A Framework for Design and Validation of Face Detection Systems

Nelson C. S. Campos*†, Heron A. Monteiro†, Alisson V. Brito‡,
Antonio M.N. Lima*†, Elmar U. K. Melcher† and Marcos R. A. Morais*†
*Graduate Program in Electrical Engineering - PPgEE
†Department of Electrical Engineering - DEE
Universidade Federal de Campina Grande - UFCG, Brazil
‡Center of Informatics
Universidade Federal da Paraíba - UFPB, Brazil

e-mail: nelson@ieee.org, heron@copin.ufcg.edu.br, alisson@ci.ufpb.br,
amnlima@dee.ufcg.edu.br, elmar@dsc.ufcg.edu.br, morais@dee.ufcg.edu.br

***Abstract*—A framework for hardware design and validation of face detection systems is discussed. The design begins at a high-level of abstraction by using a software library as the golden reference model. The golden reference model and the baseline model (model under validation) are converted to a transaction level (TLM) description or a register transfer level (RTL) description. The degree of similarity of the detected faces is used to determine whether the model under validation fulfills the prescribed design requirements. The framework is designed for real time high resolution image processing applications and is based on the universal verification methodology (UVM). As a case study, the framework was applied to validate a C++ baseline model written to implement the OpenCV library considered as the golden reference model. In this case both models have been converted to TLM and thus the time required for determining whether the baseline model is valid or not is significantly reduced. Thus one can decide as early as possible if one must proceed with the conversion of baseline model to RTL based on the result of the TLM validation test.**

***Keywords*—Hardware Design, Verification, UVM, SystemC, UVM Connect, OpenCV, Viola Jones, Rectangles Matching.**

I. Introduction

Face Detection is a complex problem in computer vision and is usually the first step towards Face Recognition. Face recognition can be applied in many fields like neuroscience, psychology, human computer interaction, video surveillance and face databases management [15] [16] [17]. Several techniques have been proposed for face detection in the past decades. Among them Neural Networks [8], Support Vector Machines [12], Hidden Markov Models [4] and Viola Jones, which is considered a state of the art method for real time applications [14].

The design flow of an image processing system to be embedded in digital cameras and smartphones, usually require the partitioning of the specification in two branches: hardware and software parts. The software branch is usually implemented in C/C++ code allowing to re-use legacy libraries. This C/C++ code yields a transaction-level description that can be integrated in a SystemC environment providing stimuli for testing the hardware branch. The hardware branch is implemented by using hardware description languages, usually SystemVerilog. The integration of the hardware and software branches is done by simulation. The final design is submitted to exhaustive tests before its physical fabrication to find bugs and thus avoid the manufacture of an incorrect system that would cost thousands to millions of dollars.

As mentioned before, this design flow is not error free and thus verification methodologies like UVM [3] and the library UVM Connect (UVMC) [7] are mandatory part of the design flow, consuming up to 80% of the project budget [11]. A problem that limits the use of UVMC in the design of image processing system is the maximum payload allowed by the UVM packer. If a transaction has a size larger than of the maximum payload it cannot be exchanged at once. In this case to exchange a large data package it would be necessary to split it into several parts having the size of allowed payload and then execute as many transactions as chunks are available [5]. As an example for exchanging a relatively modest image of 2MB it would necessary to execute 512 transactions since the maximum payload allowed for the UVM packer is 4KB. This is quite time consuming and becomes critical in the case of high resolution raw video since one second at 30 frames per seconds correspond to approximately 178 MB of data.

The main contributions of this work are: i) to discuss an alternative design flow that can be applied when the golden reference model is given in the form of a software system for face detection and ii) to mitigate the payload bottleneck due to UVM packer payload size limitation.

The paper is organized as follows: in section II we discuss about related works to integrate OpenCV with

This work was supported by CNPq, CAPES and PPgEE.

UVM Connect for fast image processing hardware design flow and testbench conceptions. In section III we endorse the background of the proposed methodology presented in this work. In section IV we discuss the results and in section V we present the conclusions.

II. Related Work

A review in related works integrating OpenCV with UVM and SystemC can be seen in [10] and [11], which they present an approach for reducing the effort of testbench implementations using SystemC and UVM. Because the reference models are conceived in SystemC and verified in UVM, UVMC was used to bridge the two SystemC/SystemVerilog sides. In these works the SystemC models call OpenCV functions to validate image and video algorithms. Their proposed design starts with the system specification, in which the application is described in C/C++ within an OpenCV environment. Once the specification step is done, SystemC is used to model the system by means of Transaction-level Modeling (TLM) [9]. The next step is to refine the abstract descriptions of the Electronic System Level (ESL) into a final structure that can be translated to a synthesizable hardware design at Register Transfer Level (RTL).

The related works discussed above gives relevant background to this paper but they do not address the limitation of the 4KB payload of UVMC that is extended with our proposed method. Furthermore, the goal of this present work is to provide a structure for quick design and validation of image processing algorithms with face detection as our case study.

III. Proposed Design Methodology

Image processing applications require large volume of data and are computationally intensive. The design of many systems should be driven with small size, weight and low power consumption constraints, which can only be achieved with a combination of hardware and software solutions, where the critical parts of the application is implemented in hardware to accelerate the system computation.

To address the size limit constraint of 4096 bytes imposed by the library UVM Connect, we present a way to expand the size of transactions. Rather than exchange the data content, that is limited to 4KB per packet, the proposed approach in this paper sends the memory address of the transaction, that is equivalent to a long unsigned integer with 64 bits. Thus, this approach leads to a significant decrease in simulation time compared with the method proposed in [5], which is based on fragmenting the data into chunks. This technique of applying the proposed method of exchanging transactions defined as memory address to yield a framework for quick design and validation of face detection systems. The output of these systems are rectangles coordinates of a likely face region. We evaluate the both systems using rectangles matching with two metrics: one proposed in [6], that is inversely proportional to the arithmetic mean area of the two rectangles and another metric proposed in this paper, that is inversely proportional to the geometric mean area of both rectangles.

There are different ways to design hardware for image processing applications. Similar to [11], we propose the design flow of Figure 1. The first step of the flow is the specification problem, that is the highest level of abstraction and describes the application. Later, a manual partitioning separates the project in two branches: hardware and software parts. In the first branch, a golden reference model written in OpenCV is mapped in TLM level, while the second branch refines a C++ baseline implementation based in [2] in TLM with a lower level of abstraction. The last stage of the design flow is the evaluation metrics of the both *reference model* and *model under validation* systems, which is done inside an UVM environment.

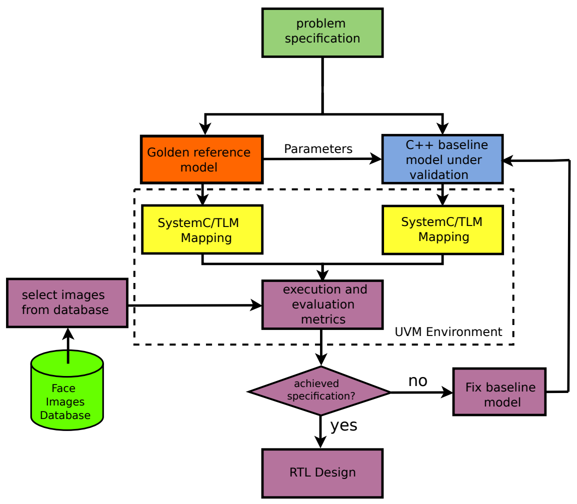

Fig. 1: The proposed design flow

A set of face images database feeds both reference model and model under validation and the execution and evaluation metrics block of the framework generates a report containing information about the number of the detected and missed faces, besides the false positive and false negatives rates. These results measures the quality of the model under validation, evaluating if the model satisfies the specification or not.

The design flow proposed in Figure 1 starts with a high-level of abstraction, using OpenCV for the specification stage. After the partitioning step, the two branches of the flow are modeled in transaction level, mapping the specification in TLM, where the communication between components do not take into account architectural details, which may be added later.

A. *TLM Mapping*

TLM Mapping is the first step within the UVM Environment. At this stage, the OpenCV application is integrated as a component within the environment, providing functions to read image files using *Direct Programming Interface* (DPI), that is an interface in which SystemVerilog calls functions from foreign languages like C or C++.

In Code 1 the *frame_sequence* class generates frame transactions, which contains the image data crossing the environment. To start the transaction, the external function *readframe* shown in Code 2 exports OpenCV functions implemented in C++ via DPI. This C++ function allocates a buffer with all the pixels inside the image.

Code 1: SystemVerilog class of *frame_sequence*

```
context function longint unsigned readframe(string filename);
class frame_seq extends uvm_sequence #(frame_tr);
  ‘uvm_object_utils(frame_seq)
  function new(string name=”frame_seq”);
    super.new(name);
  endfunction: new

  string filename = ”img.jpg”;

  task body;
    frame_tr tr = frame_tr::type_id::create(”tr”);
    start_item(tr);
    tr.a = readframe(filename);
    finish_item(tr);
  endtask:body
endclass: frame_seq
```

Code 2: An external OpenCV function to read a frame

```
extern ”C” unsigned long long readframe(const char* filename)
{
  Mat image = imread(filename, 1);
  return (unsigned long long)image.data;
}
```

To deal with the limitation of the maximum 4KB payload per transaction when using UVMC, traditional approaches are normally based on the transmission of image frames in several smaller chunks of data [5], as can be seen in Codes 3 and 4. This approach is functional but not efficient when large volume of data must be transmitted per transaction. In our case, the goal is to achieve functional verification of systems for processing high resolution images. Thus, the traditional approach results in long simulation time.

Code 3: SystemVerilog class of *frame_transaction* defined as 4KB block of data

```
‘define SIZE_CHUNK 1024
class a_tr extends uvm_sequence_item;
  int a[‘SIZE_CHUNK];

  ‘uvm_object_param_utils_begin(a_tr)
    ‘uvm_field_sarray_int(a, UVM_DEFAULT)
  ‘uvm_object_utils_end

  function new (string name = ”a_tr”);
    super.new(name);
  endfunction
endclass
```

Code 4: SystemC side of *frame_transaction* defined as 4KB block of data

```
#define SIZE_CHUNK 1024
struct a_tr {
  int a[SIZE_CHUNK];
};
UVMC_UTILS_1(a_tr, a)
```

Rather than exchanging the data content, that is limited to 4KB per packet, our proposed approach sends the memory address of the transaction, that is equivalent to a **long unsigned integer** with 64 bits. Thus, this approach leads to a significant decrease in simulation time compared with the method proposed in [5].

The outline of the transactions defined as memory address can be seen in Codes 5 and 6. This outline is the core of the UVM Environment of the proposed framework.

Code 5: SystemVerilog class of *frame_transaction* defined as memory address of the image

```
class a_tr extends uvm_sequence_item;
  longint unsigned a;

  ‘uvm_object_param_utils_begin(a_tr)
    ‘uvm_field_int(a, UVM_DEFAULT)
  ‘uvm_object_utils_end

  function new (string name = ”a_tr”);
    super.new(name);
  endfunction
endclass
```

Code 6: SystemC side of *frame_transaction* defined as memory address of the image

```
struct a_tr {
  unsigned long long a;
};
UVMC_UTILS_1(a_tr, a)
```

B. *Formal Method of the UVM Environment*

The general structure of the UVM Environment of Figure 1 is illustrated in Figure 2. The environment is composed by an *Active Agent* that stimulates the *Device Under Test*, a *Passive Agent* that collects data to be evaluated and a *Scoreboard*, which evaluates the stimuli coming from both agents.

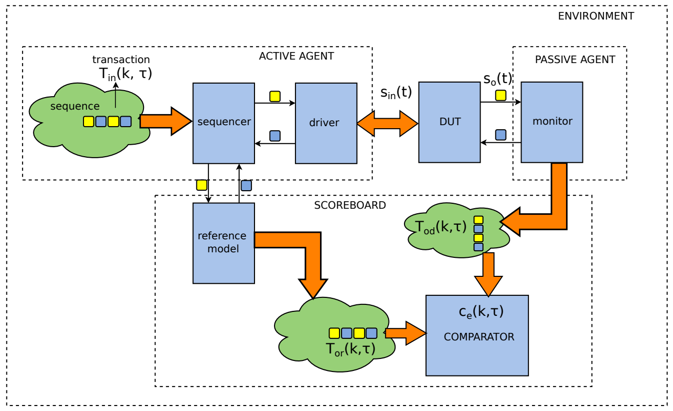

Fig. 2: General structure of the UVM Environment

For random cases, the *Active Agent* creates a sequence of transactions $T_{in}(k,\tau)$, with the form of Equation 1, where $T_{in}(k,\tau)$ is a random vector of integers $t_{in}(\cdot,\cdot) \sim \mathcal{U}(-2^{31}, 2^{31}-1)$, and $\mathcal{U}(-2^{31}, 2^{31}-1)$ denotes a discrete uniform probability distribution which stimulates the *reference model* and the *driver*. For natural images taken from a camera or a database, in the frequency ω, $T_{in}(k,\tau)$ has the power density spectrum $S(|\omega|) = \frac{c}{|\omega|^{2-\chi}}$, where $c > 0$ and $\chi < 1$ [13].

$$T_{in}(k,\tau) = \begin{bmatrix} t_{in}(0,\tau) & t_{in}(1,\tau) & \dots & t_{in}(N-1,\tau) \end{bmatrix} \quad (1)$$

The *driver* converts the transaction $T_{in}(k,\tau)$ into signals $s_{in}(t)$, according to Equation 2, where the function $D(\cdot)$ converts transactions into signals that feeds the device interface. The DUT produces the output signals $s_o(t) = DUT(s_{in}(t))$, that is collected by the *monitor*. The *monitor* defines the function $M(\cdot)$ that converts the signals $s_o(t)$ into output transactions $T_{od}(k,\tau)$, where $T_{od}(k,\tau)$ is defined in Equation 3. The operation of both functions $D(\cdot)$ and $M(\cdot)$ is depicted in Figure 3. Similarly, the output transactions of the referente model $T_{or}(k,\tau)$ has the form of Equation 4.

$$s_{in}(t) = D(T_{in}(k,\tau)) \tag{2}$$

$$T_{od}(k,\tau) = \left[\begin{array}{cccc} t_{od}(0,\tau) & t_{od}(1,\tau) & \dots & t_{od}(N-1,\tau) \end{array}\right] \tag{3}$$

$$T_{or}(k,\tau) = \left[\begin{array}{cccc} t_{or}(0,\tau) & t_{or}(1,\tau) & \dots & t_{or}(N-1,\tau) \end{array}\right] \tag{4}$$

$$T_{od}(k,\tau) = M(s_o(t)) \tag{5}$$

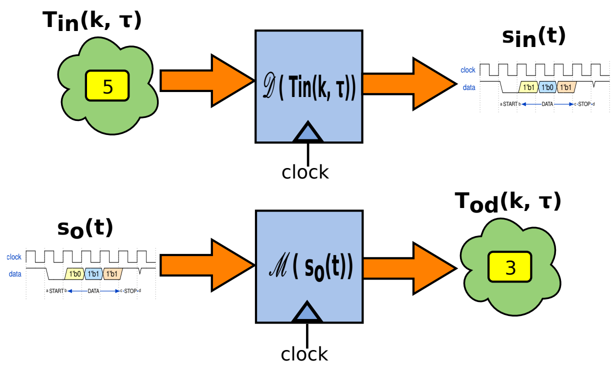

Fig. 3: Operation of the functions $D(\cdot)$ and $M(\cdot)$

The time τ of the $k-th$ transaction is related to the $t-th$ period of the clock according to equation 6, where N is the size of $T_{in}(k,\tau)$.

$$\begin{cases} k \equiv t (\bmod N) \\ \tau = \frac{t-k}{N} \end{cases} \tag{6}$$

Both transactions generated by the *monitor* and the output of the *reference model* should be compared by the *comparator* in which evaluates the comparison error $c_e(k,\tau)$. If the difference of both transactions are lower than a threshold ϵ, then they have a match, otherwise they mismatch as can be seen in equation 7.

$$c_e(k,\tau) = \begin{cases} 1, & |T_{or}(k,\tau) - T_{od}(k,\tau)| \leq \epsilon \\ 0, & otherwise \end{cases} \tag{7}$$

A pre-validation step is done when the RTL implementation is not available in the early stages of the design flow of complex projects. In such cases, the environment of Figure 2 is structured according to Figure 4. In this framework, the *driver* component is removed from the *Active Agent* and the *Passive Agent* is withdrawn, because there is no interface at this stage of the design flow. Then, the validation is at algorithmic and systemic level and the comparator evaluates transactions coming from the *reference model* and the *model under validation*, which is a previous TLM version that will be replaced by the RTL in a later stage.

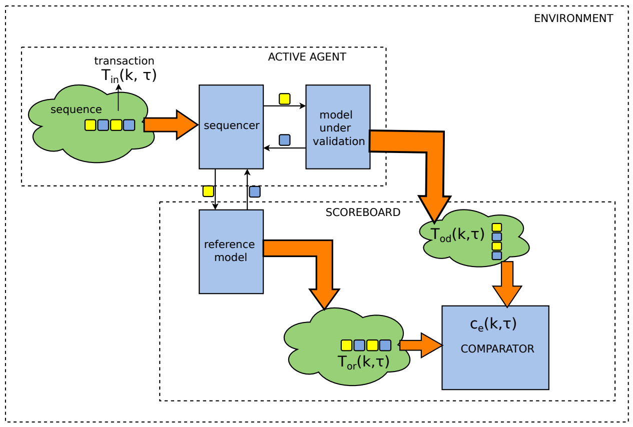

Fig. 4: Structure of a UVM Environment pre-validation step

C. Evaluation Metrics

In this subsection the *reference model* is defined as model C_1 and the *model under validation* as model C_2, both with the same input image. The output of these models are rectangle coordinates of a likely face region as shown in Figure 5.

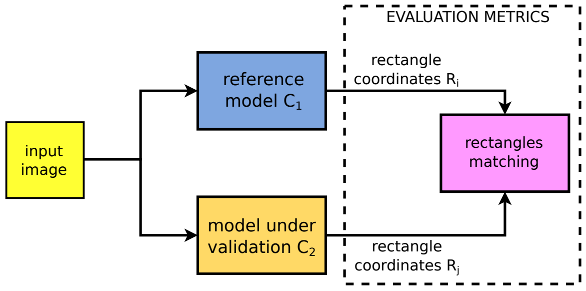
EVALUATION METRICS
input image
reference model C_1
model under validation C_2
rectangle coordinates R_i
rectangle coordinates R_j
rectangles matching

Fig. 5: Evaluation Metrics of the UVM *comparator*

The comparator of the UVM Environment performs an analysis on the output transactions coming from C_1 and C_2. To evaluate a match, the Equation 7 should be adapted to compute a similarity between rectangles. A metric which fits well to measure similarity between image targets is the dice coefficient [6].

The rectangles $R_i = (x_i, y_i, L_i)$ and $R_j = (x_j, y_j, L_j)$ have areas $S(R_i) = L_i^2$ and $S(R_j) = L_j^2$, respectively. The dice score coefficient between R_i and R_j is defined as $dsc(R_i, R_j) = \frac{2S(R_i \cap R_j)}{S(R_i)+S(R_j)} = \frac{2S(R_i \cap R_j)}{L_i^2+L_j^2}$, where $0 \leq dsc(R_i, R_j) \leq 1$. This metric gives the percentage of match between two rectangles and is computed by the comparator of the UVM Environment.

The intersection area between R_i and R_j is obtained applying Equation 8 into Equation 9. Note that the coefficient of dice is inversely proportional to the arithmetic mean area of the two rectangles R_i and R_j.

The comparator also evaluates a metric with a match normalized between 0 and 1 that is inversely proportional to the geometric mean area of the two rectangles R_i and R_j, according to Equation 10.

$$\begin{bmatrix} \delta & \zeta \\ \eta & \rho \end{bmatrix} = \begin{bmatrix} max(x_i, x_j) & min(x_i + L_i, x_j + L_j) \\ max(y_i, y_j) & min(y_i + L_i, y_j + L_j) \end{bmatrix} \tag{8}$$

$$S(R_i \cap R_j) = \begin{cases} (\zeta - \delta)(\rho - \eta), & \text{if } \zeta > \delta \text{ and } \rho > \eta \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

$$m(R_i, R_j) = \frac{S(R_i \cap R_j)}{\sqrt{S(R_i)S(R_j)}} = \frac{S(R_i \cap R_j)}{L_i L_j} \tag{10}$$

If the comparator is evaluating the transactions coming from C_1 and C_2 with the match percentage normalized by the geometric mean, then the comparison expression is given by Equation 11.

$$c_e(k, \tau) = \begin{cases} 1, & m(R_i(k, \tau), R_j(k, \tau)) > \epsilon \\ 0, & otherwise \end{cases} \tag{11}$$

As a design decision, we choose $\epsilon = 0.75$ for the threshold, similar to the tolerance established in the performance evaluation technique proposed in [6].

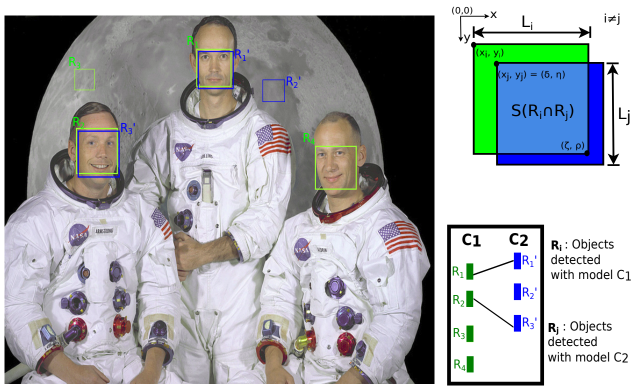

Fig. 6: Rectangles Matching (photograph from Nasa on The Commons)

To give an example, in Figure 6, the green squares R_i, where $i = 1, 2, 3, 4$ are the output vectors produced by C_1, while the blue squares R_j, for $j = 1', 2', 3'$ are produced by C_2. According to theses vectors, there were two matches, one between R_1 and R'_1, and another between R_2 and R'_3. Moreover, the model C_1 produced one false positive (detected a object that is not a face) in the region defined with the coordinates of R_3. The model C_2 had a false negative (missed a face in the region where C_1 detected R_4), consequently a mismatch occurred in the rectangle R_4 of the model C_1.

IV. Results and discussions

In subsection IV-A we will discuss the feasibility of applying the transaction model with data transmission by memory reference to the proposed framework, which has as ground-truth the Viola Jones algorithm of the OpenCV. In addition, in subsection IV-B we will analyze the operation of the validation environment by evaluating the performance of the Viola Jones algorithm with the implementation baseline of [2].

A. Extending the Transaction Payload in UVM Connect

To compare the execution time to send frame transactions from a source to a sink, two methods were proposed: the first method is is to slice the data according to [5]. The second method is our approach to define the transaction as a memory address to the image data. Table I shows the results of the simulation time to transfer images of 400KB, 4,000KB, 20,000KB and 40,000KB with the two methods.

Table I: **Method 1**: transmission by value. **Method 2**: transmission by reference.

Size of Image	Simulation Time		Speedup
	Method 1	Method 2	
400KB	1.134s	0.818s	1.39
4,000KB	4.070s	0.846s	4.81
20,000KB	16.891s	0.924s	18.28
40,000KB	33.150s	0.915s	36.23

The simulation time to send a transaction using Method 1 in seconds is computed in function of the number of chunks ($N_C = n^o$ of blocks of 4KB) according to equation 12, in which is obtained by linear interpolation of the points from Table I, where $t_{ref} \pm \alpha \approx 0.82s$ is the approximated time to send the transaction with the Method 2, α is a very slow number and $\lambda = \frac{0.00325s}{\text{block of 4KB}}$.

$$t(N_C) \approx t_{ref} \pm \alpha + \lambda N_C \tag{12}$$

It is important to say that this relation is based on the computer configuration used in the experiments. The following setup was used: Intel Pentium (R) CPU G630 with 3.6GiB of RAM Memory in a 64-bit CentOS 7.

B. Face Detection Systems Evaluation

To evaluate the performance of the *model under validation*, in which the baseline is the Viola Jones algorithm implementation of [2], we performed tests with the BIOID Database, composed by 1521 images 384x286x1 pixel resolution [1]. The Viola Jones algorithm of OpenCV, which is the *reference model*, presented 296 false positives and 40 falses negatives for the BioID Database, as can be seen in Table II. In contrast, for the same database, the *model under validation* committed 2 false positives and 417 false negatives. To evaluate the match criteria, the UVM comparator performed two metrics. The first metric is our proposed coefficient normalized by the geometric mean area of the rectangles, while the other is the dice coefficient. To compute a match, the percentage of both metrics should be at least 75%. According to Table II, using the geometric mean coefficient the comparator performed 38 matches with 83.87% of similarity and 1064 matches with 93.74%. Using the dice score theses results were very close. The number of mismatches using the geometric mean coefficient was 419 with 4.93% of similarity. Note that the total number of false negatives in both *reference model* and *model under validation*. However, using the dice score, the comparator performed 2 additional mismatches with percentage of 74%, since the established threshold is 75%.

Table II: Face Detection Rates for Database BIOID [1] (1521 images 384x286x1 pixel resolution)

Reference Model		Model Under Validation	
False Positives		**False Positives**	
frequency	n^o of falses	frequency	n^o of falses
1225	0	1519	0
269	1	2	1
25	2	–	–
2	3	–	–
False Negatives		**False Negatives**	
frequency	n^o of falses	frequency	n^o of falses
1481	0	1104	0
40	1	417	1
Percentage of Match			
geometric mean coefficient		**dice score coefficient**	
n^o of matches	percentage	n^o of matches	percentage
38	83.87%	42	83.87%
1064	93.74 %	1058	93.73%
n^o of mismatches	percentage	n^o of mismatches	percentage
–	–	2	74.00%
419	4.93%	419	4.93%

An investigation was made in the *model under validation* to analyze the discrepancy of its results with the *reference model*. It was noticed that the scale factor of the Viola Jones algorithm [14] of the *model under validation* was set at 20% and the *reference model* had set the scale factor to 10%. This issue has been solved and the *model under validation* increased the number of false positives to 5 and reduced the number of false negatives to 304 as can be seen in Table III. After the scale adjustment, using the geometric mean coefficient the comparator performed 36 matches with 83.06% of similarity and 1174 matches with 92.84%. Using the dice score theses results were very close. The number of mismatches using the two metrics was 306 with 4.89% of similarity. Thus, the framework showed its functionality to perform an automatic evaluation in both models.

Table III: Face Detection Rates for Database BIOID [1] (1521 images 384x286x1 pixel resolution) after adjusting the scale factor

Reference Model		Model Under Validation	
False Positives		**False Positives**	
frequency	n^o of falses	frequency	n^o of falses
1225	0	1516	0
269	1	5	1
25	2	–	–
2	3	–	–
False Negatives		**False Negatives**	
frequency	n^o of falses	frequency	n^o of falses
1481	0	1217	0
40	1	304	1
Percentage of Match			
geometric mean coefficient		**dice score coefficient**	
n^o of matches	percentage	n^o of matches	percentage
36	83.06%	43	83.06%
1179	92.84%	1172	92.84%
n^o of mismatches	percentage	n^o of mismatches	percentage
306	4.89%	306	4.89%

V. Conclusions

In this paper, we presented a framework for quick hardware design and validation of face detection systems. The framework is suited for image processing applications and is written using UVM Connect (UVMC). Since UVMC has the limitation of 4KB per transaction, traditional techniques are based on fragmenting the image into small pieces. However, this increases the simulation execution time, that is dependent on the data size to be transmitted. Our proposed approach only transmits the data by reference to a memory address instead. The proposed approach achieved an almost constant time for simulating large frame transactions and is independent on the size of the image. It significantly increases the performance of the simulation when a large amount of data is being transmitted. Case studies show, that the proposed approach achieved a speedup of 33.15 for the transmission of an image of 40,000KB compared to the prior approach presented in literature. Furthermore, using our proposed validation system we defined the comparator using region matching metrics to evaluate the Viola Jones implementation in SystemC/TLM of the *model under validation* versus the *golden reference model* written using OpenCV libraries reducing significantly the required time to find if the baseline model is valid or not.

References

[1] Bioid face database - facedb. https://www.bioid.com/About/BioID-Face-Database.

[2] F. comaschi. https://sites.google.com/site/5kk73gpu2012/assignment/viola-jones-face-detection.

[3] Accellera. Universal verification methodology. user's guide, 2011.

[4] Peter M Corcoran and Claudia Iancu. *Automatic Face Recognition System for Hidden Markov Model Techniques*. INTECH Open Access Publisher, 2011.

[5] Mentor Graphics Corporation. Fast packer converters. https://verificationacademy.com/verification-methodology-reference/uvmc-2.3/docs/html/files/examples/xlerate-connections/README-txt.html, 2016.

[6] David Doermann and David Mihalcik. Tools and techniques for video performance evaluation. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 4, pages 167–170. IEEE, 2000.

[7] Adam Erickson. Introducing uvm connect. *Mentor Graphics Verif. Horiz*, 8(1):6–12, 2012.

[8] R Fefaud, OJ Bernier, JE Viallet, and M Collobert. A fast and accurate face detection based on neural network. *IEEE Transactions on PAMI*, 23(1):42–53.

[9] Frank Ghenassia et al. *Transaction-level modeling with SystemC*. Springer, 2005.

[10] Michael Mefenza, Franck Yonga, and Christophe Bobda. Automatic uvm environment generation for assertion-based and functional verification of systemc designs. In *Microprocessor Test and Verification Workshop (MTV), 2014 15th International*, pages 16–21. IEEE, 2014.

[11] Michael Mefenza, Franck Yonga, Luca B Saldanha, Christophe Bobda, and Senem Velipassalar. A framework for rapid prototyping of embedded vision applications. In *Design and Architectures for Signal and Image Processing (DASIP), 2014 Conference on*, pages 1–8. IEEE, 2014.

[12] Edgar Osuna, Robert Freund, and Federico Girosit. Training support vector machines: an application to face detection. In *Computer vision and pattern recognition, 1997. Proceedings., 1997 IEEE computer society conference on*, pages 130–136. IEEE, 1997.

[13] Daniel L Ruderman. The statistics of natural images. *Network: computation in neural systems*, 5(4):517–548, 1994.

[14] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[15] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.

[16] Harry Wechsler, Jonathon P Phillips, Vicki Bruce, Francoise Fogelman Soulie, and Thomas S Huang. *Face recognition: From theory to applications*, volume 163. Springer Science & Business Media, 2012.

[17] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458, 2003.