Implementação do k-nearest neighbors – KNN em VHDL

Alexandre G. da Costa[1], **Diego P. Jaccottet**[1], **Leandro W. Dias**[1]

[1]CDTec – Universidade Federal de Pelotas (UFPEL)
CEP 96.010-610 – Pelotas – RS – Brazil

alexandre.costa@inf.ufpel.edu.br, {diego.porto.j,lwdias}@gmail.com

Resumo. *Neste trabalho foi implementado o algoritmo KNN na placa Altera Cyclone (2C35) FPGA de 35.000 LEs, afim de reconhecer indivíduos. Sendo que a entrada é o esqueleto do indivíduo, esse esqueleto é inferido a partir de filmagem tridimensional realizada pelo Kinect Versão 2 no seu SDK. Foi obtido sucesso em capturar frames, enviar pela porta serial até a placa e mostrar em qual classe, de uma serie de classes pré definidas, a posição atual da pessoa se encontra.*

1. Introdução

FPGAS são muito usadas para protótipos, onde não se justifica um projeto ASIC personalizado. Entre as vantagens das FPGAS estão o alto paralelismo e adaptabilidade, com paralelismo para dados, tarefas, pipeline ou uma mistura desses. FPGAS também são muito eficientes na questão da quantidade de bits necessários para uma tarefa. Em FPGAS a lógica de instruções e de-codificação de endereços se torna desnecessária, isso possibilita alcançar throughput muito maior que em processadores tradicionais, assim como uma redução da quantidade de energia usada. FPGAS vem se tornando cada vez mais poderosas, o que torna seu uso cada vez mais interessante [Najjar et al. 2003].

O uso de gestos é um dos meios de Interação Humano-Computador (Human-Computer- Interaction - HCI) mais naturais e intuitivos. Além disso existem aplicações em realidade virtual, linguagem de sinais, e jogos de computador. Criar sistemas para o reconhecimento de gestos ainda é um problema desafiador. Métodos usando imagens de profundidade vem se tornando cada vez mais populares, pois é mais robusto ao fundos de imagem confusos [Ren et al. 2011]. O uso do Kinect pode simplificar muitas das etapas no processamento de vídeos, já que ele captura e pré-processa as imagens [Andersson 2014].

2. Trabalhos relacionados

Choi alcançou 89% de acurácia em 2014 para o reconhecimento de gestos humanos usando redes neurais [Choi et al. 2014]. Zhang alcançou acurácia de 99.14% usando SVMs [Zhang et al. 2014]. Saha alcançou 90.83% usando ensemble trees [Saha et al. 2014]. Os trabalhos usaram uma média de 6 poses. O diferencial do nosso trabalho está no uso da FPGA.

3. Kinect Versão 2

O Kinect é um sensor equipado de uma câmera RGB, um sensor de profundidade composto de um emissor de luz infravermelho e uma câmera sensível à profundidade.

O principio básico por trás do sensor de profundidade do Kinect é a emissão de um padrão de infravermelho e a captura simultânea da imagem desse infravermelho com uma câmera tradicional equipada com um filtro, que permite capturar o infravermelho e bloquear outras formas de onda. O processador de imagens do Kinect usa as posições relativas dos pontos no padrão do infravermelho para calcular o deslocamento da profundidade em cada posição de pixel na imagem.

Cada pixel do mapa de profundidade representa a distância cartesiana, em milímetros, do plano da câmera até o objeto mais próximo, naquela coordenada (x, y) em particular. Se o valor do pixel é 0, isso indica que o sensor não encontrou nenhum objeto no seu espaço de alcance naquela localização (x, y). Essa projeção é mencionado como espaço de profundidade. Os valores correntes de profundidade são as distâncias do plano da câmera, ao invés das do sensor propriamente dito.

4. Algoritmo KNN

O algoritmo de aprendizado supervisionado KNN é utilizado para classificar um objeto não rotulado, baseado no rótulo de seus vizinhos mais próximos em um espaço de exemplos [Andersson 2014]. Essa proximidade é, frequentemente, baseada em uma métrica de distância entre dois pontos, por exemplo, a distância Euclidiana. De maneira simplificada, a regra de classificação do KNN é associar a uma amostra de teste, o rótulo da maioria das categorias de seus "k" vizinhos mais próximos.

Um conjunto de treino X consiste em n pares de vetores e rótulos, dispersos em um espaço de classes. Dado um novo par (x; θ), onde apenas a medida x é observável, o valor de θ é estimado pela utilização dos dados contidos no conjunto X com os vetores e rótulos já conhecidos (supervisionado). Um vetor x' é um vizinho mais próximo de x, se segundo a Equação:

$$d(x'_n, x) = min(x_i, x) i = 1, 2, ..., n$$

A distância mínima entre um vetor vizinho e o vetor testado são iguais ao conjunto de distâncias mínimas entre os outros vetores e o vetor testado. Normalmente, o "voto" do vizinho possui um peso, de acordo com a distância do vetor testado e seus "k" vizinhos mais próximos. Assim, θ é estimado com o rótulo desses "k" vizinhos mais próximos,como mostra a Figura 1.

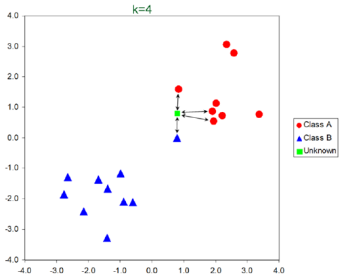


Figura 1. representação visual de 19 vetores rotulados com classe A e B e uma classe desconhecida. O vetor com classe desconhecida é portanto, rotulado com seus 4 (k=4) vizinhos mais próximos de acordo com a distância euclidiana entre eles.

5. Comunicação serial

RS-232 é uma comunicação full-duplex(ida e volta ao mesmo tempo). Os sinais são representados com voltagens em relação ao ground(chamado common). Ele possui muitas linhas de handshaking. O sinal muitas vezes está entre -12V e +12V, sendo -3V e +3V usados para absorver ruídos.

Os sinais na Cyclone para comunicação serial são `UART_TXD` para envio e `UART_RXD` para receber. O protocolo de comunicação é ilustrado na figura 2. Cada pacote possui 10 bits, 8 bits de dado e dois para start e stop. Além disso é necessário um prescaler no código que implementa esse protocolo, ele deve contar de 0 até 5208, já que 50MHz/9600 Bit/sec = 5208. 9600 é o baud rate da serial. Assim sabemos o tempo que se deve aguardar na contagem, usando o clock, para igualar o baud rate. Esse método é muito usado em FPGAs para fazer o timing com unidades externas, como memória RAM, etc.

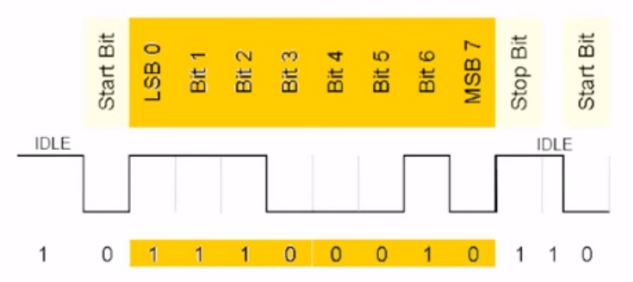


Figura 2. Esquema de transmissão serial.

6. Processo de implementação do algoritmo KNN em VHDL

O algoritmo usado na implementação VHDL foi o KNN pela distancia euclidiana. Esta é a distância entre dois pontos, que pode ser provada pela aplicação repetida do teorema de Pitágoras [Wikipedia 2015]. A distancia euclidiana entre o ponto P($p_1, p_2, p_3, ..., p_n$) e Q($q_1, q_2, q_3, ..., q_n$) é dada por:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Para facilitar a implementação em VHDL, foi considerado K igual a um.

Os pontos recebidos pela parte de aquisição de dados vem no padrão IEEE 754. O padrão 754 define regras de operação e representação sobre números binários em ponto flutuante. Para que o número esteja normalizado deve estar no formato da Figura 3.

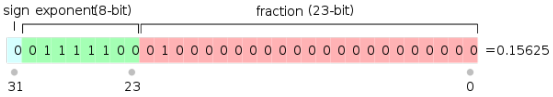


Figura 3. Distribuição dos bits. Precisão simples [Wikipedia 2015]

A precisão de representação numérica utilizada neste trabalho foi a precisão simples. Onde possui 32 bits que em representação seriam 7 digitos decimais. Destes 32 bits 1 bit é para o sinal, 8 bits para o expoente e 23 bits para representar a mantissa.

6.1. Descrição dos sinais das entradas e saídas

A figura 4 é uma visão geral da comunicação entre o Kinect e o FPGA.

Figura 4. Visão geral

- `clock`: é um pino de entrada para prover o sincronismo entre as operações aritimeticas.
- `reset`: é um pino de entrada que reinicia o processo de calculo do KNN. quando esta em 0 os registradores são reiniciados e a maquina de estados vai para o estado inicial.
- `uart_rxd`:
- `key`:
- `uart_txd`:
- `end_knn_out`: é um pino de saída que sinaliza quando terminou o calculo do KNN.
- `alb_out`:

6.2. Descrição do bloco operativo

O bloco operativo é dividito basicamente em 3 loops, onde o primeiro loop vai acumular os valores e o segundo loop serve apenas para estabilizar o calculo da raiz quadrada. Quando este processo fica pronto o valor é comparado e armazenado em um registrador. O terceiro loop serve para carregar o priximo exemplo armazenado na memória e termina quando não existem mais exemplos para serem comparados.

Para implementar este processo foram usados uma memória, um subtrator, um multiplicador, dois somadores, um acumulador, um bloco que calcula a raiz quadrada e um registrador. Na implementação das operações e da memória foram utilizadas as *megafunctions* para ponto flutuante do Quartus II.

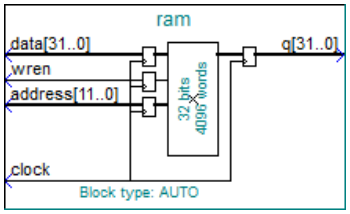


Figura 5. *Megafunction* "RAM: 1-PORT"

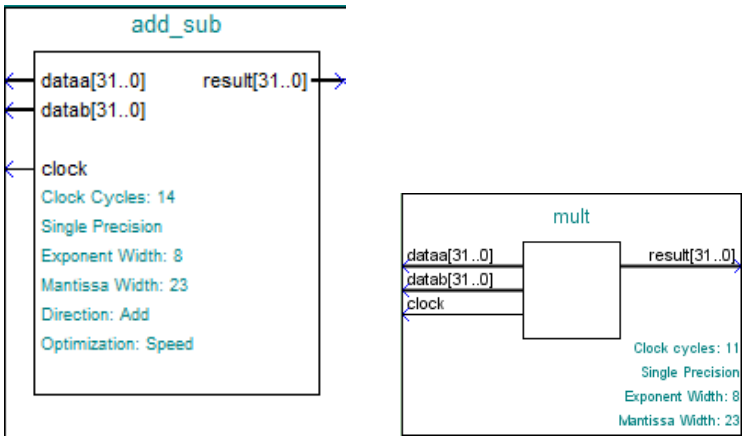
A figura 5 descreve uma memória criada pelo *MegaWizard Plug-In Manager* do Quartus II.

O modelo utilizado no trabalho tem as seguintes entradas:

- *data* (32 bits): armazena o dado;
- *wren* (1 bit): controla a escrita da memória;

- *address* (11 bits): endereço de onde vai ser armazenado o dado;
- *clock* (1 bit): relógio para sincronizar.

Este modelo de memória tem apenas uma saída de dados (q), que tem 32 bits e seu valor depende da entrada *address*. A memória é inicializada com exemplos pelo componente *altsyncram* atribuindo a entrada `init_file` um arquivo ".mif".



(a) ALTFP_ADD_SUB (b) ALTFP_MULT

Figura 6. *Megafunctions* de soma/subtração e multiplicação

A figura 6(a) descreve a *megafunction* de soma e subtração de ponto flutuante do Quartus II. Ela foi configurada com formato de precisão simples de ponto flutuante. Foi utilizado uma latência de 14 ciclos de relógio.

Para fazer a exponenciação foi utilizado um multiplicador de ponto flutuante a partir da *megafunction* `ALTFP_MULT`. A figura 6(b) descreve a *megafunction* de multiplicação de ponto flutuante do Quartus II. Ela foi configurada com formato de precisão simples de ponto flutuante. Foi utilizado uma latência de 11 ciclos de relógio.

As *megafunctions* da figura 6.2 foram utilizadas na implementação e tem as seguintes entradas:

- *dataa* e *datab* (32 bits): dados a serem operados;
- *clock* (1 bit): relógio para sincronizar.

Esta *megafunction* tem apenas uma saída de resultado (*result*) que tem 32 bits.

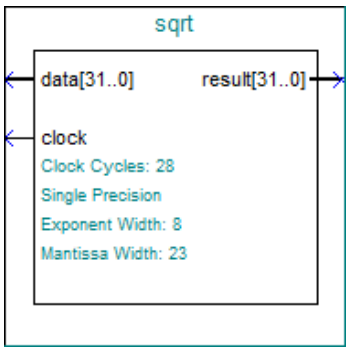


Figura 7. *Megafunction* ALTFP_SQRT

A figura 7 descreve a *megafunction* de raiz quadrada de ponto flutuante do Quartus II. Ela foi configurada com formato de precisão simples de ponto flutuante. Foi utilizado uma latência de 28 ciclos de relógio.

A *megafunction* `ALTFP_SQRT` utilizada no trabalho tem as seguintes entradas:

- *data* (32 bits): dado a serem operados;
- *clock* (1 bit): relógio para sincronizar.

Esta *megafunction* tem apenas uma saída de resultado (*result*) que tem 32 bits.

A figura 8 descreve a *megafunction* de comparação de ponto flutuante do Quartus II. Ela foi configurada com formato de precisão simples de ponto flutuante. Foi utilizado uma latência de 3 ciclos de relógio.

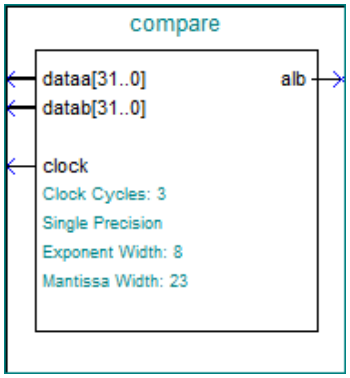


Figura 8. *Megafunction* ALTFP_COMPARE

A *megafunction* `ALTFP_COMPARE` utilizada no trabalho tem as seguintes entradas:

- *dataa* e *datab* (32 bits): dados a serem operados;
- *clock* (1 bit): relógio para sincronizar.

Esta *megafunction* tem apenas uma saída que retorna um se a entrada *dataa* é menor do que a *datab* caso contrario retorna zero.

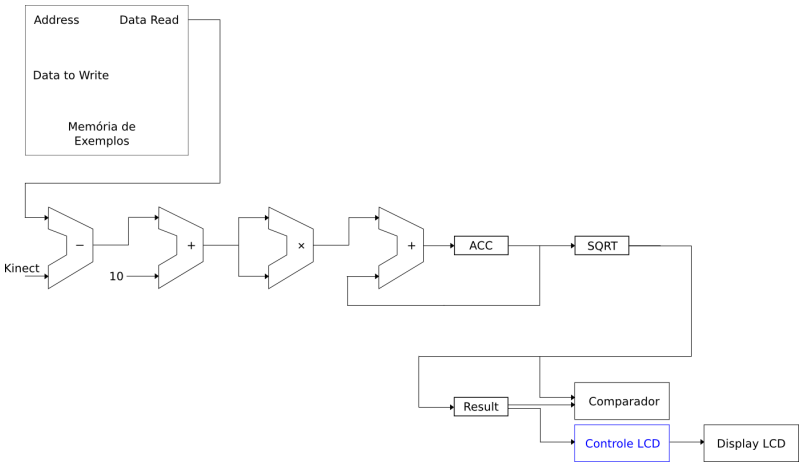


Figura 9. Bloco operativo

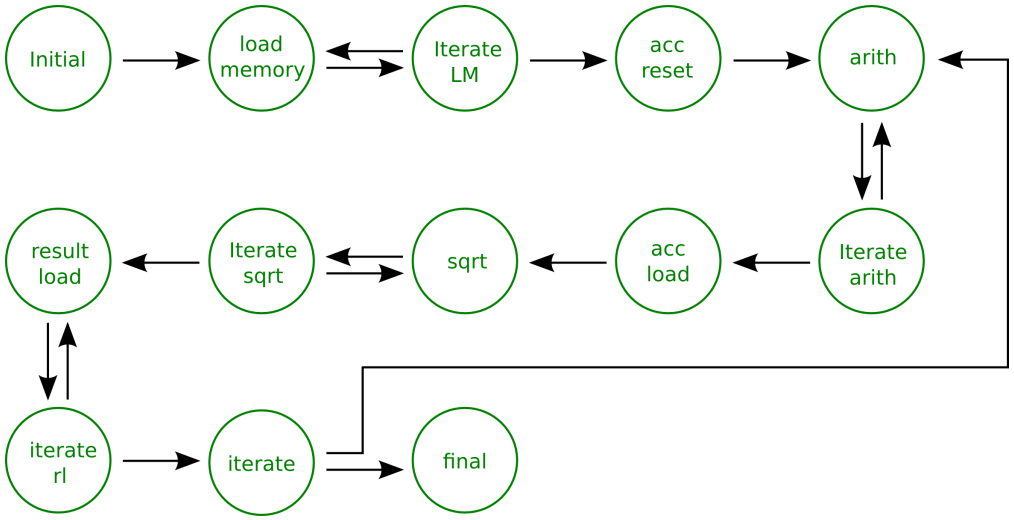
A figura 9 apresenta o bloco operativo. O caminho de dados inicia quando o bloco de aquisição de dados envia um sinal de que o *array* com o frame do Kinect já esta preenchido.

6.3. Descrição do bloco de controle

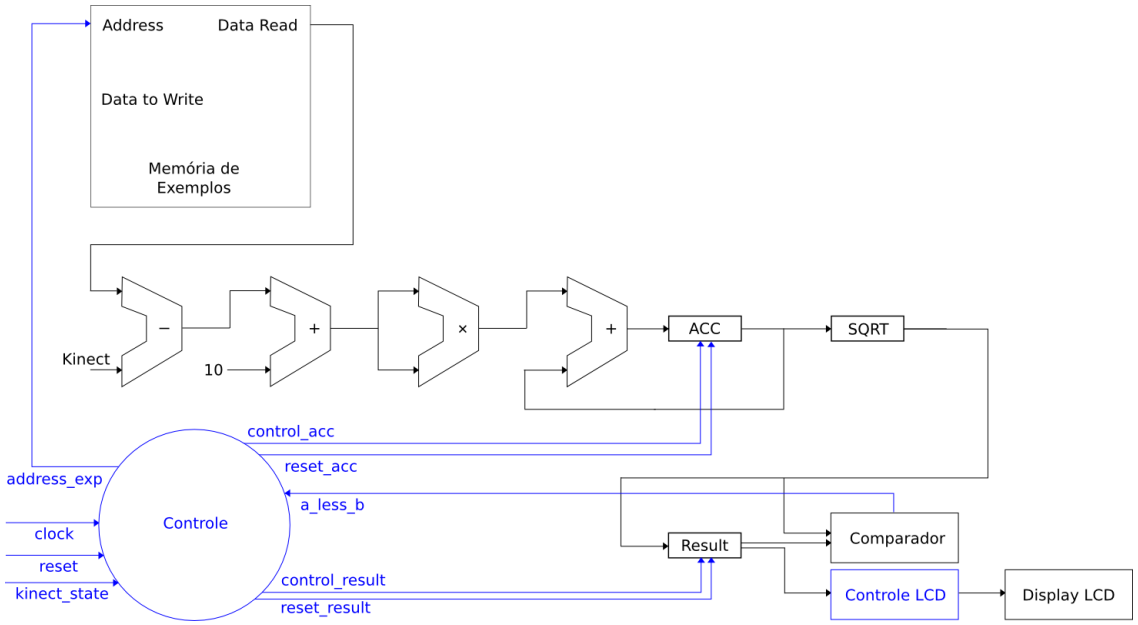
O bloco de controle é composto por 12 estados chamados de `initial`, `load_memory`, `iterate_lm`, `acc_reset`, `arith`, `iterate_arith`, `acc_load`, `sqrt`, `iterate_sqrt`, `result_load`, `iterate` e `final`.

Cada estado tem a seguinte funcionalidade:

- `initial`: responsável por resetar as variáveis e também por esperar os dados vindos do Kinect;
- `load_memory` e `iterate_lm`: Aguardam a inserção dos exemplos na memória;
- `acc_reset`: Reseta o registrador do acumulador;
- `arith` e `iterate_arith`: Estado da realização das operações matemáticas das distâncias euclidianas;
- `acc_load`: Armazena a soma temporária da distância euclideana no registrador;
- `sqrt` e `iterate_sqrt`: Realiza o cálculo da raiz quadrada da distância euclideana;
- `result_load`: Utiliza o componente comparador e carrega o resultado da distância euclideâna no registrador de resultado;
- `iterate`: Gerencia se o próximo estado será retornar para o ”arith”e continuar calculando ou se já terminou os cálculos; e deve-se ir para o estado ”final”;
- `final`: Mostra o resultado na placa e informa a interface com o KNN que a próxima entrada já pode ser enviada.



Abaixo está apresentado o bloco operativo juntamente com o bloco de controle:



7. Conclusão

Este trabalho desenvolveu um algoritmo em VHDL que...

Referências

Andersson, V. O. (2014). Identificação Biométrica com Antropometria e Caminhar Humano Utilizando o kinect. Disertação de mestrado, Universidade Federal de Pelotas, Pelotas.

Choi, J.-H., Ko, D.-H., Kim, H., and Lee, S.-G. (2014). Design of body gesture recognition system for regularity and repeatability gestures. In *Control, Automation and Systems (ICCAS), 2014 14th International Conference on*, pages 449–453. IEEE.

Najjar, W., Böhm, W., Draper, B., Hammes, J., Rinker, R., Beveridge, J. R., Chawathe, M., Ross, C., et al. (2003). High-level language abstraction for reconfigurable computing. *Computer*, 36(8):63–69.

Ren, Z., Meng, J., Yuan, J., and Zhang, Z. (2011). Robust hand gesture recognition with kinect sensor. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 759–760. ACM.

Saha, S., Datta, S., Konar, A., and Janarthanan, R. (2014). A study on emotion recognition from body gestures using kinect sensor. In *Communications and Signal Processing (ICCSP), 2014 International Conference on*, pages 056–060. IEEE.

Wikipedia (2015). Ieee 754-1985 — wikipedia, the free encyclopedia. [Online; accessed 30-June-2015].

Zhang, Z., Liu, Y., Li, A., and Wang, M. (2014). A novel method for user-defined human posture recognition using kinect. In *Image and Signal Processing (CISP), 2014 7th International Congress on*, pages 736–740. IEEE.