Gabriel Dias De Oliveira (176495)

NOTA:
Reset = 1… NÃO limpa a entrada | Reset = 0… Limpa a entrada.
Enable = 1… PERMITE uma nova entrada | Enable = 0… NÃO permite uma nova entrada.
calcula = 1… CALCULA | calcula = 0… NÃO calcula.

Multiplexador		
A	in STD_LOGIC	Primeira entrada do multiplexador.
B	in STD_LOGIC	Segunda entrada do multiplexador.
S	in STD_LOGIC	Seletor do multiplexador. Se S = 1, A é selecionado | S = 0, B é selecionado.
f	out STD_LOGIC	Saída selecionada conforme o sinal de seleção (S).

Implementar o multiplexador não foi um grande desafio visto que sua estrutura e lógica são básicas. Um eventual ponto que gerou certa incerteza foi em como construi-lo de modo a garantir seu reuso nos diferentes casos do projeto (multiplexador de 8 bits, 16, etc).
Podemos ver que o circuito funciona corretamente pelos sinais apresentados, vemos claramente que o circuito escolhe A quando S = 1 e B quando S = 0. Portanto, temos o multiplexador operando corretamente.

Multiplexador_8		
Multiplicador	in STD_LOGIC_VECTOR(7 DOWNTO 0)	Valor inicial do multiplicador, ou seja, número fornecido na entrada. Primeira entrada do multiplexador.
Multiplicador_Deslocado	in STD_LOGIC_VECTOR(7 DOWNTO 0)	Valor do multiplicador já deslocado (operação em andamento). Segunda entrada do multiplexador.
S	in STD_LOGIC	Seletor do multiplexador. Se S = 1, Multiplicador é selecionado | S = 0, Multiplicador_Deslocado é selecionado.
multiplexadorout	signal STD_LOGIC_VECTOR(7 DOWNTO 0)	Valores obtidos da seleção do multiplexador 2x1.

f	out STD_LOGIC_VECTOR(7 DOWNTO 0)	Saída selecionada conforme o sinal de seleção (S).

Como já foi dito, a implementação lógica do multiplexador não apresentou grandes problemas, entretanto, ter a ideia de combinar os mux 2x1 de modo a gerar uma saída de 8 bits foi a parte mais complicada de todo o processo.
Podemos ver pela saída que o circuito funciona corretamente, pois, sabendo que o mux 2x1 funciona temos que todos os bits de apenas uma das entradas será escolhido e, desse modo, nosso f sempre tem a resposta correta.

Multiplexador_16		
A	in STD_LOGIC_VECTOR(15 DOWNTO 0)	Valor inicial do multiplicando, ou seja, número fornecido na entrada. Primeira entrada do multiplexador.
B	in STD_LOGIC_VECTOR(15 DOWNTO 0)	Valor do multiplicador já deslocado (operação em andamento). Segunda entrada do multiplexador.
S	in STD_LOGIC	Seletor do multiplexador. Se S = 1, Multiplicando é selecionado (A) | S = 0, Multiplicando deslocado é selecionado (B).
multiplexadorout	signal STD_LOGIC_VECTOR(15 DOWNTO 0)	Valores obtidos da seleção do multiplexador 2x1.
f	out STD_LOGIC_VECTOR(15 DOWNTO 0)	Saída selecionada conforme o sinal de seleção (S).

Analogamente ao multiplexador de 8 bits, o multiplexador de 16 bits tem sua lógica muito semelhante. Ele faz uso do mux 2x1 para selecionar os bits da entrada escolhida.
Isso fica evidente na curva do circuito, onde para S = 0 ou S = 1, temos a seleção dos bits de apenas uma das entradas, logo, nosso f será igual a A ou B dependendo do sinal S.

DeslocadorEsq		
N	GENERIC INTEGER	Constante que define o tamanho da entrada e saída, tornando a manutenção do arquivo melhor. Nesta situação N =

		16.
input_vector	in STD_LOGIC_VECTOR(N-1 DOWNTO 0)	Valor do multiplicando atual que está vigente no circuito. Isto é, seja a entrada inicial ou alguma entrada já deslocada.
w	in STD_LOGIC	Sinal que define como o espaço deslocado será preenchido.
output_vector	out STD_LOGIC_VECTOR(N-1 DOWNTO 0)	Vetor (Multiplicando) deslocado uma posição para a esquerda.

Os deslocadores foram um dos componentes mais complicados de se elaborar, isso porque exigiram uma lógica / conhecimento de sintaxe de VHDL um pouco maior que os demais.
Uma decisão tomada durante a construção do circuito, foi permitir a entrada do sinal “w”, que substitui o valor deslocado, evitando assim vazios que poderiam gerar eventuais erros - por padrão, neste projeto w = 0 sempre, mas ao permitir que o “w” seja uma entrada, isso flexibiliza a utilização do sistema, pois podemos optar, eventualmente, por inserir o valor “1”, gerando algum outro comportamento / aplicação do deslocador. Observamos também que o deslocador deve possuir 16 bits, pois, somente assim, a multiplicação funcionará corretamente.
Podemos ver claramente que as saídas são as entradas, porém com um bit deslocado para esquerda e com os valores deslocados substituídos por 0.

DeslocadorDir		
N	GENERIC INTEGER	Constante que define o tamanho da entrada e saída, tornando a manutenção do arquivo melhor. Nesta situação N = 8.
input_vector	in STD_LOGIC_VECTOR(N-1 DOWNTO 0)	Valor do multiplicador atual que está vigente no circuito. Isto é, seja a entrada inicial ou alguma entrada já deslocada.
w	in STD_LOGIC	Sinal que define como o espaço deslocado será preenchido.
output_vector	out STD_LOGIC_VECTOR(N-1 DOWNTO 0)	Vetor (Multiplicador) deslocado uma posição para a direita.

Semelhante ao deslocador à esquerda, o maior desafio da construção do deslocador a direita foi a sintaxe do VHDL, ou seja, como propriamente escrever o código do circuito.
Novamente temos a presença do sinal “w” que serve para substituir o bit deslocado.
Também fica claro que o circuito funciona, uma vez que as saídas são justamente as entradas deslocadas um bit para direita.

FullAdder		
Cin	in STD_LOGIC	Sinal que simboliza o “resto” de uma soma - Deve ser levado em consideração na hora de fazer a propagação da soma.
x	in STD_LOGIC	Primeira entrada para a soma.
y	in STD_LOGIC	Segunda entrada para a soma.
s	out STD_LOGIC	Sinal que representa a soma resultante referente às entradas (Cin, x e y).
Count	out STD_LOGIC	Eventual resto gerado pela soma.

A implementação do full adder foi fácil, uma vez que ele implementou a lógica básica de soma binária. Para o contexto do projeto, o FullAdder foi pensado para ser reutilizado e expansível, de modo a garantir a criação do somador de 16 bits.
Vemos pela saída do circuito que a soma binária está coerente com o esperado, quando temos a soma de 1 + 1, geramos um *carry out,* as saídas “s” estão de acordo com o esperado.

BitsAdder		
Multiplicando	in STD_LOGIC_VECTOR(15 DOWNTO 0)	Valor do multiplicando atual.
Produto	in STD_LOGIC_VECTOR(15 DOWNTO 0)	Valor atual do produto
s	out STD_LOGIC_VECTOR(15 DOWNTO 0)	Saída resultante da soma dos respectivos 16 bits de cada valor.
COUT_ADDER	signal STD_LOGIC_VECTOR(1 TO 16);	Este signal é responsável por armazenar os carry outs de cada operação realizada nos estágios do FullAdder.

O BitsAdder é a aplicação prática e expandida do FullAdder, ou seja, ele é composto por 16 FullAdders, levando em consideração a propagação dos *carry out.*
Podemos ver que o circuito opera corretamente, uma vez que todas as somas realizadas estão condizentes com o que era esperado.

FlipFlop		
D	in STD_LOGIC	Valor de um “bit” da entrada no momento atual - pode ou não ser “aceita” pelo flipflop dependendo dos sinais.
Enable	in STD_LOGIC	Sinal lógico que permite ou não o recebimento de novas entradas (caso haja alguma alteração e o clock esteja ativo).
Reset	in STD_LOGIC	Sinal lógico que “limpa” o valor armazenado de um flipflop.
Clock	in STD_LOGIC	Sinal que “controla” o funcionamento do flipflop, permitindo (ou não) a alteração do valor armazenado.
Q	out STD_LOGIC	Armazenamento do novo bit, gerando assim uma nova saída para o registrador.

O flipflop foi uma parte fundamental do projeto, pois é a base para todos os outros registradores, desse modo, exigiu uma atenção especial entretanto, o maior problema para sua construção foi perceber como o sinal de reset, enable e o utras alterações deveriam ser “sentidas” pelo flipflop.
Como o FlipFlop é a base para todos os outros registradores, o seu funcionamento poderá ser visto nas simulações seguintes. Sendo que seu comportamento básico pode ser descrito como: Se o valores do enable está ativo e o reset desativado, quando haver um pulso de clock e uma alteração em D, nosso flipflop trocará o valor armazenado em Q, caso reset esteja ativo, Q será “limpo” e se o enable estiver desligado, então o FlipFlop não troca seu valor.

RegistradorMultiplicador		
Multiplicador	in STD_LOGIC_VECTOR(7	Valor do multiplicador atual.

	DOWNTO 0)	
Enable	in STD_LOGIC	Sinal lógico que permite ou não o recebimento de novas entradas (caso haja alguma alteração e o clock esteja ativo).
Reset	in STD_LOGIC	Sinal lógico que “limpa” o valor armazenado no registrador.
Clock	in STD_LOGIC	Sinal que “controla” o funcionamento do registrador, permitindo (ou não) a alteração do valor armazenado.
Q	out STD_LOGIC_VECTOR(7 DOWNTO 0)	Armazenamento do multiplicador no seu novo estado, para que a operação possa prosseguir.
flipflopclock	signal STD_LOGIC_VECTOR(7 DOWNTO 0)	As posições do flipflopclock armazenam as alterações vindas dos flipflops, ou seja, quando a entrada do flip flop é alterada, os “bits” que formam o multiplicador vão se alterando um a um.

O principal desafio da construção dos registradores foi sincronizá-lo em relação ao clock e alterações das entradas, ao longo do desenvolvimento fui percebendo vários ajustes que deveria ser feitos para que as alterações fossem “sentidas” pelos registradores da maneira correta, sem causar atrasos ~ erros na multiplicação.
Podemos ver que as saídas funcionam corretamente, quando o enable está ativo e o reset desativado, nós temos a saída sendo igual ao D e trocando de valor conforme o clock e D se alteram. Também, é possível ver que quando o reset é habilitado, a entrada Q é zerada - tudo funcionando conforme era esperado.

RegistradorMultiplicando		
Multiplicando	in STD_LOGIC_VECTOR(15 DOWNTO 0)	Valor do multiplicando atual.
Enable	in STD_LOGIC	Sinal lógico que permite ou não o recebimento de novas entradas (caso haja alguma alteração e o clock esteja ativo).

Reset	in STD_LOGIC	Sinal lógico que “limpa” o valor armazenado no registrador.
Clock	in STD_LOGIC	Sinal que “controla” o funcionamento do registrador, permitindo (ou não) a alteração do valor armazenado.
Q	out STD_LOGIC_VECTOR(15 DOWNTO 0)	Armazenamento do multiplicando no seu novo estado, para que a operação possa prosseguir.
flipflopclock	signal STD_LOGIC_VECTOR(15 DOWNTO 0)	As posições do flipflopclock armazenam as alterações vindas dos flipflops, ou seja, quando a entrada do flip flop é alterada, os “bits” que formam o multiplicando vão se alterando um a um.

Assim como dito anteriormente, o principal desafio da construção dos registradores foi sincronizá-lo para evitar erros na multiplicação.
Podemos ver que as saídas funcionam corretamente, quando o enable está ativo e o reset desativado, nós temos a saída sendo igual ao D e trocando de valor conforme o clock e D se alteram. Também, é possível ver que quando o reset é habilitado, a entrada Q é zerada - tudo funcionando conforme era esperado.

RegistradorProduto		
Produto	in STD_LOGIC_VECTOR(15 DOWNTO 0)	Valor do produto atual.
Enable	in STD_LOGIC	Sinal lógico que permite ou não o recebimento de novas entradas (caso haja alguma alteração e o clock esteja ativo).
Reset	in STD_LOGIC	Sinal lógico que “limpa” o valor armazenado no registrador.
Clock	in STD_LOGIC	Sinal que “controla” o funcionamento do registrador, permitindo (ou não) a alteração do valor armazenado.

Q	out STD_LOGIC_VECTOR(15 DOWNTO 0)	Armazenamento do produto no seu novo estado, para que a operação possa prosseguir.
flipflopclock	signal STD_LOGIC_VECTOR(15 DOWNTO 0)	As posições do flipflopclock armazenam as alterações vindas dos flipflops, ou seja, quando a entrada do flip flop é alterada, os “bits” que formam o produto vão se alterando um a um.

Uma decisão tomada para o projeto como um todo, foi construir um registrador específico para o produto, isso, pois, é notável que a lógica dele é idêntica a do multiplicador, porém, por ser um componente importante para a construção de todo o circuito, a decisão foi de inserir um componente próprio para o produto, permitindo assim instância-lo e tornar o fluxo/identificação muito mais simples.
Como já testamos os outros dois registradores, o teste de circuito é um pouco mais simples e valoriza principalmente a funcionalidade do enable. Notamos que quando o enable é desativado, não existe nenhuma alteração do valor armazenado, isso é fundamental para o controle do nosso multiplicador.

UnidadeControle		
Clock	in STD_LOGIC	Sinal responsável por indicar alteração no sistema - responsável por indicar o momento de checar se deve existir a transição de algum estado.
Resetn	in STD_LOGIC	Sinal que indica que a operação deve ser reiniciada .
calcula	in STD_LOGIC	Sinal lógico que habilita a execução de uma nova operação (ou não).
Multiplicando	in STD_LOGIC_VECTOR(15 DOWNTO 0)	Valor atual do multiplicando - utilizando para checagem da transição de estados.
Multiplicador	in STD_LOGIC_VECTOR(7 DOWNTO 0);	Valor atual do multiplicador - utilizando para checagem da transição de estados.
done	out STD_LOGIC	Saída que indica quando a operação foi finalizada.

Eentradas	out STD_LOGIC	Controla o enable dos registradores das entradas - responsável por garantir que os valores recebidos sigam as transições de estados corretas.
Eproduto	out STD_LOGIC	Controla o enable do registrador dos produtos - responsável por garantir que os valores recebidos sigam as transições de estados corretas.
Sentradas	out STD_LOGIC	Controla a escolha do multiplexador das entradas, permitindo escolher o valor da entrada ou deslocado.
Sproduto	out STD_LOGIC	Controla o valor do multiplexador do produto.
y	signal STATE_TYPE	Armazena qual é o estado atual do circuito.

A Unidade de Controle foi o componente mais trabalhoso a ser desenvolvido, isso, pois, possui uma lógica “grande” e que exige muita atenção aos detalhes, um grande desafio foi elaborar o projeto da máquina de estado, de modo a garantir que os sinais estivessem coerentes com o pulso do clock e/ou que os registradores / multiplexadores conseguissem sentir as alterações dos sinais. Entretanto, com tudo isso organizado e esquematizado, a implementação em VHDL foi mais fácil.

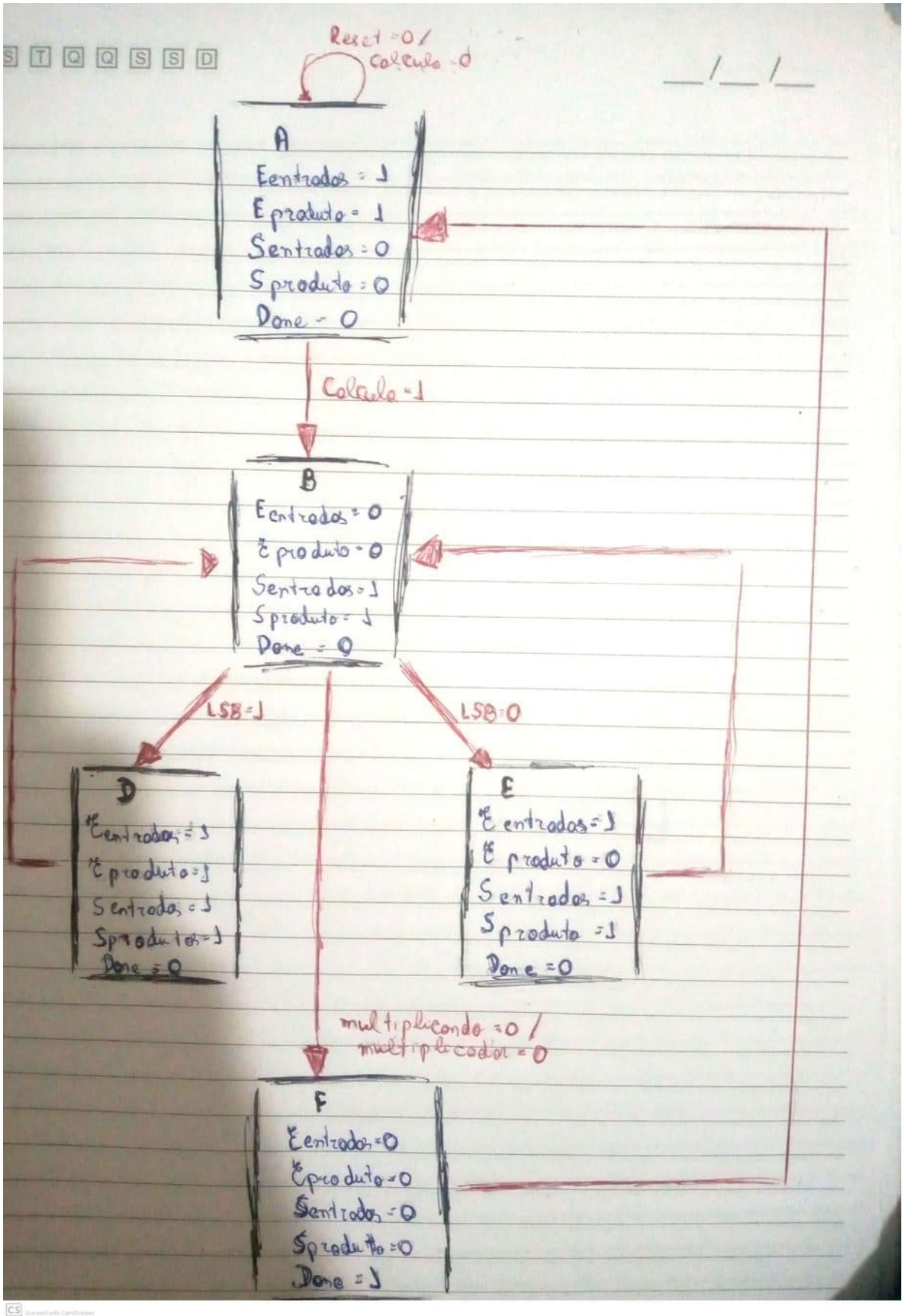
A máquina de Moore foi escolhida, pois, mesmo gerando mais estados que uma máquina Mealy, o transicionamento entre estados ficou mais claro, sendo assim, foi necessário pensar somente no critério para fazer a passagem de um estado a outro e, depois, associar as saídas corretas a cada estado - não carregar as saídas no meio das entradas facilitou a visualização de como o circuito deveria ficar.

A simulação de ondas permite ver que as transições de estados ocorrem conforme esperado, para cada evento de transição (por exemplo, se LSB = 1 temos um transicionamento, já se LSB = 0 temos outro), quando temos uma das entradas zeradas, vemos que o circuito vai para o estado final - assim como esperado.

Explicação dos estados	
A	Entradas estão sendo recebidas, os multiplexadores devem selecionar a entrada inserida e o mux do produto deve zerar seu valor. Enables ligados.
B	Valores já posicionados, temos agora que os sinais dos mux podem mudar, para pegar somente os valores deslocados e soma parcial. Notamos que os enables não estão ligados, pois ainda não completamos um "ciclo" com os valores atuais.
D	Se LSB = 1, então, uma soma ocorrerá e todos os valores, inclusive o produto, serão trocados, por isso todos os enables estão ligados.
E	Se LSB = 0, as entradas mudaram de valor, mas não haverá uma soma, então não existe nenhuma mudança no produto.
F	Se multiplicador / multiplicando = 0, então a soma foi finalizada, todas os valores são “desligados” e o done = 1 (fim).

S T Q Q S S D

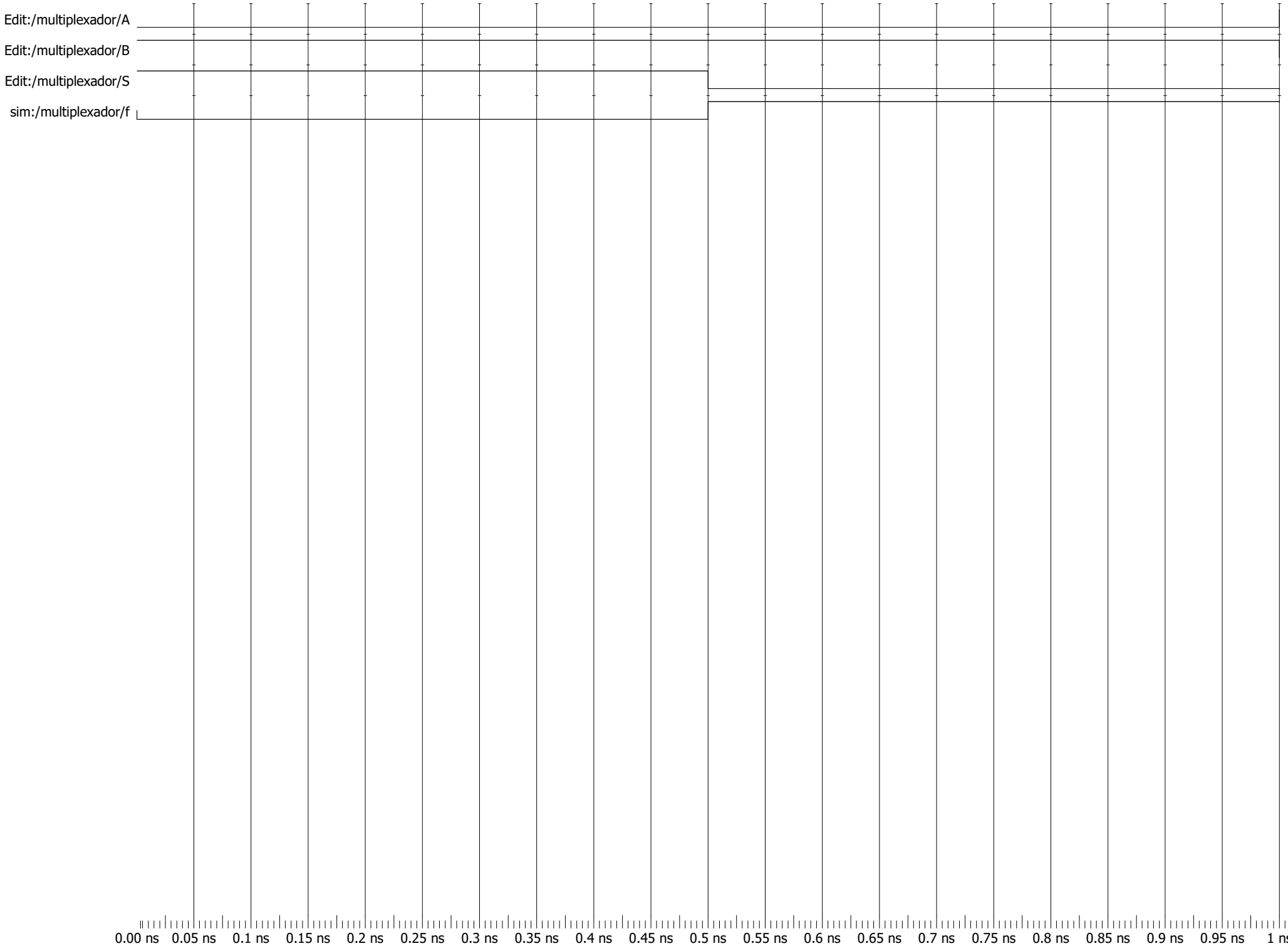
__/__/__

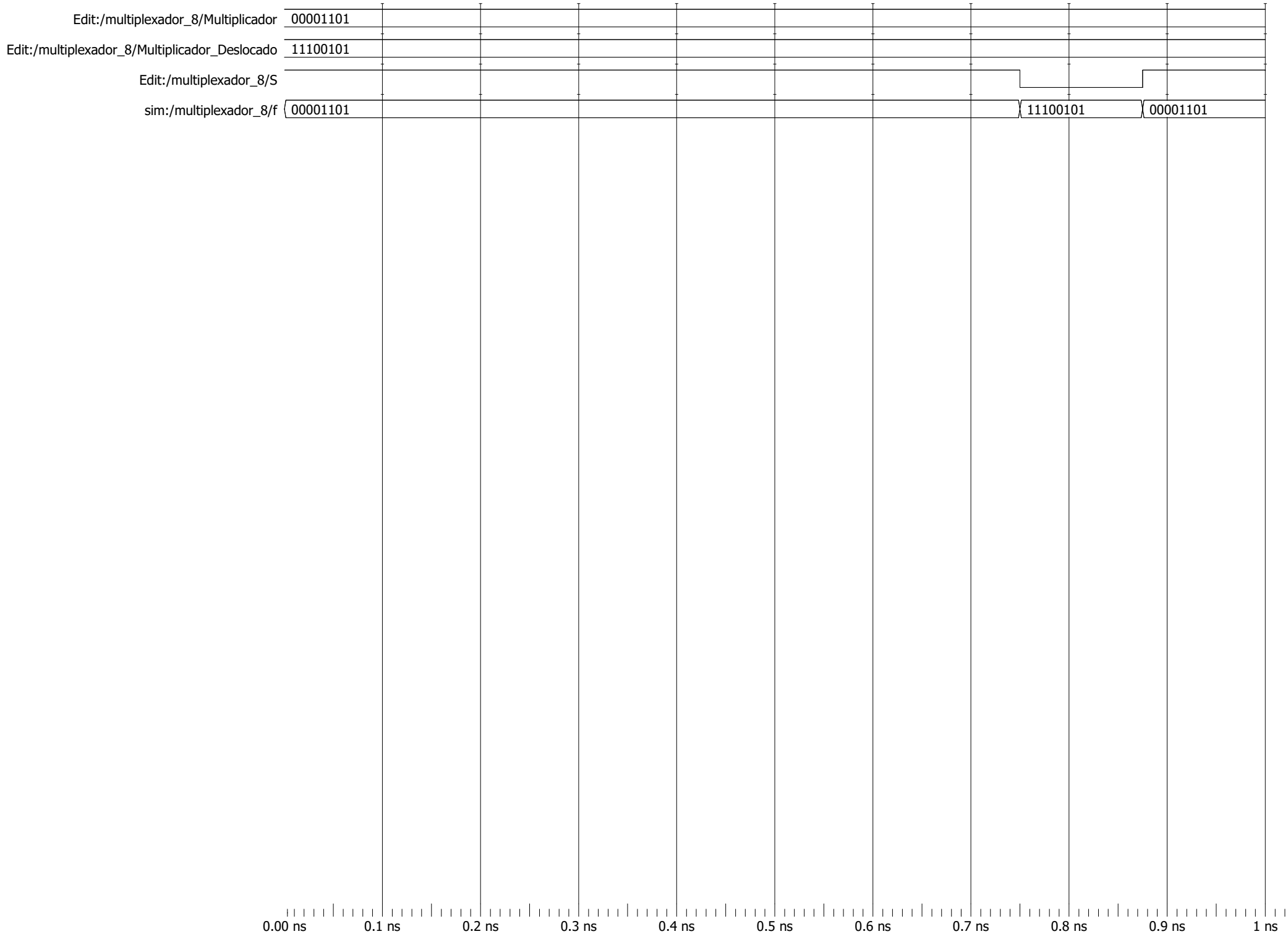


Multiplicador		
clk	in STD_LOGIC	Clock do circuito, define o "ciclo" de operações.
reset	in STD_LOGIC	Valor para controlar o reinício da operação.
Multiplicando	in STD_LOGIC_VECTOR(7 DOWNTO 0)	Valor fornecido do multiplicando.
Multiplicador	in STD_LOGIC_VECTOR(7 DOWNTO 0)	Valor fornecido do multiplicador.
R	out STD_LOGIC_VECTOR(15 DOWNTO 0)	Valor da resposta obtida..
calcula	int STD_LOGIC	Decide se deve haver uma operação ou não.
pronto	out STD_LOGIC	Sinal que indica que a operação já foi finalizada.
Enable_Entradas, Enable_Produto, Mux_Entradas, Mux_Produto	signal STD_LOGIC	Sinais de controle emitidos pela unidade de controle - a serem usados nos demais componentes.
Escolha_Multiplexador8	signal STD_LOGIC_VECTOR(7 DOWNTO 0);	Define qual foi o valor escolhido no multiplexador de 8 bits - entrada fornecida ou entrada deslocada.
Multiplicador_Deslocado	signal STD_LOGIC_VECTOR(7 DOWNTO 0);	Valor do multiplicador após ele ter sido deslocado para direita.
Escolha_Multiplexador16, Escolha_Multiplexador_Produto16	signal STD_LOGIC_VECTOR(15 DOWNTO 0)	Valores escolhidos nos multiplexadores de 16 bits, notemos que o primeiro valor é a escolha entre o multiplicando da entrada ou o multiplicando deslocado. Já o segundo diz respeito ao produto, se “0” deve ser escolhido ou algum produto já realizado.
Multiplicando_Deslocado	signal : STD_LOGIC_VECTOR(15 DOWNTO 0);	Valor do multiplicando deslocado para a esquerda.
Produto_Parcial	signal : STD_LOGIC_VECTOR(15	Produto parcial é responsável por armazenar

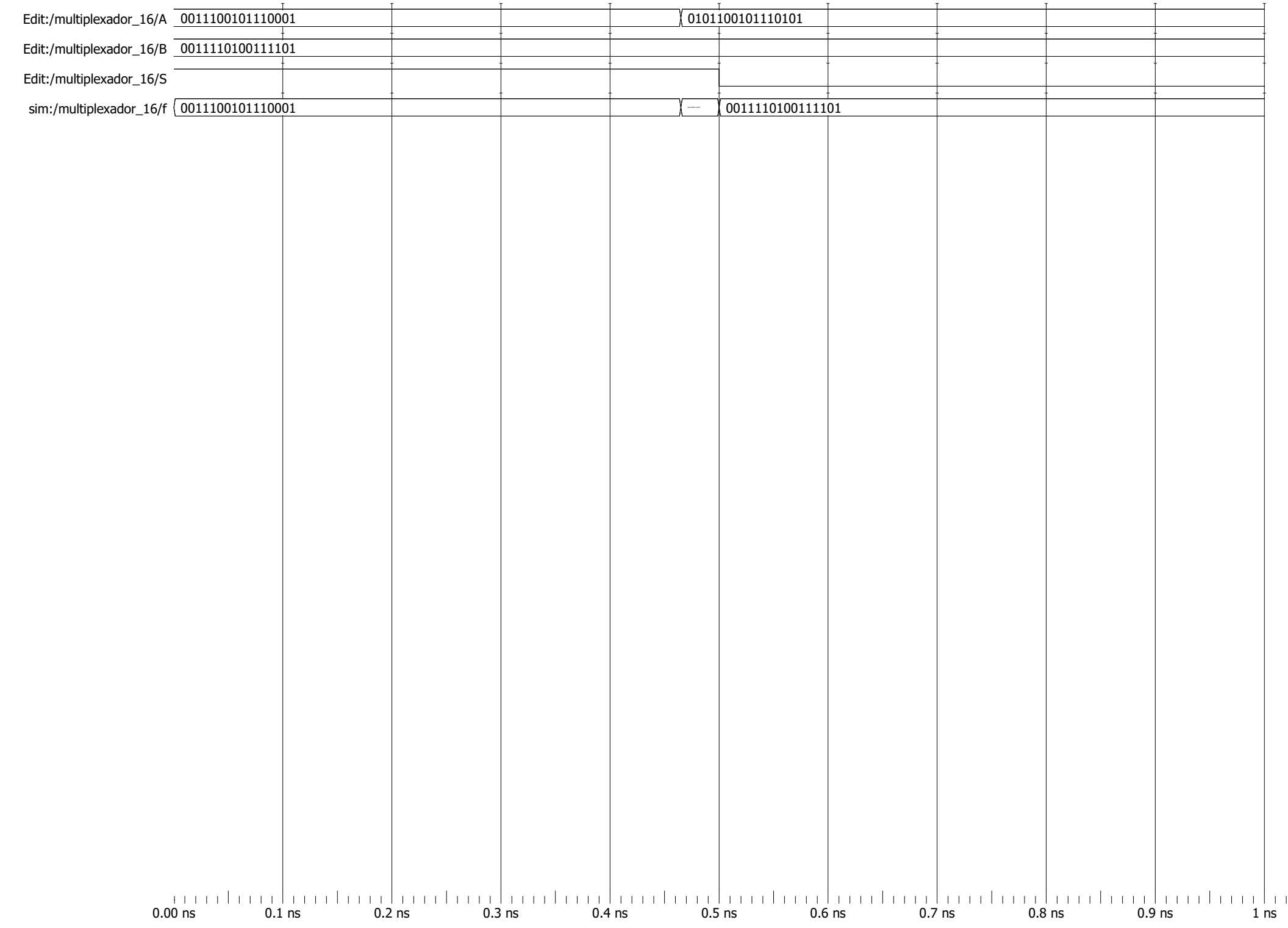
	DOWNTO 0);	a **RESPOSTA** do problema, é o valor armazenado no registrador do produto.
Produto	signal : STD_LOGIC_VECTOR(15 DOWNTO 0);	Contém o valor das somas de cada ciclo, valor que é fornecido para o registrador para substituir o valor pré-existente da soma anterior.
Multiplicando16	signal : STD_LOGIC_VECTOR(15 DOWNTO 0);	Este signal é responsável por armazenar o valor do multiplicando no momento atual da multiplicação, ou seja, o valor armazenado no registrador.
Multiplicando_Entrada	signal : STD_LOGIC_VECTOR(15 DOWNTO 0);	Como a entrada do multiplicando possui 8 bits, mas para o funcionamento do circuito é necessário que ele tenha 16, é preciso criar um sinal que agrega 8 bits para a entrada.
Multiplicador_Alterado	signal : STD_LOGIC_VECTOR(7 DOWNTO 0);	Este signal é responsável por armazenar o valor do multiplicador no momento atual da multiplicação, ou seja, o valor armazenado no registrador.

O multiplicador, apesar de parecer o circuito mais complexo de ser feito, foi um dos mais simples, isso, pois, como todos os outros componentes já estavam prontos e operando, o papel do multiplicador foi apenas de unir todos eles. Um pequeno problema encontrado ao longo do percurso foi conciliar todos os signal para que eles pudessem operar corretamente.
Por fim, é possível ver que o multiplicador está operando corretamente com uma sequência de entradas e suas respectivas multiplicações (corretas).

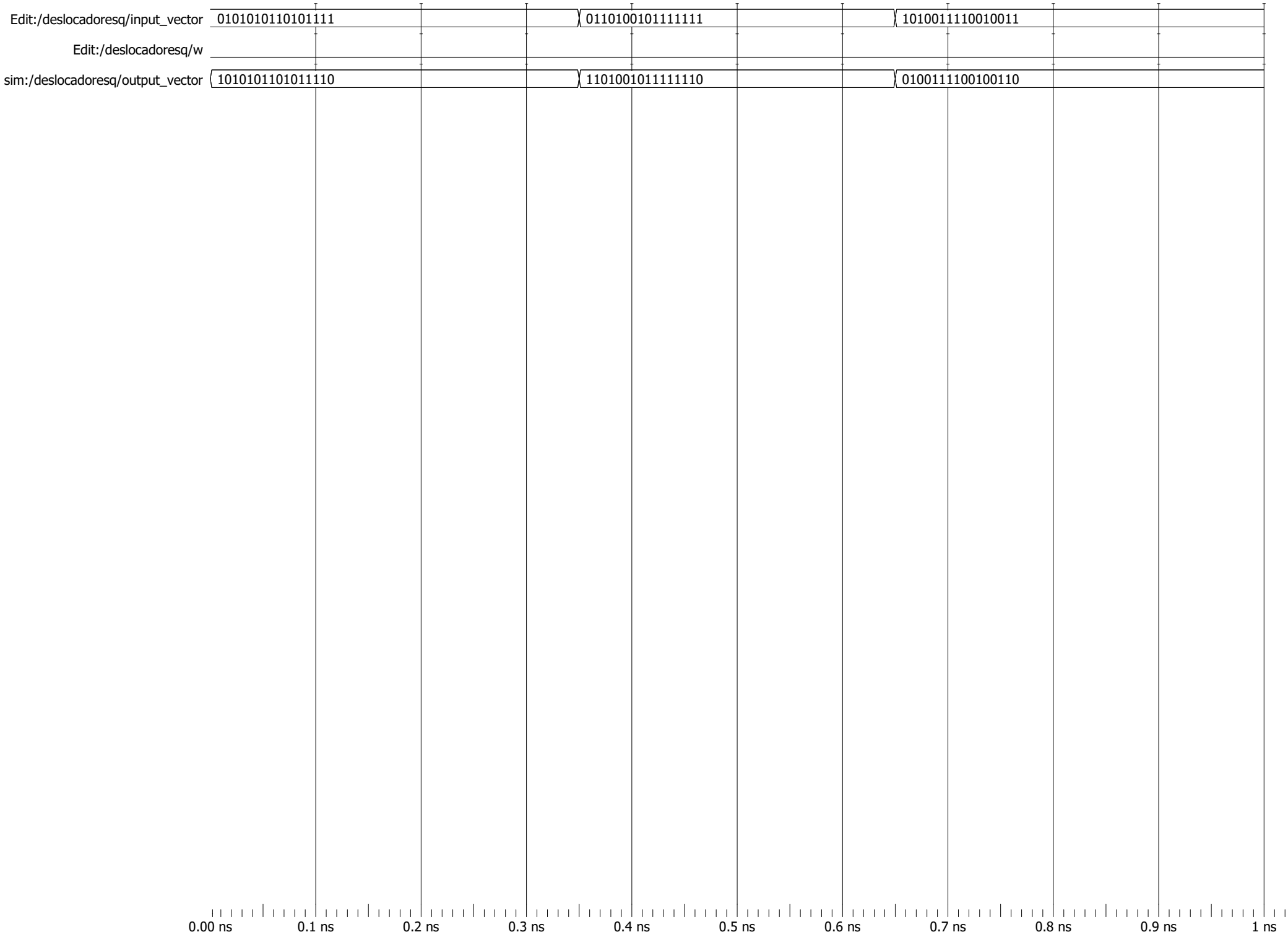

Edit:/multiplexador/A
Edit:/multiplexador/B
Edit:/multiplexador/S
sim:/multiplexador/f
0.00 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.35 ns
0.4 ns
0.45 ns
0.5 ns
0.55 ns
0.6 ns
0.65 ns
0.7 ns
0.75 ns
0.8 ns
0.85 ns
0.9 ns
0.95 ns
1 ns



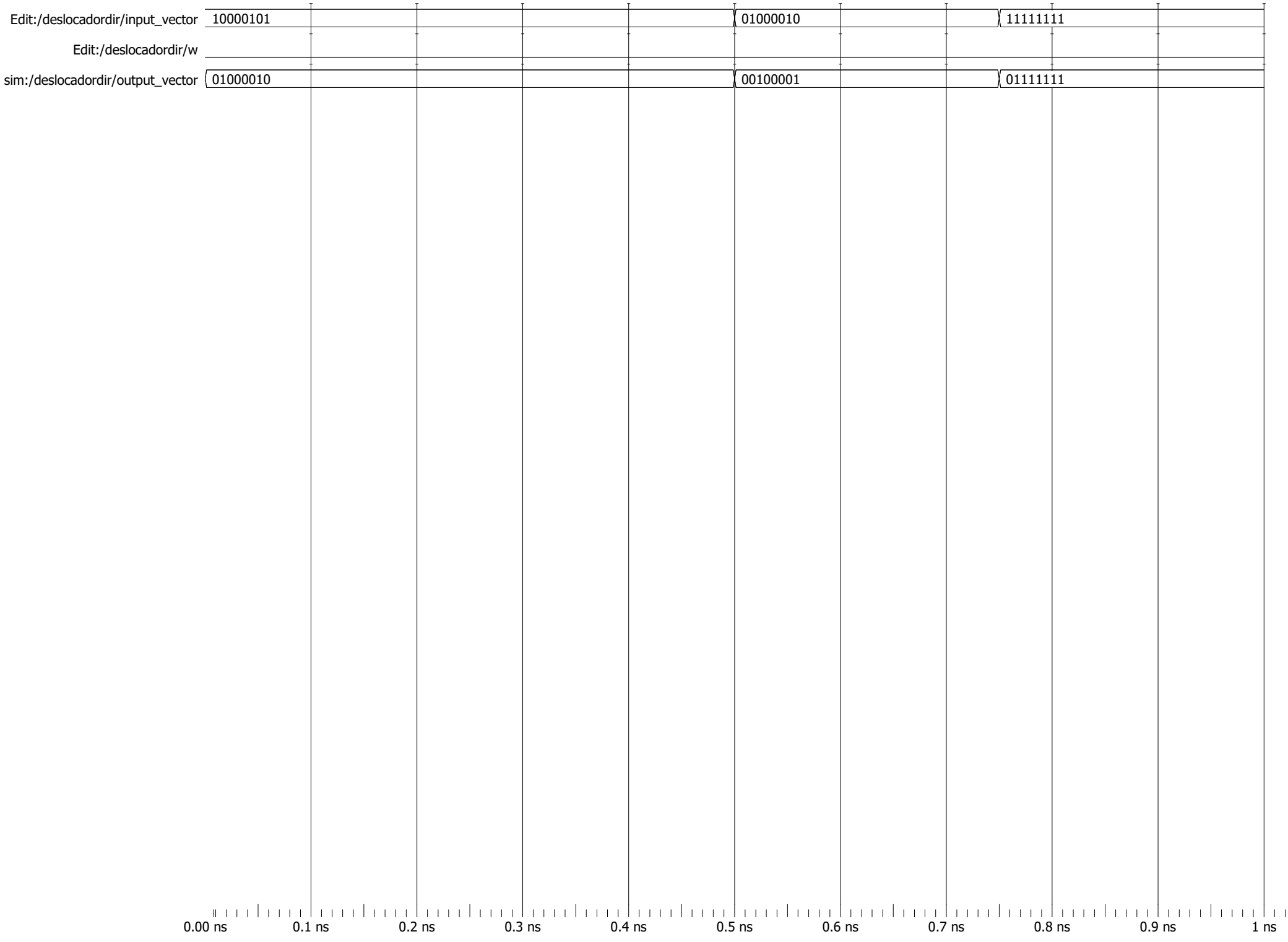
Entity:multiplexador_8 Architecture:behavior Date: Tue Nov 23 14:00:26 ART 2021 Row: 1 Page: 1



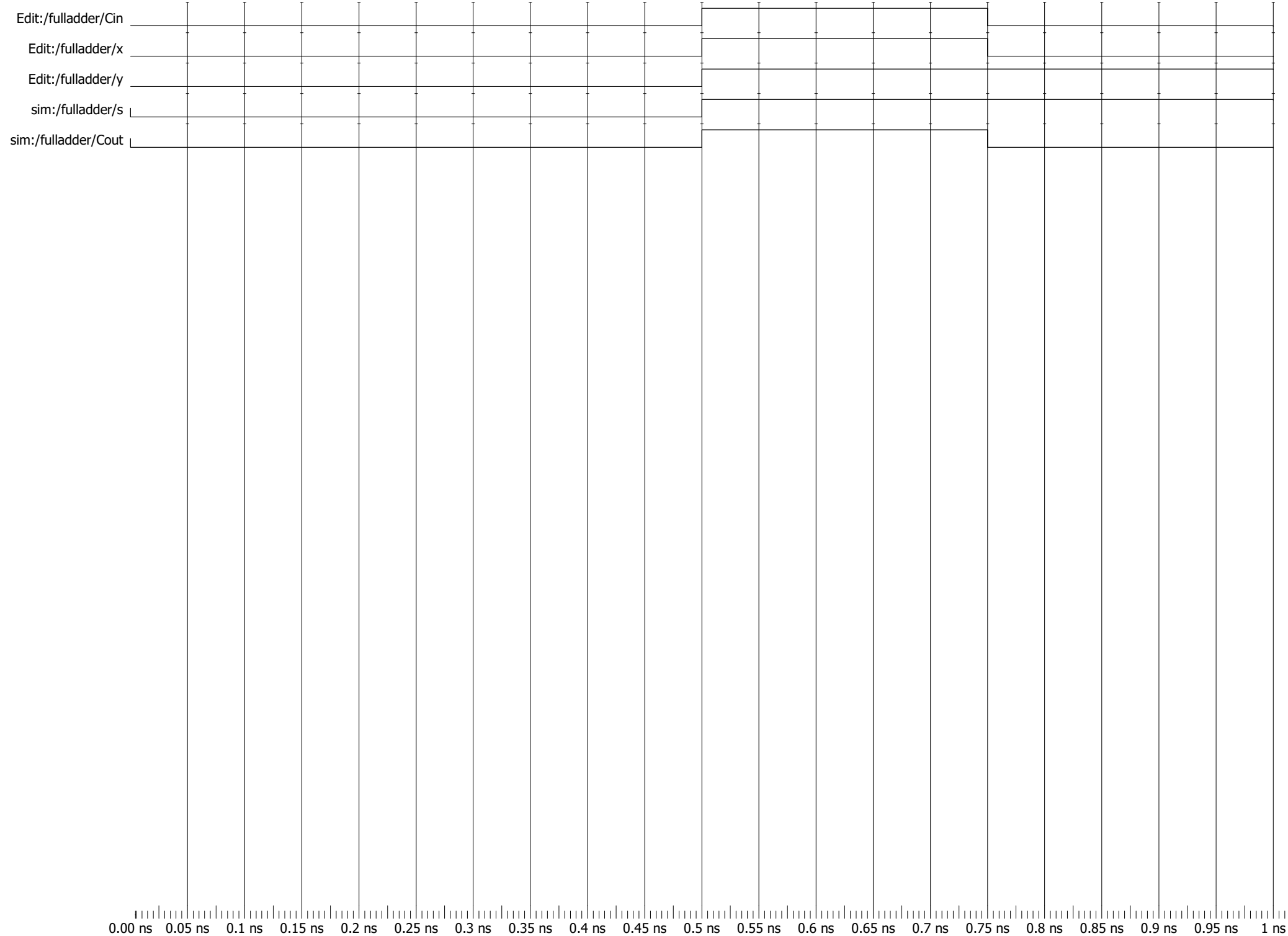
Entity:multiplexador_16 Architecture:behavior Date: Tue Nov 23 14:09:22 ART 2021 Row: 1 Page: 1

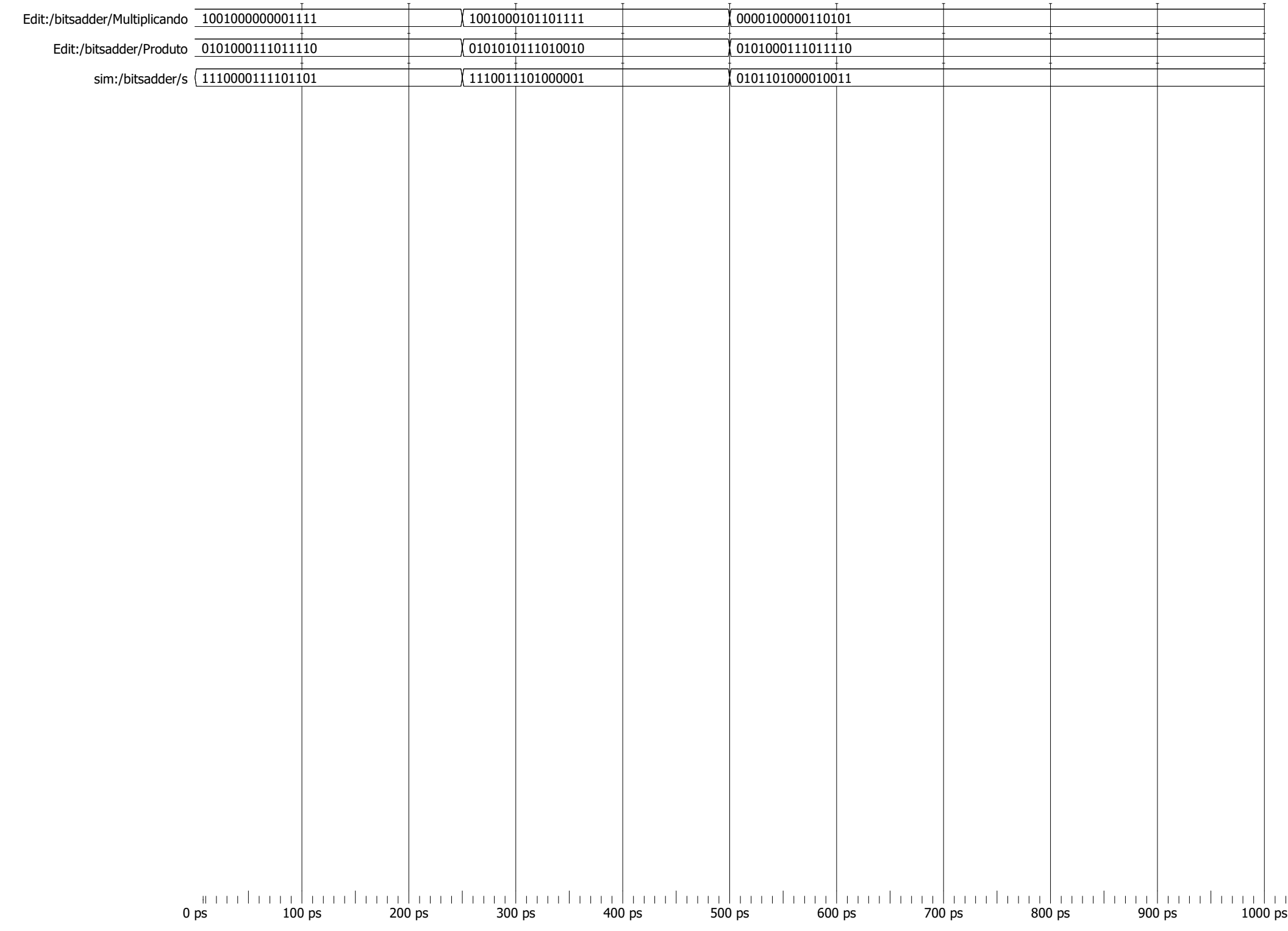


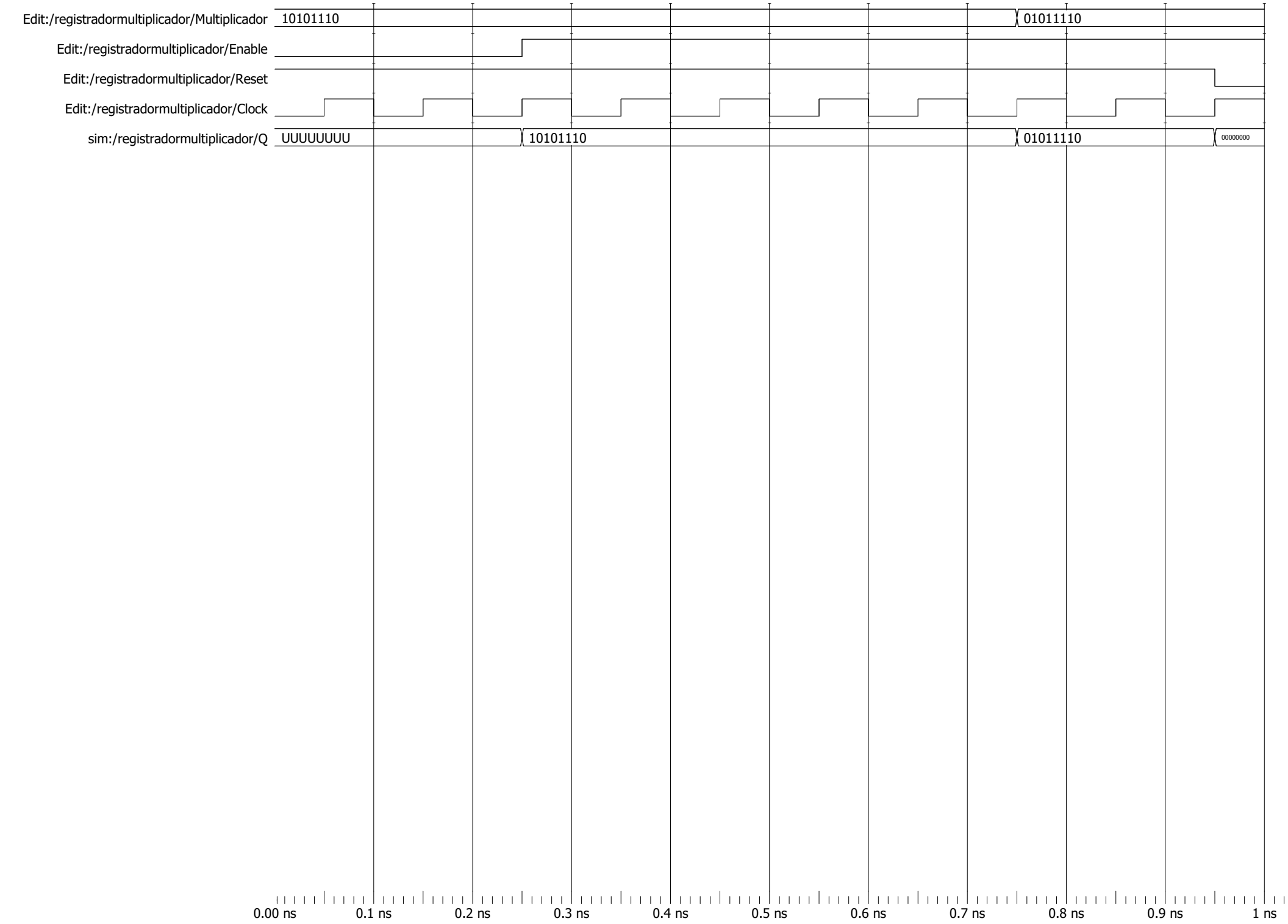
Entity:deslocadoresq Architecture:behavior Date: Tue Nov 23 14:31:44 ART 2021 Row: 1 Page: 1



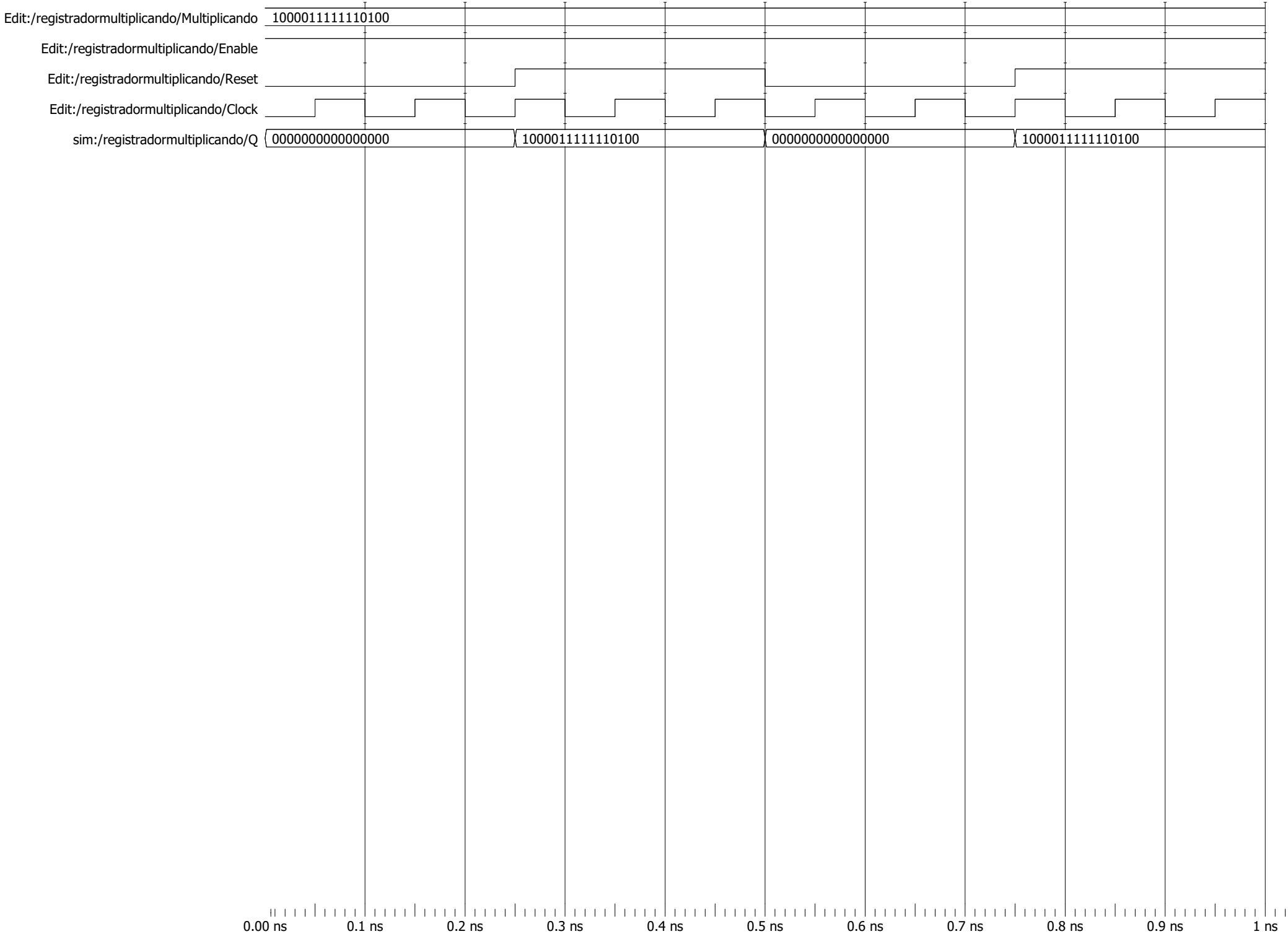
Entity:deslocadordir Architecture:behavior Date: Tue Nov 23 14:24:43 ART 2021 Row: 1 Page: 1


Edit:/fulladder/Cin
Edit:/fulladder/x
Edit:/fulladder/y
sim:/fulladder/s
sim:/fulladder/Cout
0.00 ns
0.05 ns
0.1 ns
0.15 ns
0.2 ns
0.25 ns
0.3 ns
0.35 ns
0.4 ns
0.45 ns
0.5 ns
0.55 ns
0.6 ns
0.65 ns
0.7 ns
0.75 ns
0.8 ns
0.85 ns
0.9 ns
0.95 ns
1 ns

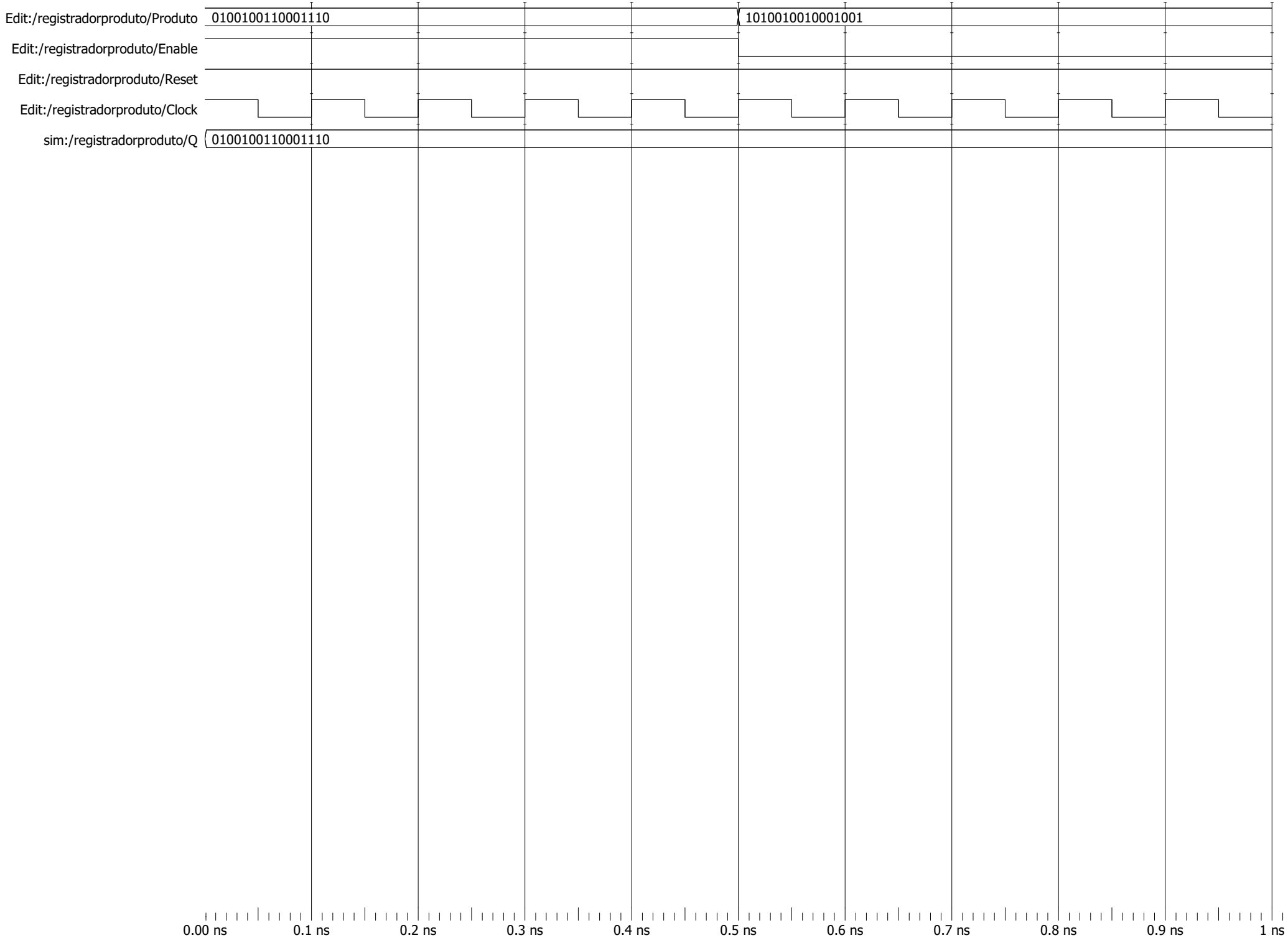

Edit:/bitsadder/Multiplicando
1001000000001111
1001000101101111
0000100000110101
Edit:/bitsadder/Produto
0101000111011110
0101010111010010
0101000111011110
sim:/bitsadder/s
1110000111101101
1110011101000001
0101101000010011
0 ps
100 ps
200 ps
300 ps
400 ps
500 ps
600 ps
700 ps
800 ps
900 ps
1000 ps



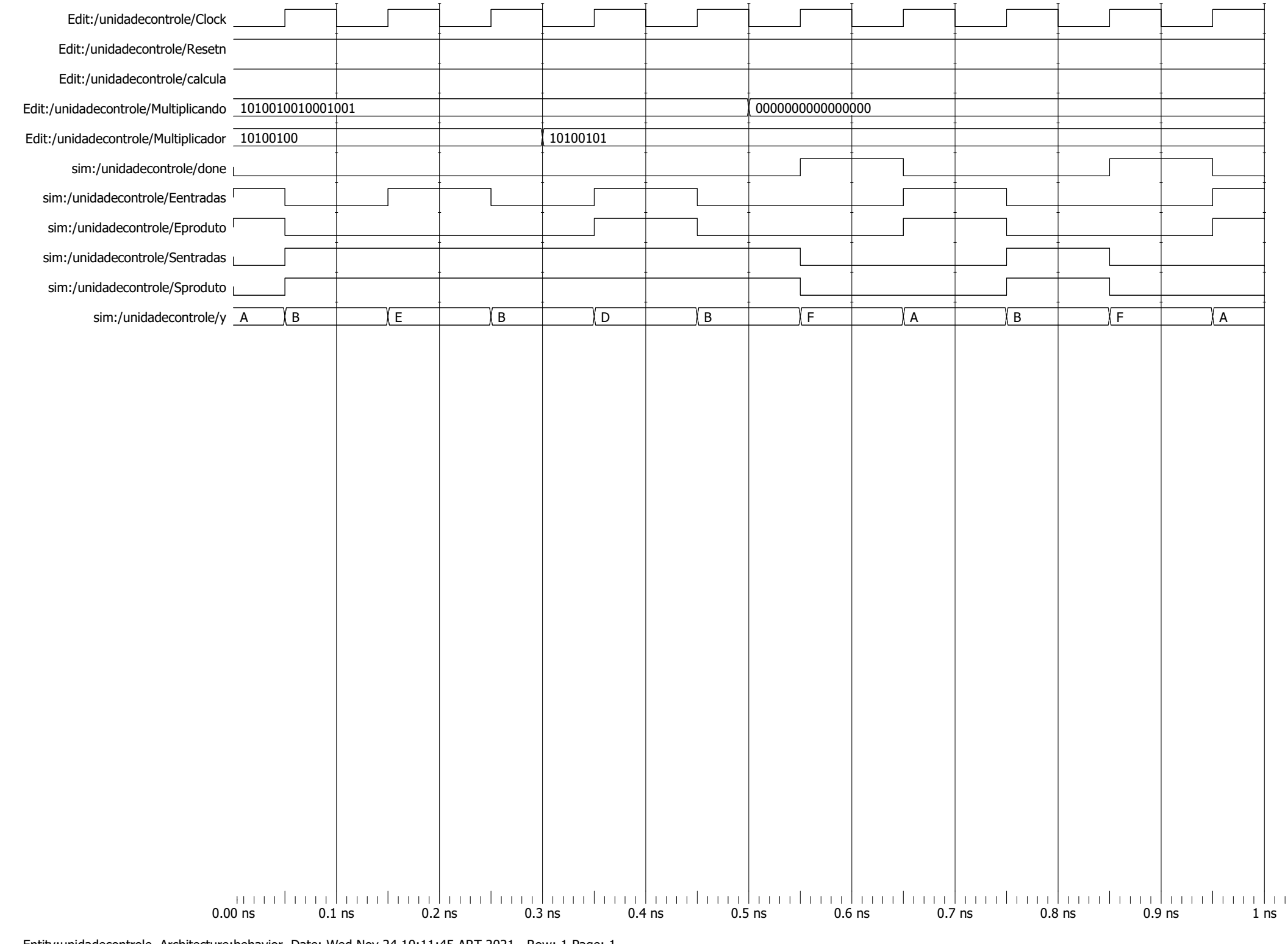
Entity:registradormultiplicador Architecture:structure Date: Wed Nov 24 08:17:34 ART 2021 Row: 1 Page: 1



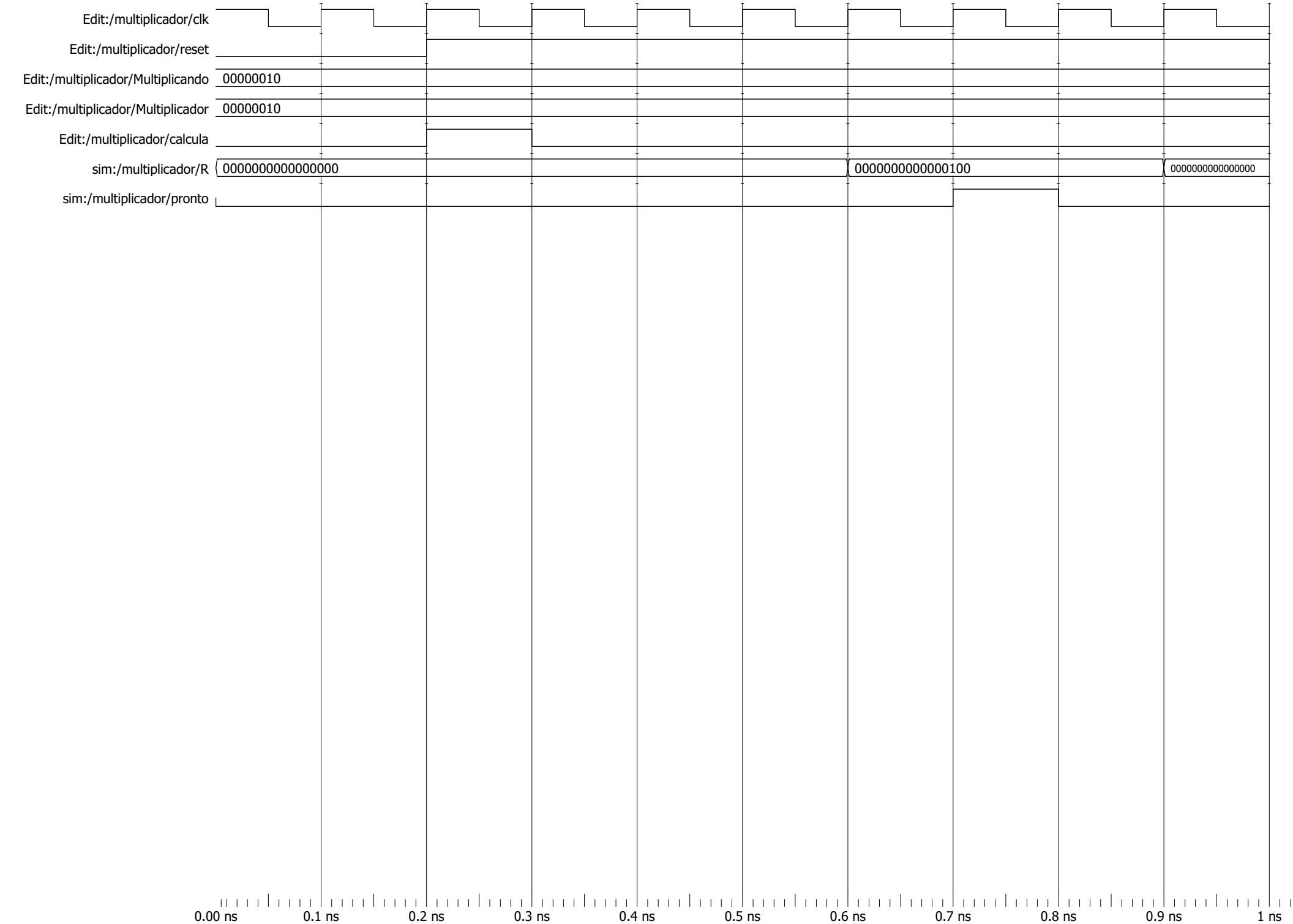
Entity:registradormultiplicando Architecture:structure Date: Wed Nov 24 08:25:23 ART 2021 Row: 1 Page: 1



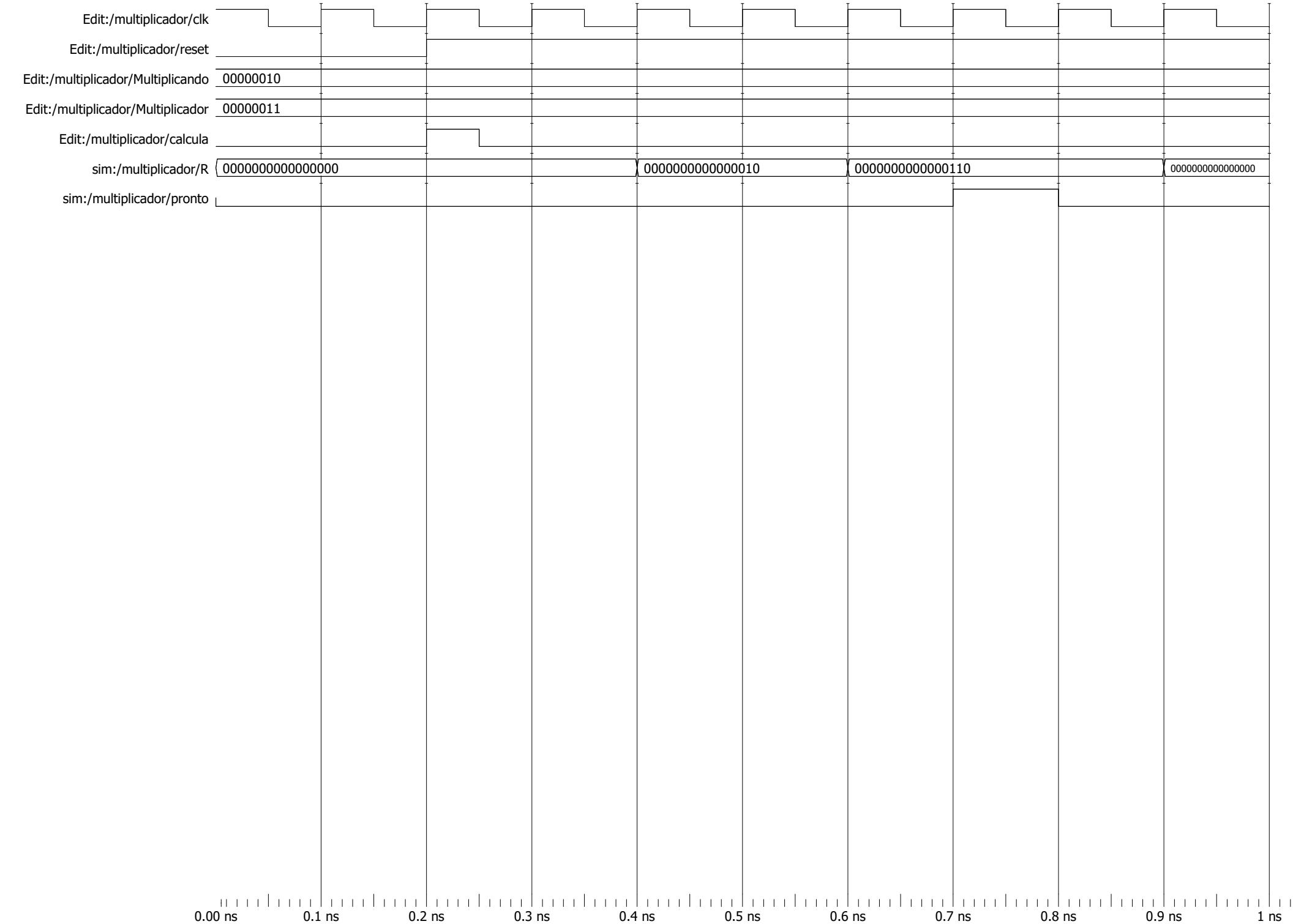
Entity:registradorproduto Architecture:structure Date: Wed Nov 24 08:34:58 ART 2021 Row: 1 Page: 1



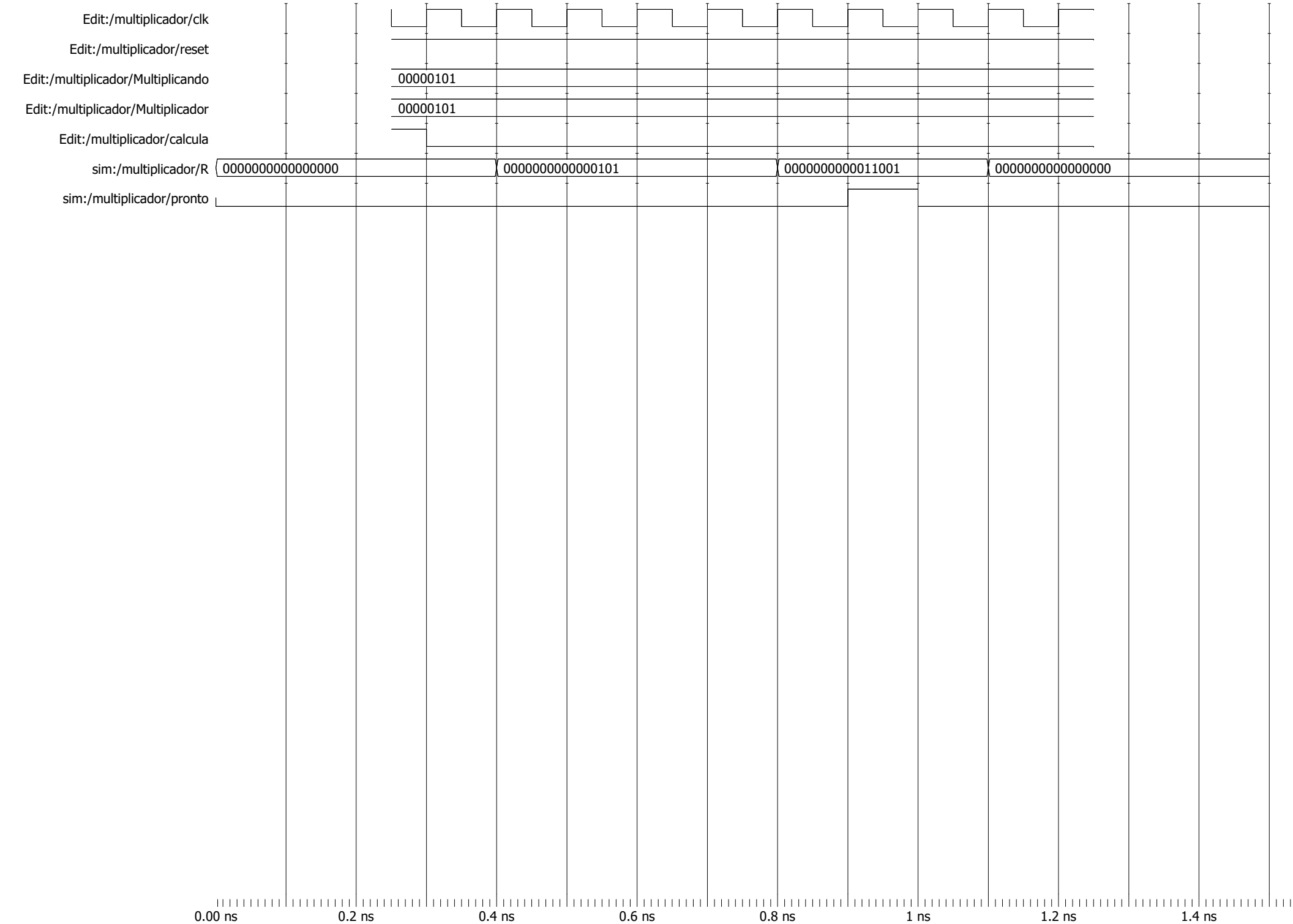
Entity:unidadecontrole Architecture:behavior Date: Wed Nov 24 10:11:45 ART 2021 Row: 1 Page: 1



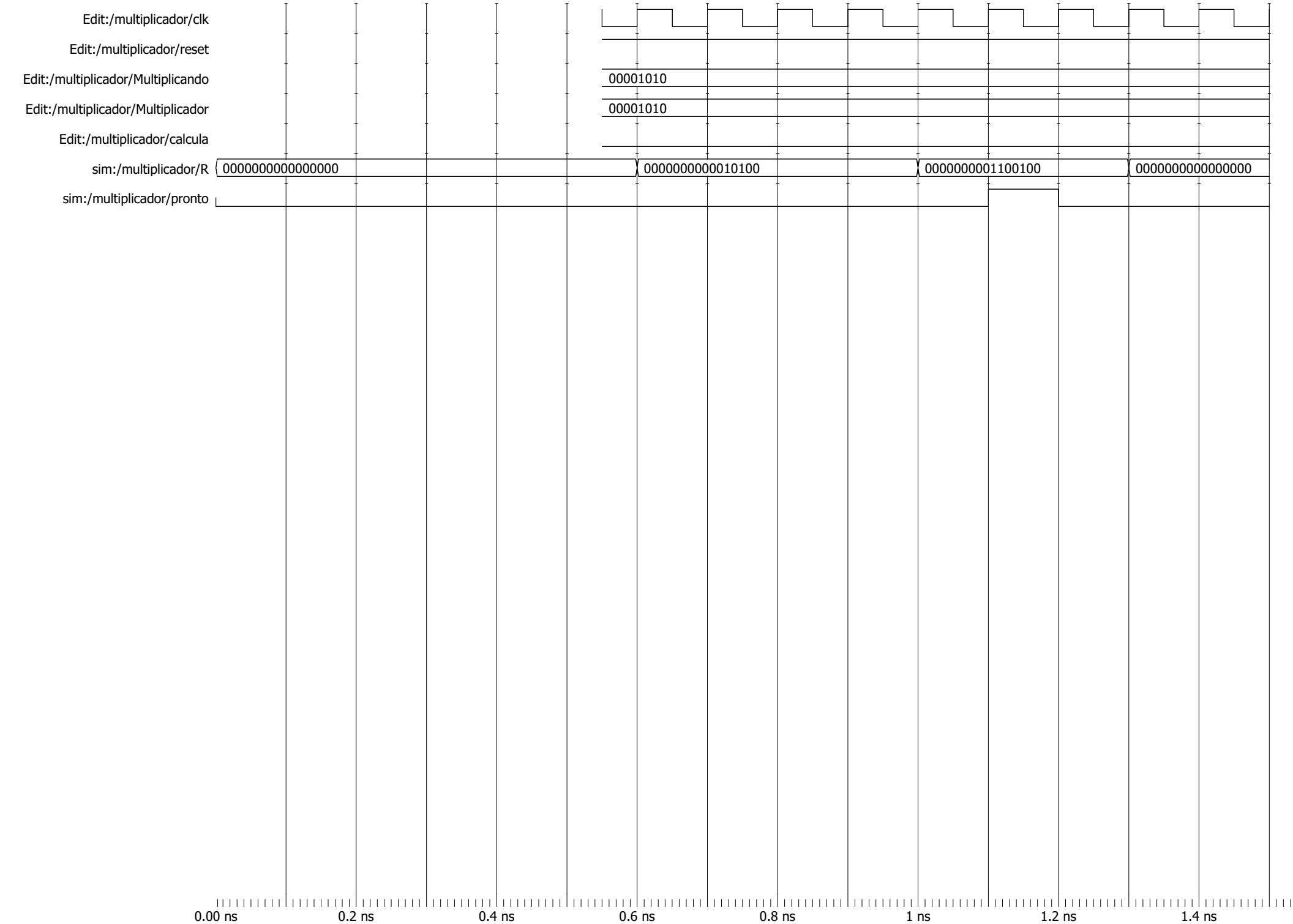
Entity:multiplicador Architecture:behavior Date: Wed Nov 24 10:39:25 ART 2021 Row: 1 Page: 1



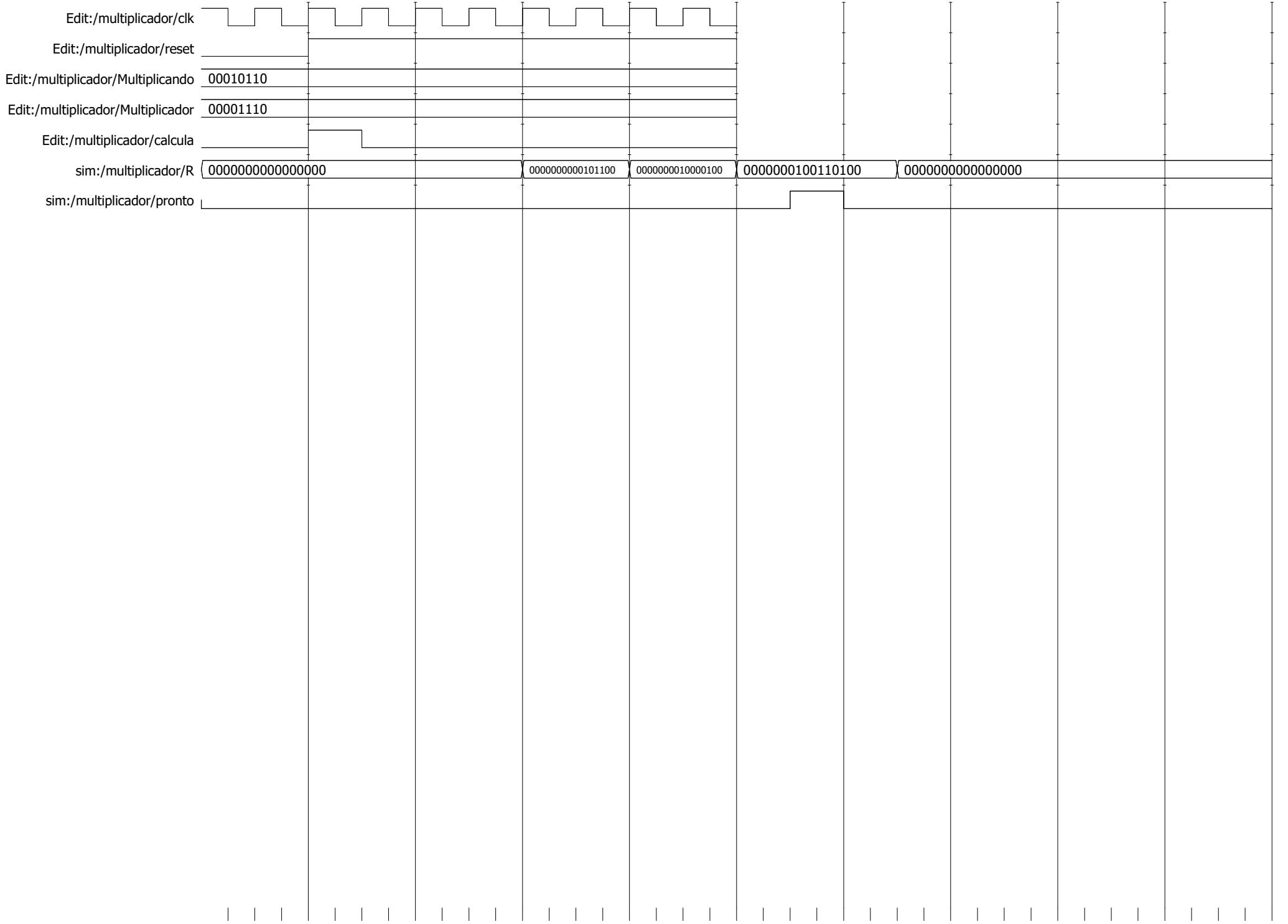
Entity:multiplicador Architecture:behavior Date: Wed Nov 24 10:30:52 ART 2021 Row: 1 Page: 1



Entity:multiplicador Architecture:behavior Date: Wed Nov 24 10:41:22 ART 2021 Row: 1 Page: 1



Entity:multiplicador Architecture:behavior Date: Wed Nov 24 10:43:22 ART 2021 Row: 1 Page: 1



Entity:multiplicador Architecture:behavior Date: Wed Nov 24 10:38:09 ART 2021 Row: 1 Page: 1