**GABRIEL CASARIN DA SILVA**
**GABRIEL PIAZZALUNGA**
**LAIS HARUMI FUKUJIMA AGUIAR**

# RECONSTRUÇÃO VOLUMÉTRICA USANDO REDES NEURAIS ACELERADAS POR HARDWARE

São Paulo
2018

**GABRIEL CASARIN DA SILVA**
**GABRIEL PIAZZALUNGA**
**LAIS HARUMI FUKUJIMA AGUIAR**

# RECONSTRUÇÃO VOLUMÉTRICA USANDO REDES NEURAIS ACELERADAS POR HARDWARE

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Eletricista.

São Paulo
2018

**GABRIEL CASARIN DA SILVA**
**GABRIEL PIAZZALUNGA**
**LAIS HARUMI FUKUJIMA AGUIAR**

# RECONSTRUÇÃO VOLUMÉTRICA USANDO REDES NEURAIS ACELERADAS POR HARDWARE

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Eletricista.

Orientador:
Bruno de Carvalho Albertini

Co-orientador:
Luckas André Farias

São Paulo
2018

Prof. X

Prof. Y

Prof. Z

Dedicatória

Blabla

# AGRADECIMENTOS

Obrigado pelos peixes.[1]

*“Epígrafe”*

-- Autor

# RESUMO

Durante a história da medicina o estudo da anatomia foi essencial para o desenvolvimento de novas técnicas. Atualmente existe documentação suficiente para o estudo de estruturas saudáveis genéricas, mas não existe um modo de estudar um indivíduo específico de forma não invasiva. Para isso torna-se interessante a possibilidade do estudo das estruturas em modelos 3D obtidos a partir de exames de imagens já existentes. Na observação desses modelos pode-se identificar anomalias e também especificidades da anatomia de cada paciente, tornando mais rápidas cirurgias, que passam a ser planejadas para maior precisão. Para possibilitar o estudo prévio da anatomia de cada paciente específico é proposta a implementação de uma rede neural para reconstrução 3D a partir de imagens de ressonância magnética. Essa rede, implementada em hardware, pode acelerar a reconstrução e aumentar a qualidade das opções disponíveis.

**Palavras-Chave** – Reconstrução 3D, Hardware, CNN.

# ABSTRACT

In the history of medicine the study of anatomy was essential for the development of new techniques. There is currently enough documentation for the study of generic healthy structures, but there is no way of studying specific individuals noninvasively. In order to do this, it is interesting to study the structures in 3D models obtained from images of existing imageing techniques. By observing these models, it is possible to identify anomalies and particularities in the anatomy of each patient, making surgeries shorter, which may now be planned for greater precision. In order to make possible the previous study of the anatomy of each specific patient, it is proposed to implement a neural network for 3D reconstruction from magnetic resonance imaging. This network, implemented in hardware, can accelerate the reconstruction and increase the quality of the available options.

**Keywords** – 3D Reconstruction, Hardware, CNN.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# SUMÁRIO

# 1 INTRODUÇÃO

## 1.1 Introdução

A reconstrução 3D de objetos tem sido explorada nas áreas de medicina e biologia, já que permitem um novo modo de abordar e analisar imagens, que também nunca foram tão precisas e detalhadas por conta da atual tecnologia de obtenção dessas imagens, através de uma representação geométrica mais precisa e intuitivamente observável das estruturas. Desde avaliação de estruturas e lesões, até observação de detalhes em algum animal ou objeto sendo estudado têm sua qualidade aumentada por conta de novos modos de se obter imagens, da maior qualidade que os equipamentos proporcionam e das novas possibilidades de processamento dessas imagens. Diversas áreas se beneficiam desse tipo de informação além da medicina e biologia, como nas áreas de robótica, arquitetura, realidade aumentada e cinema.

Os modos mais comuns de se realizar uma reconstrução 3D variam de simplesmente ordenar imagens de cortes do objeto em pontos diferentes até análise de somente uma imagem 2D e projeção para um provável modelo em três dimensões. Os modelos mais usados atualmente são implementados em software e levam métodos estatísticos de correção de deformações e ruídos. Os mais recorrentes são a reconstrução a partir de imagens de cortes paralelos do objeto, que podem ser obtidas de diversas formas e reconstrução a partir de imagens estéreo, onde são obtidas imagens do mesmo objeto a partir de dois pontos diferentes.

Esse tipo de imagem reconstruída consegue resumir as informações de muitas imagens em menos imagens que contém mais informação, como posições relativas visíveis de modo mais intuitivo. Além disso pode-se melhorar o método de exposição de uma ideia a partir de melhores modelos. Com a melhoria na qualidade das imagens o volume de informações a ser analisado e processado também aumentou muito, o que torna os métodos utilizados tradicionalmente mais lentos, já que lidamos com volume crescente de informações.

### 1.1.1 Motivação

Com o avanço das técnicas de cirurgia passou a ser necessário melhorar a precisão das ferramentas de vizualização das estruturas que serão operadas. Com isso começaram a surgir novos exames e logo surgiram também ferramentas para processar as imagens obtidas nesses exames. Há poucos anos foi realizada a primeira cirurgia estudada previamente em imagens 3D reconstruídas a partir de imagens de Ressonância magnética do coração, isso foi possível com a utilização de uma ferramenta chamada Osirix [3], que é uma das únicas ferramentas em que é possível vizualisar as imagens geradas nesses exames e também existe a opção de condensar as imagens todas num modelo 3D.

Muitas vezes é necessário medir distâncias e diâmetros, como em exames preparatórios de cirurgias vasculares, o que é mais facilmente realizado nas imagens dos cortes, que nem sempre são realizadas nos eixos ideais para realização dessas medidas. Desse modo a reconstrução 3D permite que sejam feitas medidas de distâncias em ângulos diferentes dos tradicionais ângulos de corte utilizados nos exames. Com a reconstrução 3D passa a ser possível realizar novos cortes nas direções necessárias sem a realização de novos exames, isso acelera o processo de preparação para procedimentos e também aumenta a eficiência de estudos onde somente eram vistos os cortes tradicionais.

Com a melhoria da qualidade dos exames por imagem passou a ser possível explorar novos modos de se observar as informações que estavam disponíveis, e com isso começou um novo modo de se enxergar as imagens. Com essas novas possibilidades de vizualisação tornou-se possível o desenvolvimento de novas técnicas de operações e também a descoberta de novas estruturas. Essas descobertas e novos procedimentos podem ser aumentados na mesma medida em que são melhoradas as imagens disponíveis. Além disso, como implementações em software dos algoritmos de reconstrução podem ser custosos em termos de tempo de processamento, propõe-se uma implementação em hardware para que o custo de tempo das implementações existentes sejam reduzidos.

## 1.2 Cronograma do Projeto

### 1.2.1 Cronograma do 1º Semestre

Planejamento previsto para os trabalhos realizados no primeiro semestre do ano.

| Quando | O que |
|---|---|
| Fevereiro | Definição de escopo e objetivos do projeto, pesquisa bibliográfica |
| Março | Escolha de algoritmos, métodos de comparação, organização/divisão de tarefas e testes. Obtenção de imagens para uso no treinamento. |
| Abril | Definição de modelo escolhido para seguir para a fase de implementação em Hardware. Implementação simples de rede neural em Hardware (FPGA) |
| Maio | Implementação total de algoritmo escolhido em Hardware. Testes e comparações com resultados anteriores. |
| Junho | Adaptações na implementação em hardware. Análise com diversos datasets e subsets. Elaboração final de relatório |

Tabela 1: Cronograma do 1º Semestre

### 1.2.2 Cronograma 2º Semestre

Para o segundo semestre será seguido o seguinte fluxograma de desenvolvimento, onde serão feitas iterações de melhoramento do modelo proposto pelo grupo.

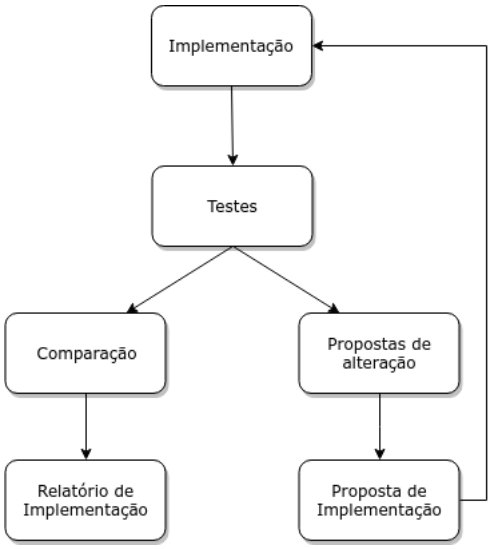


Figura 1: Fluxograma desenvolvimento 2º Semestre

| Quando | O que |
|---|---|
| Agosto | Implementação de primeira modificação proposta |
| Setembro | Testes, comparação, análise e relatório de mudanças. Paralelamente novas propostas de modificações |
| Outubro | Segunda iteração - Implementação de modificação proposta, testes, comparação, análise e relatório de mudanças. |
| Novembro | Redação de monografia |

Tabela 2: Cronograma 2º Semestre

## 1.3 Declaração da Necessidade

### 1.3.1 Identificação das Necessidades

A dificuldade de determinar previamente a anatomia do paciente indivudalment torna necessário o uso de um recurso de reconstrução 3D, para que haja estudo das estruturas de cada um antes da realização de algum procedimento invasivo. Com uma ferramenta dessas torna-se possível obter visão de novos cortes da anatomia e melhor planejamento de cirurgias e procedimentos sem que seja necessária uma operação exploratória, por exemplo. Assim pode-se densenvolver com maior precisão e menor tempo procedimentos auxiliados por máquinas controladas pelo médico, tornando alguns deles menos invasivos e mais eficientes. Havendo maior precisão e agilidade no planejamento de uma cirurgia é possivel diminuir a possibilidade de erro e diminuir o tempo de ralização dos procedimentos.

#### 1.3.1.1 Árvore de Requisitos

A árvore de requisitos organiza as necessidades identificadas e conceitos que podem solucioná-las.

#### 1.3.1.2 Ranqueamento de Necessidades

| | Rápido | Nova Abordagem | Score |
|---|---|---|---|
| Rápido | 1 | 1/3 | 0.1 |
| Nova Abordagem | 3 | 1 | 0.9 |

Tabela 3: Ranqueamento de Necessidades

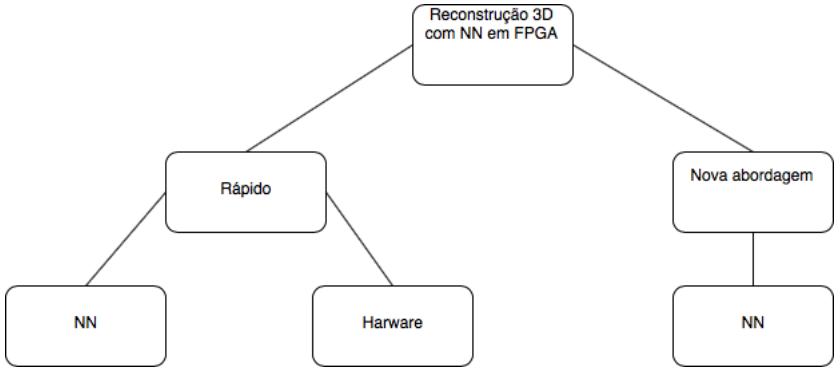


Figura 2: Árvore de Requisitos

## 1.4 Trabalhos Relacionados

### 1.4.1 Literatura

#### 1.4.1.1 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction

O artigo o artigo propõe um extensão da rede neural LSTM( Long Short-Term Memory) que é chamada de (3D-R2N2) 3D Recurrent Reconstruction Neural Network. A 3D-R2N2 é capaz de unificar reconstrução a partir de mono e multi perspectivas em uma única estrutura. Além disso, não requer segmentação, labels, calibração de câmera, palavra chaves. A estrutura da rede é subdivida em "Enconder", "3D Convolutional LSTM" e "Decoder", é apresentado o método pelo qual cada elemento é contruído. Algumas limitações são apontadas: objetos com textura alta foram representados de forma pior do que objetos com pouca textura. E o método MVS (multi view stereo) possui melhor resposta para um conjunto com mais de 30 imagens do modelo a ser representado.[2]

#### 1.4.1.2 Experimental Comparison of Three Reconstruction Algorithms

O estudo compara algoritmos analíticos de recontrução a partir de imagens de projeção cônica utilizando imagens adquiridas previamente e testando os algoritmos sob as mesmas condições comparando os resultados em termos de qualidade da imagem, ruído e tempo de processamento. Os algoritmos comparados foram Feldkamp, Grangeat e um filtro não-estacionário (NSF).

Os três algoritmos produziram imagens com qualidade muito parecida, no entanto o algoritmo Grangeat resultou em ruídos reduzidos na ordem de (1:15) em relação aos outros 2 algoritmos. Já entre Feldkamp e NSF o algoritmo NSF melhora a qualidade da imagem nas regiões próximas às bordas, mas ao grande custo de 4 vezes mais demanda

computacional. No fator a distorção o algoritmos Grangeat apresentou diversas distorções de formatos, o que não acontece de maneira substancial nos outros algoritmos.

Portanto, foi concluído que o melhor algoritmo, para manter uma boa qualidade da imagem e eficiência computacional, foi determinado o Feldkamp, já que em relação a imagem o algoritmo NSF é pouco melhor e a um custo computacional 4 vezes maior. Já o algoritmo Grangeat as distorções que foram ocorridas afetam diretamente na imagem e, portanto, não poderia ser a melhor opção apesar de reduzir ruídos.[4]

### 1.4.2 Softwares existentes

#### 1.4.2.1 Canoma (1999)

Canoma é um software lançado em 1999 pela Metacreations, que é usado para a criação de imagens 3D realistas a partir de uma ou mais fotografias, os modelos são geometricamente bem simples, para rápida prototipação. Canoma é um software para criação de objetos 3D de forma simples, onde é possível adicionar textura.[5]

#### 1.4.2.2 Paraview (2002)

Em 2000 foi iniciado o desenvolvimento do Paraview, como resultado da colaboração entre Kitware Inc. e Los Alamos National Laboratory, que foi lançado no fim de 2002. Quase paralelamente estava sendo desenvolvido um sistema de vizualisação de dados pela Kitware, que se tornou a ParaView Enterprise Edition. Em 2007 foi lançado o Paraview 3.0, que é a atual versão da ferramenta.

Paraview é uma ferramenta open source de análise de dados e vizualisação, que permite o uso de diferentes técnicas para que os dados possam ser explorados interativament, seja com vizualisações 3D ou de modo analítico. O Paraview pode ser usado tanto como aplicação como na forma de Framework, o que faz com que seja muito versátil. Os criadores do Paraview tinham como objetivo desenvolver uma ferramenta multi-plataforma, open source, que pudesse ser modificada de acordo com as necessidades e fosse intuitiva de usar. [6]

#### 1.4.2.3 OsiriX (2004)

OsiriX é um software de vizualisação de imagens médicas, que foi criado pelo radiologista Antoine Rosset MD da Suíça, que recebeu investimentos para passar um ano em

Los Angeles na UCLA estudando imagens digitais. Com esse estudo ele percebeu que seria possível realizar muito mais que o objetivo inicial de implementar um software que converte imagens DICOM para um vídeo, que ajduaria os radiologistas a criar uma base de dados educativa. Foi lançado em 2004 somente como ferramenta de organização dos dados e de vizualisação das imagens obtidas nos exames. Em 2006 o matemático e cientista da computação Joris Heuberger se juntou ao projeto e em 2005 recebeu no Apple Design Awards os prêmios Best Use of Open Source e Best Mac OS X Scientific Computing Solution. Em 2009 foi criada a OsiriX Foundation que apoia o desenvolvimento de softwares para a medicina. Em 2010 foi criada a empresa Pixmeo, que é responsável pela versão aprovada pela Anvisa do software OsiriX MD. O primeiro cirurgião que passou a utilizar o OsiriX como ferramenta para planejamento cirúrgico e publicou seu conhecimento foi Sugimoto, cirurgião do aparelho digestivo que realizava reconstruções 3D no seu computador pessoal para depois operar via laparoscópica seus pacientes com maior facilidade, sem surpresas.z[3]

#### 1.4.2.4 ImageJ (1987)

ImageJ é um software de domínio público, implementado em Java pelo National Institues of Health para processamento de imagens médicas. Esse software foi desenvolvido com uma arquitetura aberta para que fosse possível desenvolver plugins para usos específicos de cada usuário. Também podem ser personalizadas para cada necessidade a aquisição, a análise e o processamento das imagens a partir de um editor Java embutido no software.

### 1.4.3 Algortimos Existentes

Nessa seção são estudados algoritmos existentes e utilizados para reconstrução 3D de imagens, entre eles foram encontrados algoritmos analíticos e também modelos de Machine Learning. Ao final está apresentada uma tabela onde as características de cada algoritmo são resumidas para melhor vizualisação e comparação entre eles.

#### 1.4.3.1 Algoritmo de Feldkamp

O algoritmo de Feldkamp é um algoritmo analítico de reconstrução, a partir de imagem de projeção cônica. Ele é o mais eficiente e serve de benchmark para avaliação do desempenho de novos algoritmos.[4]

Esse algoritmo é uma extrapolação do algoritmo 2D fan-beam para 3 dimensões, usando algumas adaptações. Ele produz imagens com deformações, que são melhoradas do algoritmo de Grangeat. Ele usa a transformada de Radon para duas dimensões, que gera uma fórmula para a determinação do algoritmo da reconstrução 3D. Ele usa projeções cônicas de um objeto obtidas de diveros ângulos e aproxima um modelo 3D a partir do somatório das densidades obtidas de cada projeção. A implementação desse algoritmo é feita usando-se matrizes e multiplicações delas, as projeções obtidas são interpolações de projeções obtidas por diferentes sensores.[7]

#### 1.4.3.2 Algoritmo de Grangeat

O algoritmo de Grangeat é uma derivação do algoritmo de Feldkamp, que utiliza uma relação das imagens obtidas através de um feixe cônico e reconstrói um modelo 3D a partir de imagens de feixe cônico. Esse algoritmo consiste de duas etapas, primeiro é aplicada a transformada de Radon, assim como no algoritmo de Feldkamp, e depois é aplicada a transformada inversa para se obter uma imagem do objeto. Nessa segunda etapa é realizada a transformada rápida de Radon. [8]

#### 1.4.3.3 Deep convolutional encoder-decoder network

Esse modelo é a combinação de três redes neurais, que têm três funções distintas. A primeira é o Encoder, onde As imagens são traduzidas para um formato adequado para que a segunda rede que é a rede de convolução, tendo em seguida a terceira rede, que é a rede de desconvolução. Esse modelo será mais detalhado no Capítulo Treinamento. descricao[2]

#### 1.4.3.4 3D-GAN

Pode-se dividir os tipos de algoritmos de *Machine Learning* em dois grandes tipos: os (a) Classificatórios; e (b) os Generativos. Dentros dos de tipo (a) estão os algoritmos supervisionados e os não supervisionados. Dentre os de tipo (b), estão, por exemplo, as *Generative Adversarial Networks* (GANs), que gera modelos 3D a partir de um espaço probabilístico. [9].

Formalmente, seja um espaço amostral $X$ e um conjunto de *labels* $Y$. Em geral, a distribuição de probabilidade conjunta $p(x, y)$ é desconhecida. Modelos classificatórios aprendem a associar a probabilidade $p(x|y)$ de um dado $x \in X$ a uma classe $y \in Y$.

Um modelo generativo aprende a modelar cada uma das distribuições de probabilidade $p(x|y_i)$ das classes individuais de $X$. Ou seja, modelos generativos são aqueles capazes de gerar amostras que mimetizam os elementos pertencentes a um dado conjunto de dados. O método proposto em [10] utiliza a técnica de treinamento introduzida em [9] para gerar diversas imagens volumétricas em CAD obtidas do *dataset* ShapeNet [11]. Uma GAN consiste em duas redes, uma geradora e uma discriminadora, onde a discriminadora classifica objetos reais e objetos sintetizados pela geradora, e a geradora tenta enganar a discriminadora. A função *loss* é a de entropia binária cruzada:

$$L_{3D-GAN} = logD(x) + log(1 - D(G(z))$$

onde $D(x)$ é o valor da confiança com que o discriminador diz se o objeto $x$ é real ou sintético.[12]

### 1.4.4 Marching Cubes

Nesse algoritmo são gerados modelos a partir de dados médicos. A abordagem tem uma característica de "dividir e conquistar", que gera preenchimento entre as camadas das imagens, ela conecta duas camadas vizinhas de um exame através de triângulos, calculados a partir de interpolação linear. O preenchimento entre camadas é realizado a partir do cálculo do gradiente de cada imagem normalizado, esse gradiente então é usado como base para criar as sombras do modelo gerado.[13]

## 1.5 Resultados Esperados

O principal resultado esperado é um modelo de Rede Neural Convolucional acelerada por hardware que consiga construir, a partir de imagens de ressonância magnética, objetos volumétricos. Com essa ferramenta implementada se torna mais rápida e eficiente a reconstrução volumétrica de imagens, que podem ser usadas para dispositivos de realidade virtual de treinamento, que são altamente escaláveis e de importância para o treinamento de pessoas com diminuição de riscos associados a erros. Além disso esses modelos são úteis para planejamento de procedimentos menos invasivos, diminuindo o tempo exigido para cada procedimento, e assim diminuindo a quantidade de erros.

| **Algoritmo** | **Abordagem** | **Implementação Disponível** | **Tipo de Imagem Usada** | **Data** |
|---|---|---|---|---|
| Algoritmo de Feldkamp | Estatístico | CUDA e OpenCL[14] CBEA[] | Tomografia Computadorizada | 1984 |
| Algoritmo de Grangeat | Estatístico | | Tomografia Computadorizada | 1987[] |
| Algoritmo de filtro não estacionário | Estatística | | Tomografia Computadorizada | |
| Deep convolutional encoder-decoder network | Machine Learning | Caffe (source code?) | Modelos 3D da base ShapeNet | 2016 |
| 3D-GAN | 3D Generative Adversarial Network | Torch 7 e Matlab | Imagens 2.5 D de diversos objetos | 2016 |
| Marching Cubes | Estatístico | | Tomografia Computadorizada e Ressonância Magnética | 1987 |

Tabela 4: Comparativo dos algoritmos descritos

# 2 TESTES DO OSIRIX

## 2.1 Preparação das imagens

No Software OsiriX Lite foram usadas as imagens no formato recebido, e não convertidas e processadas por conta do seu funcionamento. Ele aceita imagens DICOM, que são imagens no formato produzido pelos aparelhos de exame, isso é válido para todos os tipos de exames. Esses arquivos serão descritos brevemente adiante.

### 2.1.1 O arquivo DICOM

Os arquivos DICOM (Digital Imaging and Communications in Medicine) são arquivos de imagens e vídeos médicos. Esse formato é padronizado para que sejam armazenados num arquivo só as informações do paciente, as informações do exame e a imagem.

No arquivo existem os elementos header, formado por um preâmbulo de 128 bytes, junto com um prefixo de 4 bytes ('D', 'I', 'C', 'M'), no header estão contidas as informações do paciente e algumas informações sobre a imagem como tamanho e resolução. Além do header, o arquivo tem o conjunto de dados, que é único em cada arquivo.

### 2.1.2 Criação de um vídeo da sequência de imagens

O vídeo é criado a partir das imagens obtidas na etapa anterior de conversão das informações que resultam do exame. O vídeo é criado com a colocação das imagens de cada profundidade sendo colocadas em sequência, de modo a termos uma varredura das profundidades do objeto sendo analisado.

Quando as imagens que geram o vídeo são imagens de ressonância magnética, usamos o OsiriX Lite para gerar esse vídeo, já que esse software realiza essa função.

### 2.1.3 Obtenção das imagens para reconstrução

A obtenção das imagens que serão utilizadas na reconstrução é dada a partir da separação dos frames do vídeo obtido na etapa anterior. Isso foi realizado utilizando-se um software de edição de vídeo, no caso o ffmpeg[15].

## 2.2 Osirix Lite

A ferramenta é Open Source, que é gratuito e modificável na versão OsiriX Lite, e paga se certificada pela ANVISA e outros órgãos de outros países na versão OsiriX MD, que é autorizada para uso médico, enquanto a versão gratuita não possui essa autorização por não ser certificada.

O uso do OsiriX Lite é extremamente intuitivo, apesar de haver algumas funcionalidades que juntas proporcionam melhor resultado. Por exemplo, se a reconstrução for feita diretamente das imagens do exame, são obtidos resultados muito piores, como na Figure 3. Essa reconstrução foi obtida da aplicação direta da ferramenta nas imagens de ressonância magnética. Observa-se que a reconstrução sem seleção de regiões não tem bons resultados, já que a forma que resulta do processo é irreconhecível, como observado na Figura 3.

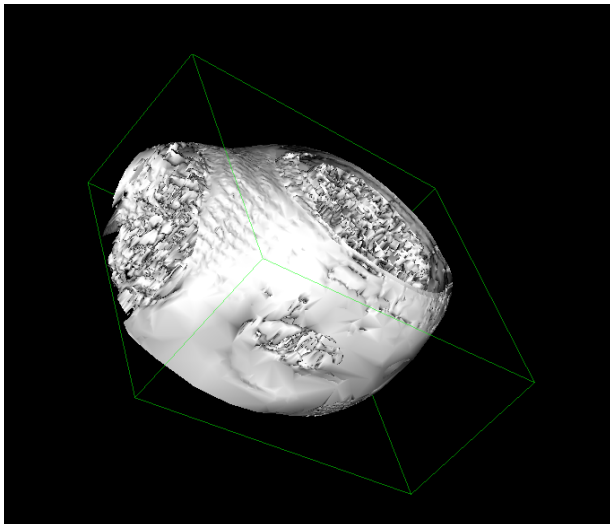

Figura 3: Reconstrução 3D de uma ressonância magnética feita no Osirix sem seleção de regiões de interesse

### 2.2.1 Preprocessamento

A reconstrução 3D feita pelo Osirix Lite se dá em duas etapas, a primeira onde são selecionadas as regiões de interesse como na Figura 4, no caso foram escolhidas as regiões dos tecidos moles, que estão representados em branco, através da escolha e seleção de uma faixa de densidades. A seleção pode ser feita automaticamente a partir de valores definidos pelo usuário, esses valores podem ser obtidos no próprio software onde é possível calcular a densidade média de uma região da imagem.

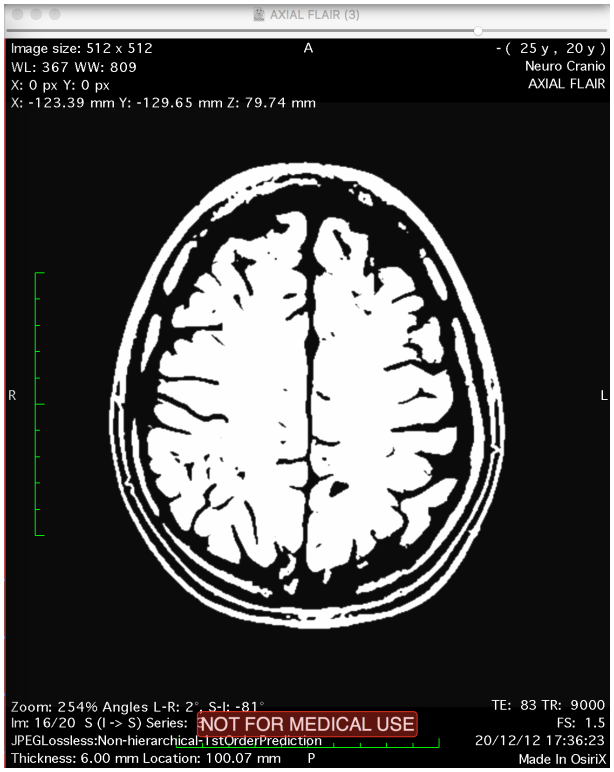


Figura 4: Seleção da região de interesse para reconstrução, tecidos moles

### 2.2.2 Reconstrução

A segunda etapa é a junção da seleção feita na primeira etapa em todas as imagens da série num modelo 3D, que é feito resultando no modelo da Figura 5. Nessa etapa é implementado o algoritmo Multiplanar Reconstructions, Curved Reconstructions.[16]

### 2.2.3 Conversão

Após a reconstrução feita pelo OsiriX, é necessário converter a imagem gerada no formato STL para nuvem de pontos em formato binário binvox por meio da ferramenta disponível em [17].

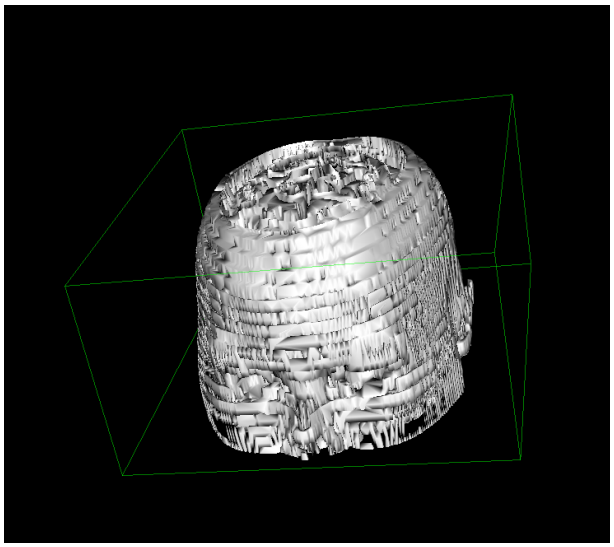

Figura 5: Reconstrução 3D de uma ressonância magnética feita no Osirix com seleção de regiões de interesse

# 3 TREINAMENTO DA REDE NEURAL

Nesse capítulo são abordadas as etapas necessárias para o treino da Rede Neural e obtenção de um modelo a ser implementado na FPGA e uma breve descrição da arquitetura escolhida.

A arquitetura escolhida é uma rede neural convolucional que utiliza blocos do tipo LSTM (Long Short Term memory), que tem como característica a persistência das informações a cada interação, e por isso é a ideal para essa tarefa. O modelo escolhido é baseado na rede 3D-R2N2[2].

## 3.1 Rede Neural Convolucional

Por não possuir unidades de reconhecimento que reaproveitem semelhanças estruturais presentes nos dados de entrada, a Rede Neural Totalmente Conectada (*Fully Connected Neural Network* ou FC), conforme aumentam a sua profundidade em número de camadas e a quantidade de neurônios em cada uma delas, possui um número de parâmetros para serem otimizados em sua fase de treinamento tão elevado que torna tal processo intratável computacionalmente. Esse fenômeno é conhecido como *curse of dimensionality*.

As Redes Neurais Convolucionais (*Convolutional Neural Networks* ou CNN) foram propostas por Yann LeCunn em [18] a fim de mitigar tal dificuldade encontrada por modelos de *Deep Learning*, trazendo o conceito de *filtro de convolução* para o contexto de Redes Neurais. As modificações arquiteturais de uma CNN são três:

(i) receptividade local

(ii) pesos compartilhados

(iii) sub-amostragem espacial

Tais modificações podem ser vistas na figura 6 (Disponível em http://colah.github.io/posts/2014-07-Conv-Nets-Modular). Cada neurônio (representados pelas caixas em verde) recebe um

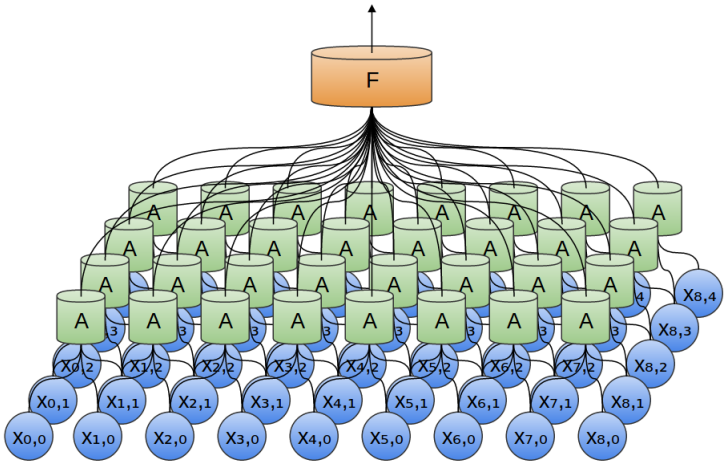


Figura 6: Camada Convolucional de uma CNN

grupo local da camada de entrada. Eles compartilham os mesmos pesos sinápticos, ou seja, todos eles implementam a mesma função $A$, que será denominada *kernel convolucional*. A repetição espacial de tais estruturas neuronais implementam em tal camada a operação de convolução discreta.

A sub-amostragem espacial é realizada após a realização da convolução e tem por objetivo reduzir a dimensionalidade da entrada da próxima camada. A técnica mais comum de sub-amostragem e a utilizada neste trabalho é denominada por *max pooling* e consiste em selecionar o output de neurônio mais representativo na localidade.

## 3.2 Rede Neural Recorrente

Redes Neurais Recorrentes têm como principal característica a persistência das informações, o que não é característica das Redes Neurais Artificiais tradicionais. Isso é obtido a partir dos laços que estão presentes na sua arquitetura[19].

O modelo simplificado da Figura 7 mostra que temos a entrada X que é avaliado pelo bloco A, que retorna o resultado Y. Mas além de aplicar a função característica, o bloco tem um laço, que armazena informação e é usado junto com as informações de entrada, o que torna possível a persistência da informação de um passo da rede neural para o próximo.

Quando se tem vários blocos desse tipo, parecido com uma sequência de redes neurais menores e com as mesmas características, obtemos algo parecido com o mostrado na Figura 8. Esse tipo de conexão entre as camadas está relacionada com listas, que acabam

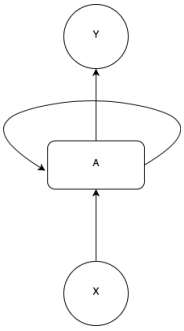


Figura 7: Bloco de uma rede neural recorrente

sendo usadas como estrutura de dados preferencial nessas redes. O uso dessas arquiteturas é muito amplo, como em reconhecimento de fala, traduções, classificação de imagens, entre outros.

Apesar dessas características das redes neurais recorrentes pode-se não obter sucesso no uso delas, especialmente em aplicações onde informações anteriores não são relevantes, como numa aplicação de controle de um motor que muda a posição de uma planta de acordo com a posição do Sol, que pode ser implementado com um simples perceptron, sem a necessidade de informações anteriores.

Para que essas características sejam possíveis são utilizadas LSTMs (Long Short Term Memory), que são um tipo específico de redes neurais que funcionam para a maior parte das tarefas com maior eficiência que as redes neurais tradicionais. Na próxima seção esse tipo de célula será explicado com maior profundidade.

Esse tipo de célula consegue utilizar informações antigas na tarefa que está sendo realizada, por exemplo, utilizar frames anteriores de um vídeo no lugar de usar somente o frame atual, o que pode melhorar o entendimento do contexto atual e melhorar a acurácia da saída.

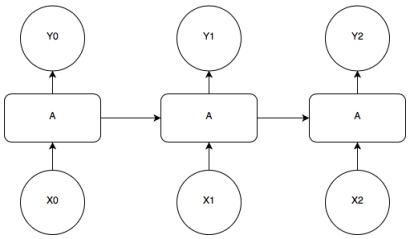


Figura 8: Sequência de Blocos de uma Rede Neural Convolucional

Apesar de todas essas características das redes neurais recorrentes, quando há uma distância muito grande entre o contexto necessário para a predição e o contexto atual elas

podem perder sua precisão ou se tornarem muito grandes e pouco práticas, como exemplo pode-se analisar a previsão de palavras em um texto, quando a informação que define a previsão está muito distante no texto não é possível prever, porque a rede recorrente somente guarda informações até certo ponto anterior. Pode-se observar na Figura 9 que a informação X0 pode ser perdida no encadeamento até o ponto onde a rede recebe a informação n+2.

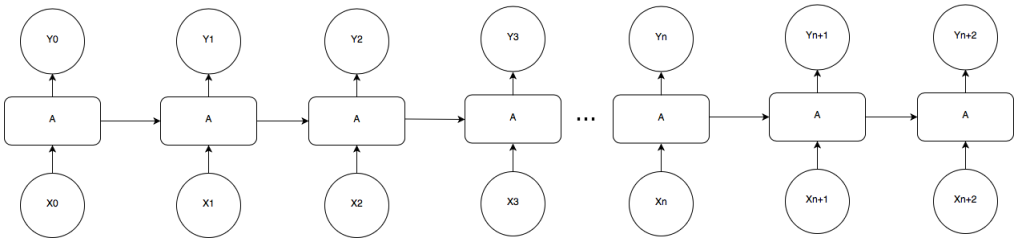


Figura 9: Sequência de Blocos de uma Rede Neural Convolucional

## 3.3 Long Short Term Memory

[20] As redes Long Short Term Memory, chamadas LSTM, são capazes de aprender informações a longo prazo. Essas redes funcionam de forma muito eficiente em uma gama grande de problemas, e por isso são muito utilizadas.

As redes neurais convolucionais são desenhadas para que o problema de dependência de informações anteriores fosse solucionado. Para isso são encadeadas várias redes do mesmo tipo, que têm uma estrutura muito simples parecida com a estrutura das redes neurais convecionais, com função de ativação de tangente hiperbólica observada na Figura 10.

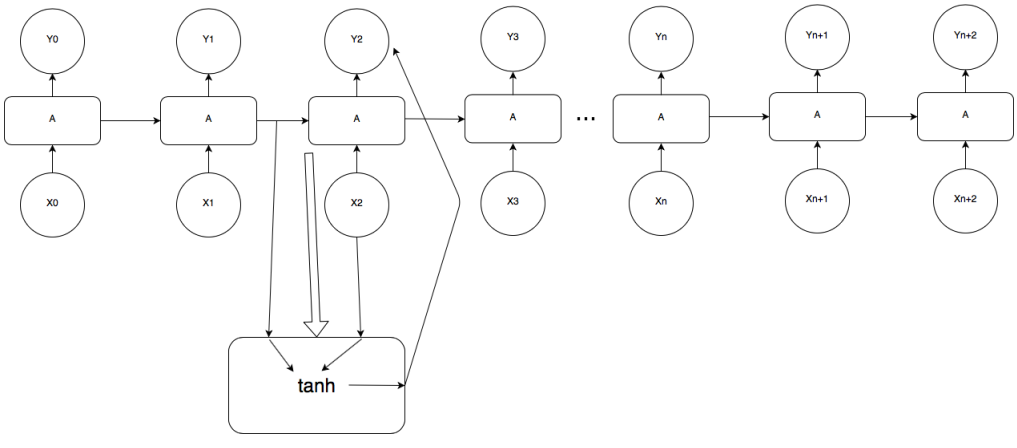


Figura 10: Bloco Long Short Term Memory

As redes LSTM têm também, internamente, uma característica de repetição. Essa repetição tem uma estrutura diferente da descrita anteriormente, porque no lugar de somente uma camada de rede neural, a LSTM tem quatro camadas, que interagem entre si de uma forma muito específica.

Na Figura 11 está detalhada a estrutura de uma célula LSTM, onde cada linha horizontal representa um vetor que vem da saída da célula anterior para o input das próximas. Os círuclos com um X dentro representam operações escalares, como a soma de vetores, onde cada componente é somada independentemente.

A principal característica da célula LSTM é o seu estado, que é a linha horizontal superior na Fogura 11. O estado é como uma transmissão de informação que percorre toda a célula com poucas interferências de interações com outras informações, muitas vezes as informações passam por ela sem ser alteradas.

As células LSTM têm a habilidade de tirar ou colocar informações no estado da célula, isso é regulado pelas estruturas chamadas gates, que são um modo de opcionalmente deixar passar informação, ou não. Essas estruturas são compostas por uma rede neural com função sigmoide e uma multuplicação escalar. A função sigmoide tem como saída valores entre zero e um, que representam a proporção em que cada componente pode ser passada para as próximas células (O valor zero representa que nada dessa componente deve ser passada e o valor um significa que a componente deve ser passada na sua totalidade). Cada célula LSTM possui três desses gates, que protegem e controlam o estado de cada uma delas.

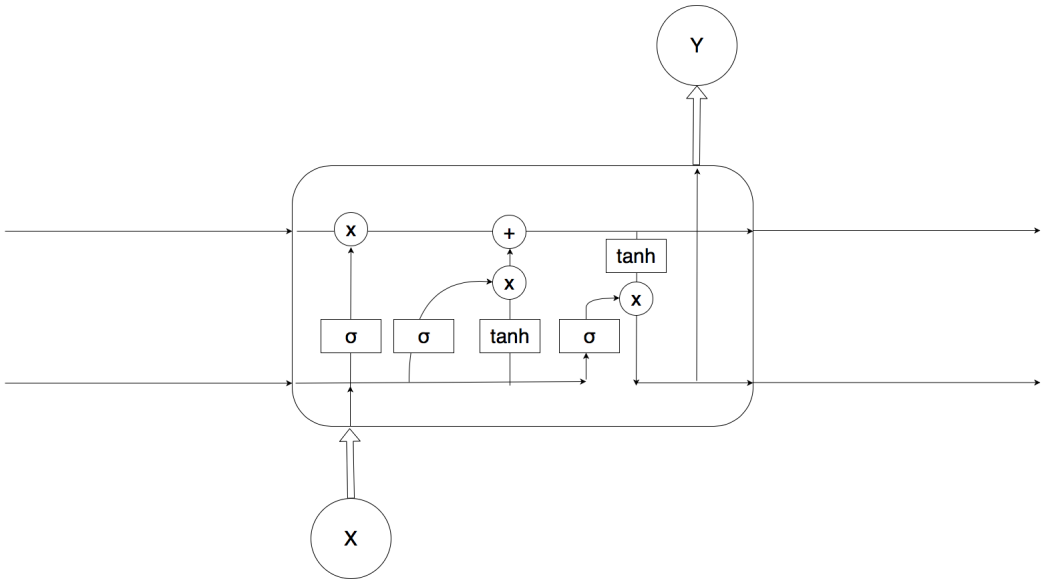


Figura 11: Detalhamento do bloco Long Short Term Memory

### 3.3.1 Camadas de uma rede LSTM

A primeira camada de uma rede LSTM é onde é decidido quais informações vão ser descartadas do estado da célula. Essa decisão é feita por uma camada de função sigmoide chamada de "forget gate layer", ou camada de porta de esquecimento. Essa camada recebe como entradas a saída da célula anterior e a entrada da célula a que ela pertence e retorna um número entre zero e um para cada componente do estado da célula. Nesse caso um valor um representa a manutenção total da componente, e um valor zero representa o descarte dessa componente. As funções dessa camada estão evidenciadas em azul na Figura 12.

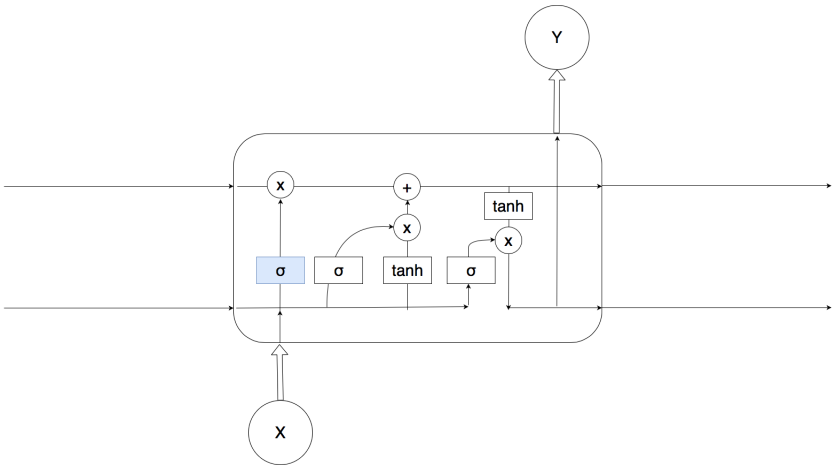


Figura 12: Camada de Esquecimento LSTM

A próxima camada é onde é decidido quais novas informações serão guardadas no estado da célula. Isso é feito em duas etapas. Para começar, é usada uma camada sigmoide chamada "input gate layer", ou camada porta de entrada, que escolhe quais valores serão atualizados. A seguir há uma camada de tangente hiperbólica, que cria um vetor com novos valores candidatos que poderão ser adicionados ao estado da célula. Essa camada está evidenciada em azul na Figura 13.

A terceira camada da célula é a camada que faz a atualização do estado da célula com as informações obtidas nas duas camadas anteriores. Para essa atualização é realizacda a mltiplicação do estado anterior pelo valor calculado na camada de esquecimento, para que sejam selecionados os valores que serão passados adiante na cadeia, pelos valores de entrada. Em seguida são somados os valores calculados na segunda camada (multiplicados entre si), obtendo-se assim os novos valores de estado da célula. Essa camada está evidenciada em azul na Figura 14.

A quarta, e última, camada de uma célula LSTM é a camada que calcula as saídas da

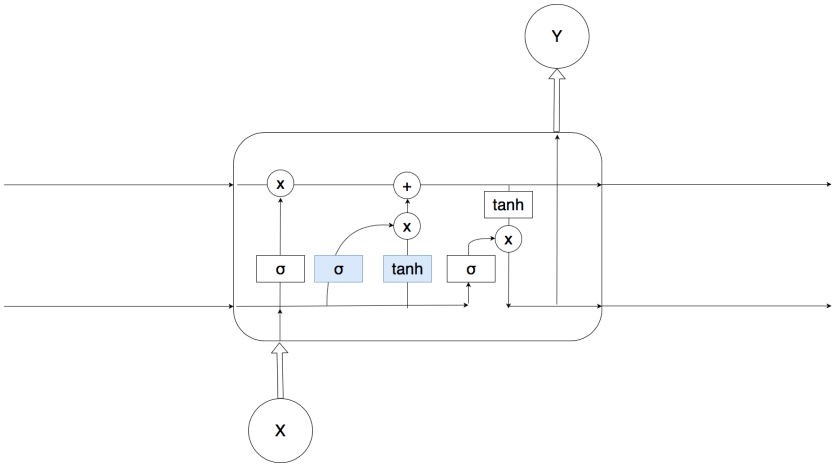


Figura 13: Camada de Entrada

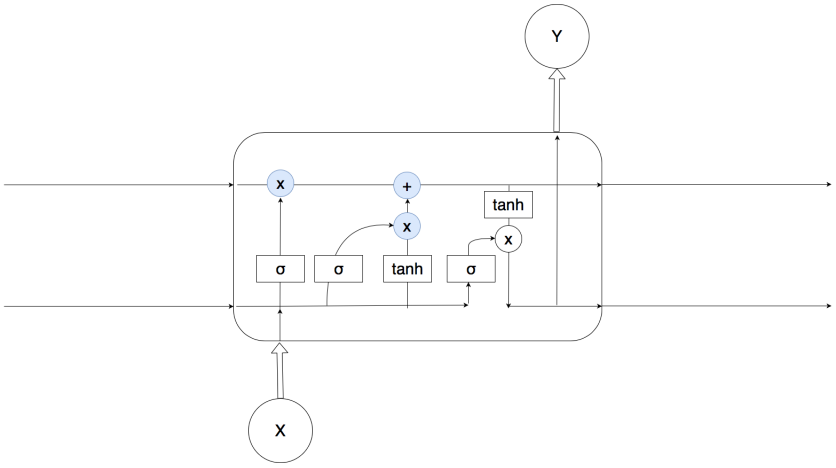


Figura 14: Camada de atualização

célula, que vai ser baseado no estado da célula. Isso é feito através de mais uma camada sigmoide, que escolhe quais partes do estado da célula serão disponibilizadas na saída da célula. A partír daí o estado atual da célula é enviado na saída de estado, pasando antes por uma tangente hiperbólica para que os valores estejam entre zero e um. Essa camada está evidenciada em azul na Figura 15.

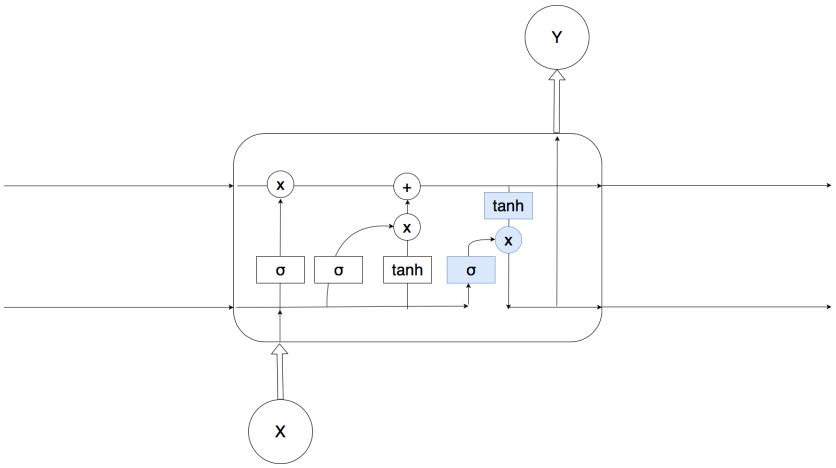


Figura 15: Camada de saída

### 3.3.2 Gated Recurrent Unit (GRU)

A unidade de porta recorrente é uma variação da LSTM. Essa variação é usada por ter menor exigência computacional. Numa rede dessas existe uma porta de atualização que controla tanto a porta de input como a porta de esquecimento. Outra diferença importante nessa variação é que a porta de reset é usada antes da transformação não-linear.[21]

## 3.4 Preparação do Ambiente

Para se obter um modelo treinado, baseado na rede 3D-R2N2[2], que foi a escolhida para o modelo foi necessária a adaptação de algumas características. Essa rede neural convolucional é implementada em Python e usa algumas bibliotecas, listadas a seguir:

- numpy: a biblioteca numérica de Pyhton
- Theano: outra biblioteca numérica do python, muito usada em aplicações que precisam de gpu
- EasyDict: biblioteca de dicionários do python

- Pillow: biblioteca de imagens do python
- pyyaml: biblioteca de serialização de dados do python
- sklearn: biblioteca de machine learning do python

Além disso foi necessária a preparação de !!!!!!

## 3.5 A Arquitetura Escolhida

Problemas de reconstrução 3D a partir de somente uma imagem ou de múltiplas imagens, a partir de diferentes ângulos, são muito diferentes, e por isso são abordados de formas diferentes. Nesse modelo a ordem com que as imagens do objeto são inseridas na rede não importa, isso é possível porque uma rede neural do tipo LSTM que faz a seleção das informações que serão guardadas em cada camada, o que faz com que informações repetidas sejam descartadas e informações escondidas em uma imagem sejam guaradas para serem obtidas de uma outra imagem.

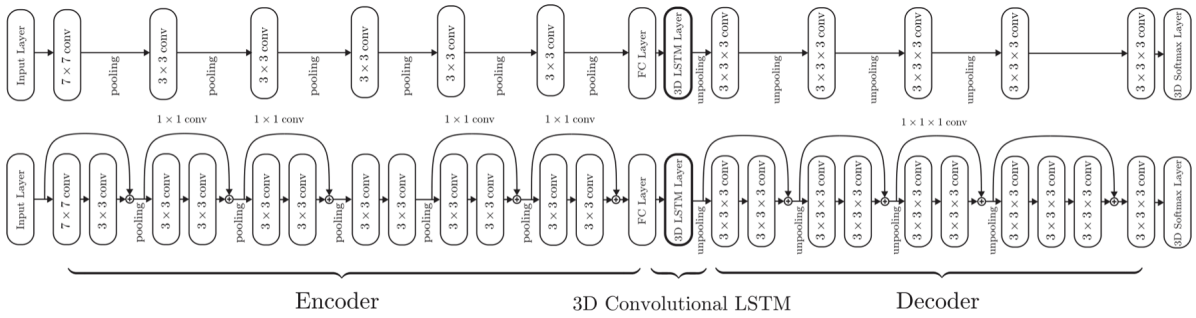


Figura 16: Arquitetura 3D Recurrent Reconstruction Network (3D-R2N2)[2]

O objetivo dessa nova arquitetura proposta é reconstruir modelos 3D a partir tanto de uma imagem única como de múltiplas imagens. A ideia principal é se aproveitar da capacidade de persistência de informação da LSTM para que o modelo na saída seja mais preciso quanto mais imagens (mais informação) for fornecida para a rede. Essa arquitetura é composta por três partes, uma Rede Neural Convolucional 2D (2D-CNN), uma LSTM Convolucional (3D-LSTM) e uma Rede Neural Desconvolucional (3D-DCNN), que estão representadas na Figura 16.

A primeira rede neural, a 2D-CNN, codifica cada imagem de entrada para features de menor dimensão. A partir disso, a rede convolucional de LSTM (3D-LSTM) recebe essas informações transformadas e seleciona as que devem ser mantidas através da atualização

de estados descrita anteriormente e para finalizar a rede 3D-DCNN decodifica os estados escondidos das células LSTM e gera uma reconstrução probabilística 3D.

### 3.5.1 Descrição da Arquitetura[2]

A arquitetura da 3D-R2N2 pode ser separada 3 em partes: a etapa de codificação (uma rede neural convolucional 2D, ou 2D-CNN), a etapa de convolução (nesse caso chamada LSTM Convolucional 3D, ou 3D-LSTM), e a camada de decodificação (nesse caso chamada rede neural deconcolucional, ou 3D-DCNN), que serão descritas a seguir.

Nessa rede, dadas uma ou mais imagens de um objeto, não importando o ângulo do qual a imagem foi obtida, a primeira etapa da rede realiza uma redução de dimensionalidade, pegando cada imagem de entrada x e transformando numa informação de menor diemnsão T(x). A partir disso, um conjunto de blocos 3D-LSTM passa a atuar, e por fim a camada 3D-DCNN decodifica os estados escondidos das unidades LSTM para gerar uma reconstrução probabilística voxelizada.

As unidades LSTM foram escolhidas porque elas conseguem prever partes escondidas e obter uma reconstrução fiel a partir de imagens do mesmo objeto vistas de outros ângulos. Essa rede atualiza as partes onde esses pedaços escondidos estão a partir das novas imagens e mantem as unidades correspondentes às outras partes.

#### 3.5.1.1 Camada de Codificação

A primeira camada da rede é a camada que recebe as imagens de entrada e codifica elas para que a segunda etapa, da rede convolucional, seja realizada. No caso dessa arquitetura existem dois tipos de blocos de 2D-CNN, um feed-forward convencional e uma variação deep residual dela. A primeira rede é formada por camadas de convolução tradicionais, camadas de pooling e unidades leaky rectified linear com uma camada altamente conectada.

Ao adicionar conexões residuais o processo de otimização é acelerado para redes muito profundas, as variações dessas redes têm uma função identidade nas suas conexões a cada duas camadas de convolução exceto no quarto par. O número de canais após cada convolução é 1 convolução por conexão. A saída do encoder tem sua dimensionalidade reduzida e passada para uma camada altamente conectada que transforma a saída num vetor de 1024 dimensões.

#### 3.5.1.2 Camada de convolução

A camada de convolução é a camada de recorrência. Essa é a camada mais importante da rede, que mantém as informações dos exemplos e atualiza a memória a cada novo exemplo. Nesse caso é usada uma arquitetura chamada 3D-LSTM, que é composta de unidades LSTM com conexões limitadas. Essas unidades são distribuídas espacialmente e cada unidade é responsável pela reconstrução de um pedaço específico da saída.

As equações que definem a 3d-LSTM são:

$$f_t = \sigma(W_f \tau(x_t) + U_f \times h_{t-1} + b_f)$$

$$i_t = \sigma(W_i \tau(x_t) + U_i \times h_{t-1} + b_i)$$

$$s_t = f_t \otimes s_t - 1 + i_t \otimes tanh(W_s \tau(x_t) + U_s \times h_{t-1} + b_s)$$

$$h_t = tanh(s_t)$$

No caso da 3D-LSTM, cada módulo somente pode ser afetado pelo estado de seus módulos vizinhos, e não por seus estados anteriores. Nesse caso, esse módulo é uma unidade baseada em GRU, descrita anteriormente. As equações que definem essas relações são:

$$u_t = \sigma(W_{f_x} \tau(x_t) + U_f \times h_{t-1} + b_f)$$

$$r_t = \sigma(W_{i_x} \tau(x_t) + U_i \times h_{t-1} + b_i)$$

$$h_t = (1 - u_t) \otimes h_{t-1} + u_t \otimes tanh(W_h \tau(x_t) + U_h \times (r_t \otimes h_{t-1}) + b_h)$$

#### 3.5.1.3 Camada de decodificação

A camada de decodificação é o último bloco da rede. Essa é a etapa onde o modelo voxelizado é gerado, e onde se obtém a saída. Esse bloco é um bloco simples com 5 convoluções e uma variação da deep residual com 4 conexões residuais seguidas de uma convolução final. Na saída da última camada, que é onde a saída é determinada. Com isso a probabilidade de preenchimento de cada voxel usando uma função softmax.

### 3.5.2 Implementação

Antes de se iniciar o treino da rede foram usados modelos 3D gerados em CAD (Desenho assistido por Computador), que são as imagens usadas como entradas da rede e as saídas esperadas para cada conjunto de imagens.

A seguir começa a fase de treino da rede, onde são usadas essas imagens e modelos 3D (as saídas esperadas), em cada parte do treinamento foram usados conjuntos de entradas e saídas com o mesmo número de imagens (perspectivas), mas entre as rodadas eram usados números diferentes de imagens, o que possibilita a reconstrução a partir de uma só imagem.

O tamanho das imagens é o mesmo, independente do número de imagens, cada uma tem 127 por 127 pixels e a saída também tem um tamanho padrão de 32 x 32 x 32 unidades. O treino foi realizado em 60000 iterações, em blocos de 36 exemplos.

Utilizando a biblioteca de Python *Theano*, usada para aplicações em Tensor, integrada com o *Numpy* foi descrita a arquitetura a ser seguida. Abaixo o exemplo de definição de camadas:

```
# To define weights, define the network structure first
        x = InputLayer(input_shape)
        conv1a = ConvLayer(x, (n_convfilter[0], 7, 7))
        conv1b = ConvLayer(conv1a, (n_convfilter[0], 3, 3))
        pool1 = PoolLayer(conv1b)

        conv2a = ConvLayer(pool1, (n_convfilter[1], 3, 3))
        conv2b = ConvLayer(conv2a, (n_convfilter[1], 3, 3))
        conv2c = ConvLayer(pool1, (n_convfilter[1], 1, 1))
        pool2 = PoolLayer(conv2c)
```

Similarmente definimos a recorrência para as camadas de convolução e atualizamos os parâmetros de cada camada.

```
        # Scan function cannot use compiled function.
        input_ = InputLayer(input_shape, x_curr)
        conv1a_ = ConvLayer(input_, (n_convfilter[0], 7, 7),
                  params=conv1a.params)
        rect1a_ = LeakyReLU(conv1a_)
```

```
conv1b_ = ConvLayer(rect1a_, (n_convfilter[0], 3, 3),
          params=conv1b.params)
rect1_ = LeakyReLU(conv1b_)
pool1_ = PoolLayer(rect1_)

conv2a_ = ConvLayer(pool1_, (n_convfilter[1], 3, 3),
          params=conv2a.params)
rect2a_ = LeakyReLU(conv2a_)
conv2b_ = ConvLayer(rect2a_, (n_convfilter[1], 3, 3),
          params=conv2b.params)
rect2_ = LeakyReLU(conv2b_)
conv2c_ = ConvLayer(pool1_, (n_convfilter[1], 1, 1),
          params=conv2c.params)
res2_ = AddLayer(conv2c_, rect2_)
pool2_ = PoolLayer(res2_)
```

Para realizar a *deconvolução* foi utilizada a regra de atualização *Stochastic gradient descent* também aplicada em Choy et al.[22] para otimização. Ao treinar a rede foram usados números variados de imagens de entrada, desde que a cada rodada de treino fossem usados sempre exemplos com mesmo número de imagens de entrada, oq ue permite que a rede reconstrua modelos com uma ou mais imagens. O parâmetro loss somente foi calculado ao fim de cada rodada para diminuir o consumo computacional e de energia e memória.

## 3.6 Obtenção das Imagens

As imagens utilizadas nessa etapa de treino são obtidas no repositório ShapeNet, que é disponível para pesquisadores de todo o mundo usarem e é a colaboração entre pesquisadores de Princeton, Stanford e TTIC.[11]

## 3.7 Resultados obtidos

Após o treino da rede neural foi possível testar o modelo. Para o teste foram utilizadas imagens do repositório disponível, na Figura 18 observam-se as imagens de entrada, e na Figura 17 observa-se a imagem do modelo gerado.

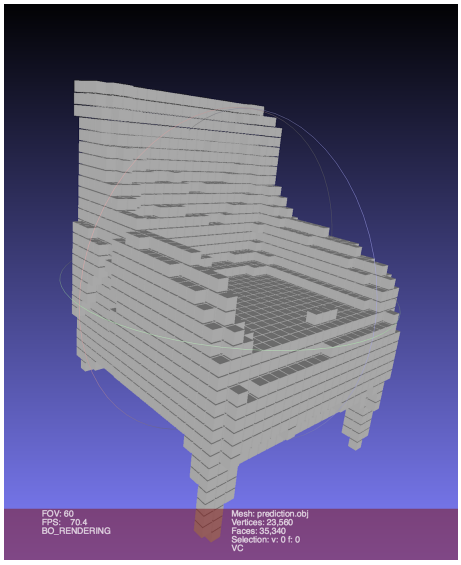


Figura 17: Resultado da reconstrução

Figura 18: Imagens base para reconstrução

Também foram realizados testes com imagens obtidas fora do conjunto disponibilizado com alguns erros a serem acertados, como o cuidado com o fundo da imagem, o que pode ser observado na Figura 19 onde há fundo colorido de cor próxima à cor do objeto, resultando em erro na Figura 20, além disso, também houve distorção das imagens para que elas se enquadrassem nas dimensões que as imagens de entrada devem ter, o que resultou em menor precisão da reconstrução.

Figura 19: Imagens base para reconstrução

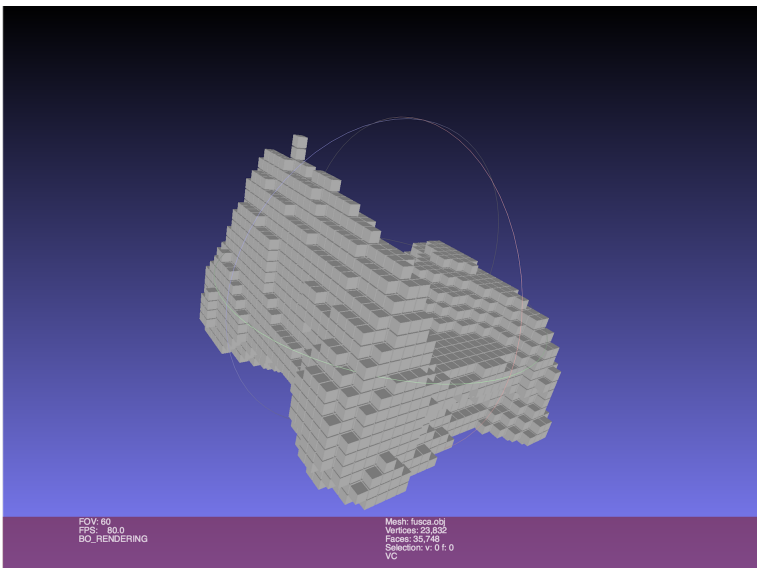


Figura 20: Resultado da reconstrução

Também houve sucesso com imagens externas, na rescontrução da câmera, com as imagens da Figura 21, que resultaram na Figura 22. Nesse teste houve distorção nas imagens de entrada, e o fundo tinha uma cor diferente do objeto, o que não prejudicou tanto a reconstrução.

Figura 21: Imagens base para reconstrução

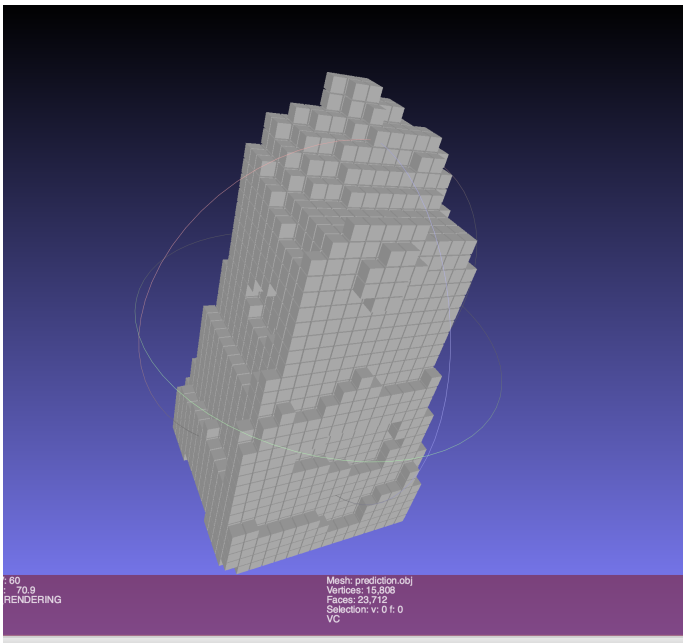


Figura 22: Resultado da reconstrução

## 3.8 Testes com Imagens de Ressonância Magnética

Para validar a rede que foi treinada com o dataset ShapeNet o código foi levemente alterado para podermos mudar a quantidade de imagens que são alimentadas na rede. Antes disso as imagens de ressonância magnética precisaram ser processadas para que elas tivessem o formato adequado para a reconstrução.

### 3.8.1 Pré-processamento das imagens de ressonância magnética

Para o pré-processamento das imagens foram realizadas algumas etapas. A primeira é a obtenção de um vídeo com a sequência das imagens, em seguida, na segunda etapa, esse vídeo tem os frames separados em imagens para que tenhamos as imagens de cada seção e usando o vídeo podemos separar em quantas imagens forem necessárias. A terceira etapa é a etapa para adequar o tamanho de cada imagem ao tamanho suportado pela rede.

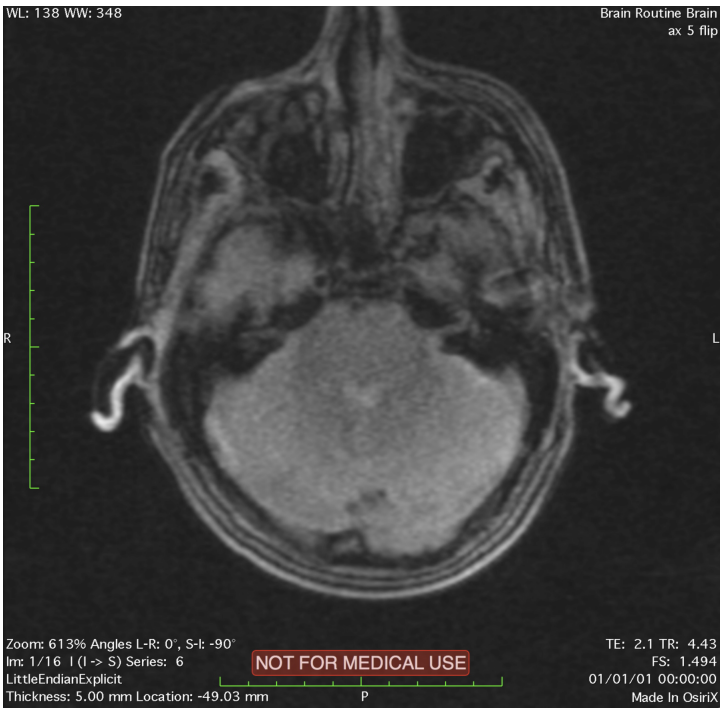


Figura 23: Video

Para gerar o vídeo com a sequência de seções foi usado o software OsirixLite, que permite a vizualização de imagens médicas. Os vídeos foram exportados no formato MOV, que tem um frame representado na Figura 23.

A separação dos frames foi feita através do software ffmpeg[15], que permie que o video seja decomposto em imagens no formato png, ideal para ser usado na rede. Depois disso

as imagens tiveram seu tamanho redefinido para 128x128 pixels, para serem alimentadas na rede.

A seguir foi testado o resultado dessa separação com pequenas alterações no código da rede para que fosse possível aumentar o número de imagens usadas na resoncstrução.

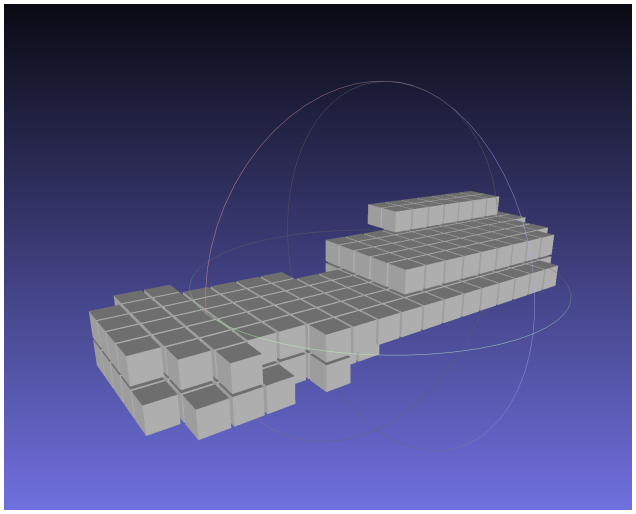

Figura 24: Primeiro Teste realizado com imagens paralelas

O primeiro teste realizado com seções paralelas não obteve bons resultados, porque a rede é treinada para imagens de diferentes perspectivas, o que torna a rede inadequada para outros tipos de imagem. O resultado pode ser explicado como a rede enxergando todas as imagens como uma perspectiva de uma imagem plana e reconstruindo como se somente houvesse essa dimensão. Uma solução para isso é um novo treinamento da rede com imagens paralelas e seus respectivos modelos.

# 4 IMPLEMENTAÇÃO EM HARDWARE

## 4.1 Escolha do Hardware

### 4.1.1 Seleção de plataforma

É necessário uma plataforma de prototipagem que permita o desenvolvimento de

### 4.1.2 Escolha de placa

A partir de um conjunto de hardwares disponíveis no laboratório de sistemas digitais foi escolhido um dispositivo SoC, System on Chip, um sistema que integra FPGAs com hard processor system (HPS, ARM-based), que consiste em um processador, periféricos, e interface de memory com a FPGA utilizando ampla largura de banda para interconexão. Ou seja, permite a flexibilidade e desempenho em um único sistema lógico programável integrado, possibilitado por serem impressas em um mesmo chip.

O dispositivo escolhido representa a melhor escolha, pois:

- Melhora a perfomance do sistema utilizando uma interconexão altamente eficiente entre processador e FPGA;
- Permite personalizações em hardware e em software;

A placa escolhida foi a Terasic Arrow SoCkit, que embarcam dual-core Cortex-A9 e FPGA Altera Cyclone V. A placa inclui periféricos de alto desempenho como memória DDR3, Ethernet, portas seriais, entre outros. Portanto avaliamos como a melhor solução para demonstrar, avaliar e prototipar o projeto.

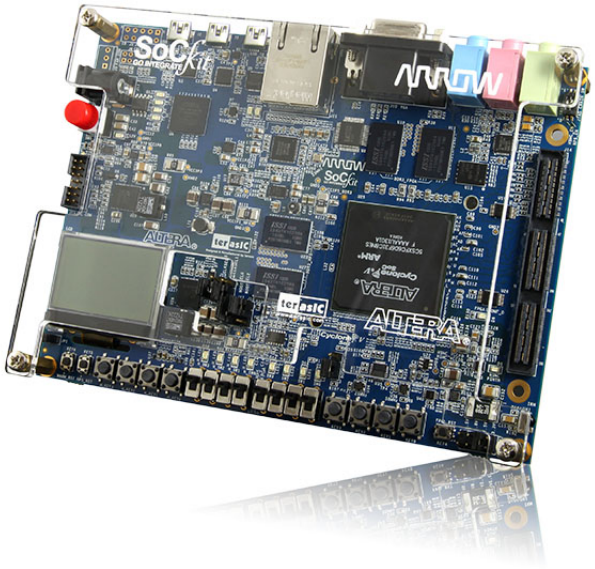


Figura 25: Placa Terasic Arrow SoCkit

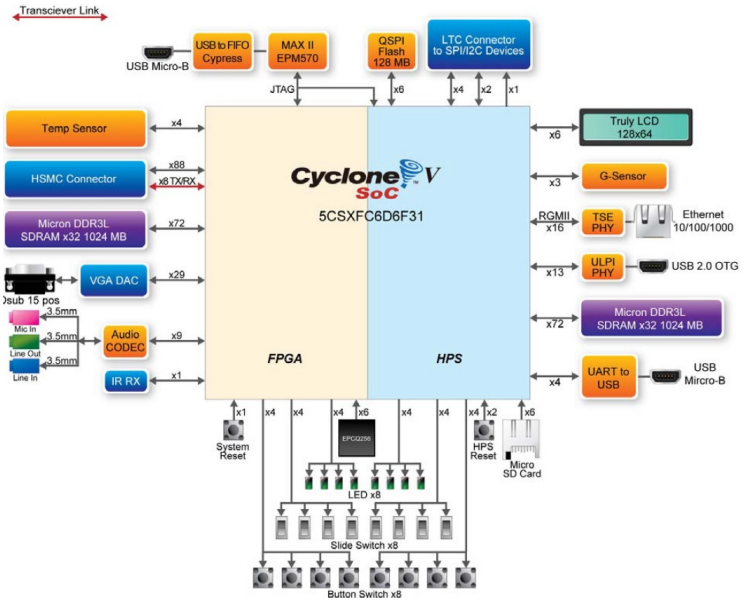


Figura 26: Diagrama de blocos do arranjo interno da placa

## 4.2 Hardware Design

### 4.2.1 Estrutura Geral

Utilizando a placa Arrow SoCkit primeiramente foi implementado um arranjo de Hardware simples que demonstra o funcionamento da interface entre o processador e a FPGA. O sistema realiza o boot utilizando a distribuição de Linux Angstrom, baseado no projeto Yocto [23].

A estrutura *Top Level* de Hardware foi separada em duas diferentes estruturas, como mostrado na figura 27. Sendo uma delas o neurônio em si e a outra chamada SoC System, que serão descritas a seguir.

### 4.2.2 Neurônio

A implementação do neurônio foi divida em alguns blocos. Essa divisão foi realizada baseada em funcionalidade do módulo.

A unidade *Multiplier-Accumulator* é responsável por armazenar temporariamente e operar valores de entradas e pesos, de forma que não se faz necessário diversas entradas de vários bits, o que causaria sobrecarga da FPGA para diversos neurônios em função de suas limitações. A figura 28 mostra a representação interna da unidade.

Foi implementado em FPGA Cyclone V um modelo de entradas sequenciais e ativação utilizando *Rectifier Linear Unit (ReLU* e uma unidade *Multiplier-Accumulator (MAC)*, Figura 29, para efetuar as funções de camada de entrada.

As *Rectified linear units*, comparadas com funções sigmoides ou funções de ativação similares, permitem treinamento rápido e eficaz de redes neurais com arquiteturas expandidas e bases de dados complexas. A função de ativação *Rectifier Linear Unit* (ReLU) é uma aproximação suavizada da função analítica:

$$f(x) = log(1 + e^x)$$

Resultando na seguinte expressão:

$$f(x) = x^+ = \max(0, x)$$

Comparando diretamente as funções tem-se a aproximação mostrada na Figura 30.

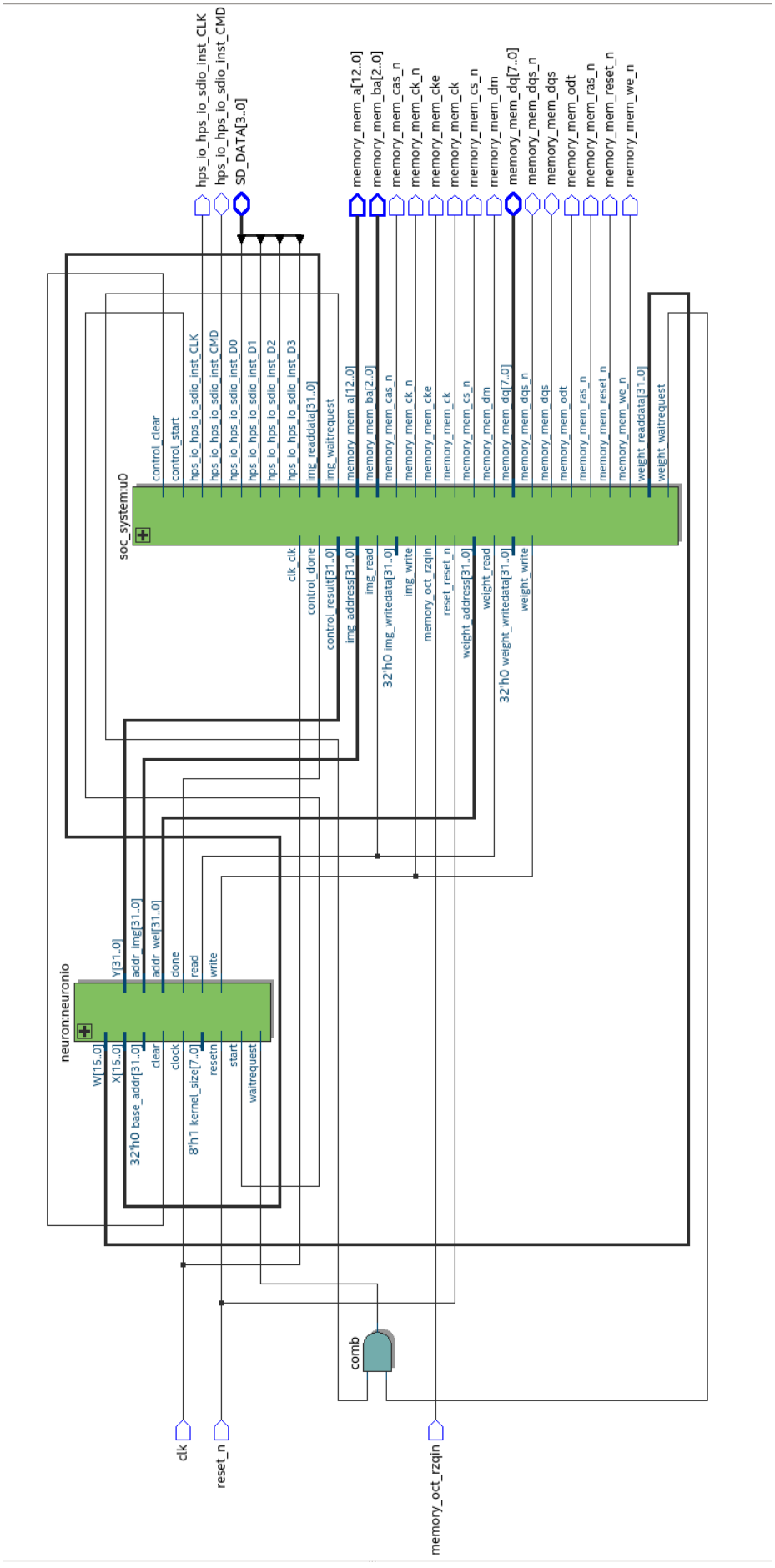


Figura 27: Estrutura Top Level

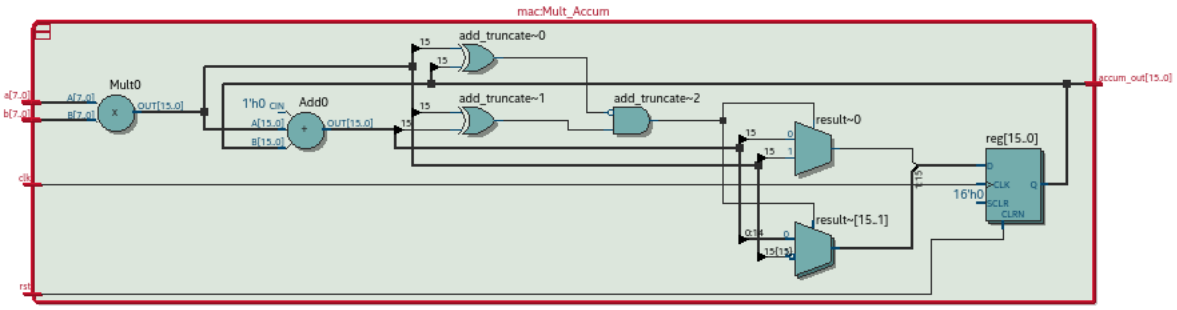


Figura 28: *Multiplier-Accumulator* Design interno

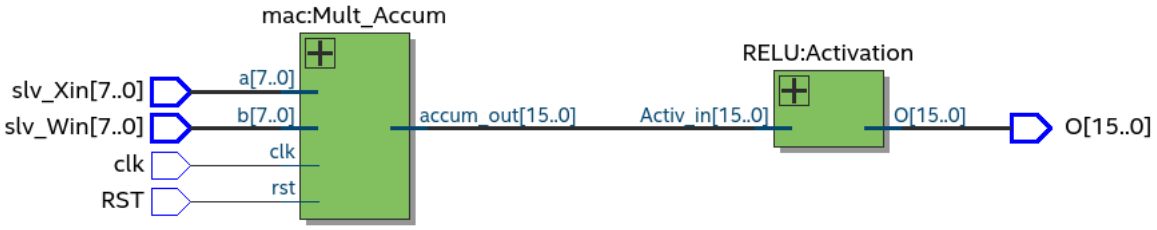


Figura 29: Neuronio simplificado

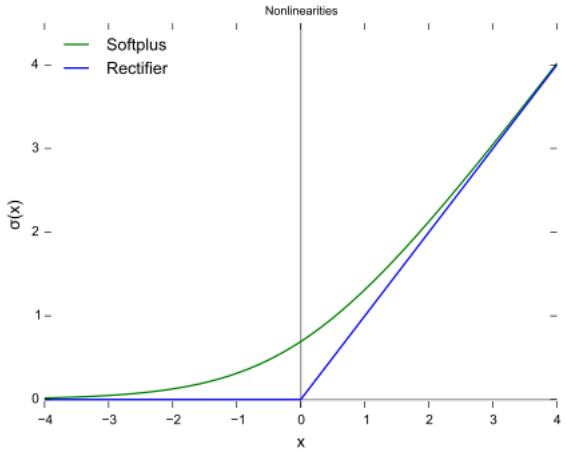


Figura 30: Comparação ReLU com *Softplus*

Vantagens:

- **Ativação dispersa:** Em uma rede inicializada aleatoriamente, aproximandamente 50% de taxa de ativação;
- **Melhor propagação gradiente:** Comparada com a função sigmoidal, bastante utilizada como função de ativação em redes neurais[24];
- **Eficiência computacional:** Envolve apenas funções de comparação, adição e multiplicação.
- **Linearmente invariante:** $\max 0, ax = a \max 0, x$ para $a >= 0$

Após carregar as informações de base, devemos formar os endereços de leitura da memória RAM onde estão gravados os pesos da rede já treinada externamento. Para isso, vamos utilizar o seguinte bloco representado na Figura 31 que, a partir do endereço base, gera novos endereços de leitura e passa para a unidade de controle da memória realizar a leitura. Internamente o arranjo para gerar os endereços atua da seguinte forma descrita pelo diagrama representado na Figura 32, onde i e j são gerados de acordo com o tamanho da matriz de convolução definida pela entrada n.

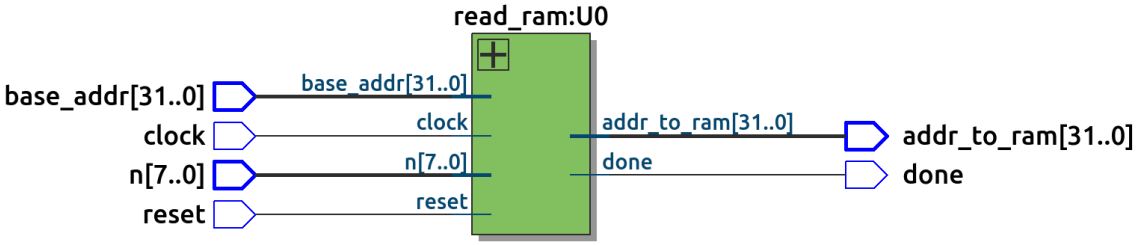


Figura 31: Bloco de geração de endereços de leitura

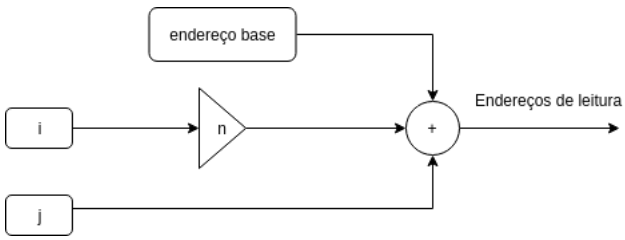


Figura 32: Diagrama de geração de endereços de leitura

O componente neuron_dp, representado na Figura 33, é o componente onde as operações são realizadas, ele é composto por um módulo de leitura da memória e um *Multiply-*

*Accumulate*. Com isso ele recebe os valores de peso (W) e de entrada (X), lidos pelo componente read_ram e realiza as operações para que sua saída Y seja obtida.

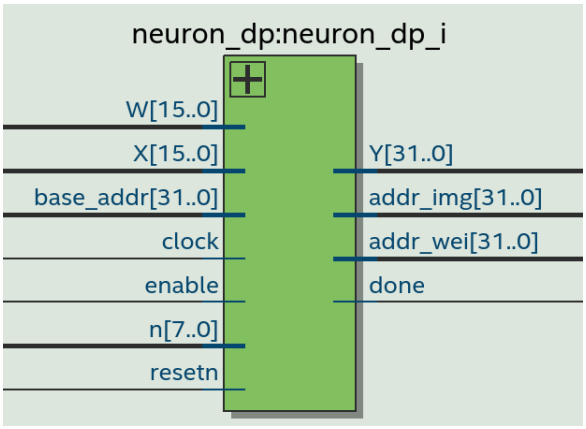


Figura 33: Neuron dp

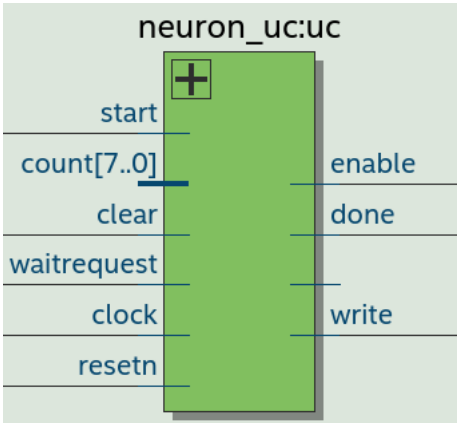


Figura 34: Unidade de Controle

### 4.2.3 *SoC System*

A descrição em bloco *Top Level* do System on Chip apresentado na figura 35 integra memórias, interfaces Avalon, *Hard Processing System* (ARM) e unidade de controle para o sistema neurônio, como apresentado na figura 36. Utilizando a ferramenta *Platform Designer*, mostrada na figura 36 é possível adicionar ligações entre componentes de propriedade intelectual da Altera, como instanciações de memórias e *Hard Processing System* já projetadas para a placa específica, basta apenas declarar no projeto o dispositivo específico utilizado. Componentes já descritos e configuráveis são disponibilizados em um catálogo de propriedades intelectuais. Ainda utilizando esta ferramenta é possível criar componentes a partir de design próprio, como é o caso das interfaces Avalon e o contro-

lador do neurônio. Permitindo, portanto, que conexões entre esses componentes sejam criadas e os blocos se relacionem de forma síncrona.

O endereçamento das pontes é fixo, como descrito na figura 37, já o endereço de base das memórias pode ser configurado permitindo flexibilidade de *offset*, além disso, a escolha de endereços evita conflitos entre componentes declarados no *Platform Designer*


Figura 35: SoC System Top Level

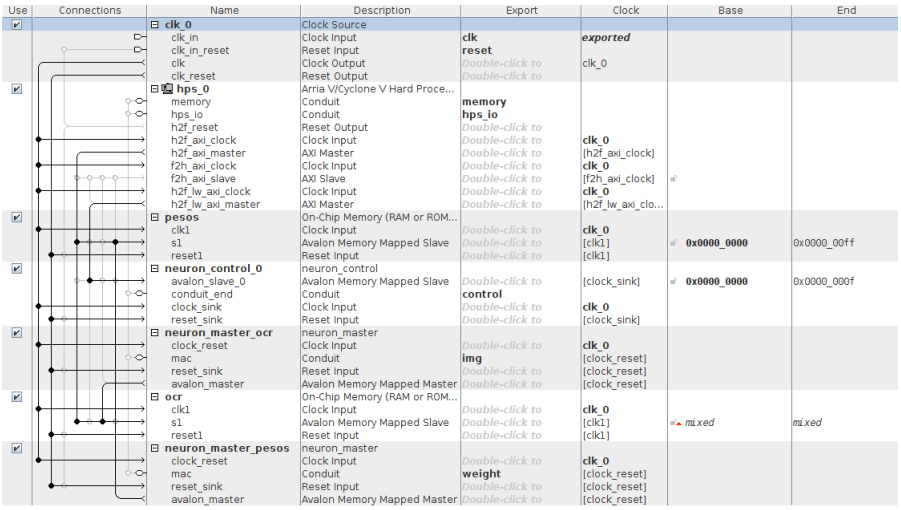


Figura 36: *Platform Designer* para o Sistema SoC

A estrutura interna do sistema é representada pela figura 38. Em primeiro olhar parece um sistema confuso devido ao número de entidades que são interconectadas, no

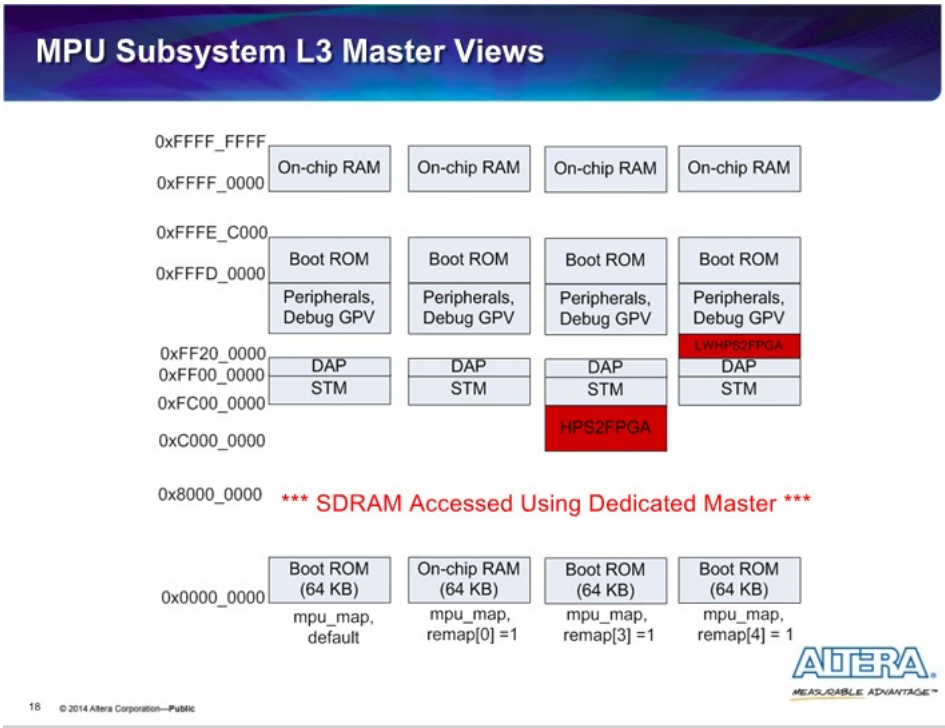


Figura 37: Endereços bridges

entanto será abaixo descrito o comportamento e a origem de cada sub-sistema:

- *Neuron Control*;
- *Hard Processing Unit - HPS*;
- Memórias de peso e imagem;
- Interfaces Avalon Master;

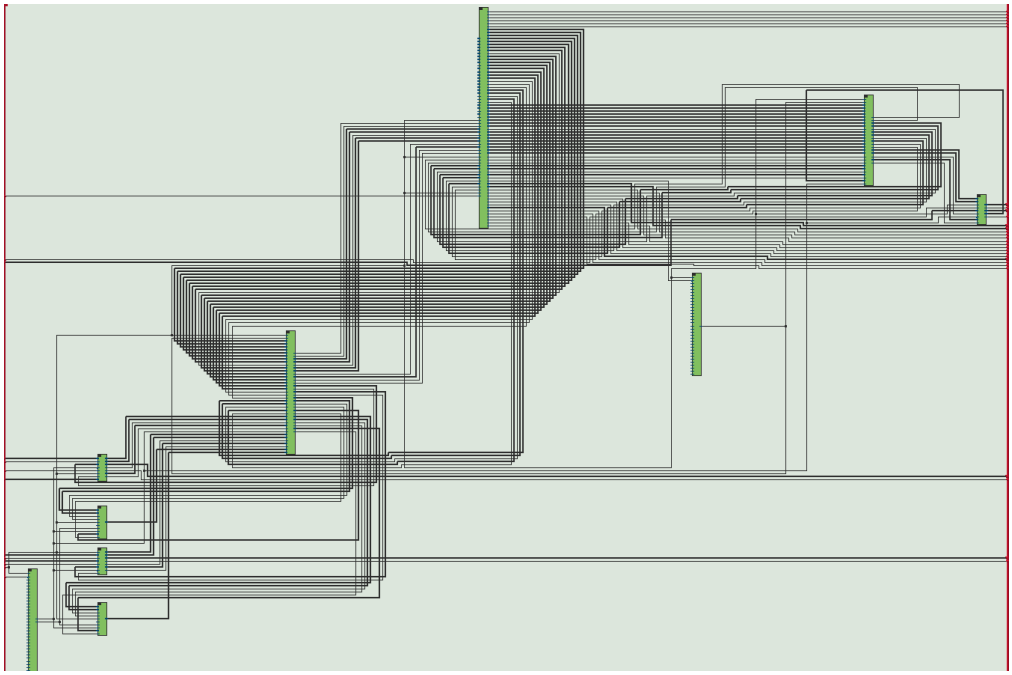

Figura 38: SoC System Interno

#### 4.2.3.1 *Neuron Control*

Portanto, o projeto de processador neuromórfico para carregar as configurações de entradas, pesos e convoluções.

Define-se 2 bits para endereçamento de modos de carregamento de inputs, como mostra a tabela 5. As informações são armazenadas em registradores e enviadas para a próxima camada, que irá interpretar esses dados.

A partir desses bits de multiplexação, uma máquina de estados gera outputs de controle, como mostrado na figura 40.

| Bits | Multiplexação |
|---|---|
| 00 | endereço base |
| 01 | número de convoluções |
| 10 | início |
| 11 | fim |

Tabela 5: Tabela de multiplexação de estados de *Neuron Control*

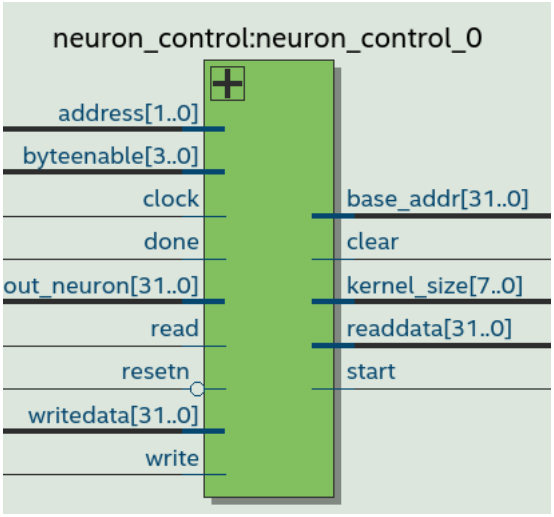


Figura 39: SoC System *Neuron Control*

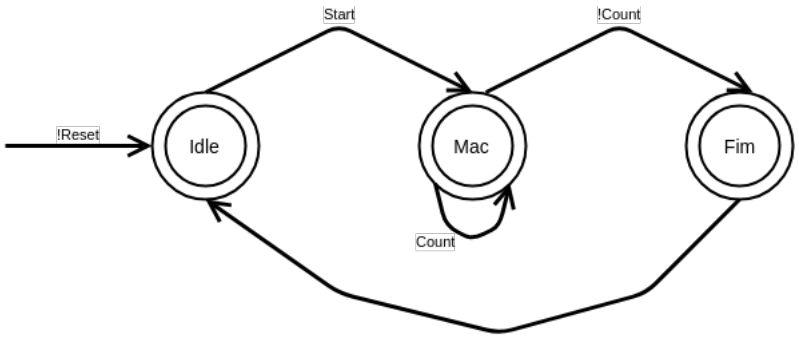


Figura 40: Máquina de estados - *Neuron Control*

#### 4.2.3.2 Memórias e Masters

O sistema de memórias é composto por duas memórias, dois controladores das memórias. Nesse projeto as memórias são acessadas ao mesmo tempo no mesmo endereço, para maior simplicidade. Todos esses componentes estão representados na Figura 42.

Cada bloco de memória é lido pelo componente f2_ram, representados na Figura 42, que tem como principal função controlar os sinais de escrita e leitura das memórias.

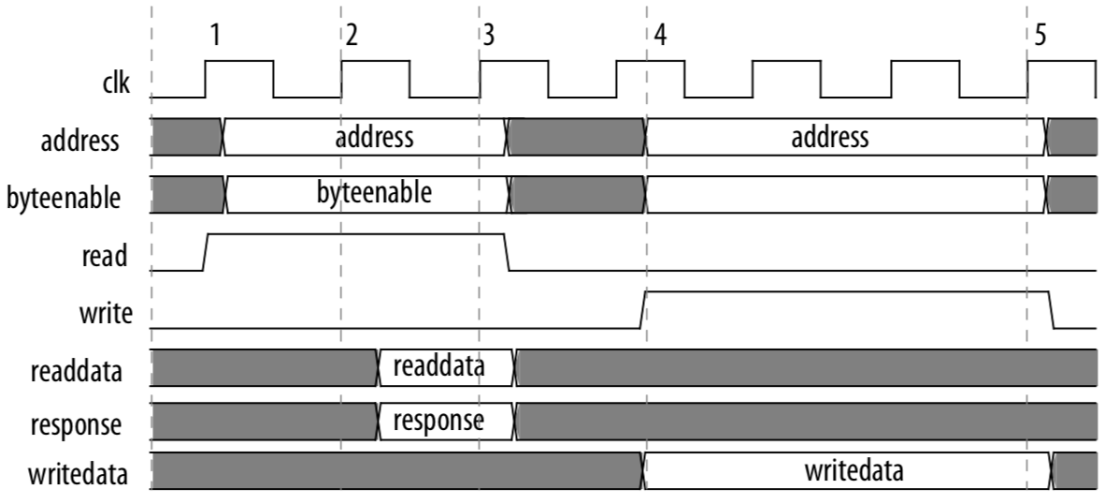


Figura 41: Funcionamento da leitura e escrita da memória

Cada bloco de memória tem um funcionamento descrito na Figura 41. Os sinais de controle do bloco são read, write e byteenable, que têm como função habilitar a leitura, a escrita e o deslocamento do endereço da memória, respectivamente. Com os sinais read e de endereço ativos o bloco passa a ter em sua saída readdata a informação disponível na memória no endereço definido na entrada address, representados no diagrama da Figura 41. Além dessa informação na saída, existe um sinal chamado response, que representa um status sobre o que está sendo lido.

Na escrita da memória é ativado o sinal write, e é colocado na entrada writedata da memória a informação a ser gravada. Além disso é especificado o endereço em address e o deslocamento em byteenable, como observado na Figura 41.

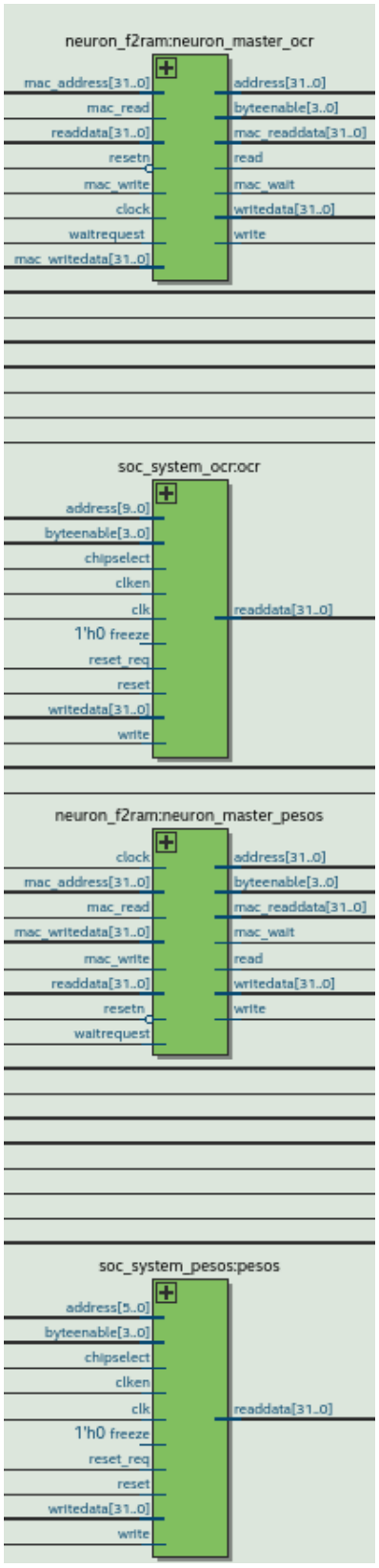


Figura 42: SoC System Memórias e Master

#### 4.2.3.3 *Hard Processing Unit* e Interconexões

O bloco HPS, representado na figura 45, é gerado a partir de uma *Intellectual Property* no catálogo disponibilizado pela Altera, configurando para o dispositivo e habilitando as pontes HPS2FPGA e LWHPS2FPGA ambas para palavras de 32-bits. Para a interface de leitura e escrita entre a FPGA e o HPS os endereços de escrita, conforme a figura 37, a *Light Weight Bridge* é 0xff200000 e a *HPS Bridge* é 0xc0000000. Blocos de interconexão, figuras 43 e 44, são criados no *Platform Designer*, a partir do relacionamento entre os blocos HPS, *Neuron Control*, Masters e memórias. A geração é automática sem necessidade de arquivos de configuração externos.

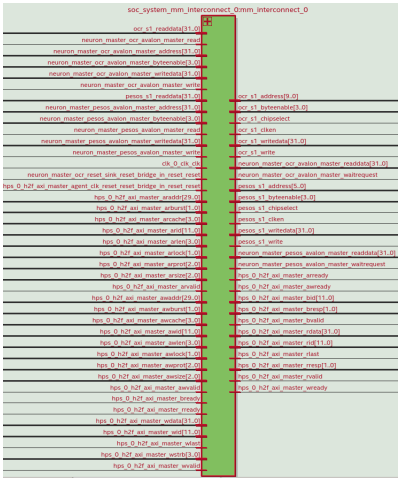


Figura 43: SoC System Interconexão

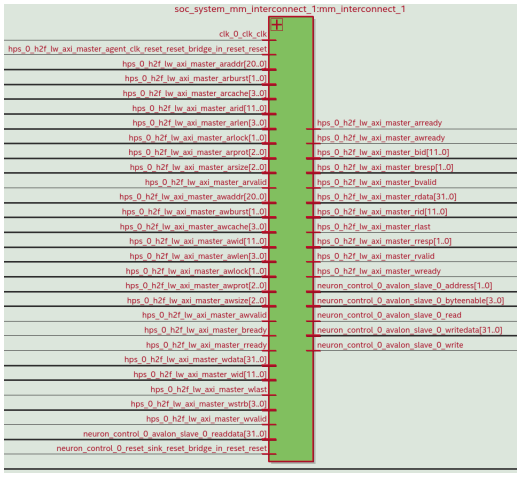


Figura 44: SoC System Interconexão 2

Figura 45: SoC System Hard Processing System

## 4.3 Descrição Lógica dos Blocos

O componente de *Neuron Control* na figura 46 atua como função de multiplexação e registro de dados. A variável *address* de 2 bits seleciona qual registro fazer a partir da entrada vinda das pontes de comunicação.

O bloco *Neuron Control* é o primeiro bloco lógico que recebe sinais de comunicação provenientes do ARM, sinais de dados e seleção, que são transmitidos a partir da interconexão gerada pela interface Avalon.

O componente *Multiply Accumulate*, descrito na Figura 28, tem como sinais de controle *reset* e *enable*. A partir do estado *Start*, quando o sinal *reset* está inativo, e o sinal *enable* é acionado desencadeiam-se as operações a serem realizadas de multiplicação e soma, como pode ser observado na Figura 47.

O componente f2_ram tem seu comportamento descrito no diagrama da Figura 48. A partir de um estado *start*, quando o sinal *neu_write* é acionado, os sinais de saída *write* e *read* têm seus valores definidos para 1 e 0, respectivamente, assim como quando o sinal neu_read é acionado, as saídas *write* e *read* recebem os valores 0 e 1, respectivamente. Quando tanto *neu_write* e *neu_read* estão desativados, as saídas do componente mantém-se com valor 0.

A unidade de controle, representada pela figura 50. Tem seu funcionamento descrito pelo diagrama ASM da figura 49, que após os valores registrados, gera sinais de controle

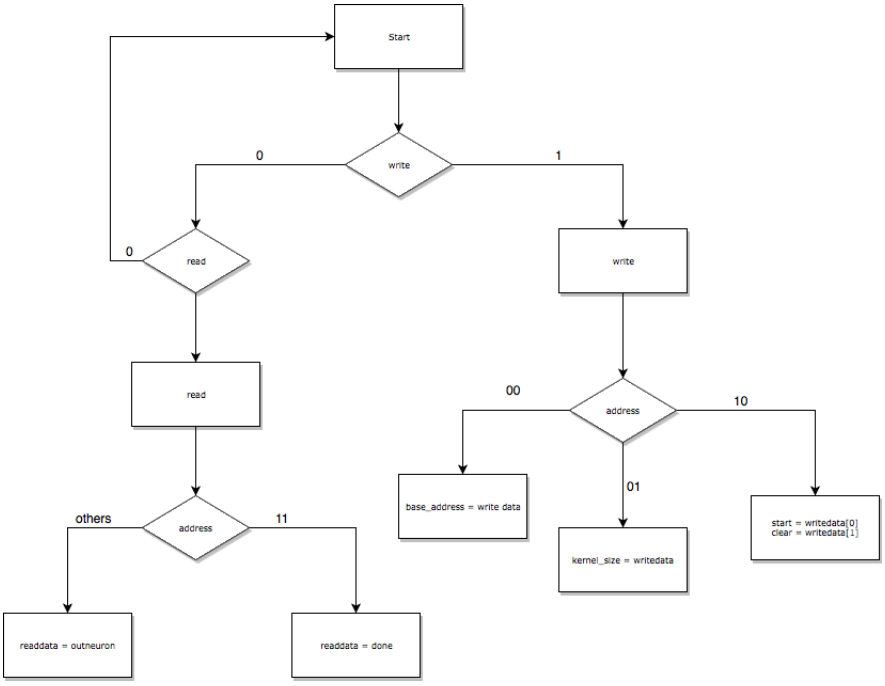


Figura 46: Diagrama ASM do componente *Neuron Control*

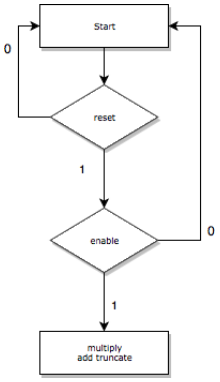


Figura 47: Diagrama ASM do componente *Multiply Accumulate*

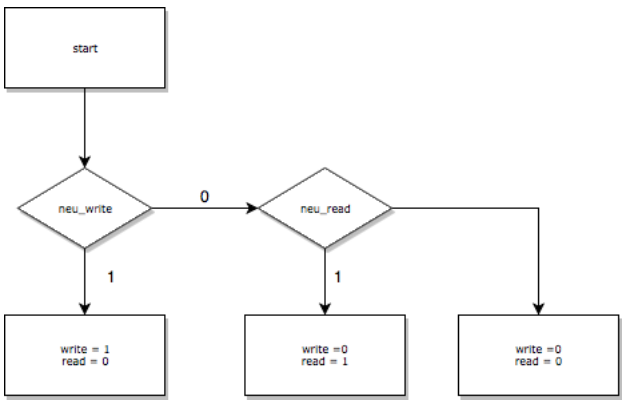


Figura 48: Diagrama ASM do componente f2_ram

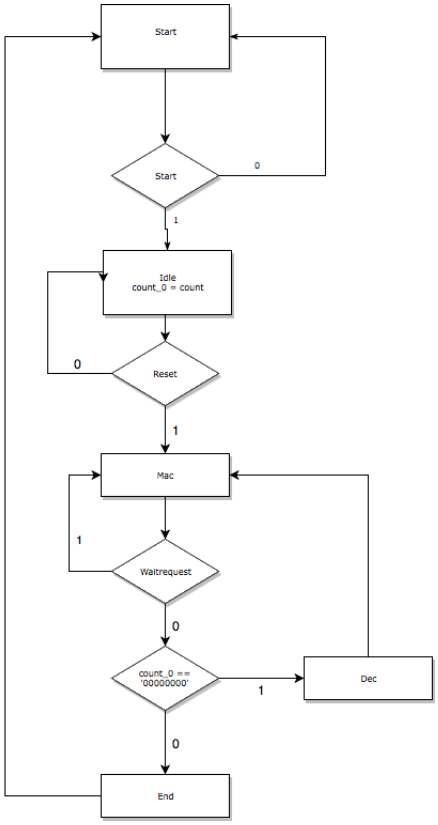


Figura 49: Diagrama ASM da Unidade de Controle

para coordenar os blocos de geração de endereços de leitura de memória RAM e de multiplicação e soma, *Mutliply Accumulate*, garantindo a sincronia e que as operações sejam feitas com os valores corretos da memória.

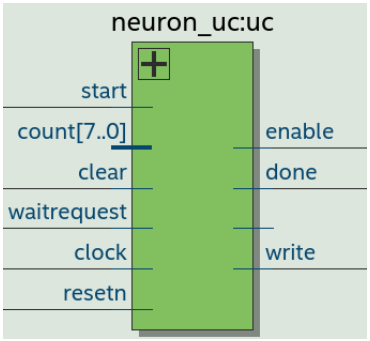


Figura 50: Bloco unidade de controle neurônio

## 4.4 Comunicação ARM com FPGA

A comunicação utiliza pontes de acesso LWH2F da figura 51, *Light Weight HPS to FPGA bridge*.

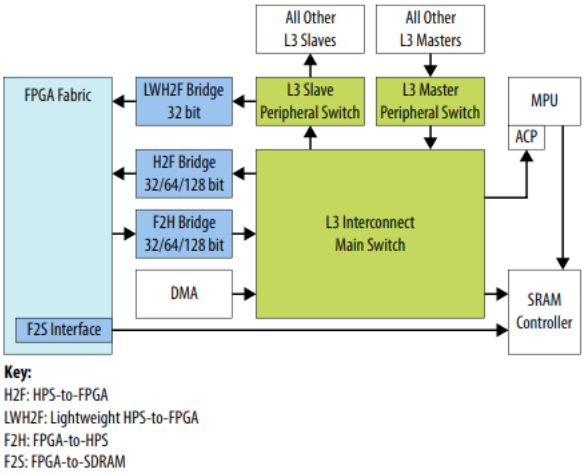


Figura 51: Diagrama de blocos das pontes entre HPS e FPGA

Por meio da interface Avalon, que funciona como intermediadora entre componentes, como memória e registradores, da FPGA e do ARM criamos um controlador que gerencia, usando método *Master-Slave*, a relação entre o sistema linux e a implementação em FPGA.

Portanto, o novo sistema atua de forma análoga a um processador dedicado a função de um neurônio, sendo a unidade de controle um intermédio entre o sistema em Linux que configura a rede e a FPGA que executa os processos neuronais.

A interface atua de acordo com a figura 52. São alocados blocos de memória que compartilham *inputs* e *outputs* entre FPGA e HPS, a escrite e leitura das memórias são coordenadas pelo bloco de controle Avalon que envia sinais de controle e recebe sinais de *status* dos processos.

No projeto utiliza-se as *bridges: HPS-to-FPGA* e *LightWeight HPS-to-FPGA* para comunicar paralelamente informações de input e peso com memórias instanciadas. As pontes de comunicação atuam como endereços compartilhados entre o ARM e a FPGA, portanto, as instanciações de endereçamento em software serão tratadas com mais detalhes na seção tratando de implementação do driver em software.

### 4.4.1 Distribuição Linux

Na parte HPS com processador baseado em ARM foi adicionado uma distribuição Linux baseada no projeto Yocto, Angstrom. As distribuições baseadas no projeto Yocto são facilmente modificadas para se adaptar às necessidades do sistema embarcado, ou seja, funciona como um ecossistema de criação de novas distros.

Para programar a FPGA e o ARM deve-se criar uma distribuição contendo os arquivos

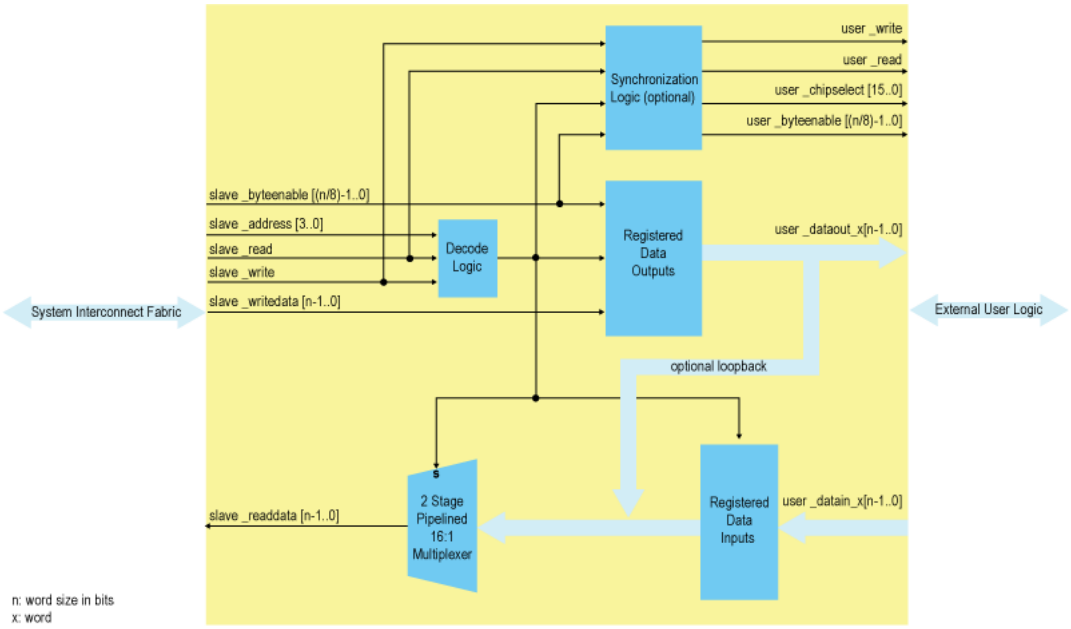


Figura 52: Diagrama de blocos da interface *Avalon*

de descrição de hardware da FPGA e do Linux, portanto, é necessário construir uma imagem do sistema a partir de novos arquivos binários gerados pelo *design*. Utilizando os arquivos gerados, como mostrados na 53, pode-se construir as dependências de uma distribuição o *U-boot* e o *Kernel*.

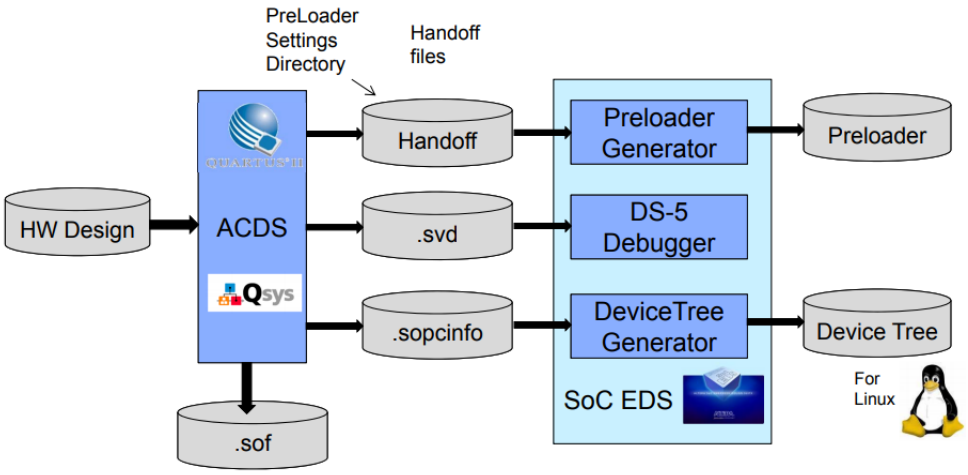


Figura 53: Diagrama para compilação de linux

Finalmente, gera-se uma imagem do sistema e, configurando a placa para entrar em modo configuração de boot, inicia-se o boot manualmente, de forma a carregar no ARM e na FPGA todos os arquivos gerados.

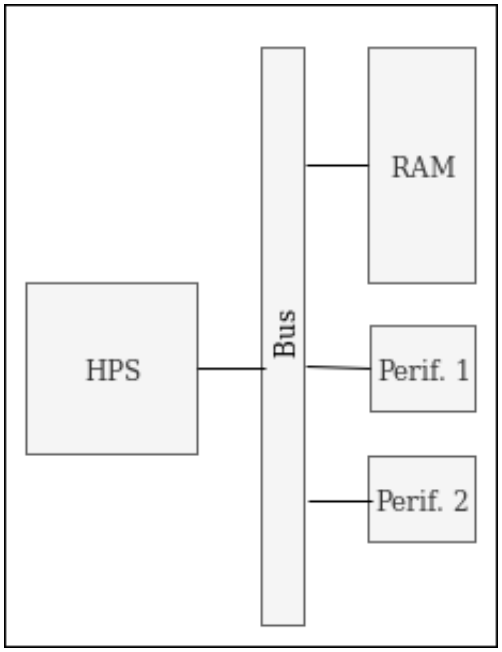


Figura 54: Memory-Mapped I/O

## 4.5 Driver - Interface Software

Com o design de Hardware implementado e carregado na FPGA é necessária uma interface em Software, produzida em linguagem C, para comunicar com os endereços de memória especificados pelas pontes *HPS-to-FPGA* e *LightWeight HPS-to-FPGA*.

### 4.5.1 *Memory-Mapped I/O*

Para que o software controle os periféricos, o processador deve ter um meio de se comunicar com eles. Esse meio de comunicação deve também ser flexível o bastante para que não seja necessário fazer mudanças na arquitetura da CPU para acomodar novos periféricos. Por isso o método que os processadores modernos utilizam é o *memory-mapped IO*, no qual um *bus* é responsável por interconectar os diversos componentes, e porções do espaço de endereçamento físico é a RAM e os demais periféricos ao processador.

### 4.5.2 Geração de *Header*

O cabeçalho, contendo informações de endereços de memória atribuidos no design de hardware, pode ser gerado utilizando a ferramenta *SOPC Builder - System-On-a-Programmable-Chip*, que com um comando cria definições de localização

# 5 CONCLUSÕES

## 5.1 Comparação de Resultados

Comparação Hardware vs Software

Para efeito de análise de desempenho de software foi escrito um código C que realiza as operações de uma convolução. Foi medido o tempo de execução dessas operações num processador Intel Core i5 (2,3 GHz), no processador baseado em ARM da placa SoC e no FPGA Cyclone V. Essa comparação pode ser vista na tabela 6.

Nessa comparação é possível observar, como esperado, que o desempenho num i5 foi melhor que no ARM, e

| Plataforma | Tempo (s) |
|---|---|
| i5 | 0.006078 |
| ARM | 0.015889 |
| FPGA | 0.000484 |

Tabela 6: Tabela Comparativa de tempos necessários para processamento de uma convolução

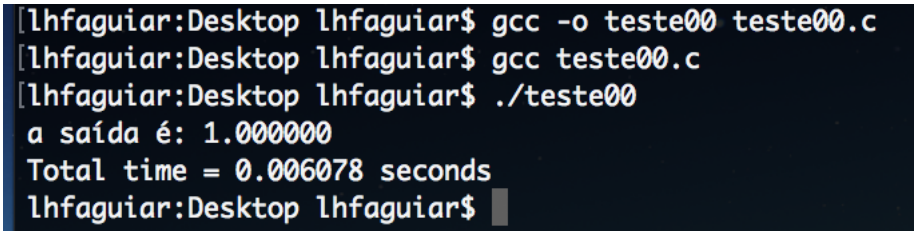


Figura 55: Tempo gasto pelo script num i5

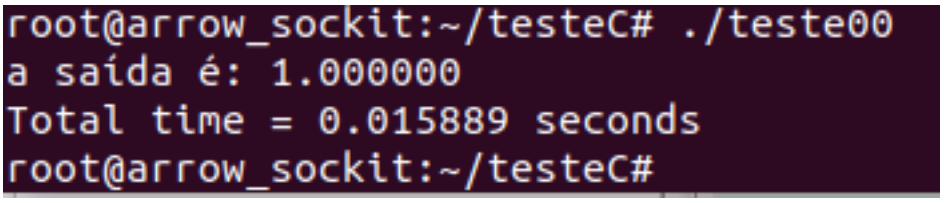


Figura 56: Tempo gasto pelo script no ARM

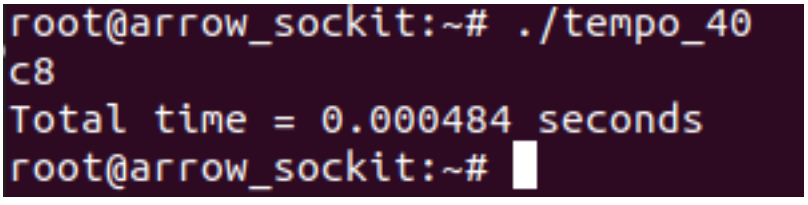


Figura 57: Tempo gasto pelo script no FPGA

# REFERÊNCIAS

1 ADAMS, D. *The Ultimate Hitchhiker's Guide to the Galaxy: Five Novels in One Outrageous Volume*. [S.l.]: Random House Publishing Group, 2010. (Hitchhiker's Guide to the Galaxy). ISBN 9780307498465.

2 CHOY, C. B. et al. 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction. In: *Proceedings of the European Conference on Computer Vision ({ECCV})*. [S.l.: s.n.], 2016.

3 OSIRIX. *Osirix History*. 2018. Disponível em: ⟨http://www.osirix-viewer.com/about/story/⟩.

4 BEAM, C.; CAO, Z.; HOLDER, L. E. Experimental Comparison of Three Reconstruction Algorithms. p. 1217–1221.

5 CANOMA. *Canoma Project*. 2017. Disponível em: ⟨http://www.canoma.com⟩.

6 SANDIA LLC (NTESS), K. I. National Technology & Engineering Solutions of. 2017. Disponível em: ⟨https://www.paraview.org/overview/⟩.

7 FELDKAMP, L. A.; DAVIS, L. C.; KRESS, J. W. Practical cone-beam algorithm. *J. Opt. Soc. Am. A*, v. 1, n. 6, p. 612–619, 1984. ISSN 1084-7529. Disponível em: ⟨http://josaa.osa.org/abstract.cfm?URI=josaa-1-6-612⟩.

8 TAM, K. C. Exact Image Reconstruction in Cone Beam 3D CT. *Review of Progress in Quantitative Nondestructive Evaluation*, v. 1, n. 6, p. 657–664, 1995. ISSN 1084-7529. Disponível em: ⟨https://www.osapublishing.org/abstract.cfm?URI=josaa-1-6-612\%5Cnhttp://link.springer.com/10.1007/978-1-4615-1987-4_81⟩.

9 GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M. Generative Adversarial Networks. *arXiv preprint arXiv: ...*, 2014. ISSN 10495258.

10 WU, J. et al. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In: *Advances in Neural Information Processing Systems*. [s.n.], 2016. p. 82–90. Disponível em: ⟨http://marrnet.csail.mit.edu/⟩.

11 CHANG, A. X. et al. ShapeNet: An Information-Rich 3D Model Repository. *CoRR*, abs/1512.03012, 2015. Disponível em: ⟨http://arxiv.org/abs/1512.03012⟩.

12 HUANG, H.; KALOGERAKIS, E.; MARLIN, B. Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces. *Eurographics Symposium on Geometry Processing*, v. 34, n. 5, p. 25–38, 2015. ISSN 17278384. Disponível em: ⟨https://people.cs.umass.edu/~hbhuang/publications/bsm/bsm.pdf⟩.

13 LORENSEN, W. E.; CLINE, H. E. Marching cubes: A high resolution 3D surface construction algorithm. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*, v. 21, n. 4, p. 163–169, 1987. ISSN 00978930. Disponível em: ⟨http://portal.acm.org/citation.cfm?doid=37401.37422⟩.

14 MUKHERJEET, S. et al. CUDA and OpenCL implementations of 3D CT reconstruction for biomedical imaging. *2012 IEEE Conference on High Performance Extreme Computing, HPEC 2012*, 2012.

15 DEVELOPERS, F. *FFmpeg*. 2016. Disponível em: ⟨http://ffmpeg.org/⟩.

16 OSIRIX. *Osirix Technical Specifications*. 2018. Disponível em: ⟨http://www.osirix-viewer.com/resources/technical-sheet/⟩.

17 MIN, P. *binvox*. 2004 – 2017. `http://www.patrickmin.com/binvox` or `https://www.google.com/search?q=binvox`. Accessed: yyyy-mm-dd.

18 LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. ISSN 00189219.

19 HOCHREITER, J. Untersuchungen zu dynamischen neuronalen Netzen. *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, p. 1–71, 1991. Disponível em: ⟨http://www.bioinf.jku.at/publications/older/3804_2.pdf\%0Ahttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Untersuchungen+zu+dynamischen+neuronalen+Netzen\#0⟩.

20 HOCHREITER, S. Long Short-Term Memory. v. 1780, p. 1735–1780, 1997.

21 CHO, K. et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 2014. ISSN 09205691. Disponível em: ⟨http://arxiv.org/abs/1406.1078⟩.

22 CHOY, C. B. et al. 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction. v. 1, p. 1–17, 2016. ISSN 16113349. Disponível em: ⟨http://arxiv.org/abs/1604.00449⟩.

23 PROJECT, Y. *Yocto Project*. 2018. Disponível em: ⟨https://www.yoctoproject.org/⟩.

24 GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011. ISSN 15324435.