Universidade de São Paulo

SSC0741 - Projeto e Implementação de Sistemas Embarcados I

Prof. Dr. Vanderlei Bonato

Co-Projeto de Hardware e Software para Reconhecimento de Dígitos Manuais

---

Autores:	*Número USP:*
Ilan S. G. de Figueiredo	7656321
Kollins G. Lima	9012931
Rodrigo A. B. Ferrari	8006421

7 de Dezembro de 2018

Conteúdo

1 Introdução

Um Co-Projeto de *hardware* e *software* é uma tentativa de aproveitar as características de próprias destas duas áreas para a criação de um sistema híbrido que funcione melhor quando comparada à uma implementação apenas em *hardware* ou em *software*. Normalmente, esta melhoria se dá em termos de performance, mas também outros fatores (como consumo de recursos, por exemplo) podem ser visados.

No mundo do *hardware*, o principal benefício é a velocidade. Enquanto que em um computador tradicional as operações são executadas em uma CPU de uso geral, com processos compartilhando recursos, além do *overhead* do próprio S.O., um circuito que executa uma lógica em *hardware* estará limitado apenas pela velocidade dos componentes eletrônicos. No entanto, implementações em *hardware* sofrem o problema de escalabilidade e de complexidade do projeto.

Projetos em *software*, por outro lado, tem a vantagem de serem escaláveis e mais simples de desenvolver, fornecendo diversas abstrações para implementação, o que faz desta uma opção muito mais agradável do ponto de vista do desenvolvedor. Além disso, o custo de desenvolvimento é mais baixo quando comparado com desenvolvimento de *hardware*, já que não é preciso fabricar o circuito (ou utilizar FPGAs ou outra plataforma para prototipagem).

Desta forma, um co-projeto busca fazer uma implementação em *software* que utilize, para determinadas operações/funções, recursos de *hardware* que possam otimizar a performance daquele trecho de código. Obviamente, esse tipo de implementação traz diversos desafios, como a habilidade e o conhecimento de diversas tecnologias, o custo de comunicação entre *hardware* e *software*, uma maior dificuldade para testar o projeto, etc.

1.1 Problema

O problema abordado neste projeto é a otimização de performance de uma rede neural convolucional (CNN), cuja aplicação é a classificação de dígitos manuais. Pretende-se analisar a aplicação desenvolvida inteiramente em *software* de modo a identificar operações/funções de menor desempenho, para que estas possam ser implementadas em *hardware*, utilizando uma FPGA.

1.2 Organização do Trabalho

Na seção 2 e 3 são introduzidos conceitos de redes neurais convolutivas, o projeto usado como referência para implementá-las e o *profiling* feito sobre suas etapas para determinar a melhor camada a ser otimizada em *hardware*; nas seções 4 e 5 são apresentados os materiais usados no projeto e os métodos aplicados; Na seção 6 são apresentados e disserta-se sobre os resultados obtidos; na seção 7, são apresentadas as conclusões finais do trabalho e sugestões para possíveis continuidades do projeto; Finalmente, nos apêndices A e B são listados os códigos nas linguagens C e *Verilog* desenvolvidos e usados ao longo do projeto, respectivamente.

1.3 Acesso ao Projeto Desenvolvido

Todos os arquivos necessários para a execução do projeto estão disponíveis em um repositório do *GitHub* criado pelos alunos: `https://github.com/rab-ferrari/ssc0741-fpga-digits-recognition`. Eles são dependentes dos *softwares* proprietários *Quartus II* e *ModelSim*, mas ao reproduzir o projeto, recomenda-se começar do projeto descrito na seção 4.1 e nele incorporar as alterações feitas ao arquivo principal e as alterações feitas através da ferramenta *QSys*.

2 Definição e treinamento da rede

Para resolver o problema de classificação de dígitos manuais, foi utilizada uma rede neural convolucional (CNN). Esta rede se mostra uma boa opção na classificação de imagens pois ela é capaz de fazer uma extração de características antes de fazer a classificação, aumentando a taxa de acerto e a robustez da rede. A estrutura da CNN é mostrada na figura 1.

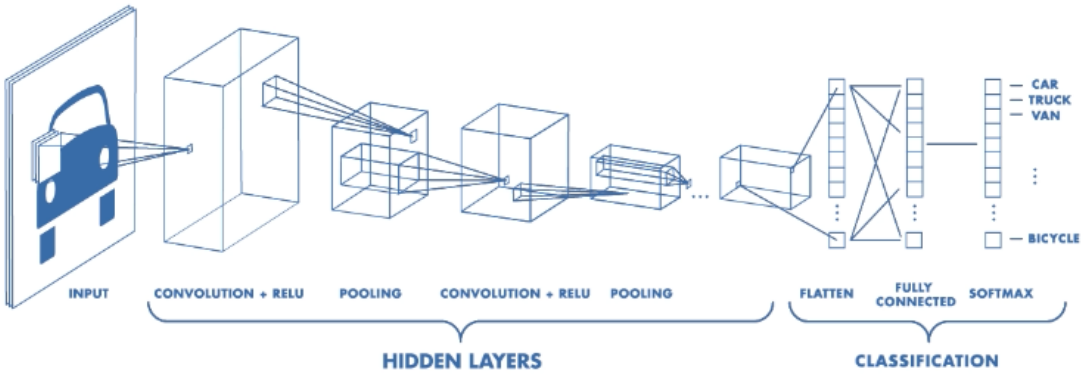

Figura 1: Estrutura da rede neural convolucional
Fonte: Daphne Cornelisse[1]

A figura 1 mostra a seguinte estrutura:

- **Convolução e RELU**: É o processo em que características da imagens são identificadas e ressaltadas. Para cada característica que se deseja extrair é preciso fazer uso de filtros. Uma característica é identificada pelo processo de convolução da imagem de entrada e o filtro definido. A figura 2 mostra como ocorre este processo.

[1] `https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050`

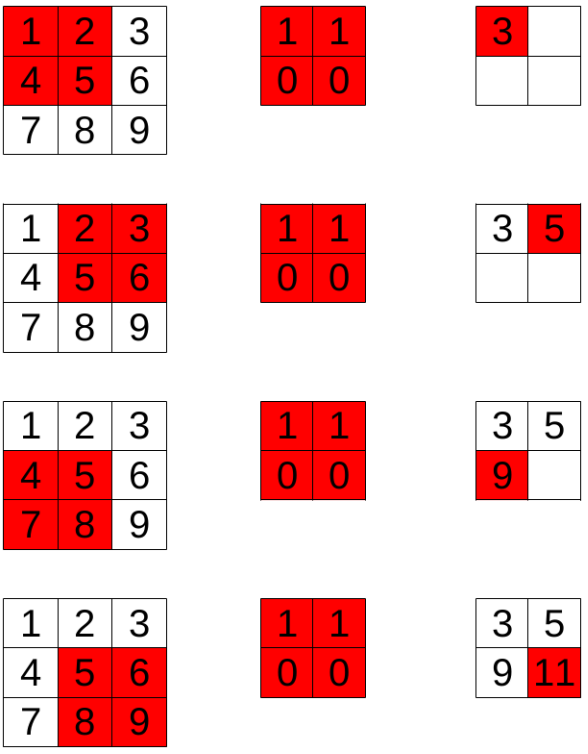

Figura 2: Processo de convolução
Fonte: Autor

A matriz à esquerda (figura 2) é a matriz de entrada (podendo representar uma imagem em escala de cinza, por exemplo). A matriz intermediária é o filtro que ressalta uma característica a ser identificada e a matriz à direita é o produto escalar entre as matrizes, conforme o filtro se desloca pela imagem de entrada.

Após a convolução, cada elemento da matriz resultante passa por uma função de ativação RELU, mostrada na figura 3. Ao final deste processo, a matriz resultante nos mostra que a característica buscada é mais evidente quanto maior for o valor numérico resultante.

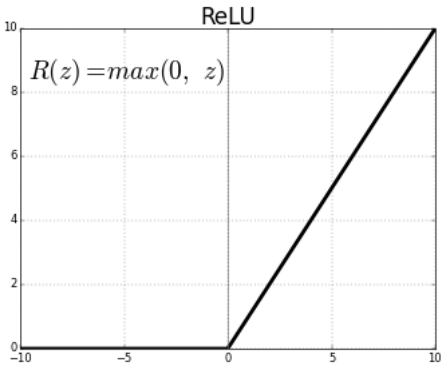

Figura 3: Função de ativação RELU
Fonte: Kanchan Sarkar[2]

- ***Pooling***: Outra etapa mostrada na figura 1 é o *pooling*. Esta é uma etapa onde a matriz resultante do processo de convolução é reduzida, de modo a preservar as principais características mas eliminando a quantidade de parâmetros, de modo a tornar a computação mais rápida nas etapas posteriores.

  A etapa de *pooling* também faz uso de filtros e pode aplicar diversos critérios para reduzir a região coberta pelo filtro, tais como média, máximo, etc. Na figura 2, por exemplo, se for definido um filtro 2x2, um *pooling* máximo transformaria a matriz em um único valor (no caso 11).

- **Classificação:** Finalmente, após fazer a extração de características da imagem, estas passam por uma rede neural totalmente conectada que fará a classificação das características de entrada para um conjunto de saída desejado (no caso deste projeto, números de 0 a 9). Para este projeto, esta rede é um MLP.

2.1 Treinamento

A primeira etapa do desenvolvimento foi o treinamento de uma rede CNN de modo a obter os parâmetros que fornecem a melhor classificação para o conjunto de treinamento.

Não foi feita uma implementação da CNN, mas sim utilizado o projeto *simple_cnn*[3] disponível no Github, criado por Can Bölük. Para o treinamento, foi utilizada a

[2] `https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec`

[3] `https://github.com/can1357/simple_cnn`

base de dados MNIST, que conta com 60 mil imagens de dígitos manuais rotulados para treinamento e mais 10 mil para validação.

Os parâmetros que foram determinados no treinamento foram os seguintes:

- Tamanho dos filtros (convolução e *pooling*);
- Quantidade de filtros (convolução);
- Passo do filtro, ou seja, qual será seu deslocamento na imagem de entrada para fazer a convolução e o *pooling*;
- Quantidade de épocas.

A tabela 1 mostra o progresso da etapa de treinamento. O treinamento da rede foi feito variando-se um parâmetro por vez e fixando a configuração de melhor resultado.

Tabela 1: Etapas de treinamento da CNN

Filtros Convolução			**Filtros *Pooling***		**Épocas**	**Erro (MSE)**	
Tamanho	**Quantidade**	**Passo**	**Tamanho**	**Passo**		**Treino**	**Validação**
4	1	1	1	1	1	24,0459	21,3762
4	10	1	1	1	1	20,7015	13,4624
4	10	1	6	1	1	13,8936	7,97005
4	10	1	6	1	10	7,28944	7,25106
7	10	1	6	1	10	6,19439	5,07604

A última linha da tabela 1 mostra a configuração de melhor resultado antes de ocorrer *overfitting*.

3 *Profiling*

Após a definição da rede, foi realizado um *profiling* para definir qual operação realizada pela CNN era a mais custosa em termos de tempo de processamento. Este *profiling* foi feito de maneira mais informal, ou seja, não foi utilizado uma ferramenta de *profiling* (como o *gprof*), mas sim a biblioteca *chrono* em C++.

O objetivo desta etapa não foi medir, precisamente, o tempo que cada operação leva para executar, mas analisar os tempos de forma relativa, ou seja, o quão lento uma operação é em relação a outra.

A figura 4 mostra a comparação de performance entre as operações realizadas pela CNN. As medições foram realizadas na placa DE2i-150, que foi utilizada para a implementação do projeto e é explicada em maiores detalhes na subseção 4.1.

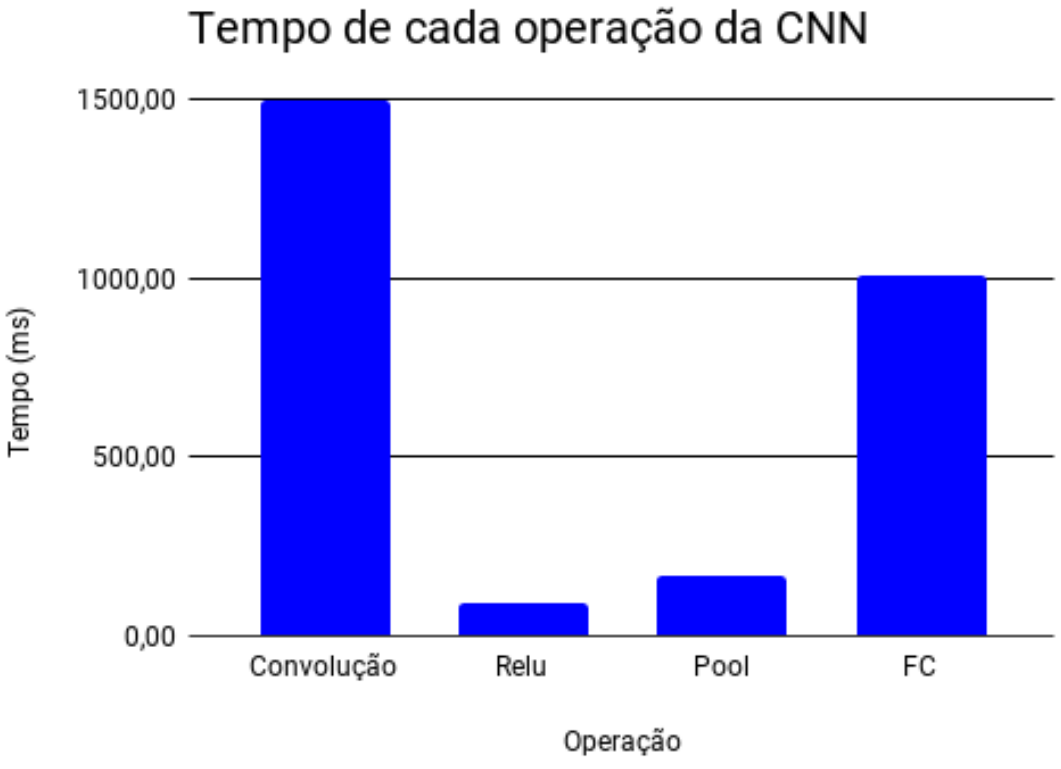

Figura 4: Tempo de execução de cada operação da CNN

Como já era esperado, a operação de convolução se mostrou a mais custosa em

termos de performance e, portanto, será esta a operação a ser implementada em *hardware*.

4 Materiais

Foram usados os seguintes materiais durante o desenvolvimento completo do projeto:

- Uma plataforma de desenvolvimento DE2i-150 da TerASIC dispondo de uma FPGA e um microcomputador integrados;
- Componentes de *hardware* adicionais para permitir a interface com o sistema operacional da plataforma (teclado, mouse, monitor);
- Um computador com sistema operacional *Windows* 7 ou superior e contendo os *softwares ModelSim* versão 6.1 ou superior e *Quartus II* versão 14.1 ou superior devidamente instalados e licenciados;

Mais detalhes sobre a plataforma de desenvolvimento De2i-150 são fornecidos a seguir na subseção 4.1.

4.1 Placa DE2i-150

A plataforma de desenvolvimento DE2i-150, da TerASIC, é uma placa híbrida que contém uma FPGA integrada. A seguir são listadas suas principais características.

- Processador Intel Atom Dual Core, 64-bits, 1.6GHz
- 2GB de memória DDR3
- 64GB de SSD
- FPGA Cyclone IV EP4CGX150DF31 com 149,760 elementos lógicos.

Originalmente, esta plataforma vem com o sistema operacional *Yocto*. No entanto, este sistema foi substituido pelo Xubuntu 18.04 64-bits por facilidade de uso.

4.1.1 PCIex

Para realizar a comunicação entre o processador Atom e a FPGA Cyclone IV, a plataforma DE2i-150 utiliza um barramento PCIex.

Para utilizar o PCIex, foi utilizado como referência o projeto *hellopci*[4]. Este consiste em um co-projeto que faz a leitura dos *switches* da placa e os envia para o processador, que retorna valores a serem exibidos nos displays 7-segmentos de volta para a FPGA, conforme o *switch* ativo.

[4] `http://rijndael.ece.vt.edu/de2i150/`

Partindo deste projeto, o grupo fez as modificações necessárias para que a comunicação ocorresse de forma a transferir uma imagem do Xubuntu para a FPGA (pixel a pixel), onde ocorrerá o processo de convolução e posteriormente a imagem retornasse para o Xubuntu para que seja feita a classificação da imagem.

5 Etapas de Desenvolvimento

Esta seção busca explicitar as etapas de desenvolvimento do projeto completo, bem como os métodos aplicados em cada uma destas.

5.1 Execução da Aplicação Completa de Classificação em Software na Plataforma

O primeiro passo no projeto foi o desenvolvimento de uma aplicação completamente em *software* implementando uma solução CNN para detecção de dígitos manuscritos. Foi usado como base o projeto descrito na seção 2, mas também foram necessárias pequenas alterações e cuidadosa análise do código fonte para determinação da futura função de convolução a ser trocada pela implementação otimizada em *hardware* e os tipos de dados a ela enviados.

5.2 Desenvolvimento de uma Aplicação Embarcada Básica de Comunicação

Para esta segunda etapa, foi usada como base a aplicação disponível no projeto de referência usado para a comunicação por meio do barramento *PCIex*, explicitado na subseção 4.1. Foram adaptados tanto o projeto e o *bitstream* gravados na FPGA por meio da ferramenta *QSys* do *Quartus*, bem como o código da aplicação para que fosse enviado um inteiro de 32 bits para a FPGA, onde esse valor seria armazenado em um buffer interno e em seguida mandado de volta para a aplicação executando no sistema operacional do microcomputador.

Em termos práticos de implementação, desenvolvido na linguagem C, o processo de envio e recebimento de dados pelo barramento é feito simplesmente por meio da leitura e escrita em um arquivo. Durante a compilação do *driver* do barramento, esse arquivo de sistema é gerado e referenciado como o acesso ao barramento.

O código dessa aplicação inicial gerado é apresentado no excerto 1, disponível no apêndice A.

5.3 Definição e Gravação do Protocolo de Comunicação

Ao contrário da aplicação disponível no projeto de referência e da aplicação inicial desenvolvida, diversos bits de dados devem ser enviados à FPGA a cada execução da convolução, a saber, $28 \times 28 = 784$ valores inteiros, correspondendo cada um a um pixel da imagem.

Para tanto, precisa-se de um sinal de sincronismo entre a aplicação e o *hardware* dedicado, bem como um sinal de *reset* para ser acionado entre o envio de imagens diferentes. Foram então definidas duas partições do barramento de 32 bits cada, uma de entrada e uma de saída.

O protocolo de comunicação aplicado consiste no uso do bit 31 como sinal de sincronização, do bit 30 como sinal de *reset* em nível lógico alto e os bits 15 a 0 como bits de dados para a entrada de dados na FPGA. Para o barramento de saída, todos os 32 bits são usados como bits de dados, mas devido à arquitetura do módulo de convolução gravado em *hardware*, um sinal de validade da saída é necessário. Define-se então o valor hexadecimal 0xFFFFFFFF como valor indicativo de um dado inválido.

O protocolo completo descrito é ilustrado na figura 5.

Adicionalmente, na FPGA o armazenamento em *buffers* de entrada e saída foi feito na borda de subida do sinal de sincronismo e o envio do *buffer* de entrada para o módulo de convolução e do módulo para o *buffer* de saída em borda de descida do *clock*. Esse cuidado é necessário para garantir que os dados estejam disponíveis e estáveis nas entradas do módulo de convolução e do barramento de saída na transição usada para cada respectiva transferência.

5.4 Teste e Validação da Comunicação para Conjuntos de Dados Sequenciais

Após definido o protocolo de comunicação, tanto a descrição de *hardware* quanto o código da aplicação foram atualizados para comportar a transmissão de uma pequena matriz de dados. Os dados foram apenas enviados à FPGA e recebidos novamente na aplicação. E execução também foi feita passo a passo para que os valores enviados pudessem ser acompanhados no *softwarae* por meio de *printfs* e do terminal e na FPGA por meio dos *displays* de 7 segmentos e dos LEDs.

Para garantia da aplicação do protocolo, máscaras binárias foram usadas no código da aplicação antes do envio e após o recebimento dos dados.

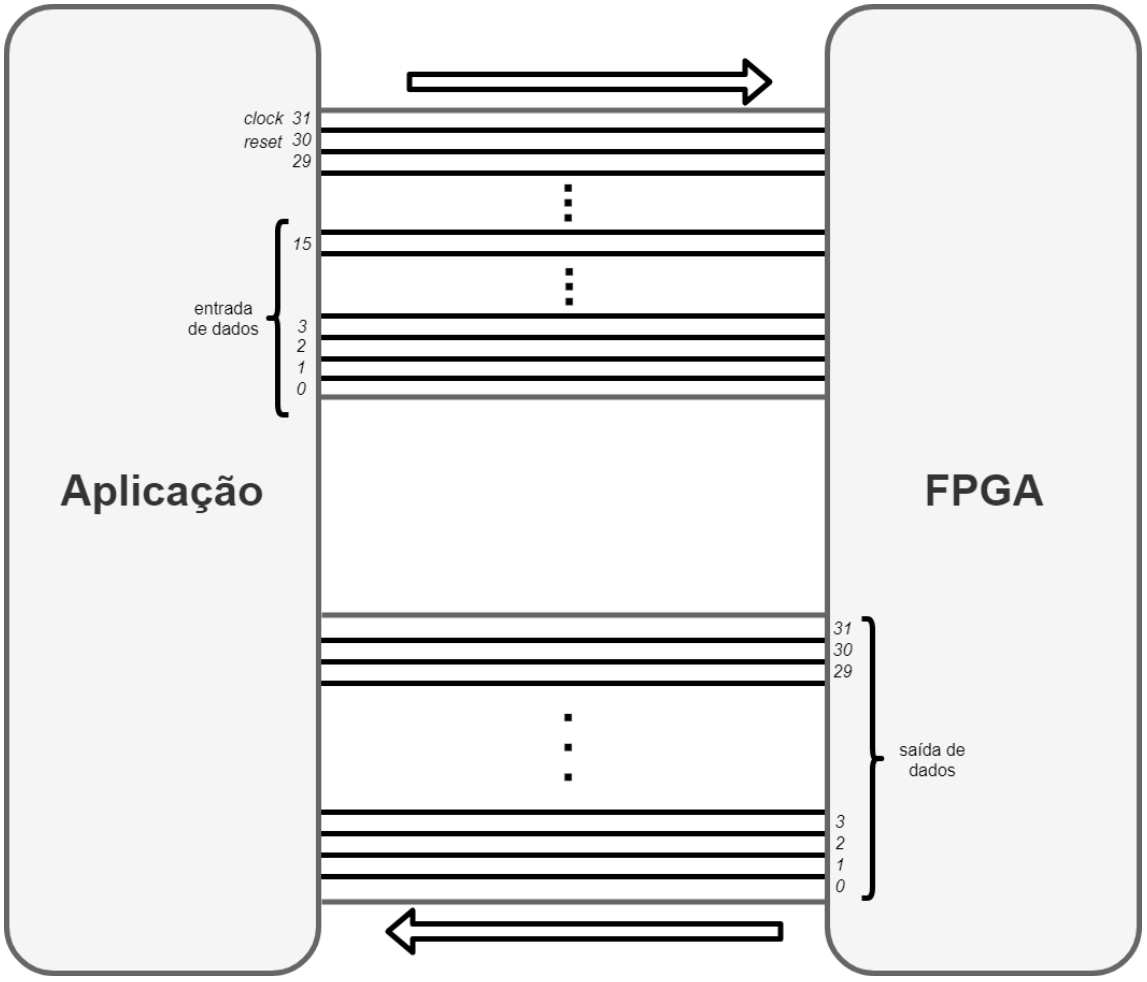

Figura 5: Ilustração do protocolo de comunicação definido e usado no microcomputador e na FPGA da plataforma.

Fonte: Autor

5.5 Gravação, Teste e Validação do Módulo de Convolução

O módulo de convolução foi descrito em *Verilog* baseando-se em uma implementação otimizada de um filtro *Sobel*[5]. A implementação faz uso de uma arquitetura do tipo *pipeline*.

Supondo a dimensão do *kernel* $K \times K$ e a da matriz de dados $N \times N$, a arquitetura conta com K^2 blocos *mac* (multiplicadores e acumuladores) e registradores (chamados registradores de *kernel*). Cada um dos acumuladores possui como entradas um

[5] `https://github.com/usmanwardag/sobel`

registrador de *kernel* precedente, um pixel (enviado ao mesmo tempo para todos os K^2 *macs*) e a saída ligada ao registrador seguinte.

A arquitetura também conta com $(N - K) \times (K - 1)$ registradores (chamados registradores de *shift* ou apenas *shift*) conectados entre as linhas de registradores de *kernel*.

O módulo possui duas saídas principais, a saída do último registrador que deverá conter o resultado da convolução e um bit de controle que indica se a saída de determinado instante é válida de acordo com a relação 1:

$$\text{saída válida} \iff \begin{cases} counter > ((K-1) * N + (K-1)) \\ counter < (M * N) + (K-1) \\ (counter - (K-1)) \ \% \ N > 1 \end{cases} \tag{1}$$

onde *counter* é uma variável de contagem interna ao módulo iniciada em zero e incrementada a cada novo pixel recebido.

O esquemático referente ao módulo de convolução fazendo uso dos módulos *mac*, *shift* e registradores de *kernel* é apresentado na figura 6.

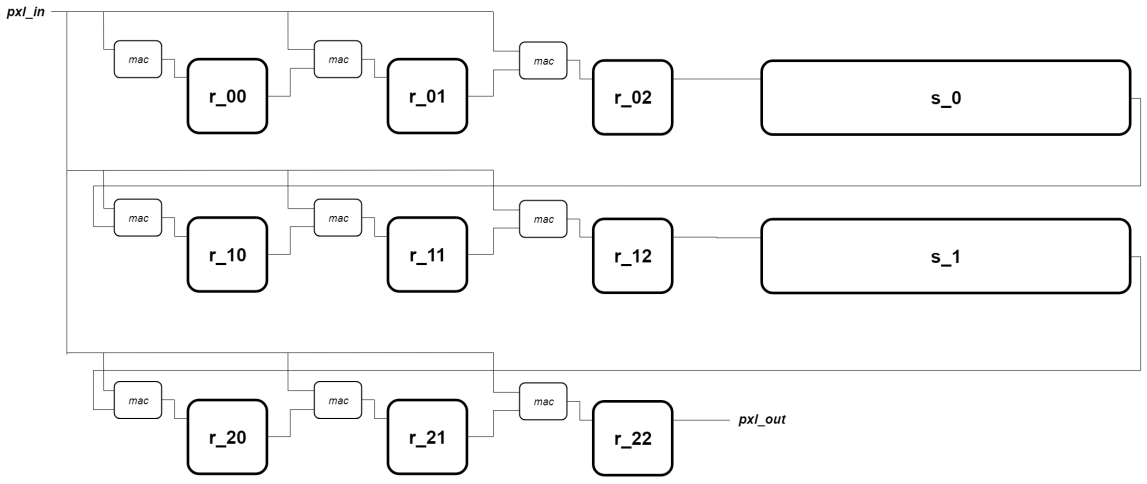

Figura 6: Esquemático da arquitetura *pipeline* do módulo de convolução usado.
Fonte: Autor

Para ser adaptado aos tipos de dados da aplicação, as quantidades de bits de entrada e saída do módulo foram atualizadas para 16 e 32 bits, respectivamente. O módulo foi inicialmente gravado e testado considerando uma matriz 5×5 como entrada, mas posteriormente alterado para suportar uma entrada de dimensão 28×28 pixels.

Ele foi inicialmente validado por meio do *ModelSim*, cujo resultado confirmando seu funcionamento e apresentado na figura 7.

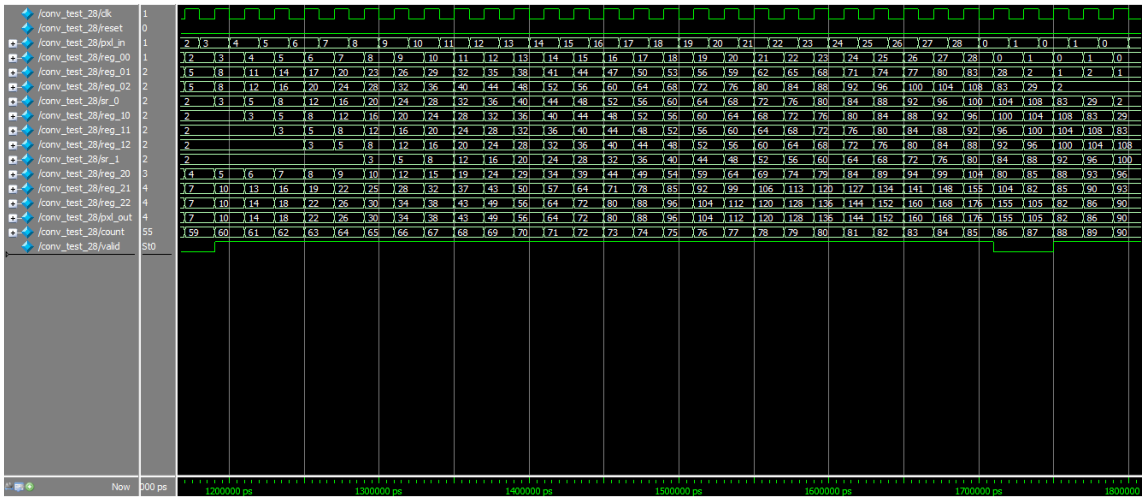

Figura 7: Resultado da simulação feita no *ModelSim* para validação do módulo de convolução.

Fonte: Autor

Também foi realizada uma validação do módulo gravado na FPGA por meio da execução da aplicação e gravação do *hardware*, consistindo no envio de dados de *benchmark* com saídas a cada instante conhecidas (inclusos os valores não válidos). Os dados foram verificados com o auxílio do *display* de 7 segmentos, dos LEDs e do valor recebido pela aplicação.

Também foram verificados, corrigidos e validados o funcionamento do *reset* e pequenas falhas na sincronização e cálculo dos pixels válidos de saída, especialmente devido ao uso de ambas as bordas do sinal de sincronização.

5.6 Gravação, Teste e Validação da Interface Completa — Integração à Aplicação de Classificação

Uma vez verificado o funcionamento do *reset*, envio da matriz de pixels e o recebimento da convolução, essas operações puderam ser modularizadas e incluídas na aplicação principal de treinamento e teste da base de dígitos no formato de uma função de convolução chamada pelo programa.

Entretanto foram incluídas adaptações adicionais como a multiplicação dos coeficientes do filtro por 100 para obter ao menos duas casas decimais de precisão (como as

operações em *hardware* são feitas com valores inteiros) e subsequente pós-tratamento dos resultados da convolução: tomar o complemento de dois se necessário e dividir o valor por 100.

Para determinar o tempo médio de execução, diversas execuções foram realizadas e foram comparadas as taxas de acerto usando as convoluções em *hardware* e em *software*.

6 Resultados e Discussão

Durante o desenvolvimento do projeto o grupo encontrou diversas dificuldades. Primeiramente, o kit de desenvolvimento DE2i-150 adotado apresentou falha no hardware durante as primeiras tentativas de instalação do sistema operacional *Xubuntu*. Este problema não somente recorreu em todas as placas disponíveis no laboratório, como também apresentou comportamento não determinístico. Após diversas tentativas de instalação o SO foi instalado, porém sem identificar a origem do problema que o causou. E mesmo após este evento o kit apresentou por vezes requereu ser reiniciado diversas vezes para inicializar corretamente, especialmente após carregado o *bitstream* contendo o módulo de acesso ao PCIex na FPGA.

Em seguida o grupo requereu mais tempo que o previsto para a instalação dos *drivers* necessários, porém isto não foi obstrutivo para o cronograma em geral. Uma maior dificuldade no entanto foi a comunicação pelo barramento *PCI Express* entre o processador do microcontrolador e o circuito de *Field-Programmable Gate Array* (FPGA).

Em particular para realizar uma comunicação efetiva e devido à natureza síncrona do funcionamento do módulo de convolução, fez-se necessária a geração de um sinal de sincronização a partir da aplicação em *software*. Como consequência direta do funcionamento síncrono da convolução, também determinou-se necessária a operação tanto na borda de subida quanto na borda de descida do sinal de sincronismo, de forma a garantir que o dado estivesse disponível e estável na entrada do módulo de convolução na transição de *clock* usada.

Uma falha fatal encontrada durante o desenvolvimento causando um congelamento total tanto do sistema operacional do microcomputador quanto da FPGA da plataforma foi atribuída à conexão direta dos barramentos PCIex ao módulo de convolução. Após ocorridas instâncias dessa falha, fizeram-se necessárias a aplicação de um *cold reboot* na FPGA e que o *bitstream* fosse regravado. A solução aplicada como *workaround* para essa falha foi a bufferização de ambas a entrada e a saída do módulo de convolução.

Um erro mais simples encontrado no funcionamento foi o retorno dos valores binários da FPGA duplicados nos 32 bits, por exemplo, o valor 0x09C2 seria retornado como 0x09C209C2. Foi então aplicada uma máscara no formato 0x0000FFFF para corretamente formatar os dados. Outro pós-processamento feito foi tomar o complemento de dois para valores negativos retornados pela FPGA, quando necessário.

Finalmente, os resultados apresentados na tabela 2, especialmente ligados ao tempo de convolução revelam que a solução otimizada em *hardware* é em torno de 20 vezes mais lenta que a solução em *software*.

Exec.	**Conv. (ms)**	***Relu* (ms)**	***Pool* (ms)**	**FC (ms)**	**Acertos (%)**
1	32723	90.72	163.5	976.3	9.91
2	32858	92.79	165.1	992.5	9.91
3	35509	95.30	170.0	1014.5	9.91
4	35176	95.56	165.6	997.5	9.91
5	34772	91.90	164.1	987.6	9.91
6	32760	90.14	161.8	972.5	9.91
7	34650	92.17	164.6	979.8	9.91
8	34177	90.53	162.1	971.8	9.91
9	33923	90.40	162.0	971.7	9.91
10	34853	93.10	164.5	985.9	9.91
Média:	34140	92.26	164.3	985.0	9.91

Tabela 2: Tempos de execução e taxas de acerto de cada um dos blocos da rede CNN para a base de dados completa em diversas execuções e usando a implementação de convolução em *hardware*.

Comenta-se que a principal causa do aumento no tempo de execução é o tempo gasto na comunicação, dependente de operações de escrita em arquivos externos e portanto de acesso à memória secundária, uma operação particularmente custosa para o sistema operacional. Em termos dos resultados observados também na tabela 2 relativos à acurácia de classificação usando a convolução implementada em *hardware*, a falta de acurácia é atribuída ao uso de valores inteiros aproximados para os coeficientes do filtro. Acredita-se que essa perda de precisão afetou grandemente os resultados de classificação finais.

É importante apontar que o desenvolvimento da rede em si antes desta ser portada para dentro do embarcado não apresentou complicações, dada a possibilidade de testá-la anteriormente com a base de dados MNIST. No entanto, devido tanto um mau funcionamento da câmera Therasic THDB-D5M e pela complexidade de se projetar a comunicação com a mesma optou-se por utilizar como teste da rede um banco com imagens estáticas. O mau funcionamento em específico foi o aumento incremental da imagem da câmera em conjunto com a queda de aquisição de quadros.

7 Conclusão

O projeto desenvolvido foi capaz de implementar, testar e validar um método de convolução otimizado em *hardware* usado em uma aplicação de rede do tipo CNN para classificação de dígitos manuscritos, portanto o objetivo geral do projeto foi atingido. O objetivo específico de fazer uso do barramento dedicado de comunicação PCIex entre a FPGA e o microcomputador também foi atingido com sucesso.

Entretanto os resultados obtidos ao final do projeto deixaram a desejar, especialmente avaliando o tempo de execução, a solução em *hardware* retornando em torno de 20 vezes pior quando comparada à solução em *software* na mesma plataforma.

Algumas otimizações poderiam ser aplicadas houvesse mais tempo de desenvolvimento para o projeto. Esses otimizações são listadas a seguir como possíveis continuidades para o projeto, particularmente úteis para futuros alunos que desejem desenvolvê-las no contexto de disciplinas similares no futuro ou mesmo de um projeto de conclusão de curso ou de pesquisa.

7.1 Sugestões de Otimizações do Projeto ou Trabalhos Futuros

A principal solução que deveria ser aplicada mas não foi é o uso de dados no formato ponto flutuante para manter uma precisão aceitável nas operações de convolução.

Algumas outras sugestões de continuidade do projeto que podem ser aplicadas em extensões deste ou poderiam ter sido aplicadas houvesse mais tempo de desenvolvimento são listadas a seguir:

1. Conforme sugerido pelo prof. Vanderlei Bonato, otimizar o protocolo de comunicação fazendo uso de ambas as bordas do sinal de sincronização para escrita no *buffer* de entrada da FPGA, dividindo o tempo de execução por 2;
2. Enviar inteiros de 8 bits a cada ciclo, podendo enviar até 4 inteiros por vez e dividindo o tempo de execução por 4;
3. Retornar inteiros de 16 ou 8 bits como resultado da convolução, podendo dividir o tempo de execução em por 2 ou 4;
4. Implementar as demais camadas da rede CNN (*pooling*, *relu* e FC) também em *hardware* e cortar toda a comunicação de retorno da matriz resultado da convolução, retornando apenas o resultado final da classificação e dividindo o tempo de execução por 2.

Como solução adicional, mas de extrema dificuldade de implantação, sugere-se alterar o *driver* do barramento PCIex para que ele faça referência a um endereço de memória primária do microcomputador ao invés de um arquivo armazenado em memória secundária.

Ao executar a aplicação com permissões adequadas para acesso a um endereço fixo da memória primária, o envio e recebimento de dados pelo barramento poderão ser feitos por meio do acesso a esse endereço, uma operação bem menos custosa ao microcomputador do que a escrita e leitura em memória secundária.

A Códigos Fonte: Linguagem C

Neste apêndice são listados os códigos fonte em linguagem C usados e desenvolvidos durante as diversas etapas do projeto, todos executados no microcomputador da FPGA.

Script 1: Código da aplicação responsável pelas leituras e escritas de teste para o barramento PCIex.

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

#define SIZE 28
#define KERNEL 3

// retorna 1 se foi escrito um bit valido no endereco 'receive'
// senao, retorna 0
int read_write_data(int dev, int data, int* receive) {
   // sinal de clock passado no bit [31], ele vai estar em nivel logico
       alto
   // borda descida (subida no modulo conv)
   data = data & 0x0000FFFF;
   write(dev, &data, 4);

   // subida (descida no modulo conv)
   data = (data | 0x80000000);
   write(dev, &data, 4);

   // leitura do valor atualizado
   read(dev, receive, 4);

   // bit '~valid' retornado na posicao 31: se o resultado
   // for negativo a saida nao eh valida
   return ((*receive) == 0xFFFFFFFF) ? 0 : 1;

   /*
   if ((*receive) == (0xFFFFFFFF)) {
      return 0;
   }
   else {
      //*receive = (~(*receive) + 1) & (0x00FF);
```

```
      return 1;
   }*/
}

int sti(int s) {
   int i;

   // complemento de dois: repetir o bit mais significativo a esquerda
   if ((s & 0x00008000) == 0x00008000) return 0xFFFF0000 | s;
   else return 0x00000000 | s;
}

// inicializar convolucao fpga, resetando registradores de convolucao
// e deixando clk em nivel logico baixo
void reset(int dev) {
   // binario 1 no bit 30 da entrada --> sinal de reset
   // gera um pulsos com sinal de reset ativo, desativa reset
   // e deixa o clock em nivel logico baixo
   int data = 0x70000000;
   write(dev, &data, 4);
   data = 0xF0000000;
   write(dev, &data, 4);

   data = 0x70000000;
   write(dev, &data, 4);
   data = 0xF0000000;
   write(dev, &data, 4);

   // ao final, deixa o clock em nivel logico baixo sem reset
   //data = 0x00000000;
   //write(dev, &data, 4);
}

int main() {
   int img_in[SIZE][SIZE];
   float img_out[SIZE-KERNEL+1][SIZE-KERNEL+1];
   int i, j;
   int dev = open("/dev/de2i150_altera", O_RDWR);
   //int receive[SIZE*SIZE];
   int receive;
   int inc = 0;
   int read_pos = 0;
   int m = 0;
```

```
   int n = 0;
   //signed short s = -2;

   //printf("%X, %X, %X, %X\n", s, sti(s), (int) s, (int) -s);
   //getchar();

   printf("Device: %d\n", dev);

   // Para delay inicial
   // ( primeiro dado nunca eh lido/retornado)
   //write_data(dev, 1);
   //read_data(dev);

   // declaracao da matriz de dados
   int k = 1;
   printf("Dados enviados: \n");
   for (i = 0; i < SIZE; i++){
      if ((i % 2) == 0) k = 1;
      else k = 0;
      for (j = 0; j < SIZE; j++){
        /*if ((i % 2) == 0) {
         img_in[i][j] = (0xFFFF) & k;
         k = k + 1;
        } else {
         img_in[i][j] = (0xFFFF) & k;
         k = (k == 0) ? 1 : 0;
        }*/
        img_in[i][j] = 1;
        printf("%X ", img_in[i][j]);
        img_out[i][j] = 0;
      }
      printf("\n");
   }

   getchar();
/*
   for (i = 0; i < SIZE; i++) {
      for (j = 0; j < SIZE; j++) {
         printf("%X ", img_in[i][j]);
      }
      printf("\n");
   }*/
```

```
// resetar registradores de convolucao
reset(dev);

// envio e leitura dos dados
printf("Envio e Leitura\n");
for (i = 0; i < SIZE; i++){
   for (j = 0; j < SIZE; j++){
      inc = read_write_data(dev, img_in[i][j], &receive);
      if (inc) {
         img_out[m][n++] = sti(receive & 0x0000FFFF)/100.0;
         if (n > (SIZE - KERNEL)) {
            n = 0;
            m++;
         }
      }
      //printf("write[%d][%d]: %d, ", i, j, img_in[i][j]);
      //printf("read: %d\n", sti(receive & 0x0000FFFF));
      //read_pos += inc;
      //getchar();
   }
}

for (i = 0; i < KERNEL; i++) {
   inc = read_write_data(dev, 1, &receive);
   if (inc) {
      img_out[m][n++] = sti(receive & 0x0000FFFF);
   }
   //printf("write[%d][bonus]: thr, ", i);
   //printf("read: %X\n", (receive & 0x0000FFFF));
   //read_pos += inc;
   //getchar();
}

printf("Convoluted result: \n");
for (i = 0; i < SIZE-KERNEL+1; i++) {
   for (j = 0; j < SIZE-KERNEL+1; j++) {
      //img_out[i][j] = receive[i*(SIZE-KERNEL+1) + j] & 0x0000FFFF;
      printf("%.2f ", img_out[i][j]);
   }
   printf("\n");
}
```

```
   //printf("%read_pos: d\n", read_pos);

   close(dev);
   return 0;
}
```

Script 2: Código da aplicação completa de treinamento (em C++), teste e exibição dos resultados de acurácia e tempos de execução dos diferentes blocos, incluindo a função de convolução implementada em *hardware*.

```
#include <cassert>
#include <cstdint>
#include <cstdio>
#include <iostream>
#include <fstream>
#include <algorithm>
#include "byteswap.h"
#include "CNN/cnn.h"
#include <chrono>

#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

#define SIZE 28
#define KERNEL 3

int dev;

using namespace std;

typedef std::chrono::milliseconds ms;

float train( vector<layer_t*>& layers, tensor_t<float>& data,
    tensor_t<float>& expected )
{
   for ( int i = 0; i < layers.size(); i++ )
   {
      if ( i == 0 )
         activate( layers[i], data );
      else
         activate( layers[i], layers[i - 1]->out );
   }
```

```
   tensor_t<float> grads = layers.back()->out - expected;

   for ( int i = layers.size() - 1; i >= 0; i-- )
   {
      if ( i == layers.size() - 1 )
         calc_grads( layers[i], grads );
      else
         calc_grads( layers[i], layers[i + 1]->grads_in );
   }

   for ( int i = 0; i < layers.size(); i++ )
   {
      fix_weights( layers[i] );
   }

   float err = 0;
   for ( int i = 0; i < grads.size.x * grads.size.y * grads.size.z; i++ )
   {
      float f = expected.data[i];
      if ( f > 0.5 )
         err += abs(grads.data[i]);
   }
   return err * 100;
}

float validate( vector<layer_t*>& layers, tensor_t<float>& data,
    tensor_t<float>& expected )
{
   for ( int i = 0; i < layers.size(); i++ )
   {
      if ( i == 0 )
         activate( layers[i], data );
      else
         activate( layers[i], layers[i - 1]->out );
   }

   tensor_t<float> grads = layers.back()->out - expected;

   float err = 0;
   for ( int i = 0; i < grads.size.x * grads.size.y * grads.size.z; i++ )
   {
      float f = expected.data[i];
```

```
      if ( f > 0.5 )
         err += abs(grads.data[i]);
   }
   return err * 100;
}

//*************************************************************************\\
//*************** FUNCOES DE ACESSO/LEITURA/ESCRITA NA FPGA
    ************\\
//*************************************************************************\\


// inicializar convolucao fpga, resetando registradores de convolucao
// e deixando clk em nivel logico baixo
void fpga_reset(int dev) {
   // binario 1 no bit 30 da entrada --> sinal de reset
   // gera um pulsos com sinal de reset ativo, desativa reset
   // e deixa o clock em nivel logico baixo
   int data = 0x70000000;
   write(dev, &data, 4);
   data = 0xF0000000;
   write(dev, &data, 4);

   data = 0x70000000;
   write(dev, &data, 4);
   data = 0xF0000000;
   write(dev, &data, 4);
}

// faz conversao de uma palavra de 16 bits para uma de 32 bits
// (tratamento para complemento de dois)
int sti(int s) {
   int i;

   // complemento de dois: repetir o bit mais significativo a esquerda
   if ((s & 0x00008000) == 0x00008000) return 0xFFFF0000 | s;
   else return 0x00000000 | s;
}

// retorna 1 se foi escrito um bit valido no endereco 'receive'
// senao, retorna 0
int fpga_read_write_data(int dev, int data, int* receive) {
   // sinal de clock passado no bit [31], ele vai estar em nivel logico
```

```
      alto
   // borda descida (subida no modulo conv)
   data = data & 0x0000FFFF;
   write(dev, &data, 4);

   // subida (descida no modulo conv)
   data = (data | 0x80000000);
   write(dev, &data, 4);

   // leitura do valor atualizado
   read(dev, receive, 4);

   // valor 0xFFFFFFFF inidica um resultado nao valido: se o resultado
   // for negativo a saida nao eh valida
   return ((*receive) == 0xFFFFFFFF) ? 0 : 1;
}

//*************************************************************************\\
//************************ FIM DAS FUNCOES DA FPGA
    **********************\\
//*************************************************************************\\

void my_convolution (vector<layer_t*>& layers, tensor_t<int>& data_fpga){
   // HARDWARE CONVOLUTION

   // ponteiro de arquivo global e ponteiro inicializado e fechado na main
   //int dev = open("/dev/de2i150_altera", O_RDWR);
   int receive;
   int i, j, inc;
   int m = 0;
   int n = 0;
   int qtde_filtros = 1;

   // resetar registradores de convolucao
   fpga_reset(dev);

   // envio e leitura dos dados
   //printf("Envio e Leitura\n");
   for (int z = 0; z < qtde_filtros; z++) {
      for (i = 0; i < SIZE; i++){
         for (j = 0; j < SIZE; j++){
            inc = fpga_read_write_data(dev, data_fpga(i, j, 0), &receive);
            //printf("%d ", data_fpga(i,j,0)); // test
```

```
      // aplicar mascara e dividir por 100 (coeficientes reescalados para
      // fazer apenas operacoes com inteiros na fpga)
      if (inc) {
              layers[0]->out(m,n++,z) = ((float) sti(receive &
                  0x0000FFFF)) / 100.0;
              if (n > (SIZE - KERNEL)) {
                 n = 0;
                 m++;
              }
           }
        }
      }
      // loop adicional para pegar ultima linha
   for (i = 0; i < 2 * KERNEL; i++) {
      inc = fpga_read_write_data(dev, 1, &receive);
      if (inc) {
         layers[0]->out(m,n++,z) = sti(receive & 0x0000FFFF) / 1000.0;
         if (n > (SIZE - KERNEL)) {
                 n = 0;
                 m++;
              }
      }
   }
   }
/*
   printf("Convoluted result: \n");
   for (i = 0; i < SIZE-KERNEL+1; i++) {
      for (j = 0; j < SIZE-KERNEL+1; j++) {
         printf("%.2f ", layers[0]->out(i,j,0));
      }
      printf("\n");
   }
   getchar();*/
/*
   //Sending...
   printf("Imagem\n\n");
   for (int x = 0; x < 28; x++) {
       for (int y = 0; y < 28; y++) {
        //send_to_fpga(data_fpga(x,y,0);

        //Test
        printf("%d ", data_fpga(x,y,0));
```

```
        }
        printf("\n");
    }

    int qtd_filtros = 1;

    //Receiving...
    printf("Convolucao\n\n");
    for (int z = 0; z < qtd_filtros; z++) {
        for (int x = 0; x < 26; x++) {
            for (int y = 0; y < 26; y++) {
                //layers[0]->out(x,y,z) = receive_from_fpga();

                //Test
                //layers[0]->out(x,y,z) = y + x*26;
                //printf("%f ", layers[0]->out(x,y,z));

            }
            //printf("\n");
        }
    }
*/
}

void convolution (vector<layer_t*>& layers, tensor_t<float>& data){
    activate (layers[0], data);
}

void relu (vector<layer_t*>& layers, tensor_t<float>& data){
    activate(layers[1], layers[0]->out);
}

void pool (vector<layer_t*>& layers, tensor_t<float>& data){
    activate(layers[2], layers[1]->out);
}

void fc (vector<layer_t*>& layers, tensor_t<float>& data){
    activate(layers[3], layers[2]->out);
}

void forward( vector<layer_t*>& layers, tensor_t<float>& data )
{
```

```
    convolution(layers, data);
    relu(layers, data);
    pool(layers, data);
    fc(layers, data);

    /*
   for ( int i = 0; i < layers.size(); i++ )
   {
      if ( i == 0 )
         activate( layers[i], data );
      else
         activate( layers[i], layers[i - 1]->out );
   }
    */

}

struct case_t
{
   tensor_t<float> data;
    tensor_t<int> data_fpga;
   tensor_t<float> out;
};

uint8_t* read_file( const char* szFile )
{
   ifstream file( szFile, ios::binary | ios::ate );
   streamsize size = file.tellg();
   file.seekg( 0, ios::beg );

   if ( size == -1 )
      return nullptr;

   uint8_t* buffer = new uint8_t[size];
   file.read( (char*)buffer, size );
   return buffer;
}

vector<case_t> read_train_cases()
{
   vector<case_t> cases;

   uint8_t* train_image = read_file( "train-images.idx3-ubyte" );
```

```
    uint8_t* train_labels = read_file( "train-labels.idx1-ubyte" );

    //Relacionado com arquivo MNIST (http://yann.lecun.com/exdb/mnist/)
    //Leitura do tamanho em Big-endian
    uint32_t case_count = byteswap_uint32( *(uint32_t*)(train_image + 4) );
    for ( int i = 0; i < case_count; i++ )
    {
        //Tamanho dos dados para ler do arquivo MNIST (dado = 28x28,
            sada=10 classes)
        case_t c {tensor_t<float>( 28, 28, 1 ), tensor_t<int>( 28, 28, 1 ),
            tensor_t<float>( 10, 1, 1 )};

        //Leitura pixel a pixel (row-wise)
        uint8_t* img = train_image + 16 + i * (28 * 28);

        //Leitura dos labels
        uint8_t* label = train_labels + 8 + i;

        for ( int x = 0; x < 28; x++ ){
            for ( int y = 0; y < 28; y++ ){
                c.data( x, y, 0 ) = img[x + y * 28] / 255.f;
                    c.data_fpga( x, y, 0 ) = img[x + y * 28];
                }
            }

        for ( int b = 0; b < 10; b++ )
            c.out( b, 0, 0 ) = *label == b ? 1.0f : 0.0f;

        cases.push_back( c );
    }
    delete[] train_image;
    delete[] train_labels;

    return cases;
}

vector<case_t> read_test_cases()
{
    vector<case_t> cases;

    uint8_t* test_image = read_file( "t10k-images-idx3-ubyte" );
    uint8_t* test_labels = read_file( "t10k-labels-idx1-ubyte" );
```

```
    //Relacionado com arquivo MNIST (http://yann.lecun.com/exdb/mnist/)
    //Leitura do tamanho em Big-endian
    uint32_t case_count = byteswap_uint32( *(uint32_t*)(test_image + 4) );
    for ( int i = 0; i < case_count; i++ )
    {
        //Tamanho dos dados para ler do arquivo MNIST (dado = 28x28,
            sada=10 classes)
        case_t c {tensor_t<float>( 28, 28, 1 ), tensor_t<int>( 28, 28, 1 ),
            tensor_t<float>( 10, 1, 1 )};

        //Leitura pixel a pixel (row-wise)
        uint8_t* img = test_image + 16 + i * (28 * 28);

        //Leitura dos labels
        uint8_t* label = test_labels + 8 + i;

        for ( int x = 0; x < 28; x++ )
            for ( int y = 0; y < 28; y++ )
                c.data( x, y, 0 ) = img[x + y * 28] / 255.f;

        for ( int b = 0; b < 10; b++ )
            c.out( b, 0, 0 ) = *label == b ? 1.0f : 0.0f;

        cases.push_back( c );
    }
    delete[] test_image;
    delete[] test_labels;

    return cases;
}


int main()
{
    //Leitura da base de dados - treino
    vector<case_t> cases = read_train_cases();

     vector<case_t> cases_test = read_test_cases();

    dev = open("/dev/de2i150_altera", O_RDWR);
    printf("Device: %d\n", dev);
        if (dev < 0) {
            printf("Erro ao acessar barramento!\n");
```

```
        return -1;
    }

    vector<layer_t*> layers;

    //1 = passo do filtro
    //3 = extenso do filtro (tamanho do filtro)
    //1 = nmero de filtros
    //size = tamanho da entrada
    conv_layer_t * layer1 = new conv_layer_t( 1, 3, 1, cases[0].data.size
        );
    relu_layer_t * layer2 = new relu_layer_t( layer1->out.size );

    //1 = Passo do filtro
    //1 = Tamanho do filtro
    pool_layer_t * layer3 = new pool_layer_t( 1, 1, layer2->out.size );
    fc_layer_t * layer4 = new fc_layer_t(layer3->out.size, 10);

    layers.push_back( (layer_t*)layer1 );
    layers.push_back( (layer_t*)layer2 );
    layers.push_back( (layer_t*)layer3 );
    layers.push_back( (layer_t*)layer4 );

    double amse = 0, amse_test = 0;
    long ic = 0, ic_test = 0;

    for ( int ep = 1; ep <= 1; ep++ )
    {
       for ( case_t& t : cases )
       {
          float xerr = train( layers, t.data, t.out );
          amse += xerr;

          ic++;

          // if ( GetAsyncKeyState( VK_F1 ) & 0x8000 )
          // {
          //    printf( "err=%.4f%\n", amse / ic );
          //    goto end;
          // }
       }
       cout << "Train " << ep << " err=" << amse/ic << endl;
/*
```

```
        for ( case_t& t : cases_test )
      {
         float xerr = validate( layers, t.data, t.out );
         amse_test += xerr;

         ic_test++;

         // if ( GetAsyncKeyState( VK_F1 ) & 0x8000 )
         // {
         //    printf( "err=%.4f%\n", amse / ic );
         //    goto end;
         // }
      }
      cout << "Validation " << ep << " err=" << amse_test/ic_test << endl;
*/
   }

   cout << "Your filters:" << endl;

   for (int i = 0; i < layer1->filters.size(); i++){
      cout << "---------------------------------"<<endl;
      for (int j = 0; j < layer1->filters.at(i).size.x; j++){
         for (int k = 0; k < layer1->filters.at(i).size.y; k++){
            cout << layer1->filters.at(i).get(j,k,0) << " ";
         }
         cout << endl;
      }
      cout << "---------------------------------"<<endl;
   }

   long matrix[10][10];
   int previsto, fato;
   float max;

   for (int i = 0; i < 10; i++){
      for(int j = 0; j < 10; j++){
         matrix[i][j] = 0;
      }
   }

    long hits = 0;
    long iteration = 0;
    double elapsed_conv = 0;
```

```
double elapsed_relu = 0;
double elapsed_pool = 0;
double elapsed_fullcon = 0;

std::chrono::duration<double> elapsed_cv, elapsed_rl, elapsed_pl,
    elapsed_fc;

for ( case_t& t : cases_test )
{
    auto start_cv = std::chrono::high_resolution_clock::now();
    my_convolution(layers, t.data_fpga);
    //convolution(layers, t.data);
    auto finish_cv = std::chrono::high_resolution_clock::now();

    elapsed_cv += finish_cv - start_cv;
    std::chrono::milliseconds cv_ms =
        std::chrono::duration_cast<ms>(elapsed_cv);

    elapsed_conv += cv_ms.count();

    auto start_relu = std::chrono::high_resolution_clock::now();
    relu(layers, t.data);
    auto finish_relu = std::chrono::high_resolution_clock::now();

    elapsed_rl += finish_relu - start_relu;
    std::chrono::milliseconds rl_ms =
        std::chrono::duration_cast<ms>(elapsed_rl);

    elapsed_relu += rl_ms.count();

    auto start_pl = std::chrono::high_resolution_clock::now();
    pool(layers, t.data);
    auto finish_pl = std::chrono::high_resolution_clock::now();

    elapsed_pl += finish_pl - start_pl;
    std::chrono::milliseconds pl_ms =
        std::chrono::duration_cast<ms>(elapsed_pl);

    elapsed_pool += pl_ms.count();

    auto start_fc = std::chrono::high_resolution_clock::now();
    fc(layers, t.data);
    auto finish_fc = std::chrono::high_resolution_clock::now();
```

```
        elapsed_fc += finish_fc - start_fc;
        std::chrono::milliseconds fc_ms =
            std::chrono::duration_cast<ms>(elapsed_fc);

        elapsed_fullcon += fc_ms.count();

//    forward( layers, t.data );
      tensor_t<float>& out = layers.back()->out;

      for ( int i = 0; i < 10; i++ )
      {
         if (t.out(i,0,0) == 1.0){
            fato = i;
            break;
         }
      }

      max = out(0, 0, 0);
      previsto = 0;

      for (int i = 1; i < 10; i++){
         if (out(i,0,0) > max){
            max = out(i, 0, 0);
            previsto = i;
         }
      }

      matrix[fato][previsto] += 1;

        if (fato == previsto) hits += 1;
        iteration += 1;
   }

   for (int i = 0; i < 10; i++){
      for(int j = 0; j < 10; j++){
         printf("%lu ", matrix[i][j]);
      }
      printf("\n");
   }

    cout << "Conv : " << elapsed_conv/iteration << endl;
    cout << "Relu : " << elapsed_relu/iteration << endl;
```

```
    cout << "Pool : " << elapsed_pool/iteration << endl;
    cout << "FC : " << elapsed_fullcon/iteration << endl;

    cout << "Hits: " << ((double) hits)/iteration << endl;
   return 1;

    /*
   while ( true )
   {
      uint8_t * data = read_file( "test.ppm" );

      if ( data )
      {
         uint8_t * usable = data;

         while ( *(uint32_t*)usable != 0x0A353532 )
            usable++;

#pragma pack(push, 1)
         struct RGB
         {
            uint8_t r, g, b;
         };
#pragma pack(pop)

         RGB * rgb = (RGB*)usable;

         tensor_t<float> image(28, 28, 1);
         for ( int i = 0; i < 28; i++ )
         {
            for ( int j = 0; j < 28; j++ )
            {
               RGB rgb_ij = rgb[i * 28 + j];
               image( j, i, 0 ) = (((float)rgb_ij.r
                         + rgb_ij.g
                         + rgb_ij.b)
                        / (3.0f*255.f));
            }
         }

         forward( layers, image );
         tensor_t<float>& out = layers.back()->out;
         for ( int i = 0; i < 10; i++ )
```

```
            {
               printf( "[%i] %f\n", i, out( i, 0, 0 )*100.0f );
            }

            delete[] data;
         }

         struct timespec wait;
         wait.tv_sec = 1;
         wait.tv_nsec = 0;
         nanosleep(&wait, nullptr);
      }
      */
      close(dev);
      return 0;
}
```

B Módulos de Hardware: *Verilog*

Neste apêndice são listados os códigos fonte em linguagem *Verilog* usados e desenvolvidos durante as diversas etapas do projeto para as descrições de *hardware*.

Script 3: Descrição de *hardware* do bloco de registradores.

```
// Register Unit (for simulation)

`timescale 1ns / 1ps

module register(
    input clk,
    input reset,
    input [15:0] in,
    output [15:0] out
    );

reg [15:0] data;
assign out[15:0] = data[15:0];

always @(posedge clk) begin
   if (reset) begin
      data[15:0] <= 16'b0000000000000000;
   end
   else begin
      data[15:0] <= in[15:0];
   end

end

endmodule
```

Script 4: Descrição de *hardware* do bloco *shift*.

```
// Shifts data by a fixed depth.
// Optimize in future by finding a way to create 2-d arrays

module shift
(
  input clk,
  input [15:0] data_in,
  output [15:0] data_out
```

```
);

// Depth = D = n-k; for now assume n to be 5
// --> n = 28 (image 28x28), k = 3 (filter 3x3)
// --> D = 25
parameter D = 25;

// Define holding register for each bit
reg [D-1:0] hr_0; reg [D-1:0] hr_1; reg [D-1:0] hr_2; reg [D-1:0] hr_3;
reg [D-1:0] hr_4; reg [D-1:0] hr_5; reg [D-1:0] hr_6; reg [D-1:0] hr_7;
reg [D-1:0] hr_8; reg [D-1:0] hr_9; reg [D-1:0] hr_10; reg [D-1:0] hr_11;
reg [D-1:0] hr_12; reg [D-1:0] hr_13; reg [D-1:0] hr_14; reg [D-1:0]
    hr_15;

always @ (posedge clk) begin
  hr_0 [D-1:0] <= {hr_0[D-2:0], data_in[0]};
  hr_1 [D-1:0] <= {hr_1[D-2:0], data_in[1]};
  hr_2 [D-1:0] <= {hr_2[D-2:0], data_in[2]};
  hr_3 [D-1:0] <= {hr_3[D-2:0], data_in[3]};
  hr_4 [D-1:0] <= {hr_4[D-2:0], data_in[4]};
  hr_5 [D-1:0] <= {hr_5[D-2:0], data_in[5]};
  hr_6 [D-1:0] <= {hr_6[D-2:0], data_in[6]};
  hr_7 [D-1:0] <= {hr_7[D-2:0], data_in[7]};
  hr_8 [D-1:0] <= {hr_8[D-2:0], data_in[8]};
  hr_9 [D-1:0] <= {hr_9[D-2:0], data_in[9]};
  hr_10 [D-1:0] <= {hr_10[D-2:0], data_in[10]};
  hr_11 [D-1:0] <= {hr_11[D-2:0], data_in[11]};
  hr_12 [D-1:0] <= {hr_12[D-2:0], data_in[12]};
  hr_13 [D-1:0] <= {hr_13[D-2:0], data_in[13]};
  hr_14 [D-1:0] <= {hr_14[D-2:0], data_in[14]};
  hr_15 [D-1:0] <= {hr_15[D-2:0], data_in[15]};

end

assign data_out[0] = hr_0[D-1]; assign data_out[1] = hr_1[D-1];
assign data_out[2] = hr_2[D-1]; assign data_out[3] = hr_3[D-1];
assign data_out[4] = hr_4[D-1]; assign data_out[5] = hr_5[D-1];
assign data_out[6] = hr_6[D-1]; assign data_out[7] = hr_7[D-1];
assign data_out[8] = hr_8[D-1]; assign data_out[9] = hr_9[D-1];
assign data_out[10] = hr_10[D-1]; assign data_out[11] = hr_11[D-1];
assign data_out[12] = hr_12[D-1]; assign data_out[13] = hr_13[D-1];
assign data_out[14] = hr_14[D-1]; assign data_out[15] = hr_15[D-1];
```

```
endmodule
```

Script 5: Descrição de *hardware* do bloco multiplicador e acumulador *mac*.

```
// Multiply Accumulate Unit

`timescale 1ns / 1ps

module mac(
    input [15:0] in,
    input [15:0] w,
    input [15:0] b,
    output [15:0] out
    );

wire [15:0] d;
assign d = w * in;
assign out = d + b;

endmodule
```

Script 6: Descrição de *hardware* do bloco de convolução.

```
`timescale 1ns / 1ps

module conv(
      input clk,
      input reset,
      input [15:0] pxl_in,

   output [31:0] reg_00, output [31:0] reg_01, output [31:0] reg_02,
       output[31:0] sr_0,
   output [31:0] reg_10, output [31:0] reg_11, output [31:0] reg_12,
       output[31:0] sr_1,
   output [31:0] reg_20, output [31:0] reg_21, output [31:0] reg_22,
   output [31:0] pxl_out,
   output [9:0] count,
   output valid
    );

   //Define constants
   parameter N = 28; //Image columns
   parameter M = 28; //Image rows
```

```
parameter K = 3;  //Kernel size

// Intermediate wires
wire [31:0] wire_00; wire [31:0] wire_01; wire [31:0] wire_02;
wire [31:0] wire_10; wire [31:0] wire_11; wire [31:0] wire_12;
wire [31:0] wire_20; wire [31:0] wire_21; wire [31:0] wire_22;

// 3*3 kernel
//integer kernel_00 = 1; integer kernel_01 = 2; integer kernel_02 = 1;
//integer kernel_10 = 0; integer kernel_11 = 0; integer kernel_12 = 0;
//integer kernel_20 = 1; integer kernel_21 = 2; integer kernel_22 = 1;
// NOSSOS FILTROS:
// -0.751269 0.866374  -0.353931
// 0.511118  2.13662   0.286361
// -0.405954 0.887376  -0.222453
integer kernel_00 = -75; integer kernel_01 = 87; integer kernel_02 =
    -35;
integer kernel_10 = 51; integer kernel_11 = 21; integer kernel_12 = 29;
integer kernel_20 = -40; integer kernel_21 = 89; integer kernel_22 =
    -22;

// Row : 1
mac mac_00(pxl_in, kernel_00, 0, wire_00);
register r_00(clk, reset, wire_00, reg_00);

mac mac_01(pxl_in, kernel_01, reg_00, wire_01);
register r_01(clk, reset, wire_01, reg_01);

mac mac_02(pxl_in, kernel_02, reg_01, wire_02);
register r_02(clk, reset, wire_02, reg_02);

shift row_1(clk, reg_02, sr_0);

// Row : 2
mac mac_10(pxl_in, kernel_10, sr_0, wire_10);
register r_10(clk, reset, wire_10, reg_10);

mac mac_11(pxl_in, kernel_11, reg_10, wire_11);
register r_11(clk, reset, wire_11, reg_11);

mac mac_12(pxl_in, kernel_12, reg_11, wire_12);
register r_12(clk, reset, wire_12, reg_12);
```

```
shift row_2(clk, reg_12, sr_1);

// Row : 3
mac mac_20(pxl_in, kernel_20, sr_1, wire_20);
register r_20(clk, reset, wire_20, reg_20);

mac mac_21(pxl_in, kernel_21, reg_20, wire_21);
register r_21(clk, reset, wire_21, reg_21);

mac mac_22(pxl_in, kernel_22, reg_21, wire_22);
register r_22(clk, reset, wire_22, reg_22);

assign pxl_out = reg_22;

// Valid bit logic

reg [9:0] counter = 0;
reg temp = 0;

always @(posedge clk) begin
   if (reset == 1) begin
      temp <= 0;
      counter = 0;
   end else begin
      counter = counter + 1;

      // The logic below needs some revisiting to scale properly
      //  counter > 12 and counter < 27
      if (counter > ((K-1)*N + (K-1)) && counter < (M*N) + (K-1)) begin
         // (counter - 2) % 5 > 1 --> 14, 15, 16; 19, 20, 21; 24, 25, 26
         if ((counter - (K-1)) % N > 1) begin
            temp <= 1;
         end else begin
            temp <= 0;
         end
      end else begin
         temp <= 0;
      end
   end
end

assign count = counter;
assign valid = temp;
```

```
endmodule
```

Script 7: Descrição de *hardware* do bloco de teste para o módulo *conv*.

```
`timescale 1ns / 1ps

module conv_test;

   // Inputs
   reg clk;
   reg reset;
   reg [15:0] pxl_in;

   // Outputs
   wire[31:0] reg_00; wire[31:0] reg_01; wire[31:0] reg_02; wire [31:0]
       sr_0;
   wire[31:0] reg_10; wire[31:0] reg_11; wire[31:0] reg_12; wire [31:0]
       sr_1;
   wire[31:0] reg_20; wire[31:0] reg_21; wire[31:0] reg_22;

   wire [31:0] pxl_out;
   wire [9:0] count;
   wire valid;

   // Instantiate the Unit Under Test (UUT)
   conv uut (
      .clk(clk),
      .reset(reset),
      .pxl_in(pxl_in),
      .reg_00(reg_00),
      .reg_01(reg_01),
      .reg_02(reg_02),
      .sr_0(sr_0),
      .reg_10(reg_10),
      .reg_11(reg_11),
      .reg_12(reg_12),
      .sr_1(sr_1),
      .reg_20(reg_20),
      .reg_21(reg_21),
      .reg_22(reg_22),
      .pxl_out(pxl_out),
      .count(count),
```

```
      .valid(valid)
   );

initial begin
   // Initialize Inputs
   clk = 0;
   reset = 0;
   pxl_in = 0;

   //#20 reset = 0;

   // 5*5 image
   #20 pxl_in = 1; #20 pxl_in = 2; #20 pxl_in = 3; #20 pxl_in = 4; #20
       pxl_in = 5;
   #20 pxl_in = 0; #20 pxl_in = 1; #20 pxl_in = 0; #20 pxl_in = 1; #20
       pxl_in = 0;
   #20 pxl_in = 1; #20 pxl_in = 2; #20 pxl_in = 3; #20 pxl_in = 4; #20
       pxl_in = 5;
   #20 pxl_in = 0; #20 pxl_in = 1; #20 pxl_in = 0; #20 pxl_in = 1; #20
       pxl_in = 0;
   #20 pxl_in = 1; #20 pxl_in = 2; #20 pxl_in = 3; #20 pxl_in = 4; #20
       pxl_in = 5;

   end
   always #10 clk = ~ clk;

endmodule
```

Script 8: Descrição de *hardware* do módulo principal gravado na FPGA — *pcihello*.

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module pcihello(

   ////////////// CLOCK //////////
   CLOCK_50,  // BANK 4
   CLOCK2_50,  // BANK 7
   CLOCK3_50,  // BANK 3A

   ////////////// LED (High Active) //////////
```

```
    LEDG,
    LEDR,

    ////////////// KEY (Active Low) //////////
    KEY,

    // switches
    SW,

    ////////////// SEG7 (Low Active) //////////
    HEX0,
    HEX1,
    HEX2,
    HEX3,
    HEX4,
    HEX5,
    HEX6,
    HEX7,

    ////////////// PCIe //////////
    PCIE_PERST_N,
    PCIE_REFCLK_P,
    PCIE_RX_P,
    PCIE_TX_P,
    PCIE_WAKE_N,

    ////////////// Fan Control //////////
    FAN_CTRL
);

//=======================================================
//  PARAMETER declarations
//=======================================================


//=======================================================
//  PORT declarations
//=======================================================

////////////// CLOCK //////////
input                       CLOCK_50;
input                       CLOCK2_50;
input                       CLOCK3_50;
```

```
////////////// LED (High Active) //////////
output          [8:0]    LEDG;
output         [17:0]    LEDR;

////////////// KEY (Active Low) //////////
input           [3:0]    KEY;
input           [17:0]   SW;

////////////// SEG7 (Low Active) //////////
output          [6:0]    HEX0;
output          [6:0]    HEX1;
output          [6:0]    HEX2;
output          [6:0]    HEX3;
output          [6:0]    HEX4;
output          [6:0]    HEX5;
output          [6:0]    HEX6;
output          [6:0]    HEX7;

////////////// PCIe //////////
input                    PCIE_PERST_N;
input                    PCIE_REFCLK_P;
input           [1:0]    PCIE_RX_P;
output          [1:0]    PCIE_TX_P;
output                   PCIE_WAKE_N;

////////////// Fan Control //////////
inout                    FAN_CTRL;


//=======================================================
//  REG/WIRE declarations
//=======================================================

parameter SIZE = 5;
parameter KERNEL = 3;

wire [31:0] outbus;
wire [31:0] inbus;

wire clk_boy;
wire conv_clk;
wire [31:0] tmp;
```

```
wire [31:0] pxl_out;
wire valid;
wire reset;
reg [31:0] img[27:0][27:0];
reg [31:0] outBuf = 32'b00000000000000000000000000000000;
reg [31:0] inBuf = 32'b00000000000000000000000000000000;

//TEST
wire [9:0] count;

integer pixelx = -1;
integer pixely = -1;
integer send_receive = 0;

wire [31:0] reg_00; wire [31:0] reg_01; wire [31:0] reg_02;
wire [31:0] reg_10; wire [31:0] reg_11; wire [31:0] reg_12;
wire [31:0] reg_20; wire [31:0] reg_21; wire [31:0] reg_22;


integer i;
integer j;
initial begin
    for (i=0; i<=27; i=i+1)
      for (j=0; j<=27; j=j+1)
          img[i][j] = 32'b00000000000000000000000000000000;
end

//=======================================================
//  Structural coding
//=======================================================

    pcihellocore u0 (
        .pcie_hard_ip_0_rx_in_rx_datain_0   (PCIE_RX_P[0]),     //
           pcie_hard_ip_0_rx_in.rx_datain_0
        .pcie_hard_ip_0_tx_out_tx_dataout_0 (PCIE_TX_P[0]),   //
           pcie_hard_ip_0_tx_out.tx_dataout_0
        .pcie_hard_ip_0_powerdown_pll_powerdown (PCIE_WAKE_N), //
           pcie_hard_ip_0_powerdown.pll_powerdown
        .pcie_hard_ip_0_powerdown_gxb_powerdown (PCIE_WAKE_N), //
                           .gxb_powerdown
        .pcie_hard_ip_0_refclk_export       (PCIE_REFCLK_P),        //
           pcie_hard_ip_0_refclk.export
        .pcie_hard_ip_0_pcie_rstn_export    (PCIE_PERST_N),
```

```
        .hexport_external_connection_export (inbus),      //
           hexport_external_connection.export
        .inport_external_connection_export  (outbus)      //
           inport_external_connection.export
    );


    conv c0 (
      .clk (~clk_boy),
      .reset (reset),
      .pxl_in (inBuf[15:0]),
      .reg_00 (reg_00), .reg_01 (reg_01), .reg_02 (reg_02),
      .reg_10 (reg_10), .reg_11 (reg_11), .reg_12 (reg_12),
      .reg_20 (reg_20), .reg_21 (reg_21), .reg_22 (reg_22),
      .pxl_out (pxl_out),
      .count (count),
      .valid (valid)
   );

   ////////////// FAN Control //////////
//assign FAN_CTRL = 1'b0; // turn off FAN

assign clk_boy = inbus[31];
assign conv_clk = ~clk_boy;
assign outbus[31:0] = outBuf[31:0];
assign reset = SW[0] | inbus[30];

always@(posedge clk_boy) begin
   // salva valor no buffer de entrada (que eh enviado
   // para o modulo de convolucao)
   inBuf[31:0] = inbus[31:0];

   if (valid == 1) begin
      outBuf[31:0] = pxl_out[31:0];
   end else begin
      outBuf[31:0] = 32'b11111111111111111111111111111111;
   end
end

assign LEDG[7] = valid;
assign LEDG[6:0] = count[6:0];

assign LEDR[17] = reset;
```

```
assign LEDR[16] = ~valid;
assign LEDR[15:0] = outBuf[15:0];

assign HEX7 = inBuf[31:24];
assign HEX6 = inBuf[23:16];
assign HEX5 = inBuf[15:8];
assign HEX4 = inBuf[7:0];

assign HEX0 = outBuf[7:0];
assign HEX1 = outBuf[15:8];
assign HEX2 = outBuf[23:16];
assign HEX3 = outBuf[31:24];

endmodule
```