A Multi-FPGA Distributed Embedded System for the Emulation of Multi-Layer CNNs in Real Time Video Applications

J. Javier Martínez-Alvarez , Javier Toledo-Moreo, Javier Garrigós-Guerrero, J.Manuel Ferrandez-Vicente
Dept. de Electrónica, Tecnología de Computadoras
Universidad Politécnica de Cartagena
Cartagena, Spain
jjavier.martinez@upct.es

***Abstract*— This paper describes the design and the implementation of an embedded system based on multiple FPGAs that can be used to process real time video streams in standalone mode for applications that require the use of large Multi-Layer CNNs (ML-CNNs). The system processes video in progressive mode and provides a standard VGA output format. The main features of the system are determined by using a distributed computing architecture, based on Independent Hardware Modules (IHM), which facilitate system expansion and adaptation to new applications. Each IHM is composed by an FPGA board that can hold one or more CNN layers. The total computing capacity of the system is determined by the number of IHM used and the amount of resources available in the FPGAs. Our architecture supports traditional cloned templates, but also the (simultaneous) use of time-variant and space-variant templates.**

Keywords-component; Multi-FPGA; Embedded Syseim, Emulation CNN

I. INTRODUCTION

Different FPGA-based standalone systems have been proposed to digitally emulate CNNs. Between the most recent, in [1] the authors make use of the Falcon [2] architecture, while in [3] a new approach is developed, derived from the model proposed for the Carthago architecture [4]. In this paper, our interest is to develop a multi-FPGA distributed system, allowing the versatile emulation of large ML-CNNs. The proposed system must be able to target standalone applications processing real time video channels. The novel aspects that our system incorporates are, in first place, a distributed structure, consisting of multiple FPGAs, enabling increased computing capacity; secondly, it has been adapted to support sequential architectures, like Carthago, which was designed and implemented to achieve an excellent compromise between area and speed.

The main features of the system are derived by using a distributed computing architecture, based on Independent Hardware Modules (IHM), which facilitate system expansion and adaptation to new applications. Our aim is to emulate a CNN by distributing the electronic circuitry through different IHMs in order to increase the versatility and computational performance of the platform. The system will support an expansion interface to allow for cascading of multiple IHM modules and facilitate the incorporation of new modules to the system. Every IHM processes a video data stream received at its input interface and provides an output compatible with the input of the next module. Each IHM is composed by an FPGA board that can hold one or more CNN layers. The total computing capacity of the system is determined by the number of IHM used and the amount of resources available in the FPGAs. The system incorporates a frame grabber which corrects the latency introduced by the IHM and provides a VGA compatible output. Fig. 1 shows the structure of the proposed system.

Both, IHMs and interface blocks, must be designed to work in a data flow configuration, and support a distributed sequential computing architecture. For this, IHM modules must support Processing Elements (PEs) pipelining to run multiple Euler iterations simultaneously, while IHM interfaces have to be able to support and propagate the control signals and data buses in real time with enough versatility.

To facilitate the understanding of the proposed methodology and to verify its functionality, a system has been developed for the particular case in which both the IHMs and theirs expansion interfaces have been adapted to the CNN Carthago architecture.

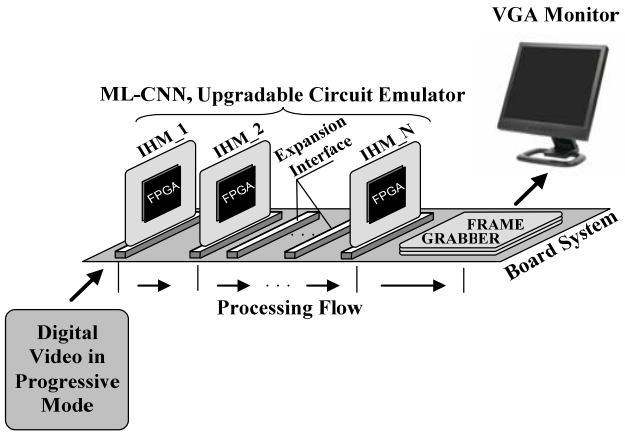


Figure 1. Outline of the proposed distributed computing system

II. CARTHAGO-BASED IHM MODULE

The electronic circuit inside the IHM and the expansion interfaces have been adapted to use an enhanced version of the Carthago architecture [1], which was originally developed to emulate the discrete Euler model of a CNN using templates of size 3x3, Dirichlet boundary conditions and the standard output function. PEs of this architecture has been designed to maximize the use of internal resources of the Virtex4 FPGAs, without requiring the use of external memory. Our architecture was then optimized in area and speed to be able to process real time standard video. Finally, the initial Carthago architecture has been enhanced to support space and time variant templates. The following items summarize the main features of the EPs used to test a specific case of system:

- One Euler iteration is carried out by a single PE that performs: 2 convolutions (one with each template, A and B), the addition with the bias and the evaluation of the standard output function.
- PEs support traditional cloned templates, but also the (simultaneous) use of time-variant and space-variant templates. Here, the templates are of size 3x3.
- Every PE propagates its result to the next PE, together with its input value, in order to perform inter-neuronal functions and to develop structures that use the CNN input data in successive iterations.
- The latency of the EPs depends on the template size, and its value is equal to **n*r+1** pixels, where **n** is the number of pixels per line of the image and **r** is the radius of the template. For templates of size 3x3, the latency of each EP will be a line and a pixel.
- PEs operate internally with 18 bit, full-precision, fixed point arithmetic. Inputs and outputs are truncated to 9 bits and templates use 18 bits coefficients.
- PEs have been designed with an area-optimized architecture that uses the fewest possible resources of the FPGA: 1 BlockRAM and 1 DSP48 block per convolution.
- PEs work internally at 410 MHz and require 17 clock cycles to process one pixel. This is equivalent to a rate of 24.1 megapixels/s and a total computing capacity of 850 MOP/s by PE (considering 18 MAC per pixel).
- PEs have been designed to process standard VGA video format: 640x480 @ 50-70Hz. However, it would possible to work with 1024x1024 images at a rate of 22 frames/s.

The high performance in area and speed of the PEs was obtained through a supersegmented sequential architecture based on two interdependent clock regions, which was optimized to maximize the use of resources embedded in the FPGA. In order to achieve maximum operating speed, PEs architecture was described in VHDL language using low-level hardware primitive instantiation techniques, and implemented through a customized placing and routing process with the use of RPMs (Relationally Placed Macros). Fig. 2 shows, in a simplified form, the internal schema and connections used by

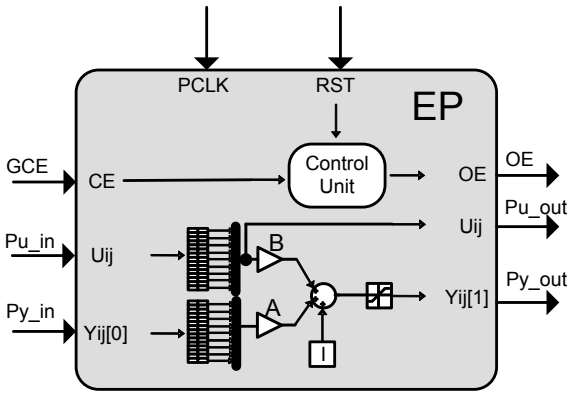


Figure 2. A simple Carthago IHM module connected to the expansión interface.

EPs Carthago architecture. More details and implementation-specific parameters can be reviewed in [4] and [5].

III. IHM INTERFACE

The expansion interface is designed to expand the system and provides the modules (either IHMs or frame grabber) with the video data and control signals necessary to perform a sequential processing. On the other hand, the modules connected to the interface, supply the processed video data and control signals to be used in further stages of the chain.

Fig. 3 shows the input/output ports defined for the expansion interface and a simple connection example. The IHM module used for this example has been configured with a simple internal layout consisting of a pipeline of several EPs connected in series. In general, the complexity of the internal structure of each IHM, and the number of EPs used, depend essentially on the type of processing required by the application and available resources in the FPGA, which are both independent of the interface.

In order for the system to support sequential computing, the expansion interface has been designed using a simple scheme based on pairs of ports: one input and one output, among which a direct connection can be established. This feature allows IHM modules to be connected and disconnected from the system quickly taking into account only two considerations: the first one is that IMHs must be plugged and unplugged when the system is powered off and, second, that pairs of ports on an unused connector must be bypassed so that information can reach the following modules. This feature allows over-sizing the number of connectors in the system and using only those which are necessary according to each application. Once the IHMs have been connected and the system powered, the FPGAs can be independently configured with the required structure of EPs.

The video signal provided by the interface is supplied to the modules through two input ports: the *Pu_in*, which spread the video stream to the chain from the main entrance of the system, and *Py_in*, which forwards the video data that, has been processed by a given stage, to the following. Within each IHM,

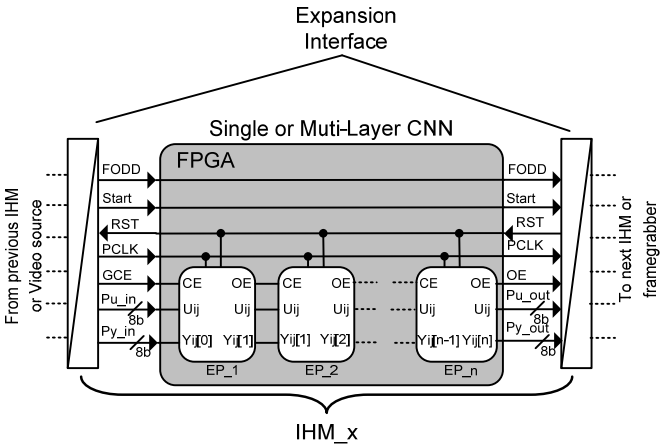


Figure 3. A simple Carthago IHM module connected to the expansión interface.

the two video streams are combined and processed by the EPs. The result is returned to the interface through the output port *Py_out* to be used in the next module. The output port, *Pu_out,* propagates the system's video input (*Pu_in*) to the next module, without any special processing but the necessary delay stages to keep the system synchronized.

Other ports used by the expansion interface are: *PCLK* (Pixel Clock), *RST* (Reset), *GCE* (General Cell Enable), *OE* (Output Enable), *Start* and *FODD*. The *CGE* port is used to enable the modules connected to the interface, while *OE* is an output port that is activated when the IHM provides valid results on its ports *Pu_out* and *Py_out*. To maintain consistency of information and synchronization processing, *OE* output must be connected to the *CGE* input of the next module. Signals *Start* and *FODD* are generated by the main video source and are propagated through the system chain to be used by the frame grabber. These signals are used to enable the frame grabber and to check whether the video stream is being generated in progressive or interlaced mode, respectively.

IV. System Architecture

Fig. 4 shows a detailed diagram of the system and the internal structure of the frame grabber. As shown, the digital video source is connected in the first position to the system chain, using the same expansion interface. The video source provides video data and control signals needed for synchronization. These are mainly *Vsyn*, used for enabling the frame grabber, and *Hsyn*, used to initiate processing by the first module in the chain (IHM_1).

While the video stream is propagated along the chain, with a latency determined by each IHM, the modules are activated and involved in processing. Since the latency of the modules produces a mismatch between video stream and frame synchronization, the frame grabber is used to correct the mismatch.

The frame grabber is connected to the system using the last expansion connector. The input that receives comes from the last module (IHM) and the output it provides is compatible with standard VGA. The frame grabber is designed with an architecture divided into two clock regions in order to simultaneously deal with video input timing (PCLK) and VGA output timing (VGA_CLK). For the frame grabber to work properly, the vertical synchronism frequency of the input video must be multiple of the VGA vertical synchronism frequency.

When the video processed by the chain reaches the frame grabber, information is stored in a static RAM with capacity for 2 frames. This memory has been divided into two blocks, what allows storing the incoming frame, while reading the previous one and sending it to the VGA output. When an incoming frame is written in one of the two blocks, the hardware automatically switches them, changing from write to read mode and vice versa. The writing and reading of both blocks is done by inserting 1 write cycle for every 3 read cycles. In each write cycle 4 pixels are stored in memory simultaneously. This is possible through the use of an input register that links the last 4 pixels received in a 32 bits word. The words read from memory, corresponding to the outgoing frame, are carried to a dual clock FIFO of 512 words to decouple the two clock regions. Since the number of reading cycles is at least 3 times the number of write cycles, the FIFO always has enough information. Each word read from the FIFO using the clock VGA_CLK is then stored in the output register and demultiplexed to separate the information into 4 independent pixels. To synchronize the pixel output, the architecture incorporates a standard VGA synchronization generator circuit. Finally, the video stream goes through the DAC and together with the synchronization signals is output by the VGA connector.

V. Implemetation

A particular case of the proposed system, comprising 2 IHMs, has been implemented on three FPGA prototyping boards to verify our design methodology and components functionality. This prototype consists of: a digital video camera (OV7620), two IHM modules, a frame-grabber and a VGA monitor. The camera is set to progressive mode, 640x480 gray scale images at 30 fps, and a digital output of 8 bits/pixel. The IHM modules have very small dimensions (7x8 cm) and have been implemented by standalone PICO E-12 boards [6], which include a Xilinx FPGA, model XC4VFX12, with 32 DSP48 and 36 BlockRAMs. Finally, the frame grabber has been implemented using the low cost SP3-EVL1500 board [7] comprising: an XC3S1500 Xilinx FPGA, 1 MB of external memory and a DAC from Analog Device, model ADV7125.

The system functionality has been evaluated using a 3-layer ML-CNN, with space-variant templates. The 3 layers are applied in sequence over the entire image. Taking a typical Euler iteration as the base unit, the 3 layers represent a computational cost of x1, x5, and x5, respectively. While the ML-CNN can be implemented in a single IHM comfortably, layers were divided between two IHM modules to verify the operation of the system when the circuit is distributed over different modules. Thus, IHM_1 implements the first 2 layers and IHM_2 the third one.

The selected algorithm performs a space-variant processing, which is a function of the quadrant of the image. To this end, PEs have been programmed with four sets of templates per

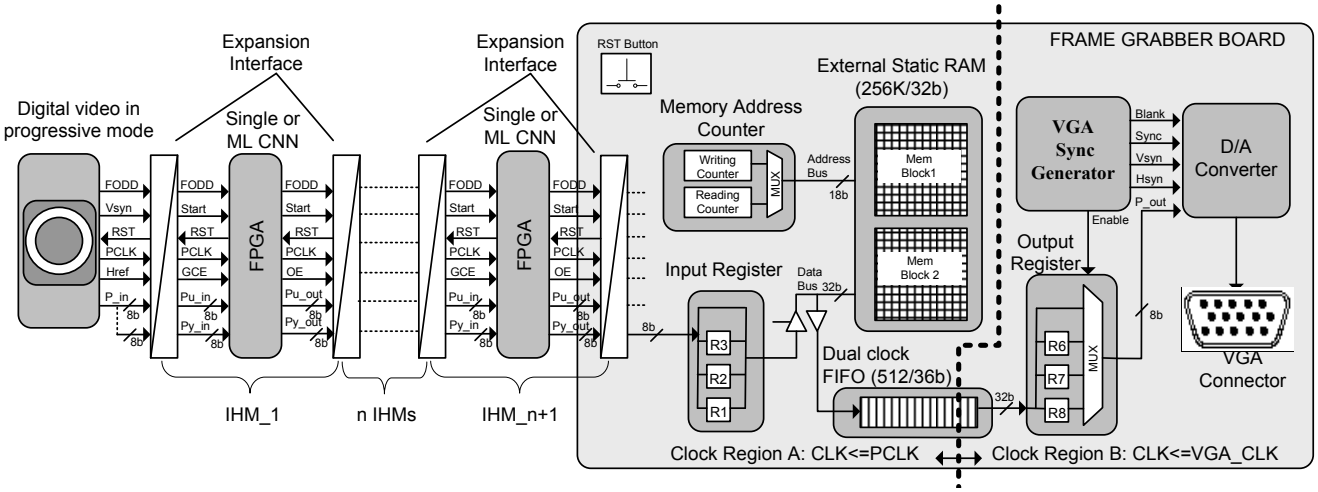


Figure 4. Complete system architecture and internal structure of the Frame Grabber module

layer, which will be applied depending on the position of the pixel being processed.The pixels of the upper-left quadrant are processed using the template layers: diffusion, edge detection and identity, in that order. The pixels of the upper-right quadrant used: inverting, edge detection and diffusion. For the lower-left quadrant, the three layers are the same: the identity template. Finally, for the lower-right quadrant, we selected also the same diffusion template for the 3 layers. Fig. 5 shows the implementation of the proposed system and the result after applying the ML-CNN to the scene captured by the camera.

The photograph in Fig. 5.c was taken with short exposure time to capture the falling of the ball. The VGA monitor shows a certain delay in the fall as a result of the delay that is inserted by the frame-grabber. The figure includes, in the lower-right corner, a picture of an IHM board.

The implemented prototype is limited in processing power by both IHM and the FPGAs used (XC4VFX12). The total computing capacity is 32 Euler iterations simultaneously, as the FPGAs only features 32 DSP48 blocks. The prototype can be easily extended by adding new expansion connectors and new IHM modules.

VI. Conclusion

In this work a versatile, multi-FPGA based system has been developed, allowing the emulation of large ML-CNNs. The system can be used in standalone applications to process real time standard video channels. A prototype, consisting of 2 IHM modules, has been implemented to emulate a 3-layer ML-CNN with space-variant templates.

The test of the prototype have demonstrated the system's correct operation and validated the methodology proposed in this paper.

Acknowledgment

This work has been partially supported by the Fundación Séneca de la Región de Murcia through the research projects 08801/PI/08 and 08788/PI/08, and by the Spanish Government through project TIN2008-06893-C03.

References

[1] Z. Voroshazi, A. Kiss, Z. Nagy, P. Szolgay, "Standalone FPGA Based Emulated-Digital CNN-UM System," in *Proc. Cellular neural networks and their applicaions*, 2008, pp. 4–4.

[2] Z. Nagy and P. Szolgay, "Configurable Multi-Layer CNN-UM Emulator on FPGA," *IEEE Trans. Circuits Syst. I*, vol. 50, pp. 774–778, 2003.

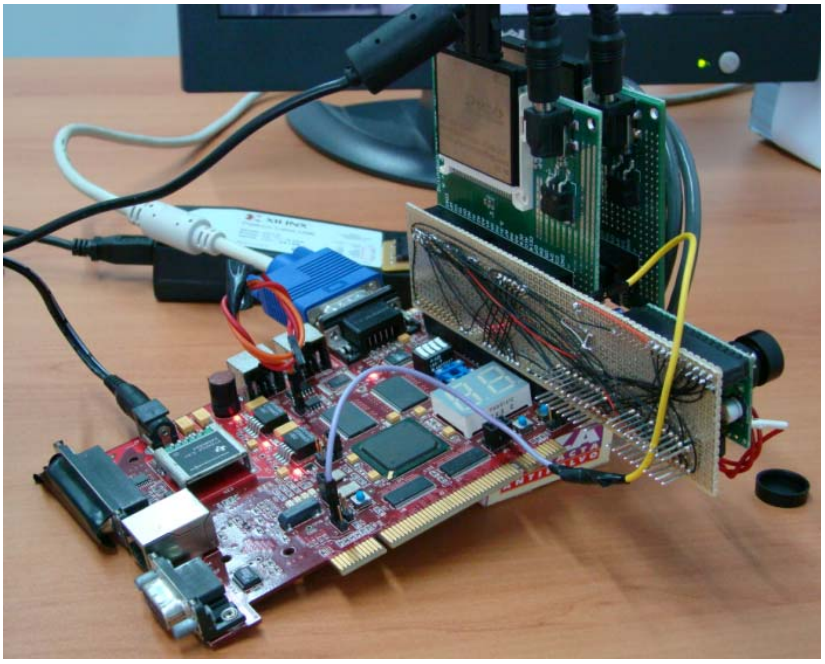
[3] K. Kayaer, V. Tavsanoglu, "A new approach to emulate CNN on FPGAs for real time video processing,", in *Proc. Cellular neural networks and their applicaions*, 2008, pp. 23-28

[4] J. Javier Martínez, F. Javier Garrigós, F. Javier Toledo, José Manuel Ferrández Vicente, "High performance implementation of an FPGAbased sequential DT-CNN,". IWINAC (2) 2007, pp. 1−9.

[5] J. Javier Martínez, F. Javier Toledo, E. Fernández, J. Manuel Ferrández "A retinomorphic architecture based on Discrete-Time Cellular Neural Networks using reconfigurable hardware", Neurocomputing, vol. 71, Elsevier, January 2008, pp. 766-775

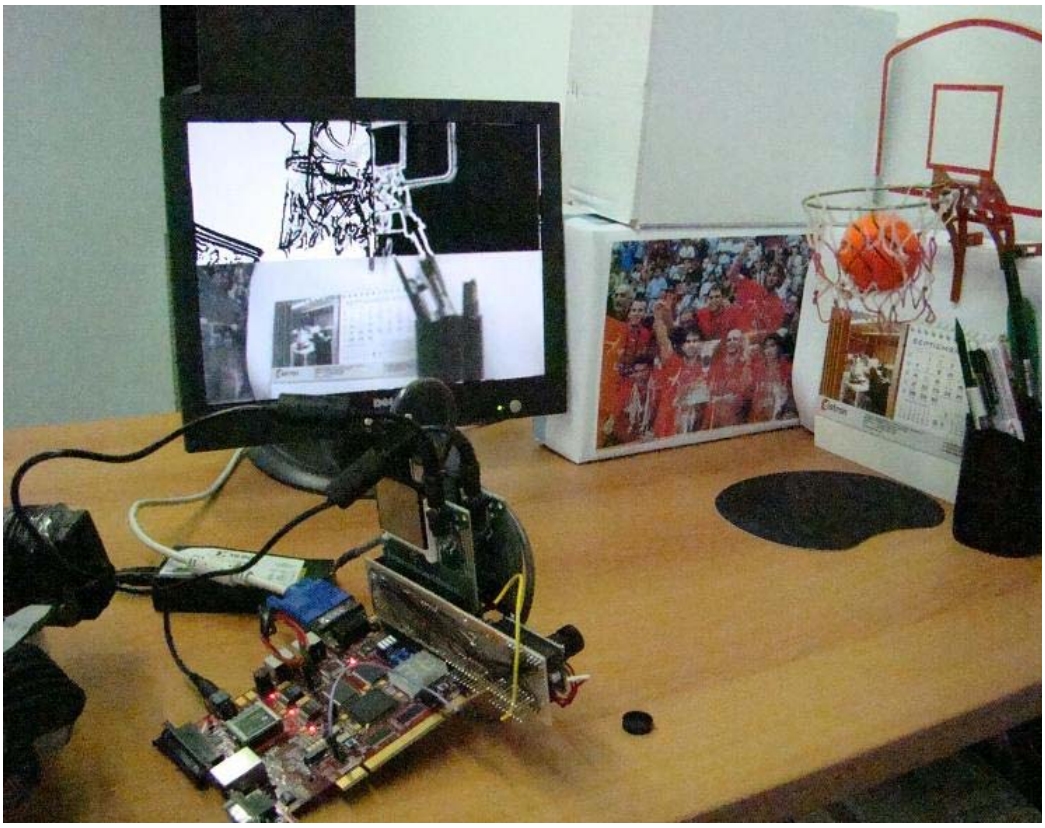
[6] "Card Products," [Online] http://www.picocomputing.com

[7] "Line Card," [Online] http://www.avnet.com/



(a)

(b)



(c)

Figure 5. Photographs of the system. a) Prototype consisting of the frame grabber board, 2 IHM modules and the digital video camera OV7620. b) Detail of a IHM module and the interface expansion connector. c) Photograph of the system and test bench