GrayScaleAccel: Acelerador de Escala de Cinza em FPGA

Thomas M. S. e Silva, Arlindo Fernandes, Sílvio R. Fernandes

Departamento de Computação – Universidade Federal Rural do Semi Árido (UFERSA)
Av. Francisco Mota, 572 - Bairro Costa e Silva, Mossoró RN – Brasil

{thomasmaikon, arlindo_fernandes22}@hotmail.com, silvio@ufersa.edu.br

Abstract. *In this paper, we present GrayScaleAccel, a Hardware-Software solution that accelerates a grayscale algorithm up to 183 times comparing to the software version. This accelerator uses the Ultra96 board and the PYNQ framework, which combined it is a platform with a tightly integrated processor and FPGA compatible with a high productivity Python framework. The FPGA synthesis still suggests that there are many resources to be used to explore the parallelism of this application or others that are even more ambitious.*

Resumo. *Neste artigo é apresentado o GrayScaleAccel, uma solução Hardware-Software que acelera um algoritmo de escala de cinza em até* 183 vezes comparado a versão puramente em *software. Este acelerador utiliza a placa Ultra96 e o framework PYNQ que favorecem esta solução devido a plataforma ter processador e FPGA fortemente integrados e ser compatível com framework de alta produtividade em Python. Os resultados de síntese ainda sugerem que há muitos recursos no FPGA para serem utilizados para exploração de paralelismo desta aplicação ou outras ainda mais ambiciosas.*

1. Introdução

Estamos vivendo na era da informação, onde grandes volumes de dados digitais são gerados em todo instante, o que cria um dilema dependência/exigência entre dispositivos e aplicações. Especificamente no Processamento Digital de Imagens (PDI), grandes quantidades de operações são realizadas em uma enorme quantidade de dados sistematicamente, de modo que diversas soluções paralelas têm sido empregadas, alimentando ainda mais esse ciclo de interdependência de dados e esforço computacional. Dessa forma, as GPUs (*Graphics Processing Unit*) têm explorado muito bem esse paralelismo ao custo de muita energia. Assim, os FPGAs (*Field-Programmable Gate Array*) podem ser uma alternativa para implementar o paralelismo requerido a um menor custo energético.

As principais etapas do PDI são formação e aquisição da imagem, digitalização da imagem, pré-processamento, segmentação, pós-processamento, extração de atributos, classificação e reconhecimento. O pré-processamento serve para eliminar possíveis ruídos ou informações desnecessárias nas etapas seguintes. Em diversas aplicações a conversão de imagens em tons de cinza é muito usada como pré-processamento. Dessa forma, desenvolver uma solução robusta de conversão em escala de cinza, é essencial para um alto desempenho em PDI, caso contrário há um gargalo que pode comprometer todo o restante da aplicação.

Portanto, neste artigo propomos um acelerador em FPGA para o algoritmo de conversão de imagens em escala de cinza para reduzir o tempo de execução dessa etapa, mantendo a qualidade da imagem. Na seção 2 apresentamos a contextualização do algoritmo alvo bem com a plataforma heterogênea escolhida. Seguida pelos detalhes da implementação,

na seção 3. Os resultados experimentais e de síntese estão na seção 4, seguida pelas conclusões e referências bibliográficas.

2. Contextualização

Nesta seção são apresentados o algoritmo de conversão de imagens do formato RGB (*Red, Green and Blue*) para escala de cinza e algumas aplicações que o utilizam, e por fim a plataforma escolhida para o desenvolvimento do acelerador de escala de cinzas proposto.

2.1. Algoritmo de Escala de Cinza

Em PDI, entre as etapas de pré-processamento, a conversão de imagens em escala de cinza é uma etapa muito comum. Existem inúmeros algoritmos, desde técnicas baseadas nas médias ponderadas dos canais de imagem RGB, até estratégias para gerar uma representação mais perceptivamente precisa. No estudo de [Kanan and Cottrell 2012] são apresentados alguns dos algoritmos com complexidade linear mais utilizados, dentre eles destaca-se o algoritmo de luminância, por ser simples e projetado para se assemelhar a percepção do olho humano.

O algoritmo em luminância utiliza o espaço de cores denominados por YIQ. A componente Y representa o brilho da imagem (luminância), I quantidade de tons de azul ou laranja e Q representa a quantidade de tons de verde ou roxo (crominância) na imagem, de modo que a conversão de RGB para YIQ é definida pela equação (1):

$$\begin{matrix} Y \\ I \\ Q \end{matrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}596 & -0{,}274 & -0{,}322 \\ 0{,}211 & -0{,}523 & 0{,}312 \end{bmatrix} x \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (1)$$

O modelo YIQ é usado pelo padrão de televisão NTSC, onde a componente Y tem toda a informação de saturação necessária para as imagens em escala de cinza, uma vez que as componentes I e Q são usadas apenas para modulação e decodificação para a TV. Dessa forma, a equação (1) pode ser reduzida a equação (2).

$$Y = 0{,}299 * R + 0{,}587 * G + 0{,}114 * B \quad (2)$$

Além de PDI e inteligência artificial, a conversão de imagem em escala de cinza tem sido usada em aplicações com diferentes propósitos. Como no sistema de detecção de sonolência de motoristas em tempo real em [Yan et al. 2016], ou para melhorar a identificação e extração de informações textuais [Papamarkou and Papamarkos 2013], [Bukhari et al. 2009]. Enquanto [Kang et al. 2020] e [Ding et al. 2019] sugerem o uso de escala de cinza para detecção e rastreamento térmico de objetos.

2.2. Placa Ultra96 e o Ambiente PYNQ

As plataformas CPU-FPGA possibilitam um bom equilíbrio entre desempenho computacional e consumo de energia, contudo sempre existiu o desafio na construção de aplicações heterogêneas, principalmente no que tange a comunicação entre CPU e FPGA. Felizmente, diversos fabricantes de plataformas heterogêneas têm investido no suporte para o programador, como a Xilinx[1], sua família de SoC Zynq e o *framework* PYNQ[2].

[1] https://www.xilinx.com/
[2] PYNQ - Python productivity for Zynq: http://www.pynq.io/

A plataforma adotada neste trabalho é Ultra96[3], a qual é construída com o SoC (*System-on-Chip*) Xilinx Zynq UltraScale+ compatível com o *framework* PYNQ. Como os demais SoC dessa família, o sistema é dividido em duas partes: PS (*Processing System*), onde se encontra o processador, e PL (*Programmable Logic*), onde se localiza o FPGA[4]. A placa ainda possui suporte a sistemas operacionais (SO) e ferramentas para personalização. Para comunicação padrão entre PS e PL, a Xilinx utiliza o AXI (*Advanced eXtensible Interface*) que possui as variações AXI4 *full*, AXI4-lite e AXI-Stream.

Para o desenvolvimento de soluções para o ambiente heterogêneo a Xilinx disponibiliza diversas ferramentas [Andersson [S.d.]] como Vivado, Xilinx SDK, *PetaLinux Tools*, etc. Com o intuito de aumentar ainda mais a abstração e a produtividade de soluções heterogêneas a Xilinx lançou PYNQ, um projeto *open-source* para as plataformas Zynq, Zynq UltraScale+, Zynq RFSoC, Alveo e AWS-F1. PYNQ é um acrônimo para ***Py**thon Productivity for Zy**nq***, que funde a produtividade do Python com a aceleração fornecida pela lógica programável no Zynq ou Zynq MPSoC.

PYNQ é um *framework* dividido em 4 camadas, na mais baixa está o SO baseado em Linux Ubuntu com os *drivers* da Xilinx, podendo ser customizado com *PetaLinux Tools*. Neste *framework* os projetos pré-compilados para o FPGA são chamados de *Overlay* e acessados por objetos do Python com esse mesmo nome.

Os FPGA *Overlays* (bibliotecas de *hardware*) incluem, pelo menos, um *bitstream* e metadados do projeto Vivado com os IPs disponíveis e informações de mapeamento de memória, periféricos, entre outros, para as aplicações, os quais devem estar no mesmo diretório e terem os mesmos nomes. A API ainda provê classes para acesso a GPIO, interrupções e interfaces AXI (mestre e escravo) para comunicação entre PS e PL da plataforma, mapeadas em memória ou por DMA. A imagem padrão do SO inclui um servidor Web Jupyter[5] integrado e as APIs para carregamento dinâmico de *bitstreams* para o FPGA da placa, bem como envio e recebimento de dados entre a aplicação e FPGA de forma interativa. Assim, as aplicações podem ser desenvolvidas por meio de Jupyter Notebook diretamente em um navegador, acessando o servidor que está executando na placa.

O *framework* PYNQ, e as respectivas placas compatíveis com ele, têm sido usados para construção de aceleradores para as mais diversas aplicações, como processamento de áudio [Stornaiuolo et al. 2019], detecção de objetos [Elsaadouny et al. 2020], criptografia [Shaher et al. 2020] e sobretudo inteligência artificial [Rybalkin et al. 2018], [Stornaiuolo et al. 2018] e [Vohra and Fasciani 2019].

Portanto, a placa Ultra96 foi escolhida pois permitiu o desenvolvimento do acelerador em FPGA com comunicação direta com o processador por meio do DMA. Além disso, a compatibilidade com o *framework* PYNQ permitiu a equipe focar no problema sem se preocupar com o desenvolvimento de *drivers*, chamadas de sistemas ou outros mecanismos para comunicação CPU-FPGA.

3. Implementação do Acelerador Proposto

O trabalho proposto, chamado de *GrayScaleAccel*, é uma solução heterogênea que implementa um acelerador em FPGA para o algoritmo de escala de cinza, transformando

[3] Ultra96 web site. https://www.96boards.org/product/ultra96/

[4] UltraScale+ MPSoC Data Sheet: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[5] Project Jupyter. https://www.jupyter.org

imagens do espaço de cores RGB para YIQ, usando a plataforma Ultra96. A Figura 1 ilustra o diagrama de blocos do *GrayScaleAccel* nesta placa. A parte implementada em *software* encontra-se no processador ARM, na parte PS da placa, o qual é responsável por ler o arquivo da imagem RGB disponibilizando-a na memória. A partir disso, o FPGA (parte PL), tem acesso à informação via DMA (*Direct Memory Access*) que envia para o acelerador propriamente dito (*GrayScale IP*), por *streaming*, e recebe o resultado da mesma forma à medida que vai sendo calculado pelo IP. No final, uma imagem em escala de cinza estará na memória disponível para o *software* usá-la, por exemplo para salvar em outro arquivo.

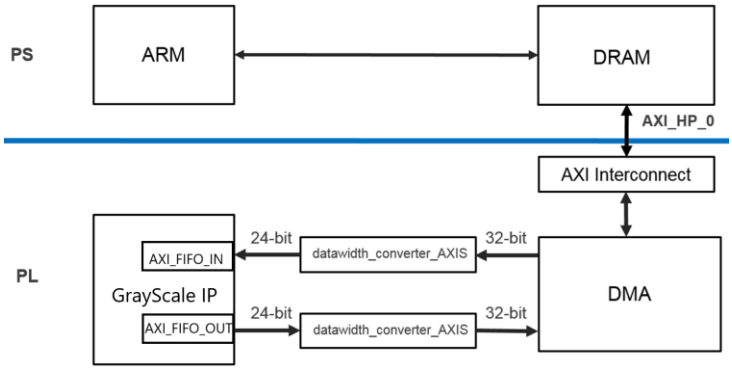


Figura 1. Diagrama de blocos do *GrayScaleAccel*

3.1 Implementação em *Software*

Inicialmente o algoritmo foi implementado em Python, de forma sequencial, para servir como *baseline* tanto para estratégia de implementação heterogênea quanto para comparação do desempenho do acelerador. Também como forma de comparação, foi implementada uma versão do *software* vetorizada, por meio da biblioteca *Numpy*. Essa biblioteca é conhecida pelo bom desempenho sobre *loop* e operações com matrizes, uma vez que foi construída por meio de *threads* e primitivas de outras linguagens como C++ que otimizam e paralelizam o código de forma transparente. Python foi escolhida pois o framework de integração CPU+FPGA é disponibilizado nesta linguagem.

A Figura 2, é usada para ilustrar como a aplicação e a integração com o *hardware* pode ser feita. Inicialmente um arquivo é aberto, a imagem RGB é lida e os dados são enviados para uma função que a converte para uma equivalente em escala de cinza, e por fim salva a imagem retornada em tons de cinza em arquivo em formato RBG. As linhas 15-17 e 21-23 (Figura 2) são usadas para exemplificar como as 3 opções, respectivamente, *software* sequencial (SW-Seq), *software* vetorizado (SW-Vec) e *hardware* acelerador (HW) podem ser usados para a conversão em escala de cinza de forma abstrata para o programador.

3.2 Implementação heterogênea *Software-Hardware*

A implementação heterogênea (HW) mantém a mesma estrutura inicial do *software*, mas a função "*GrayScaleHW*" apresentada na Figura 2 realiza a comunicação com a acelerador em FPGA. Tal comunicação é realizada por meio das funções do PYNQ que enviam e recebem dados via DMA. Dessa forma é necessário incluir a biblioteca *pynq*, importando o *Overlay* e *Xlnx*. O acelerador propriamente dito é formado por conversores de dados de 32 para 24 bits, e vice-versa, e o *GrayScale IP*, implementado usando Xilinx Vivado.

```python
import time
from PIL import Image

resolution = "640x360" #"640x360" "1280x720" "1920x1080"
imageInHardware = hardware(resolution, '../ip_repo/design_1.bit')
imageInSoftware = software(resolution)
file = Image.open("../images/imagem-"+resolution+".jpg")
img = file.load()
width, height = file.size
input_array = np.array(file)
execution_time = []

for i in range(10):
    start_time = time.time()
    result = imageInSoftware.GrayScaleSW(img, width, height)
    #result = imageInSoftware.GrayScaleSW_Vec(input_array, width, height)
    #result = imageInHardware.GrayScaleHW(input_array, width, height)
    stop_time = time.time()
    execution_time.append(stop_time-start_time)

result.save("../images/imagem-"+str(resolution)+"-SW.jpg")
#result.save("../images/imagem-"+str(resolution)+"-SW-Vec.jpg")
#result.save("../images/imagem-"+str(resolution)+"-HW.jpg")
print(np.mean(execution_time))
print(np.std(execution_time))
```

Figura 2. Implementação da aplicação

Os conversores de dados são usados devido a largura de dados padrão do AXI Stream, mas a aplicação utiliza 24 bits, com um byte para cada componente de cor. As interfaces de entrada e saída do IP foram implementadas como FIFOs (*First-In First-Out*) utilizando o protocolo AXI-Stream. O IP implementa uma máquina de estados que recebe os 24 bits em um ciclo, realiza o cálculo do valor de Y, de forma paralela, também em um ciclo e no terceiro ciclo envia 24 bits correspondente a YYY, pronto para escrever a imagem em escala de cinza em um arquivo RGB padrão. A Figura 3 ilustra o trecho principal do código do *GrayScale IP* em SystemVerilog, que mostra o paralelismo das multiplicações da componente R, G e B pelas respectivas constantes. Como não há unidade de ponto flutuante no *hardware*, para o cálculo do valor de Y (Figura 3), foi utilizado o artifício de multiplicar as constantes, originalmente com três casas decimais na equação (2), como unidades de milhar inteiras, e após a soma das 3 parcelas, o resultado é dividido por 1000 para o valor equivalente.

```
assign ConvertedR = R * 299;
assign ConvertedG = G * 587;
assign ConvertedB = B * 114;
assign Y = (ConvertedR + ConvertedG + ConvertedB)/1000;
```

Figura 3. Trecho do GrayScale IP em SystemVerilog

4. Resultados

Nesta seção serão discutidos os resultados obtidos, em relação a qualidade das imagens geradas pelo *GrayScaleAccel* (HW), seu desempenho comparado com a versão *baseline* (puramente em *software* sequencial – SW-seq) e uma versão do *software* vetorizado (SW-Vec), além dos resultados de síntese no FPGA.

No experimento foi usada uma mesma imagem em 3 resoluções diferentes (640x360, 1280x720 e 1920x1080), que seguem uma proporção de 2 e 3 vezes em relação a primeira

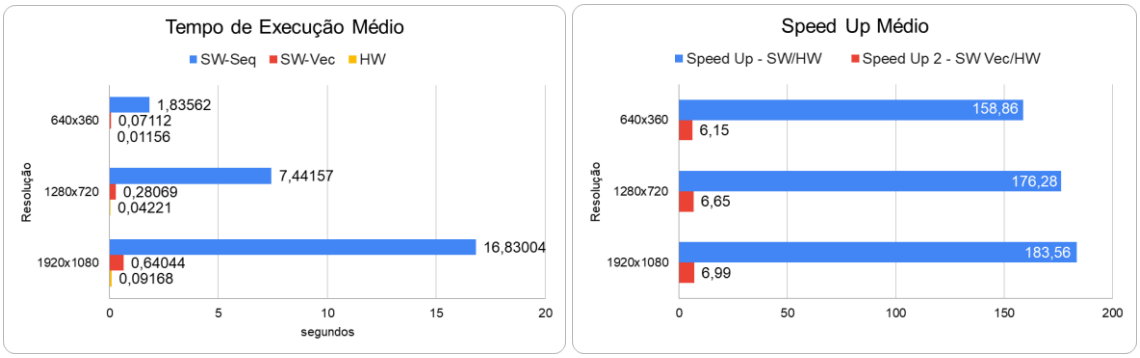
resolução. Para cada versão da função de conversão avaliada (SW-Seq, SW-Vec e HW) foram realizadas 10 execuções, para cálculo da média do tempo de execução de cada uma.

Inicialmente, verificou-se a qualidade das imagens geradas pelo acelerador, para perceber o impacto do artifício matemático usando apenas constantes inteiras no algoritmo YIQ no FPGA. Para tanto foi utilizado o erro quadrático médio ou MSE (do inglês *Mean Square Error*), como mostrado na Tabela 1 para as três resoluções de imagem, comparando os pares de imagem "SW-Seq x HW", "SW-Vec x HW" e "SW-Seq x "SW-Vec". Quanto menor o valor do MSE significa que as imagens comparadas são mais parecidas, ou seja, os pixels possuem mais valores próximos ou iguais. É possível perceber que a comparação entre os mesmos pares é quase constante, havendo um ligeiro aumento no erro acumulado em imagens com resoluções maiores, além do que imagens geradas pela versão SW-Vec é a que mais se diferencia das demais. Sobretudo, esses resultados mostram que artifício usado no *hardware* para lidar com operações de ponto flutuante não interferiu na qualidade das imagens geradas.

Tabela 1. MSE comparando as imagens das versões SW e HW

Resolução	SW-seq X HW	SW-vec X HW	SW-seq X SW-vec
640x360	0,000256076388888888	1,33127170138888	1,330798611
1280x720	0,000348307291666666	1,33404079861111	1,333466797
1920x1080	0,000483217592592592	1,314470486	1,313729745

O gráfico da Figura 4(a) apresenta a média do tempo médio de execução, em segundos, e os respectivos desvios padrões (Tabela 2), que mostram como há pouca variação, principalmente nas versões SW-Vec e HW, nos assegurando que a média representa bem as versões comparadas.



(a) **(b)**

Figura 4. Desempenho das versões do conversor: (a) tempo de execução; (b) *speed up*

Tabela 2. Desvio padrão dos tempos de execução

Resolução	Std SW-Seq	Std SW-Vec	Std HW
640x360	0,00248	0,00010	0,00041
1280x720	0,01720	0,00045	0,00035
1920x1080	0,03696	0,00069	0,00060

O tempo de execução é relacionado ao cálculo propriamente dito, desconsiderando os processos de abrir o arquivo da imagem original e salvar a imagem em escala de cinza em outro arquivo, e neste caso, os melhores desempenhos, são respectivamente HW, SW-Vec e SW-Seq.

Para efeito de comparação, calculamos a aceleração (*speed up*) do acelerador (versão HW) em comparação as duas versões em *software*. Na Figura 4(b) podemos perceber que há

aceleração com o *GrayScaleAccel* em todos os casos e que ela aumenta com o crescimento da resolução da imagem, chegando até 6,9 x em relação a versão SW-Vec e até 183 x sobre SW-Seq.

Por fim, temos os resultados de síntese do *GrayScaleAccel* no FPGA, para o dispositivo ZU3EG presente no Zynq UltraScale+ MPSoC da placa utilizada, os quais são apresentadas na Tabela 3. Os recursos mais limitados neste FPGA são DSP, usados nas multiplicações, e *Block RAM*, usados principalmente nas FIFOs. Com as quantidades disponíveis, poderíamos paralelizar o cálculo de até 120 pixels ou alcançar no máximo 19 portas de entrada/saída do acelerador. Dessa forma, há possibilidade melhorias no *GrayScaleAccel* ou utilizá-lo como um dos estágios de *pipeline* em um processamento digital de imagens mais complexo.

Tabela 3. Resultados de síntese no FPGA

Elementos	Disponível	Utilizado
DSPs	360	3 (0,83%)
LUT Logic	70560	4571 (6,48%)
LUT Memory	28800	287 (0,99%)
FlipFLop	141120	5952 (4,22%)
Block RAM	216	11 (5,09%)
Período do *clock* / frequência	2,743 ns / 364 MHz	

5. Conclusões e Trabalhos Futuros

Neste artigo foi apresentado o *GrayScaleAccel*, um acelerador para um algoritmo de conversão de imagens RGB para escala de cinza, usando a placa Ultra96. Essa placa é uma plataforma heterogênea com processador e FPGA fortemente integrados e compatível com o *framework* PYNQ, voltado para solução rápidas e heterogêneas usando a linguagem Python.

A parte do acelerador em *hardware* foi implementado em SystemVerilog com interfaces *AXI Stream*, o qual foi integrado a aplicação em Python. A comunicação com o *software* se deu por meio do DMA, abstraída pelas bibliotecas do *framework* PYNQ.

A validação se deu por meio de três versões do algoritmo de escala de cinza. Uma versão em *software* sequencial (SW-Seq), uma versão *software* vetorizada e paralelizada (SW-Vec) e a versão que utiliza o acelerador em FPGA (HW). Todas as versões utilizaram a mesma imagem com três resoluções diferentes, as quais foram executadas 10 vezes. As imagens geradas por todas as versões são muito semelhantes, verificado por meio do MSE. Os resultados de tempo de execução mostraram que o *GrayScaleAccel* apresenta melhor desempenho em qualquer resolução de imagem, sobretudo nas maiores, acelerando em até 6,9 x em relação a SW-Vec e até 183x sobre SW-Seq. Com tal desempenho, a versão HW é a única que conseguiria fazer a conversão em escala de cinza de imagens 640x360 e 1280x720 para aplicações de vídeos em tempo real, considerando 23 imagens por segundo.

O *GrayScaleAccel* utiliza muito pouco dos recursos disponíveis no FPGA, de modo que é possível aumentar a largura dos buffers de comunicação ou a quantidade deles, possibilitando mais paralelismo deste acelerador. Também é possível utilizá-lo como parte de um acelerador mais complexo para detecção de bordas nas imagens e posteriormente utilização em algoritmos de detecção de objetos para utilização em aplicações de aprendizado de máquinas.

Referências

Andersson, S. ([S.d.]). Zynq design from scratch. Part 41. http://svenand.blogdrives.com/archive/202.html#.XQQwgojMOUl.

Bukhari, S. S., Breuel, T. M. and Shafait, F. (2009). Textline information extraction from grayscale camera-captured document images. In 2009 16th IEEE International Conference on Image Processing (ICIP).

Ding, W., Kang, B., Zhou, Q., Lin, M. and Zhang, S. (2019). Grayscale-thermal Tracking via Canonical Correlation Analysis Based Inverse Sparse Representation. In ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

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

Kanan, C. and Cottrell, G. W. (2012). Color-to-Grayscale: Does the Method Matter in Image Recognition? PLOS ONE, v. 7, n. 1, p. e29740.

Kang, B., Liang, D., Ding, W., Zhou, H. and Zhu, W.-P. (2020). Grayscale-Thermal Tracking via Inverse Sparse Representation-Based Collaborative Encoding. IEEE Transactions on Image Processing, v. 29, p. 3401–3415.

Papamarkou, I. and Papamarkos, N. (2013). Conversion of color documents to grayscale. In 21st Mediterranean Conference on Control and Automation.

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

Shaher, I. M., Mahmoud, M., Ibrahim, H., Ali, M. and Mostafa, H. (2020). Implementation of a Hardware Accelerator for a Real-time Encryption System. In *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*.

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

Stornaiuolo, L., Perini, M., Santambrogio, M. D. and Sciuto, D. (2019). FPGA-Based Embedded System Implementation of Audio Signal Alignment. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

Vohra, M. and Fasciani, S. (2019). PYNQ- Torch: a framework to develop PyTorch accelerators on the PYNQ platform. In *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*.

Yan, J.-J., Kuo, H.-H., Lin, Y.-F. and Liao, T.-L. (2016). Real-Time Driver Drowsiness Detection System Based on PERCLOS and Grayscale Image Processing. In 2016 International Symposium on Computer, Consumer and Control (IS3C).