

Projeto do Processador Neander em VHDL

INF01175 – Sistemas Digitais para Computação

Aline Hommerding Amorim - 00301570

Descrição do Trabalho

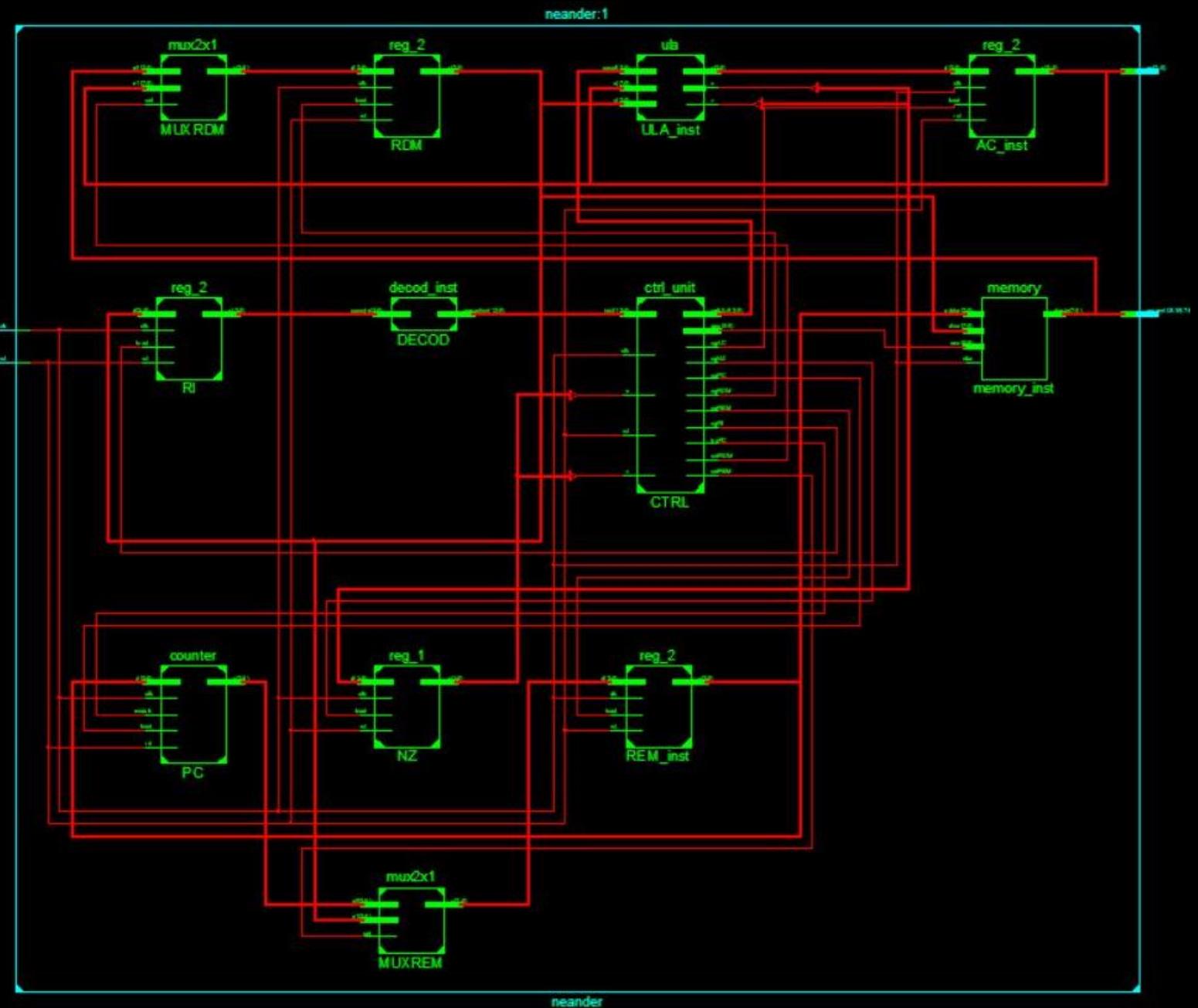
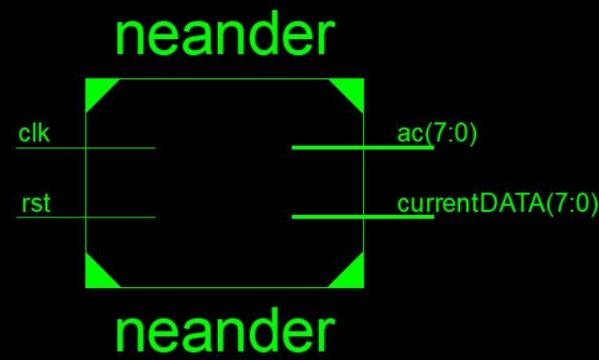
Descrição do Trabalho

- O trabalho aqui descrito tem por objetivo implementar o computador hipotético NEANDER utilizando a linguagem de descrição de hardware VHDL (VHSIC Hardware Description Language).
- O simulador utilizado para sua execução é o ISE® Design Suite, da companhia Xilinx.
- Os algoritmos a serem implementados para fim de teste são: soma de duas matrizes de dimensões 2x2, multiplicação de duas matrizes de dimensões 2x2 e cálculo do delta de uma função quadrática, dados os valores de a, b e c.
- Duas instruções extras serão adicionadas ao computador: a instrução SUB (subtração) e a instrução MUL (multiplicação).

Descrição do Neander

- O computador hipotético NEANDER é uma máquina muito simples, desenvolvida para fins didáticos. Seu conjunto de instruções é limitado a **NOP** (no operation), **STA** (store), **LDA** (load), **ADD** (add), **OR** (logical or), **AND** (logica and), **NOT** (logical not), **JMP** (jump to address), **JN** (jump if negative), **JZ** (jump if zero) e **HLT** (halt).
- Sua arquitetura possui endereços e dados de **8 bits**, com esses dados sendo representados em **complemento de 2**.
- O computador possui um acumulador de 8 bits (**AC**), um apontador de programa de 8 bits (**PC**), um registrador para endereços de memória de 8 bits (**REM**), um registrador para dados da memória de 8 bits (**RDM**), um registrador de instruções de 8 bits (**RI**) e um registrador de estado com 2 códigos de condição : negativo (**N**) e zero (**Z**).
- Todos os sinais de controles utilizados pela máquina são produzidos pela Unidade de Controle.

Projeto do Neander



Algoritmos testados

Algoritmo ① – Soma de matrizes

- O algoritmo de número ①, como indicado na descrição do trabalho, soma duas matrizes de dimensões 2x2.
- Sua execução é feita de modo simples e direto, com somas consecutivas de cada uma das posições das matrizes.

$$\begin{bmatrix} 128 & 129 \\ 130 & 131 \end{bmatrix} + \begin{bmatrix} 132 & 133 \\ 134 & 135 \end{bmatrix} = \begin{bmatrix} 136 & 137 \\ 138 & 139 \end{bmatrix}$$

Esquema dos endereços de entradas e saídas.

LDA 128	ADD 134
ADD 132	STA 138
STA 136	LDA 131
LDA 129	ADD 135
ADD 133	STA 139
STA 137	HLT
LDA 130	

Código em Assembly.

Algoritmo ② – Multiplicação de matrizes

- O algoritmo de número ②, como indicado na descrição do trabalho, multiplica duas matrizes de dimensões 2x2.
- Sua execução é feita do modo mais direto possível, calculando as posições da matriz resultado uma a uma. Como o cálculo de um elemento resultado exige a soma de duas multiplicações, é utilizada uma variável auxiliar para guardar uma destas multiplicações.

$$\begin{bmatrix} 128 & 129 \\ 130 & 131 \end{bmatrix} * \begin{bmatrix} 132 & 133 \\ 134 & 135 \end{bmatrix} = \begin{bmatrix} 136 & 137 \\ 138 & 139 \end{bmatrix}$$

142 – variável auxiliar

Esquema dos endereços de entradas e saídas.

LDA 128	LDA 128	LDA 130	LDA 130	HLT
MUL 132	MUL 133	MUL 132	MUL 133	
STA 142	STA 142	STA 142	STA 142	
LDA 129	LDA 129	LDA 131	LDA 131	
MUL 134	MUL 135	MUL 134	MUL 135	
ADD 142	ADD 142	ADD 142	ADD 142	
STA 136	STA 137	STA 138	STA 139	

Código em Assembly.

Algoritmo ③ – Cálculo do Δ de uma função

- O algoritmo escolhido de número ③, calcula o delta de uma função quadrática dados os valores de a, b e c. A equação seguida é a seguinte: $\Delta = b^2 - 4 * a * c$.
- Sua execução utiliza as instruções “extras” SUB e MUL. Dois endereços foram utilizados além dos que guardam as entradas e saídas – um para o armazenamento de uma constante (4) e outro para uma variável auxiliar, que guarda o valor de $4 * a * c$.

$$\begin{aligned}\Delta &= b^2 - 4 * a * c \\ 131 &= 129^2 - 4 * 128 * 129 \\ &133 - \text{constante } 4 \\ &134 - \text{variável auxiliar}\end{aligned}$$

Esquema dos endereços de entradas e saídas.

```
LDA 128      STA 131
MUL 133      HLT
MUL 130
STA 134
LDA 129
MUL 129
SUB 134
```

Código em Assembly.

Testbench

Descrição do testbench utilizado

- Por questão de conveniência, o clock utilizado para todas as simulações exibidas neste relatório operava com um período de 10ns (25MHz). Para a obtenção dos dados de tempo, este período foi alterado para adequar-se aos 50MHz pedidos.
- O testbench é simples – mantém o sinal de reset ligado por 100ns, depois libera-o para permitir que a computação tenha início.

```
constant clk_period : time := 10 ns;
clk_process :process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;
```

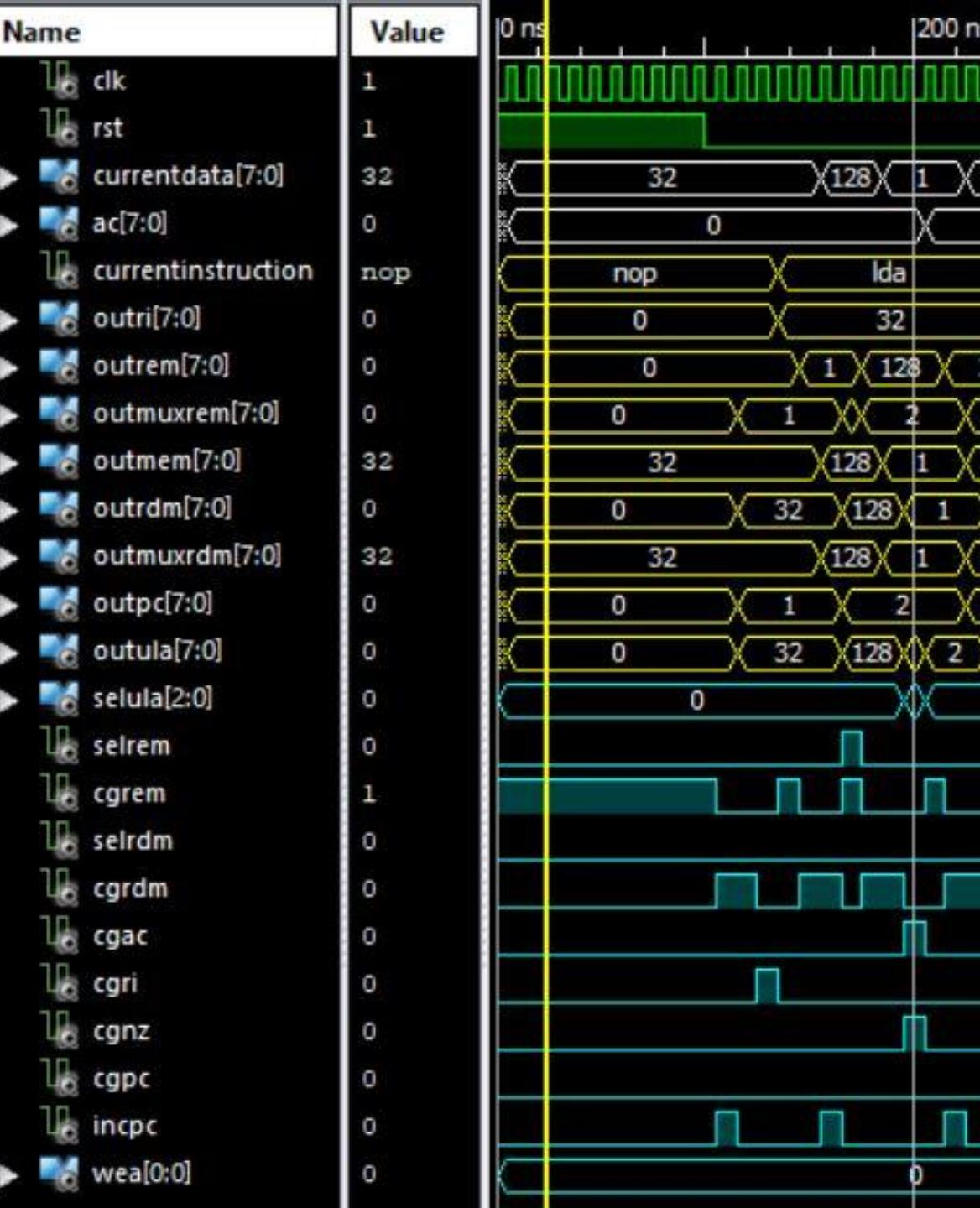
```
stim_proc: process
begin
    -- hold reset state for 100 ns.
    rst <= '1';
    wait for 100 ns;
    rst <= '0';
    wait;
end process;
```

Simulações e memórias

Figura X: Modelo de simulação sem atraso.

Modelo de simulação sem atraso

- Verde – Entradas do módulo neander.vhd.
 - clk – clock
 - rst – reset
- Branco – Saídas do módulo neander.vhd.
 - currentdata – saída atual da memória
 - ac – saída atual do acumulador
- Amarelo – Sinais dos registradores (internos ao módulo).
 - outri – saída do registrador de instruções.
 - outrem – saída do registrador de endereços de memória.
 - outmuxrem – saída do multiplexador do REM.
 - outmem – saída atual da memória.
 - outrdm – saída do registrador de dados da memória.
 - outmuxrdm – saída do multiplexador do RDM.
 - outpc – saída do contador de programa.
 - outula – saída da unidade aritmética e lógica.
- Azul – Sinais de controle (internos ao módulo).



Modelo de simulação com atraso

- Verde – Entradas do módulo neander.vhd.
 - clk – clock
 - rst – reset
- Branco – Saídas do módulo neander.vhd.
 - currentdata – saída atual da memória
 - ac – saída atual do acumulador
- Amarelo – Sinais dos registradores (internos ao módulo).
 - ri_s – 4 bits mais significativos do RI.
 - rem_inst_s – saída do registrador de endereços de memória.
 - outmuxrem – saída do multiplexador do REM.
 - rdm_s – saída do registrador de dados da memória.
 - outmuxrdm – saída do multiplexador do RDM.
 - pc_temp – saída do contador de programa.
 - outula – saída da unidade aritmética e lógica.

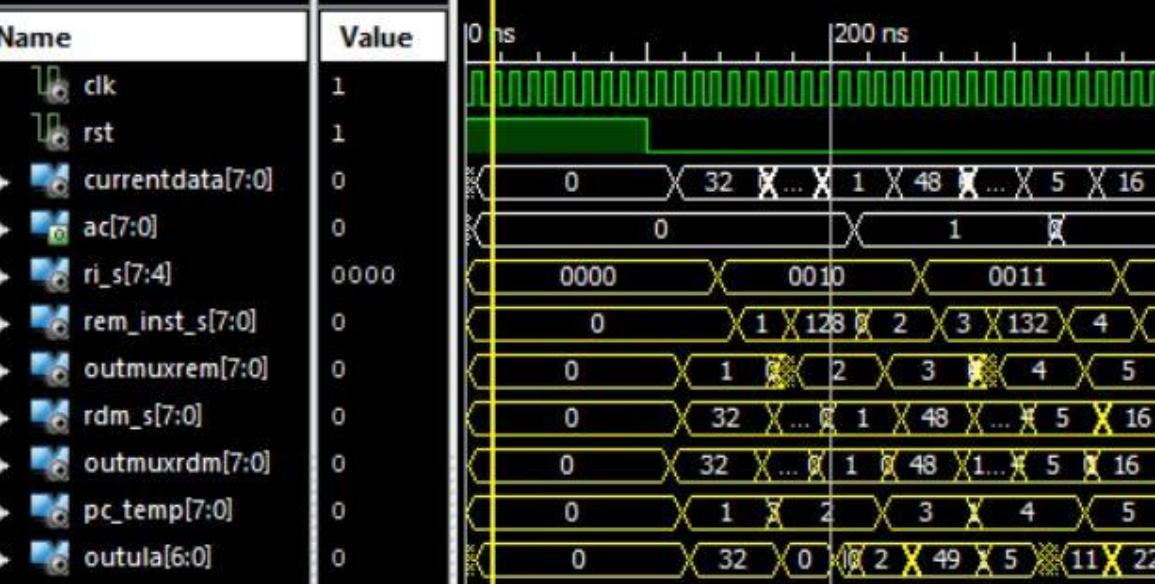


Figura X: Modelo de simulação com atraso.

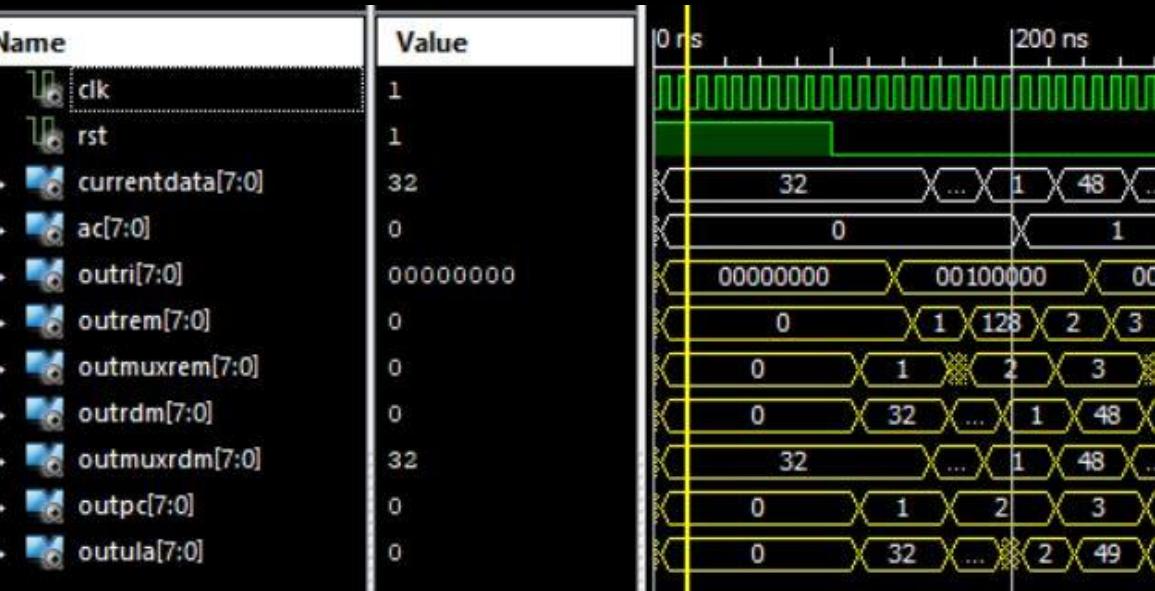


Figura X: Simulação sem atraso para comparação.

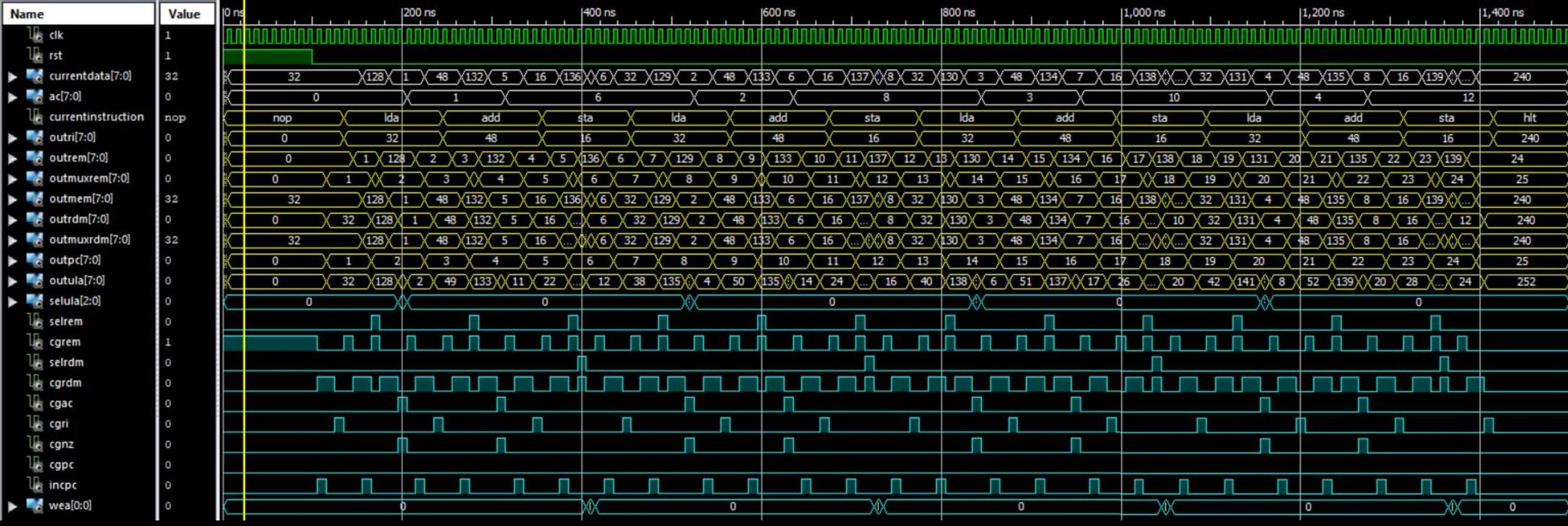
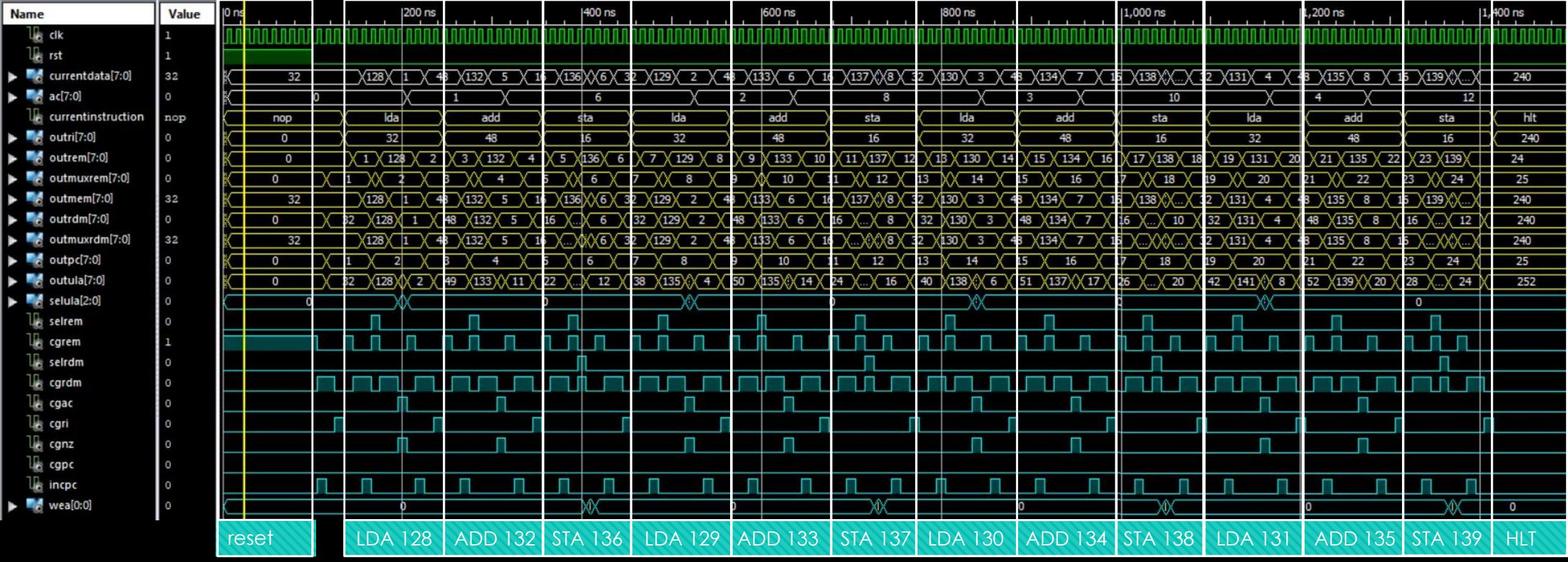


Figura X: Simulação completa sem atraso do algoritmo ①.

Simulação sem atraso do algoritmo ①



Análise da simulação sem atraso do algoritmo ①

A simulação segue as instruções do algoritmo ① descritas anteriormente.

	0	1	2	3
0	32	128	48	132
4	16	136	32	129
8	48	133	16	137
12	32	130	48	134
16	16	138	32	131
20	48	135	16	139
24	240	0	0	0

Figura X: Instruções do algoritmo ① na memória.

128	1	2	3	4
132	5	6	7	8
136	0	0	0	0

Figura X: Entradas e saída do algoritmo ① antes de sua execução

128	1	2	3	4
132	5	6	7	8
136	6	8	10	12

Figura X: Entradas e saída do algoritmo ① após sua execução.

Memória da simulação sem atraso do algoritmo ①

De acordo com as matrizes iniciais $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ e $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, o algoritmo ① executa a soma $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 3 & 8 \\ 10 & 12 \end{bmatrix}$.

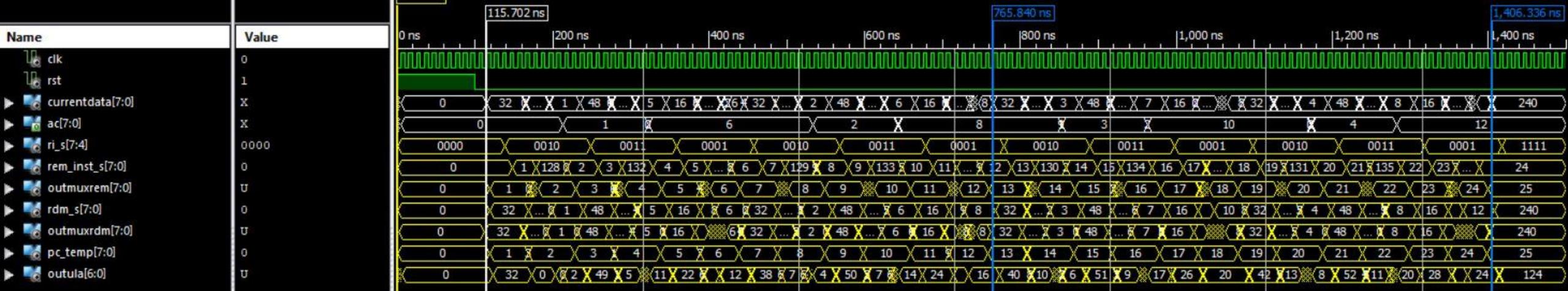


Figura X: Simulação completa com atraso do algoritmo ①.

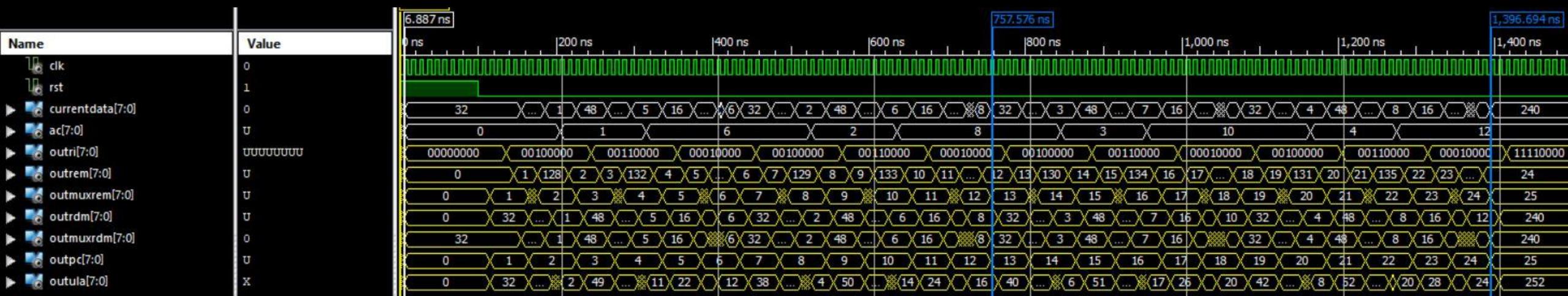


Figura X: Simulação sem atraso do algoritmo ① com os mesmos dados para comparação.

Simulação com atraso do algoritmo ①

A simulação com atraso (acima) possui áreas que claramente sofreram um atraso em relação à simulação sem atraso (abaixo).

Seu início (instrução 32 carregada no RI) ocorre 108.8ns depois do início da simulação sem atraso.

Seu meio (LDA 130) ocorre 8.2ns depois do meio da simulação sem atraso.

Seu fim (instrução HLT carregada no RI) ocorre 9.6ns depois do fim da simulação sem atraso.

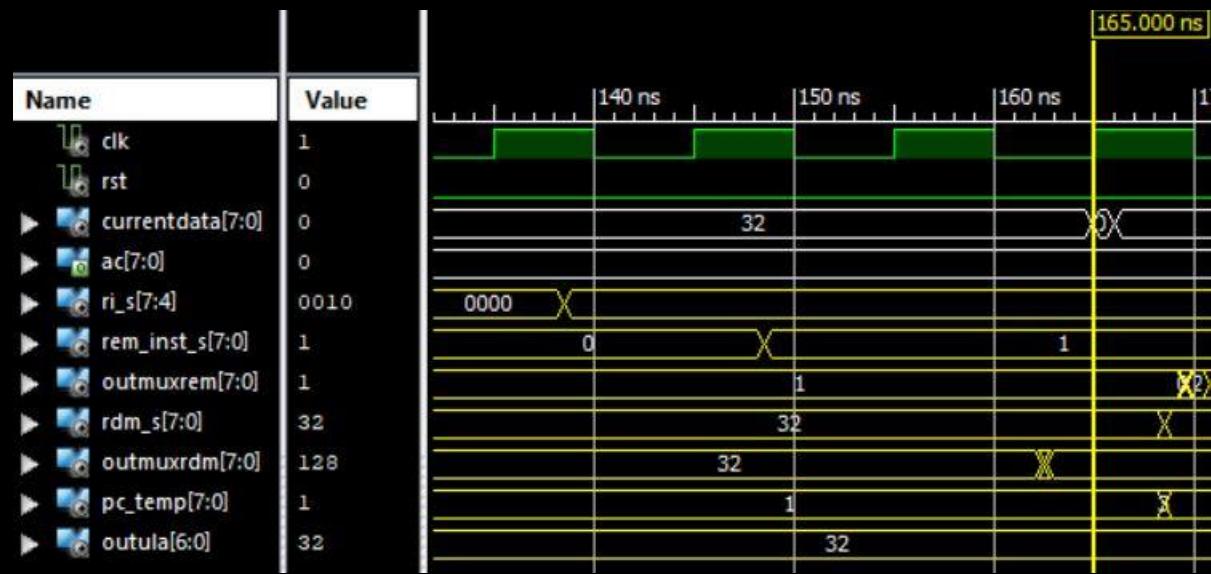


Figura X: Simulação com atraso do algoritmo ①.

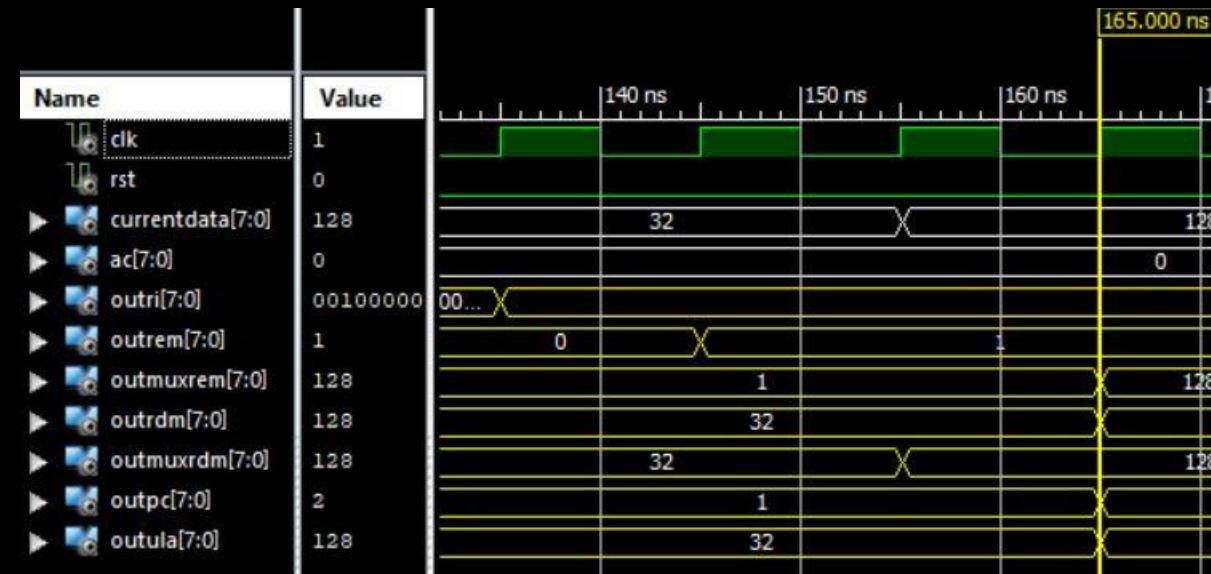


Figura X: Simulação sem atraso do algoritmo ① com os mesmos dados para comparação.

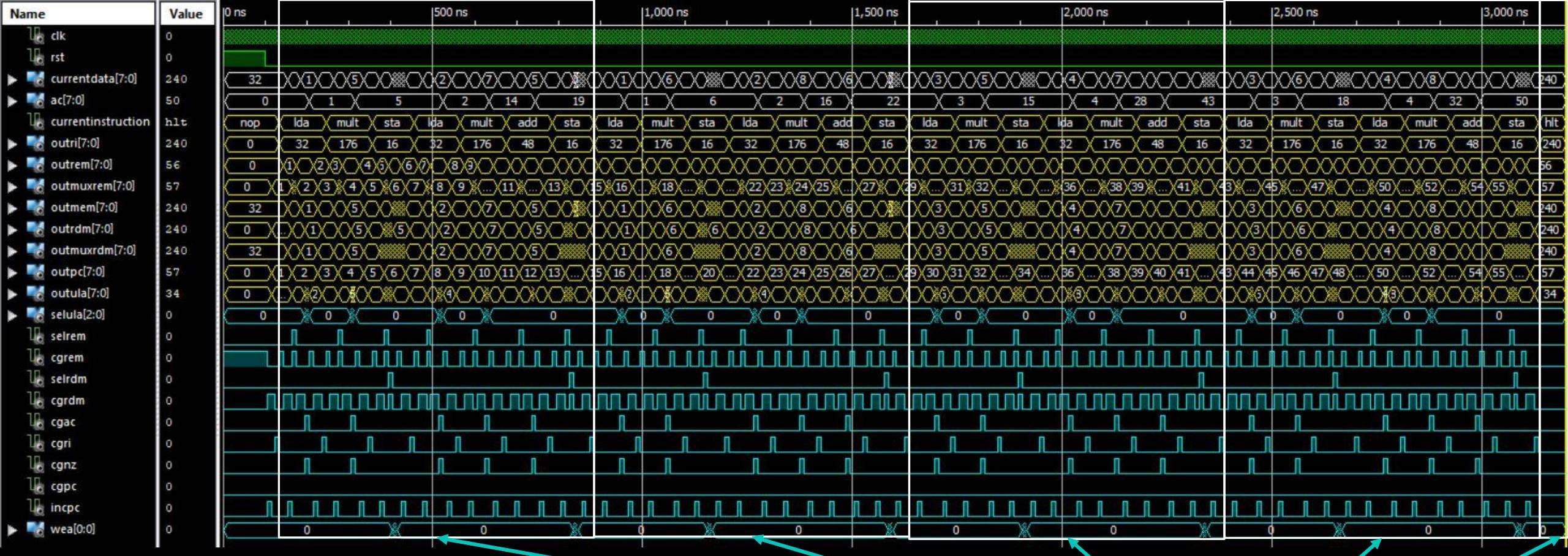
Análise pontual da simulação com atraso do algoritmo ①

No pequeno tempo exibido, fica evidente a diferença entre a simulação com e sem atraso. No instante de 165ns, o dado **currentdata** da simulação com atraso sofre uma pequena alteração para o valor de 0 antes de receber seu valor correto, 128. Enquanto a troca na simulação com atraso ocorre por volta do instante 166.2ns, na simulação sem atraso, ocorre no instante 155.3ns. Uma diferença de 11.1ns.



Figura X: Simulação completa sem atraso do algoritmo ②.

Simulação sem atraso do algoritmo ②



Análise da simulação sem atraso do algoritmo ②

LDA 128	LDA 128	LDA 130	LDA 130	HLT
MUL 132	MUL 133	MUL 132	MUL 133	
STA 142	STA 142	STA 142	STA 142	
LDA 129	LDA 129	LDA 131	LDA 131	
MUL 134	MUL 135	MUL 134	MUL 135	
ADD 142	ADD 142	ADD 142	ADD 142	
STA 136	STA 137	STA 138	STA 139	

	0	1	2	3	28	32	130	176	132
0	32	128	176	132	32	16	142	32	131
4	16	142	32	129	36	176	134	48	142
8	176	134	48	142	40	16	138	32	130
12	16	136	32	128	44	176	133	16	142
16	176	133	16	142	48	32	131	176	135
20	32	129	176	135	52	48	142	16	139
24	48	142	16	137	56	240	0	0	0

Figura X: Instruções do algoritmo ② na memória.

128	1	2	3	4
132	5	6	7	8
136	0	0	0	0

Figura X: Entradas e saída do algoritmo ② antes de sua execução

128	1	2	3	4
132	5	6	7	8
136	19	22	43	50

Figura X: Entradas e saída do algoritmo ② após sua execução.

Memória da simulação sem atraso do algoritmo ②

De acordo com as matrizes iniciais $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ e $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, o algoritmo ① executa a multiplicação $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$.

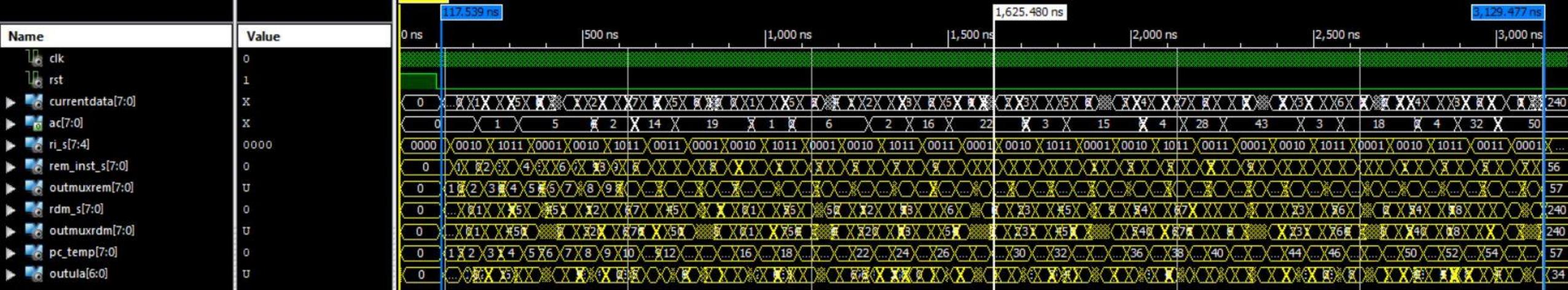


Figura X: Simulação completa com atraso do algoritmo ②.

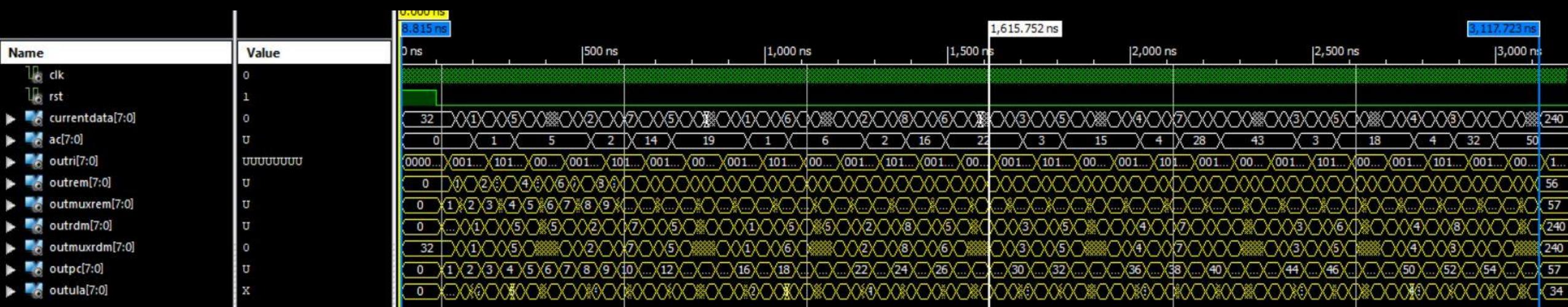


Figura X: Simulação sem atraso do algoritmo ② com os mesmos dados para comparação.

Simulação com atraso do algoritmo ②

A simulação com atraso (acima) possui áreas que claramente sofreram um atraso em relação à simulação sem atraso (abaixo).

Seu início (instrução 32 carregada no RI) ocorre 108.7ns depois do início da simulação sem atraso.

Seu meio (LDA 130) ocorre 9.7ns depois do meio da simulação sem atraso.

Seu fim (instrução HLT carregada no RI) ocorre 11.7ns depois do fim da simulação sem atraso.

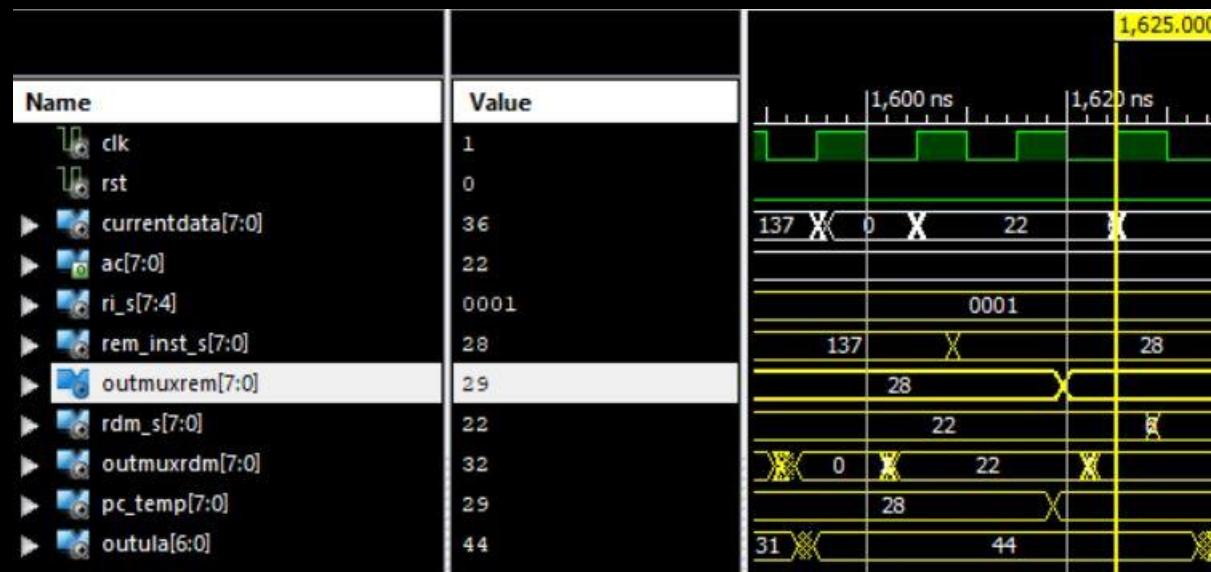


Figura X: Simulação com atraso do algoritmo ②.

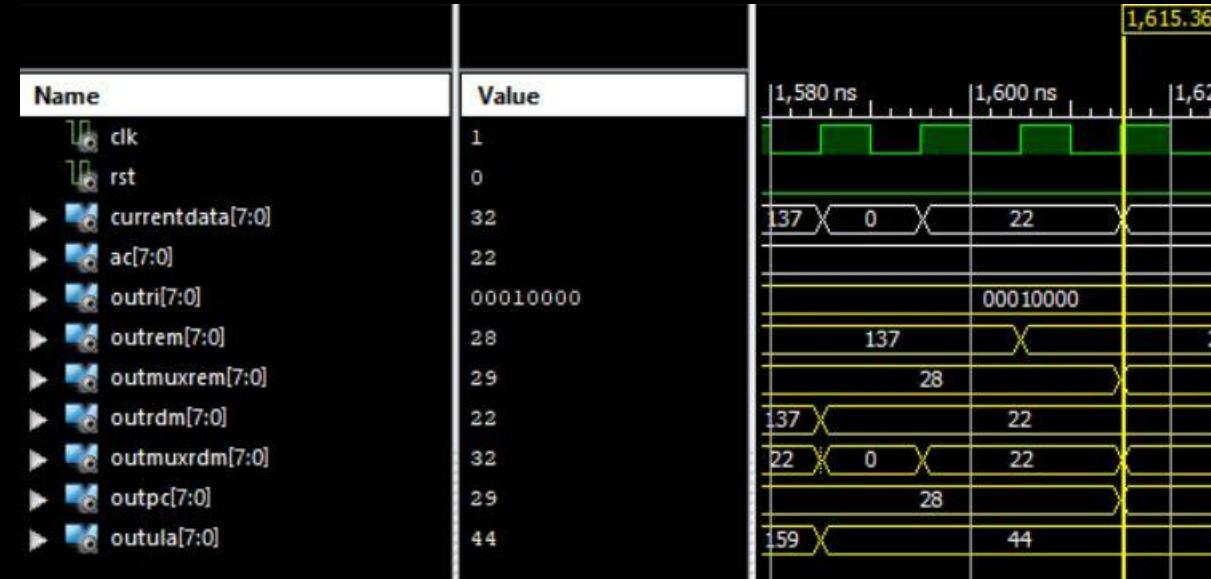


Figura X: Simulação sem atraso do algoritmo ② com os mesmos dados para comparação.

Análise pontual da simulação com atraso do algoritmo ②

No tempo exibido (meio da simulação) é possível perceber que o dado currentdata, além de possuir um atraso de 10ns na primeira figura, demora mais para se estabilizar após a troca do valor 22 para o valor 32, representado no gráfico como um "X" com linhas mais grossas.

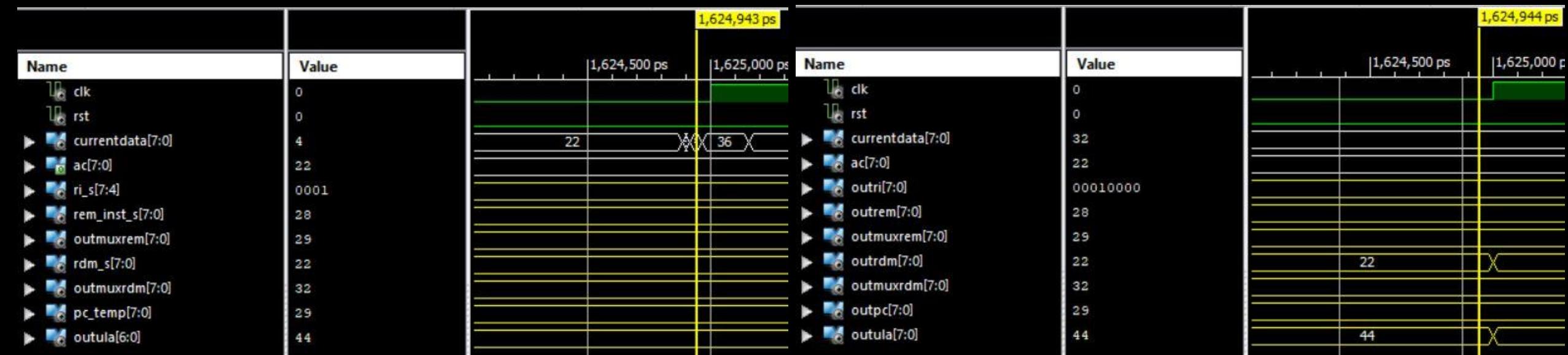


Figura X: Simulação com atraso do algoritmo ②.

Figura X: Simulação sem atraso do algoritmo ② com os mesmos dados para comparação.

Análise pontual da simulação com atraso do algoritmo ②

No tempo exibido é possível observar que o dado currentdata está trocando do valor 22 para o valor 32 no instante de 1,624,94ps na simulação com atraso, enquanto, na simulação sem atraso, já se encontra estabilizado com o valor de 32.

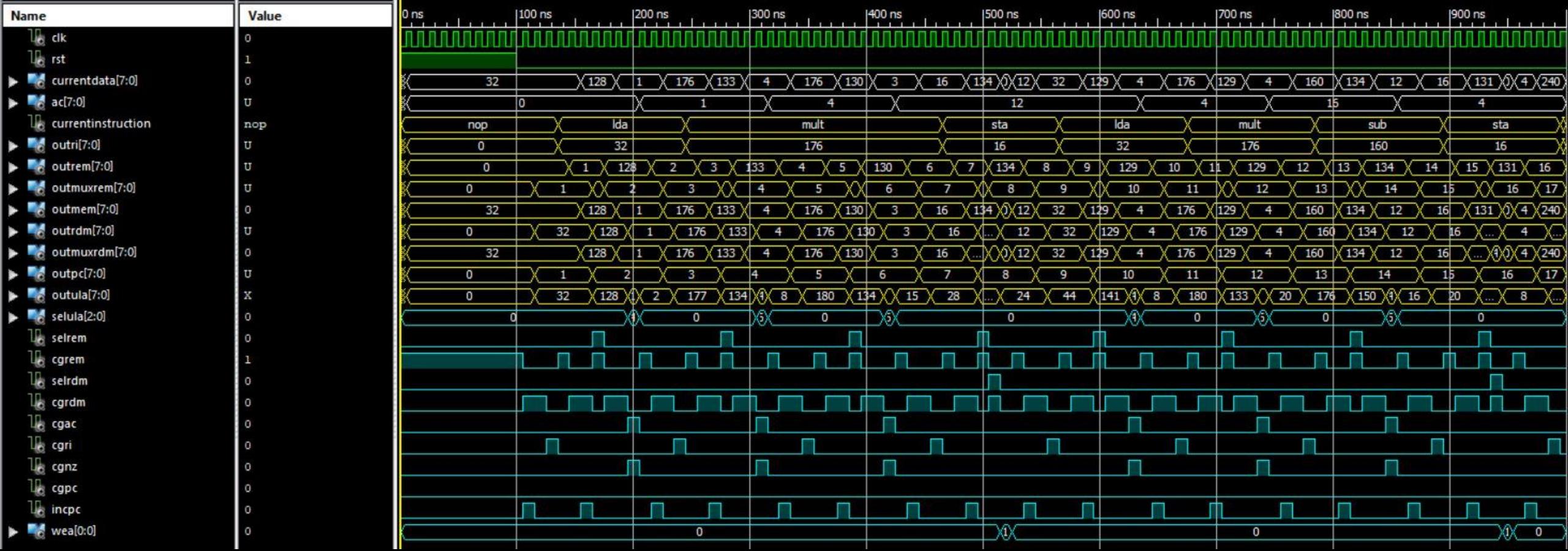
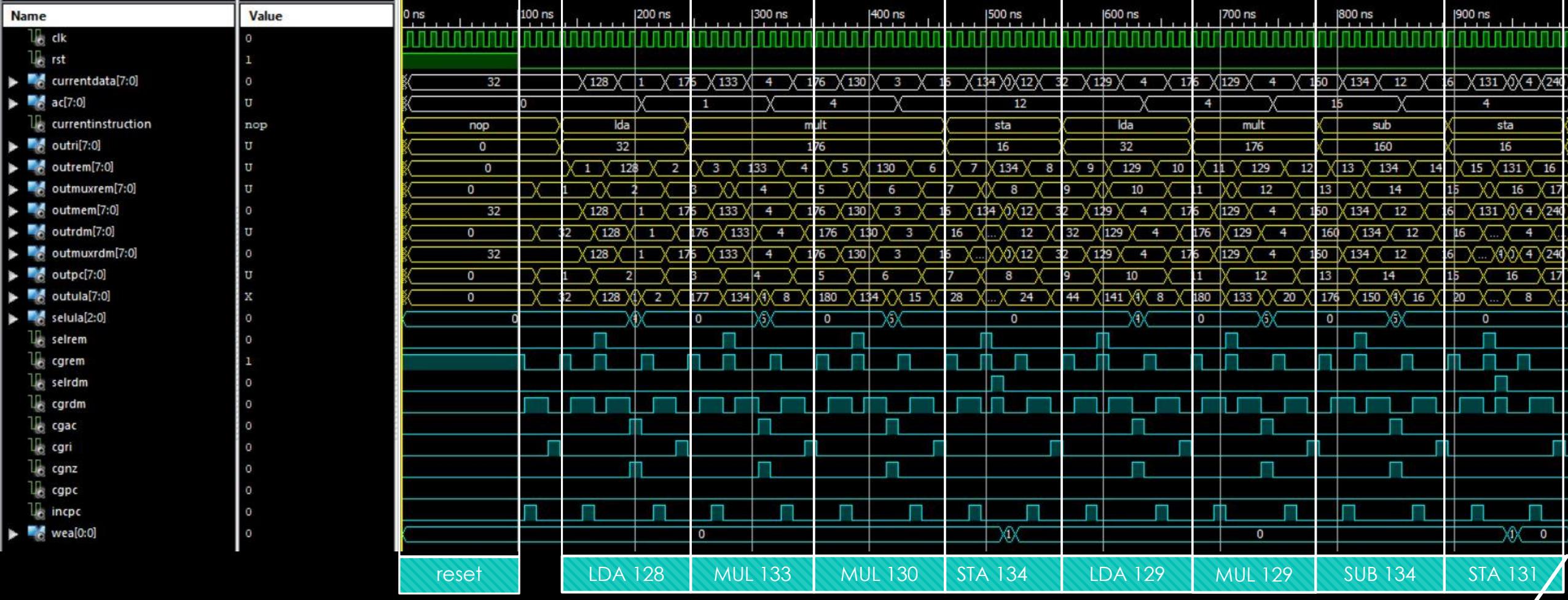


Figura X: Simulação completa sem atraso do algoritmo ③.

Simulação sem atraso do algoritmo ③



Análise da simulação sem atraso do algoritmo ③

HLT

	0	1	2	3
0	32	128	176	133
4	176	130	16	134
8	32	129	176	129
12	160	134	16	131
16	240	0	0	0

Figura X: Instruções do algoritmo ③ na memória.

128	1	4	3	0	
-----	---	---	---	---	--

Figura X: Entradas e saída do algoritmo ③ antes de sua execução

128	1	4	3	4	
-----	---	---	---	---	--

Figura X: Entradas e saída do algoritmo ③ após sua execução.

Memória da simulação sem atraso do algoritmo ③

De acordo com os dados iniciais $a = 1$, $b = 4$ e $c = 3$ o algoritmo ① executa o cálculo $\Delta = b^2 - 4 * a * c$ como $\Delta = 4^2 - 4 * 1 * 3 = \Delta = 16 - 12 = 4$. O resultado é guardado no endereço 131 (quarto endereço da segunda e terceira figuras).

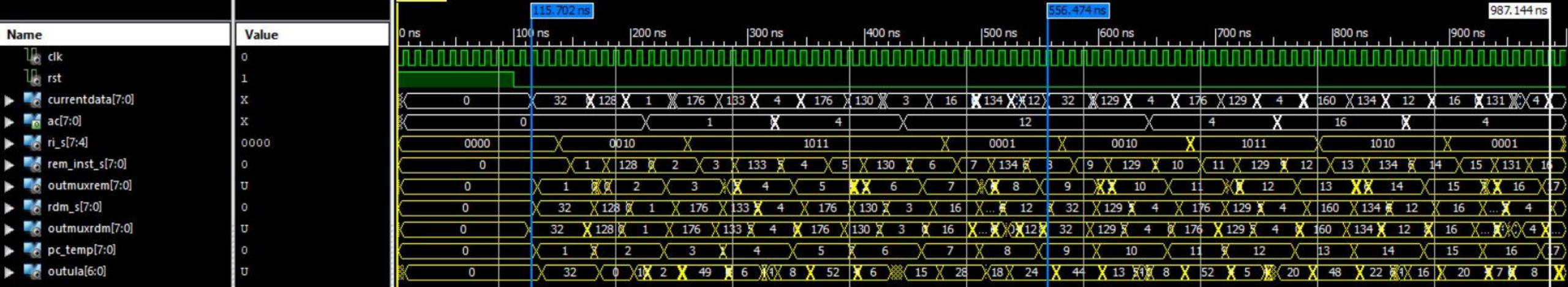


Figura X: Simulação completa com atraso do algoritmo ③.

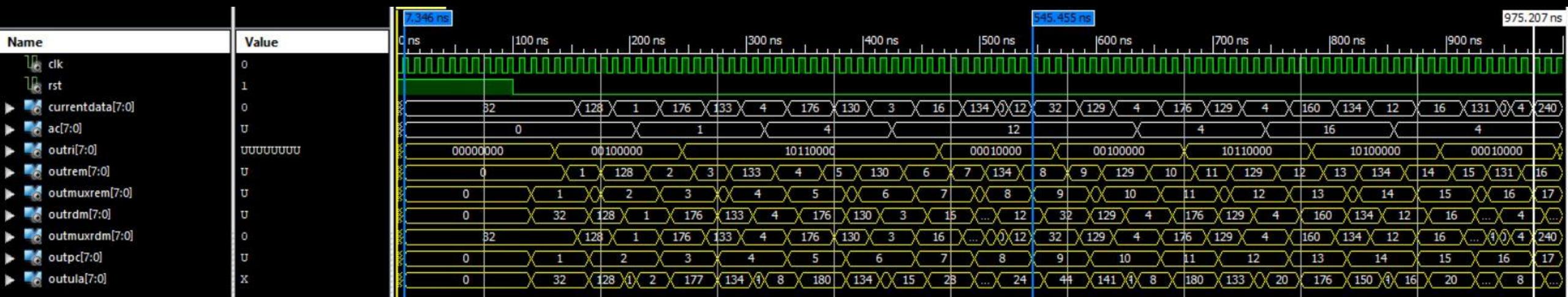


Figura X: Simulação sem atraso do algoritmo ③ com os mesmos dados para comparação.

Simulação com atraso do algoritmo ③

A simulação com atraso (acima) possui áreas que claramente sofreram um atraso em relação à simulação sem atraso (abaixo).

Seu início (instrução 32 carregada no RI) ocorre 108.3ns depois do início da simulação sem atraso.

Seu meio (LDA 129) ocorre 11.0ns depois do meio da simulação sem atraso.

Seu fim (instrução HLT carregada no RI) ocorre 11.9ns depois do fim da simulação sem atraso.

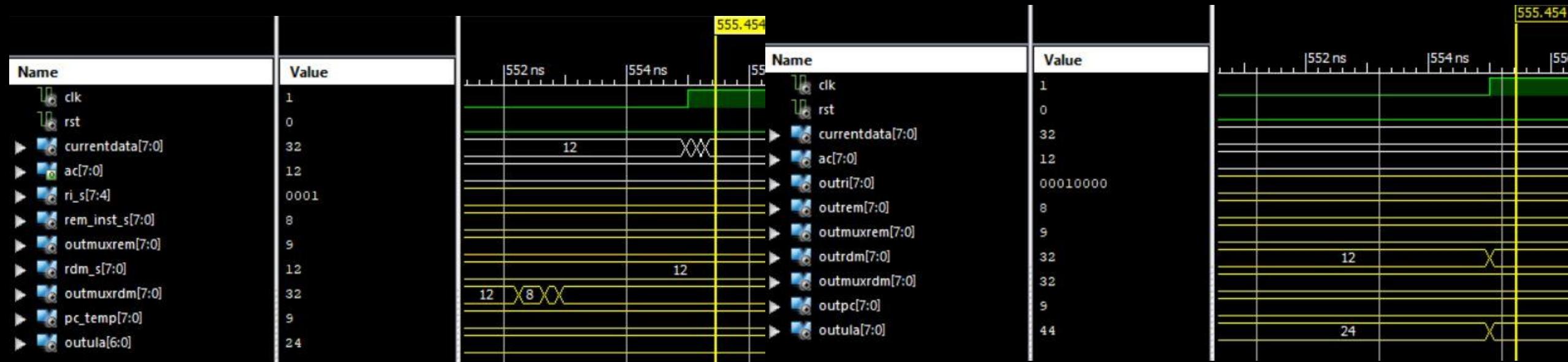


Figura X: Simulação com atraso do algoritmo ③.

Figura X: Simulação sem atraso do algoritmo ③ com os mesmos dados para comparação.

Análise pontual da simulação com atraso do algoritmo ③

No tempo exibido é possível observar que o dado **currendata** está trocando do valor 12 para o valor 32 no instante de 555.454ns na simulação com atraso, enquanto, na simulação sem atraso, já se encontra estabilizado com o valor de 32.

Dados do projeto

Desempenho por algoritmo

Programa	Nº de Instruções Executadas	Tempo de execução em # de ciclos de relógio (c.c.)	Tempo de execução em segundos (Neander operando a 50 MHz)
Soma de matrizes	13	135	0.000002691
Multiplicação de matrizes	29	307	0.000006136
Cálculo do delta de uma função	9	93	0.000001855

Dados de área e frequência

- FPGA device: SPARTAN3E-Starter Board
- Número de 4-LUTs: 133/9312.
- Número de Slice Flip-Flops: 48/9312.
- Número de BRAM: 1/20.
- Número de MULT e ADD DSP: 1/20.
- Máxima frequência de operação estimada pela ferramenta ISE: 137.855MHz.
- Máxima frequência de operação simulada: 78.808MHz.

Referências

Referências

- <https://dcc.ufrj.br/~gabriel/neander.php>