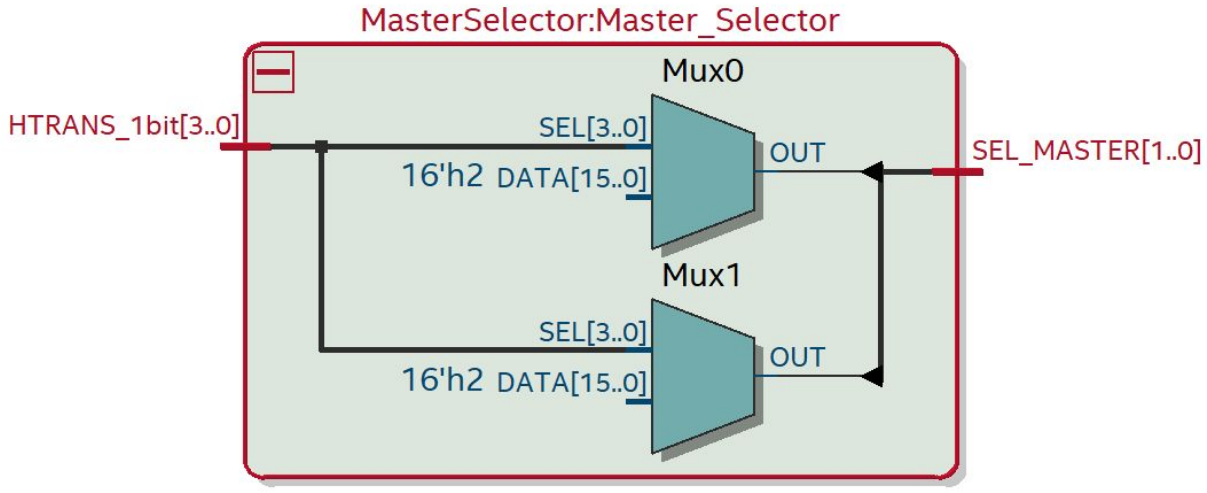



**Figura 9 -** RTL final do seletor de mestre.

### iv. Registrador

O registrador é necessário na parte interna do árbitro para que seja adicionado uma “etapa síncrona”, ou seja, os sinais do mestre selecionado para acessar o barramento só serão efetivamente passados para o mesmo depois da borda de subida do clock do registrador, o que permite que não haja problemas de sobreposição entre as transferências no barramento e outros. Com isso, o registrador é um componente básico - com RTL mostrado a seguir - e é instanciado

no módulo do árbitro com as entradas concatenadas e as saídas já separadas pelos bits certos de cada sinal do mestre com permissão de acessar o barramento.

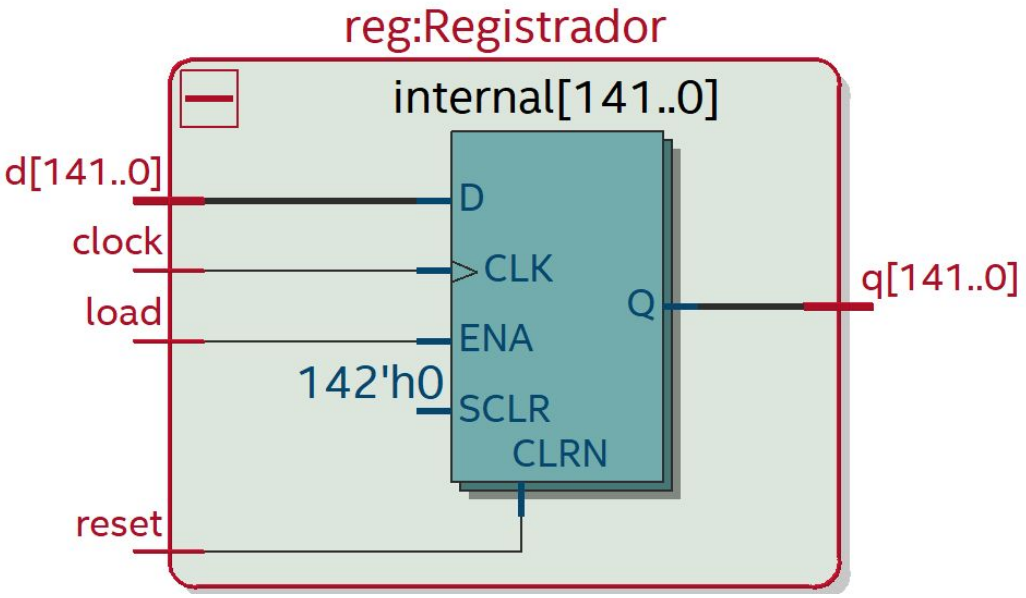



**Figura 10 -** RTL final do registrador.

### d. Multiplexador do AMBA

O multiplexador do barramento é importante para multiplexar o barramento de leitura de dados e responder sinais dos escravos para os mestres. Ou seja, ele tem como entradas as saídas dos 16 escravos, que são os sinais: HRDATA dado de leitura que foi selecionado pelo decodificador, HREADYOUT que quando em HIGH indica que a transferência foi finalizada no barramento e HRESP quem em HIGH indica erro na transferência e em LOW o status é “OK”. Além de ter como entrada o sinal MuxSelect seletor, cuja lógica é descrita no decodificador, pois é sua saída, e este sinal designa os valores dos sinais de qual escravo que será passado a frente no barramento como saída do componente do multiplexador, que são: HRDATA, HRESP e HREADY. Ou seja, como tem-se 16 escravos, utilizou-se de 4 bits:

- Para MuxSelect=’0000’ HRDATA, HRESP e HREADY recebem estes sinais do Escravo 0;
- Para MuxSelect=’0001’ HRDATA, HRESP e HREADY recebem estes sinais do Escravo 1;
- Para MuxSelect=’0010’ HRDATA, HRESP e HREADY recebem estes sinais do Escravo 2;

- … Até que para MuxSelect=’1111’ HRDATA, HRESP e HREADY recebem estes sinais do Escravo 15.

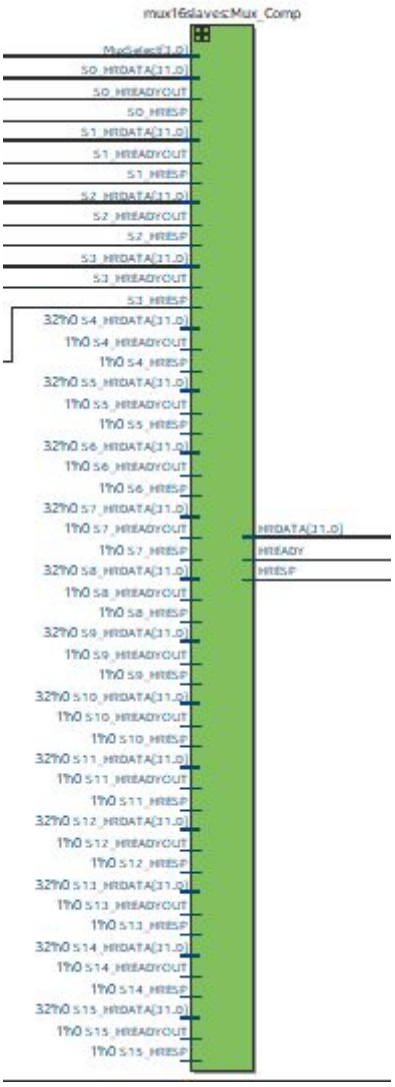



**Figura 11 -** RTL final do multiplexador do AMBA - consultar código para detalhes internos.

## e. Decodificador do AMBA

Esse módulo tem a função de decodificar cada transferência e oferecer o sinal de seleção para o escravo que está envolvido na transferência. Esse módulo tem como entradas a saída HADDR do mestre selecionado pelo árbitro e como saídas o selecionador do multiplexador e o sinal HSEL que indica que a transferência atual é pretendida para o escravo indicado. A lógica implementada para ativar o sinal HSEL representando que aquele escravo está com a transferência, é a seguinte, dado que podem ser 16 escravos, usa-se 4 bits:

- Para o sinal HADDR=’0000’, HSEL[0] recebe ‘1’, caso contrário recebe ‘0’;

- Para o sinal HADDR=’0001’, HSEL[1] recebe ‘1’, caso contrário recebe ‘0’;
- … até para o sinal HADDR=’1111’, HSEL[15] recebe ‘1’, caso contrário recebe ‘0’;

Já a lógica para selecionar os sinais de qual escravo que irá para o mestre, fica da seguinte maneira:

- Sinal MuxSelect<=”0000” quando HSEL[0]=’1’;
- Sinal MuxSelect<=”0001” quando HSEL[1]=’1’;
- ...até Sinal MuxSelect<=”1111” quando HSEL[15]=’1’;

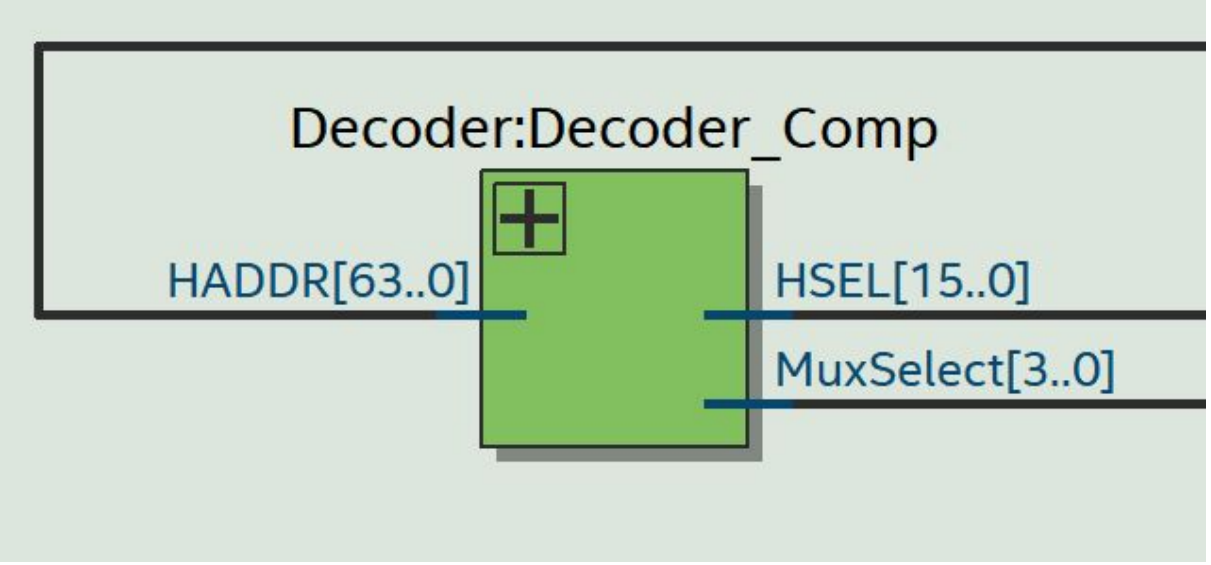



**Figura 12a -** Sinais do decodificador do AMBA.

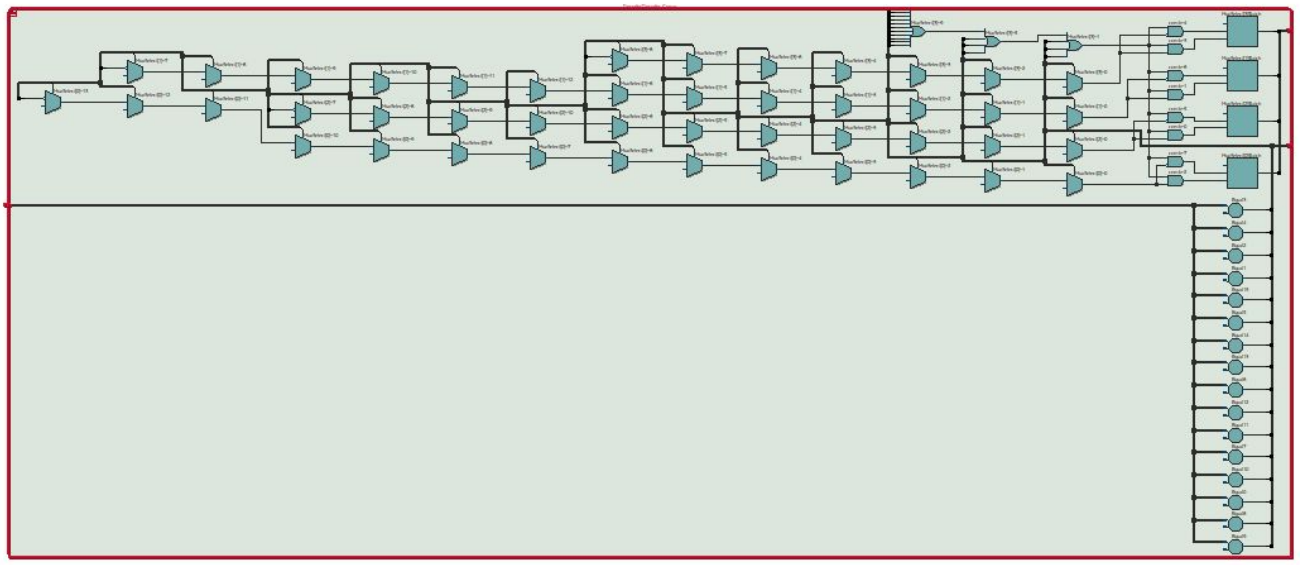


**Figura 12b -** RTL final do decodificador do AMBA - consultar código para detalhes internos.

# 4. Dependências

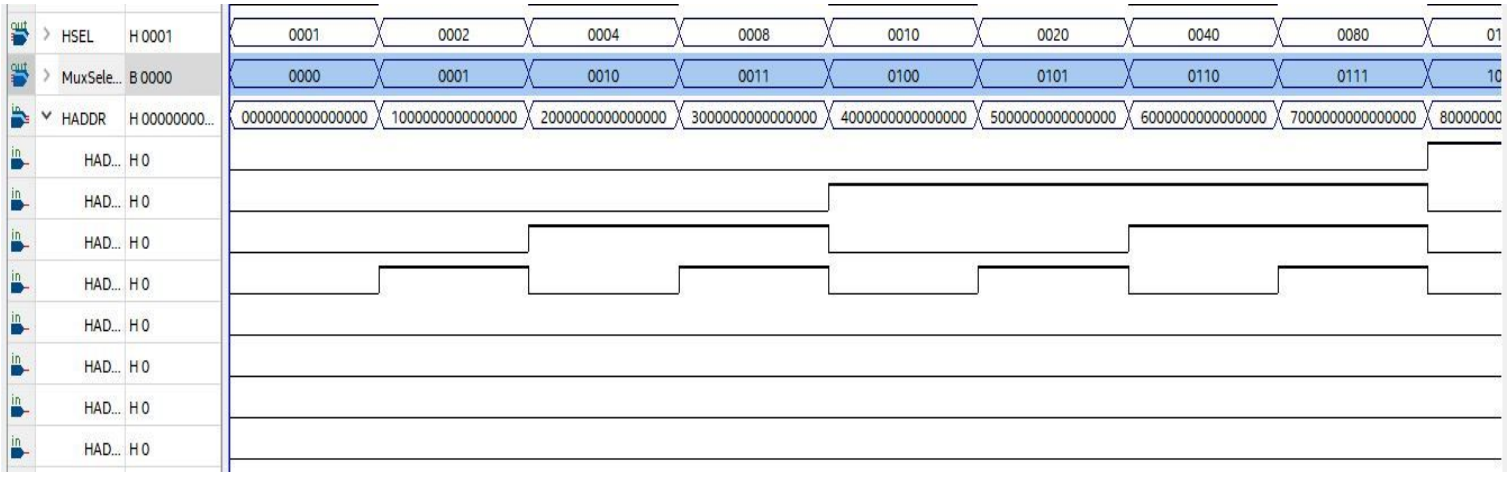
Como mencionado no planejamento do projeto, o mesmo supostamente não possui dependências. Ou seja, os módulos que são acrescentados externamente são os mestres e escravos, que podem ser derivados do trabalho dos outros grupos como: processador, memória de instruções, memória de dados, placa de vídeo, interface serial, e outros. Entretanto, se o projeto tentasse se adequar a cada interface feita pelos grupos, seria inviável finalizá-lo. Por isso, optou-se por implementar uma interface padrão para mestres e escravos, de forma que, com isso, os grupos que precisarem ligar seus módulos ao barramento, devem se adequar à especificação da interface produzida neste projeto, e não o contrário.

# 5. Testes Unitários

Cada componente utilizado no barramento foi testado separadamente, para isolar possíveis erros de implementação.

## a. Teste do Decodificador AMBA

**Esperado**: acionamento do **HSELx** de acordo com o endereço solicitado **HADDR**;




**Resultado**: OK

**Figura 13 -** Teste Unitário do Decodificador AMBA (1/2)

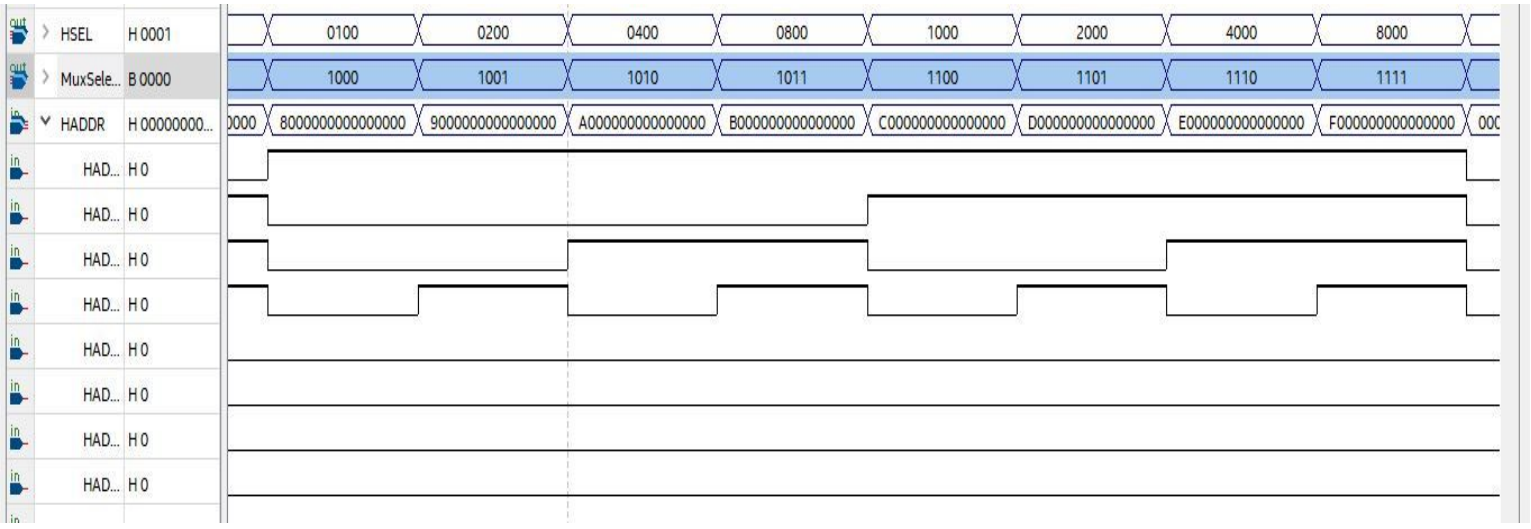



**Figura 14 -** Teste Unitário do Decodificador AMBA (2/2)

## b. Teste do Multiplexador AMBA

**Esperado**: selecionar as saídas do escravo correto para retornar aos mestres, de acordo com o seletor MuxSelector;

**Resultado**: OK

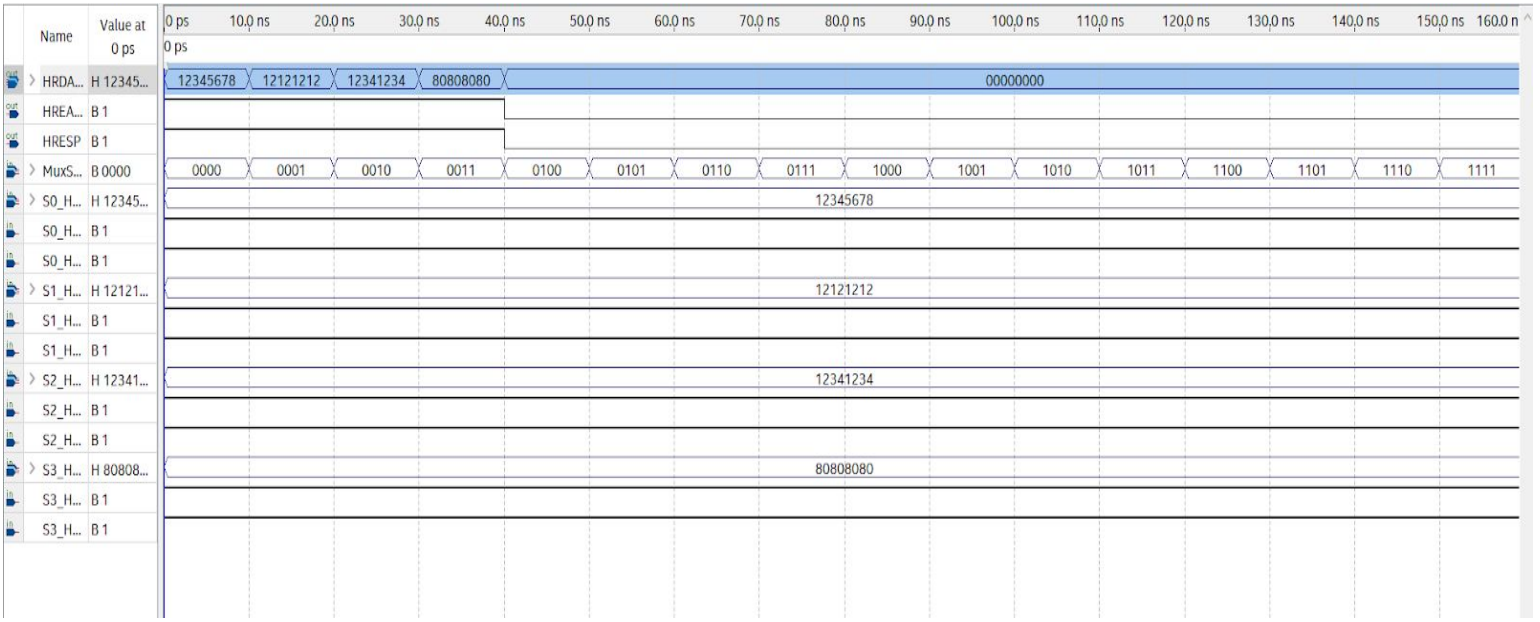



**Figura 15 -** Teste Unitário do Multiplexador AMBA

## c. Teste do Registrador

**Esperado**: carregar <u>sincronamente</u> a entrada **D** na saída **Q** quando o **LOAD** estiver acionado;

**Resultado**: OK

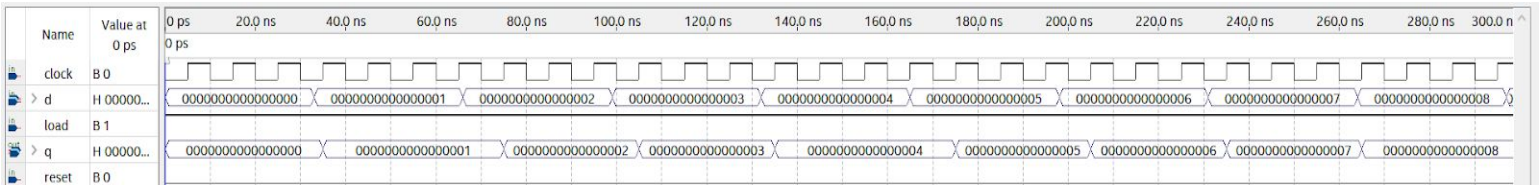



**Figura 16 -** Teste Unitário do Registrador do Árbitro

## d. Teste do Multiplexador de Mestres do Árbitro

**Esperado**: selecionar a saída do master correto através do sinal **selectMaster** (que é gerado no árbitro de acordo com a prioridade dos mestres).

**Resultado**: OK

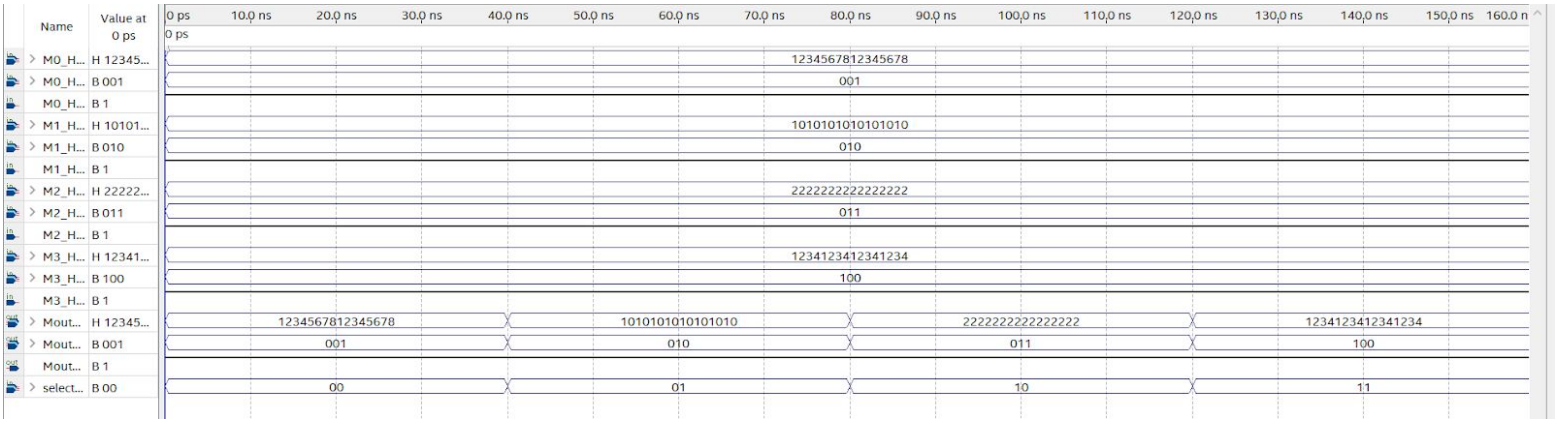



**Figura 17 -** Teste Unitário do Multiplexador de Mestres do Árbitro

## e. Teste do Decodificador de Mestres do Árbitro

**Esperado**: atualizar os sinais que retornam a cada um dos maester corretamente de acordo com o sinal **selectMaster** (que é gerado no árbitro de acordo com a prioridade dos mestres).

**Resultado**: OK

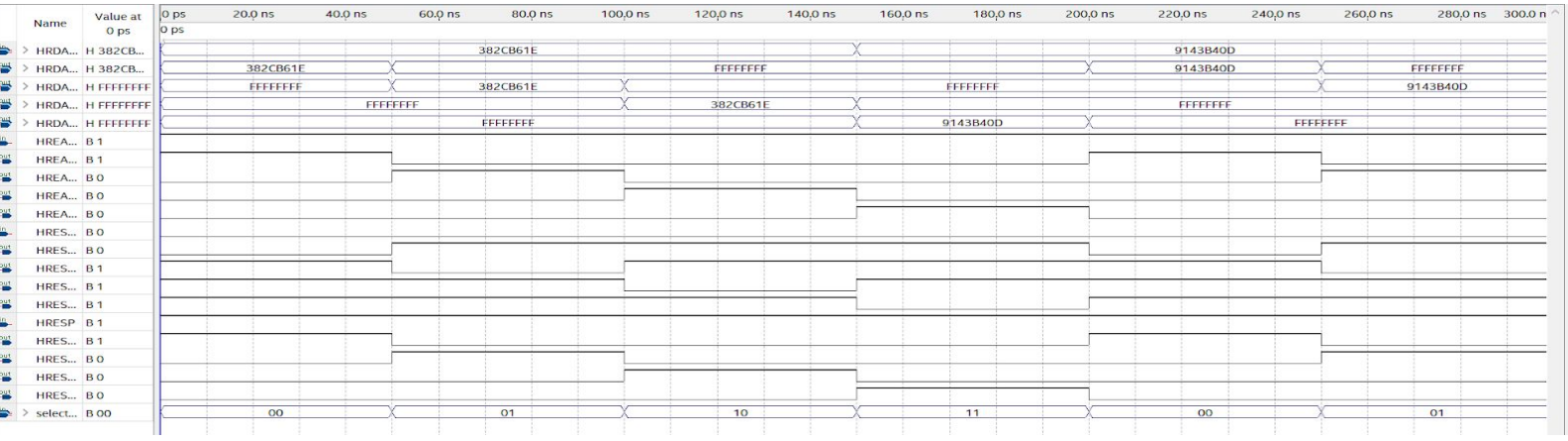


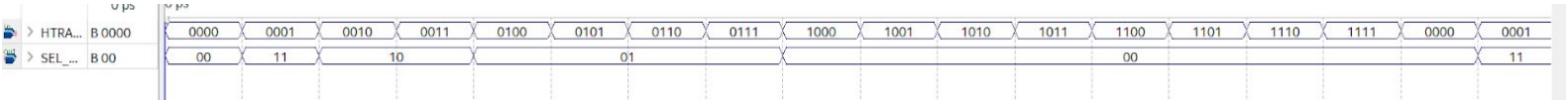

**Figura 18 -** Teste Unitário do Decodificador de Mestres do Árbitro

## f. Teste do Seletor de Mestres do Árbitro

**Esperado**: escolher (atualizando **SEL_MASTER**) o mestre que acessa o barramento através do sinal **HTRANS_CONCATENATED** que concatena todos os bits **HTRANS[1]** de cada mestre. A escolha é feita de acordo com a prioridade fixa, em que os mestres das primeiras posições são mais importantes.

**Resultado**: OK

**Figura 19 -** Teste Unitário do Decodificador de Mestres do Árbitro




## g. Teste do Árbitro

**Esperado**: selecionar as saídas do mestre mais prioritário (primeiras posições) quando há acesso concomitante por vários mestres

**Resultado**: OK

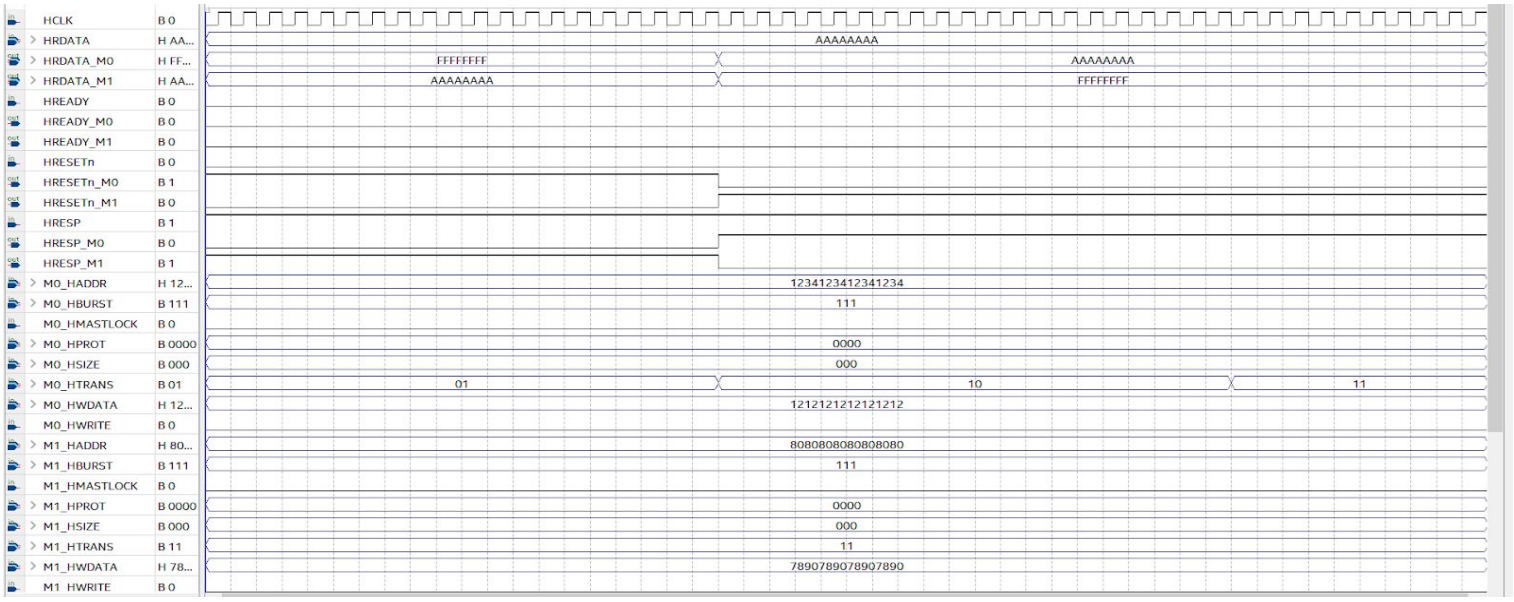



**Figura 20 -** Teste Unitário do Árbitro

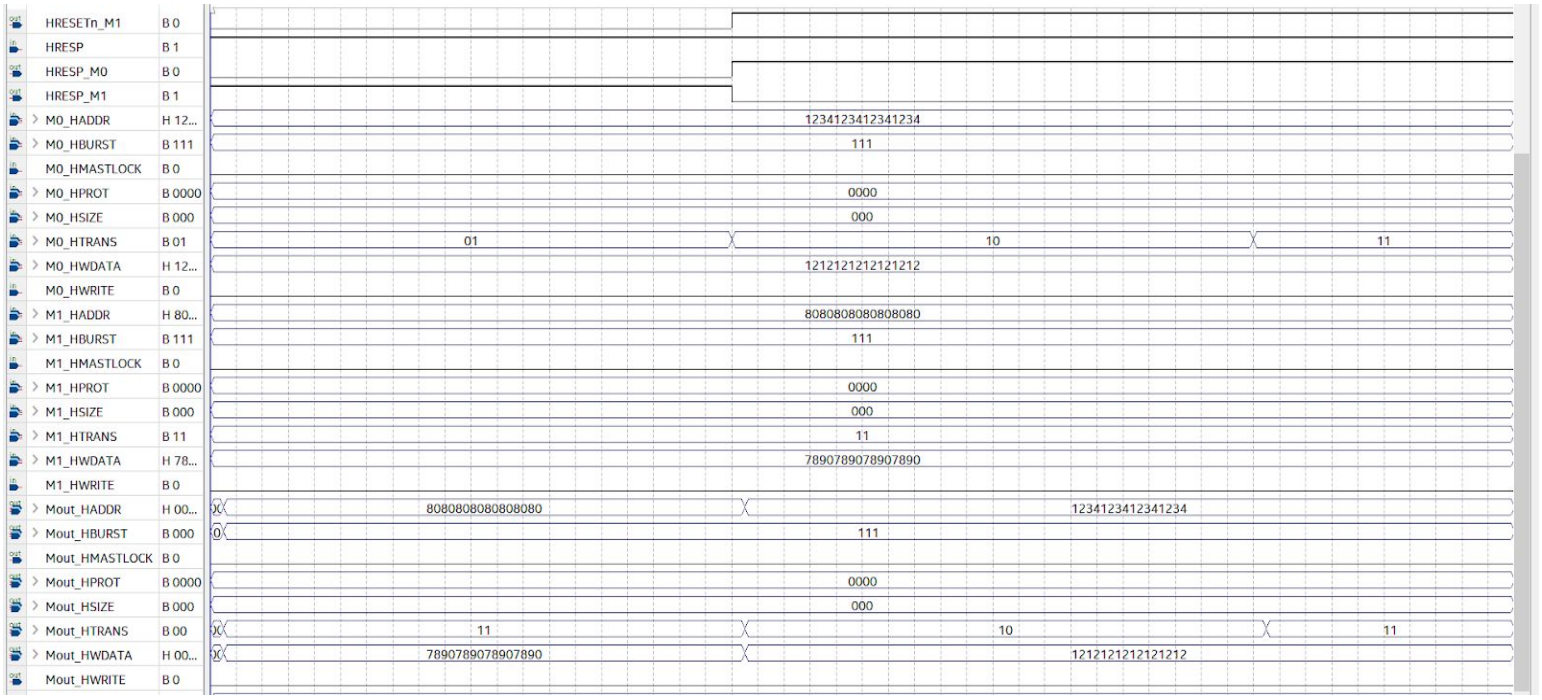



**Figura 21 -** 2º Teste Unitário do Árbitro

# 6. Simulações

## a. Simulação de Mestre

A simulação dos mestres é feita diretamente com a variação das entradas no barramento, controlando-as através do Waveform Simulator.

## b. Simulação de Escravo

Para fazer o mock-up de um escravo AMBA, utilizou-se uma máquina de estados simples, que simula passos de um escravo genérico. O escravo é iniciado num estado de IDLE e, caso seja selecionado pelo barramento (através do **H_SELx**), ele parte para o estado de WRITING ou READING (dependendo do sinal **H_WRITE**). Após um tempo arbitrário de leitura/escrita, a máquina vai para o estado RESPONSE, onde aciona os sinais de resposta para o mestre. Por fim, o escravo volta para o estado de ESPERA, por ter concluído o acesso requisitado.

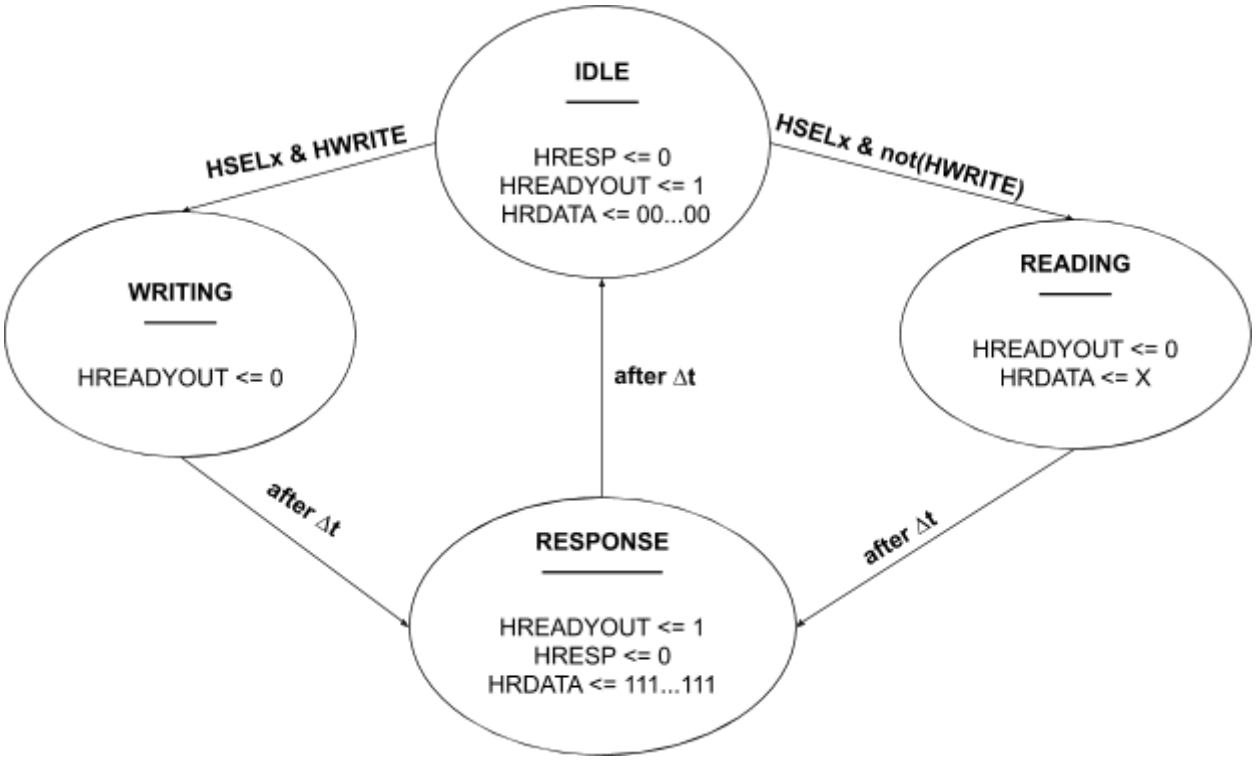



**Figura 22 -** Diagrama de Estados do MockUp de Escravo

Este componente foi testado para algumas entradas aleatórias de HADDR, variando o sinal de HWRITE para simular quando ‘0’ a leitura e quando ‘1’ a escrita no escravo.

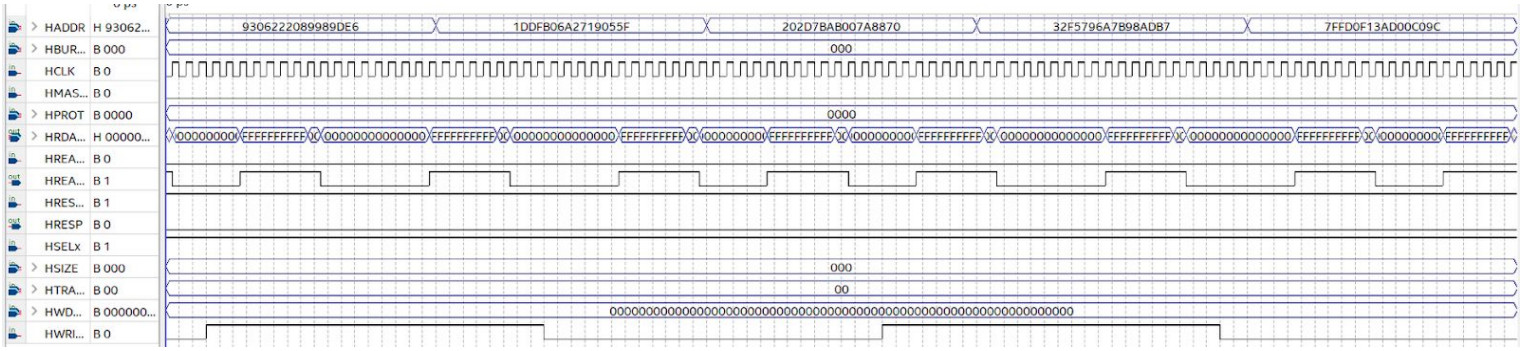



**Figura 23 -** Simulação da FSM do escravo

O ponto principal a ser notado e que será utilizado nos próximos testes é o que indica que um escravo está realizando operações, seja de escrita ou leitura. Há variação na saída de HRDATA, mas porém, o sinal mais perceptível que indica atividade do escravo é o seu HREADYOUT, que na carta de tempo fica 2 sinais abaixo do HRDATA, que varia sempre que está ativo. No caso em que o escravo

está inativo, este sinal fica sempre ativo, indicando que ele está pronto para realizar operações, porém, quando fica inativo, isto indica que uma operação está ocorrendo neste escravo.

## c. Caso Básico de Leitura e Escrita (1 mestre e 3 escravos)

Nessa simulação, é validado o caso mais simples de uso do barramento: um mestre lendo endereços de diferentes escravos.

1. Primeiramente, um mestre inicia a comunicação solicitando a leitura (HWRITE = 0) de um endereço qualquer do primeiro escravo (0000xxx...xxx);
2. Em seguida, o primeiro escravo passa para o estado de leitura e seu sinal **HREADYOUT_0** passa para 0, começando a variar enquanto ocorrem operações de leitura. Além disso, no estado de leitura da máquina de estados, a saída HRDATA é colocada em $8000\ldots.0000_{16}$;
3. Após 200 ns (metade do tempo mostrado no gráfico), muda-se o endereço HADDR para um endereço que se inicie por 0010, selecionando o escravo de número 2, alterando também a operação para escrita (HWRITE = 1). Com isso, pode-se notar então que após o último ciclo de leitura do escravo 0, seu sinal HREADYOUT_0 fica sempre 1 e, então, o sinal **HREADYOUT_2** passa a variar, mostrando que o escravo 2 está sendo escrito. Além disso, para mostrar que a operação é de escrita, foi definido na máquina de estados que o estado de escrita coloca a saída HRDATA em $0000...0001_{16}$.

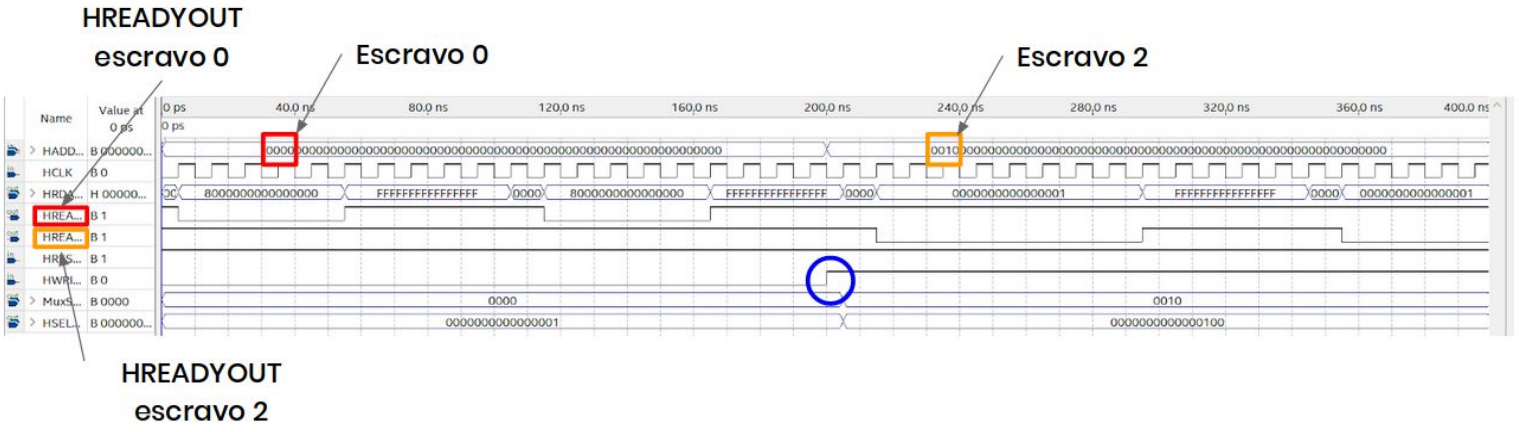



**Figura 24 -** Simulação das operações de leitura e escrita por um mestre em dois escravos.

## d. Concorrência de Mestres (2 mestres e 3 escravos)

Nessa simulação, testa-se a arbitragem de mestres ao ocorrer concorrência de acesso ao barramento. Os mestres têm sua prioridade de acordo com sua posição, de forma que o Mestre 0 tem prioridade 0 e o Master 1 tem prioridade 1. Quanto menor o valor da prioridade, mais prioritário é o mestre.

1. Primeiramente, o Mestre 0 solicita a leitura (HWRITE = 0) de um endereço do Escravo 0 (HADDR iniciado por “0000”), e ao mesmo tempo, o Mestre 1 solicita a leitura de um endereço do Escravo 2 (HADDR iniciado por “0010”). Ambos iniciam a transferência com HTRANS = “11”, indicando que ambos querem realizar a transferência;
2. Em seguida, o árbitro seleciona o mestre mais prioritário (Mestre 0) e o acesso ao Escravo 0 é concedido à ele, como pode-se notar devido à variação do sinal **HREADYOUT_0**. Enquanto isso, o sinal **HREADYOUT_2** se mantém ativo;
3. Após metade do tempo da simulação, o mestre 0 desliga o sinal de HTRANS, indo para “00”. Não há variação no mestre 1 e, portanto, este passa a ter prioridade de acesso ao barramento, podendo realizar transferência com o escravo 2. Como pode-se notar na carta de tempo, o sinal **HREADYOUT_2** passa a variar, mostrando que a operação do mestre 1 é efetiva.
4. Vale notar neste teste que a mudança de mestre e escravo só se torna efetiva em uma borda de subida do clock, impedindo possíveis problemas de sincronia. Como já dito, isto ocorre devido ao registrador inserido na saída do árbitro.

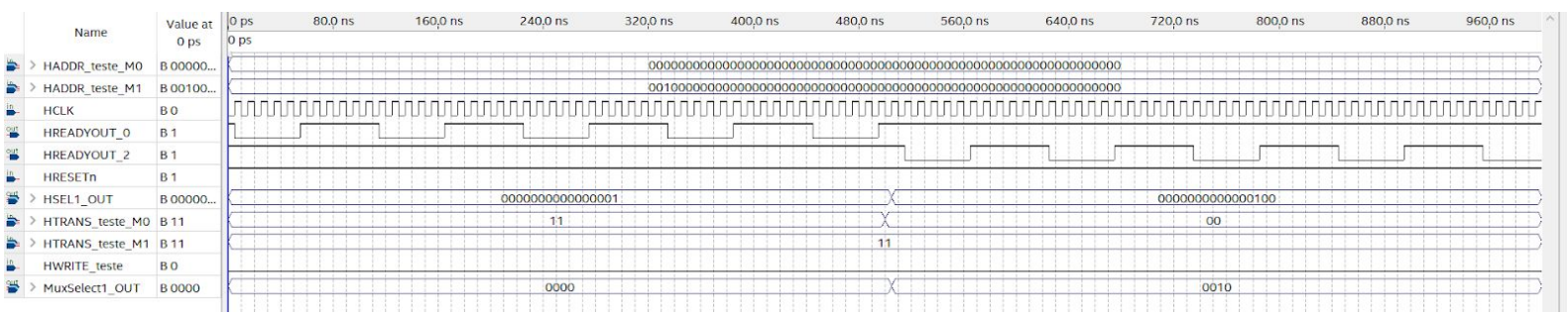



**Figura 25** - teste concorrência de mestres.

# 7. Conclusão

Primeiramente é importante destacar que antes de iniciar o desenvolvimento do projeto fez-se necessário todo um estudo da parte teórica de barramentos, parte essa que também foi contemplada brevemente ao longo do relatório, porque era um tema até então desconhecido pelo grupo e pela turma. E, então, após tal estudo, a maior dificuldade foi transformá-lo em prática, ou seja, no código VHDL em si. Com isso, é possível concluir facilmente que o projeto contribuiu amplamente para o aprendizado dos conceitos de barramento como um todo, pois a teoria por si só soava abstrata mas com a prática tornou-se palpável e compreensível.

Não obstante, pôde-se perceber ao longo da concepção do projeto que, antes de iniciar efetivamente qualquer linha de código, era preciso tomar diversas decisões e fazer escolhas de projeto. Com isso, é fácil de enxergar como o projeto se assemelha aos tantos outros feitos fora da sala de aula, no mercado, pois bem como estes, o barramento possuía algumas maneiras diferentes de ser construído.

Desta forma, a primeira escolha foi com relação a qual tipo de barramento seria implementado, se ASB, APB ou AHB, e optou-se pelo AHB, como dito anteriormente neste documento, devido a maior facilidade de encontrar documentação, por ser um barramento de alta velocidade e que, de certa maneira, permite que sejam interconectados vários mestres. A segunda grande escolha, que poderia modificar o projeto quase completamente em termos de código, e de fato mudou, foi com relação a quantos mestres poderiam ser interligados ao barramento. O projeto iniciou-se para um mestre só, mas, como sugerido pelo professor, alterou-se para que pudessem ser colocados vários mestres.

Esta última escolha, apesar de mais certeira, trouxe uma série de problemas durante o processo de construção do barramento, porque apesar de a implementação já estar quase pronta para que fosse ligado somente um mestre, para adicionar mais mestres da maneira mais correta, utilizando componentes genéricos, adicionou-se uma complexidade extra e o grupo não conseguiu fazer com que funcionasse desta maneira. Por isso, ao final decidiu-se por deixar que fossem aceitos até quatro mestres e dezesseis escravos, com o porém de que o

usuário final deveria, então, colocar as entradas dos sinais de mestres e escravos que não fosse utilizar em aberto. Isso facilitou a implementação do ponto de vista de sintaxe - dado que o grupo teve dificuldade com o genérico - mas dificultou do ponto de vista manual e visual, porque o circuito adquiriu uma complexidade em número de sinais que não havia sido prevista anteriormente.

Por fim, pode-se concluir que o funcionamento do barramento foi um sucesso, apesar das dificuldades ao longo o projeto, ao final o grupo conseguiu não só implementar todos os componentes do barramento, como testar cada componente separadamente e, então, simular o funcionamento de todos os componentes ao mesmo tempo. E, assim, demonstrou-se através de testes e simulações que o barramento teve seu funcionamento correto, com operações de leitura, escrita, e com um ou mais mestres tentando acessar ao barramento concorrencialmente.

# 8. Referências


- http://www.univasf.edu.br/~romulo.camara/novo/wp-content/uploads/2013/11/Barramento-AMBA-.pdf
- AMBA 3 AHB-Lite Protocol v1.0 Specification - ARM [http://www.eecs.umich.edu/courses/eecs373/readings/ARM_IHI0033A_AMBA_AHB-Lite_SPEC.pdf]
- https://slideplayer.com.br/slide/352229/