Proposta de Implementação em Hardware de SVM multi-kernel para Aplicações em IoT

Luiz F. Carbonera Filho [*] **Maria G. F. Coutinho** [*]
Marcelo A. C. Fernandes [*,**]

[*] *Laboratório de Aprendizagem de Máquina e Instrumentação Inteligente, nPITI/IMD, UFRN, Natal, RN, Brasil (e-mail:luiz.carbonera.100@ufrn.edu.br, gracielly@dca.ufrn.br).*
[**] *Departamento de Engenharia de Computação e Automação, UFRN, Natal, RN, Brasil (e-mail: mfernandes@dca.ufrn.br).*

Abstract: This work aims to propose a hardware implementation of a machine learning technique (ML) for applications in the internet of things (IoT). The technique in question is the support vector machine (SVM), which are ML algorithms efficient in solving classification and regression problems. This work aims to develop a proposal for dedicated hardware for *multi-kernel* SVM and show its potential concerning processing time and power consumption about solutions associated with embedded systems in microcontrollers. The article presents a reference implementation of SVM on field-programmable gate array (FPGA) and compares processing time and consumption results with other implementations. The results show that the SVM in dedicated hardware, implemented in FPGA, has better consumption and processing time than its versions in microcontrollers (MCU).

Resumo: Este trabalho tem como objetivo uma proposta de implementação em Hardware de uma técnica de aprendizagem de máquina (*machine learning* - ML) para aplicações em internet das coisas (*internet of things* - IoT). A técnica em questão envolve uso das máquinas de vetores de suporte (*support vector machine* - SVM), estes sendo algoritmos de ML eficientes na resolução de problemas de classificação e regressão. O objetivo do trabalho é desenvolver uma proposta de hardware dedicado para SVM *multi-kernel* e mostrar seu potencial em relação ao tempo de processamento e ao consumo energético frente a soluções associadas com sistemas embarcados em microcontroladores. O artigo apresenta uma implementação de referência do algoritmo SVM em *field-programmable gate array* (FPGA) e compara os resultados de tempo de processamento e consumo entre diferentes implementações. Os resultados mostram que o algoritmo SVM em hardware dedicado, implementado em FPGA, possui melhores resultados de consumo e tempo de processamento que suas versões em microcontroladores (MCU).

Keywords: Machine learning; SVM; IoT; Hardware; FPGA.
Palavras-chaves: Aprendizagem de máquina; SVM; IoT; Hardware; FPGA.

1. INTRODUÇÃO

O avanço da Internet das coisas (*internet of things* - IoT) tem-se mostrado amplamente difundido nas casas, indústrias e cidades. As aplicações que possuem elementos de IoT tornaram-se parte do cotidiano da sociedade. Por sua vez, a inteligência artificial (*Artificial Intelligence* - AI) vem ganhando cada vez mais destaque em diversas pesquisas nos últimos anos. A integração entre essas duas importantes áreas emergentes, IoT+AI, garante amplas possibilidades de aplicação e desenvolvimento tecnológico.

Diversos trabalhos da literatura exploram as vantagens desta associação IoT+AI (Osuwa et al., 2017), como é o caso de trabalhos envolvendo identificação de ruídos anômalos (Janjua et al., 2019; Gupta et al., 2020), identificação e predição de doenças (Ganesan and Sivakumar, 2019; Otoom et al., 2020; Gunawan et al., 2018), monitoramento de segurança dos dispositivos utilizados nas aplicações IoT (Liu et al., 2019; Wang et al., 2019), entre vários outros.

Dentre os algoritmos de AI, as técnicas de aprendizagem de máquina (*machine Learning* - ML) tornaram-se um campo importante de discussão devido o equilíbrio entre complexidade de implementação, recursos utilizados e resultados encontrados para as tecnologias de IoT (IoT+ML). As máquinas de vetores de suporte (*support vector machine* - SVM) são algoritmos de ML eficientes na resolução de problemas de classificação e regressão. É possível encontrar na literatura trabalhos de diferentes áreas que exploram implementações de SVM em sistemas IoT (Liou et al., 2018; Liu et al., 2019; Otoom et al., 2020; Ganesan and Sivakumar, 2019; Jain et al., 2019; Orrù et al., 2020; Ramadurgam and Perera, 2021).

Trabalhos envolvendo o uso de SVMs para predição de doenças são encontrados em Liou et al. (2018); Gunawan

et al. (2018); Ganesan and Sivakumar (2019); Otoom et al. (2020). O trabalho de Liou et al. (2018) realiza classificação de sinal biomédico em tempo real através de algoritmos SVMs implementados em uma plataforma IoT. No trabalho de Gunawan et al. (2018) foi desenvolvido um sistema para detecção e monitoramento do nível de hidratação da urina baseado na utilização de um sensor de cor e da técnica SVM para a classificação do nível de hidratação. Um microcontrolador Arduino Uno foi utilizado para a construção do dispositivo.

Em Ganesan and Sivakumar (2019) foram utilizados sensores de saúde combinados com um dataset de doenças cardíacas para o desenvolvimento de um sistema de predição de doenças cardíacas baseado em ML. Os resultados foram obtidos para quatro diferentes técnicas de ML, incluindo a SVM, que obteve o segundo melhor resultado entre elas. Em Otoom et al. (2020) é apresentado um framework baseado em IoT para identificação e monitoramento de casos de COVID-19. O framework coleta dados de sintomas dos usuários em tempo real para identificação dos casos suspeitos. Os experimentos foram realizados utilizando oito algoritmos de ML, entre eles o SVM, que esteve entre os algoritmos que obtiveram os melhores resultados.

No entanto, há dois fatores importantes a serem considerados durante a implementação de sistemas IoT: a latência e o consumo energético dos dispositivos envolvidos. É importante observar também que a utilização de algoritmos de AI+ML, geralmente, são computacionalmente custosos, o que pode gerar uma maior dificuldade em lidar com a questão do tempo de resposta da aplicação. Diante disso, algumas alternativas vêm sendo exploradas para acelerar implementações de AI+ML em sistemas IoT, bem como possibilitar a redução do consumo de energia.

Exemplo de aplicações em que a baixa latência pode significar redução de custos para indústrias é a predição de falhas em seus equipamentos. Em Dong et al. (2014), é abordado um método que combina SVM e o modelo de Markov para alertar sobre a degradação de rolamentos, onde a variável de análise é a vibração do equipamento. Já em Orrù et al. (2020), abordagens de SVM e *multilayer perceptron* (MLP) foram analisadas para predizer falhas em bombas centrífugas nas indústrias de gás e óleos a partir de medidas de sensores no equipamento. Outro exemplo é observado em Gao and Hou (2016), onde a técnica SVM é utilizada para apontar falhas em maquinários da linha de produção de processos químicos, tendo como referência a simulação da planta industrial Tennessee Eastman process (TEP). Jain et al. (2019) apresenta uma abordagem para predição automática de falhas, onde o modelo de SVM é elaborado com base na análise dos dados atuais e histórico do dispositivo, por consequência, das características mais relevantes para identificar uma quebra de sistema.

Uma alternativa que vem se mostrando promissora para alcançar baixa latência e consumo é a utilização de implementações em hardware reconfigurável, utilizando field-programmable gate array (FPGA), como apresentado em Lopes et al. (2019); Minamisawa et al. (2019); Dias et al. (2021); Silva et al. (2020); Guimarães and Fernandes (2020); da S. Medeiros et al. (2020); Coutinho et al. (2019); Noronha et al. (2019). Nesta abordagem, é possível alcançar o equilíbrio entre alta velocidade de processamento, ou *throughput*, e baixo consumo de energia, o que pode ser crucial em diversas aplicações de IoT.

A implementação em FPGA do Sparse Ising Model, que possui aplicações em áreas como neurociência e economia, demonstrado em Minamisawa et al. (2019), apresentou alta velocidade para encontrar as respostas a partir do uso dos algoritmos de Travelling Salesman Problem (TSP) e SVM. No trabalho de Lopes et al. (2019) é apresentada uma implementação paralela de SVM em FPGA, utilizando o algoritmo Descida do Gradiente Estocástico (Stochastic Gradient Descent - SGD). A proposta foi capaz de alcançar um speedup de mais de 10.000× em comparação com uma implementação em software, e até mais de 300× em comparação com outras implementações de SVM em FPGA do estado da arte.

Assim, este trabalho tem como objetivo desenvolver uma proposta de hardware dedicado de um algoritmo SVM *multi-kernel* e mostrar seu potencial em relação ao tempo de processamento e ao consumo energético frente às soluções associadas com sistemas embarcados em microcontroladores (MCUs). Através disso, objetiva-se viabilizar a utilização da técnica SVM em aplicações IoT. O artigo apresenta uma implementação de referência da fase de inferência do SVM em FPGA e compara características de tempo de processamento e consumo com outras implementações em MCUs.

2. *SUPPORT VECTOR MACHINE* - SVM

As máquinas de vetores de suporte consistem em um método supervisionado de aprendizagem de máquina, capaz de realizar classificação. Também é possível empregar esta técnica em problemas de regressão. Tradicionalmente, uma SVM realiza a classificação binária, ou seja, entre duas classes. No entanto, é possível empregar várias SVMs em conjunto para efetuar a identificação de várias classes.

A Figura 1 ilustra a arquitetura de uma SVM de classificação binária, implementada neste trabalho, o qual possui como entrada um vetor $\mathbf{x}$ de dimensão P expresso como

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_P \end{bmatrix} \tag{1}$$

onde cada i-ésima entrada, x_i é aplicada na camada de funções de *kernel* que, por sua vez, calcula a similaridade entre o vetor de entrada, $\mathbf{x}$, e cada m-ésimo vetor centro, $\mathbf{c}_m$ expresso como

$$\mathbf{c}_m = \begin{bmatrix} c_{1,m} \\ \vdots \\ c_{P,m} \end{bmatrix}. \tag{2}$$

A quantidade de funções de *kernel* é definida por M, que representa o número de vetores de suporte extraídos dos dados de treinamento pela solução do problema. Diferentes funções de *kernel* podem ser utilizadas no treinamento da SVM, as mais comuns são as funções de *kernel* linear, polinomial e gaussiana. Neste trabalho, implementou-se uma SVM de classificação binária utilizando a função linear e a função polinominal de grau 3. A função de *kernel* linear pode ser expressa como

$$K(\mathbf{x}, \mathbf{c}_m) = \mathbf{x}^T \mathbf{c}_m. \tag{3}$$

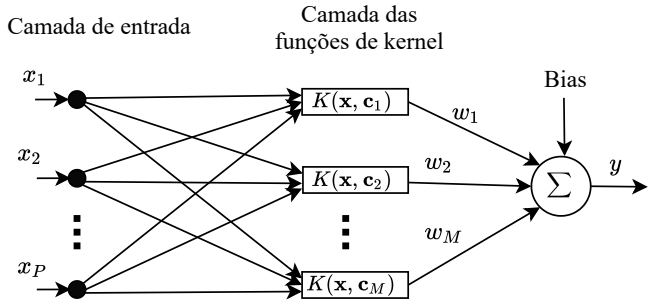

Figura 1. Arquitetura de uma SVM para classificação binária.

Já a função polinomial é definida por

$$K(\mathbf{x}, \mathbf{c}_m) = (\mathbf{x}^T \times \mathbf{c}_m + 1)^q. \tag{4}$$

onde q corresponde ao grau do polinômio.

Os M valores retornados da camada de funções de *kernel* são caracterizados pelo vetor $\mathbf{l}$ expresso como

$$\mathbf{l} = \begin{bmatrix} l_1 \\ \vdots \\ l_M \end{bmatrix} = \begin{bmatrix} K(\mathbf{x}, \mathbf{c}_1) \\ \vdots \\ K(\mathbf{x}, \mathbf{c}_M) \end{bmatrix} \tag{5}$$

A saída da SVM é expressa pela equação

$$y = \mathbf{l}^T \mathbf{w} \tag{6}$$

onde $\mathbf{w}$ é o vetor de pesos que é expresso como

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_M \end{bmatrix}. \tag{7}$$

3. MODELO DE REFERÊNCIA EM HARDWARE

A arquitetura geral da SVM proposta neste trabalho encontra-se ilustrada na Figura 2. A estrutura proposta é composta por dois módulos principais: módulo de funções de *kernel* e módulo de camada de saída. O hardware foi projetado para trabalhar com um tempo de amostragem t_s, no qual a cada n-ésimo instante de tempo t_s um vetor de entrada $\mathbf{x}(n)$ é processado, gerando uma saída $y(n)$. O *throughput* pode ser expresso como $1/t_s$ amostras por segundo (*samples per second* - SPS).

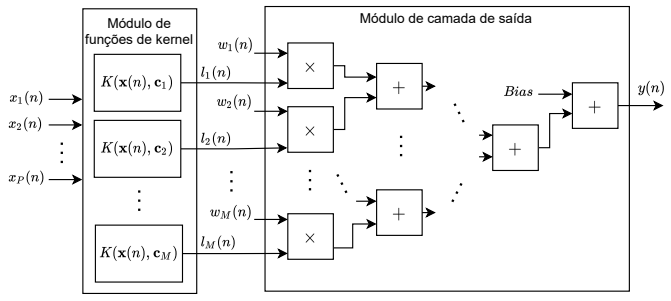

Figura 2. Arquitetura geral do hardware proposto para a SVM.

A proposta foi projetada para trabalhar no formato ponto flutuante de 32 bits objetivando uma comparação justa com as soluções em sistemas embarcados utilizando implementações em linguagem C. Cada bloco de multiplicação foi projetado para realizar a operação em 5 ciclos de *clock*.

No caso dos blocos de adição, cada um foi projetado para realizar sua operação em 7 ciclos de *clock*. É importante destacar que a implementação em hardware proposta neste trabalho utiliza um estrategia chamada de *full parallel*, onde tenta-se obter uma paralelização máxima de todas as operações Dias et al. (2021); Coutinho et al. (2019).

O módulo funções de *kernel*, executa os M *kernels* em paralelo gerando o vetor $\mathbf{l}(n)$ em cada n-ésimo instante de tempo. No caso do *kernel* linear, expresso pela Equação 3, foram utilizados blocos de multiplicação em paralelo, para realizar a operação entre as entradas e os centros encontrados na fase de treinamento, e uma estrutura de blocos de adição em cascata, para unir o resultados de acordo com a Equação 3. A Figura 3 apresenta a proposta de referência em hardware para o *kernel* linear. Como os multiplicadores estão em paralelo, o tempo de execução do *kernel* linear em ciclos *clock*, $N_{clock}^{\text{Linear}}$, pode ser expresso como

$$N_{clock}^{\text{Linear}} = 5 + 7 \times \lceil \log_2(P) \rceil. \tag{8}$$

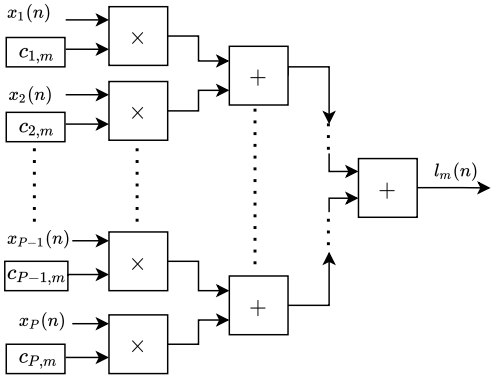

Figura 3. Estrutura geral do *kernel* Linear.

A construção do *kernel* polinomial, expresso pela Equação 4, foi realizada seguindo três etapas: a multiplicação dos centros pelas entradas, a soma desses resultados e da constante 1, e a potenciação desse resultado pela ordem desejada (q). A Figura 4 detalha o *kernel* polinomial no qual o produto interno da entrada, $\mathbf{x}$, pelo m-ésimo centro, $\mathbf{c}_m$, é semelhante ao *kernel* anterior. Já a potenciação é realizada como $\lceil \log_2(q) \rceil$ multiplicadores em cascata. O tempo de execução do *kernel* polinomial em ciclos *clock*, N_{clock}^{Poli}, pode ser expresso como

$$N_{clock}^{\text{Poli}} = 5 + 7 \times \lceil \log_2(P) \rceil + 7 + 5 \times \lceil \log_2(q) \rceil. \tag{9}$$

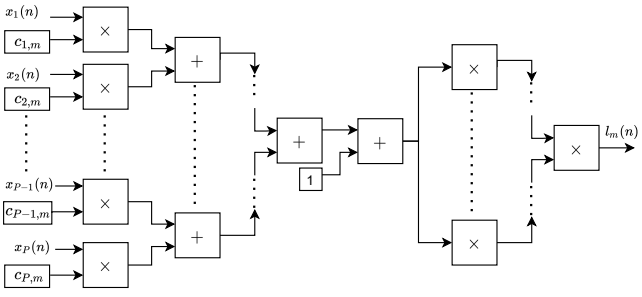

Figura 4. Estrutura geral do *kernel* polinomial.

O módulo de camada de saída, como apresentado na Figura 2, faz o produto interno entre a saída dos *kernels*, $\mathbf{l}$

e o vetor de pesos $\mathbf{w}$, de acordo com a Equação 6. O tempo de execução do módulo de camada de saída em ciclos *clock*, $N_{clock}^{\text{OutLayer}}$, é expresso como

$$N_{clock}^{\text{OutLayer}} = 5 + 7 \times \lceil \log_2(M) \rceil + 1. \quad (10)$$

Assim, o tempo total de processamento da SVM proposta em ciclos de *clock*, pode ser caracterizado como

$$N_{clock}^{\text{SVM}} = N_{clock}^{\text{OutLayer}} + \begin{cases} N_{clock}^{\text{Linear}}, \text{ se o } \textit{kernel} \text{ é linear} \\ N_{clock}^{\text{Poli}}, \text{ se o } \textit{kernel} \text{ é polinomial.} \end{cases} \quad (11)$$

4. MATERIAIS E MÉTODOS

4.1 Dataset *utilizado*

Para validar o funcionamento do protótipo da fase de inferência da SVM em hardware, foi utilizado o *Dataset* das flores Iris (*Iris Data Set*), introduzido por Fisher (1936) e Anderson (1936). Esse conjunto possui quatro características como entradas (*sepal length*, *sepal width*, *petal length* e *petal width*) e três diferentes classes de flores como saída (Virginica, Versicolor e Setosa). Entretanto, para simplificar a implementação, foram utilizadas apenas duas entradas, $P = 2$, (*petal length* e *petal width*) e duas classes com saída (Virginica e Versicolor).

4.2 Parâmetros e implementação na plataforma Quartus

O modelo de referência para inferência da SVM em hardware apresentado na seção anterior foi implementado na plataforma de desenvolvimento Quartus Prime Lite Edition, versão 16.1. Foi implementada uma estrutura utilizando os parâmetros apresentados na Tabela 1. O treinamento da SVM foi realizado previamente utilizando o algoritmo *sequential minimal optimization* (SMO) (Noronha et al., 2019) no software MATLAB (License number: 596681).

Tabela 1. Parâmetros utilizados para a implementação em hardware do modelo de referência da SVM, apresentado na Seção 3.

Parâmetro	Valor
Número de entradas (P)	2
Número de *kernels* lineares (M)	16
Número de *kernels* polinomiais (M)	6
Grau do *kernel* polinomial (q)	3
Representação numérica	Float 32 bits

As Figuras 5 e 6 ilustram a implementação dos *kernels* linear e polinomial no Quartus Prime, respectivamente. Em ambos os casos, os *kernels* construídos foram replicados de acordo com o número de kernels especificados na Tabela 1.

4.3 Metodologia para obtenção dos resultados

Para armazenar os valores associados as amostras do *Dataset* da Iris, que serão classificadas pela SVM em hardware, foram utilizados componentes de memória to tipo *read-only memory* (ROM). Foram utilizadas 50 amostras do *dataset* para validar a implementação em hardware e devido a isto foram necessárias a utilização de memórias ROM com 64 endereços de 32 bits cada. A Figura 7 ilustra a estrutura de envio das entradas. Além do resultado final, cada sinal transmitido foi derivado para ser saída, para fins de análise da simulação.

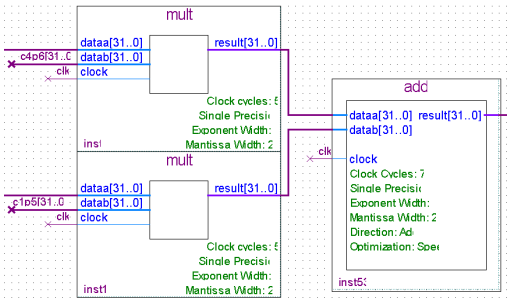

Figura 5. Estrutura do *kernel* Linear implementado no Quartus Prime.

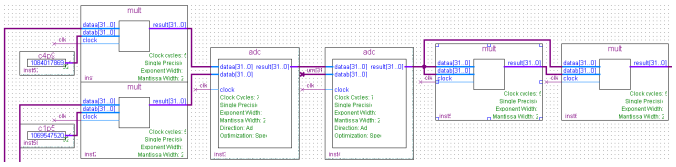

Figura 6. Estrutura do *kernel* Polinomial de grau $q = 3$ implementado no Quartus Prime.

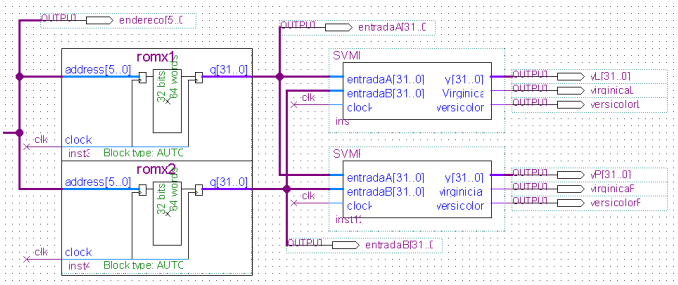

Figura 7. Estrutura de envio das entradas para a SVM via memórias ROM.

5. RESULTADOS

5.1 Resultados de síntese

O processo de síntese foi realizado com intuito de obter os resultados de alocação de recursos em hardware com base na Tabela 1. Os resultados estão apresentados na Tabela 2. O motivo de escolha do dispositivo FPGA deu-se pela disponibilização de multiplicadores internos. Somente dispositivos com pelo menos 532 multiplicadores de 9-bits embarcados poderiam ser utilizadas na implementação da arquitetura proposta. Para reduzir o uso de multiplicadores internos, basta-se escolher a opção de construir os multiplicadores através elementos lógicos e registradores. Isso aumentaria o número de elementos lógicos e registradores utilizados, mas possibilitaria o uso de uma placa com menor quantidade de multiplicadores internos.

A frequência máxima de operação, $f_{\max}$, do protótipo baseado nos parâmetros apresentados na Tabela 1 foi de $f_{\max} = 83{,}02\,\text{MHz}$ para o caso em que a temperatura alcance 100 graus Celsius. Portanto, o tempo crítico da aplicação foi de 12,045 ns (nanossegundos). A Tabela 3

Tabela 2. Resultados da alocação de recurso no FPGA, após o processo de síntese.

Família do Dispositivo	Cyclone IV E
Modelo	EP4CE115F29I8L
Total de elementos lógicos	56.977/ 144.480 (50%)
Total de registradores	28095
Total de pinos	146 / 529 (28%)
Total de bits de memória	6.912 / 3.981.312 (<1%)
Total de multiplicadores de 9 bits embarcados	441 / 532 (83%)

Tabela 3. Tempo de execução em ciclos de *clock* com base nos parâmetros da Tabela 1 e Equação 11.

Variável	*Kernel*	Ciclos de *clock*
$N_{clock}^{\text{Linear}}$	Linear ($P = 2$)	12
N_{clock}^{Poli}	Polinomial ($P = 2$, $q = 3$)	29
$N_{clock}^{\text{OutLayer}}$	Linear ($M = 16$)	34
$N_{clock}^{\text{OutLayer}}$	Polinomial ($M = 6$)	27
N_{clock}^{SVM}	Linear	46
N_{clock}^{SVM}	Polinomial	56

apresenta o tempo de execução em ciclos de *clock* com base nos parâmetros da Tabela 1 e Equação 11.

Baseado no valor da frequência máxima de operação, $f_{\max}$, pode-se estimar o *throughput*, th, pela equação expressa como

$$th = \frac{1}{t_s} = \frac{f_{\max}}{N_{clock}^{\text{SVM}}} \tag{12}$$

Considerando o maior valor para número de ciclos de *clock* (*kernel* polinomial) na Tabela 3 e a frequência máxima, $f_{\max} = 83{,}02$, tem-se que

$$th = \frac{83{,}02\,\text{MHz}}{56} \approx 1{,}48\,\text{Mega SPS (MSPS)}. \tag{13}$$

É importante ressaltar que cada *sample* é um ponto flutuante de precisão simples, ou seja, 32 bits.

5.2 Resultados de validação

Com base nas informações apresentadas na seção de materiais e métodos (Seção 4), foi realizada uma simulação com precisão de bit para validar o hardware implementado. Para realizar a simulação, utilizou-se a ferramenta ModelSim com os seguintes parâmetros:

- *Clock*: Dado que o tempo crítico foi de 12,045 ns, a simulação foi realizada para um *clock* um período de 16 ns, (tempo maior que o tempo crítico) ocasionando em uma frequência de 62.5 MHz. O valor de 16 ns foi escolhido por ser a potência de 2 mais próxima do tempo crítico e facilitar a divisão dos contadores internos.
- *Run lenght*: Para calcular a *waveform* de todas as amostras, foi selecionado um *run length* de 100 μs.
- Dado que o *clock* escolhido foi de 16 ns, a frequência de mudança de endereços da memória ROM foi de $\frac{62{,}5\,\text{MHz}}{62} = 1{,}008\,\text{MHz}$.

Exemplos do resultado da simulação encontram-se nas Figuras 8 e 9. Observa-se que o tempo em que a classificação ocorre é de aproximadamente 1000 ns. Dividindo esse valor pelo período (16 ns), obtém-se aproximadamente 63 ciclos de *clock*, sendo 56 deles referentes ao algoritmo de inferência da SVM e o resto relativo aos circuitos de controle para leitura da ROM.

Uma vez obtidos os resultados pela plataforma ModelSim, foi realizada uma simulação em Matlab, para fins de comparar os resultados encontrados no primeiro caso. O método utilizado para avaliar a precisão e acurácia dos resultados simulados foi o cálculo do erro quadrático médio (*Mean square error* - MSE) entre os resultados da simulação na plataforma da ModelSim e os obtidos em Matlab. O *kernel* linear alcançou um MSE de $4{,}13 \times 10^{-10}$, ou seja, praticamente nulo, e o MSE do polinomial foi de $3{,}67 \times 10^{-5}$, maior que o caso linear, porém também muito próximo de zero. A diferença no caso polinomial pode ser explicada devido a utilização do formato ponto flutuante com 64 bits utilizado pelo Matlab (*double*). A não linearidade do *kernel* polinomial de grau $q = 3$ ressaltou a maior diferença com a implementação proposta em hardware que utilizou formato em ponto flutuante de 32 bits.

6. COMPARAÇÃO COM OUTROS SISTEMAS EMBARCADOS

O hardware proposto neste trabalho foi comparado com três microcontroladores (MCUs): ATMega2560 (arduino Mega), ATMega328P (arduino Uno) e ATMega 328 (arduino Nano). A primeira análise realizada foi no tempo de execução. O código utilizado para os três MCUs consiste na execução sem paralelismo das operações do kernel linear de apenas uma entrada. Um sinal é emitido quando essa operação é concluída, possibilitando a análise da frequência de cálculo das amostras através do uso de um osciloscópio. A Figura 10 apresenta as conexões realizadas para anotar as medidas: o canal 1 do osciloscópio DSO-2250 USB da Hantek ® recebe informações dos impulsos gerados no pino D13 do Arduino Mega, e essa informação é transmitida para a tela do computador através da comunicação USB do osciloscópio.

Um exemplo de análise encontra-se na Figura 11, onde o osciloscópio anota a frequência dos impulsos emitidos pelo ATMega2560 quando este encerra o cálculo de uma amostra.

Dado que foram utilizadas 50 amostras, o tempo total de execução também foi calculado. Os resultados obtidos são apresentados na Tabela 4, onde encontram-se os valores de período, *throughput* e tempo total de execução do algoritmo. Os resultados apresentados na Tabela 4 indicam que os três microcontroladores obtiveram desempenho semelhante.

Tabela 4. Período, *throughput* e tempo total medidos em cada microcontrolador utilizado e no hardware proposto

Dispositivo	Período	*Throughput*	Tempo total
Hardware proposto	675 ns	1,48 MPS	33,75 us
ATMega328	727 us	1,375 KPS	36,35 ms
ATMega328P	736 us	1,358 KPS	36,80 ms
ATMega2560	752 us	1,329 KPS	37,60 ms

Pode-se observar que o *speedup* entre a proposta em hardware dedicado apresentada neste artigo e as soluções

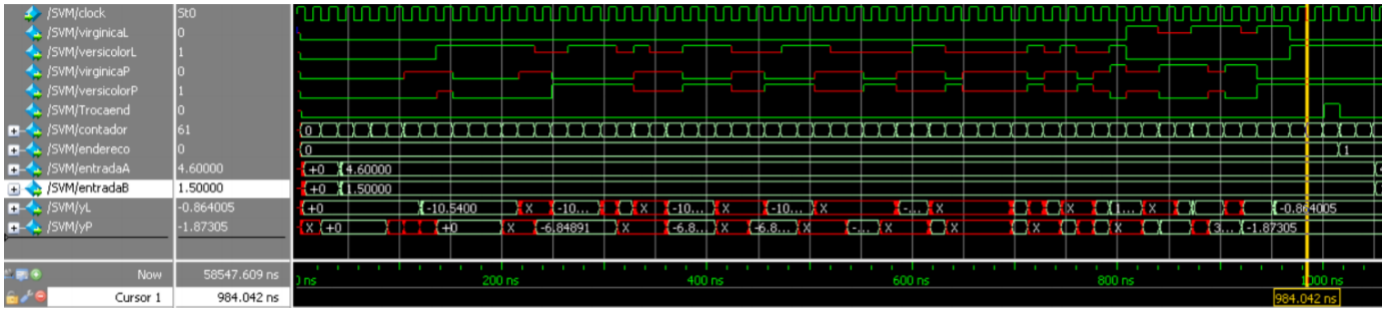

Figura 8. Simulação na plataforma ModelSim para a primeira amostra.

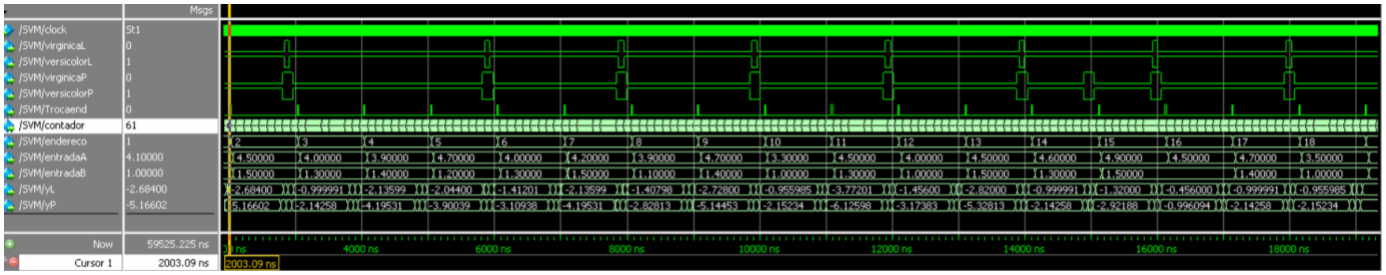

Figura 9. Simulação na plataforma ModelSim das amostras 2 à 18.

Figura 10. Estrutura utilizada para coletar a frequência de cálculo da inferência.

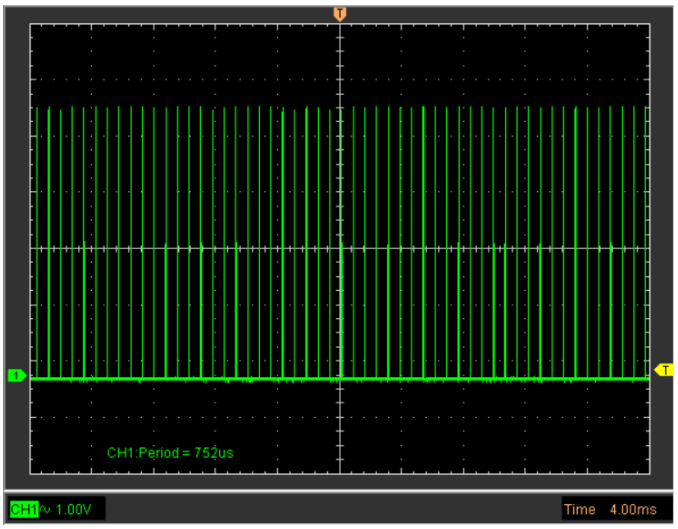

Figura 11. Frequência de execução das amostras na ATMega2560, medida por osciloscópio.

baseadas em microcontroladores é na ordem de 1000× ($\frac{1{,}48\,\text{MPS}}{1{,}375\,\text{KPS}} \approx \frac{1{,}48\,\text{MPS}}{1{,}358\,\text{KPS}} \approx \frac{1{,}48\,\text{MPS}}{1{,}329\,\text{KPS}} \approx 1000$), valor bem significativo, principalmente em aplicações de IoT que necessitem uma alta velocidade de processamento ou baixa latência.

Além disso, foi analisado o consumo energético dos três microprocessadores utilizados. Para isso, foi medida a voltagem e a corrente utilizadas pelas placas para obter a potência de cada dispositivo. Dois microcontroladores foram utilizados: um para realizar o processo de inferência, e outro para realizar as medidas sobre o primeiro. A estrutura final utilizada encontra-se na Figura 12, onde o Arduino Uno (1) realiza o processo de inferência, enquanto que o Arduino Mega (2) recebe as informações de corrente através do amperímetro (3) ligado em série entre o microcontrolador e a fonte de tensão, e a informação de tensão é coletada através do voltímetro (4), ligado em paralelo entre os pólos positivo e negativo da bateria. Dada a necessidade de conectar os módulos de medições, foi utilizado uma bateria de 9 Volts (5) devido a praticidade de adaptação dos seus fios conectores para realizar as medidas.

O consumo energético dos microcontroladores foi calculado multiplicando o tempo necessário para operar as 50 entradas das amostras inseridas pela potência encontrada. As medidas foram realizadas utilizando uma bateria de 9V como fonte de alimentação externa. Os resultados encontrados, descritos na Tabela 5, são semelhantes no quesito de consumo energético, uma vez que os três microcontroladores são da mesma família de dispositivos.

Por fim, foi analisado o consumo do hardware projetado neste artigo através da ferramenta *Power Analyzer Tool*, presente no software do Quartus Prime. Para comparar o consumo do hardware dedicado com os MCUs testados,

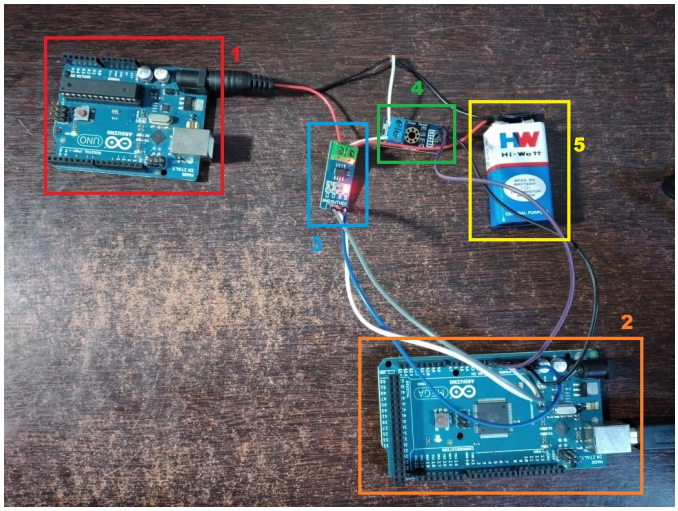

Figura 12. Estrutura utilizada para medir tensão e corrente utilizados em um microcontrolador.

Tabela 5. Voltagem, Corrente, Potência e Consumo medidos em cada microcontrolador.

MCU	Voltagem	Corrente	Potência	Consumo
ATMega328	7 Volts	188 mA	1,32 W	47,98 mJ
ATMega328P	5,3 Volts	260 mA	1,38 W	50,78 mJ
ATMega2560	6 Volts	218 mA	1,30 W	48,88 mJ

analisou-se o consumo do hardware para um *clock* de 80,3 KHz, tal que o *throughput*, *th* seja equivalente aos dos microcontroladores, ou seja, utilizando a Equação 12, tem-se

$$th = \frac{80{,}3\,\text{MHz}}{56} \approx 1{,}4\,\text{kSPS}. \quad (14)$$

A potência utilizada pelo sistema projetado no FPGA Cyclone IV E foi de 190,23 mW, cujo valor é aproximadamente 7 vezes menor que o registrado pelos MCUs. Este dado é bastante significativo pois mostra que soluções em hardware dedicado podem também ter um ganho no consumo de potência.

Os resultados obtidos neste artigo mostram que a utilização de SVM com hardware dedicado em FPGA podem apresentar soluções que atendam aos requisitos de baixo consumo e alta velocidade (*throughput*). Por exemplo, a solução demonstrada aqui pode-se trabalhar com o dobro do *throughput* apresentado nas soluções em MCU e, ainda assim, apresentar uma economia de consumo de potência.

7. CONCLUSÃO

A importância de utilizar tecnologias IoT junto de algoritmos de ML continua crescendo em todas as esferas sociais. Dada essa importância, o uso de ambas ferramentas em larga escala traz consigo o problema da otimização. A solução não pode ser apenas alcançada, agora os métodos de resolução do problema precisam ser rápidos, com baixo custo energético e com baixo espaço físico utilizado. Ou seja, a solução precisa ser eficiente.

Esse artigo demonstra a vantagem de gerar arquiteturas específicas para um problema que emprega ML, neste caso a fase de inferência de uma SVM, para alcançar os resultados desejados de forma eficiente. Ao comparar o sistema projetado com diferentes tipos de microcontroladores comumente utilizados, observou-se que o método utilizado pela arquitetura construída apresentou melhor desempenho e menor consumo energético para encontrar as respostas desejadas.

O uso de soluções eficientes para resolver problemas de ML produz benefícios para todas as esferas que a usufruem, seja pela velocidade ou pela economia de consumo. Novas aplicações em IoT podem ser mais complexas, uma vez que os algoritmos de ML não ocuparão tanto de sua capacidade, permitindo maiores avanços nessas tecnologias e maior quantidade de problemas resolvidos otimamente.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES).

REFERÊNCIAS

Anderson, E. (1936). The species problem in iris. *Annals of the Missouri Botanical Garden*, 457–509.

Coutinho, M.G., Torquato, M.F., and Fernandes, M.A. (2019). Deep neural network hardware implementation based on stacked sparse autoencoder. *IEEE Access*, 7, 40674–40694.

da S. Medeiros, D.R., Torquato, M.F., and Fernandes, M.A. (2020). Embedded genetic algorithm for low-power, low-cost, and low-size-memory devices. *Engineering Reports*, 2(9), e12231.

Dias, L.A., Damasceno, A.M., Gaura, E., and Fernandes, M.A. (2021). A full-parallel implementation of self-organizing maps on hardware. *Neural Networks*.

Dong, S., Yin, S., Tang, B., Chen, L., and Luo, T. (2014). Bearing degradation process prediction based on the support vector machine and markov model. *Shock and Vibration*, 2014.

Fisher, R.A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 179–188.

Ganesan, M. and Sivakumar, N. (2019). Iot based heart disease prediction and diagnosis model for healthcare using machine learning models. In *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*, 1–5. IEEE.

Gao, X. and Hou, J. (2016). An improved svm integrated gs-pca fault diagnosis approach of tennessee eastman process. *Neurocomputing*, 174, 906–911.

Guimarães, C.J.B. and Fernandes, M.A. (2020). Real-time neural networks implementation proposal for microcontrollers. *Electronics*, 9(10), 1597.

Gunawan, A.A., Brandon, D., Puspa, V.D., and Wiweko, B. (2018). Development of urine hydration system based on urine color and support vector machine. *Procedia Computer Science*, 135, 481–489.

Gupta, N., Khosravy, M., Patel, N., Dey, N., Gupta, S., Darbari, H., and Crespo, R.G. (2020). Economic data analytic ai technique on iot edge devices for health monitoring of agriculture machines. *Applied Intelligence*, 50(11), 3990–4016.

Jain, L., Pothireddy, N., Malikireddy, S.K.R., Parekh, V., Algubelli, B., Veluru, C., and Kumar, D. (2019). Automated failure prediction and self-healing of edge devices in internet of things infrastructure.

Janjua, Z.H., Vecchio, M., Antonini, M., and Antonelli, F. (2019). Irese: An intelligent rare-event detection system using unsupervised learning on the iot edge. *Engineering Applications of Artificial Intelligence*, 84, 41–50.

Liou, S.W., Kurniadi, D., Zheng, B.R., Xie, W.Q., Tien, C.J., and Jong, G.J. (2018). Classification of biomedical signal on iot platform using support vector machine. In *2018 IEEE International Conference on Applied System Invention (ICASI)*, 50–53. IEEE.

Liu, L., Yang, J., and Meng, W. (2019). Detecting malicious nodes via gradient descent and support vector machine in internet of things. *Computers & Electrical Engineering*, 77, 339–353.

Lopes, F.F., Ferreira, J.C., and Fernandes, M.A.C. (2019). Parallel implementation on fpga of support vector machines using stochastic gradient descent. *Electronics*, 8(6). doi:10.3390/electronics8060631. URL `https://www.mdpi.com/2079-9292/8/6/631`.

Minamisawa, A., Iimura, R., and Kawahara, T. (2019). High-speed sparse ising model on fpga. In *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 670–673. IEEE.

Noronha, D.H., Torquato, M.F., and Fernandes, M.A. (2019). A parallel implementation of sequential minimal optimization on fpga. *Microprocessors and Microsystems*, 69, 138–151.

Orrù, P.F., Zoccheddu, A., Sassu, L., Mattia, C., Cozza, R., and Arena, S. (2020). Machine learning approach using mlp and svm algorithms for the fault prediction of a centrifugal pump in the oil and gas industry. *Sustainability*, 12(11), 4776.

Osuwa, A.A., Ekhoragbon, E.B., and Fat, L.T. (2017). Application of artificial intelligence in internet of things. In *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*, 169–173. IEEE.

Otoom, M., Otoum, N., Alzubaidi, M.A., Etoom, Y., and Banihani, R. (2020). An iot-based framework for early identification and monitoring of covid-19 cases. *Biomedical Signal Processing and Control*, 62, 102149.

Ramadurgam, S. and Perera, D.G. (2021). An efficient fpga-based hardware accelerator for convex optimization-based svm classifier for machine learning on embedded platforms. *Electronics*, 10(11). doi:10.3390/electronics10111323. URL `https://www.mdpi.com/2079-9292/10/11/1323`.

Silva, S.N., Lopes, F.F., Valderrama, C., and Fernandes, M.A. (2020). Proposal of takagi–sugeno fuzzy-pi controller hardware. *Sensors*, 20(7), 1996.

Wang, H., Barriga, L., Vahidi, A., and Raza, S. (2019). Machine learning for security at the iot edge - a feasibility study. In *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW)*, 7–12. doi:10.1109/MASSW.2019.00009.