

Universidade Federal do ABC

Carolina Zambelli Kamada

Simulações de Redes Neurais Pulsadas em FPGA utilizando
padrão de ponto flutuante Half-Precision

Santo André

2018

Título: Simulações de Redes Neurais Pulsadas em FPGA utilizando padrão de ponto flutuante Half-Precision

Autor: Carolina Zambelli Kamada

Orientador: Prof. Dr. João Ranhel

Trabalho de Graduação apresentado como requisito para a obtenção do título de Bacharel em Engenharia de Informação pela Universidade Federal do ABC.

Banca Examinadora:

Prof Dr. Amaury Krueel Budri

Universidade Federal do ABC

Prof Dr. Helói Francisco Gentil Genari

Universidade Federal do ABC

Santo André

2018

Resumo

As simulações de redes neurais pulsadas exigem grande demanda computacional. Com o intuito de atender tal demanda, desenvolvemos em FPGA um processador dedicado para computar um modelo específico de neurônio, o modelo de Izhikevich. Além disso, para otimizar recursos de hardware, investigamos o uso do formato de representação numérica de ponto flutuante half-precision (padrão IEEE 754-2008) aplicado ao processador dedicado. Half-precision apresenta significativa economia de memória e banda de transferência pois utiliza apenas 16 bits. Contudo, o padrão IEEE 754-2008 requer que toda operação aritmética seja realizada no formato single-precision de 32 bits. Assim, foi desenvolvido um conversor para transformar o formato single-precision em half-precision para o armazenamento em memória e depois convertido para single-precision novamente para efetuar as operações aritméticas.

Os processadores dedicados para executar o modelo de Izhikevich e o conversor foram testados comparando seus resultados com o mesmo processo desenvolvido em software que utiliza single-precision 32 bits. Também foram usados três métodos canônicos em neurociência para assegurar que as conversões não causaram alterações relevantes para a simulação, tanto de um neurônio, quanto para uma rede.

Após a análise foi possível concluir que é possível implementar o modelo de Izhikevich para redes neurais pulsadas utilizando formato half-precision para armazenamento e transferência de dados. Nosso trabalho comprovou que é possível utilizar a representação half-precision para armazenar dados de redes neurais pulsadas não apenas com modelos Izhikevich mas também com o modelo Leak Integrate & Fire (LI&F), sem prejuízo na representação e com economia de memória, largura de banda, e energia.

Abstract

The Spiking Neural Network simulations demand high computational effort. In order to meet this demand, we developed in FPGA a dedicated processor to compute a specific neuron model, the Izhikevich model. In addition, to optimize hardware resources, we also investigated the use of half-precision floating-point (IEEE 754-2008 standard) applied to the dedicated processor. Half-precision presents significant memory savings and bandwidth because it uses only 16 bits. However, the IEEE 754-2008 standard requires that all arithmetic operations be performed in 32-bit single-precision format. Thus, a converter was developed to transform the single-precision format into half-precision for storage in memory and then converted to single-precision again to perform the arithmetic operations.

The dedicated processors to execute the Izhikevich model and the converter were tested by comparing their results with the same process developed in software using 32-bit single-precision. Three canonical neuroscience methods were also used to ensure that the conversions did not cause relevant changes to the simulation of either a neuron or a network.

After the analysis it was possible to conclude that it is possible to implement the Izhikevich model to test pulsed neural using half-precision format for storage and transfer of data. Our work proved that it is possible to use the half-precision representation to store pulsed neural network data not only with Izhikevich models but also with the Leak Integrate & Fire (LI & F) model, without loss of representation and memory economy, bandwidth, and energy.

Sumário

| | |
|--|----|
| 1. Introdução | 1 |
| 2. FPGA..... | 3 |
| 2.1 Dispositivos Lógico Programáveis | 3 |
| 2.2 Fluxo de Desenvolvimento | 4 |
| 2.3 Ponto Flutuante padrão IEEE-754..... | 5 |
| 2.3.1 O Padrão IEEE-754 para 32 bits | 6 |
| 2.3.2 O Padrão IEEE-754 para 16 bits | 7 |
| 3. Redes Neurais Pulsadas..... | 9 |
| 3.1 Dinâmica Neural | 9 |
| 2.4 O Modelo de Hodgkin-Huxley..... | 10 |
| 2.5 O Modelo de Izhikevich..... | 12 |
| 4. Metodologia | 13 |
| 4.1. Cálculo do Neurônio segundo o modelo de Izhikevich | 14 |
| 4.3. Cálculo de um Neurônio segundo o modelo de Izhikevich com conversão para half-precision | 17 |
| 4.4. Análise da curva FxI | 19 |
| 4.5. Análise do Comportamento de uma Rede..... | 19 |
| 5. Resultados | 20 |
| 5.1. Cálculo do Neurônio de Izhikevich..... | 20 |
| 5.2. Conversores de fp32 para fp16 e de fp16 para fp32..... | 21 |
| 5.3. Cálculo do Neurônio de Izhikevich com conversão..... | 21 |
| 5.4. Análise da curva FxI | 23 |
| 5.5. Análise do Comportamento de uma rede | 24 |
| 5. Conclusões | 25 |

1. Introdução

A aplicação de redes neurais pulsadas em engenharia tem sido limitada em grande parte devido aos custos computacionais. Este trabalho mostra a possibilidade de reduzir tal custo em simulações de redes neurais pulsadas de larga escala e em tempo real. Reduzindo o número de bits necessário para armazenar parâmetros e variáveis da rede é possível reduzir o espaço em memória, banda e frequência de *clock*, resultando menor consumo de energia, sendo de grande relevância para simulações com sistemas embarcados.

Nas redes neurais artificiais, como as utilizadas em deep learning, o grande volume de dados a ser processado e a necessidade de modelos complexos para classificar imagens, textos e áudios, aumentam cada vez mais o custo computacional. Como exemplo, na Figura 1 podemos observar a evolução da demanda de memória para redes neurais artificiais usadas em *deep learning*. Por outro lado, na Figura 2 observamos uma projeção para o consumo de banda para a mesma aplicação. O círculo azul na figura representa a performance das implementações atuais.

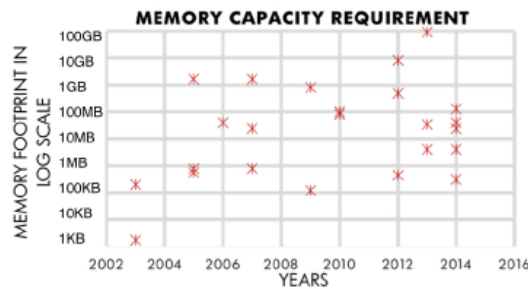


Figura 1: Evolução do consumo de memória utilizada para redes neurais artificiais [7]

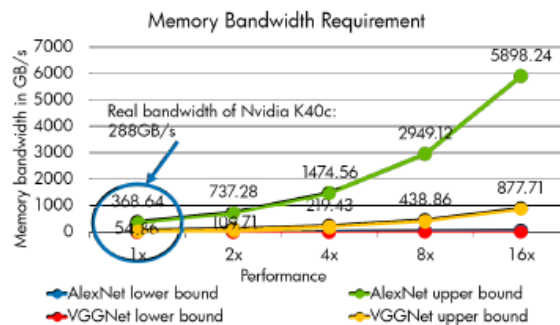


Figura 2: Previsão de consumo de banda para modelos típicos de Deep Learning, AlexNet e VGCNet [7]

Nos modelos de redes neurais pulsadas, chamamos de pulso a informação booleana transmitida de um neurônio para o outro. Neurônios são calculados de acordo com equações matemáticas que mimetizam seus comportamentos, reproduzindo fenômenos que observamos nos tecidos nervosos biológicos. Cada “modelo” ou “tipo” de neurônio é caracterizado por uma série de variáveis que representam aspectos biofísicos ou comportamentais dos neurônios naturais. As variáveis comportamentais estão associadas à dinâmica com que os pulsos ocorrem e pela inter-relação com os demais neurônios. As variáveis biofísicas refletem aberturas de canais iônicos, concentrações de

neurotransmissores, além de outros aspectos que são geralmente simplificados em simulações de redes para fins de aplicações em engenharia. Contudo, toda rede neural possui uma estrutura de interligação entre neurônios, chamada “topologia”, que representa a força de ligação entre os neurônios. Computar a evolução da interação temporal entre todos os neurônios é uma tarefa computacionalmente custosa, e quanto mais variáveis biológicas levamos em conta, mais custosa essa tarefa se torna.

Existem diversas técnicas para a otimização de simulação em redes neurais, como o processamento paralelo descrito por D. Zhaoxia em [7]. No entanto, tais técnicas apesar da otimização ainda dependem dos recursos de hardware disponíveis.

Para atender a demanda por recursos de hardware e otimizar simulações em redes neurais artificiais tradicionais, D. Zhaoxia e colegas [7] propõem a redução da precisão numérica para 8 bits no padrão de ponto flutuante, obtendo apenas 1% de perda em acurácia. Já para redes neurais pulsadas, alguns projetos como o apresentado por D. Pani em [3] e D. Thomas em [8], desenvolveram hardwares dedicados em FPGA com intuito de atender a demanda computacional para tais simulações, em larga escala e em tempo real. Na mesma linha, o presente trabalho propõe unir as abordagens de [7], [3] e [8], projetando um hardware dedicado para o cálculo deste modelo em FPGA e investigar o desempenho do mesmo modelo com a precisão numérica reduzida de 32 bits para 16 bits para armazenamento e transferência de dados.

Os estudos em redes neurais pulsadas encontram-se em sua fase inicial, sendo assim, ainda são poucas as aplicações já desenvolvidas nesta área. Um exemplo é a aplicação das redes neurais pulsadas para reconhecimento de caracteres encontrado em [17]. Para o avanço desta área é de extrema importância que seja possível superar as barreiras impostas pelo alto custo computacional destes modelos, tornando o trabalho aqui descrito muito relevante.

2. FPGA

2.1 Dispositivos Lógico Programáveis ¹

A origem da FPGA está atrelada ao desenvolvimento dos circuitos de lógica programável (PLDs), que são componentes eletrônicos utilizados para construir sistemas digitais. Estes nasceram da necessidade da indústria em criar circuitos integrados (CIs) cuja funcionalidade pudesse ser reconfigurada. Essa flexibilidade tem como consequência o melhor uso do espaço das placas de circuito impresso (PCBs), menor consumo de energia, maior facilidade de manutenção, aumento da complexidade na funcionalidade, a reprogramação do dispositivo com o circuito instalado na placa, além de outras vantagens.

Assim, a indústria de componentes eletrônicos passou a criar CIs que não tivessem funções pré-definidas, como os CIs discretos, e sim, que fossem circuitos programáveis ou reprogramáveis. A ideia é que qualquer função lógica pode ser obtida a partir de portas AND, OR, NOT interligadas internamente.

Os primeiros arranjos lógicos programáveis implementaram dois níveis de planos lógicos AND-OR. Uma evolução desses CIs trazia um plano AND programável e um plano OR pré-configurado, aumentando a flexibilidade. Este último dispositivo ficou conhecido pelo acrônimo PAL (Programmable Logic Arrays). Esse arranjo torna os componentes mais rápidos, menores e mais baratos, e o projetista deveria optar apenas pelo tipo de arranjo de saída que desejava.

Outro avanço importante foram introduzidos pela criação dos Elementos Lógicos. Tais estruturas compõem a nova base de dispositivos programáveis, mais genéricas e versáteis que as tradicionais matrizes de AND-OR dos arranjos lógicos programáveis. O principal conceito introduzido neste tipo de tecnologia é a LUT (Look-up Table). Uma LUT (tabela de consulta) é composta de um multiplexador (MUX) associado a um ou mais registradores (basicamente, flip-flops do tipo “D”).

A primeira FPGA, baseada em memória estática, foi proposta por Wahlstrom em 1967. Sua arquitetura permite tanto a lógica configurável quanto a interconexão, utilizando um fluxo de bits configuráveis.

A sigla FPGA (Field Programmable Gate Array) se referem a uma família de CIs reprogramáveis que usam conjuntos de elementos lógicos (LEs) como unidades básicas para construção da lógica configurável.

A arquitetura de uma FPGA consiste de três macrocomponentes:

- Blocos Lógicos Programáveis, que implementam funções lógicas;
- Interconectores Programáveis, que implementam funções lógicas;
- Bloco de I/O, que são usados para fazer as conexões off-chip.

¹ Esta seção é uma adaptação de *FPGA Architecture: Survey and Challenges* disponível em [13]

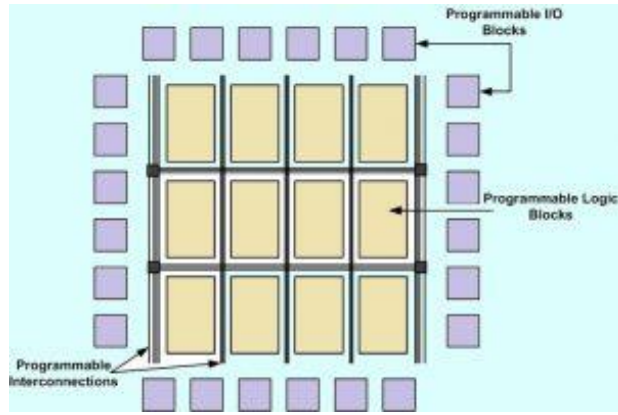


Figura 3: Estrutura Básica de uma FPGA [13]

2.2 Fluxo de Desenvolvimento ²

O desenho assistido por computador CAD (Computer Aided Design, em inglês) é um software que implementa facilmente a lógica do circuito desejado usando um dispositivo lógico programável, assim como a FPGA. Um fluxo típico CAD de uma FPGA pode ser ilustrado na figura 4:

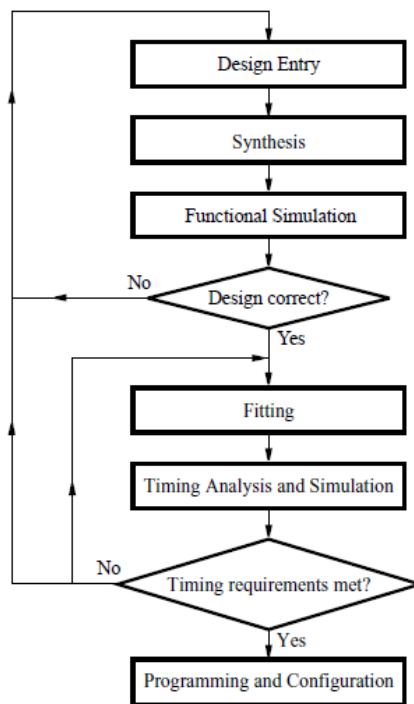


Figura 4: Fluxo CAD típico [14]

² Esta seção é uma adaptação de *Quartus II Introduction Using Schematic Design* disponível em [14]

Um fluxo CAD envolve os seguintes passos:

1. Design de Entrada – É a especificação do circuito desejado por um diagrama esquemático ou por meio de uma linguagem descritiva de hardware como o VHDL ou Verilog.
2. Síntese – O design desejado é sintetizado em um circuito que consiste de elementos lógicos (LEs), fornecidos pelo chip FPGA.
3. Simulação Funcional – A síntese do circuito é testada para verificar a sua correlação funcional. Esta etapa não considera nenhuma falha relacionada ao tempo.
4. Fitting – A ferramenta CAD Fitter determina e determina a colocação dos LEs definidos na netlist nos LEs da FPGA real; ele também escolhe fios de roteamento no chip para fazer as conexões necessárias entre os LEs específicos.
5. Análise do Tempo – A propagação do atraso ao longo dos passos no circuito fitted é analisada para providenciar uma expectativa de performance do circuito.
6. Simulação de Tempo – O circuito fitted é testado para verificar tanto a correção funcional quanto a temporal.
7. Programação e Configuração – O circuito desejado é implementado na FPGA física.

Neste trabalho foram executados os passos de 1 a 6.

2.3 Ponto Flutuante padrão IEEE-754

Muitos sistemas complexos em comunicações, militares, médicos e outras aplicações são primeiro simulados ou modelados usando processamento de dados de ponto flutuante, usando o software C ou MATLAB. No entanto, a implementação final em hardware é quase sempre executada usando aritmética de ponto fixo ou inteiro. Para isso, os algoritmos são cuidadosamente mapeados em um intervalo dinâmico limitado e redimensionados através de cada função no “*datapath*”³. Isso requer várias etapas de arredondamento e saturação e, se feitas de maneira inadequada, podem afetar adversamente o desempenho do algoritmo. Projetos dessa natureza geralmente requerem verificação extensiva durante a integração dos componentes para garantir que a operação do sistema corresponda aos resultados da simulação e não tenha sido comprometida indevidamente [12].

³ Literalmente, datapath pode ser traduzido como “caminho para os dados”. Trata-se de um conjunto de fios que transportam dados entre registradores, memória e unidades de lógica e/ou aritmética.

Por outro lado, fabricantes de FPGA oferecem *IPs*⁴ otimizados que implementam circuitos que executam operações de pontos flutuantes em seus CIs. Tais *IPs*, implementam em hardware os circuitos que executam operações de adição, subtração, multiplicação, divisão, multiplicação de matrizes entre outros. Utilizamos os *IPs* de circuitos de adição, subtração e multiplicação em ponto-flutuante no desenvolvimento deste trabalho.

A seguir, é dada uma breve descrição do padrão de ponto flutuante IEEE-754.

2.3.1 O Padrão IEEE-754 para 32 bits

Em 1985 o IEEE padronizou a representação para 32 e 64 bits. A padronização define a quantidade de bits alocados para a mantissa e para o expoente, assim como as definições de zero, infinito e os casos de números indeterminados. O padrão para 32 bits é chamado de single precision.

Para 32 bits, ficou-se definido da seguinte forma;

| | | |
|------------|-------------------|-------------------------|
| b0 | b1 b2 b3 b8 | b9 b10 b30 b31 |
| Sgn: 1 Bit | Expoente: 8 Bits | Significativos: 23 Bits |

De acordo com Rajaraman [6], o padrão utiliza uma representação deslocada ou polarizada dada pelo uso do “*bias*”⁵. Este é um valor subtraído do expoente (sem sinal) armazenado para determinar o valor real deste campo. Essa convenção foi adotada para facilitar a comparação entre dois números, uma vez que expoentes negativos à primeira vista se parecem com valores de maior magnitude (sem sinal).

O padrão ainda classifica os números como normal e subnormal. O primeiro se refere aqueles que possuem ao menos um bit um, e o segundo os que possuem todos os bits zerados e, portanto, são número muito próximos do zero.

A tabela 1 mostra as representações adotadas pelo padrão

Tabela 1: Representações para o padrão fp32

| | | | | |
|-----------------------|--------------------------|-----|----------|--------------------------|
| Zero | 0 | 0/1 | 00000000 | 000000000000000000000000 |
| ± Infinito | $\pm \infty$ | 0/1 | 11111111 | 000000000000000000000000 |
| Maior Normal Positivo | $3,404 \times 10^{38}$ | 0 | 11111110 | 111111111111111111111111 |
| Menor Normal Positivo | $1,1755 \times 10^{-38}$ | 0 | 00000001 | 000000000000000000000000 |
| Subnormal | 2^{-29} | 0 | 00000000 | 000000000000000000000001 |

⁴ *IP*: **propriedade intelectual**, ou seja, um circuito descrito em alguma linguagem, funcionalmente testado e aprovado por quem o produziu, disponibilizado para terceiros livremente ou por meio de pagamento. Utilizamos as *FPGAs* da Intel, mas os *IPs* foram criados pela equipe da antecessora Altera, assim, preferi manter a referência a quem originalmente gerou as propriedades intelectuais.

⁵ *Bias*: ou “valor fixo” que desloca o expoente do valor zero para metade do valor possível de representação numérica dos “n” bits, o que evita que se utilize um bit específico para sinal no expoente.

Seria natural pensar que o regime subnormal se inicia a partir da menor representação de um número em regime normal. No entanto, existe um gap de representação entre estes dois regimes, sendo o maior valor subnormal dado por $0,999999988 \times 2^{-126}$.

Além disso, qualquer número que exceda o maior número normal positivo será considerado um *overflow*; e qualquer número que seja menor que o valor subnormal será considerado um *underflow*; ou seja, ambos valores impossíveis de serem representados no formato floating single-precision.

2.3.2 O Padrão IEEE-754 para 16 bits

Em 2008, o padrão foi revisto e a ele foi incorporado as representações para 16 bits e 128 bits. A razão para inclusão do formato de 16-bits (*half-precision*, em inglês) é que este formato era utilizado pela indústria de videogame para armazenar valores que não exigem grande precisão, como por exemplo valores de cores e texturas em jogos. O formato é vantajoso porque usa metade da quantidade de bits comparado ao *single-precision*, o que diminui o espaço de memória; e, mais importante, a banda de transferência para jogos pela internet ou em ambientes de intranet. Progressivamente, outras áreas começaram a utilizar o formato para redução de recursos de memória e banda, quando possível.

Para 16 bits, ficou definido a seguinte representação;

| | | |
|------------|------------------|-------------------------|
| b0 | b1 b2 b3 b4 b5 | b6 b7..... b15 |
| Sgn: 1 Bit | Expoente: 5 Bits | Significativos: 10 Bits |

Nesta representação, o “*bias*” assume um valor igual a 15, sendo assim, o expoente é limitado ao intervalo de -14 a 15 . Dessa forma teremos;

Tabela 2: Representações para o padrão fp16

| | | |
|-----------------------|---------------------|-----------------------|
| Maior Normal Positivo | $1 + (1 - 2^{-11})$ | 65504 |
| Menor Normal Positivo | 2^{-14} | $0,61 \times 10^{-4}$ |
| Subnormal | 2^{-24} | $5,96 \times 10^{-8}$ |

A indústria de videogames, computação gráfica e outros setores têm usado o padrão de 16 bits para gravar e transmitir dados. A tabela 3 mostra a capacidade de representação para este formato. Ela também pode ser confrontada com a tabela 4, que mostra a capacidade de representação para o padrão de ponto fixo também em 16 bits.

Tabela 3: Capacidade de Representação para fp16

| formato | intervalo | resolução | |
|-----------|--|-----------|-----------------------|
| subnormal | $\pm \{5.96 \times 10^{-8}, 6.09 \times 10^{-5}\}$ | 2^{-24} | 5.96×10^{-8} |
| normal | $\pm \{0.5, 0.9995\}$ | 2^{-11} | 4.88×10^{-4} |
| normal | $\pm \{4, 7.996\}$ | 2^{-8} | 3.90×10^{-3} |
| normal | $\pm \{16, 31.984\}$ | 2^{-6} | 1.56×10^{-2} |
| normal | $\pm \{32, 63.968\}$ | 2^{-5} | 3.12×10^{-2} |
| normal | $\pm \{64, 127.937\}$ | 2^{-4} | 6.25×10^{-2} |
| normal | $\pm \{128, 255.875\}$ | 2^{-3} | 1.25×10^{-1} |

Tabela 4: Capacidade de Representação para ponto fixo

| formato | BITS | | | binário | | decimal | |
|---------|------|------------|-------------|------------|-------------|------------|----------------------|
| | +/- | <i>int</i> | <i>frac</i> | <i>int</i> | <i>frac</i> | <i>int</i> | <i>resolução</i> |
| Q1.15 | 1 | 0 | 15 | 0 | 2^{-15} | 0 | 3×10^{-5} |
| Q4.12 | 1 | 3 | 12 | 2^3-1 | 2^{-12} | 7 | 2.4×10^{-4} |
| Q5.11 | 1 | 4 | 11 | 2^4-1 | 2^{-11} | 15 | 2^{-11} |
| Q6.10 | 1 | 5 | 10 | 2^5-1 | 2^{-10} | 31 | 2^{-10} |
| Q7.9 | 1 | 6 | 9 | 2^6-1 | 2^{-9} | 63 | 2^{-9} |
| Q8.8 | 1 | 7 | 8 | 2^7-1 | 2^{-8} | 127 | 3.9×10^{-3} |

3. Redes Neurais Pulsadas

Esta seção faz uma breve introdução aos conceitos básicos das redes neurais pulsadas, passando pelo primeiro modelo biologicamente plausível, o modelo de Hodgkin-Huxley, e finaliza abordado o modelo de Izhikevich que é o modelo central deste trabalho.

3.1 Dinâmica Neural

Segundo a descrição encontrada em Bears et al [15], um neurônio típico é composto por três partes com funcionalidades distintas; detritos, corpo celular ou soma e axônio, sendo:

- ✓ Dendritos: pode ser pensado como o dispositivo de entrada. Estes são responsáveis por coletar os sinais vindos de outros neurônios e transmitir para o corpo celular;
- ✓ Corpo Celular: é a central de processamento. Ele “computa” todo o sinal recebido e caso este ultrapasse um certo valor limite ele gera uma resposta;
- ✓ Axônio: é responsável por transmitir a resposta gerada para outros neurônios.

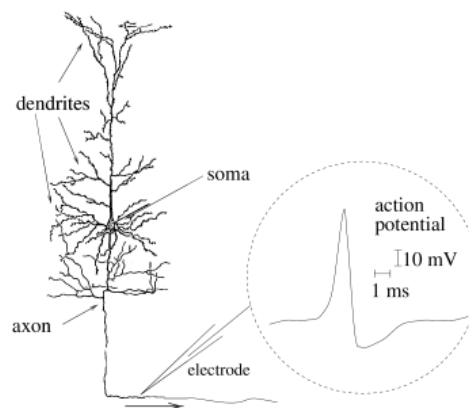


Figura 5: Representação da estrutura básica de um neurônio e identificação da localização do potencial de ação [2]

Um outro conceito importante é a sinapse. Ela é descrita por W. Gerstner [2] como a junção de dois neurônios. Considere que o neurônio envie informações através de uma sinapse, é uma terminologia comum nos referirmos ao neurônio que está enviando a informação como neurônio pré-sináptico e o que está recebendo a informação como neurônio pós-sináptico.

A informação transmitida de um neurônio ao outro é chamada de pulso, que carrega apenas a informação da ocorrência do disparo ou não. Isso porque todo pulso possui a mesma duração e amplitude. Esse consiste de um impulso elétrico e pode ser observado experimentalmente posicionando um eletrodo no corpo celular.

Os pulsos recebidos por um neurônio pós-sináptico causam a despolarização de sua membrana, ou seja, canais de sódio são abertos permitindo o influxo de sódio. Caso a despolarização atinja uma carga específica, diremos que o limiar de disparo foi atingido e ocorrerá o potencial de ação, ou seja, o neurônio emitirá um pulso com duração média

entre 1 e 2 milissegundos (ms). Neste momento, o neurônio fica positivo com relação ao ambiente extracelular. Este processo é comumente chamado de dinâmica neural.

Quando o pulso recebido facilita a abertura dos canais de sódio, eles são chamados de impulso excitatórios e inibitórios caso contrário. A somatória dos pulsos que chegam a um neurônio é chamada de potencial pós-sináptico inibitório/excitatório (IPSP/EPSP ou inhibitory/excitatory postsynaptic potential, em inglês).

No entanto, o quanto um pulso é capaz de provocar tal “perturbação” nos canais de sódio do neurônio pós-sináptico está majoritariamente relacionado a dois fatores. O primeiro é a força da sinapse. Se a sinapse for fraca o neurônio pós-sináptico será fracamente despolarizado no caso de um impulso excitatório, e será amplamente polarizado no caso de uma sinapse forte. O segundo fator é o tempo em que um determinado pulso, ou um conjunto deles, atingem um neurônio. Estes dois fatores causam o surgimento de alguns grupos de neurônios com ação conjunta, ou seja, que disparam de acordo com alguma relação. Em 1949 foi proposto por Hebb em [16] que estes fatores são responsáveis pelo surgimento das assembleias celulares.

Na literatura encontramos que o entendimento das assembleias neurais é um fator de extrema importância para a compreensão de como o cérebro representa, memoriza e processa a informação (ver síntese em [4]).

2.4 O Modelo de Hodgkin-Huxley

Os primeiros cientistas a investigar o comportamento biofísico de um neurônio foram Hodgkin e Huxley[2]. Numa série de experimentos com lulas gigantes eles mediram com sucesso a dinâmica de abertura e fechamento dos canais de sódio e potássio e a descreveram em termos de equações diferenciais. O artigo publicado em 1952 com seus experimentos e uma bela teoria matemática lhes concederam um prêmio Nobel em 1963.

A transmissão de informação eletroquímica dos neurônios como um circuito elétrico em termos de capacitores e resistores, como pode ser conferido na Figura 6.

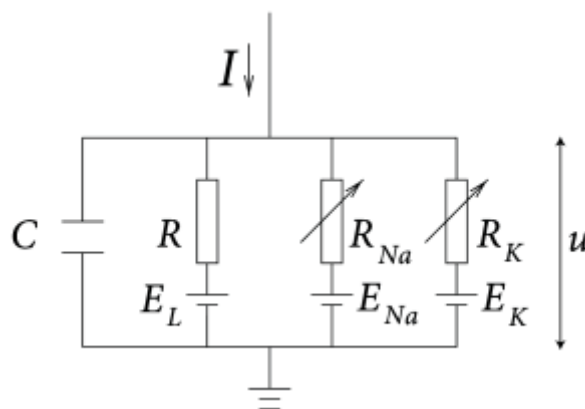


Figura 6: Esquema geral do modelo de Hodgkin e Huxley. [2]

No modelo padrão, existem apenas três tipos de canais, sódio (Na), potássio (K) e um de vazamento não específico (L), e assim, definidos R_{Na} , R_K e R_L que correspondem as respectivas resistências e E_{Na} , E_K e E_L os respectivos potenciais de equilíbrio. Também temos a u , que biologicamente é a voltagem que atravessa a membrana celular.

Dado este modelo, podemos então definir:

$$I_C = C \frac{du}{dt} = -\sum_k I_k(t) + I(t) \quad (1)$$

Onde $\sum_k I_k$ é a soma das correntes iônicas que atravessam a membrana celular.

Segundo Ermentrout, G. Bard em [11], a equação final do modelo padrão é definida em termos da condutância. Para descrevê-las os autores utilizaram um método chamado *voltage clamp* para isolar as correntes e poder computar as condutâncias. Neste método, uma corrente oposta à corrente que flui dos canais dependentes de voltagem é injetada ao neurônio e um circuito de retroalimentação é utilizado para manter os valores do potencial da membrana constantes. Dessa forma, Hodgkin e Huxley obtiveram as seguintes expressões que descrevem o comportamento estatístico das condutâncias dos canais de sódio e de potássio:

$$g_K = \overline{g}_K n^4 \quad \text{e} \quad g_{Na} = \overline{g}_{Na} m^3$$

Sendo \overline{g}_K e \overline{g}_{Na} condutâncias máximas e os parâmetros m , h e n descrevem a probabilidade de abertura e fechamento dos canais dependentes de voltagem, possuindo valores entre 0 e 1. Dessa forma, n^4 representa a probabilidade do canal de potássio estar aberto, e de forma semelhante m^3 representa a probabilidade do canal de sódio estar aberto. Sendo assim a forma final da equação de Hodgkin e Huxley é dada pela seguinte equação:

$$\sum_k I_k = g_{Na} m^3 h (u - E_{Na}) + g_K n^4 h (u - E_K) + g_L (u - E_L) \quad (2)$$

Os parâmetros de abertura e fechamento dos canais satisfazem as seguintes equações:

$$\frac{dn}{dt} = \alpha_n(V)(1 - n) - \beta_n(V)n = (n_\infty(V) - n)/\tau_n(V), \quad (3)$$

$$\frac{dm}{dt} = \alpha_m(V)(1 - m) - \beta_m(V)m = (m_\infty(V) - m)/\tau_m(V), \quad (4)$$

$$\frac{dh}{dt} = \alpha_h(V)(1 - h) - \beta_h(V)h = (h_\infty(V) - h)/\tau_h(V). \quad (5)$$

Sendo $X = m, n$ ou h , então:

$$X_\infty(V) = \frac{\alpha_X(V)}{\alpha_X(V) + \beta_X(V)}$$

$$\tau_\infty(V) = \frac{1}{\alpha_X(V) + \beta_X(V)}$$

Para ajustar aos valores experimentais, Hodgkin e Huxley escolheram os seguintes valores: $\overline{g}_{Na} = 120 \text{mS/cm}^3$, $\overline{g}_K = 36 \text{mS/cm}^3$, $\overline{g}_L = 0,3 \text{mS/cm}^3$, $E_{Na} = 50 \text{mV}$, $E_K = -77 \text{mV}$, $E_L = 54,4 \text{mV}$, obtendo:

$$\begin{aligned}
\alpha_n(V) &= 0,01(V + 55)/(1 - \exp(-(V + 55)/10)), \\
\beta_n(V) &= 0,125 \exp(-(V + 65)/80), \\
\alpha_m(V) &= 0,1(V + 40)/(1 - \exp(-(V + 40)/10)), \\
\beta_m(V) &= 4 \exp(-(V + 65)/18), \\
\alpha_h(V) &= 0,07(\exp(-(V + 55)/20)), \\
\beta_h(V) &= 1/(1 + \exp(-(V + 35)/10)).
\end{aligned}$$

Apropriadamente calibrada, o modelo de Hodgkin-Huxley é capaz de representar uma vasta gama de propriedades de neurônios reais. No entanto, apesar de realístico, o modelo é descrito por quatro equações diferenciais (2-5) que representam mudanças temporais e que estão relacionadas entre si, tornando o modelo complexo. Técnicas de resolução como o método de Euler podem ser empregadas adotando o parâmetro de tempo Δt (time-step). Quanto menor o valor de Δt , maior o esforço computacional, uma vez que mais cálculos por segundo são exigidos, podendo chegar a limites de intratabilidade computacional. Por outro lado, se escolhermos Δt grande, passos da abertura e fechamento dos canais iônicos serão perdidos, obtendo simulações com pouca acurácia e/ou instáveis [11].

Considerando que um neurônio do córtex está conectado a aproximadamente 10^4 outros neurônios [15], e que a no mínimo a cada 1ms (duração do pulso) todos os valores devem ser recalculados (tempo real), pensar em uma simulação nesta escala é uma tarefa computacionalmente difícil. Sendo assim, vários cientistas procuraram desenvolver modelos matemáticos mais simples, reduzindo os custos computacionais e obtendo os resultados semelhantes do ponto de vista fenomenológicos.

2.5 O Modelo de Izhikevich

Dado o alto custo computacional dos modelos de Hodgkin-Huxley, alguns modelos foram desenvolvidos posteriormente, um exemplo são os modelos integrate-and-fire, brevemente descrito em [9]. No entanto, apesar de serem eficientes computacionalmente, estes são incapazes de produzir alguns comportamentos e dinâmicas observadas em neurônios do córtex cerebral.

Utilizando a teoria matemática das bifurcações e formas normais de redução, o matemático Eugene Izhikevich propôs um novo modelo dada por duas equações diferenciais descritas nas equações a seguir:

$$v' = 0,04v^2 + 5v + 140 - u + I \quad (6)$$

$$u' = a(bv - u) \quad (7)$$

Com a seguinte equação auxiliar para $v \geq 30mV$:

$$v \leftarrow c \quad (8)$$

$$u \leftarrow u + d \quad (9)$$

Nestas equações, diremos que v representa o potencial da membrana u representa a variável de recuperação da membrana, que leva em conta a ativação da corrente iônica de K^+ e inativação da corrente iônica de Na^+ , isto é, a forma que o potencial da membrana é reestabelecido ao equilíbrio.

Após o pulso atingir seu ápice em 30mV, a voltagem da membrana e a variável de recuperação são resetadas de acordo com a equação auxiliar descrita anteriormente. A somatória das correntes sinápticas é injetada através da variável I .

Os parâmetros a, b, c e d estão envolvidas no processo de recuperação da membrana;

- a determina a escala de tempo em que a recuperação ocorre. Valores menores de a , resultam numa recuperação mais lenta.
- b descreve a sensibilidade da variável u as flutuações no potencial da membrana.
- c descreve o valor do potencial da membrana após a ocorrência do pulso, causado pela abertura dos canais de K^+ .
- d descreve o valor de u após a ocorrência do pulso, causado pelo fechamento dos canais de Na^+ .

Para a resolução das equações (3) e (4) é utilizado o método de Euler. Dado o time-step Δt , a equação diferencial \dot{y} será aproximada para $y(t + \Delta t) = y(t) + \Delta t \dot{y}(t)$. A questão da possível instabilidade gerada por este método é solucionada pelo autor computando as equações duas vezes, cada uma por um fator 0,5 como pode ser melhor descrito por E. M. Izhikevich em [1].

Uma grande vantagem desse método é que cada equação necessita ser computada apenas uma vez a cada time-step, sendo $\Delta t = 1ms$ o adotado por Izhikevich. Isto é, a cada 1 ms todos os neurônios pertencentes à rede devem ser recalculados. Se a rede contiver “ n ” neurônios, isso significa que devemos recalcular na ordem de n^2 parâmetros.

Para otimizar o processamento das equações de Izhikevich, é proposto neste trabalho o desenvolvimento de um hardware dedicado em FPGA, utilizando técnicas de pipeline. Como foi dito, para diminuir a quantidade de memória utilizada, diminuir a taxa de transferência e ainda o consumo de energia, este trabalho avaliou a possibilidade de criar um processador em FPGA que opera em ponto flutuante, mas que armazena os dados em 16 bits, no formato half-precision descrito pelo IEEE 754, discutido em 1.3.

Note que todas as variáveis que desejamos codificar de 32 bits para 16 bits e vice-versa estão definidas em um intervalo específico de valores requeridos em simulações de redes neurais pulsadas. Nestas redes, o potencial de membrana dos neurônios pode variar entre -70mV até 30mV. Por outro lado, os pesos sinápticos podem ter valores tão pequenos quanto $1/10^4$ (0,0001), que representa a conexão de um neurônio a 10.000 vizinhos, ou podem ter valores perto de uma centena, que representaria uma conexão sináptica bastante forte. Estes são os valores que precisam ser considerados a cada iteração da rede, e em ambos os casos se encaixam dentro da capacidade de representação em half-precision. Dessa forma, podemos intuir que não teremos problemas em usar half-precision.

4. Metodologia

O presente trabalho foi desenvolvido em 5 etapas;

1. Desenvolvimento de um processador em hardware usando técnicas de pipeline para a execução das equações de Izhikevich.
2. Desenvolvimento de um conversor de single-precision para half-precision e vice-versa.
3. Introdução do conversor no processador desenvolvido na etapa 1. De tal forma que os valores de resposta das equações de Izhikevich em single-precision foram convertidos para half-precision para serem guardados na memória. Em seguida, esses valores foram lidos e convertidos para single-precision para serem imputados na próxima iteração do cálculo das equações de Izhikevich.
4. Avaliação dos resultados obtidos na etapa 3 utilizando um método canônico da neurociência.
5. Desenvolvimento de uma aplicação que computa as equações de Izhikevich para uma rede de neurônios. A avaliação nesta etapa também foi feita utilizando um método canônico da neurociência.

As etapas de 1 a 3 foram realizadas no Quartus Prime 17.0 para a configuração de uma Cyclone V – EP45CSEMA5F31C6, em linguagem Verilog. Já as etapas 4 e 5 foram realizados em software utilizando linguagem C++.

4.1. Cálculo do Neurônio segundo o modelo de Izhikevich

Foi implementada a equação de Izhikevich para um neurônio. Para esta simulação foram escolhidos para os parâmetros a, b, c e d , os valores característicos um padrão de disparo dito regular, que são os mais comuns no córtex cerebral. Foram seguidos os valores de acordo com E. M. Izhikevich em [1], sendo eles:

$$a = 0,02; b = 0,2; c = -65 \text{ e } d = 2.$$

Foram utilizados os módulos de soma e subtração (IPs Intel floating point) com latência de 7 clocks, multiplicação com latência de 5 clocks, todos disponibilizados pela Altera, e mais um módulo de retardo, uma memória do tipo FIFO, para fazer cópias sucessivas dos registradores afim do resultado ser usado num tempo de clock posterior. Este último tem latência igual ao tempo que se deseja deslocar.

O tempo de cada operação foi controlada por um outro registrador que computava cada subida de clock a partir do início da simulação. O tempo de latência do processador é de 40 clocks.

O esquema geral do pipeline pode ser conferido na Figura 7 e o custo para o módulo pode ser conferido na figura 8.

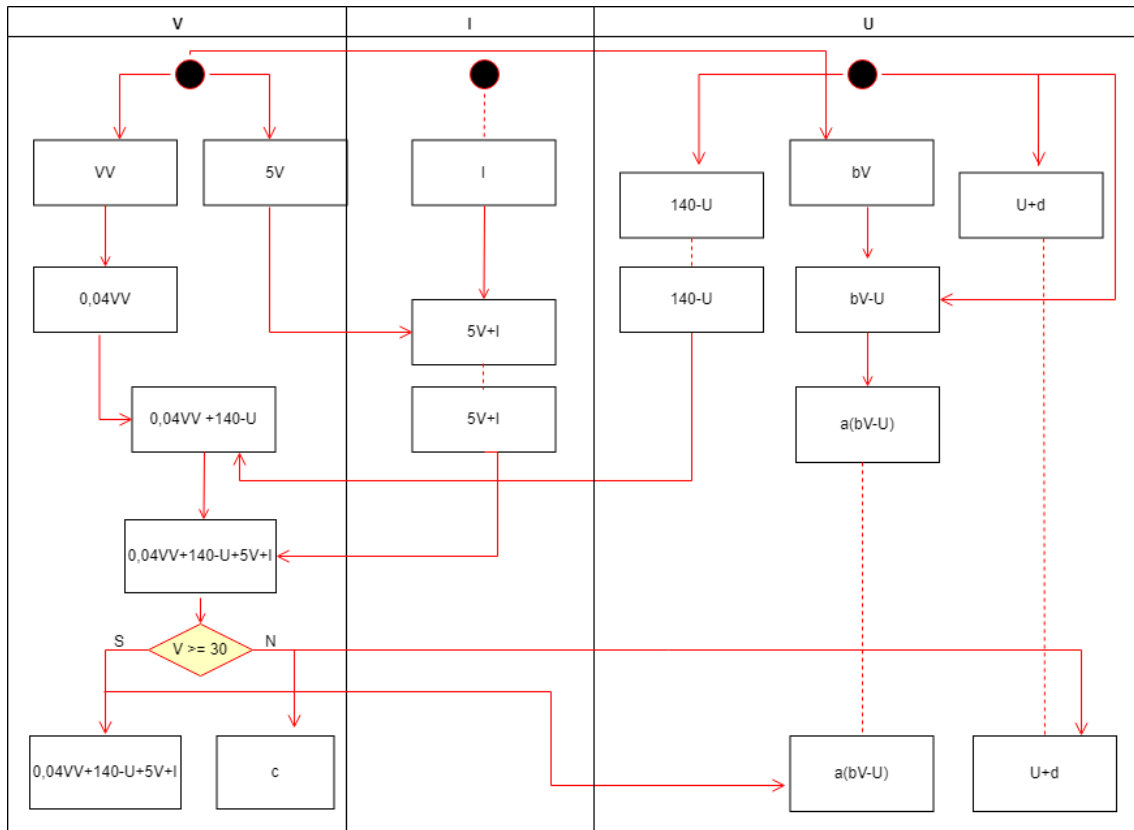


Figura 7: Esquema geral do pipeline para a execução das equações de Izhikevich. As linhas tracejadas indicam o uso do módulo de retardo

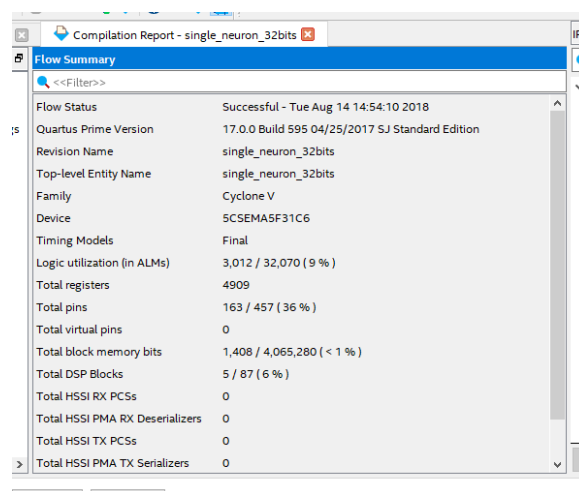


Figura 8: Custo para a implementação de um neurônio segundo o modelo de Izhikevich

Neste trabalho, foram simuladas apenas 10 operações, cada uma recebendo de entrada o valor calculado anterior, sendo -70mV o valor inicial para V e -14mV o valor inicial para U, apenas para demonstrar graficamente o funcionamento do circuito.

A equação de Izhikevich para um neurônio também foi implementada em software, utilizando linguagem Python para que os resultados possam ser confrontados. Neste passo utilizou-se o pacote Numpy para converter todos os números para single-precision.

Os resultados serão apresentados e discutidos na seção 5.1.

4.2. Conversores de single-precision para half-precision e de half-precision para single-precision

Para converter um float de 32 bits para um de 16 bits, foram feitas as seguintes conversões para operação em região normal:

- ✓ $Exp_{fp16} = exp_{fp32} - 127 + 31$ (bias fp32) + 31 (bias fp16)
- ✓ Mantissa fp16 = mantissa fp32 truncado em 10 bits

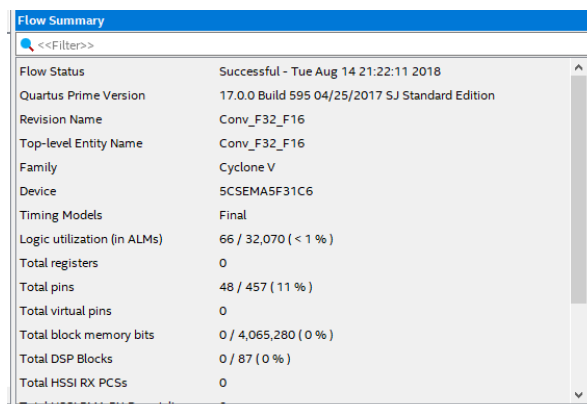
Para este regime temos que:

$$113 \leq Exp_{fp16} \leq 142$$

Para melhorar um pouco mais a precisão, fizemos uma adaptação para afim de considerar uma pequena região de subnormal que compreende os valores $103 \leq Exp_{fp16} \leq 112$. Sendo:

- ✓ $Exp_{fp16} = 0$
- ✓ Mantissa fp16 $\in [10^{-9}, 10^{-8}, 10^{-7}, \dots, 10^{-1}]$ (um para cada valor inteiro do intervalo)

O custo para o módulo pode ser conferido na figura 9.



| Flow Summary | |
|-----------------------------|---|
| Flow Status | Successful - Tue Aug 14 21:22:11 2018 |
| Quartus Prime Version | 17.0.0 Build 595 04/25/2017 SJ Standard Edition |
| Revision Name | Conv_F32_F16 |
| Top-level Entity Name | Conv_F32_F16 |
| Family | Cyclone V |
| Device | 5CSEMA5F31C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 66 / 32,070 (< 1 %) |
| Total registers | 0 |
| Total pins | 48 / 457 (11 %) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 4,065,280 (0 %) |
| Total DSP Blocks | 0 / 87 (0 %) |
| Total HSSI RX PCSs | 0 |

Figura 9: Custo para a implementação do conversor de fp 32 para fp16

Por outro lado, para converter um float de 16 bits para um de 32 bits, foram consideradas as operações inversas das descritas acima. O custo para o módulo pode ser conferido na figura 10.

| Flow Summary | |
|-----------------------------|---|
| Flow Status | Successful - Tue Aug 14 21:36:08 2018 |
| Quartus Prime Version | 17.0.0 Build 595 04/25/2017 SJ Standard Edition |
| Revision Name | Conv_F16_F32 |
| Top-level Entity Name | Conv_F16_F32 |
| Family | Cyclone V |
| Device | 5CSEMA5F31C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 43 / 32,070 (< 1 %) |
| Total registers | 0 |
| Total pins | 48 / 457 (11 %) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 4,065,280 (0 %) |
| Total DSP Blocks | 0 / 87 (0 %) |
| Total HSSI RX PCSs | 0 |

Figura 10: Custo para a implementação do conversor de fp 16 para fp32

Como prova de conceito foi realizada a seguinte simulação:



E então foi comparado os resultados de entrada e saída para os seguintes valores;

- $6,1 \times 10^{-4}$ – menor positivo normal com precisão de 16 bits
- 65504 – maior positivo com precisão de 16 bits
- $1 + 10^{-10}$ - maior float após 1 com precisão de 16 bits
- 6×10^{-5} – para testar as condições de limite
- 65505 - para testar as condições de limite

Neste módulo os valores que deveriam representar um *underflow* foram configurados para serem resetados para 0 e os valores que deveriam representar um *overflow*, foram configurados para serem resetados para o maior positivo a ser representado pelo padrão.

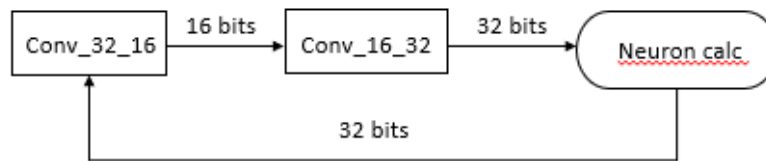
Os resultados serão apresentados e discutidos na seção 5.2.

4.3. Cálculo de um Neurônio segundo o modelo de Izhikevich com conversão para half-precision

Nesta etapa estamos interessados em saber se as conversões afetam o cálculo do modelo de neurônio pulsado de Izhikevich.

Como foi dito, o padrão IEEE determina que a realização de cálculos aritméticos seja feita no mínimo com 32 bits além disso, estamos interessados numa economia em memória de transferência e banda.

Sendo assim, consideramos um projeto com o seguinte esquema geral:



O custo para o módulo pode ser conferido na figura 11.

| Flow Summary | |
|-----------------------------|---|
| Flow Status | Successful - Tue Aug 14 14:29:12 2018 |
| Quartus Prime Version | 17.0.0 Build 595 04/25/2017 SJ Standard Edition |
| Revision Name | single_neuron_calcV4 |
| Top-level Entity Name | single_neuron_calcV4 |
| Family | Cyclone V |
| Device | 5CSEMA5F31C6 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 3,168 / 32,070 (10 %) |
| Total registers | 4665 |
| Total pins | 195 / 457 (43 %) |
| Total virtual pins | 0 |
| Total block memory bits | 1,408 / 4,065,280 (< 1 %) |
| Total DSP Blocks | 5 / 87 (6 %) |
| Total HSSI RX PCSs | 0 |

Figura 11: Custo para a implementação de um neurônio segundo o modelo de Izhikevich com as conversões de fp 32 para fp16 e vice-versa.

Foram simulados 10 ciclos, cada uma recebendo de entrada o valor calculado anterior, sendo -70mV o valor inicial para V e -14mV o valor inicial para U, apenas para demonstrar graficamente o funcionamento do circuito.

Para verificar o correto funcionamento do modelo adotado é preciso que o mesmo reproduza de forma suficientemente fiel o comportamento biológico. Para isso, existem alguns métodos canônicos em neurociência para fazer esta análise, descritos com maiores detalhes em [15]. Adotaremos três deles, descritos na seção 4.4, análise da curva FxI e 4.5, predição do próximo disparo. O terceiro método canônico pode ser descrito como o acompanhamento das formas de onda em regime subthreshold (sub-disparo) do neurônio, que foi descrito no trabalho “*Half-precision Floating Point on Spiking Neural Networks Simulations in FPGA*” [9], e pode ser visto na figura 17 abaixo. Esse método consiste basicamente em verificar se há diferença significativa para os valores de tensão de membrana quando convertidos de single para half-precision. No entanto, estamos interessados em verificar se o modelo com as conversões para half-precision se comporta da mesma maneira que o modelo sem a conversão.

Os resultados serão apresentados e discutidos na seção 5.3.

4.4. Análise da curva FxI

Este método analisa a frequência de disparo de um dado neurônio após ser injetado uma corrente constante. O objetivo é verificar se a conversão causa mudanças na taxa de disparo do neurônio. Logo, o processo de injetar a corrente constante em um neurônio foi reproduzido em ambos os modelos, com e sem conversão, com os seguintes parâmetros:

- Corrente contínua de 4 a 50 pA
- Duração de 5.2s
- Frequência foi medida de 0,2 a 5,2 segundos.

Os resultados serão apresentados e discutidos na seção 5.4.

4.5. Análise do Comportamento de uma Rede.

O método 4.4 analisa o comportamento de um neurônio. No entanto este método não garante que o erro gerado nas conversões de single-precision para half-precision não se propague pela rede e mude a sua dinâmica.

Para verificar se o comportamento dos dois modelos (com e sem conversão), foram simuladas uma rede para cada um. Inicialmente foi criada uma topologia idêntica a descrita no artigo de E. M. Izhikevich [1], com 200 neurônios totalmente conectados, com o objetivo de gerar todos os parâmetros iniciais. Estes parâmetros foram salvos para serem utilizados como entrada das redes criadas posteriormente. Em seguida, duas redes com a mesma topologia, uma com todo o processo sendo realizado em 32 bits, e a segunda seguindo o padrão de conversão como no experimento 4.3, ambas carregando todas as variáveis iniciais e parâmetros geradas na topologia inicial.

O esquema geral pode ser verificado na Figura 12.

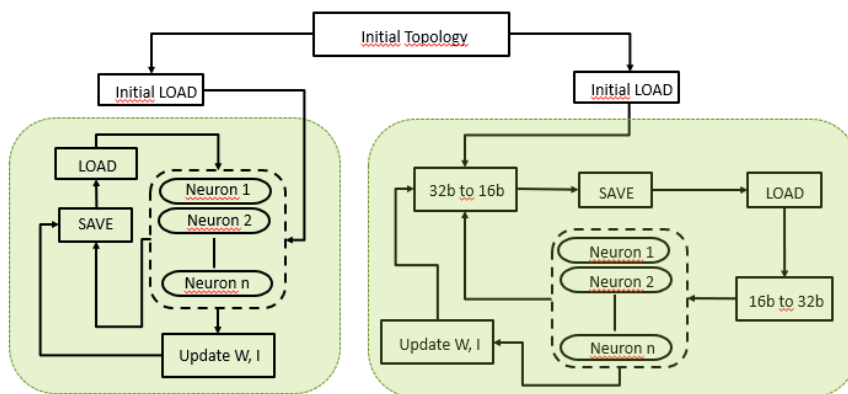


Figura 12: Esquema geral do experimento 4.5

O objetivo desta etapa é analisar os resultados e encontrar a formação de grupos de neurônios que se comportam de acordo com algum padrão. Sendo assim, se o mesmo padrão for observado nas duas simulações poderemos concluir que a conversão para half-precision não alterou o comportamento da rede.

Os resultados serão apresentados e discutidos na seção 5.5.

5. Resultados

5.1. Cálculo do Neurônio de Izhikevich

Na figura 13 podemos conferir o resultado da simulação proposta em 4.1 obtida pelo software ModelSim. Para melhor visualização os dados são mostrados na Tabela 5.

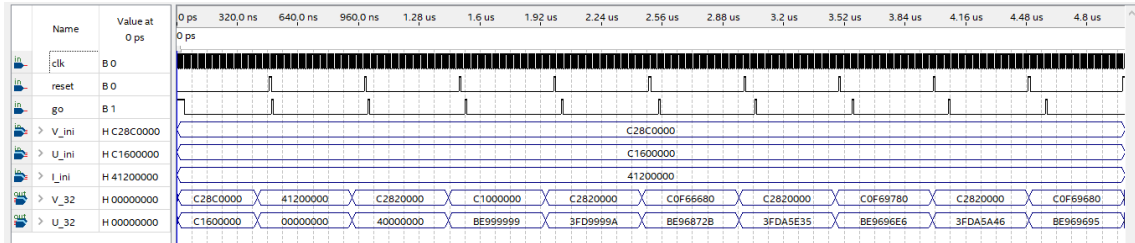


Figura 13: Resultado do experimento 3.1

Tabela 5: Valores obtidos para o Potencial de Membrana para o experimento 4.1

| Loop | Potencial da Membrana [mV] | |
|------|----------------------------|----------|
| | Hexa | Decimal |
| 0 | C28C0000 | -70 |
| 1 | 41200000 | 10 |
| 2 | C2820000 | -65 |
| 3 | C1000000 | -8 |
| 4 | C2820000 | -65 |
| 5 | C0F66800 | -7,7 |
| 6 | C2820000 | -65 |
| 7 | C0F69800 | -7,706 |
| 8 | C2820000 | -65 |
| 9 | C0F69800 | -7,70588 |

O gráfico a seguir mostra a comparação do modelo simulado em FPGA e em software. O traço laranja que indica os resultados para a FPGA foi deslocado em -10 mV para uma melhor visualização, uma vez que os resultados foram exatamente os mesmos.



Figura 14: Comparação entre as simulações em software e na FPGA para o experimento 3.1

Com a comparação do modelo simulado em software, demonstramos o correto funcionamento do processador desenvolvido para executar as equações de Izhikevich.

Sendo assim, obtemos um processador que poderá operar na frequência escolhida e que após o período de latência de 40 clocks, a cada pulso de clock posterior ele libera um novo resultado. Ou seja, para uma frequência de 120 MHz, o processador é capaz de computar 120 milhões de cálculos por segundo após o período de latência.

5.2. Conversores de fp32 para fp16 e de fp16 para fp32

Na figura 15 podemos conferir o resultado da simulação proposta em 4.2 e para uma melhor visualização verificados na Tabela 6.

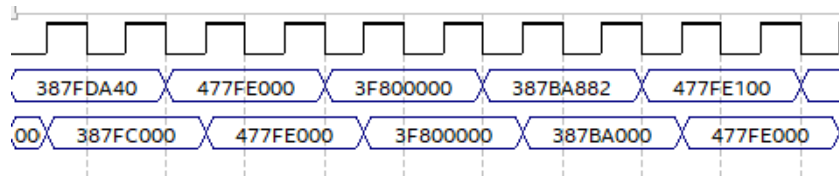


Figura 15: Resultado do experimento 3.2

Tabela 6: Resultados em hexa e decimal de entrada e saída do experimento 3.2

| IN_32 | Decimal | OUT_32 | Decimal |
|----------|-----------------------|----------|--------------------------|
| 387FDA40 | $0,61 \times 10^{-4}$ | 387FC000 | $0,60975 \times 10^{-4}$ |
| 477FE000 | 65504 | 477FE000 | 65504 |
| 3F800000 | $1 + 10^{-10}$ | 3F800000 | $1 + 10^{-10}$ |
| 387BA882 | 6×10^{-5} | 387BA000 | $5,9992 \times 10^{-5}$ |
| 477FE100 | 65505 | 477FE000 | 65504 |

Podemos notar que as diferenças obtidas com as conversões são muito pequenas, o que acreditamos ser suficiente para atender a proposta, uma vez que trabalhamos em uma faixa específica de valores. Também podemos notar no último exemplo o tratamento para o valor que caracterizaria um overflow, note que ele foi resetado para o maior valor a ser representado pelo padrão.

Com o conversor operando corretamente mesmo para regiões sensíveis, podemos adicioná-lo ao processador desenvolvido em 3.1.

5.3. Cálculo do Neurônio de Izhikevich com conversão

Na figura 16 podemos conferir o resultado da simulação proposta em 4.3 e para uma melhor visualização verificados na Tabela 7.

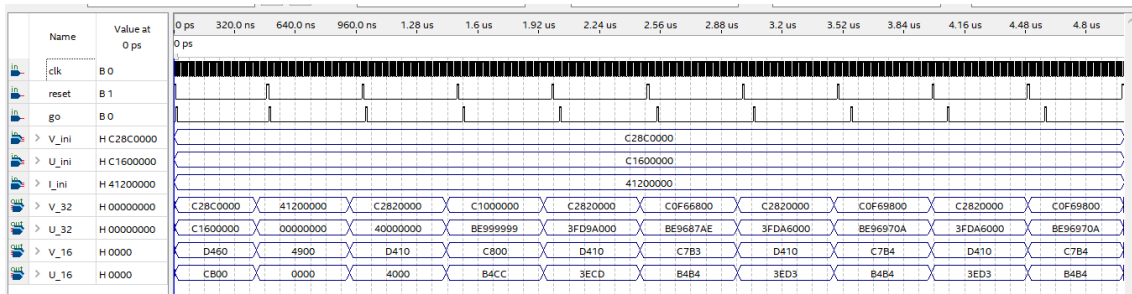


Figura 16: Resultado do experimento 3.3

Tabela 7: Valores obtidos para o Potencial de Membrana para o experimento 3.3

| Loop | Potencial da Membrana [mV] | | |
|------|----------------------------|------|----------|
| | Hexa | | Decimal |
| 0 | C28C0000 | D460 | -70 |
| 1 | 41200000 | 4900 | 10 |
| 2 | C2820000 | D410 | -65 |
| 3 | C1000000 | C800 | -8 |
| 4 | C2820000 | D410 | -65 |
| 5 | C0F66800 | C7B3 | -7,7 |
| 6 | C2820000 | D410 | -65 |
| 7 | C0F69800 | C7B4 | -7,706 |
| 8 | C2820000 | D410 | -65 |
| 9 | C0F69800 | C7B4 | -7,70588 |

O gráfico a seguir mostra a comparação do modelo simulado em FPGA com fp32, em software e com as conversões para fp16. O traço laranja que indica os resultados para a FPGA em fp32 foi deslocado em -10 mV, o verde para os resultados da FPGA com as conversões para fp16 foi deslocado em -20 mV (em relação a simulação em software) para uma melhor visualização, uma vez que os resultados foram exatamente os mesmos.

Comparação entre os resultados em Software e FPGA com fp32 e FPGA com conversão para fp16

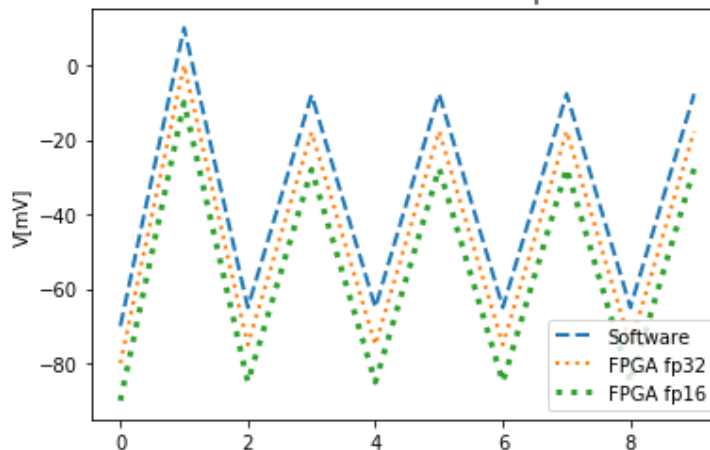


Figura 17: Comparação entre as simulações em software e na FPGA o experimento 3.3

Comparando ao modelo simulado em software, verificamos o correto funcionamento do processador desenvolvido para executar as equações de Izhikevich com a atuação do conversor da forma descrita em 3.3. Sendo assim, além de desenvolver um processador capaz de executar milhões de cálculos por segundo, temos um processador que utiliza metade da banda em comparação com o desenvolvido em 3.1, cujos valores de operação em regime sub-disparo (contínuo) da tensão de membrana não representam nenhuma variação significativa no regime individual dos neurônios nem na rede.

5.4. Análise da curva FxI

Os resultados obtidos da análise foram plotados em um gráfico ilustrado na Figura 18. Os pontos quadrados azuis representam os valores simulados em single-precision, já os pontos em laranja representam os valores simulados para o modelo com conversão para half-precision.

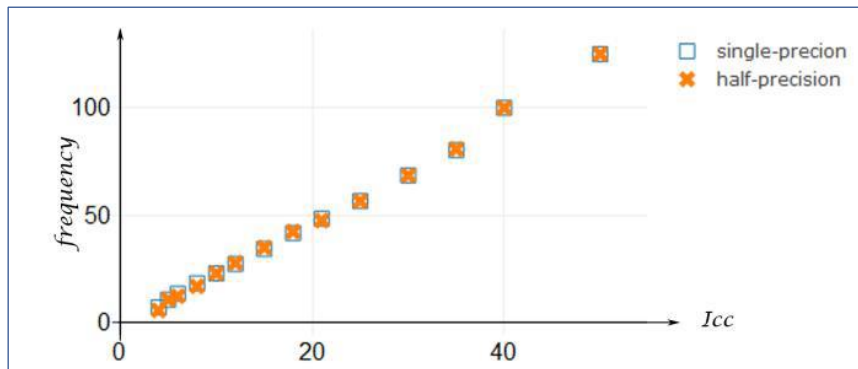


Figura 18: Curva FxI para o modelo de Izhikevich. [9]

É possível observar que os resultados simulados são quase idênticos, indicando que converter os valores do potencial da membrana para half-precision para armazená-lo e em seguida recuperá-lo convertendo para single-precision, usando o mesmo no cálculo da próxima iteração, não afeta a frequência de disparo do neurônio. Isso está em conformidade do que se poderia esperar, uma vez que os valores de tensão em regime sub-disparo (subthreshold) quase não sofre alterações, como vimos na seção anterior.

O comportamento de resposta a um estímulo constante seguido por um período de estabilização é chamado de adaptação de frequência de pulso (spike-frequency adaptation). Este processo está relacionado com a classificação de neurônios em fásico, fásico-tônico e tônico. Os tônicos, são neurônios que mantém a resposta a taxa de resposta quase constante. Os neurônios fásicos têm uma alta taxa de resposta apenas no início, e os fásico-tônico possuem uma taxa de resposta intermediária. Dessa forma, este é um importante resultado para verificar se mesmo na presença das conversões numéricas o modelo se mantém análogo com o modelo sem conversão. Mais detalhes da relevância e aplicação da análise da taxa de disparo do neurônio pode ser encontrada em Bears [15].

5.5. Análise do Comportamento de uma rede

Conforme foi introduzido na seção 3.1, uma forma de analisar uma rede e decidir se os resultados de ambos os modelos (com e sem conversão) permanecem homogêneos, é acompanhar a formação e o comportamento de assembleias neurais.

Uma vez que estes grupos forem observados, podemos utilizar um outro método canônico em neurociência que é a predição de disparos neurais. Isso é possível pois de acordo com os estudos descritos por Ranhel em [5], numa rede grupos de neurônios interagem entre si de acordo com um certo padrão. Isto é, caso um grupo de neurônios dispare, ele dá início com probabilidade estatisticamente significativa ao disparo de um outro grupo de neurônios.

Dessa forma, apesar dos 200 neurônios simulados, foram escolhidos dois grupos A e B totalizando 20 neurônios para serem acompanhados.

O resultado pode ser acompanhado na Figura 19:

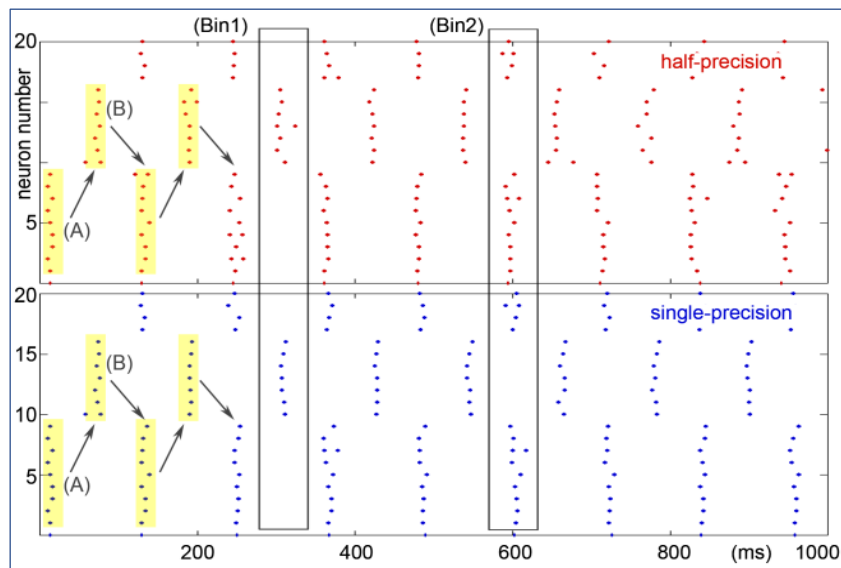


Figura 19: Traço de disparos para a rede do experimento 4.5 [9]

Podemos observar os neurônios do grupo A disparam estes demoram um certo tempo para atingir o grupo B e quando isto acontece o grupo B dispara. Note que este padrão se repete para o modelo com e sem conversão. Esta observação já permite concluir que a dinâmica neural não foi afetada com a introdução da conversão para half-precision ao modelo de Izhikevich.

Apesar dos resultados relevantes, também foram escolhidas duas janelas de intervalo (Bin 1 and Bin 2) afim de comparar o tempo exato do disparo do neurônio, para uma análise mais profunda. Podemos notar que estes tempos não são exatamente os mesmos. No entanto este resultado já era esperado devido ao ruído injetado na rede.

5. Conclusões

As redes neurais pulsadas têm sido simuladas em software com pontos flutuantes por um longo período. Para as tecnologias atuais a simulação em tempo real e redes de larga escala calculadas em single-precision é um problema complexo. Uma solução é desenhar processadores dedicados de ponto fixo. Esta era a proposta inicial do trabalho. Contudo, deparou-se com o problema de altas taxas de transferência e amplo uso de memória para implementar uma rede relativamente pequena.

Foi feita opção pelo ponto flutuante porque são mais eficientes na representação numérica, são de fácil conversão, e são compatíveis com a maioria dos programas de computador; dessa forma, apresentam boa relação custo benefício. Um problema que existe neste caso é utilizar 32 bits de memória, barramentos maiores e mais ciclos de clock para transferência de grandes quantidades de dados. Dessa forma, uma possível solução seria armazenar os valores reais no formato half-precision.

Neste trabalho além da implementação de um processador dedicado para calcular o modelo neural de Izhikevich em hardware, também desenvolvemos um conversor do padrão numérico single-precision para half-precision float e vice-versa, além de aplicarmos as conversões no processador do modelo de neurônio. Também analisamos o comportamento tanto do neurônio individualmente quanto da rede para o modelo de Izhikevich com e sem conversão. No artigo que escrevemos para IJCNN-2018 do IEEE, fizemos ainda a mesma análise para os neurônios LI&F [9]. Dessa forma, podemos concluir que é possível implementar o cálculo computacional dos neurônios de redes neurais pulsadas em hardware, usando half-precision como forma de armazenagem de dados, desde que seus parâmetros e variáveis se encaixem nas faixas de precisão possíveis para o formato.

Muitos benefícios podem surgir de uma implementação em half-precision no lugar de uma em single-precision, a mais importante é a economia de memória, barramentos mais simples, menor frequência de clock, o que ao final se traduz em redução de consumo de energia, essenciais para implementação de Redes Neurais Pulsadas em sistemas embarcados.

6. Referências

- [1] E. M. Izhikevich, “Simple model of spiking neurons,” *IEEE Trans. Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.
- [2] W. Gerstner, W. Kistler, R. Naund and L. Paninski, “Neural Dynamics”, *Cambridge University Press*.
- [3] D. Pani, P. Meloni, G. Tuvèri, F. Palumbo, P. Massobrio, and L. Raffo, “An FPGA Platform for Real-Time Simulation of Spiking Neuronal Networks.,” *Front. Neurosci.*, vol. 11, p. 90, 2017.
- [4] J. Ranhel, “Neural assembly computing,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 23, no. 6, 2012.
- [5] J. Ranhel, M. Lobo Netto, and E. Del Moral Hernandez, “How complex behavior emerge from spikes,” in *Cognitive Science: Recent Advances and Recurring Problems, 1st ed.*, J. . Adams, Frederick; Pessoa, O. Jr; Kogler, Ed. Wilmington, USA: Vernon Press, 2017, pp. 197–218.
- [6] Rajaraman, V. “IEEE Standard for Floating Point Numbers”, *Supercomputer Education and Research Centre Indian Institute of Science*.
- [7] D. Zhaoxia, C. Xu, Q. Cai and P. Faraboschi, “Reduced-Precision Memory Value Approximation for Deep Learning”, *Hewlett Packard Labs* .
- [8] D. Thomas, W Luk, “FPGA Accelerated Simulation of Biologically Plausible Spiking Neural Network”, *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*.
- [9] C. Zambelli, J. Ranhel, “Half-precision Floating Point on Spiking Neural Networks Simulations in FPGA”, *2018 IEEE – World Congresso n Computacional Intelligence*.
- [10] Eugene M. Izhizivich, “Neural Systems in Neuroscience”, MIT Press.
- [11] Ermentrout, G. Bard, Terman, David H, “Mathematical Foundations of Neuroscience”, Springer.
- [12] Altera “White Paper- Taking Advantage of Advances in FPGA Floating-Point IP Cores”.
- [13] I. Kuon, R. Tessier, J. Rose, “FPGA Architecture: Survey and Challenges”, *Foundations and TrendsR_ in Electronic Design Automation Vol. 2, No. 2 (2007) 135–253*.
- [14] Altera, “Quartus II Introduction Using Schematic Design”.
- [15] Bears, Connor & Paradiso, “Neurociências - Desevendando o Sistema Nervoso” - 4ª Ed. 2017.
- [16] D. O. Hebb, “The Organization of Behavior: A Neuropsychological Theory”, New York JOHN WILEY if SONS, Inc. London CHAPMAN i HALL, Limited.
- [17] K. Rice, M. Bhuiyan, T. Taha, C. Vutsinas, M. Smith, FPGA “Implementation of Izhikevich Spiking Neural Networks for Character Recognition”, 2009 International Conference on Reconfigurable Computing and FPGAs.

ANEXO 1

Artigo apresentado no IJCNN 2018

“Half-precision Floating Point on Spiking Neural Networks Simulations in FPGA”

Half-precision Floating Point on Spiking Neural Networks Simulations in FPGA

Carolina Zambelli

CECS – Engenharia de Informação
UFABC – Univ. Federal do ABC
Santo André, Brazil
caroll.zambelli@gmail.com

João Ranhel

CECS – Engenharia de Informação
UFABC – Univ. Federal do ABC
Santo André, Brazil
joao.ranhel@ufabc.edu.br

Abstract— The use of half-precision floating-point numbers (hFP) in simulations of spiking neural networks (SNN) was investigated. The hFP format is used successfully in computer graphics and video games for storage and data transfer. The IEEE 754-2008 standard settles that arithmetic operations must occur at least on single-precision floating-point format (sFP). This means that it is necessary to convert hFP to sFP for arithmetical operations and reconvert the results to hFP before storing it. The influence of successive conversions when simulating SNN is the main concern of this article. Three methods were used to evaluate the impact of hFP on SNNs: (i) F-I curve, (ii) subthreshold regime, and (iii) the time for the next spike. We have tested the leaky integrate-and-fire and the Izhikevich's neuron model; both presented similar results. The data show that SNNs simulated with sFP present equivalent results when compared to the ones simulated with hFP with identical topology. Such results are important because hFP requires half of the memory space, simpler buses, and lower bandwidth for transferring data. We may infer they require lower clock frequency consequently lower power consumption. These are essential factors for real-time simulation of SNN on embedded electronics. The sFP to hFP conversion circuits, and vice versa, may be implemented using few logical blocks in a field-programmable gate arrays (FPGA) with no relevant latency. We conclude that data in the hFP format are suitable for SNNs synthesized in FPGAs, even though such implementations require conversion circuits.

Keywords—*neurocomputing; artificial neural network; half-precision floating; neuromorphic circuits; real-time SNN;*

I. INTRODUCTION

Spiking neural networks (SNN) differ from traditional artificial neural networks (ANN) mainly because the former incorporates the time into their operating model, besides the synaptic weights. Usually, the neuron outputs are updated every iteration in ANN, whereas in the SNN the neurons fire only after reaching a certain value in the membrane potential. When it occurs, a spike is generated [1] [2] [3]. The spike is a Boolean information that travels along axons to other neurons which, in turn, increase or decrease their membrane potentials in accordance with received spikes. The sum of all spikes reaching neurons at a specific time causes excitatory/inhibitory postsynaptic potential (EPSP/IPSP) in a recipient neuron. Some SNN explicitly consider the spike propagation delay (SNNwD), mimicking biological networks. In order to consider spike

propagation delay, SNNwDs ought to compute some kind of temporal convolution at each iteration. This is computationally demanding and generally requires high data transfer rates from and to the memory.

In the last decades various neuron models were developed for SNN, each one demanding certain amount of computational effort (see [4]). To increase performance and meet the SNN computational demand, several projects implement hardware processors for computing neuron models (see [5] [6] [7] [8]), or even for full neuromorphic network simulation [9].

We have worked on hardware implementation in FPGA of a real-time SNNwD calculated using single-precision floating-point numbers (sFP). Our network computes spike-timing dependent plasticity (STDP) [10] [11], propagation delays, as well as homeostatic synaptic (short-term) plasticity [12]. We realized that even when the FPGA has enough internal RAM, the number and complexity of internal buses, the rate of memory data transferences and the clock speed outgrow the FPGA capabilities. Further, if an external memory is required, the memory interfaces uses lots of FPGA pins, which increases the cost of the FPGA device.

One solution may be to encode neural network variables and parameters using the half-precision format (hFP) as defined by the IEEE Standard for Floating Point (IEEE 754-2008). This format uses only 16-bit data opposite to 32 bits used in single-precision format. In this paper, we describe our findings on the influences of half-precision data on SNN simulation. Firstly, we analyzed if it is mathematically viable to simulate SNN with leak integrate-and-fire (LIF) neuron [13], described in section II. Mathematically, there is no problem on saving LIF data using hFP after scaling some variables. Then, we simulated SNN with two different neuron models: LIF and the Izhikevich's model (IZH) [14]. We also simulated SNNwD for both neuron models. It was used three canonical forms for checking if simulations remain homogenous for both hFP and sFP: (i) F×I curve, (ii) subthreshold regime curve, and (iii) the timing interval for the next firing; whose methods are described on section III.

The results are conclusive for these networks, modeled with the two models of neurons mentioned above, with or without delay in spike propagation: hFP numbers can be used in the SNN. During network simulations, the calculations are done in the sFP format; therefore, it is necessary to convert sFP to hFP,

and vice versa, for each operation. The benefits of using hFP for storage and data transfer far outweigh the cost of extra hardware required for conversion. Discussions on hFP restrictions and applicability in the SNN are in section IV; and the conclusions are in section V.

Engineering applications of SNNs have been limited mainly due to computational costs. This work shows that it is possible to reduce some computational requirements to simulate SNN or SNNwD in real time. By reducing the number of bits for data encoding, it is possible to reduce memory space, bandwidth and clock frequency, resulting in lower power consumption, mandatory in SNN simulations within embedded systems.

II. FLOATING POINT AND SPIKING NEURAL NETWORKS

This section holds a brief comparison among hFP, sFP and fixed-point formats. It is a common practice to use fixed-point numbers in FPGA and hardware projects. For instance, when implementing a SNN on hardware, Pani and collaborators used Q3.4 format for storing synaptic weights, meaning 4 bits left for the fractional portion over a 7-bit word [9]. It means also that the remaining bits are all that the designer have for encoding the sign bit and the integer portion. Table I shows some of the Q.i.f capabilities for encoding signed-numbers on 16-bit vectors:

TABLE I. 16-BITS Q.I.F CAPABILITIES

| Format | BITS | | | binary | | decimal | |
|--------|------|-----|------|---------|-----------|---------|----------------------|
| | +/- | int | frac | int | frac | int | resolution |
| Q1.15 | 1 | 0 | 15 | 0 | 2^{-15} | 0 | 3×10^{-5} |
| Q4.12 | 1 | 3 | 12 | 2^3-1 | 2^{-12} | 7 | 2.4×10^{-4} |
| Q5.11 | 1 | 4 | 11 | 2^4-1 | 2^{-11} | 15 | 2^{-11} |
| Q6.10 | 1 | 5 | 10 | 2^5-1 | 2^{-10} | 31 | 2^{-10} |
| Q7.9 | 1 | 6 | 9 | 2^6-1 | 2^{-9} | 63 | 2^{-9} |
| Q8.8 | 1 | 7 | 8 | 2^7-1 | 2^{-8} | 127 | 3.9×10^{-3} |

Despite requiring more complex arithmetical circuits, floating-point formats are widely used in computers due to its flexibility, the wide range of representation, and accuracy.

A. Floating-point Format

In the IEEE 754 standard, a sFP number N is encoded in 32 bits with 1 *sign* bit (s), 8 bits for *exponent* (E), and 24 bits for *significand* precision (m), encoded as follow:

$$N = (-1)^s \times 2^E \times m \quad (1)$$

$$E = p - bias \quad (2)$$

where m has the form: $1.b_2b_{21}...b_0$ when ($254 \geq E \geq 1$), called normal encoding. If $E=0$, m assumes the form: $0.b_2b_{21}...b_0$, called subnormal (denormal) encoding. When $E=255$ and $m=0$ then N represents $\pm \infty$ (infinity); but if $E=255$ and $m \neq 0$ it is not a number (NaN). Once p may vary from 0 to 254, the normal mode may have $N_{\min} \approx \pm 1.17 \times 10^{-38}$ ($E = -126$) and $N_{\max} \approx \pm 1.7 \times 10^{+38}$ ($E = +127$). In subnormal mode ($E = -126$) and m may have all bits zero but $b_0=1$. It means that in subnormal the value of N may be: $N \approx -1^s \times 2^{-126} \times 2^{-22} \approx -1^s \times 2^{-149} \approx \pm 1.4 \times 10^{-45}$.

The hFP is defined in IEEE 754-2008 as the 16-bit base 2 format, having 1 *sign* bit (s), 5 bits for *exponent* (E), and 10 bits

for *significand* (m). The value N is calculated as in (1) and (2) with adaptations: the bias is +15; m has the form: $1.b_9b_8...b_0$ when ($30 \geq E \geq 1$), which is the normal encoding. If $E=0$, m has the form: $0.b_9b_8...b_0$, the subnormal encoding. When $E=31$ and $m=0$, N represents $\pm \infty$ (infinity); but if $E=31$ and $m \neq 0$ it is a NaN. Once p varies from 0 to 31, the normal mode may have $N_{\min} \approx \pm 6.1 \times 10^{-5}$ ($E = -14$) and $N_{\max} \approx \pm 65504$ ($E = +15$). In subnormal mode ($E = -24$) and m may have all bits zero but $b_0=1$; so, N may be: $N \approx -1^s \times 2^{-14} \times 2^{-10} \approx -1^s \times 2^{-24} \approx \pm 5.96 \times 10^{-8}$.

In order to convert from hFP to sFP, vice-versa, some transformations are required. It is out of the scope of this paper to discuss floating-point numeric conversions. We implemented converters in both, FPGA and C++ for simulating the SNNs. For our purpose herein, it is important to consider which range of values can be encoded in hFP, showed on table II.

TABLE II. 16-BITS HALF-PRECISION CAPABILITIES

| format | range | resolution | |
|-----------|--|------------|-----------------------|
| subnormal | $\pm \{5.96 \times 10^{-8}, 6.09 \times 10^{-5}\}$ | 2^{-24} | 5.96×10^{-8} |
| normal | $\pm \{0.5, 0.9995\}$ | 2^{-11} | 4.88×10^{-4} |
| normal | $\pm \{4, 7.996\}$ | 2^{-8} | 3.90×10^{-3} |
| normal | $\pm \{16, 31.984\}$ | 2^{-6} | 1.56×10^{-2} |
| normal | $\pm \{32, 63.968\}$ | 2^{-5} | 3.12×10^{-2} |
| normal | $\pm \{64, 127.937\}$ | 2^{-4} | 6.25×10^{-2} |
| normal | $\pm \{128, 255.875\}$ | 2^{-3} | 1.25×10^{-1} |

The videogame industry, graphic computing, among other sectors have used hFP for recording and transmitting data. Our primary aim on this investigation is to answer whether hFP is a suitable format for memorizing data in SNN simulations.

B. Analytical investigation

An initial analysis can be done over the equations used as a neuron model. However, there are complex neuron models based on dynamical systems that make similar analytical approach harder, if not impossible. In this sense, we have analyzed the implications of hFP numbers to the simplest neuron model, the LIF. The differential equation for this model is

$$\tau \cdot \frac{d}{dt} v = -(v - v_e) + RI(t) \quad (3)$$

where v is the membrane potential, τ is the time constant RC for the leaky integrator, R is the resistance while C is the membrane capacitance; v_e is the membrane equilibrium potential, and I is the sum of all input currents, coming from all spikes that reaches the neuron at a moment ' t '. In biological neurons R is typically $10 \text{ M}\Omega/\text{cm}^2$; C is typically $1 \mu\text{F}/\text{cm}^2$; which means for a typical $20 \mu\text{m}$ spherical cell the capacitance $C \approx 50 \text{ pF}$, $R \approx 200 \text{ M}\Omega$ (so, $\tau \approx 10 \text{ ms}$). For cells of different sizes, it is common to vary R from $1 \text{ M}\Omega$ to $200 \text{ M}\Omega$ or so; and C from 1 pF to 100 pF . Such values are out of the hFP range capabilities. However, it is possible to rescale the equation so that values can fit on the representation range of hFP.

We used the Euler's forward method for computing the differential equation (3), so the computing equation becomes:

$$v_{t+1} = v_t + \left((v_e - v_t) + RI(t) \right) / \tau \cdot h \quad (4)$$

where h is the time step (Δt) used for numerical integration. We used $h=0.001$, besides the lower the h the better the integration results at the cost of increasing computational steps. Now, let us analyze the unities: $v_e \approx -0.065$ V, and the maximum value for the membrane potential when the neuron fires is $v \approx +0.030$ V. The current inputs in the neurons are at the order of pA. Thus, a contribution to a EPSP may be: $\Delta v = R \cdot I \approx 100 \cdot 10^6 \times 100 \cdot 10^{-12} \approx 0,01$ V. If we multiply both sides of (4) by 1000 we scale the membrane potential to values that are better encoded in hFP. Moreover, we can previously solve a fraction $\lambda = h/\tau$, which may result a number in the range $\lambda=1,000$ for ($10^{-3}/10^6 \Omega * 10^{-12} F$), and $\lambda = 0.05$ for ($10^{-3}/200 * 10^6 \Omega * 100 * 10^{-12} F$), which fits well on hFP format. Thus, our equation for LIF becomes:

$$v_{t+1} = v_t + \left((v_e - v_t) + R I_{(t)} \right) \cdot \lambda \quad (5)$$

where λ is the previous calculation of (h/τ). All values are in mV, R is in M Ω , and the sum of currents I is in μA (no more in pA). In fact, because spikes are binary, EPSP/IPSP are calculated by the sum of the synaptic weights (w_{ij}) of all fired neurons. Thus, the EPSP or IPSP are calculated by:

$$\Theta_j = \sum_{k=1}^K w_{kj} \quad (6)$$

where Θ_j is the total excitatory or inhibitory current at the time 't' caused by all spikes coming from all K neurons fired at a past 't-n' ms; and w_{ij} is the synaptic weight from neuron i to j .

As the neuroscience points out [3], real neurons may have 10,000 connections or so. Let us then suppose 10,000 synapses are contributing to Θ_j at the time 't'. In order to cause an EPSP of 10 mV in a $R=100$ M Ω , the networks need $I=V/R = 10/100 = 0,1$ μA (values rescaled); but this is the contribution of 10,000 synapses, which means synaptic weights must be $w_{ij} \approx 10^{-5}$. The current vector I is always computed in sFP. However, synaptic weights changes as the network learns; thus, w_{ij} should be stored as hFP. As said, the minimum normalized value in hFP is $\pm 6.10352 \times 10^{-6}$, while in subnormal hFP it is 5.96046×10^{-8} . It seems enough for encoding 10^{-5} values required for 10,000 w_{ij} . But we can scale synaptic weights. Another extreme situation is when a single spike forces a neuron to fire, so $\Theta_j = 10$ mV, which means $I = \sum w_{ij} = 0,1$. Considering these cases, we can multiply w_{ij} by 1000, and then divide Θ_j by the same amount after calculating I . By doing so, the range for encoding synaptic weights is $\{100 \leq w_{ij} \leq 0,1\}$.

The pair of differential equations for Izhikevich's neuron model (IZH) are well known (see [14] [15]). The author used the Euler's forward method for computing the differential equations with a time step 1ms, the same we used above. Fortunately, he scaled the model equations, so the values are already in the same range we considered for the LIF model. We can use the same treatment used to encode synaptic weights in this model.

Therefore, a preliminary analysis shows us that it is possible to run a simulation converting values to hFP. Besides, in order to simulate a SNN we have to inject noise. We will see later that the amount of injected noise is much greater than the error caused by misrepresentation or truncation in the conversion from hFP to sFP and vice versa.

III. METHODS AND MATHERIAL

Firstly, we have created in FPGA a circuit that converts sFP to hFP numbers (S2H), and vice versa (H2S). We also have created functions in C++ that make the same conversions, so we test them on computers to make sure their results match.

In order to check the network functionality operating on both, sFP and hFP, we initially created a topology with all variables encoded in sFP. The original topology is saved with sFP codification for later recovering. Two networks with the same topology must have the same synaptic weights matrix and the same parameters for the neuron models. SNN with spike propagation delay (SNNwD) also requires the same propagation delay among the neurons in the two networks. To run under the same circumstances, each network received the same noise vector as they run for the same period of time, which is long enough for having stabilized raster plot pattern.

We run the original network on sFP format and the results are saved on files. Concurrently, we run the same network topology with variables and parameters converted to hFP, and the results are also saved. Then, we compared the results of both simulations. In this sense, the unique difference between two simulations is the data encoding format.

This is the standard procedure adopted for three methods we used for evaluating divergences among simulations, described further. These three methods are common in neurocomputing for testing how accurate is a neuron model. We adapted them for investigating variances emerging from equivalent networks simulated with differently encoded variables.

The kernel that realizes the network calculations are implemented on hardware, by floating-point unities on Intel's FPGA. We have also simulated the networks on C++ programs. As said, the networks are calculated with sFP numbers, but one of these simulations has its variables and results saved on hFP. Let us describe how it happens.

A. Memorizing vs Calculation

Although variables and results in half-precision networks are saved in hFP format, calculations are always realized using sFP 32-bits format, as settled in the IEEE 754-2008 standard. It means that, for instance, before calculating the membrane potential, the value of V_H stored at 't-1' as hFP is converted to V_S , a sFP number. The calculation is then realized and the result is converted back to hFP before storing V_H at the time 't'. This procedure is applied to every calculation, such as summation of the vector current I , changes in synaptic weights W , etc.

In FPGA, the conversion circuits are synthesized only on logic gates (combinational logic), so the conversion operation has no latency. As we have implemented pipelined circuits, it takes one clock pulse for latching the result; hence, the latency is one clock cycle per conversion. On Cyclone V SE 5CSEBA6U23I7 SoC/FPGA the conversion circuits use 12 logical blocks (ALM) and 16 registers. In C++, the functions take more time to perform conversions, but this does not significantly increase the simulation time. In addition, we use them only to validate the results obtained on hardware.

B. Curves $F \times I$ in Single Neurons

The Neuroscience literature points out three canonical methods for determining if a neuron model is a good approximation of the behavior of biological neurons.

One method is to inject a step of constant direct current in the neuron model. After a stabilization interval the observer can measure the firing frequency of the neuron, then plot the obtained frequencies as a function of injected currents; the so called $F \times I$ curve. We used this method in order to compare the $F \times I$ curve obtained by the network encoded on sFP and hFP formats. Fig.1 and Fig.2 show the curves obtained for the LIF and IZH neuron models respectively.

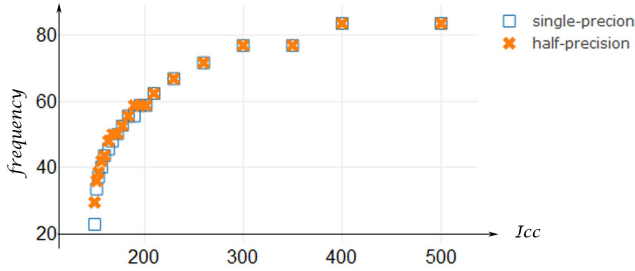


Fig. 1. $F \times I$ curves for LIF model. A step DC current from 150 to 500 pA was injected in the neurons for 5.2 seconds. Frequency was measured from the time interval between 0.2 to 5.2 sec. One curve shows the values obtained with the network simulated with sFP, the other with hFP. Values are nearly identical.

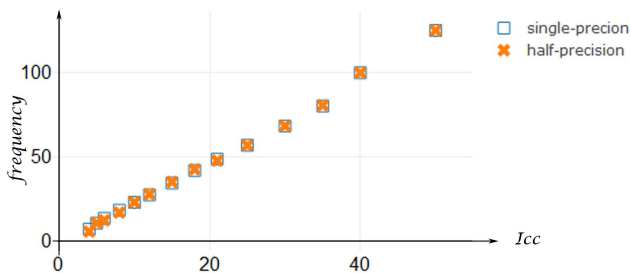


Fig. 2. $F \times I$ curves for IZH model. A step DC current from 4 to 50 pA was injected in the neurons for 5.2 seconds. Frequency was measured from 0.2 to 5.2 sec. One curve shows the frequencies obtained with the network simulated with sFP and the other with hFP numbers. Equally, values are almost identical. For details of the parameters used in the neural model, please refer to the text..

For LIF model we used the following parameters: membrane resistance 100 M Ω , capacitance 50 pF, membrane equilibrium potential -65 mV, recovering potential -70 mV, threshold potential -52 mV and absolute refractory period 10 ms. For IZH model: $a = 0.02$, $b = 0.2$, $c = -65$ mV, $d = 6$ and the $Vr = -70$ mV for excitatory neurons. For all inhibitory neurons: $a = -0.02$, $b = -1$, $c = -60$ mV, $d = 8$ and $Vr = -63.8$ mV.

The curves are nearly identical, with small deviations due to the method used on frequency calculation, since we extracted a mean over only 5 seconds. Frequency can be obtained with more accurate values if longer simulation periods are used.

C. Subthreshold behavior

Another classical method to determine if a neuron model is accurate is to observe if the voltage membrane at the

subthreshold regime is alike to the real neurons. We injected a sinusoidal current of 120 pA (peak-to-peak) added to a 20 pA random noise on a LIF network simulated with sFP numbers. This signal puts the neuron on subthreshold regime and, eventually, noise can cause a spike. The same current vector, converted to hFP numbers, was injected in a network with similar topology. Then, we compared if the membrane voltage on randomly selected neurons are similar on both simulations. Fig. 3 shows one result for LIF neuron.

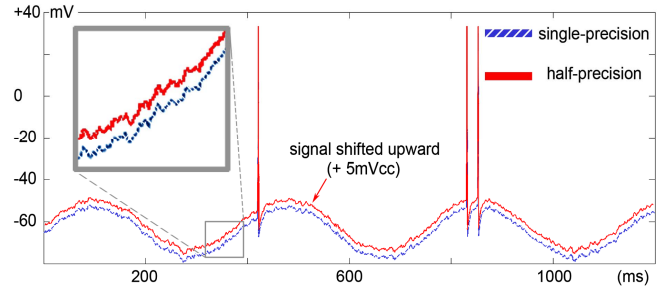


Fig. 3. Subthreshold Response for hFP and sFP. A sinusoidal current (120 pA peak-to-peak) with a noise (20 pA normal distribution) is injected to LIF neurons. This current alone normally cannot cause a spike, so at 420 and 830 ms we forced two spikes. The intention is to observe the membrane potential V at the subthreshold regime and after firing a spike. Note that a spontaneous spike due to noise repeated in both simulations at ≈ 850 ms. The hFP signal was shifted upward in the graph by $+5$ mV for visualization purpose.

The voltage membrane in both simulations are practically identical; thus, we shifted upward the hFP signal for better visualization. We forced two spikes at 420 ms and 830 ms in both simulation for observing the post-synaptic membrane response. Note that a spontaneous spike caused by noise at ≈ 850 ms appeared in both simulations. The results are quite similar for IZH model, so the graphic was omitted.

These results can be explained because the encoding errors on values at this range is quite low (see table II). For instance, for a voltage of -65 mV, the encoding error is 2^{-4} , or 0.065 mV; resulting -65.06 mV or -65.04 mV. The injected sinusoidal current (120 pA) cause a 12 mVpp EPSP/IPSP that is summed to 2 mVpp (peak-to-peak) of noise. Considering the ratio of the noise by the truncate error:

$$\varepsilon = \frac{2.0 \text{ mVpp}}{0.065} = 30.7$$

what means that the injected noise is about 30 times greater than the encoding error caused by the conversions of sFP to hFP. However, it must be considered that the encoding error may have cumulative effects, positive and negative, and noise may compensate possible influences on successive operations. Due to that, only simulations can show the influence of successive conversions over the networks. For this reason, we used massive simulation applied to the third method.

D. Predicting When the Next Spike Occurs

Being able to predict when a neuron will fire is a good indication that the neural model faithfully reproduces the natural phenomenon. This is the third method usually used for testing neuron models: prediction of when the next spike occurs. We

adapted this method for measuring if a neuron spike occurs at the same time on SNN simulated in sFP and hFP.

Fig.4 shows two simulations for the same topology using the "neural assembly computing" approach [16], [17] in an IZH network. For this topology, a set of neurons (A) are fully connected to another set (B) which are also fully connected to the set (A). Let us call them assemblies A and B . Considering this, when the A neurons fire together their spikes take certain time and reach the B neurons, so the B neurons also fire nearly simultaneously. After firing, spikes from B travels to the A neurons in a certain time interval, so neurons on A fires almost together again; and the cycle repeats. After a stabilization period, it is expected that firing time interval between assemblies remain nearly the same and repeats indefinitely.

At the top of Fig. 4 we see the hFP raster, while at the bottom we see the raster of the sFP simulation. We then choose two bin windows (Bin 1 and Bin 2) and compare the time in which a specific neuron fired for hFP and for sFP simulation. It is not expected that they fired exactly at the same time due to the noise, which may shift the individual firing time.

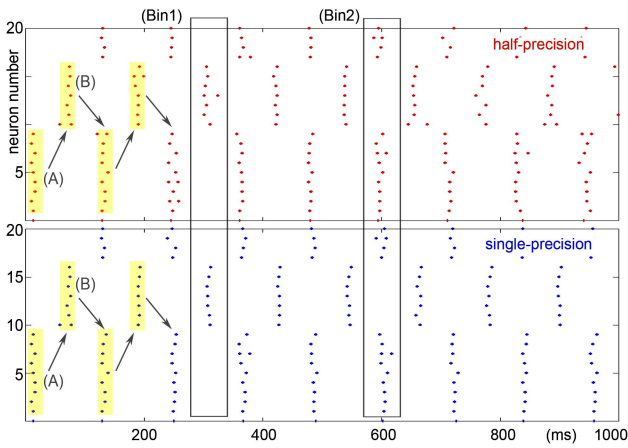


Fig. 4. Raster plot of half- vs single-precision firing time. Raster-plot of 20 neurons in a SNNwD. On top the hFP and below the sFP simulation of the same topology. We analyzed the firing time differences for the same neuron on each simulation. Note that the general firing pattern remains the same throughout the simulation and small differences are mostly due to noise.

It is important to note that the primary network topology has propagation delays randomly distributed, which spread in time the spikes from an assembly to another, called *polychronization* [18]. However, once defined the primary topology the time intervals are the same for simulations on sFP and hFP formats.

As said, noise cause spike time shifting of few milliseconds. In fact, important in these configuration is to predict that the whole assembly A and the assembly B will fire at regular time interval, which guarantees that the network is executing what is expected. We work mostly with spike coincidence detection and the simulations with different encoding formats showed no relevant differences in results. However, the choice of parameters for the neuron model greatly influences the results. For instance, the absolute refractory period is important when using *polychronous* neural assemblies. Neurons with short recovering time can fire bursts due to multiples temporally shifted excitations coming from other assemblies. A burst-firing

neuron on an assembly A may cause a burst-firing pattern on neurons on an assembly B , so a positive feedback may cause the whole network to fire disorderly. In this sense, the choice of the correct parameters for neurons are much more relevant than the variables encoding formats.

IV. RESULTS AND DISCUSSIONS

A mathematical analysis revealed that it is possible to use hFP format to store data, variables and parameters for SNN and SNNwD. However, it is analytically impossible to predict the behavior of a fully connected neural network just by analyzing the equations of the neuron model. Thus, we have adapted three canonical methods used for evaluating neuron models for investigating the behavior of the network simulated on hFP and sFP formats.

A. Results and Interpretation

The curves $F \times I$ of both neurons (LIF & IZH) are almost identical for hFP and sFP, both simulated on hardware and software. It indicates that to convert $V_{[i]}$ to hFP, to store it, to reconvert the value for sFP, and then to calculate the next iteration do not affect the final firing frequency. In the case of IZH model the value of $U_{[i]}$ is also converted and restored to single-precision float. However, this procedure only guaranties that hFP does not affect the ICC operation regime.

The subthreshold regime analysis showed that the process of converting and restoring hFP format does not affect the subthreshold values of the membrane potential, because the injected noise has an impact ≈ 30 times greater on $V_{[i]}$ (and $U_{[i]}$ for IZH model) than the encoding error at this numerical range.

Predicting the time for next spike is important because it shows if the conversion to hFP format changes the timing for occurring the next spike. This method shows the network dynamics. In order to obtain the temporal dynamics, we turned to *neural assembly computing* approach. If one set of neurons triggers another set that triggers the first set back, we can expect reverberating operation. Thus, the time each neuron set fire must remain somehow constant. The raster plot in Fig.4 shows a simulation of 20 neurons, but we have also investigated the dynamics for 200 neurons. The results show some shift in individual firing time due to noise injection, but the network firing pattern remain the same. It means that converting to the hFP format then converting back to the sFP format does not alter the network firing pattern significantly.

As expected, no discrepancies occurred between the values calculated in hardware (FPGA) and in software (C++ programs).

B. Advantages and Viability

There are several gains on using hFP to encoding variables and parameters when simulating SNN. At the expense of numeric precision and operational range, hFP has some advantages over 32-bit sFP binary formats, mainly concerning memory and bandwidth requirements. Consider a SNN with n neurons fully connected that requires a 2D array of $W_{n \times n}$ for encoding synapse among n pre-synaptic neurons to n post-synaptic neurons. For simulating spike-timing dependent plasticity (STDP) this network also requires at least another 2D

array. Another 2D array if the network takes into account the synaptic homeostasis. Every additional feature simulated on the network demands memory and data flow. It may become prohibitive to simulate SNN in sFP format with several features on embedded systems, or even in real-time dedicated hardware. Converting data to hFP may be a solution.

For the case we are focused on, the benefit comes more from bandwidth. A real-time SNN hardware with propagation delay requires temporal convolution for checking which spike reaches each neuron at certain time. It means high data transfer rates from and to the memory. If it is possible to transfer two hFP values in a 32-bits bus, instead of a single sFP value, the required bandwidth falls to the half. It may be translated also to bus requirements, to the amount of pin connections, etc. It also means that the circuit can operate at the half of the clock speed, possibly a clock speed that fits to lower-price FPGAs, resulting also on less power consumption.

V. CONCLUSIONS

Spiking Neural Networks have been simulated in software with floating-point numbers for long time. For the current technologies, it is difficult to simulate real-time large-scale networks calculated by single-precision floating-point (sFP) on computers. One solution is to design dedicated processors with fixed-point numbers. The drawback is that these are ad-hoc solutions; since these encoding schemes have no standards. However, circuits for calculating floating-point arithmetic are currently efficient and cost-effective. One problem is that values use 32-bits memories, larger buses, or more clock cycles for serial conveying. A solution may be to store real values on half-precision 16-bit floating-point (hFP) data format.

In this paper we analyzed the behavior of SNNs calculated on sFP format, but that have data storered on the hFP format. We conclude that the Izhikevich's neuron model can be directly implemented in hFP, since the model was scaled to have variables and parameters in numeric ranges that fit on hFP format. On the other hand, the LIF model must have the equations scaled to fit the values of the variables in a numeric range compatible with the half-precision floating point format.

All calculations in the SNN are still performed in the sFP format. Hence, conversions between formats must be done in, be it in digital circuits (hardware) or by software functions. Using three methods, we tested whether converting values into sFP and converting the results back to hFP would affect the SNN operation and dynamics. The data revealed that no relevant distortion occurs; thus, it is possible to store and exchange SNN data in hFP format.

Many benefits can arise from the use of 16-bit floating instead of 32-bit, the most important are less memory usage, simple buses, lower clock frequency, among others. We are focused on the implementation of SNN in the FPGA, in which case the benefits of using hPF for storage and data exchange overtake the cost of the conversion hardware.

REFERENCES

- [1] R. Brette *et al.*, "Simulation of networks of spiking neurons: a review of tools and strategies," *J. Comput. Neurosci.*, vol. 23, no. 3, 2007.
- [2] H. Paugam-Moisy and S. Bohte, "Computing with spiking neuron networks," in *Handbook of Natural Computing. 1st Edition*, vol. 1, G. Rozenberg, T. Bäck, and J. N. Kok, Eds. Heidelberg, Germany: Springer-Verlag, 2010, pp. 1–47.
- [3] E. R. Kandel, J. H. Schwartz, and T. M. Jessel, *Principles of Neural Science. 4th ed.* New York, NY: McGrall-Hill Health Prof. Division, 2000.
- [4] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE Trans. Neural Networks*, vol. 15, no. 5, pp. 1063–1070, 2004.
- [5] D. Thomas and W. Luk, "FPGA Accelerated Simulation of Biologically Plausible Spiking Neural Networks," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, 2009, pp. 45–52.
- [6] P. O'Connor, D. Neil, S. C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," *Front. Neurosci.*, no. 7 OCT, 2013.
- [7] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing*, vol. 74, no. 1–3, 2010.
- [8] K. L. Rice, M. A. Bhuiyan, T. M. Taha, C. N. Vutsinas, and M. C. Smith, "FPGA Implementation of Izhikevich Spiking Neural Networks for Character Recognition," in *2009 International Conference on Reconfigurable Computing and FPGAs*, 2009, pp. 451–456.
- [9] D. Pani, P. Meloni, G. Tuvèri, F. Palumbo, P. Massobrio, and L. Raffo, "An FPGA Platform for Real-Time Simulation of Spiking Neuronal Networks.," *Front. Neurosci.*, vol. 11, p. 90, 2017.
- [10] J. Sjöström and W. Gerstner, "Spike-timing dependent plasticity," *Scholarpedia*, vol. 5, no. 2, p. 1362, 2010.
- [11] D. E. Feldman, "The Spike-Timing Dependence of Plasticity," *Neuron*, vol. 75, pp. 556–571, 2012.
- [12] T. Keck *et al.*, "Integrating Hebbian and homeostatic plasticity: the current state of the field and future research directions," *Philos. Trans. R. Soc. B Biol. Sci.*, vol. 372, no. 1715, 2017.
- [13] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: single Neurons, Populations, Plasticity*. Cambridge, UK: Cambridge Univ. Press, 2002.
- [14] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.
- [15] E. M. Izhikevich, *Dynamical Systems in Neuroscience: the Geometry of Excitability and Bursting*. Cambridge, MA: The MIT Press, 2007.
- [16] J. Ranhel, "Neural assembly computing," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 23, no. 6, 2012.
- [17] J. Ranhel, M. Lobo Netto, and E. Del Moral Hernandez, "How complex behavior emerge from spikes," in *Cognitive Science: Recent Advances and Recurring Problems*, 1st ed., J. . Adams, Frederick; Pessoa, O. Jr; Kogler, Ed. Wilmington, USA: Vernon Press, 2017, pp. 197–218.
- [18] E. M. Izhikevich, "Polychronization: computation with Spikes," *Neural Comput.*, vol. 18, no. 2, pp. 245–282, 2006.