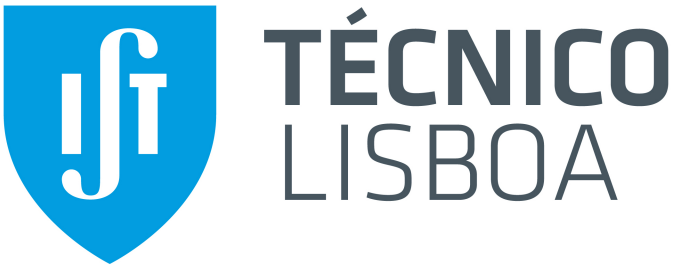

Instituto Superior Técnico

Mestrado Integrado Em Engenharia Eletrotécnica e de Computadores

Co-Projecto Hw/Sw

Data Input/Output - Parte II

Trabalho realizado por:	Número
Ion Ciobotari	84075
Mihir Odhavji	84151

Grupo nº3

02 de Junho de 2019

Conteúdo

1 Introdução

O âmbito principal deste projeto é desenvolver um programa que identifica um número desenhado numa imagem. A identificação será realizada através do uso de uma rede neuronal treinada, fornecida pelo professor responsável. Este programa será implementado usando componentes em *software* e *hardware*. Numa última fase, será realizada uma otimização do programa que tem como objetivo principal reduzir o tempo de execução.

2 Implementação

A identificação da imagem é realizada transformando a matriz, que retém informação sobre a imagem, num vetor de tamanho igual a 10. O índice do valor mais elevado deste vetor e a normalização deste valor indica-nos, respetivamente o valor associado à imagem e a sua exatidão. O método usado para se obter este vetor final pode ser observado pela figura 1, onde observa-mos que a computação pode ser dividida em várias camadas.

É de notar que a primeira e terceira camadas são determinadas com o recurso a um ficheiro com pesos que caraterizam a rede neuronal. Este ficheiro, tal como uma implementação em *software* são fornecidos pelo professor responsável. Desta forma, a análise duma implementação única em *software* não será realizada e todas as otimizações serão realizadas sobre a implementação em *software* fornecida.

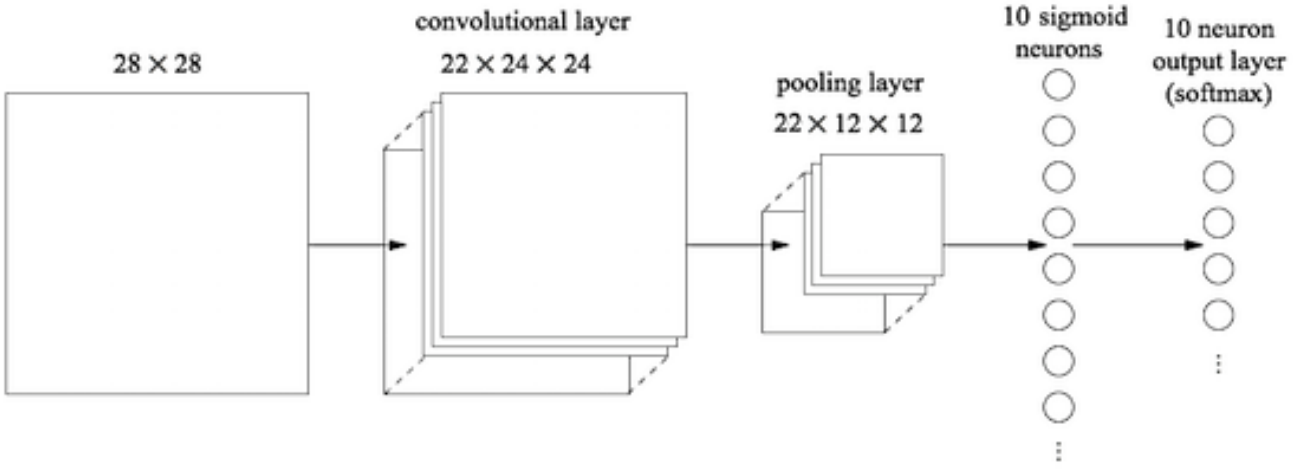

Figura 1: *Arquitetura do programa.*

2.1 Otimização com recurso a hardware

Uma primeira otimização realizada trata-se do desenvolvimento de um IP que executa uma ou mais camadas do programa. Desta forma, foram desenvolvidos dois IPs, denominados por *layer-1*, e *layer-1_2*, que executam a primeira camada e as primeiras duas camadas, respetivamente, em hardware. Também é possível desenvolver um IP capaz de executar todas as camadas em hardware. Contudo, este IP retira a necessidade da componente em *software* que vai contra os objetivos deste projeto.

Ambos os IPs esperam à entrada 22 matrizes *kernel* de tamanho 5×5 e a matriz da imagem transformada de tamanho 576×25. A escolha desta dimensão para a matriz da imagem deve-se de forma a

facilitar os cálculos realizados pelo IP. No caso do IP *layer-1_2*, juntamente com as matrizes *kernel* espera-se 22 valores de *bias*, também fornecidos no ficheiro de pesos. A saída destes IPs serão uma matriz de tamanho 22×576, para *layer-1*, e uma matriz 22×144, para *layer-1_2*.

A partir das tabelas 1 e 2, observa-mos que implementação destes IPs permite-nos alcançar um *speedup* aproximadamente igual 2, sendo que o IP *layer-1_2*, possui um maior valor de *speedup*. Contudo, observa-mos pela tabela 3 que o IP com maior *speedup* também consome uma maior quantidade de recursos.

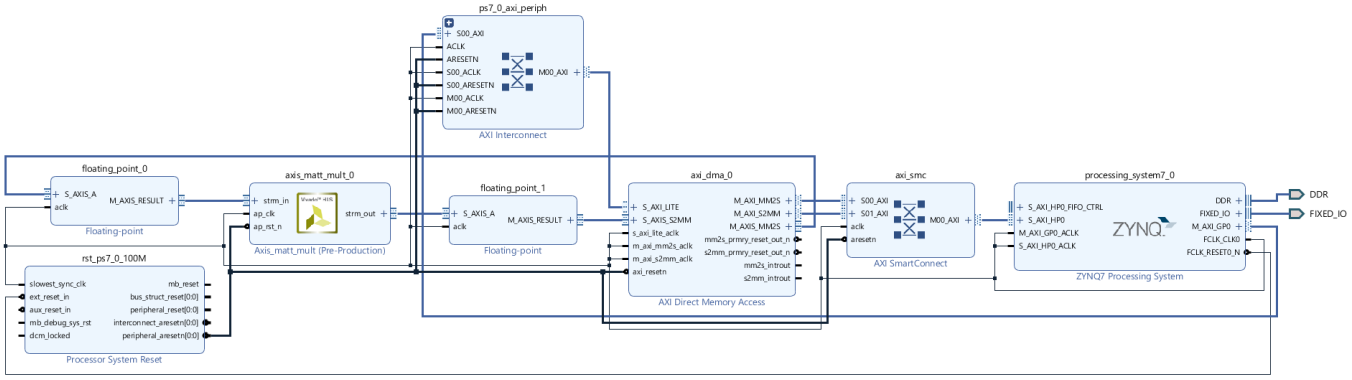

Figura 2: *Diagrama de blocos da implementação em software e hardware.*

2.2 Otimização com recurso a multiprocessadores

Outra otimização foi adaptar o programa desenvolvido no ponto anterior de forma a fazer uso dos dois processadores presentes na placa ZYBO utilizada. Assim, é possível processar mais do que uma imagem no mesmo instante de tempo. A figura 3 representa o diagrama de blocos da implementação em paralelo.

Esta implementação permite processar duas imagens simultaneamente, o processador 0 gere as imagens ímpares e o processador as imagens pares. O processador 0 responsabiliza-se por escrever na terminal os resultados obtidos. No caso quando é pedido processar um número ímpar de imagens, a última imagem é processada pelo processador 0. O sincronismo entre os processadores é efetuado a partir do uso de um semáforo. Este semáforo é definido na região de memória da RAM1, que não se encontrava usada.

Pelas tabelas 1 e 2 observa-mos que, como esperado, para 1 imagem não houve alteração no *speedup*, pois o processador 0 realiza todo o processamento. Para 100 imagens, esta implementação duplicou o *speedup* das configurações em *software* e o uso dos dois IPs desenvolvidos. Para além disso, comparando os diagramas de blocos das figuras 2 e 3, esperamos que esta implementação necessite do dobro dos recursos, que pode ser comprovado observando a tabela 3.

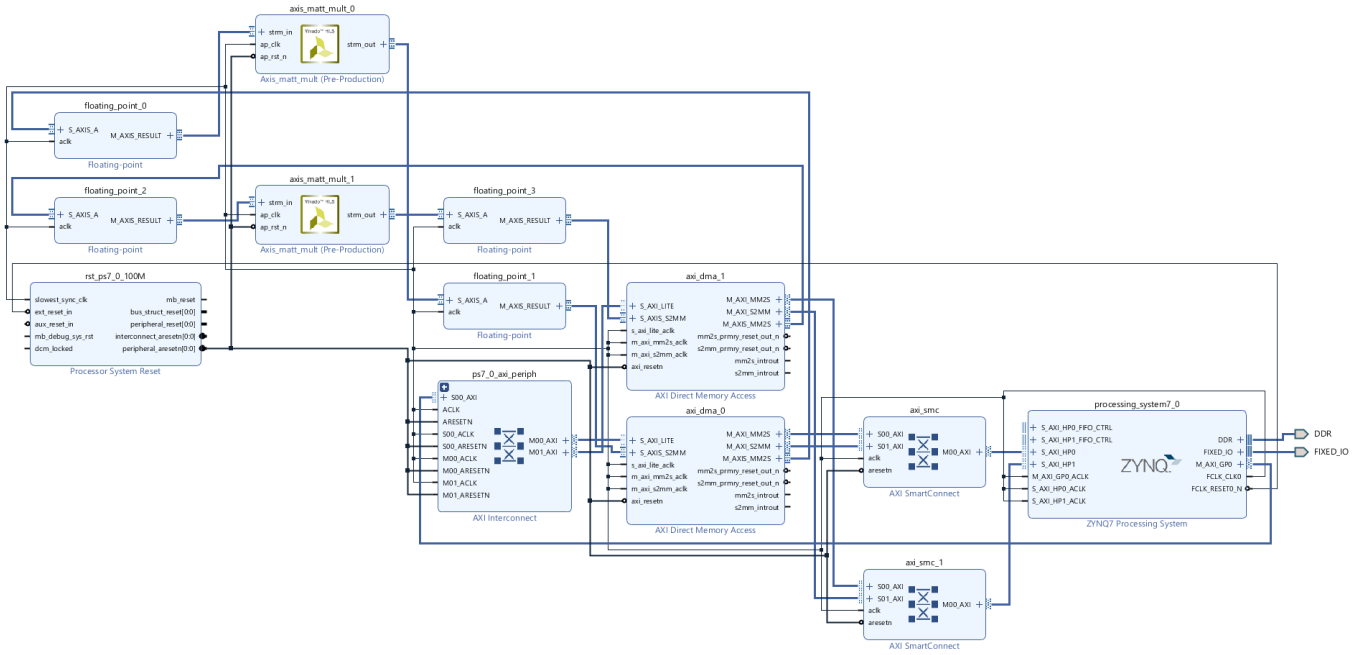

Figura 3: *Diagrama de blocos da implementação em software e hardware.*

2.3 Otimizações finais

A última otimização realizada trata-se de diminuir os bits das palavras de entrada e saída do IP que produz o maior speedup, no nosso caso o IP *layer-1_2*. Assim, este IP, denominado por *layer-1_2v2*, foi modificado de forma a efetuar a leitura de palavras de 16 bits e escrita de palavras de 32 bits, ao contrário da leitura de 32 btis e escrita de 64 bits efetuados pelos IPs anteriores.

Com esta implementação, como se pode observar pelas tabelas 1 e 2, o *speedup* aumentou cerca de 20% comparado com o IP *layer-1_2*. Pela tabela 3 observamos que os recursos necessários para implementação são inferiores aos recursos necessários para a implementação do IP *layer-1*.

Contudo, esta implementação possui uma pior exatidão de resultados. Este facto foi comprovado observando o resultado da imagem 19 entre todas as implementações. As implementações em *software* e dos IPs *layer-1* e *layer-1_2* indicam que a imagem 19 corresponde ao número 3 com uma exatidão cerca de 37.64%, enquanto que o IP *layer-1_2v2* indica que esta imagem corresponde ao número 8 com uma exatidão cerca de 38.38%.

Tabela 1: *Tempo de execução em μs do programa em várias implementações*

nº imagens	**Sequencial**			**Paralelo**			
	sw	*layer-1*	*layer-1_2*	sw	*layer-1*	*layer-1_2*	*layer-1_2v2*
1	36291	19954	17976	36290	20015	17995	14825
100	3628733	1995967	1796411	1815540	1020602	914794	75684

Tabela 2: *Speedup do programa em várias implementações*

nº imagens	**Sequencial**		**Paralelo**			
	layer-1	*layer-1_2*	sw	*layer-1*	*layer-1_2*	*layer-1_2v2*
1	1.82	2.02	1.00	1.81	2.02	2.45
100	1.82	2.02	2.00	3.56	3.97	4.80

Tabela 3: *Quantidade de recursos utilizados, em percentagem, em várias implementações*

	Sequencial		**Paralelo**		
	layer-1	*layer-1_2*	*layer-1*	*layer-1_2*	*layer-1_2v2*
LUT	22.97	23.98	44.88	46.96	42.08
FF	17.74	18.11	34.72	35.45	32.10
DSP	5.00	6.25	10.00	12.50	5.00
BRAM	31.67	31.67	63.33	63.33	35.00

Tabela 4: *Quantidade de recursos disponíveis*

LUT	**FF**	**DSP**	**BRAM**
17600	35200	80	60

3 Conclusão

Ao longo deste projeto observamos que a combinação de *software* e *hardware* proporciona uma grande de diminuição do tempo de execução de um programa, sendo possível diminuir ainda mais este tempo de execução incluindo outras implementações.

Uma primeira implementação que foi estudada trata-se do uso de multiprocessadores. O uso de multiprocessadores que permite diminuir o tempo de execução por um fator igual ao número total de processadores usados para executar o programa a custo de memória, que aumenta pelo mesmo fator. Por exemplo, ao usar dois processadores diminuímos o tempo de execução para metade, mas aumentamos a memória necessária para o seu dobro, usando três processadores diminuímos para um terço do tempo original e a memória para o triplo.

A segunda implementação estudada trata-se da diminuição do tamanho do tipo de dados que estão a ser processados, ou seja, por exemplo, diminuir o tamanho de um inteiro de 4 bytes, originais, para 2 bytes. Esta implementação permite uma pequena diminuição do tempo de execução e de memória.

Concluindo, neste projeto foram tomados em conta três aspetos do programa, o tempo de execução, a memória utilizada e a exatidão dos resultados. Pelos resultados obtidos nas várias implementações realizadas concluímos que é possível otimizar um programa em dois destes aspetos, sendo que o terceiro sofrerá uma penalização. Por exemplo, caso queiramos otimizar o programa em tempo e exatidão, sofremos em memória, que se traduz no seu aumento, quando que, escolhendo tempo e memória, sofremos na exatidão de resultados.