ACELERADOR DO CÁLCULO DE DISTÂNCIA EUCLIDIANA EM HARDWARE

Arlindo Fernandes de Aquino Neto, Silvio Roberto Fernandes de Araujo,

Resumo: Este trabalho apresenta uma solução heterogênea de hardware e software para aceleração do cálculo de distância euclidiana ao quadrado para o algoritmo *K-means*. O acelerador em hardware foi validado por meio de simulações em situações distintas utilizados dados sintéticos, podendo constatar o atendimento de requisito funcional, em relação a corretude, mas também sugerindo um ganho de desempenho em relação a uma versão sequencial.

Palavras-chave: FPGA ; *K-means* ; Distância Euclidiana

1. INTRODUÇÃO

Nos dias atuais, os avanços tecnológicos dos dispositivos digitais resultaram em um aumento significativo na quantidade de dados processados e armazenados em diversas áreas de aplicação. Em muitas dessas aplicações é fundamental a mineração e classificação dos dados, por isso vem sendo bastante utilizados algoritmos de clusterização que fazem o agrupamento de dados visando classificá-los em grupos (*clusters*), seguindo um determinado critério. Entre os algoritmos de agrupamento de dados, o K-means é o algoritmo popularmente usado devido à sua relativa simplicidade e eficiência. Este algoritmo agrupa dados a partir de métricas de similaridade e também é classificado como um algoritmo de aprendizado de máquina não supervisionado. Ele tem como objetivo criar K *clusters* e classificar os dados de um conjunto por meio das suas similaridades dentro desses *clusters*. Entretanto, o K-means é computacionalmente custoso, uma vez que o tempo de processamento é proporcional ao conjunto de dados e quantidade de *clusters*. Assim, para grandes conjuntos de dados, dependendo da infraestrutura computacional, pode ser inviável a utilização deste algoritmo. Portanto, surge a necessidade do desenvolvimento de uma solução que consiga aumentar o desempenho do K-means em grandes conjuntos.

Visto que a demanda por alto desempenho computacional tem aumentado o interesse no uso de soluções que combinam *hardware* e *software* para acelerar cálculos massivos, combinando desempenho com eficiência energética. Assim, não é nenhuma surpresa o emergente interesse no uso de *Field Programmable Gate Arrays* (FPGAs) para o desenvolvimento sistemas acelerados.

Portanto, neste trabalho propomos uma solução heterogênea de *hardware* e *software* para a distância euclidiana do algoritmo de K-means, no qual desenvolvemos um acelerador para ser sintetizado em FPGA com o objetivo de reduzir o tempo de execução desse algoritmo. O trabalho está organizado com a seção 2 apresentando a contextualização sobre computação de alto desempenho e o algoritmo de K-means. Na seção 3, apresentamos o acelerador proposto e os resultados e discussões, seguidas pelas seções de conclusões e referências bibliográficas.

2. REFERENCIAL TEÓRICO

Nessa seção são apresentados as técnicas e os métodos utilizados na computação de alto desempenho para aumentar o desempenho computacional e a plataforma FPGA e sua integração com a aplicação em software. Também apresentamos o algoritmo *K-means* e o cálculo de distância euclidiana, que é uma métrica de similaridade popularmente utilizada para clusterização e também um processo computacionalmente custoso, sendo por isso o foco do acelerador proposto neste trabalho.

2.1 Computação de alto desempenho

A procura por um maior desempenho, do ponto de vista de tempo de execução, proporcionou o desenvolvimento de diversas técnicas nas arquiteturas e organizações de processadores. Dentre as principais abordagens podemos destacar: pipeline, processadores superescalares, aumento da frequência de *clock* e os processadores com múltiplos núcleos (multicores), que conseguiram manter o aumento de desempenho mesmo mantendo as taxas de *clock* mais baixas. Estes modelos de processadores são rotulados como de propósito geral (CPU - *Central Processing Unit*), por possuir um caminho de dados fixo, apresenta uma grande flexibilidade como solução computacional para quaisquer problemas que possam ser descritos por meio das instruções do processador, embora o desempenho seja usualmente equivalente ao somatório do tempo de execução de cada instrução do programa.

Na ponta oposta à flexibilidade estão os processadores de propósito específico (ASIC - *Application-Specific Integrated Circuit*) que são planejados como solução de um único problema, sendo assim seu circuito é destinado exclusivamente para tal, dessa forma apresentando um melhor desempenho. Ainda existem os

processadores gráficos (GPU - *Graphics Processing Unit*) que têm sido usados também para processamento geral, mostrando-se com excelente desempenho para aplicações no estilo SIMD (*Single Instruction, Multiple Data*) mas com um grande custo computacional. O modelo intermediário entre os processadores de propósito geral e específico são os FPGAs (*Field Programmable Gate Array*), que dispõem de uma arquitetura reconfigurável formada por uma matriz de blocos lógicos reprogramáveis que combinados permitem a definição de qualquer função digital em tempo de projeto. Por via de regra, a frequência de operação do FPGA está abaixo dos processadores de propósito geral, o que representaria um menor desempenho. Entretanto sua natureza altamente reconfigurável possibilita criar soluções específicas para várias aplicações, explorando o grande potencial de paralelismo permitindo atingir desempenho até superior ao destes processadores de propósito geral. Essa baixa frequência dos FPGAs proporciona consequentemente um baixo consumo energético [1,2].

No entanto, a demanda por alto desempenho computacional (HPC - *High-Performance Computing*) e eficiência energética tem estimulado o crescimento de arquiteturas heterogêneas que combinam processadores convencionais (CPUs) com co-processadores ou aceleradores como GPUs, FPGAs e ASICs. Diversos estudos apontam a combinação CPU-FPGA como a mais promissora, sob o ponto de vista de eficiência energética, devido ao alto desempenho, baixo consumo de potência e a capacidade de reconfiguração para aceleração de aplicações diferentes [2,3].

O desenvolvimento de soluções para o ambiente heterogêneo CPU-FPGA tem estimulado diversos fabricantes de plataformas heterogêneas em investido no suporte para o programador, como a Xilinx , com sua família de SoC Zynq e o *framework* PYNQ (um acrônimo para *Python Productivity for Zynq*) que funde a produtividade do Python com a aceleração fornecida pela lógica programável no Zynq ou Zynq MPSoC.

O *framework* PYNQ e as respectivas placas compatíveis com ele, têm sido usados para construção de aceleradores para as mais diversas aplicações, como processamento de imagem [4] processamento de áudio [5], detecção de objetos [6], inteligência artificial [7] e particularmente no algoritmo de clusterização K-means [8,9,10].

2.2 K-means

O algoritmo K-means possibilita que os conjuntos de dados sejam separados e agrupados a partir de métricas de similaridade. Cada grupo criado é chamado de *cluster*. Portanto, o objetivo do algoritmo é gerar K *clusters* a partir das similaridades entre os dados de um conjunto e criar os pontos representativos centrais de cada *cluster*, que por sua vez são chamados de centróides [11]. A vista disso, K-means é frequentemente usado para reconhecer e encontrar padrões, como em segmentação de imagens [12], ou relação de proficiência e situação socioeconômica dos candidatos do ENEM [13], ou em conjuntos de dados massivos [14].

O algoritmo de K-means, a cada iteração, agrupa todos os dados com base na distância entre eles e cada centróide, que são inicializados inicialmente de forma aleatória, mas ao final de cada iteração são atualizados como a média aritmética dos pertencentes ao seu *cluster* [15,16]. Diversas métricas podem ser usadas para calcular a distância dos dados de cada indivíduo e os centróides, sendo a distância euclidiana ao quadrado (que será apresentada na seção 2.1.3) uma das mais utilizadas. Logo, é possível descrever seu funcionamento pelo fluxograma apresentado na Figura 1.

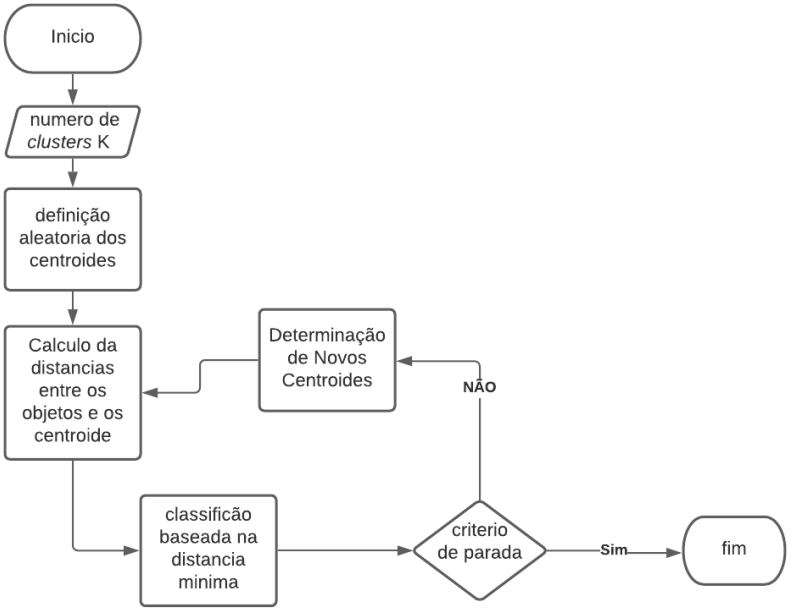

Figura 1.Fluxograma do algoritmo K-means. (Autoria própria).

A inicialização aleatória dos centróides é o passo inicial do algoritmo, portanto a primeira iteração se trata da definição aleatória da localização dos k centróides, onde k é determinado de acordo com a quantidade desejada de grupos. Como forma de otimização, os centróides iniciais são definidos aleatoriamente entre os dados a serem classificados, evitando que os centróides sejam localizados muito distantes e aumentar o tempo de convergência dos agrupamentos. A partir da segunda iteração, os centróides passam a ser determinados pela média aritmética dos dados que foram atribuídos ao cluster [17]. Além disso, ainda segundo [17], no passo seguinte, são calculados as distâncias de todos os dados a todos os centróides, as quais são usadas no passo seguinte, que faz a seleção dos dados que apresentam as menores distâncias a cada um dos centróides, de tal forma que um *cluster* será composto por um centróide e todos os dados que se aproximam mais a ele. E por fim, o teste de parada do algoritmo realiza a paralisação do mesmo ou permite o prosseguimento para a próxima iteração, caso não seja atingida a quantidade máxima de iterações desejada ou ainda houver significativa mudança nos clusters entre uma iteração e a seguinte. Dessa forma, o resultado do algoritmo são os k *clusters*, sobre os quais é possível a realização de análises quanto às características que os agrupam, além de ser possível a simplificação da coleta das informações que os dados trabalhados transmitem.

2.2.1 Distância Euclidiana

Matematicamente, a distância euclidiana entre dois pontos é o comprimento de um segmento de linha entre eles [18,19]. Se u = (x_1, y_1) e v = (x_2, y_2) são dois pontos no espaço euclidiano, então a distância euclidiana entre u e v é dada por:

$$EU(u,v) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (1)$$

Para pontos que possuírem n dimensões, como A = $(x_1, x_2,..,x_n)$ e B= $(y_1, y_2,..,y_n)$, a distância euclidiana entre A e B pode ser calculada por (2) e generalizada em (3).

$$EU(a,b) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \quad (2)$$

$$= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \quad (3)$$

Essa equação é conhecida por calcular distâncias entre diferentes tipos de objetos, como a distância de um ponto a uma linha. Em matemática, o conceito de distância foi generalizado para abstrair espaços métricos. Em algumas aplicações, especialmente ao comparar distâncias, pode ser mais vantajoso omitir a raiz quadrada final no cálculo das distâncias euclidianas. É chamado de distância euclidiana ao quadrado o resultante dessa omissão [20]. Pode ser expressa como uma soma de quadrados. Dado dois pontos P = $(x_1, x_2,..,x_n)$ e Q= $(y_1, y_2,..,y_n)$:

$$EU^2(P,Q) = (x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2 \quad (4)$$

$$= \sum_{i=1}^{n} (x_i - y_i)^2 \quad (5)$$

3. ACELERADOR PROPOSTO

Neste trabalho propomos a construção de uma acelerador em *hardware* para síntese em FPGA do cálculo da distância euclidiana conforme equação (5) da seção anterior. Assim, nesta seção será apresentada a metodologia para construção do acelerador, bem como a arquitetura proposta e resultados de simulação.

3.1. Metodologia

A metodologia de construção do acelerador em hardware segue os seguintes passos:

1 - Identificação do problema e estudo das soluções encontradas na literatura

2 - Implementação de uma solução em software que servirá de *baseline* do ponto de vista funcional e para efeito de comparação de desempenho em tempo de execução

3 - A partir do *baseline* identificar os caminhos críticos da solução que afetam o desempenho e/ou possíveis trechos paralelizáveis

4 - Desenhar a arquitetura do acelerador

5 - Desenvolver o acelerador por meio de uma linguagem de descrição de hardware (HDL - *Hardware Description Language*)

6 - Realizar testes de validação dos componentes da arquitetura individualmente e de forma integrada

7 - Desenvolver uma versão de software integrada ao acelerador

8 - Realizar teste de validação e de desempenho temporal em comparação com a versão *baseline*

Dessa forma, neste trabalho foram realizados os passos de 1 a 6 do acelerador de distância euclidiana. Para isso, inicialmente o algoritmo de K-means foi implementado em Python, de forma sequencial, para ser analisado o seu funcionamento. Visto o grande número de iterações do cálculo da distância euclidiana na execução do K-means e a facilidade de paralelismo, a função de cálculo da distância euclidiana foi isolada. A partir disso, foi feito um diagrama de bloco do acelerador, visto na Figura 3, que ilustra todos os seus componentes para a implementação do acelerador paralelizado para a distância euclidiana ao quadrado do algoritmo K-means.

A implementação do acelerador é uma solução heterogênea que implementa um acelerador em FPGA que será utilizado na plataforma Ultra96 [21]. A Figura 2 ilustra o diagrama de blocos do acelerador nesta placa. A parte em software encontra-se no processador ARM, na parte PS da placa, o qual é responsável por passar os valores parâmetros (K,N e I), ler o arquivo de dados que serão classificados e disponibilizados na memória. A partir disso, o FPGA (parte PL), tem acesso à informação dos centróides e indivíduos via DMA (*Direct Memory Access*) que envia por *streaming* para o bloco Dist. Euclidiana (destacado em azul). No final, o bloco Dist. euclidiana calcula a distância euclidiana dos dados e envia os resultados à medida que os dados vão sendo classificados para serem usados pelo *software*.

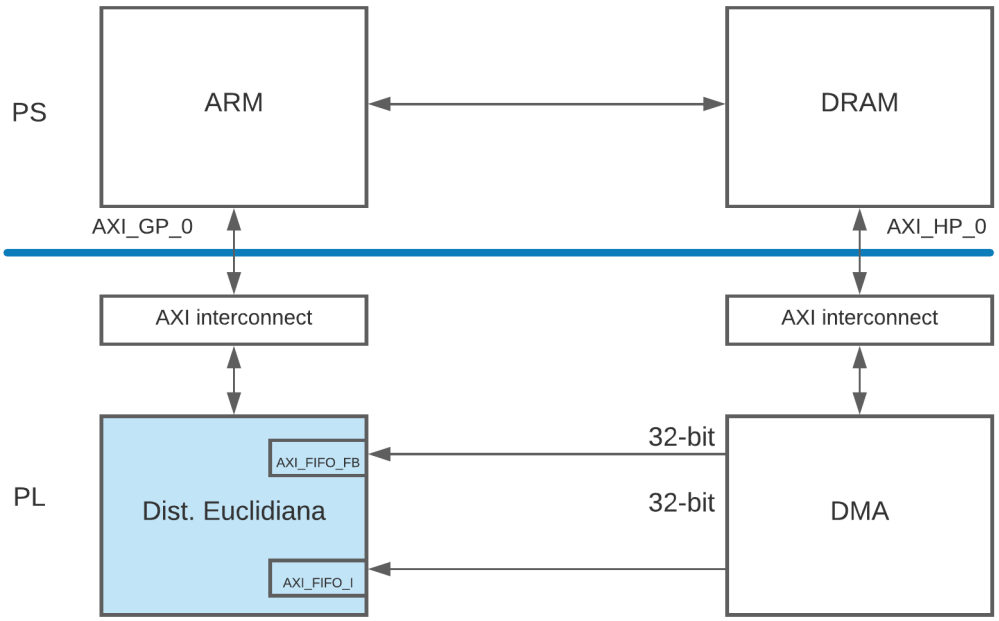

Figura 2. Diagrama de blocos do acelerador na placa Ultra96 . (Autoria própria)

A implementação heterogênea do acelerador foi projetado no padrão de DUT (*Design Under Test*) com o auxílio do EDA Playground [22], que é um ambiente online mantido pela Doulos [23], que dispões vários simuladores comerciais, suporta várias HDLs e metodologias de desenvolvimento e validação e verificação.

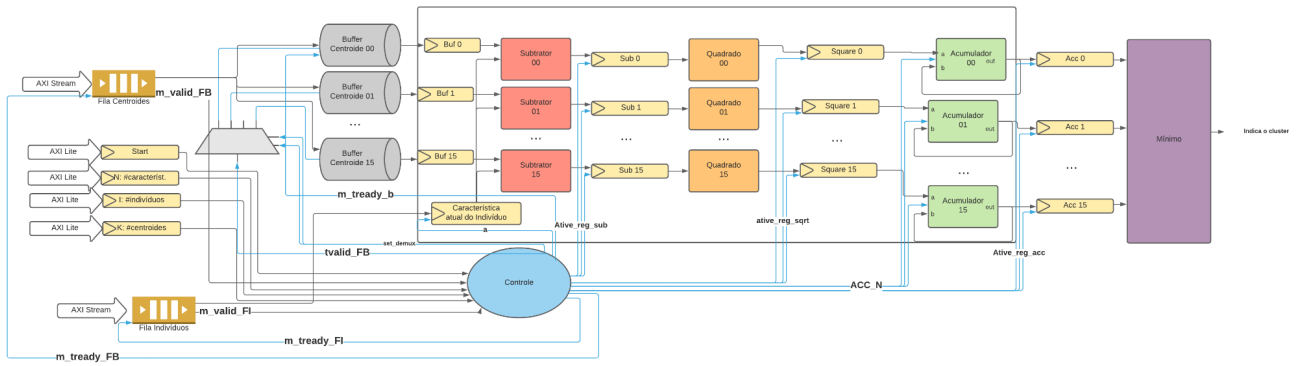

Figura 3. Diagrama de blocos da arquitetura proposta . (Autoria própria)

Assim, cada componente da arquitetura foi implementado em SystemVerilog [24] com um respectivo *testbench* para validação funcional, usado para simulação no EDA Playground. Em seguida foi realizada a integração de todos os módulos e o respectivo teste de integração também por meio de *testbench* do acelerador completo, os quais geram os resultados que serão apresentados na seção 3.3.

3.2. Arquitetura do Acelerador

O acelerador proposto, tem como entrada os parâmetros de *K, N, I* e *Start*, os quais significam respectivamente, o número de centróides, o número de características de cada indivíduo, o número de indivíduos do conjunto que serão classificados nos *clusters* e um sinal para o início do processo. Além dessas entradas, o módulo dispõe de uma saída que indica respectivamente o indivíduo e a qual cluster ele foi incluído/classificado. A quantidade de dados, provenientes dos centróides e dos indivíduos, não são fixas para atender problemas genéricos dessa natureza, mas apresentam limites pela arquitetura. Nesta versão da arquitetura a quantidade de *cluster* (K) máximo é 16, mas o usuário pode configurar com um valor em potência de 2 este máximo, enquanto a quantidade de características (N) e quantidade de indivíduos (I) são limitados pela largura de bits do canal de transmissão, ou seja, cada um pode receber qualquer valor até o limite de 2^{32}.

A transmissão das informações, ou seja, as características do centróides e indivíduos a serem classificados, são enviados pelo *software* para o acelerador por *streaming*, logo a arquitetura do acelerador dispõe de armazenamento em FIFOs (*First In First Out*) para estes dados, enquanto que os parâmetros (K, N e I) que determinam as quantidades deles são mantidas em registradores.

Os dados de todos os centróides chegam em uma mesma FIFO e são separados entre os submódulos de *buffers.* As FIFOs descartam as informações à medida que são lidas para dar espaço para novas informações que vão chegando por *streaming*. Já os buffers, mantém as informações para serem reutilizadas, por isso os centróides que precisam ser comparados com todos os indivíduos são mantidos em *buffers*. Enquanto os dados dos indivíduos são úteis apenas durante a comparação com os centróides. A arquitetura dispõe de dezesseis *buffers*, onde cada um representa um centróide fixo que não será atualizado durante a execução. A quantidade de *buffers* ativos é determinada pelo parâmetro K. Os dados que saem da FIFO de centróides chegam simultaneamente em todos os *buffers* ativos, tendo como controle de escrita um sinal *tvalid_fb***,** que habilita a escrita do dado no *buffer* desejado, por meio de um *demux*/*seletor*, regulado pelo submódulo de Controle. Já os dados da fila de indivíduos são lidos conforme o cálculo da distância euclidiana vai acontecendo.

Após o preenchimento dos *buffers* de todos os centróides, o primeiro dado da fila de indivíduos é salvo no registrador de característica atual, iniciando assim o cálculo da distância. A característica atual do indivíduo atual é comparada em paralelo com todos os centróides, executando em pipeline 3 das 4 fases do cálculo de distância. Isso quer dizer que enquanto uma característica está em um estágio do pipeline a próxima característica estará na fase anterior, permitindo também o paralelismo das etapas do cálculo de distância e consequente aumento da performance. As 4 fases do cálculo de distância são: subtração entre o valor das características do indivíduo atual e os centróides, o quadrado da subtração, o acúmulo desse valor com os valores calculados das características anteriores, e a identificação do mínimo valor acumulado entre todos os centróides. O pipeline é construído com as três primeiras fases desse cálculo de distância, e entre os respectivos módulos de hardware responsáveis pelo cálculo há registradores que armazenam o resultado intermediários, como pode ser visto na Figura 3. A última fase do cálculo de distância só pode ser realizada após todas as características serem acumuladas, a qual é realizada por uma árvore de mínimos.

Após o acúmulo de todas as características, esses valores, que são as distâncias do indivíduo a cada centróide, são salvas e passadas para o módulo da árvore de mínimos que é ilustrado na Figura 4. Esse componente compara as distâncias e retorna o menor valor de distância entre o indivíduo e o centróide mais próximo junto a um id (identificador) desde centróide. Essas comparações são realizadas paralelamente na forma de uma árvore, em que cada nó desta árvore é submódulo que recebe dois valores e retorna o menor deles. Com isso, as distâncias que foram calculadas chegam na primeira camada da árvore e são comparadas paralelamente em pares

e passadas para próxima camada até chegar na camada raiz, onde será retornado a distância entre o indivíduo e o centróide mais próximo e um id do centróide. Ao todo, a árvore possui 4 camada, onde a primeira possui 8 comparadores (para receber dos 16 distâncias), a segunda com 4 comparadores para receber os resultados da camada anterior, seguida por uma camada com 2 e a última com 1 comparador. A depender do número de centróides configurados o número de comparadores e/ou camadas ativados se altera até os limites anteriormente apresentados.

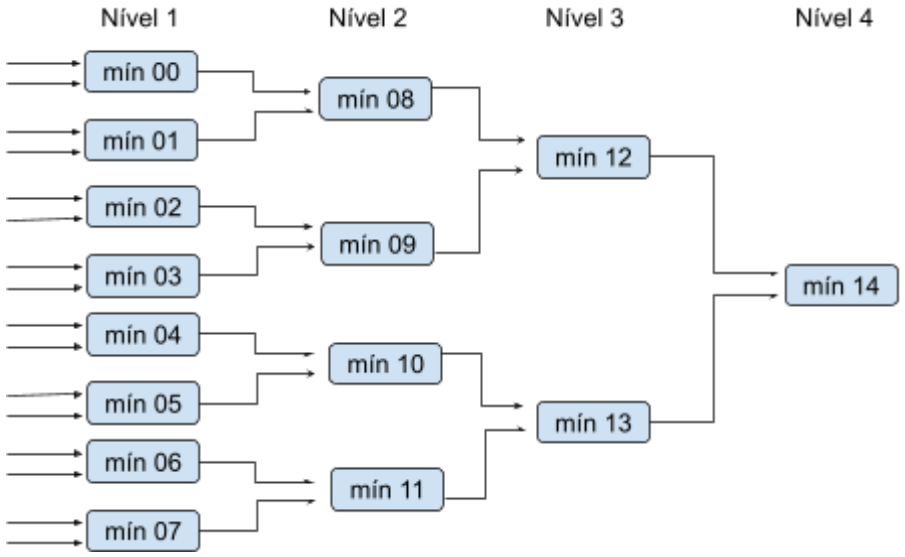

Figura 4. Árvore de mínimos para comparação de distâncias (Autoria própria).

Por fim, o módulo de controle é responsável por gerir todo o funcionamento dos demais componentes da arquitetura do acelerador, e é implementado por por meio de uma máquina de estados, conforme Figura 5. O controle usa as entradas *K, N, I, Start* e um sinal de dado válido(*valid_fi)* vindo da FIFO de indivíduos para tomada de decisão de alternância entre os estados. O estado inicial é IDLE onde espera-se pelo sinal de Start para iniciar o processo e passar para o estado de SET_CENTROID, no qual inicia-se a gravação dos dados nos buffers específicos por meio da seleção adequada do Dmux. No estado BUFFER_WRITE, é feita a escrita propriamente dita dos dados no *buffer* de um centróide, incrementado a quantidade de dados/características (registrador count_n) recebidas por centróide, retornando para o SET_CENTROID onde serão verificados se todos os centróides já foram gravados(registrador count) quando passa para o estado de espera dos dados do indivíduo atual (IDLE_I). Em seguida, os dados do indivíduo são lidos e comparados paralelamente com os dados dos centróides, de modo que o controle contabiliza a quantidade de dados lidos e habilita a escrita nos registradores do pipeline, até que todos os dados sejam contabilizados para dar início a árvore de mínimos.

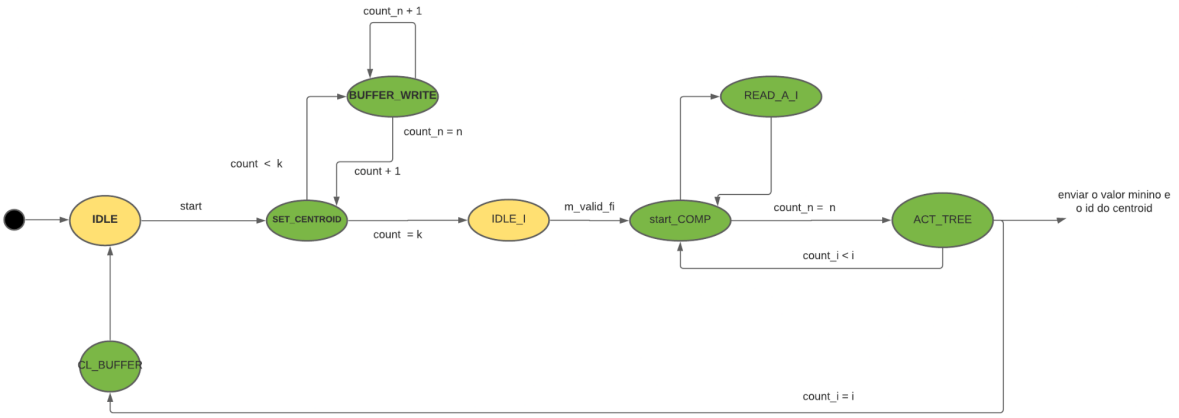

Figura 5. Máquina de estados do módulo de controle (Autoria própria)

No estado Start_COMP inicia-se o cálculo da distância do indivíduo com os centróides, no estado READ_A_I começamos a ler os dados do indivíduo da FIFO, após a leitura de uma característica do indivíduo, retornamos para o Start_COMP onde é verificado se todas as características do indivíduo já foram lidas (registrador count_n). Ao final, o controle entra no estado estado de ACT_TREE, onde é avaliado se a quantidade de indivíduos computados (registrador count_i) é igual a de indivíduos esperados (I), se não, o controle retorna para o estado de Start_COMP e é feito o cálculo da distância euclidiana do próximo indivíduo

até que os todos os indivíduos sejam computados, passando para o estado de CL_BUFFER onde é feito o reset dos *buffers* e retornamos para o estado inicial (IDLE).

3.3. Resultados e Discussões

Nesta seção serão discutidos os resultados obtidos por meio dos *testbenchs* do acelerador proposto em situações distintas utilizando dados sintéticos com o intuito de facilitar a análise dos resultados obtidos com os resultados esperados, depois é observado o impacto do aumento do número de centróide e do número de características no tempo de execução do acelerador.

O acelerador proposto foi simulado em três casos de uso, com uma amostra de dados sintéticos sendo representados da seguinte forma: Centróide X = (x_1, x_2,..,x_n) e indivíduo Y = (y_1, y_2,..,y_n), onde cada um valor x_n e y_n É uma das características dos dados da amostra, tendo o intuito de facilitar os cálculos a seguir, foi utilizado o mesmo valor para cada característica dos dados da seguinte maneira: Centróide X = (x, x, ..., x) e indivíduo Y = (y, y, ..., y). No primeiro caso utilizamos um conjunto de dados com quatro indivíduos para dois centróides onde cada um possui quatro características. No segundo caso aumentamos o número de centróides do conjunto para quatro. E no último, mantemos a quantidade de indivíduos e centróides, mas aumentamos para oito o número de características em todos os dados do conjunto. Com o objetivo de verificar o funcionamento correto da arquitetura bem como a latência para obtenção dos resultados, utilizamos um conjunto de dados bem pequenos com valores inteiros para as características.

Na primeira simulação foi feito um *testbench* com o seguinte conjunto de dados: Centróide 1 = (2, 2, 2, 2), Centróide 2 = (3, 3, 3, 3) e indivíduos representados por: Q = (4, 4, 4, 4), P = (2, 2, 2, 2), T = (1, 1, 1, 1) e S = (0, 0, 0, 0). Neste caso, cada indivíduo e centróides possuem 4 características. utilizando a distância euclidiana ao quadrado, presente na equação (4), no indivíduo Q em relação ao centróide 1, chegamos ao seguinte resultado:

$$D\_EU^{\,2}(1, Q) = (2 - 4)^{\,2} + (2 - 4)^2 + (2 - 4)^2 + (2 - 4)^2 = 16$$

Para o mesmo individuo Q em relação ao centróide 2 temos a distância:

$$D\_EU^{\,2}(2, Q) = 4$$

Observando as distâncias vemos que indivíduo Q esta mais proximo do centróide 2, classificando como pertencente ao grupo dele, ou seja, ao cluster 2.

Calculando as distâncias dos indivíduos restantes temos as seguintes distâncias:

$D\_EU^{\,2}(1, P) = 0$, $D\_EU^{\,2}(2, P) = 4$

$D\_EU^{\,2}(1, T) = 4$, $D\_EU^{\,2}(2, T) = 16$

$D\_EU^{\,2}(1, S) = 16$, $D\_EU^{\,2}(2, S) = 32$

Com os resultados obtidos observamos que indivíduos P ,T e S estão mais próximos do centróide um, o que classifica os indivíduos pertencentes ao grupo um.

O *hardware* proposto é um bloco IP (*Intellectual Property*), que será sintetizado em FPGA. Suas interfaces são: *K, N, I, Start, FI_m00_axis_tdata, FB_m00_axis_tdata, out* e *ID_out*, conforme na Figura 6. As entradas K, N, I possuem 32 bits e o Start tem um bit, os quais tem o objetivo de receber os valores que definem a quantidade de centróide, o número de características, a quantidade de indivíduos que serão classificados e o sinal de início do processo respectivamente. Já as entradas *FI_m00_axis_tdata* e *FB_m00_axis_tdata* possuem a mesma quantidade de bits e tem o objetivo de receber os dados dos centróides e indivíduos que serão salvos na FIFOS de buffers e indivíduos, nessa ordem. As saídas *out* e *ID_out* tem o propósito de entregar os resultados dos menores valores do cálculo da distância euclidiana dos indivíduos classificados e o id do centróide mais próximo a cada indivíduo, na devida ordem.

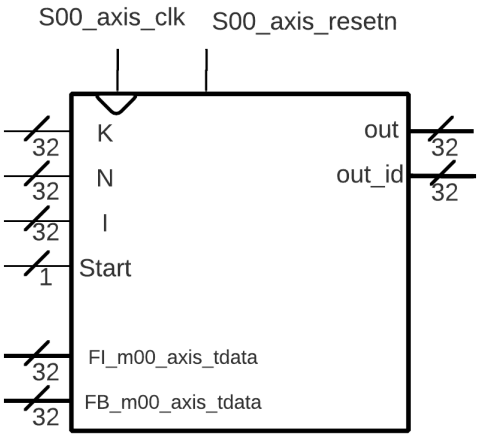

Figura 6. DUT do acelerador proposto (Autoria própria)

A forma de onda da simulação do acelerador proposto executado no EDA Playground para o primeiro caso de uso é apresentada na Figura 7. Podemos observar por meio das saídas de menor distância encontrada(out) e identificador do *cluster* que o indivíduo foi classificado (ID_out) que obtivemos os resultados como esperado, tendo o primeiro dado válido com 48 ciclos de clock e a cada 10 ciclos a classificação de um outro indivíduo.

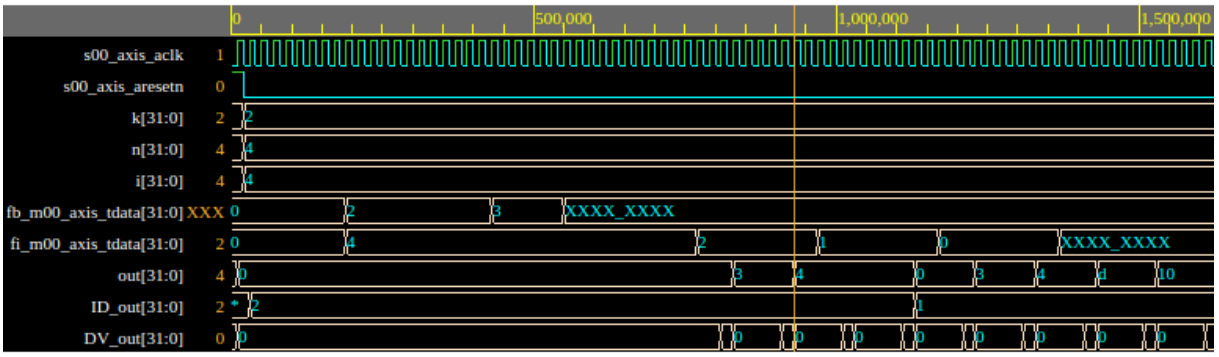

Figura 7. Simulação do caso um (Autoria própria).

Na simulação do segundo caso de uso apresentado na Figura 8. Foi mantido o mesmo grupo de indivíduos Q , P , T e S e adicionamos aos Centróides os seguintes dados: $3 = (5, 5, 5, 5)$ e $4 = (1, 1, 1, 1)$, calculando as distâncias dos indivíduos para os novos centróides, temos os seguintes valores.

$D\_EU^{2}(3, Q) = 4$, $D\_EU^{2}(4, Q) = 36$

$D\_EU^{2}(3, P) = 36$, $D\_EU^{2}(4, P) = 4$

$D\_EU^{2}(3, T) = 64$, $D\_EU^{2}(4, T) = 0$

$D\_EU^{2}(3, S) = 100$,$D\_EU^{2}(4, S) = 4$

Comparando as distâncias dos indivíduos até os centróides 1, 2, 3 e 4 classificamos os dados da seguinte forma: Q no *cluster* 3, P no *cluster* 1 e T e S no *cluster* 4. Com o aumento da quantidade de centróides foi observado uma alteração no tempo de entrega do primeiro dado de 48 para 58 ciclos, um adicional de 10 ciclos em consequência do aumento de dados armazenados nos *buffers*, porém mantivemos o tempo de 10 ciclos de diferença entre cada uma das classificações seguintes. Devido ao paralelismo das operações, a quantidade de centróide não intervém no tempo da classificação dos indivíduos, uma vez que são comparados simultaneamente a todos os centróides.

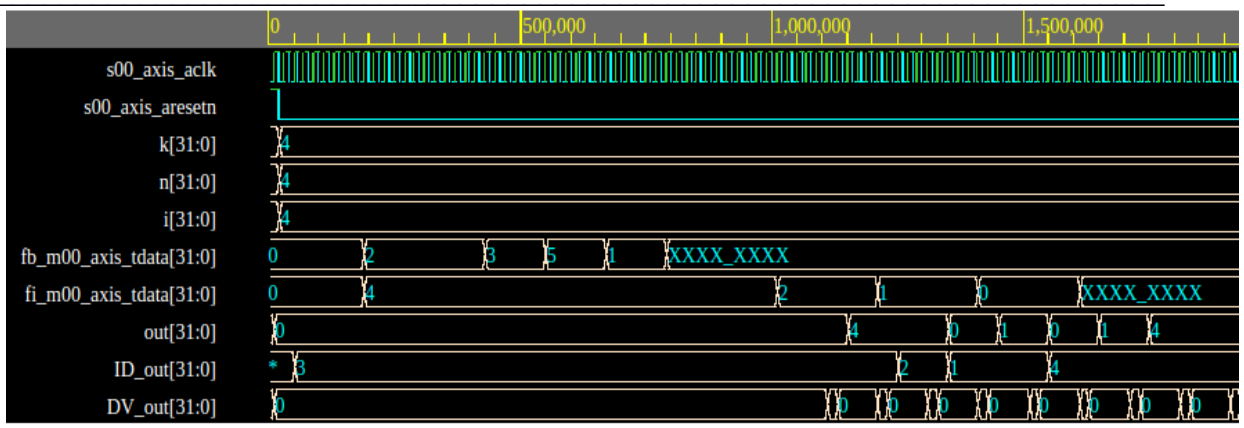

Figura 8. Simulação do caso dois (Autoria própria).

Na simulação do terceiro caso de uso apresentado na Figura 9, foi feito um testbench mantendo os centróides 1 e 2 e indivíduos Q, P, T, S porém aumentado de quatro para oito características, resultando nos seguintes dados: Centróide 1 = (2,2,2,2,2,2,2,2), Centróide 2 = (3,3,3,3,3,3,3,3) e indivíduos: Q = (4,4,4,4,4,4,4,4), P = (2,2,2,2,2,2,2,2), T = (1,1,1,1,1,1,1,1) e S = (0,0,0,0,0,0,0,0). Com isso, conseguimos as mesmas classificações do primeiro caso, porém com os valores das distâncias dobrados. Observando a simulação verificamos que o tempo de entrega do primeiro resultado foi de 66 ciclos e o intervalo para a classificação dos indivíduos seguintes foi para 18 ciclos. Com isso percebemos que quantidade de características dos dados age em todo tempo de execução, em virtude do maior número de características a serem gravadas nos *buffers* e comparadas para a classificação do indivíduo.

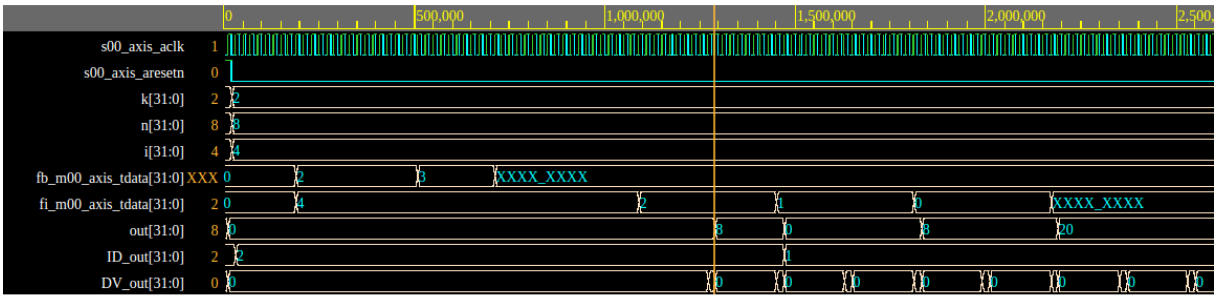

Figura 9. Simulação do caso três (Autoria própria).

A versão atual da arquitetura ainda não foi integrada ao *software*, portanto a respeito do desempenho do *hardware* como acelerador para o K-means, podemos fazer uma análise teórica, por meio de algumas suposições do funcionamento do algoritmo de forma sequencial, como normalmente acontece em uma implementação em *software*. Supondo que o algoritmo de K-means é executado de forma sequencial em uma máquina hipotética que executa uma operação por ciclo, assim para o cálculo da distância entre um indivíduo e um centróide, teríamos: 1 ciclo para a subtração, 1 ciclo para elevar ao quadrado e 1 ciclo para acumular o resultado anterior, ou seja, consumindo 3 ciclos por características. Em seguida, a comparação de todas as distâncias levará K-1 ciclos para cada indivíduo, onde K é o número de centróides. Logo, para K centróides e Y indivíduos com C características cada, o tempo para classificar cada indivíduo será de ((3*C)*K*Y) + Y*(K-1) ciclos, sem levar em consideração o tempo de acesso à memória para buscar os dados. Se usamos os parâmetros da primeira simulação: 2 centróides, 4 indivíduos e 4 características no cenário desta máquina hipotética, temos um total de (3x2x4x4)+(4*1) = 100 ciclos para classificarmos todos os indivíduos.

O resultado de simulação do acelerador para este mesmo cenário levou 48 ciclos para classificar o primeiro indivíduo e mais 10 ciclos para cada indivíduo seguinte devido o pipeline, resultando um total de 78 ciclos para classificar os quatro indivíduos. Em relação a execução sequencial hipotética o acelerador é teoricamente 1,28x mais rápido. Vale salientar que o tempo da simulação levou em consideração o tempo de acesso aos dados dos *buffers*, logo se esse tempo for desconsiderado uma vez que a máquina hipotética não contabilizou o acesso a memória, o acelerador leva 3*C*Y + Y*($\log_2$K) ciclos. Assim, para este exemplo, o acelerador precisa de 28 ciclos para classificar os 4 indivíduos, resultando em uma aceleração de 3,5x em relação ao modelo sequencial da máquina hipotética.

4. CONCLUSÕES

Neste trabalho foi apresentado um acelerador para a distância euclidiana, com o objetivo de aumentar o

desempenho da execução do algoritmo K-means. A partir de simulações podemos constatar o atendimento de requisito funcional, em relação a corretude, mas também podemos fazer inferências a respeito das variáveis que impactam no tempo de respostas do acelerador.

O ganho de desempenho do acelerador é proveniente do uso de paralelismo na comparação de indivíduos com centróide e uso de pipeline nas etapas do cálculo da distância, além do paralelismo empregado na árvore de mínimos usada na classificação. No que diz respeito ao pipeline, o aumento da quantidade de características e da quantidade individuais tornará a eficácia do acelerador ainda mais evidente em comparação a soluções puramente em *software.*

Como trabalhos futuros, o acelerador será sintetizado para o FPGA da plataforma Ultra96. Em seguida, vamos integrá-lo ao software em python utilizando o *framework* PYNQ, para utilizar dados reais e para medir com exatidão o desempenho do acelerador em relação a versão *baseline* sequencial em software, e posteriormente comparado a uma versão paralela do *software* Além disso, novas versões do acelerador poderão incluir a atualização dos centróides e assim realizar todo o algoritmo de K-means.

5. REFERÊNCIAS BIBLIOGRÁFICAS

[1] A. Putnam et al., “A reconfigurable fabric for accelerating large-scale datacenter services,” in 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), 2014, pp. 13–24.

[2] Y. k Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, “A quantitative analysis on microarchitectures of modern CPU-FPGA platforms,” in 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2016, pp. 1–6.

[3] E. S. Albuquerque et al., “An FPGA-based accelerator for multiple real-time template matching,” in 2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI), 2016, pp. 1–6.

[4] SILVA, T. ; AQUINO NETO, A.; Fernandes, S.R. . GrayScaleAccel: Acelerador de Escala de Cinza em FPGA. In: Workshop de Iniciação Científica em Arquitetura de Computadores e Computação de Alto Desempenho, 2020, Virtual. XXI Workshop de Iniciação Científica em Arquitetura de Computadores e Computação de Alto Desempenho, 2020.

[5] Stornaiuolo, L., Santambrogio, M. and Sciuto, D. (2018). On How to Efficiently Implement Deep Learning Algorithms on PYNQ Platform. In 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI).

[6] Elsaadouny, M., Barowski, J. and Rolfes, I. (2020). FPGA Based Accelerator for Buried Objects Identification. In 2020 43rd International Conference on Telecommunications and Signal Processing (TSP).

[7] Rybalkin, V., Pappalardo, A., Ghaffar, M. M., et al. (2018). FINN-L: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs. arXiv:1807.04093.

[8] L. A. Dias, J. C. Ferreira and M. A. C. Fernandes, "Parallel Implementation of K-Means Algorithm on FPGA," in IEEE Access, vol. 8, pp. 41071-41084, 2020, doi: 10.1109/ACCESS.2020.2976900.

[9] L. Andrade Maciel, M. Alcântara Souza and H. Cota de Freitas, "Reconfigurable FPGA-Based K-Means/K-Modes Architecture for Network Intrusion Detection," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 8, pp. 1459-1463, Aug. 2020, doi: 10.1109/TCSII.2019.2939826.

[10] J. Canilho, M. Véstias and H. Neto, "Multi-core for K-means clustering on FPGA," 2016 26th International Conference on Field Programmable Logic and Applications (FPL), 2016, pp. 1-4, doi: 10.1109/FPL.2016.7577313

[11] K. M. A. Patel and P. Thakral, "The best clustering algorithms in data mining", *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, pp. 2042-2046, Apr. 2016.

[12]Dhanachandra, Nameirakpam; Manglem, Khumanthem; JinaChanu, Yambem. Image Segmentation Using K -means Clustering Algorithm and Subtractive Clustering Algorithm. In: Procedia Computer Science. Vol. 54, 2015, Pg. 764-771.

[13] MAGALHÃES, MARÍLIA. Relação entre proficiência do Enem e situação socioeconômica dos candidatos ao ensino superior por meio do algoritmo k-means:Orientadora: Luiza Helena Felix. 2020. 14f. TCC(Graduação) - Curso de Bacharelado em Ciência e Tecnologia, DCEN, Universidade Federal Rural do Semi-Árido - UFERSA, Mossoró. 2020.

[14] B. Bahmani, B. Moseley, A. Vattani, R. Kumar and S. Vassilvitskii, "Scalable K-means", *Proc. VLDB Endowment*, vol. 5, no. 7, pp. 622-633, 2012.

[15] JAIN,A. K. Data clustering: 50 years beyond K-means, Pattern Recognit. Lett. Michigan, USA, v. 31, n. 8, p. 651–666, 2010. Disponível em: <Data clustering: 50 years beyond K-means - ScienceDirect.

[16] LLOYD, S. P. Least squares quantization in PCM.: Special issue on quantization, IEEE Trans. Inf.Theory, vol. 28, no 2, p. 129–137, 1982. Disponível em: <Least squares quantization in PCM - IEEE Journals & Magazine.

[17]MATTE, M. K. Impacto do uso da desigualdade triangular para acelerar o algoritmo k-means, Centro Universitário Campo Limpo Paulista, São Paulo. Unifaccamp, 2020.

[18] Mrs. M. D. Malkauthekar and Mrs. S. D. Sapkal, “Experimental Analysis of Facial Images using Fisher Discriminant and PCA” IEEE sponsored IACC 2008, Patiala.

[19] Mrs. M. D. Malkauthekar, Mrs. S. D. Sapkal and Mr. V.P. Kshirsagar, “Face Recognition Using Nearest

Neighbor Method", ICSCI-2008, Hyderabad.
[20] Spencer, Neil H. (2013), "5.4.5 Squared Euclidean Distances", *Essentials of Multivariate Data Analysis*, CRC Press, p. 95, ISBN 978-1-4665-8479-2
[21] Ultra96 "Ultra96-PYNQ's documentation[online]". Available: https://ultra96-pynq.readthedocs.io/en/latest/index.html.
[22] Doulos, "Doulos web page"[Online]. Available: https://doulos.com/.
[23] EDA Playground , "EDA Playground web page "[Online]. Available: https://www.edaplayground.com/.
[24] Rich, D.I.. (2003). The evolution of SystemVerilog. Design & Test of Computers, IEEE. 20. 82- 84. 10.1109/MDT.2003.1214355.