Hardware-Software Embedded Face Recognition System

M.J. Avedillo
IMSE-CNM
(CSIC/Univ. Sevilla)
c/Americo Vespucio
41092-Sevilla, Spain
avedillo@imse-cnm.csic.es

A. Barriga
IMSE-CNM
(CSIC/Univ. Sevilla)
c/Americo Vespucio
41092-Sevilla, Spain
barriga@imse-cnm.csic.es

L. Acasandrei
IMSE-CNM
(CSIC/Univ. Sevilla)
c/Americo Vespucio
41092-Sevilla, Spain
laurentiu@imse-cnm.csic.es

J.M. Calahorro
ETSII
Univ. Sevilla
a/ Reina Mercedes s/n
41012-Sevilla, Spain

ABSTRACT
This paper describes the design and implementation of a hardware-software embedded system for face recognition applications in images and/or videos. The system has hardware components to speed up the face detection and recognition stages. It is a system suitable for applications requiring real-time, due that the response times are deterministic and bounded. The system is based on a previous implementation that had accelerated the image capturing process, and the face detection. This paper will focuses in the face recognition acceleration.

Keywords
Hardware-software codesign, embedded system, face recognition, FPGA implementations, high level synthesis

1. INTRODUCTION
This communication presents the design of an embedded system to accelerate the recognition of faces in images and/or videos. A recognition system consists of four steps: 1. *Face detection* to detect if there is a face in the image (it provides the location and size of the face in the image); 2. *Face alignment* to locate the position of the face and, using geometric transformations, normalizes it with respect to geometric properties, such as size and pose, and photometric such as lighting; 3. *Feature extraction* to provide a feature vector with information to distinguish faces from different individuals according to geometric or photometric variations; 4. *Recognition step* in which the extracted feature vector is compared with the vectors in a database.

The system (Fig. 1) receives data from an image sensor (camera). Each frame is stored in internal memory and is processed by the system. The processing performed by the recognition algorithm requires two components: a software application and the hardware accelerators. The software application runs on a processor, and realizes the initialization and control of the hardware, as well as the recognition algorithms. The hardware accelerators can accelerate those tasks that constitute the "bottleneck" of the recognition algorithm. Thus, the proposed system (Fig. 1) is a hardware-software solution, which includes hardware accelerators to implement the most computationally expensive part of the face recognition algorithms: capturing images from the camera, processing for face detection and recognition algorithm acceleration. This paper focuses on the description of the Face Recognition Acceleration.

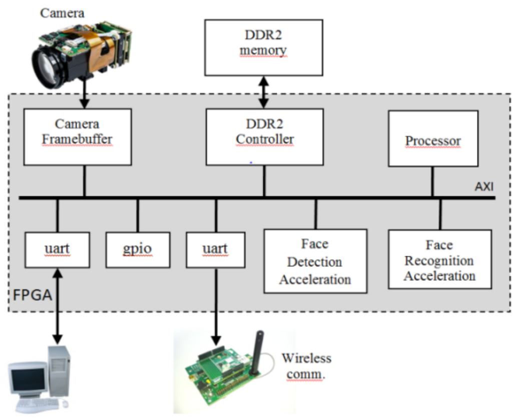

Figure 1. Block diagram of the embedded system

2. HARDWARE ACCELERATOR DESIGN
The face recognition algorithms implemented require matrix product operations which constitute the "bottleneck" of the system, limiting the operation speed. So it was decided to implement this part in hardware. The design of the hardware accelerator has been carried out using the high level synthesis tool from Xilinx Vivado-HLS [Xil14]. Vivado-HLS

generates an RTL description starting from a C/C++ algorithm level description. It realizes the scheduling and the resource allocation in order to map the algorithm to hardware. It also generates a description as IP module so that it can be used as a peripheral of a processor.

This methodology allows the designer to start the design of the system to be implemented in hardware from high-level descriptions. This means that algorithmic descriptions are made in a high-level programming language (in our case C++). Vivado-HLS takes as input the high level description and is able of generating a circuit that implements the desired algorithm. The designer can set restrictions, using directives (pragmas), on latency, throughput or hardware resources. As we will see, this methodology allows for exploration of the design space (due to automation), and design optimization through the application of directives.

Specification

The multiplication of the row vector containing the information of the input image by the principal component matrix (generated in the training stage and stored in the database) is computationally expensive. For this reason, it was decided to design a circuit to perform this operation.

The hardware block multiplies two matrices of fixed point numbers. The first matrix has one row and as many columns as the number of pixels of the image. That is, its size is *1*DIM*1. The dimension of the second matrix, *B*, is *DIM*1* *DIM*2, where *DIM*2is the number of principal components. The elements of the matrices use a 32 bits signed fixed-point representation, with 16 bits for the fractional part. This is given by the type of data used in the software application that it aims to accelerate.

The accelerator works in streaming mode, so it receives and sends data sequentially. There is only one port for receiving the two operand matrices. This is determined by the architecture requirements of the hardware platform, which has been designed to optimize data transfer between the processing system and the accelerator. The operating frequency of the system is 100 MHz.

Description

Figure 2 shows the description in C++ language. The ***accelerator*** function is the top function and it is the function to be synthesized using Vivado-HLS. In this top function, interfaces for the hardware module are configured and the function that encapsulates the functionality of the hardware block (***accelerator_core***) is called.

The pragmas with the directives HLS INTERFACE and HLS RESOURCE are responsible for managing the sending and receiving of data arrays through the AXI buses. The first two associate FIFO communications protocols to the input and output ports to meet the interface requirements of the block. The other three pragmas add the required adapters in order to connect the module to an AXI4Lite bus for control, and to an AXIStream for data transfer. Finally, the function ***accelerator_core*** is called and the multiplication is performed.

In function ***accelerator_core,*** the first pair of nested loops models the storing of the first array in internal memory (array *a*). Next, the system stores the first column of the second array (first principal component) in internal memory (array *b*) and, then, the multiplication *a* by *b* is carried out. That is, it calculates the first element of the resulting array and stores it in internal memory (array *out*). The procedure is repeated for each column of the second matrix. Finally, the loop labeled *converter* describes the sending of the elements of the resulting matrix.

```
void accelerator(AXI_VAL in_stream[num.pix + num.pix
* num.comp], AXI_VAL out_stream[num.comp]) {

#pragma HLS INTERFACE ap_fifo port=in_stream
#pragma HLS INTERFACE ap_fifo port=out_stream
#pragma HLS RESOURCE variable=in_stream core=AXIS
metadata="-bus_bundle INPUT_STREAM"
#pragma HLS RESOURCE variable=out_stream core=AXIS
metadata="-bus_bundle OUTPUT_STREAM"
#pragma HLS RESOURCE variable=return core=AXI4LiteS
metadata="-bus_bundle CONTROL_BUS"

accelerator_core<ap_fixed<32, 16>, num.pix,
num.comp, 4, 5, 5>(in_stream,out_stream);
return;
}

template<typename T, int DIM, int DIM2, int U, int
TI, int TD>
void accelerator_core(AXI_VAL in_stream[DIM + DIM *
DIM2],

AXI_VAL out_stream[DIM2]) {
#pragma HLS INTERFACE ap_fifo port=in_stream
#pragma HLS INTERFACE ap_fifo port=out_stream
T a[UNO][DIM];
T b[DIM][UNO];
T out[UNO][DIM2];
assert(sizeof(T) * 8 == 32);
for (int i = 0; i < UNO; i++)
  readA_int: for (int j = 0; j < DIM; j++) {
                int k = i * DIM + j;
                a[i][j] = read_stream<T, U,
                    TI,TD>(in_stream[k]); }

for (int i = 0; i < DIM2; i++) {
  read_B_int: for (int j = 0; j < DIM; j++) {
                 int k = j + DIM * (i + 1);
                 b[j][0] = read_stream<T,U,
                   TI, TD>(in_stream[k]); }
  T sum = 0;
  Multiplier: for (int id = 0; id<DIM; ++id){
              sum += a[0][id] * b[id][0]; }
  out[0][i] = sum; }

converter:for (int j = 0; j < DIM2; j++) {
out_stream[j] = write_stream<T, U, TI,
             TD>(out[0][j], j == (DIM2 - 1)); }
return; }
```

Figure 2. C++ algorithm description.

AXI_VAL is a structure which represents the sending of data through the AXI bus.

The initial description has been validated in Vivado-HLS environment by running various functional tests. Obtained results have been checked against those of the multiplication in the original software application.

Synthesis

Once it has been verified that the software description of the multiplication algorithm works properly, the next step is to obtain an RTL implementation. This requires synthesizing the C++ description.

Three different solutions have been generated. Each of them with different optimization directives applied. The accelerator is optimized using pipeline. Table I summarizes the obtained results. The results are for an accelerator module with *DIM*1=*320* and *DIM*2=160 on a Zynq-7020device.

	Latency (cycles)	BRAM	DSP	FF	LUT
Sol1	564644	3	4	294	316
Sol2	512966	3	4	306	365
Sol3	104646	3	4	271	305

Table 1. Synthesis results

Solution 1 corresponds to the synthesis of the C++ description without applying any directive. That is, this solution does not include any optimization. In Solution 2, pipeline directives are applied to all the loops, except the one in which the multiplication is performed (loop labeled *Multiplier*). Solution 3 is the most optimized one since it applies the optimizations of previous solution, but also applies pipeline to the multiplication loop. Precisely this is the loop causing the bottleneck in the matrix multiplication algorithm.

It is noted that the three solutions are similar in terms of used resources. No differences are observed, as expected, in the number of memory blocks and DSPs. The differences in terms of latency (in cycles) are significant. Solution 2 slightly improves Solution 1. For Solution 3, which apply pipeline to the calculation loop, latency is significantly lower than in Solutions 1 and 2.

Furthermore, concerning the use of hardware resources, it is observed that Solution 2 requires slightly more LUTs and flip-flops. This is due to the inclusion of pipeline stages which increases the number of required circuit elements. However, it is noted that Solution 3 requires less LUTs and flip-flops. The reason is that, although more pipeline is added, the synthesis tool has achieved a better optimization of the resulting hardware.

Figure 3 shows the latency, measured in clock cycles for each of the three solutions. The impact of the use of directives in the system performance is graphically observed.

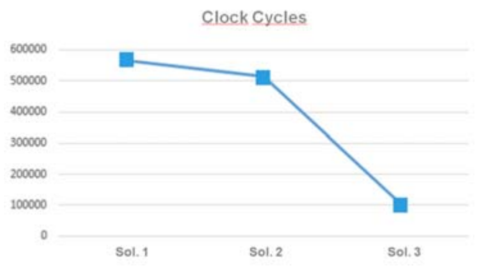

Figure 3. Response time of each implementation.

3. FACE RECOGNITION SYSTEM DESIGN

Once a Pcore containing the description of the hardware accelerator for facial recognition has been generated with Vivado-HLS, the next step is the design of a hardware platform incorporating it as peripheral and the development of a software application which makes use of it.

The system has been implemented in the Xilinx ZedBoard development board, which incorporates a Zynq-7000 SoC device. The hardware platform consists of five components: the ARM processor, the AXI buses, a timer, a DMA controller and the IP accelerator module. Figure 4 outlines the architecture of the system.

The AXI buses are responsible of the communication between the various components of the circuit. The ARM processor, in the PS in Figure 4, runs the software part of the face recognition system. It sends the data to the hardware accelerator. In order to configure the processor, it is needed to enable the S_AXI_ACP interface to communicate with the bus for sending and receiving data.

The timer is used to make performance analysis of the proposed system. Specifically, it is used to measure the number of clock cycles consumed by the accelerator module, in order to compare it with the number of cycles without using accelerator.

The DMA is responsible of the communication between the PS and the IP accelerator module. It provides the input matrices to the hardware accelerator and receives the resulting matrix from it. Also it must make this data available to the processor so that it continues its execution. The DMA operates in burst mode. To reduce the memory access time, the maximum number of data to be transmitted in each burst has been set to 256 Bytes.

The starting point for the design of the embedded system is the OpenCV's baseline face detection/recognition application [Ope10]. The OpenCV baseline face recognition application was modified in order to adapt it to a standalone embedded face recognition system. We have considered that the majority of embedded environments are capable of running C applications with or without operating system (OS) support. This means that the resulting application code has to be compatible to C compilers and, at the same time it should be platform independent.

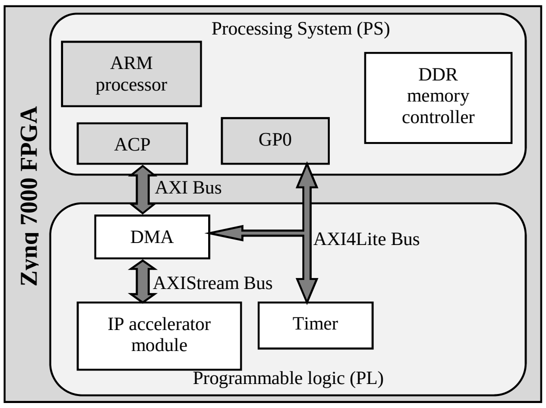

Figure 4. Hardware platform

Another consideration made is the fact that most of the SoC have no floating point support. Because of this, the developed application uses integer operations instead of floating point operations in order to preserve the generality of the application for the embedded system world. Additionally other modifications were made to accelerate the algorithms. Finally, the part of the software application related to the matrices multiplication, needed for projecting the image, has been replaced. This calculation is performed by the IP accelerator module, so it has been necessary to develop different functions to work directly with the system peripherals (DMA and the IP module).

Once the IP module has completed the execution and the DMA has transferred the result of the multiplication, the processor continues running the software application.

4. FACE RECOGNITION SYSTEM EVALUATION

To validate the developed system and to evaluate its performance, different tests have been carried out. Specifically, two types of tests have been made, one with an all-software recognition system, which does not include hardware accelerator, and the other with the system that includes the hardware accelerator. The three implementations of the hardware accelerator described in Section 2 have been tested to experimentally evaluate how optimizations affect the final result.

We have used two image databases containing images of different sizes in order to check the acceleration in relation to the size of the matrices to be multiplied. One database is the AT&T face databases [Att02] (resolution of 92×112 pixels). In AT&T database there are 10 grayscale images for each of the 40 individuals who compose it. The second database is Faces94 [Spa94] (resolution of 180×200 pixels). It contains 3040 color images of faces divided into 20 images for each of the 152 individuals who compose it.

Tests were carried out on four types of systems: the purely software embedded face recognition system and the three hardware platforms with accelerator, as mentioned previously. These three solutions are the hardware acceleration module without optimization (not including pipeline stages), the module that includes pipeline stages in all loops except the multiplication one, and the module including pipeline in all loops.

Figure 5 shows the results obtained for Faces94 database. We can see the impact produced when inserting pipeline on the accelerator, which makes this operation even faster. For the small images (ATT), the multiplication is about 11 times faster, and for the large images (Faces94) it is 17 times faster.

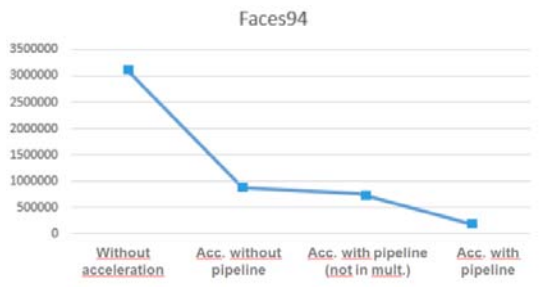

Figure 5. Acceleration measures (in clock cycles)

5. CONCLUSIONS

This communication described an embedded system that accelerates the face recognition process in images or videos. The implemented face identification system is suitable for applications requiring real time because the time response is deterministic.

6. ACKNOWLEDGMENTS

This work was supported in part by Ministerio de Economía y Competitividad under the Project TEC2014-57971-R, co-financed by FEDER.

7. REFERENCES

[Att02] AT&T Laboratories Cambridge, Database of Faces. [On-line] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

[Ope10] OpenCV Reference Manual. 2010.

[Spa94] Spacek, L. Collection of Facial Images: Faces94. [On-line] http://cswww.essex.ac.uk/mv/allfaces/faces94.html.

[Xil14] Xilinx. Vivado Design Suite User Guide High-Level Synthesis. UG902 (v2014.1) May 30, 2014.