



Avaliação 04

Conversão de código em C para circuito

Curso:
Disciplina:
Professores:

Mestrado em Computação Aplicada
PSD18926 – Projeto de Sistemas Digitais
Cesar Albenes Zeferino, Douglas Rossi de Melo e
Altamiro Susin

Aluno
George de Borba Nardes

1 Introdução

Este relatório apresenta o desenvolvimento de um processador dedicado partindo de um algoritmo escrito em linguagem C. O algoritmo escolhido para o desenvolvimento foi o de retropropagação do erro de uma camada de uma rede neural artificial junto com a atualização dos pesos da camada com base no gradiente e na taxa de aprendizado. Uma máquina de estados de alto nível foi descrita partindo do algoritmo em C, seguindo a metodologia de projeto apresentada na disciplina. Da máquina de estado de alto nível derivou-se o caminho de dados e a máquina de estados finitos do controlador. Os detalhes e diagramas do projeto e da implementação serão apresentados nas seções seguintes.

2 Algoritmo em C

O algoritmo escolhido para este projeto é apresentado na [Listagem 1](#). O algoritmo trata das operações envolvidas no cálculo do gradiente dos pesos e entradas de uma camada totalmente conectada bem como na atualização dos pesos da camada. Como parâmetros de entrada, a função recebe uma matriz de pesos (weights), um vetor de vises (bias), um vetor de entrada (input), um vetor de gradientes (gradient), um vetor de erro da camada anterior (error_prev), e a taxa de aprendizado (learning_rate). INPUT_SIZE e OUTPUT_SIZE definem o tamanho do vetor de entradas e a quantidade de neurônios da camada, respectivamente. O primeiro laço de repetição (linha 5) calcula o erro da camada anterior e o terceiro laço de repetição (linha 17) calcula o gradiente em relação aos pesos e atualiza os pesos com base nesse gradiente e na taxa de aprendizado. É importante notar que a taxa de aprendizado atua dividindo o gradiente por uma potência de 2.

Listagem 1: Função de retropropagação do erro e atualização dos pesos de uma camada totalmente conectada

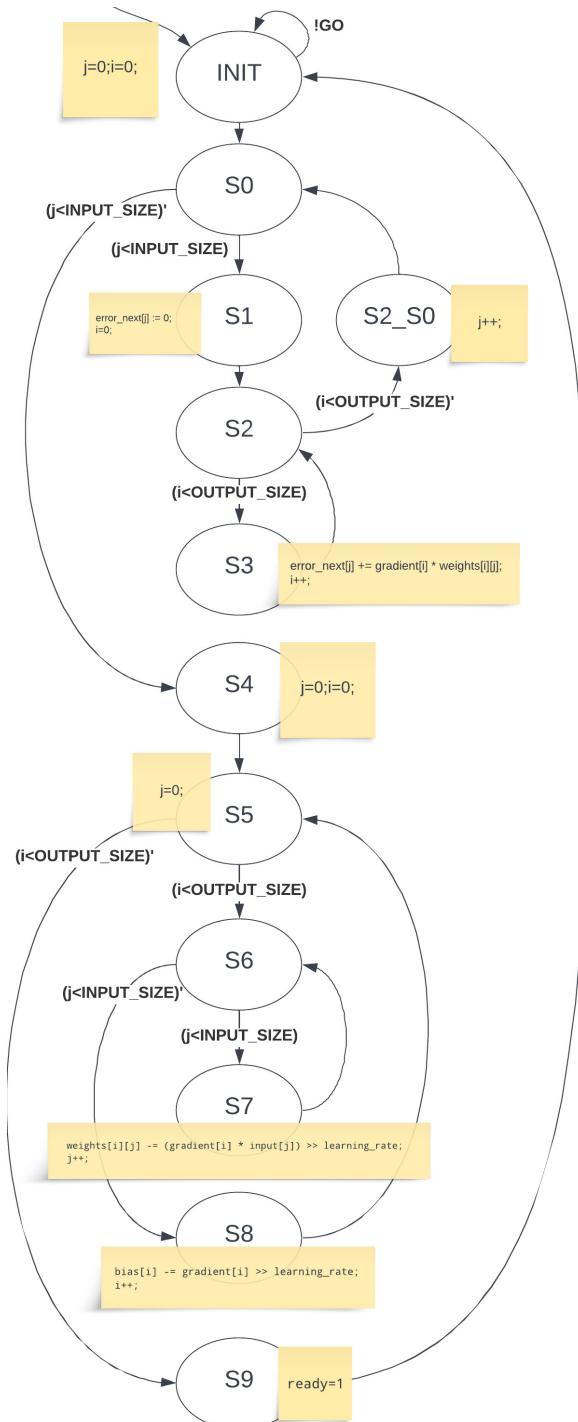
```
1 void backward(int weights[][INPUT_SIZE], int bias[OUTPUT_SIZE], int input[INPUT_SIZE],
2 int gradient[OUTPUT_SIZE], int error_prev[INPUT_SIZE], int learning_rate)
3 {
4     // Calcula o erro da camada anterior
5     for (int j = 0; j < INPUT_SIZE; j++)
6     {
7         error_prev[j] = 0;
8         for (int i = 0; i < OUTPUT_SIZE; i++)
9         {
10             error_prev[j] += gradient[i] * weights[i][j];
11         }
12     }
13     int weights_gradient;
14     int bias_gradient;
15
16     // Calcula o gradiente em relação aos pesos
17     for (int i = 0; i < OUTPUT_SIZE; i++)
18     {
19         for (int j = 0; j < INPUT_SIZE; j++)
20         {
21             weights_gradient = (gradient[i] * input[j]);
22
23             // update weight
24             weights[i][j] -= weights_gradient >> learning_rate;
25         }
26         bias_gradient = gradient[i];
27
28         // update bias
29         bias[i] -= bias_gradient >> learning_rate;
30     }
31 }
```

3 Projeto RTL

3.1 Máquina de estados de alto nível

A máquina de estados de alto nível que é apresentada na Figura 1 foi criada Partindo do código em C apresentado na seção anterior. O estado INIT inicializa o sistema. Os estados de S0 a S3 realizam as operações dos dois primeiros laços de repetição do algoritmo. Os estados de S4 a S8 realizam as operações dos dois últimos laços de repetição do algoritmo. O estado S9 sinaliza a finalização da execução do sistema.

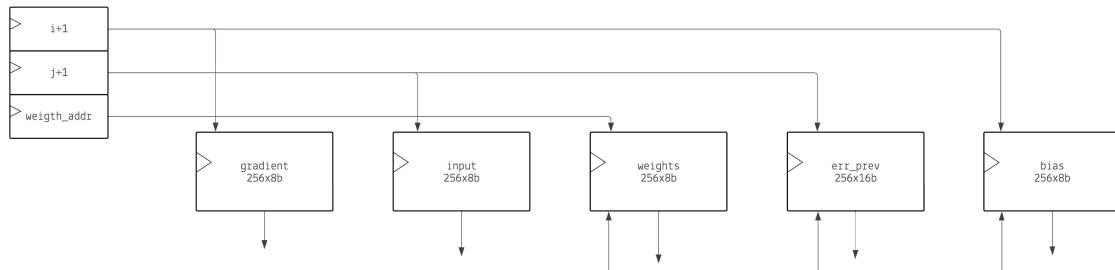
Figura 1: Máquina de estados de alto nível criada a partir do código em C.



3.2 Componentes de Memória

Um bloco de memória foi criado para cada variável recebida pela função de retropropagação, conforme mostra a [Figura 2](#). Os blocos de gradient e input são acessados apenas para leitura, tendo em vista o escopo reduzido deste trabalho¹. Os blocos weights, err_prev e bias são acessados para escrita e leitura. Os dados iniciais são carregados para memória em tempo de compilação da simulação. Os valores dos blocos weights, err_prev e bias são atualizados com base nos cálculos do sistema. Os endereços de leitura e escrita são fornecidos pelos indexadores e cada bloco possui um sinal que habilita escrita e leitura que deverá ser sinalizado pelo bloco de controle.

Figura 2: Blocos de memória e endereçadores.



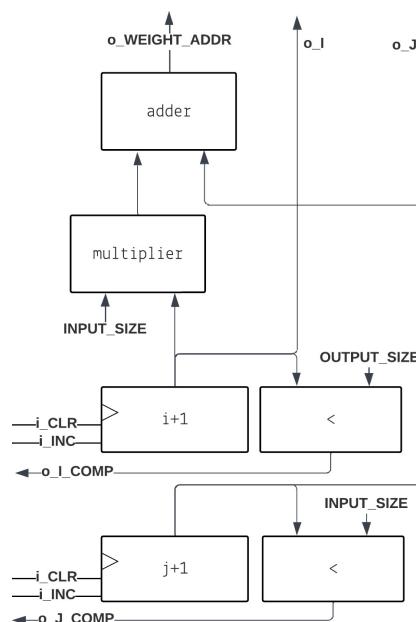
3.3 Endereçador

As variáveis que servem para endereçar os blocos de memória foram agrupadas em um único componente denominado endereçador. Este projeto adota o método de endereçamento de matrizes em vetores por meio de um multiplicador que calcula o pulo no endereço de memória conforme o indexador do eixo I obedecendo a equação

$$weight_addr = i * INPUT_SIZE + j \quad (1)$$

. A [Figura 3](#) apresenta os componentes do endereçador.

Figura 3: Endereçador dos blocos de memória.

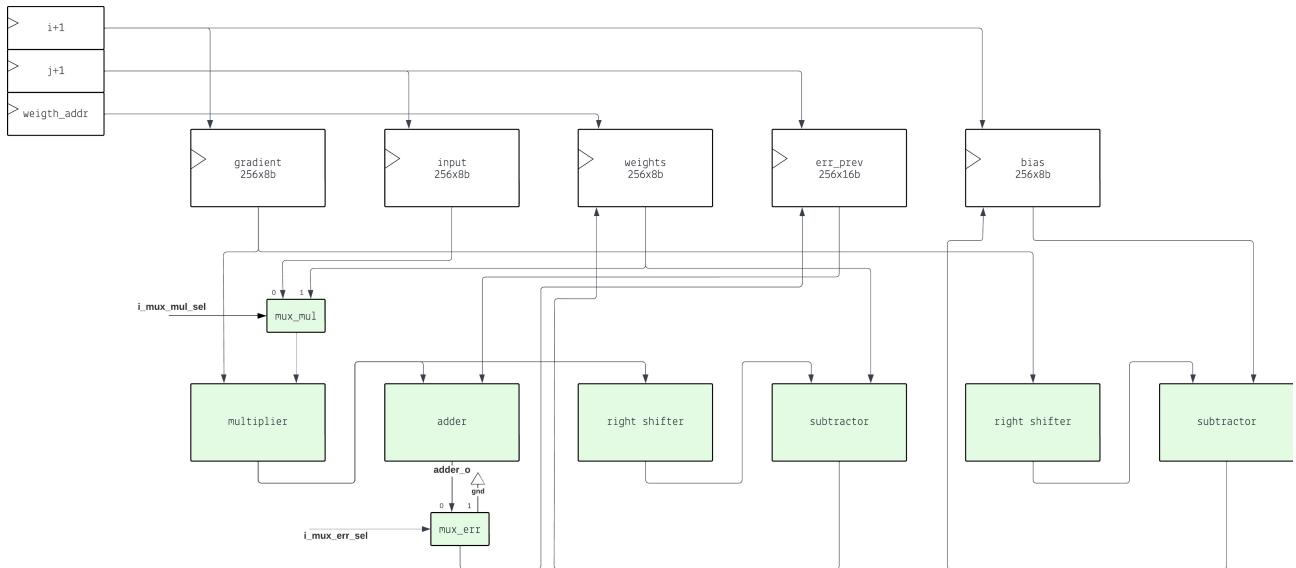


¹Numa implementação completa esses blocos teriam informações calculadas por outras camadas.

3.4 Caminho de dados

Com base na máquina de estados de alto nível, os componentes necessários foram selecionados para compor o caminho de dados. Desses componentes, buscou-se reutilizar o multiplicador por ser o componente mais custoso. Para viabilizar a reutilização do multiplicador, um multiplexador foi adicionado para chavear entre os sinais de entrada do componente. Destaca-se que, embora haja dois subtratores, nenhum tipo de paralelismo foi explorado no sistema. A Figura 4 apresenta o caminho de dados projetado para o sistema.

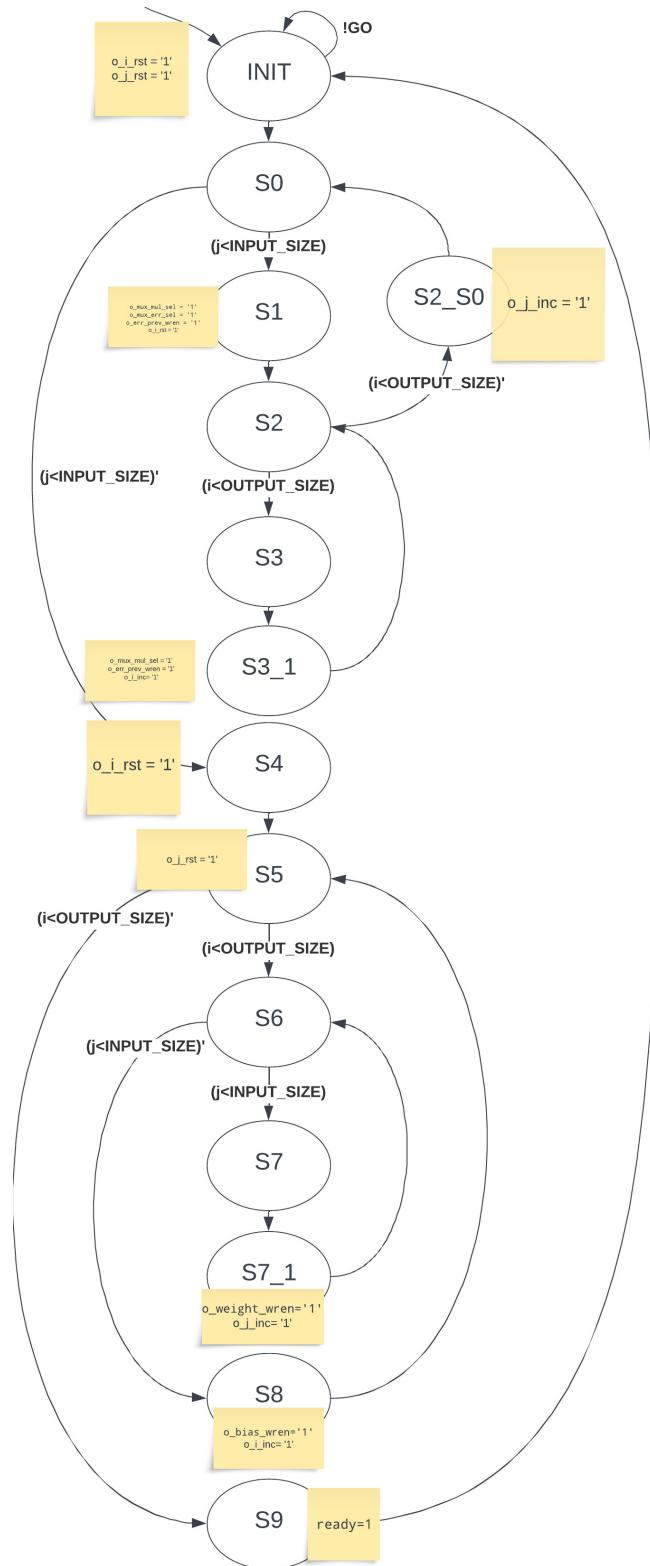
Figura 4: Caminho de dados (blocos em verde) e componentes de memória e endereçadores.



3.5 Controlador

A máquina de estados do controlador do sistema foi descrita com base na máquina de estados de alto nível. Todos as operações de alto nível realizadas na máquina de alto nível foram substituídas por sinais que controlam os componentes do bloco de caminho de dados, de endereçamento e de memória. A Figura 5 apresenta os estados da máquina de estados finita do controlador bem como os sinais que são habilitados para controle dos demais componentes.

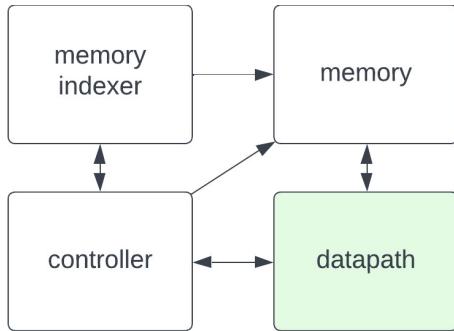
Figura 5: Maquina de estados finita do controlador.



3.6 Entidade de topo

A Figura 6 apresenta uma visão geral dos principais componentes do sistema.

Figura 6: Visão geral do processador.



4 Resultados de implementação

A Figura 7 apresenta o resultado da síntese da entidade de topo. A Figura 8 apresenta o resultado da síntese do caminho de dados. A Figura 9 apresenta o resultado da síntese dos blocos de memória. A Figura 10 apresenta o resultado da síntese do controlador. A Figura 11 apresenta o resultado da síntese do bloco endereçador.

Figura 7: Síntese da entidade de topo.

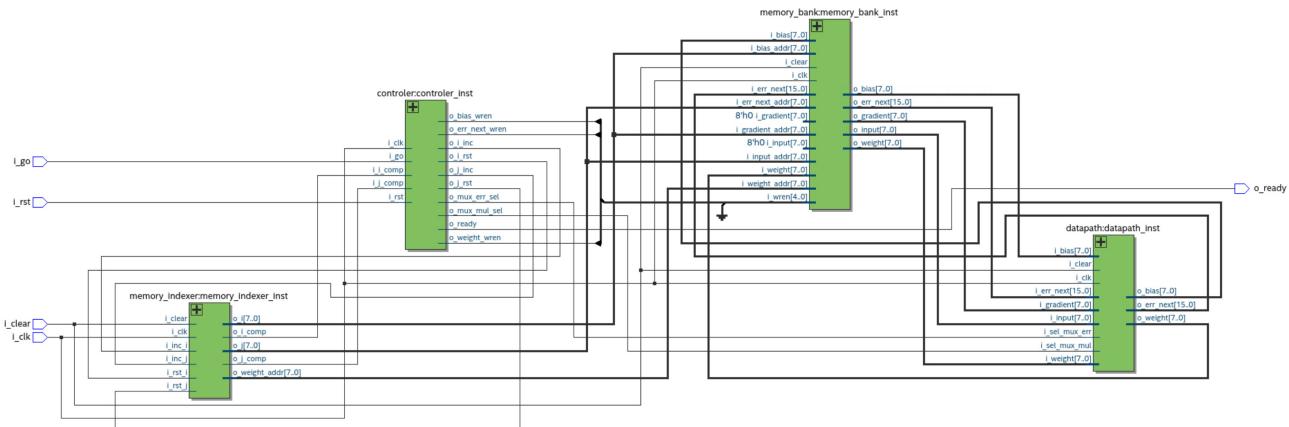


Figura 8: Síntese do caminho de dados.

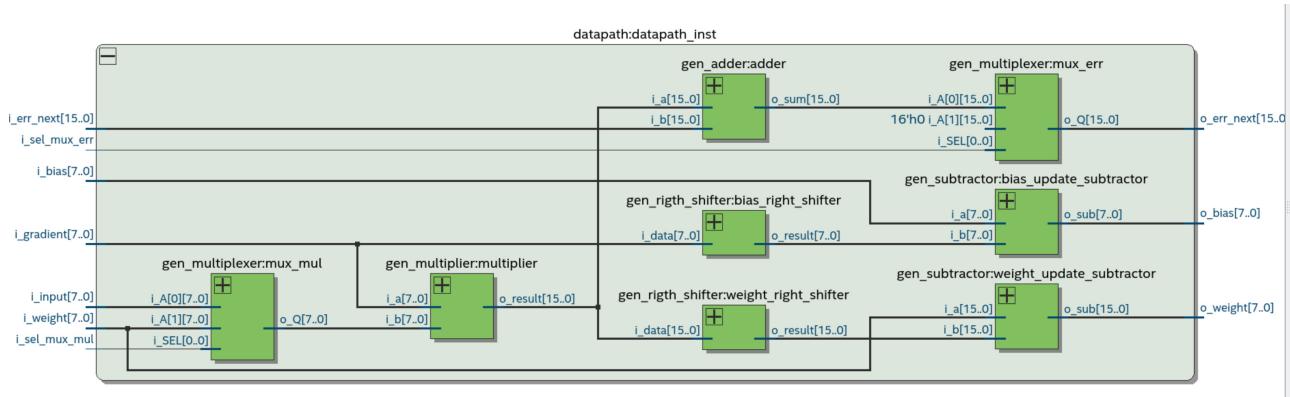


Figura 9: Síntese dos blocos de memória.

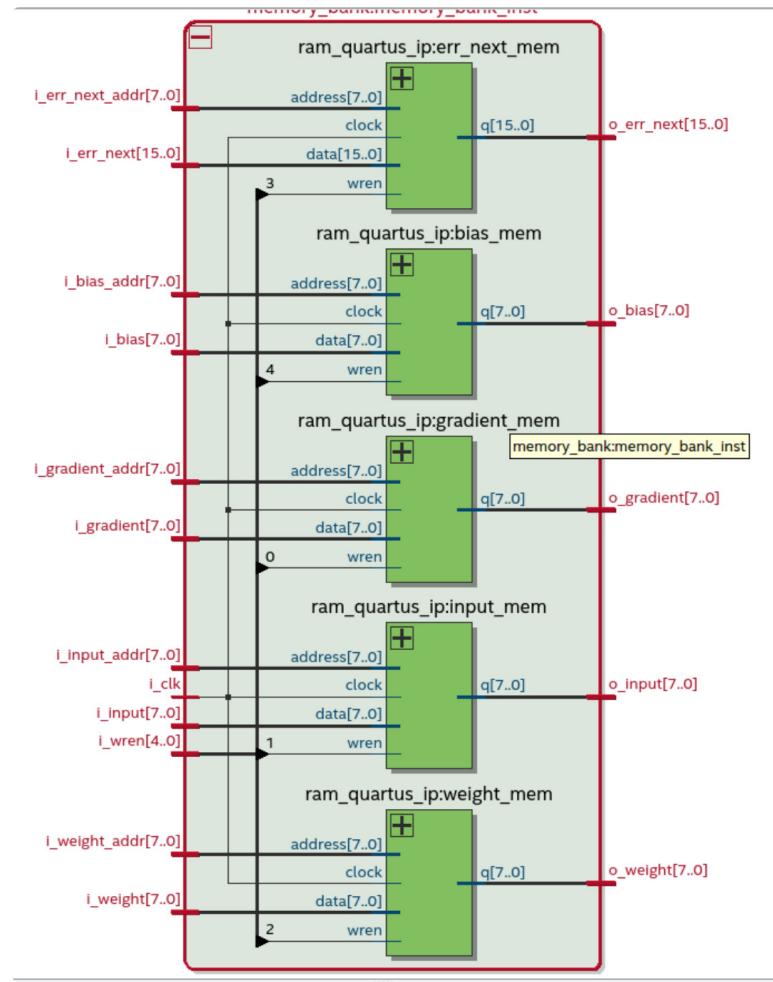


Figura 10: Síntese do controlador.

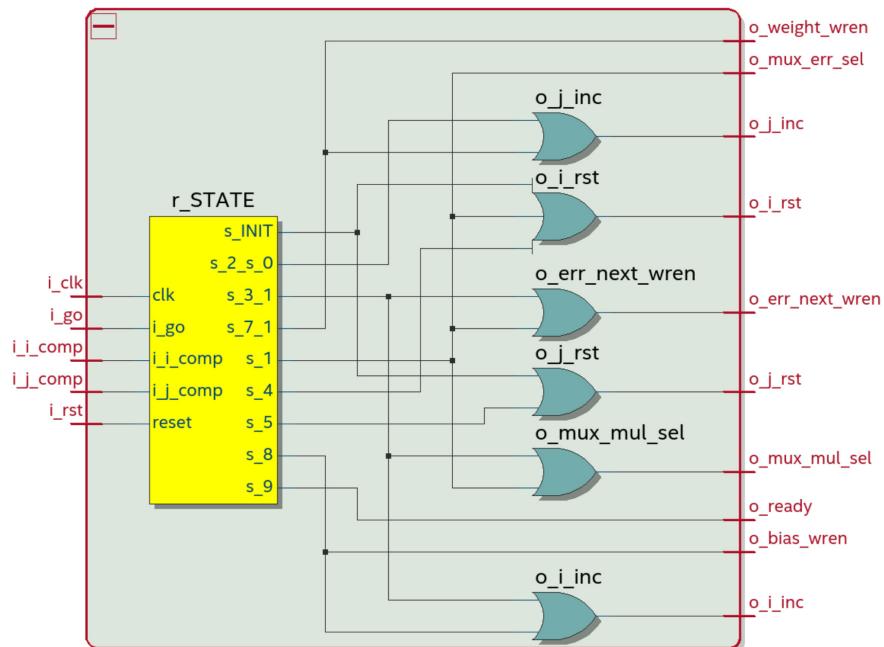
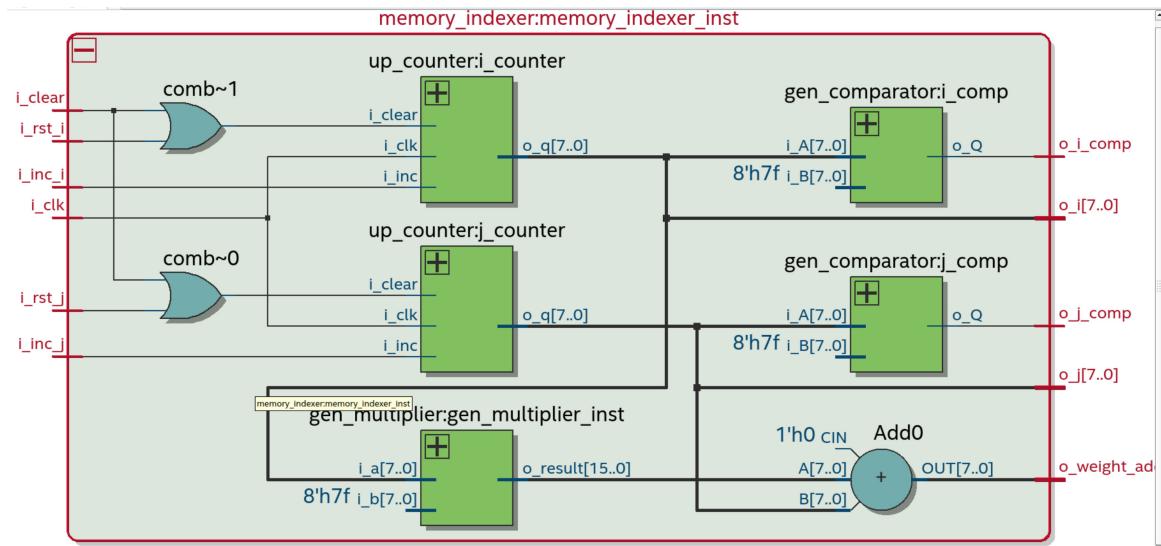


Figura 11: Síntese do endereçador.



4.1 Simulação

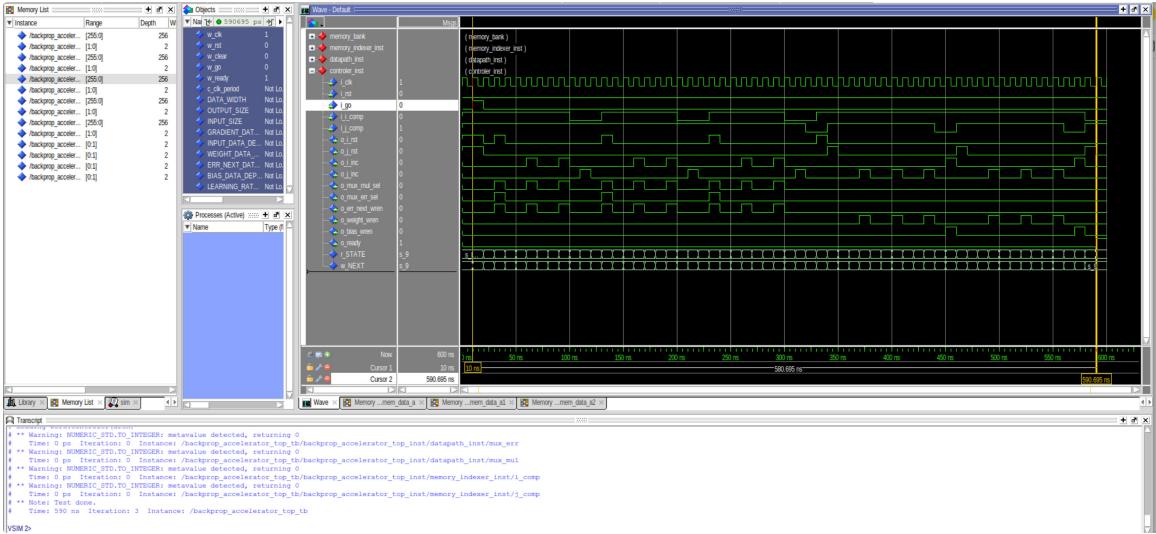
O algoritmo apresentado na [Listagem 2](#) apresenta os testes realizados na implementação em C. Os resultados esperados foram calculados manualmente e a execução do algoritmo resultou nos valores esperados o que mostrou a corretude do algoritmo.

O teste de bancada realizado sobre a descrição do circuito considerou como entrada os mesmos valores utilizados no algoritmo de teste em C. A [Figura 12](#) apresenta o diagrama de forma de onda resultante da simulação. Os valores resultantes do processamento foram verificados nos arquivos de memória. O sistema levou 58 ciclos de clock para realizar o cálculo da função considerando `INPUT_SIZE = 3` e `OUTPUT_SIZE = 2`.

Listagem 2: Código para teste da função implementada em C.

```
1 int main()
2 {
3     // definição de pesos e vieses
4     int weights[OUTPUT_SIZE][INPUT_SIZE] = {{1, 2, 3}, {3, 2, 1}};
5     int bias[OUTPUT_SIZE] = {7, 8};
6
7     // entrada da camada
8     int input[INPUT_SIZE] = {25, 15, 10};
9
10    // saída produzida pela camada (calculado previamente)
11    int output[OUTPUT_SIZE] = {92, 123};
12
13    // Exemplo de gradiente calculado externamente
14    int output_gradient[OUTPUT_SIZE] = {-5, 5};
15
16    // gradiente da entrada da camada (a ser calculado)
17    int err_prev[INPUT_SIZE];
18
19    // taxa de aprendizado (como fator de deslocamento a direita)
20    int learning_rate = 6;
21
22    // Retropropagação do erro e atualização dos pesos
23    backward(weights, bias, input, output_gradient, err_prev, learning_rate);
24
25    /**
26     * Saídas esperadas
27     * 3 4 4
28     * 2 1 1
29     */
30    print_weights(weights);
31
32    /**
33     * Saídas esperadas
34     * 8 8
35     */
36    print_vector(bias, OUTPUT_SIZE);
37
38    /**
39     * Saídas esperadas
40     * 10 0 -10
41     */
42    print_vector(err_prev, INPUT_SIZE);
43
44    return 0;
45 }
```

Figura 12: Diagrama de forma onda resultante da simulação sistema.



4.2 Uso de Recursos

A Tabela 1 apresenta um resumo do uso de recursos pelo sistema implementado. Por não aproveitar nenhum tipo de paralelismo, o circuito utilizou poucos recursos lógicos.

Tabela 1: Recursos utilizados para o circuito.

Recurso	Usado	Disponível
ALMs	39	32.070
Registradores	29	64.140
Memory bits	12.288	4.065.280

5 Conclusão

Este relatório apresentou o projeto, implementação e resultados de um processador para um algoritmo de retropropagação de erro para uma camada de rede neural totalmente conectada. Os resultados de simulação mostraram o correto funcionamento do sistema. Os códigos de implementação podem ser encontrados no repositório por meio deste link: <https://github.com/georgenardes/backpropagation_processor>.

Algumas limitações do trabalho devem ser apontadas. Primeira, a quantidade de neurônios e a quantidade de atributos de entrada da camada é pequena e não representa casos de usos comuns, nos quais a quantidade é maior que 128. Segunda, nenhum tipo de paralelismo foi explorado para acelerar o processamento das operações. Terceira, a quantidade de bits da saída do multiplicador foi reduzida simplesmente desconsiderando os bits mais significativos, o que pode gerar inconsistências quando o valor resultante necessitar de mais bits para representação. Quarta, o trabalho considerou apenas a etapa de retropropagação de uma única camada, ficando de fora a etapa de propagação e o processamento de uma sequência de camadas. Quinta, camadas com muitos neurônios podem demandar o acesso à memória externa, o qual não foi implementado.

Portanto, sugestões para trabalhos futuros incluem testar o processamento com camadas de maior quantidade de neurônios e atributos, paralelizar as operações, tratar a quantização da saída do multiplicador e processar uma sequência de camadas tanto para propagação quanto para retropropagação. Além disso, para suportar maiores cargas, será necessário implementar o acesso a memória externa externa.

Figura 13: Conteúdo da memória após execução da simulação.

