An FPGA Based Fast Face Detector

Mohammad S. Sadri, Nasim Shams, Masih Rahmaty, Iraj Hosseini,
Reihane Changiz, Shahed Mortazavian, Shima Kheradmand, Roozbeh Jafari
FPGA Lab, Department of Computer and Electrical Engineering,
Isfahan University of Technology, Isfahan, Iran, Tel: +98-311-391-5428,
3514 Boelter Hall, UCLA, Los Angeles, CA 90095, Tel: 310-267-5243
sadri@sarv-net.com, {nasim, masih, iraj, reihane, shahed, shima}@mamsadegh.com, rjafari@cs.ucla.edu

Abstract
A face detection algorithm based on maximal rejection classification and skin color model is proposed. It is then computationally optimized by using an edge detector and a neural network.
The obtained architecture is then implemented on Xilinx Virtex-II Pro FPGA. At first we divide the algorithm into two parts: The first part is implemented using FPGA's internal logic such as slices and flip-flops. This makes a highly parallel structure for performing regular computations in a very small period of time. For the second part of the algorithm which requires irregular computations, internal PowerPC embedded processor of the FPGA is used.
Additional optimizations in the face detection algorithm itself, together with optimum design partitioning, and sophisticated assignment of tasks to each part of FPGA, create a really high performance face detector. The algorithm may also be modified for detection of other non-face objects.

Overall System Architecture
Our face detection system consists of four important parts [9]:

1- A skin color filter, which omits the parts of an image except those with skin color.
2- An edge detector that transforms each pixel of the image to a 0 or 1 value.
3- Rotation and sub frame generation: takes the main frame and generates different sub frames with different sizes and degrees of rotation.
4- MLP: detects if a sub frame contains any faces.
5- A decision system to merge and analyze the results of MLP.

Figure 1 shows different parts of the system and their connection. Arbitration and Decision will be implemented using Virtex-II Pro's embedded PowerPC microprocessor; all of the remaining parts will use normal FPGA's resources.
Input images first are transmitted to a buffer, which is mainly a fast memory outside FPGA. They go through three steps; Illumination correction tries to balance the illumination for different parts of the image. A lookup table is used for omitting portions of the image with no skin color, the result finally goes, through an edge detector, and then, it is stored in buffer. (In practice however, skin color filtering and edge detection are done simultaneously.)
Sub frame generation system then, generates lots of sub frames with different sizes from the edge detected image. Each sub frame is then, rotated and scaled to the appropriate size for the neural network. Neural network generates a real value to indicate if this sub frame contains a face image. This value is stored and used for the final arbitration.
Using FPGA resources such as slices, block memories and hardware multipliers we build different parts of our system. A fast wide memory interface provides low latency fast access to a memory outside FPGA. Hardware multipliers are used for required computations in the neural network and finally block memories are used either as cache memories or look up tables. The rest of the circuit is built using normal FPGA flip flops, gates and look up tables.
The entire system, works as a pipeline. Some units in this pipeline are more complicated. They need a larger time period to finish their tasks. This makes a bottleneck for the performance of entire system. Considering this, neural network and sub frame generation seems to be the most important parts of the system; therefore we focus on the design of these two.

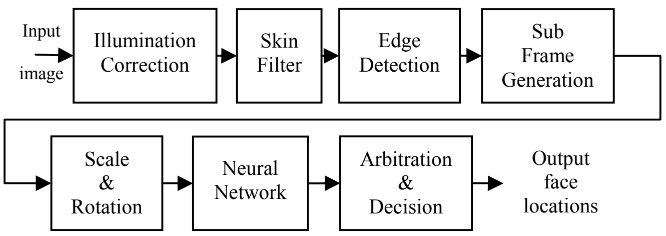

Figure 1. Overall system architecture.

Illumination Correction
Since illumination variation influences color characteristics of the image (i.e. color of the object changes if the illuminant color changes),which in turn influences the skin color detection and subsequently face detection, some algorithms are applied to the image in order to achieve adaptation to different lighting and illumination.
There are two main groups of algorithms for reducing the

effect of illumination condition on the image [3]: color constancy methods (i.e. color by correlation) and color compensation methods. Color by correlation method seems to be more robust [4] [5], however the second group is easier to implement [5].

Gray World algorithm is used for illumination correction in our design. This algorithm is based on the presumption that the average color, which is reflected from the surfaces, corresponds to the color of the illumination [3]. We do not describe the hardware design details for this part of the system in this paper.

Skin Color Filter

A lookup table is used to eliminate regions in the image that have no faces. Another technique is to use distinct ranges on the R, G and B or H, S and V value of color map and omit each pixel with values outside this range. Employing look up tables however yields more precise results [6].

Lookup table is a one-dimension array that uses one bit for each color of the color map to show if it is a skin color or not.

Using both RGB and HSV lookup tables in parallel helps us to select regions containing faces more reliably and it can reduce the effect of illumination as well [7].

Using 8 bit values for each of R, G and B, makes a look up table of 16Mbits, which should be implemented using off chip memory.

Edge Detection

Input color image should be converted into a gray scale image. Gray scale image then goes through a simple edge detector.

For each pixel of gray scale image, the gray value is equal to $\sqrt[3]{R.G.B}$. This computation is intensive however it is possible to use an approximation for the above formula.

df/dx and df/dy values for each pixel of the gray scale image are computed by using a sobel filter. A simple module can be used for computing the deference values. The gradient value is then computed. Gradient computation needs hardware multipliers. Putting some thresholds on the obtained image results an output with each pixel to be a one bit 0 to 1 value.

Sub Frame Generation

To detect the portions of the image which contain faces, a window will sweep the entire image. Each segment is scaled and analyzed. Different windows with different sizes should be used. Sub frame generation process is as follows:

1- The initial window with size $x_{SubFrame}$ is selected.
2- Begin from the top left most pixel of the image.
3- A complete window of $x_{SubFrame} \times x_{SubFrame}$ pixels is read from memory and goes to scaling module.
4- The window moves by J pixels. If reached the end of a row, it jumps to beginning of the next row. Vertical and horizontal jump values are both J pixels.
5- When reached the end of picture, the next window size is chosen, and we go to 2.

The produced sub frames are then scaled by a factor of S^2 to make a smaller picture. For each degree of rotation, a distinct area of this picture is sent to the neural network.

The total number of sub frames can be computed as follows:

Total number of sub frames =

$$= \sum_{i=1}^{N_{WS}} \frac{\left(x_T - x_{SubFrame}(i)\right)\left(y_T - x_{SubFrame}(i)\right)}{J(i)^2} \qquad (1)$$

In the above formula N_{WS} is the total number of different sizes for square windows. x_T is the input image width and y_T is height. J is equal to jump value for each window size.

FPGA Based Scaling and Rotation

Input to neural network should contain a fixed number of pixels. All of the produced sub frames with different sizes should be scaled to a proper size. For each sub frame, we send total number of R borders to MLP. Each border contains xy elements and indicates a portion of the main edge detected image. The width and the height of this portion is x and y respectively. Figure 2 shows this.

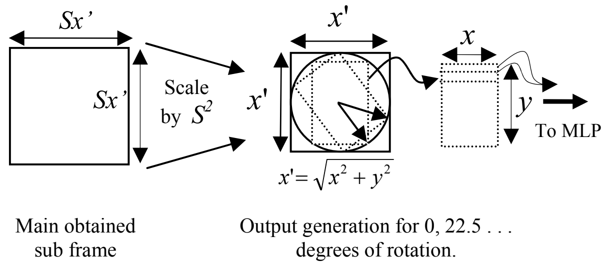

Figure 2. Scaling and rotation operations

For each of window sizes, a complete sub frame should first be read from the main memory outside FPGA. Scaling is performed for each sub frame. For the second, third and the rest of sub frames, however, we do not read the entire sub frame from outside memory. We only read the difference.

Each complete sub frame divides into smaller parts with width and height of J and Sx' respectively. In each memory access, we only read one of these slices. This slice is transferred to a dual port block memory inside FPGA. Then, it's pixels are sent to the scaling system. This leads to a very low number of clock cycles for doing memory access and scaling operations.

For z scale factor of S, we should choose one pixel O from each group of S^2 pixels. Each pixel is represented by one bit. e.g $S = \sqrt{7}$ means that from a group of 7 pixels, we should choose one pixel, and transfer this to the scaled image. The value for O is computed using the following method:

$$O = \begin{cases} 1 & \textit{Number of } 1s \textit{ in the group} > K_1 \\ 0 & \textit{Otherwise} \end{cases} \qquad (2)$$

This method for scaling operation is relatively accurate. In practice an input slice of a sub frame is segmented so that each segment contains $S^2 \times S^2$ pixels, and each segment is a square area. From each group of S^4 pixels, S^2 of them are chosen as the output. Figure 3 shows this.

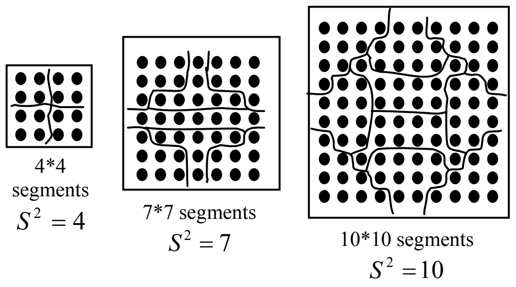

Figure 3. Grouping for scale operation.

Each slice of a sub frame consists of $J \times Sx'$ pixels and JSx'/S^4 segments. We choose J to be always an integer multiplicand of S ($J = kS, k = 1 \text{ or } 2 \text{ or } 3$). For each value of S, scaling module reads a square of $S^2 \times S^2$ pixels from block memory into an array of flip flops. Afterwards, it counts the number of ones for each of the areas shown in figure 3. The output of each area is one if the count is greater than K_1. As illustrated in figure 4, scaling module contains an array of $(S^2)_{\max}$ registers.

Memory read operations is performed as twice fast as the normal clock frequency of the chip. This is feasible because block memories are faster than other modules. All operations in scaling module can be done in one clock cycle; therefore, there is no need to increase the clock frequency of the scaling module itself.

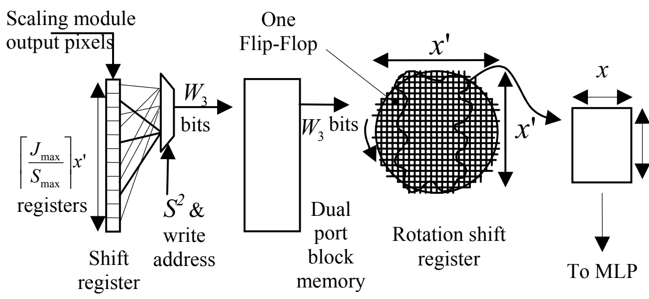

Figure 5. Rotation and grouping operations.

Each output of scaling module contains S^2 pixels and final output for each input slice is a $\lceil J/S \rceil x'$ pixels slice of the $x' \times x'$ square. Output pixels from scaling module should be placed in proper locations in the final scaled slice. We again use the same technique as shown in figure 3 in order to place pixels in proper locations. Figure 5 illustrates how this is done.

We choose $W_1 = J_{\max}$. In the worst case, the total number of clock cycles required to read one complete slice from the outside memory is $2Sx'$.

We also $W_2 = (S^2)_{\max}$ bits, hence, total number of clock cycles for reading one complete slice from block memory (and so scaling it) is:

$$\frac{J}{S^2}\left\lceil \frac{Sx'}{S^2} \right\rceil \times S^2 = J\left\lceil \frac{x'}{S} \right\rceil \qquad (3)$$

In which J/S^2 is the number of segments in each row and Sx'/S^2 is the total number of segment rows. If the neural network module is capable of accepting x pixels in each clock cycle, then total numbers of $R.y$ clock cycles are needed for each complete sub frame. So these inequalities should be met:

$$\frac{J}{2}\left\lceil \frac{x'}{S} \right\rceil + \frac{J}{2S} < R.y \qquad (4)$$

$$2Sx' < R.y \qquad (5)$$

Inequalities (4) and (5) state that scaling module should work faster than the neural network. The second term in (4) indicates the latency of grouping and writing scaled outputs to the second block memory.

After executing the scaling operations, the output is represented with a matrix of size $x' \times x'$. A special set of pixels must be chosen from this matrix and sent to the MLP module. Input to MLP is mainly a $x.y$ pixels segment of scaled image. Figure 2 shows how we obtain this segment for each degree of rotation.

Rotation module has a very simple architecture [8]. It has a total number of $\pi(x'/2)^2$ flip-flops. The architecture of rotation module is not complex however it utilizes a large portion of FPGA flip-flops as well as routing resources. Each register in the rotation chain requires a multiplexer which enables to perform either load or shift operations. Registers are connected to each other, so that in each clock cycle, the image in the circle shown in Figure 5 is rotated by r degrees. Rotation with various degrees can be performed by continuing the rotation operation for multiple clock cycles.

An FPGA Based MLP

A simple MLP [13] is used to detect if an area in the picture contains a face. The following is the problem description:

Suppose that we need an MLP to be implemented on an FPGA, with the following conditions and limitations:

- Topology of the MLP: one input layer with N_1 nodes, one hidden layer with N_2 nodes, and one output layer. Nonlinear functions are used for the input and hidden layer, and linear function for output.

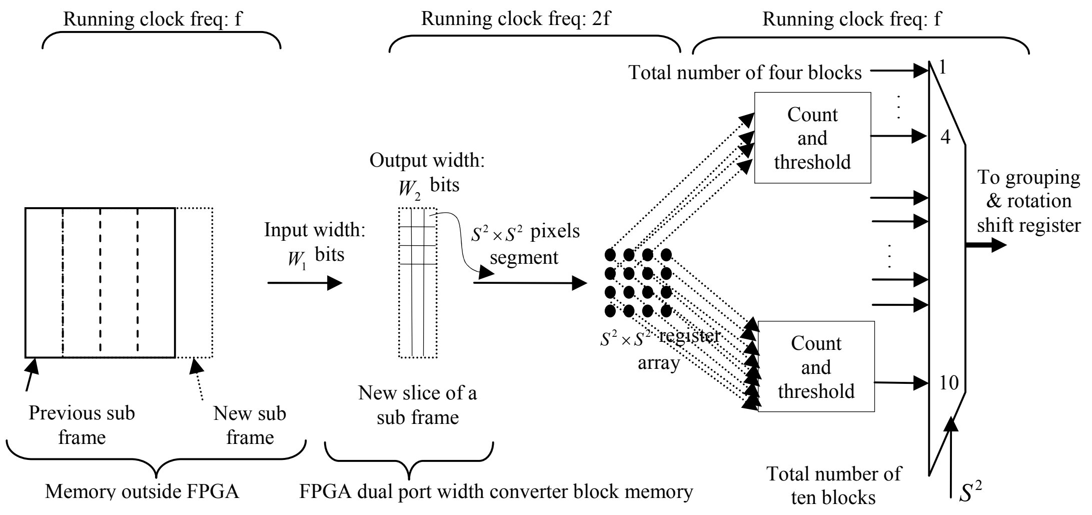

Figure 4. Sub frame generation and scaling in detail

- MLP must produce 1 for faces and -1 for non face inputs.
- For *i*-th scale value, input frame is $x(i)$ pixels in width and $y(i)$ pixels in height.
- To compute the final output of the MLP for each input, K pixels are read in each FPGA clock cycle.
- MLP weights: q bits signed, fractional format.
- Weights are stored in FPGA block memories.
- Clock frequency for weight memories: f_M and for the rest of the circuit : f
- Each block of memory provides up to W bits of data in each clock cycle.
- For each input node of the MLP, M blocks of memories are used for weight storage.

The input layer has the largest logic and is the main bottle neck for the speed of our system, so we mainly consider the design of this layer for optimization. Each node in the first layer has $x.y$ inputs, and there is a weight for each input, however only K pixels are read each clock cycle. Therefore we have the total number of $N_1.K$ weights in each clock cycle which is equal to $K.q$ bits of data for each node.

Equation (6) represents the required memory width and total number of block memories for each node in input layer:

$$K.q = \frac{f_M}{f} W.M \tag{6}$$

Equation (6) demonstrates how decreasing the running frequency of memory modules increases memory width and the total number of memory blocks. In practice memory width is limited to 72 bits for Virtex-IIP devices. Thus, for large values of $K.q$ we need to increase f_M or M however, the total number of available block memories in each FPGA is limited and they are one of the most valuable logic resources. In Virtex-IIP, f_M can be increased up to 400MHz. Equation (7) shows the number of clock cycles required to process each input.

$$\textit{Clock cycles required for one input} = \frac{x(i)y(i)}{K} \tag{7}$$

Our MLP works for two different dimension sizes: 18*22 pixels and 36*44 pixels. All obtained frames are scaled to one of the two dimensions stated.

(1) Shows total number of sub frames for each input. If we have R different degrees of rotation exists for each sub frame, the total number of FPGA clock cycles, needed for processing one complete image by MLP is:

Cycles per frame

$$= R\sum_{i=1}^{N_{WS}} \frac{(x_T - x_{SubFrame}(i))(y_T - y_{SubFrame}(i))}{J(i)^2} \frac{x(i)y(i)}{K} \tag{8}$$

Since the neural network is the slowest module in the face detection pipeline, it determines the speed of the entire system:

Frames per second =

$$\frac{f}{\textit{Cycles per frame}} = \frac{f_M W M}{K.q.(\textit{Cycles per frame})} \tag{9}$$

Top most important limitations of implementing an MLP on FPGA are:

1- Large number of simultaneous multiplication and addition operations.
2- Large number of weight values which yields large number of block memories working in parallel for providing needed data bits. Total number of $N_1 Kq$ bits of data should be read from memory simultaneously.

As described in the previous sections, the inputs to the MLP are only zero and one values. Thus, the multiplication does not have to be performed in the input layer of the MLP. This saves greatly on the consumption of FPGA hardware resources. The input layer contains only simple addition circuitry.

In each N_1 nodes, K additions should be done in each clock cycle. K input bits to MLP indicate which one of the loaded K weights should participate in addition. A pipeline is implemented to maximize the speedup of the system.

As we mentioned, weight memory frequency, f_M is different from f, which is the clock frequency of the rest of the neural network. There should be a relation between f and f_M. Usually $f_M = k.f, k = 2,3$. This means that, for each normal clock cycle, two or three weight values are read from the memory. So, the real limitation on the execution frequency of the entire face detection system is imposed by the frequency of block memories and in fact, not the normal FPGA logic.

Performing addition on $x(i).y(i)$ input numbers, each with q bits of data, results a $\lceil q + \log_2 x(i)y(i) \rceil$ bits value. But the nonlinear function, used for each neuron, limits the number of required bits. For instance, using tansig nonlinear function enables us to store the result of addition in the range of:

$$-4 < \sum_{j=1}^{x(i)y(i)} w(j) < 3.998$$

Results out of this range can be simply replaced by these limits. This poses a limit on the number of bits for storing the result of accumulation in each node. This can greatly decrease the amount of hardware.

Implementation of non-linear function is done through the memory blocks. One or two memory blocks are used to perform look up operations for all of the nodes in the input and hidden layer. As mentioned above, the input value to non-linear look up tables is expressed with $q + k$ bits in which k is a small value (2 or 3) and the output is again a q-bit value. Accordingly, non-linear look up operations consume a small amount of block memory resources.

Computations in the second layer needs embedded hardware multipliers of Virtex-IIP. Each of the nodes in the hidden layer can use one dedicated hardware multiplier. Each of the multipliers in a Virtex-IIP FPGA is capable of doing a single cycle pipelined 18*18 bits multiplication. In practice, q is in the range of 7 to 12 and therefore, one multiplier suffices for each hidden node.

The result of each multiplication is q^2 bits, which can be converted into a q or $q + k$ bits value by omitting LSB bits. Accumulation logic is similar to the circuitry in the input layer. Non-linear look up is also the same. Total number of N_1 clock cycles is required for computing the accumulation values in hidden layer.

Output layer has a linear function and therefore, special look up circuitry is not needed. Output value can be $q + k$ bits. Output value will be passed to arbitration module.

Arbitration and Decision

Initially arbitration module normalizes input values to either one or zero using a special threshold. This threshold depends on the rate of detection against false positive. The outputs of the MLP for each scale value are stored in a separate array. Values in this array indicate the output of the MLP for that special location of the image.

The network has some invariance for scaling and positioning which may result in multiple answers for a particular face. False detections often occur with less consistency [9]. Thus for each detection we count the number of other detections within a specified neighborhood of that detection. If the number is above a threshold, then that location is classified as a face. In order to prevent overlaps, once each location is considered as a face any other detection in a specified neighborhood are discarded.

The above analysis is performed for each array. Final results are merged into one output array, indicating the location for face objects.

A program running under embedded processor in Virtex-IIP device is responsible for the above task. Embedded PowerPC processor operates with the same clock frequency as the rest of the chip. Since there is a small amount of computations, for each of the scales, processing time for this module is less than the processing time of the MLP and therefore, this part of the system will not limit the final frame rate.

Practical Results

We implemented most of the parts of our system in real hardware. Table 1 shows chosen values for jumps, window sizes and scales. As described before, neural network accepts two different dimensions: 18*22 pixels and 36*44. Table 1 shows which mode is used for every window size.

For every input with size of 800*600 pixels, the total number of 112780 sub frames is generated. Each individual sub frame is analyzed with 10 different degrees of rotation in 22.5 degrees increments.

Total numbers of 2.099×10^7 cycles is required to analyze one complete frame ($K = 36$). Running at 200MHz clock frequency, the face detector can process up to 9 complete frames per second.

In practice, however, it is not required to process the entire frame. Usually face objects have a very smaller area compared to the total area of the image. Thus, the skin color filter output is zero most of the time. Our face detection system can gain a significant speed-up. For instance, if only 25% of an image contains face objects, the system is able to process up to 27 frames per second.

Number of rotation angles for which we analyze the image, is another important factor. For instance, if only vertical direction is considered, the processing speed would reach up to 90 frames per second.

Frame size x_F, y_F	18*22	26*32	32*39	36*44	51*63	63*77	72*88	81*99	96*117	114*140
Surrounding rectangle size $x_{SubFrame}$	29*29	42*42	51*51	57*57	82*82	100*100	114*114	128*128	152*152	181*181
MLP input size	18*22	18*22	18*22	36*44	36*44	36*44	36*44	36*44	36*44	36*44
s^2	1	2	3	1	2	3	4	5	7	10
Jump value (J)	3	4	6	6	8	9	12	15	21	30

Table1. Window sizes, jump and scale values.

Experimental results show that 8 or 9 precision bit is sufficient for neural network weights. Using 9 bits for weight values and a neural network with 9 neurons in input layer, 6 neurons in hidden layer and 1 in output layer, the total number of $9 \times 36 \times 9$ bits data must be loaded from memory ($K = 36$). This means 21 block memories (two blocks per node) with maximum possible output port width of 72 bits [12] should be used in parallel to provide the needed weight values. Each block stores 55 values, and therefore, 25% of available space in each block memory is consumed.

Design parameters, yet can be simply modified to accommodate various special tasks. For example if we assume $f = 100MHz$ and $f_M = 400Mhz$, only 11 blocks of memory is utilized. Changing the jump value or scale values or number of scales can greatly change the resource utilization and frame per second speed. A SVM can be used instead of our simple MLP to obtain better classification results [10] [11].

Experimental results show that the neural network consumes approximately 5×10^3 LUTs, Rotation uses 2.5×10^3 LUTs while scaling and grouping module use 10^3 and 2.5×10^3 respectively. Since the rest of the system consumes up to 7×10^3 LUTs an XC2VP20 device [12] can hold the entire implementation.

Out tests show that the system can reach very good accuracy in detecting faces if a good skin color filter and well trained MLP are used.

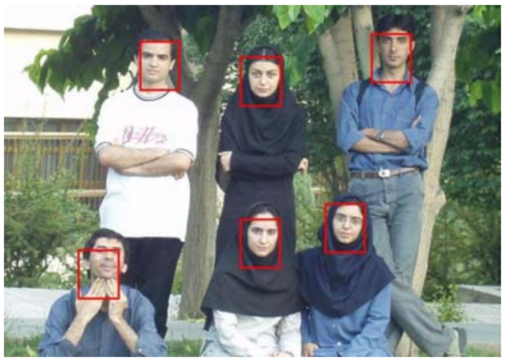
Figure 6. 800*600 Input image, simulation outputs for our system.

9 Bits quantized weights	Output – Real weights	Output – Quantized weights
Face object	1.0006	0.9685
Non Face object	*0.0396*	*0.0073*

Table 2. Shows the weight quantization effect in MLP.

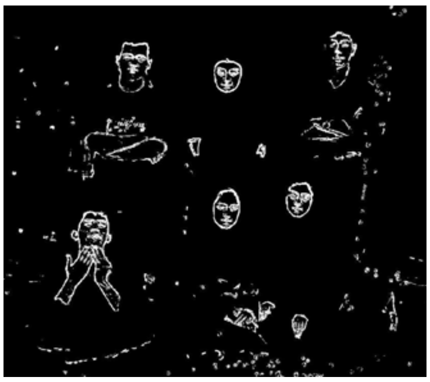
Figure 7. Input image to sub-frame generation

References

1. Muthukumar Venkatesan and Daggu Venkateshwar Rao, *Hardware Acceleration of Edge Detection Algorithm on FPGAs,*
2. J. F. Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (6):769–798, November 1986.
3. Jure Kova·c, Peter Peer, Franc Solina , *Eliminating the Influence of Non-Standard, Illumination from Images,*
4. Kobus Barnard, Lindsay Martin, and Brian Funt, *Colour by Correlation in a Three-Dimensional Colour Space*, Sixth European Conference on Computer Vision 26th June - 1st July, 2000
5. Brian Funt, Kobus Barnard and Lindsay Martin , *Is Machine Colour Constancy Good Enough?,* 5th European Conference on Computer Vision (ECCV '98) 2-6 June, 1998
6. Jeonghee Park; Jungwon Seo; Dongun An; Seongjong Chung; *Detection of human faces using skin color and eyes,* Multimedia and Expo, 2000. ICME 2000. 30 July-2 Aug. 2000
7. Ikeda, O.; *Segmentation of faces in video footage using HSV color for face detection and image retrieval*, International Conference on Image Processing, 14-17 Sept. 2003
8. Robert D. Turney and Chris H. Dick, *Real Time Image Rotation and Resizing, Algorithms and Implementations,*CORE SOLUTIONS GROUP, XILINX, INC.
9. H. A. Rowley, S. Baluja, and T. Kanade, *Rotation invariant neural network-based face detection*, Computer Science Technical Report, CMU, Pittsburgh, 1997.
10. Haizhou Ai,L. Ying,Guangyou Xu, *A Subspace Approach To Face Detection With Support Vector Machines,* ICPR 2002, August 11-15
11. R.A. Reyna-Rojas et all , *Implementation of the SVM Generalization Function on FPGA*, GSPx 2003
12. Xilinx Inc., *Virtex II Pro Platform FPGAs Complete data sheet,*
13. Howard demuth, Mark Beale, *Neural Network Toolbox for use with Matlab,* Version 4.0, July 2002.