EURASIP Journal on Applied Signal Processing 2005:7, 1–15

An FPGA-Based People Detection System

Vinod Nair
Centre for Intelligent Machines, McGill University, Montreal, Quèbec, Canada H3A 2A7
Email: vnair@cim.mcgill.ca

Pierre-Olivier Laprise
Centre for Intelligent Machines, McGill University, Montreal, Quèbec, Canada H3A 2A7
Email: plapri@cim.mcgill.ca

James J. Clark
Centre for Intelligent Machines, McGill University, Montreal, Quèbec, Canada H3A 2A7
Email: clark@cim.mcgill.ca

Received 15 September 2003; Revised 12 August 2004

This paper presents an FPGA-based system for detecting people from video. The system is designed to use JPEG-compressed frames from a network camera. Unlike previous approaches that use techniques such as background subtraction and motion detection, we use a machine-learning-based approach to train an accurate detector. We address the hardware design challenges involved in implementing such a detector, along with JPEG decompression, on an FPGA. We also present an algorithm that efficiently combines JPEG decompression with the detection process. This algorithm carries out the inverse DCT step of JPEG decompression only partially. Therefore, it is computationally more efficient and simpler to implement, and it takes up less space on the chip than the full inverse DCT algorithm. The system is demonstrated on an automated video surveillance application and the performance of both hardware and software implementations is analyzed. The results show that the system can detect people accurately at a rate of about 2.5 frames per second on a Virtex-II 2V1000 using a MicroBlaze processor running at 75 MHz, communicating with dedicated hardware over FSL links.

Keywords and phrases: computer vision, FPGA, people detection, smart camera.

1. INTRODUCTION

This paper describes a system for detecting people in images, implemented on a field-programmable gate array (FPGA). People detection is an important subtask in many computer vision applications, such as automated video surveillance, human activity recognition, and smart room systems. The output of a people detector can be used, for instance, to infer a person's location in a scene or to track the person over time. Such location and tracking data can then be analyzed to automatically generate a human-understandable description of what the person might be doing, or raise an alarm if the person's behavior seems unusual.

Many vision applications often involve a large number of cameras. For example, wide-area surveillance networks use tens to hundreds of cameras to monitor many different scenes. Sending the video from all the cameras to a single central workstation for processing can be prohibitively expensive because of the need for high-bandwidth transmission. An attractive alternative is to perform the processing on the camera itself with a fast and inexpensive FPGA chip. In recent years, FPGA technology has become increasingly powerful, less expensive, and more practical for use in real-time vision applications. Our long-term goal is to build a framework in which a large number of cameras cooperate to carry out a collective task (such as surveillance), with each performing its own FPGA-based video analysis, and exchanging high-level, low-bandwidth information over a network.

As a first step toward such a framework, we have implemented a single-camera people detection system on an FPGA. The system is demonstrated on a corridor surveillance application. Here the task is to detect people appearing in an office corridor from the JPEG-compressed frames provided by a fixed network camera. Example frames from the camera are shown in Figure 1. By "detecting people," we mean computing an accurate bounding box for each fully visible person in a frame.

Our approach to people detection uses a classifier trained by a supervised machine-learning algorithm. This is in contrast to the traditional approach based on techniques such as background subtraction or motion detection. Such tech-

(a) (b) (c) (d)

Figure 1: Example images of the scene under surveillance.

niques are relatively easy to port on to a chip because they mainly involve simply differencing images. An example of this approach implemented on an FPGA is [1]. However, detection methods based on background subtraction or motion can often fail because of sudden changes in scene illumination, shadows, similar pixel values shared by the foreground and the background, and other difficult background modeling problems. As explained later, the learned classifier-based approach is not affected by these problems and therefore can be more robust and accurate. The detection algorithm we use is developed by Viola and Jones [2]. This algorithm is described in Section 3.

Such an approach to people detection is also an unusual choice for FPGA implementation, since vision and image processing algorithms successfully ported to FPGAs tend to be of the data-streaming or small-neighbourhood filter variety. Many of the more complex algorithms are frequently deemed unsuitable for FPGA implementation due to the necessity for floating-point operations or because of the limited amount of rapid, on-chip memory compared to the volume of data that needs to be processed. Instead of avoiding these problems by changing algorithms, we explore methods to mitigate their effects on system performance, thereby expanding the range of algorithms suitable for FPGA. Details of the firmware implementation of the people detection algorithm and JPEG decompression are provided in Sections 4 and 5.

Another contribution of our work is an algorithm that combines the JPEG decompression with the detection algorithm in a hardware-friendly manner. The Viola-Jones detector needs to compute an "integral image" of a video frame in order to detect people in that frame. We present a method for computing an approximate integral image directly from JPEG, without computing the fully decompressed frame. This method is simpler to implement on an FPGA than a full-fledged JPEG decompressor, it takes up less space on the chip, and it requires fewer computations compared to calculating the integral image from a fully decompressed frame. The algorithm for approximating the integral image is described in Section 6. The firmware implementation details of the approximation algorithm are given in Section 7.

An overview of the system we have implemented is given in Section 2. Accuracy results for the trained detector are presented in Section 8.

2. SYSTEM OVERVIEW

People detection is rarely an end in itself, but more often an intermediate step in a higher-complexity algorithm. Our approach to this problem is to have multiple, independent camera-based processing units communicating high-level information about a scene instead of raw data. In the more complex applications, one can expect that the camera network will be highly heterogeneous, with cameras implementing low-level algorithms and passing on their results to cameras with higher-level algorithms, in an analogous manner to the multiple levels of processing found in the human visual system.

To simplify the design of the various modules that would be required in such a network, it is important to have a powerful yet flexible and adaptive framework. Cost is also a factor due to the large number of nodes that should compose the network. In fact, it should be expected that the system's true power comes from distributing the processing load over multiple nodes rather than from the processing power of the individual nodes in the network.

The desire to have as many nodes as possible combined with the requirement that each node be capable of executing moderately complex algorithms in real time were the guiding constraints in node design. This required the solution to be as cost-effective as possible, yet still capable of high-performance image processing. Reconfigurable computing architectures have shown time and again an ability toward accelerating image processing tasks [3, 4, 5, 6, 7], and are therefore ideally suited to this application.

Current advances in FPGA technologies are making it possible to envisage the design of a system on a programmable chip (SOPC), in which all the components which previously required separate components on a printed circuit board (PCB) can be fit onto a single FPGA chip. This allows embedding of a microprocessor on the FPGA itself instead of placing it alongside on a PCB. Xilinx provides two avenues to this effect, the MicroBlaze soft processor for the Virtex-II family of chips, and the IBM PowerPC processor embedded into the Virtex-II Pro family of FPGAs.

Using such an approach, the microprocessor can be used to control the general behavior of the system, as well as the complex network communication protocols. In parallel, an application-specific firmware (ASFW) module can be configured to do the bulk of the processing, taking full advantage of

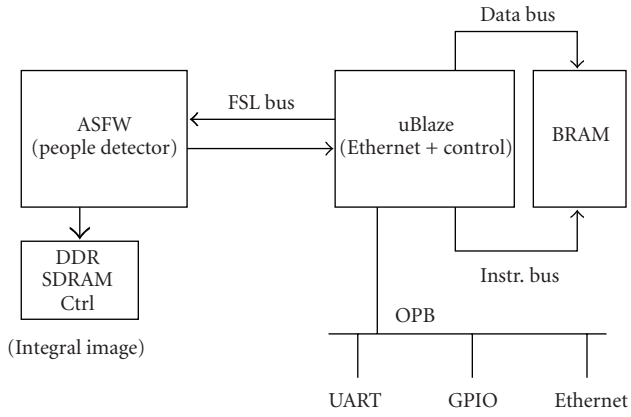

Figure 2: Organization of IP cores on the FPGA for a single-chip solution.

the parallelism offered by custom hardware designs. This also permits a tight, optimized coupling between the data processing elements and the data source, such as the camera or memory.

The more conventional approach to a reconfigurable computing architecture is to have a dedicated microprocessor ASIC using an FPGA as a reconfigurable coprocessor. Such a configuration has many advantages, namely, performance, but it sacrifices some of the flexibility and cost-effectiveness of a single-chip solution.

Other limitations were imposed by the prototyping platform that was initially chosen. Although the video input will be tightly coupled to the FPGA in future versions, no readily available commercial board was found to satisfy this requirement along with all the others. The chosen prototyping board was Insight Memec's V2MB1000 board, which provides a large variety of I/O, such as LVDS pins, which will be necessary for high-speed board-to-board communication. It was decided that the actual source of the video feed had limited impact on the functionality of the system, and that reasonable performance estimates could still be garnered from the implementation.

For the node application example described in this paper, the source images were gathered from a network camera with onboard JPEG compression and HTTP server over Ethernet, Axis Communication's NetCam 200+. Unfortunately, the live stream from this camera is limited to 1 352 $\times$ 288 pixel frame per second. Having a JPEG video source allowed exploration into the possibilities of integrating real-time image compression and decompression into a standard computer vision algorithm. Integrating compression and decompression of a data stream into the application (whether using JPEG or other means) is an important step toward reducing the bandwidth required in transferring data between network nodes or even between the FPGA and memory. In fact, hardware implementations of computer vision and image processing algorithms are more often limited by the speed of their I/O than by the speed at which they are able to process data, which is why most of the algorithms implemented in hardware are of the streaming filter variety, where memory demands are limited. Compressing the data that is to be transferred therefore has the potential of improving the system performance by increasing the effective bandwidth as long as compression and decompression of the data have a limited effect on the speed of the overall system.

The global system organization can be seen in Figure 2. Execution starts when the MicroBlaze processor retrieves the JPEG image from the network camera using HTTP over Ethernet. The JPEG data is separated from the header and passed to the JPEG decoder and people detection module, which is described in more detail in this paper. The integral image (an intermediate format needed by the algorithm) is stored in DDR SDRAM for the use of the application-specific firmware, to which it is tightly coupled with a controller optimized for this specific application to minimize overhead costs.

3. THE PEOPLE DETECTION ALGORITHM

People detection is performed using the Viola-Jones algorithm, which is a general method that can be used to learn a detector for any type of object. Here we give a brief summary of the algorithm as it applies to our work. For a full description, see [2].

3.1. *Viola-Jones detection algorithm*

The basic detection strategy of the Viola-Jones algorithm consists of two steps. First, an image classifier is trained to accurately classify cropped people and nonpeople images (examples of such images are shown in Figure 3). Then, instances of the "people" object are detected in a given video frame by "scanning" the frame with the classifier. Scanning involves sliding a window over the frame, and the classifier is asked to label the subimage defined by each window position as either "people" or "nonpeople." When the window is right on top of a person in the image, the classifier will (hopefully) label it as "people" and thus detect the person. People of different sizes can be detected by scanning at many different window scales. Note that this strategy does not make any assumptions about the scene background or interframe difference, so it is not affected by problems such as sudden illumination changes, shadows, and other difficult background modeling problems that commonly plague detection methods based on background subtraction or motion.

The Viola-Jones algorithm uses the AdaBoost learning algorithm [8] and a set of cropped people and nonpeople images to train the image classifier. Learning is performed on a local feature-based representation of the training images, instead of on the raw pixel representation. These features are rapidly computable and allow for efficient scanning at different scales. Examples of the three types of features we use in our detector are shown in Figure 4, as superimposed on a cropped image. For all three types, the numerical value of the feature is the absolute difference between the sum of the white pixels and the sum of the black pixels. The features are essentially local edge and bar detectors with different orientations. For each type, thousands of features can be defined by varying the location and the size of the feature within the

Figure 3: Examples of cropped people and nonpeople images.

Figure 4: The three feature types used by the people detector.

cropped image. We use a total of 21804 such features for all three types. Therefore, during training, each cropped image is represented as a 21804-dimensional feature vector.

Most of these features are likely to be useless for classifying people and nonpeople images. The AdaBoost learning algorithm is used to select the few features that are actually useful for accurate classification. The final classifier obtained at the end of the training contains only a small fraction of the initial pool of features. So when a frame is scanned using the classifier, only $O(100)$ features have to be computed per subimage, rather than all 21804.

These features can be computed rapidly using what Viola and Jones call the "integral image." Given a grayscale image, its integral image is a matrix whose element at the rth row and cth column is the sum of all pixels up to and including row r and column c of the grayscale image. Once the integral image is computed, the sum of any rectangular region of pixels in the image can be computed with only four additions, as explained in [2]. This means that the value of a local image feature can be computed very quickly at any scale. So when scanning a frame for people at various scales, it is not necessary to resize the frame at those scales with an image pyramid. Instead, the classifier itself can be "resized" simply by rescaling the features it uses. As a result, scanning is computationally much more efficient than with the pyramid approach. What makes this algorithm particularly attractive for hardware implementation is that most of the arithmetic operations involved during detection are additions.

To improve the speed of the detection process, a cascade of strong classifiers is constructed, instead of just one strong classifier. This strategy is based on the observation that almost all subimages encountered when scanning a frame belong to the negative class (i.e., nonpeople). Therefore, the structure of an efficient classifier should be geared toward rejecting negative instances as quickly as possible, with large amounts of classification effort expended only on the rare positive instances. To classify a subimage, it is passed through a cascade sequence of strong classifiers until one of them rejects it as a negative instance. If the subimage survives all stages of the cascade, then it is labeled as a positive instance. The further down the cascade a strong classifier is, the more features it contains. As a result, the small initial cascade stages quickly eliminate the "easy" nonpeople instances. More computational effort is required to reject the remaining, more ambiguous nonpeople instances by the subsequent larger stages. Although classifying a subimage as "people" requires evaluating all the features in the entire cascade, such subimages are rare. Therefore, the average number of features computed per subimage, which determines the overall speed of the detector, still remains fairly small.

The dimensions of the network camera frames we use are 352×288. Scanning is done only in a 216×288 rectangular region in the middle of the frame that corresponds to the part of the corridor where people actually appear. The size of the scan window is restricted to be a multiple of 16×48 (width $\times$ height). Allowed multiples go from 1.0 (16×48) to 6.0 (96×288) in increments of 0.25. The number of pixels by which the subwindow is shifted horizontally and vertically is computed as 25% of its width and height, respectively. In each frame, 3079 subimages have to be classified during scanning.

4. PEOPLE DETECTION IN FIRMWARE

At first glance, the implementation of Viola and Jones' detection algorithm is rather straightforward. The module implementation's flowchart is shown in Figure 5. Having stored the image in integral image format, the sum of pixels in a rectangular region of any size only requires the addition or subtraction of the region's four corners. Therefore, all that is

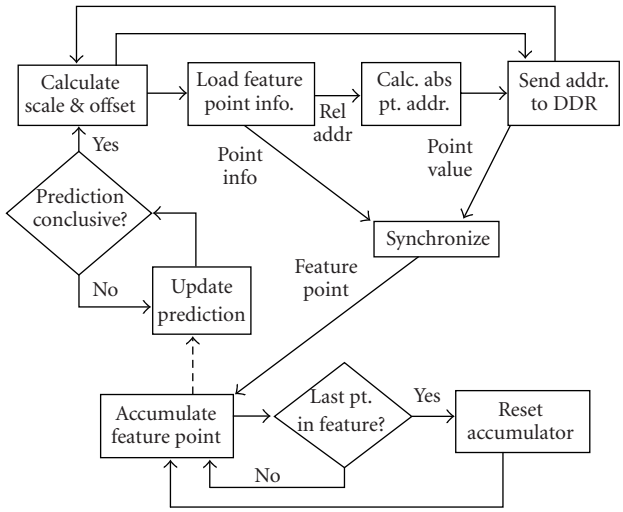

Figure 5: Flowchart for the firmware implementation of the people detection algorithm.

required to calculate a feature's value is a simple accumulator circuit. Each accumulated point is either multiplied by ± 1 or ± 2, and the features that were used can have either 6 or 8 points of interest. The only difficulty stems from the limited amount of on-chip memory available. Since the integral image is much too large to fit on-chip, it must be stored in an external memory, which is necessarily much slower to access. In the current implementation, the off-chip memory is a DDR SDRAM with a 16-bit data bus working at 100 MHz. This allows for a transfer rate which can never surpass 32 bits per 10 nanoseconds. With the targeted FPGA system clock speed also at 100 MHz, the system would be receiving at most one integral point per cycle, even without taking into account addressing overhead and memory refresh times. Various methods were considered in order to compress the integral image and allow greater throughput, but the necessity to access widely separate points in a pseudorandom order made this a difficult task. Given that we have a priori knowledge of the patterns with which data points can be fetched, it should be possible to optimize the memory controller for this application, for example, by inserting memory refreshes in natural pauses in the flow, but this would be far from trivial and is therefore left as future work.

Classifier training determines the position, size, and type of features that are required to detect a person. These values are all given with respect to a 16×48 template window, which is then shifted and scaled to detect people of different sizes in varying places in the image. The address of a point in external memory therefore depends both on its position in the template window, and on the template's position and scale in the image at any given point in the scan. Training also determines the feature's threshold, its weight in the stage, and the global threshold for each stage. In order to minimize delays, this information should all be stored on-chip, but this can require a large amount of memory if one is not careful. Consequently, it is necessary to organize the information so as to most tightly pack it into the available memory formats.

The first step is to determine the minimum required data width for each signal. This is simple for most signals, but some require more detailed analysis, and can usually be split into absolute and relative ranges. Analysis of absolute ranges is simply a question of finding the maximum and minimum values for a variable and scaling it to fit an integer of minimum size. An example of such a signal is the feature threshold, which is determined by the feature size and maximum pixel value, such that it can necessarily fit into 18 bits. However, analysis of the training data shows that no threshold ever needs more than 15 bits. Since the system can be reconfigured through judicious use of parameters if this changes, the optimal size for a given training session can be used for storage.

Relative ranges, such as that of the feature weights, only have relevance one with respect to the others, and can be represented as numbers of arbitrary precision between zero and one. These ranges can never be fully covered, but statistical analysis of the training results yields the error associated with a given data width. For example, normalizing the weights to 18-bit integers allows for all but 0.13% of the weights to have a unique value compared to the full floating-point representation.

Once the data widths have been optimized, the signals which are always retrieved together can be packed into a single word in memory to reduce the number of memory accesses. This mapping, however, should be encapsulated in such a way as to present the functional separations rather than the actual ones in order to facilitate code reuse and maintenance.

5. CALCULATING THE INTEGRAL IMAGE FROM JPEG

Computing the integral image of a grayscale frame is simple (see [2] for details) if the frame is not compressed. In our case, however, it is compressed in JPEG format. The JPEG decompression algorithm involves computing the inverse discrete cosine transform (DCT) [9], which requires nontrivial hardware resources and computational effort. Therefore, we seek to avoid computing the inverse DCT. It is possible to obtain the integral image directly from the DCT coefficients because both the forward and inverse discrete cosine transforms are linear transformations, which means that the coefficients are linear combinations of pixel values and vice versa. So the pixel sums required in the integral image computation can be obtained through linear combinations of the DCT coefficients.

Figure 6 shows how such a direct method would work, and for comparison, the gray box shows the indirect method of computing the integral image. But computing the inverse DCT and computing the integral image from the DCT coefficients are roughly equivalent since both the grayscale frame and its integral image contain the same amount of information, and the conversion between the integral and grayscale forms is trivial compared to the inverse DCT. Therefore, calculating the integral image directly from the DCT coefficients

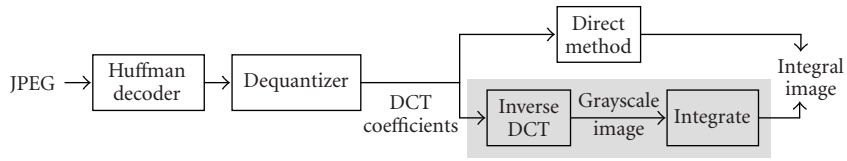

Figure 6: Once the JPEG image's DCT coefficients are decoded, the standard method would be to perform an inverse DCT and integrate the resulting grayscale image. Instead, the integral image can be extracted directly from the coefficients.

requires about as much effort as the inverse DCT itself. However, it may be possible to directly compute an *approximate* integral image with fewer computations.

5.1. Extraction of DCT coefficients

The extraction of DCT coefficients from a JPEG stream requires first that a Huffman-encoded value be decoded, which is then used to decode the bits in the stream which encode the coefficient's actual value. This necessitates the use of a Huffman table which is transmitted with the image. However, JPEG encoders (including the one used for this experiment) generally use the same Huffman table for all the images that they generate. Having verified whether this is the case for a particular encoder, and with knowledge that all future images in the series will come from the same encoder, it is possible to only extract the table from the first image received or, in a prototyping environment, to hardcode the tables into the FPGA's configuration bitstream. The quantization tables used in Section 7 may be treated in the same manner.

It was clear from the start that the speed at which the JPEG decoding module processes data would be limited at the input by the fact that the data is being sent over a 10/100 Ethernet line, which has a maximum transfer rate of 10 ns/bit, and at the output by the integral image module, which needs to write its results to external memory. Therefore, a simple serial lookup table approach to Huffman decoding, such as the one described in [9], should be sufficient to meet data rate limitations at both ends. Once the Huffman decoding is complete, decoding the coefficient's value and index is relatively simple to do in parallel at a small additional cost in complexity. Simplified block diagrams of these modules' implementations can be seen in Figures 7, 8, and 9.

Tests on a source image suggest that if the decoding hardware were only slightly limited by input speed (overhead of 2 cycles per 16 bits of data), the hardware should take approximately 75 kcycles to treat a typical image, which translates to 1.5 milliseconds for a worst-case 20 nanoseconds minimum period. However, as will be seen in Section 7, most of this time can be absorbed by the calculation of the integral image, with a simple FIFO buffer to synchronize the modules.

6. AN EFFICIENT ALGORITHM FOR APPROXIMATING THE INTEGRAL IMAGE

We have developed an algorithm for calculating an approximate integral image that needs significantly fewer computations and hardware resources than the inverse DCT. The basic idea is to compute the integral image exactly at some points in the image and then approximate it everywhere else by interpolation. The JPEG compression algorithm partitions a grayscale image into nonoverlapping 8×8 pixel blocks and computes the 64 DCT coefficients for each block. These coefficients can be obtained from the JPEG data by Huffman decoding and dequantization [9]. Since the DC coefficient of a block encodes the average pixel value of that block [10], the sum of all pixels in an 8×8 block can be calculated from its DC coefficient alone. Using all such local 8×8 block sums of an image, it is possible to compute the exact value of the integral image at the bottom-right corner of every 8×8 block in the image, as shown by the example in Figure 10a. Suppose that S_1, S_2, S_3, and S_4 are the 8×8 block sums for the four blocks shown in Figure 10a. Then the exact value of the integral image at point A is S_1, at point B is $S_1 + S_2$, at point C is $S_1 + S_3$, and at point D is $S_1 + S_2 + S_3 + S_4$. The rest of the integral image can then be filled in by interpolating these exact values, but the resulting approximate integral image may have a large error compared to the true integral image.

The approximation error can be reduced, at a greater computational expense, if we divide up each 8×8 block into four 4×4 blocks and calculate all the 4×4 block sums from the DCT coefficients. Then the exact integral image value can be obtained in a similar manner as above at four times more points than before, as shown in Figure 10b. Reducing the error further by computing the exact integral image values even more densely further diminishes the benefits of avoiding the inverse DCT. (Taken to the extreme, reducing the error to zero by computing the exact integral image values everywhere becomes roughly equivalent to the inverse DCT.) How much approximation error can be tolerated in the integral image should be determined by how the error affects the detection accuracy of the people detector. As the results in Section 8 show, the approximate integral image computed from 4×4 block sums provides a reasonable balance of high people detection accuracy and low computational effort, and therefore shows that is the approximation level we have chosen.

Given an 8×8 block of DCT coefficients, how can the 8×8 and 4×4 block sums be calculated for that block? If we consider the DCT coefficients in the block to be a 64-dimensional vector $\mathbf{d}$, and the corresponding 8×8 block of pixels to be a 64-dimensional vector $\mathbf{p}$, then we can write

$$\mathbf{p} = A\mathbf{d}, \tag{1}$$

where A is the constant 64×64 matrix representing the inverse DCT. Computing the sum of an arbitrary set of pixels

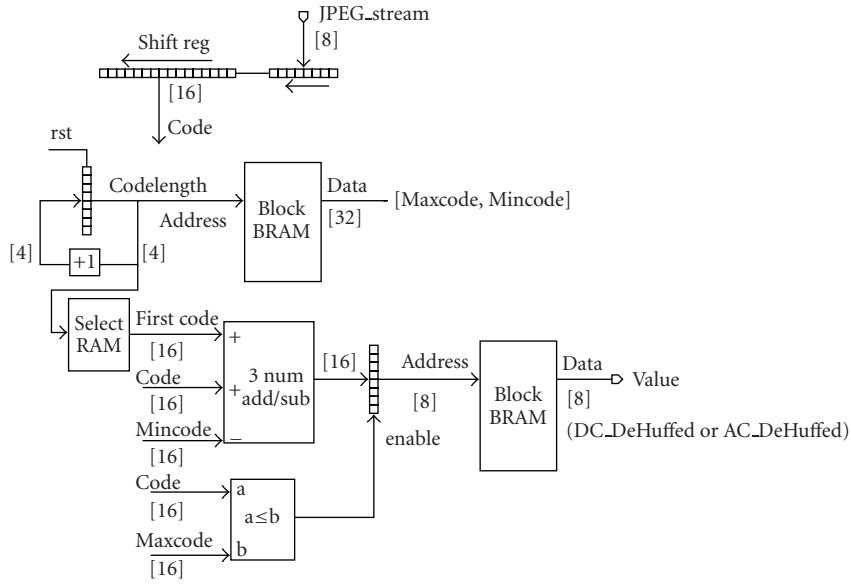

Figure 7: Simplified block diagram for Huffman decoding module.

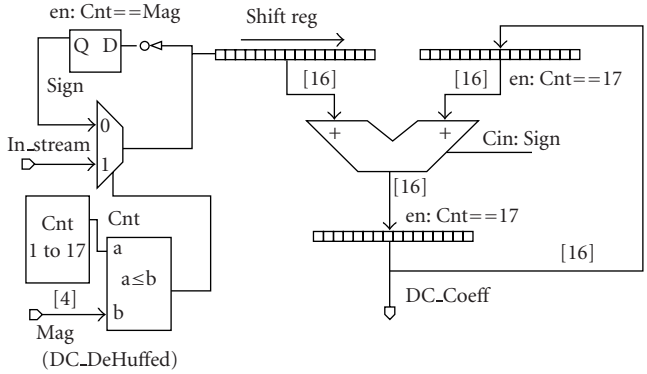

Figure 8: Simplified block diagram for decoding JPEG DC coefficients.

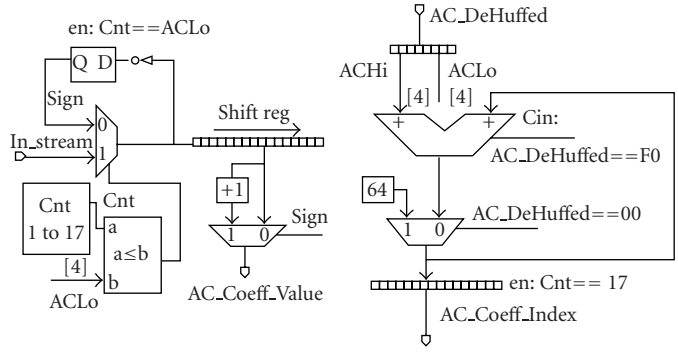

Figure 9: Simplified block diagram for decoding JPEG AC coefficients.

within an 8×8 block is the same as taking the dot product of $\mathbf{p}$ with a 64D vector $\mathbf{i}$ whose components corresponding to the pixels included in the sum are 1 and all other components are 0. So the sum S of the pixels can be written as

$$S = \mathbf{i}^t \mathbf{p} = \mathbf{i}^t A \mathbf{d}. \tag{2}$$

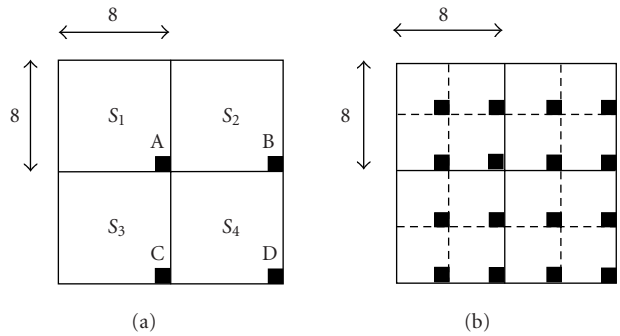

FIGURE 10: Black squares denote the pixel locations where the exact value of the integral image is computed from the sum of 8×8 and 4×4 pixel blocks.

Let $\mathbf{r}^t = \mathbf{i}^t A$. Then we get

$$S = \mathbf{r}^t \mathbf{d}. \tag{3}$$

For a given $\mathbf{i}$ (i.e., a given set of pixels to add up in an 8×8 block), $\mathbf{r}$ is a constant vector that can be precomputed independently of $\mathbf{d}$.

For example, to find the sum of all pixels in any 8×8 block, we set all components of $\mathbf{i}$ to 1 and then compute $\mathbf{r}^t = \mathbf{i}^t A$. The components of the resulting $\mathbf{r}$ turn out to be all zeros except for the one that multiplies the DC coefficient, which has a value of 8. This means that the sum of an 8×8 block can be computed by multiplying the block's DC coefficient by 8. Once $\mathbf{d}$ is computed for a particular 8×8 DCT block, the pixel sum is given by the dot product of $\mathbf{r}$ and $\mathbf{d}$. Note that the number of additions and multiplications needed to compute the dot product of $\mathbf{r}$ with any $\mathbf{d}$ is equal to the number of nonzero components of $\mathbf{r}$.

To find the four exact integral image values in an 8×8 block, the sums of the shaded pixels denoted by S_1, S_2, S_3 and S_4 in Figure 11a are needed. Direct computation of these sums requires 100 additions and multiplications because the $\mathbf{r}$ vector for each sum contains 25 nonzero components. However it is possible to obtain the 4×4 block sums indirectly with fewer additions and multiplications using the pixel sums denoted by W_1, W_2, W_3, and W_4 in Figure 11b. The advantage of these sums is that they can be computed with a total of only 27 additions and multiplications—W_1 and W_2 need 5 adds and multiplies each, while W_3 needs 17. As mentioned before, W_4, the sum of all pixels in a block, is calculated by multiplying the block's DC coefficient by 8, which can be done with shifts. Then the 4×4 block sums can be computed as follows:

$$\begin{aligned} S_1 &= \frac{W_1 + W_2 + W_3 - W_4}{2}, \\ S_2 &= W_2 - S_1, \\ S_3 &= W_1 - S_1, \\ S_4 &= W_3 - S_1. \end{aligned} \tag{4}$$

6.1. Interpolating exact integral image values

Once the exact integral image values are obtained, the rest of the image is filled in by interpolation. There are many different types of interpolation methods that can be used here, but to keep the computation hardware-friendly, we assume that the integral image values are approximately linear within a 4×4 neighborhood and use simple local linear interpolation. This is equivalent to assuming that the pixel values in a 4×4 neighborhood of the grayscale image are equal, because integrating a constant pixel neighborhood results in a linear integral image neighborhood.

The interpolation can be done in two steps: initially the integral image consists of 5×5 neighborhoods of the kind shown in Figure 12a. The black squares are the points where the integral image values have already been computed and the white squares are the missing points. In the first step, the gray squares in Figure 12a are obtained using the equations shown there. The four gray squares along the border of the 5×5 neighborhood are computed by averaging the two nearest black squares, and the middle gray square is computed by averaging all four black squares.

After the first interpolation step, the integral image consists of 3×3 neighborhoods of the kind shown in Figure 12b. The procedure for filling in the remaining missing points in the 3×3 neighborhood is analogous to that of the 5×5 neighborhood, as shown by the equations in Figure 12b. A valid integral image must be nondecreasing (since grayscale pixels are never negative), and it can be easily shown that the interpolated values computed using the equations in Figure 12 do satisfy the nondecreasing requirement, provided that the exact integral image values satisfy them.

This interpolation scheme is suitable for hardware implementation because it only requires additions and divisions by 2 and 4, which can be done with shifts. The total work needed to fill in an 8×8 block is 27 multiplications, 64 additions, and 20 shifts. On the other hand, the inverse DCT is an $O(n \log_2 n)$ algorithm, so it requires on the order of $64 * \log_2 64 = 384$ multiplications and additions for an 8×8 block. Basically, the savings in computational effort and hardware resources come from replacing the multiplications in the inverse DCT algorithm with shift operations.

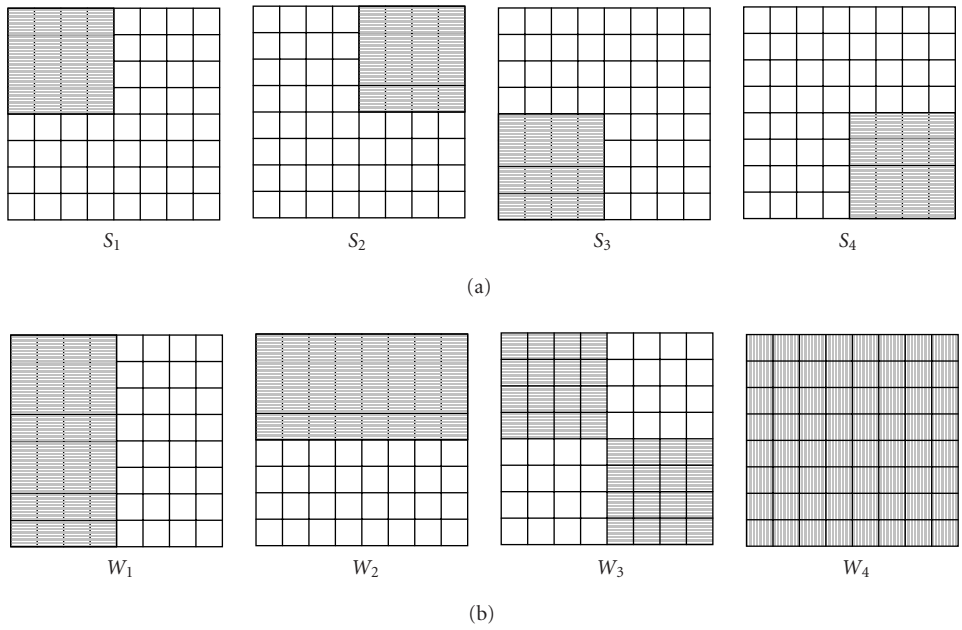

FIGURE 11: Two alternative sets of pixel sums (of the shaded regions) that can be used to compute the four exact integral image values in an 8×8 block. Set (b) requires fewer additions and multiplications to compute from DCT coefficients than set (a).

The algorithm for computing the approximate integral image is related to the idea of decompressing a JPEG image by "scaled decoding." Scaled decoding is a feature of the JPEG format that allows efficient decompression of an image at either 1/2, 1/4, or 1/8 of its original resolution. Our algorithm can be thought of as first computing a grayscale image by scaled decoding at a lower-resolution, but still maintaining the same dimensions as the original image by filling in the missing pixels with replicas. Then this lower-resolution grayscale image is integrated to obtain an approximate integral image.

7. FIRMWARE IMPLEMENTATION OF THE APPROXIMATE INTEGRAL IMAGE

The algorithm described in Section 6 uses sums of DCT coefficients multiplied by a constant to exactly calculate the half-block integral points. Since the coefficients are fed sequentially to the module by the JPEG coefficient extractor, this can be implemented using a multiply and accumulate (MAC) circuit for each point, as illustrated in the simplified block diagram in Figure 13. Careful inspection of the coefficient multipliers shows that their values are dependent on the index of the multiplied DCT coefficient rather than the position in the position of the point that it is being accumulated for. This suggests that a single multiplier can be shared by all the points. The MAC circuit also intrinsically takes advantage of zero runs in the JPEG stream, since not accumulating these coefficients is the same as accumulating 0. This means that the time needed to calculate a point is dependent on the number of nonzero coefficients in the image.

However, JPEG DCT coefficients only contain information about the 8×8 block that they are in, and are totally independent of their position in the image. An integral image point, on the contrary, is dependent on all the points above it and to its left in the image. It is therefore necessary to offset each block-integral point extracted from the DCT coefficients by the integral image as it has been accumulated so far. Referring to Figure 14, where the grayed-out portions are the blocks that have already been received, and the black squares are the half-block integral points that are extracted directly from the DCT coefficients, it is evident that going from the block-integral points that are extracted from the coefficients to the final image-integral points requires points from the blocks immediately above and to the left of the current block. For any given 8×8 block decoded from JPEG, the desired image-integral points, $(r+3, c+3)_i$, $(r+7, c+3)_i$, $(r+3, c+7)_i$, $(r+7, c+7)_i$, where $(r, c)_i$ is the position of the upper-left pixel of the block in the image, can be calculated from the block-integral points $(3,3)_b$, $(7,3)_b$, $(3,7)_b$, $(7,7)_b$, with the origin at $(0,0)_b$ in the upper-left corner of the block, according to the following:

$$(r+i, c+i)_i = (i, j)_b + (r-1, c+j)_i + (r+i, c-1)_i - (r-1, c-1)_i \quad (5)$$

This dependence on previous points requires the use of some form of memory. Although it would be possible to refer to the image stored off-chip, this would incur significant delays, making on-chip caching preferable. Since JPEG blocks are read from left to right and top to bottom, it is only necessary to keep the integral points from a single row of the image, in addition to the final column of the previous block.

The astute reader will notice that no mention has yet been

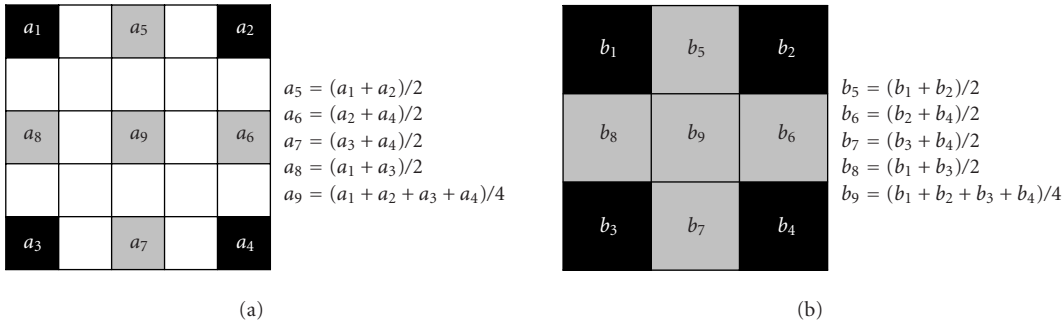

Figure 12: Interpolation scheme for (a) 5×5 and (b) 3×3 neighborhoods.

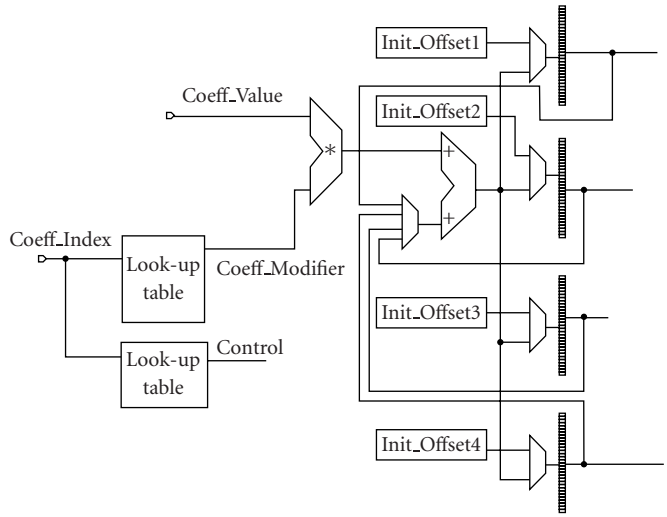

Figure 13: Simplified block diagram of calculation of half-block integral points.

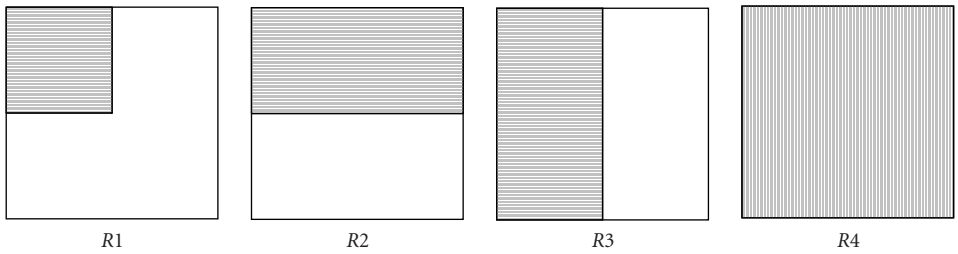

Figure 14: Illustration of integral image being constructed from 8×8 blocks. The grayed-out portions are the blocks for which the integral image has already been calculated, and the black squares are the points which are extracted exactly from the DCT coefficients.

made of the quantization factor required by JPEG decompression. When a JPEG image is encoded, each DCT coefficient is divided by a quantization factor chosen according to its index in the block. The reason that this is not dealt with by the coefficient extractor is that, similarly to the coefficient modifiers, the quantization factor is fixed for a particular index value, and is known in advance, which means that the two multiplicative factors can be combined offline, so that only one multiplication is required online.

In an effort to minimize the memory taken up by the various tables, it is necessary to optimize the number of bits that need to be stored. A close study of the JPEG standard reveals that 14 bits are needed to store all possible coefficient values. Consequently, the largest useful quantization factor also requires 14 bits. Since the coefficient modifiers' absolute values are all less than 3 (excluding the DC modifiers, which are powers of 2, and can be taken care of with shifts), 3 bits are required to store the modifiers' whole parts in 2's complement notation. The number of bits reserved for the fractional part will increase precision, but will not otherwise

limit the range, and is therefore temporarily left undefined. The combined "quantized" modifier consequently requires 17 bits to represent its whole part. Since multiplication of the quantized coefficient by the quantization factor simply restores the original 14-bit coefficient, the final result should also fit in 17 bits for a properly encoded image. These values are then accumulated to give a 32-bit integer, the value of the integral image at that point. Since the result is expected to be an integer, the fractional part is only useful in intermediate results, and rounding off to the closest integer according to the MSB of the fractional part should be enough to correct for any lack of precision in intermediate calculations, as long as the accumulated error in the block is under 0.5.

Calculating the integral image directly from the JPEG coefficients has the obvious advantage of eliminating the need for an explicit integrator. In fact, calculating the integral image directly is equivalent to decompressing the image. One might wonder why linear interpolation is used instead of simply storing a smaller image, since the images are essentially equivalent. Although a high-resolution image is not required by this algorithm to detect people, the features will be misaligned at large scales unless they are placed at what is essentially subpixel resolution at small scales. The method that was chosen to achieve this was to duplicate pixels to allow more precise placement of features. Although this could have been achieved by fully decompressing a smaller image, it was evaluated that the bottleneck was more likely to be in storing the image to memory rather than in receiving the compressed data. A tradeoff can be achieved between the size of the input stream and the complexity of the on-chip decompresser. This is due to the observation that JPEG decompression does not scale linearly with the resolution. While a full-resolution decompression would require 64 accumulators, one for each pixel in the block, a (1/4)-resolution scan only requires 4 accumulators, or 1/16th of that needed for the full resolution. To give a feel for the amount of resources saved by this method, the module calculating the 4 exact points takes up 400 slices in a Virtex-II FPGA (each slice contains 2 flip-flops and 2 four-input lookup tables). The modules approximating the remaining 60 points take up collectively less than 100 slices. Even by limiting estimates to the storage space required for the DCT coefficients' accumulators, calculating the exact values of the 60 remaining integral-image points would require more than taking 960 slices. This would have severe impacts on both placement and routing efforts for the entire module, possibly resulting in a reduced minimum period.

7.1. Putting it all together

Once the various modules have been designed, a method still needs to be chosen to allow them to communicate. In an attempt to maximize flexibility and code reuse, a single, common interface protocol was required for all the modules so that they could be swapped in and out easily without affecting adjoining modules. And, of course, this must be achieved while minimizing the impact on system performance.

The transport layer chosen to satisfy these various constraints was the fast simplex link (FSL) unidirectional point-to-point bus protocol already used by Xilinx for communication with its MicroBlaze processor. This protocol essentially boils down to a 33-bit wide (32 bits of data and 1 of control) first-in first-out (FIFO) buffer with all of the traditionally associated synchronization flags. Using this protocol has many advantages, namely, that any module using it can be plugged directly into the processor, is very low overhead, and most of the more complex protocols already use FIFOs for synchronization of the different modules, making it a simple matter to adapt them to simulate the FSL bus on one end. Figure 15 shows the layout of the different modules in the data path, as well as the flow of data through them. The JPEG stream is received by the MicroBlaze and sent to the JPEG decoding module after the header has been stripped off. This stream is decoded, and the nonzero JPEG coefficients are sent to the integral-image module. The integral image points are then sent to the memory controller through the person detection module. Once the integral image is received, the person detection algorithm is started, and the size and coordinates of the bounding boxes for any positive matches are sent back to the MicroBlaze. Once the image has been scanned at all scales, the MicroBlaze is informed so that it can start over the process.

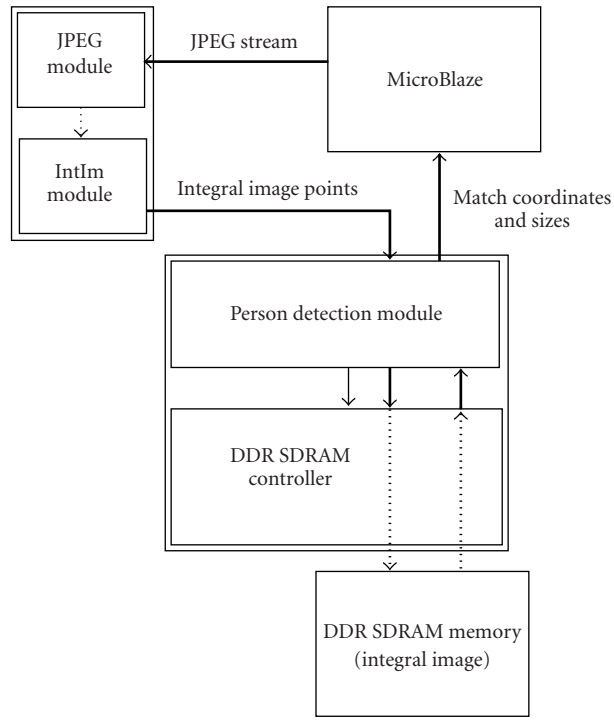

Figure 15: Flow of data through functional modules—bold arrows are FSL bus connections, thin arrows are single control lines, and bold dashed arrows are module-specific interconnections.

8. PEOPLE DETECTION RESULTS

Now we present the results of training the cascaded people detector using AdaBoost. We also give the results of evaluat-

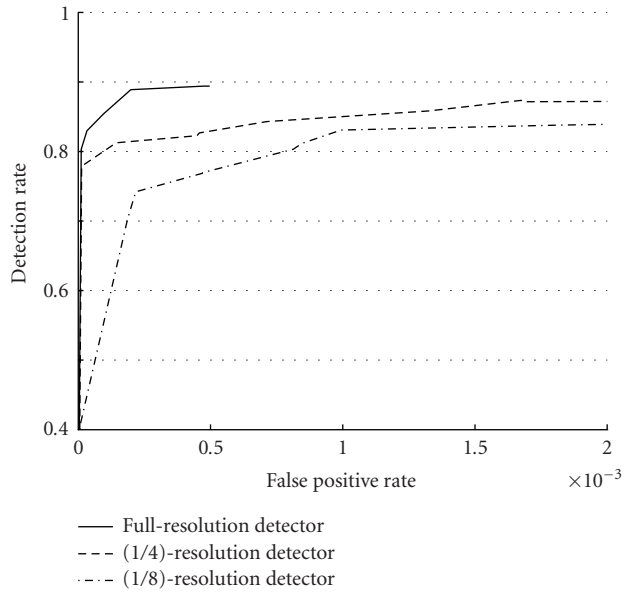

Figure 16: Receiver operating characteristics for detectors at varying integral image resolutions.

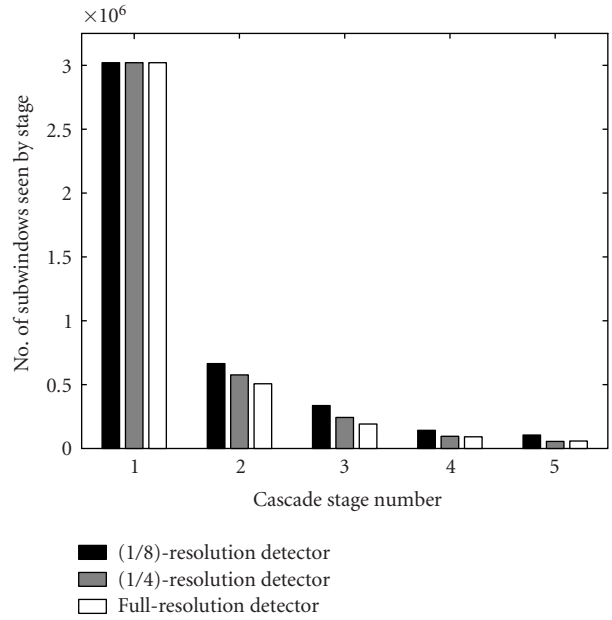

Figure 17: Number of subwindows seen by each stage of the cascade.

ing the accuracy of the trained detector on a test set.

8.1. Training the detector

We use 2252 people instances and the same number of nonpeople instances to train the first stage of the cascaded classifier. Each subsequent stage is trained with the same 2252 people images and 2252 false positives of the previous stages collected from a set of 1500 frames. The number of cascade stages and the size of each stage (i.e., the number of local image features in each stage) are determined automatically with a validation set containing 1585 people instances and 2 901 421 nonpeople images. Because of on-chip memory restrictions, we constrain the cascade construction to a maximum of 5 stages with the maximum sizes of (in layer order) 20, 50, 100, 250, and 500. These values are found empirically to be sufficient for constructing a reasonably accurate and fast classifier. The details of the cascade construction algorithm can be found in [2]. When using (1/4)-resolution level for approximating the integral image, the training algorithm generates a 5-stage cascade classifier with stage sizes 20, 50, 98, 205, and 310, for a total of 683 features.

8.2. Test results

To evaluate the accuracy of the trained detector, we test it on a set of frames containing 981 people instances and 1 246 644 nonpeople instances. (These instances are not used either for training or validation.) The receiver operating characteristic (ROC) of the detector on this test set is shown in Figure 16. For comparison, we also show the ROC for detectors trained on exact integral images and (1/8)-resolution approximate integral images. (Note that for each detector, the same resolution is used in approximating the integral images during both training and testing.) Clearly, the ROC improves as the approximation error of the integral image decreases. So there is a tradeoff between the accuracy of the detector and the computational effort needed for approximating the integral image.

It is also important to compare the average number of features computed per subwindow by each detector. This is because the computational savings obtained by approximating the integral image may be lost if the approximation error causes the detector to be computationally more expensive during detection. On the test set, the full-resolution detector computes 40.48 features per subwindow, the (1/4)-resolution detector computes 40.43 features per subwindow, while the (1/8)-resolution detector computes 47.94 features per subwindow. So the (1/4)-resolution and the full-resolution detectors require approximately the same amount of computation during detection. However, the (1/8)-resolution detector needs more computations because the subwindows get past more of its cascade stages, as shown by Figure 17. Therefore, the savings provided by the (1/8)-resolution approximate integral image are lost during the detection process.

The benefit of the cascade structure can be seen from Figure 17. For all three detectors, the first stage is able to remove almost 75% of the subwindows even though it contains only 20 or fewer features. Only about 1.8% of all subwindows reach the last stage of the full-resolution and (1/4)-resolution detectors, while about 3.5% of subwindows reach the last stage of the (1/8)-resolution detector. As these results show, the cascade structure allows the computational effort expended on a subwindow to be determined by how difficult the subwindow is to classify. Since a vast majority of the subwindows can be classified easily, this strategy results in significantly more efficient detection compared to using one large single-stage detector.

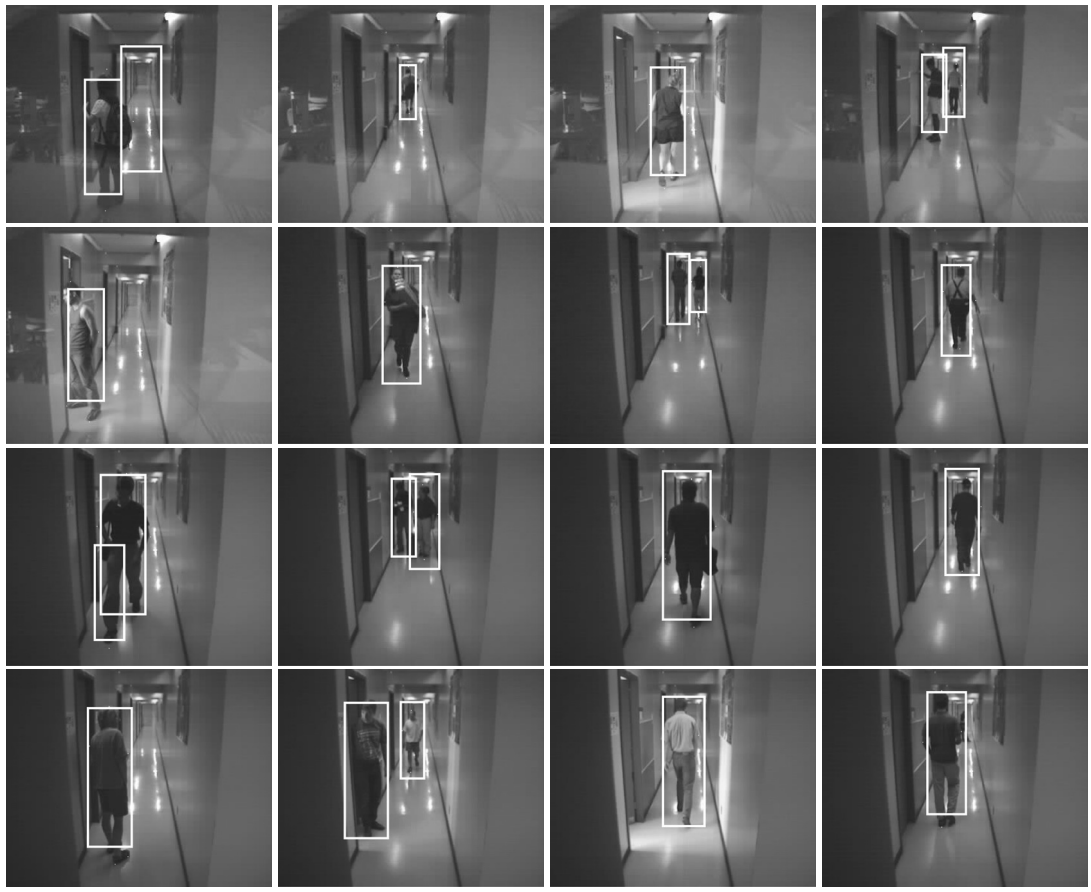

Figure 18: Examples of frames scanned with the (1/4)-resolution detector.

Figure 18 shows examples of frames scanned using the (1/4)-resolution detector. Since the detector is insensitive to slight shifts and small size differences in a people instance, it almost always detects a single people instance multiple times during scanning. But many false positives tend to be isolated detections. So isolated detections are ignored and highly overlapping detections are averaged to obtain a single detection window.

9. FIRMWARE PERFORMANCE ANALYSIS

Given the primitive interfaces available for board-computer communications in the prototyping setup, extensive tests could not be performed on the physical implementation of this algorithm. However, a detailed analysis of the algorithms accuracy was extracted from the full software implementation, as shown in Section 8. Given the functional equivalence between these two approaches, the analysis found therein should hold true for the hardware implementation as well. Exact timing numbers are equally difficult to extract from the physical chip. However, synthesis tools are very proficient at estimating the internal delays with greater precision than might even be achieved through direct measurement. Combining these estimates with functional models and extrapolating using the statistical distributions observed in the software implementation, it is possible to get an accurate measure of what the system's average performance should be under various circumstances.

Since the design was made in a fully synchronous manner, it should be sufficient to know whether or not all timing requirements have been met to know whether the functional simulation is an accurate reflection of the actual implementation. This allows us to use a functional model of the memory provided with the memory controller to get a cycle-accurate functional simulation. This is necessary since the exact time taken in retrieving data from a given address depends on many memory-dependent factors. Using an accurate functional module for the memory allows these delays to be taken into account in functional simulations, and thus allows accurate timing estimates to be extracted from these simulations. The average memory access time was calculated by observing a large number of memory accesses during the person detection module's normal operation. The mean access time per point was then calculated by dividing the total number of points fetched from memory by the total time that it took to fetch these points. This spreads the memory overhead over the entire operation, giving a more accurate estimate than using local measures. It is important to note that memory throughput could be further optimized by customizing the memory controller to take advantage of natural

pauses in memory accesses to refresh or activate the memory banks in view of future accesses. This would have the effect of lowering the average memory access time, thereby increasing the overall performance.

The creation and storage of the integral image to memory for a 352×288 image, cropped to 216×288 by the hardware, is approximately 30 milliseconds. This is governed by the number of points written to memory, which is fixed from one image to the next, and therefore should be relatively constant. Evaluation of the average frame rate of the detector is complicated by the fact that the number of points that need to be calculated varies according to the number of near-people windows in the image. Using a sample space of 981 frames containing people, it is found that on average, 40.43 features are required in each of the 3079 windows of an image. Assuming an average of 7 points per feature, this means that there are approximately 870 000 points evaluated in an average frame. Given a memory access time hovering around 350 nanoseconds per point, it can be estimated that frames containing people can be treated at the rate of approximately 1 frame every 0.3 seconds. In comparison, treating an image that has very few false positives can take as little as 25 milliseconds once the integral image has been written to memory.

Of course, this is assuming that the memory controller is able to run at 100 MHz. Although this should be possible, recent versions of the synthesis tools have unexpectedly been unable to meet the timing constraints at such speeds. This forces the use of the on-chip digital clock managers (DCM) to synthesize a slower clock. Although the DCMs allow synthesis of the most rational number multiples of the input clock, 75 MHz is sufficiently slow and reduces complications in crossing clock domains. With the design running at this speed, the frame rate with a subject in the image should drop from around 3 fps to slightly under 2.5 fps. This constraint makes running the design at full speed less of an issue. This turns out to be quite useful, as the synthesis tools were not quite able to meet the 10-nanoseconds period requirement, even by using the highest effort level for placement and routing, due to some paths of excessive length in the MicroBlaze processor. While careful floorplanning should make it possible to run the design at 100 MHz, the memory controller's speed limitations make this not worth the effort, especially considering that it would be significantly more trouble.

Synthesizing the design without the MicroBlaze processor (leaving only the framework's ASFW) reveals that the 100 MHz constraint could be satisfied if the MicroBlaze were replaced by an off-chip processor. Given that the place and route tools abandon their search for greater performance once the requirements are met, it is possible that this design could run slightly faster still, but the trouble that the tools had in achieving even this level of performance hint that this would probably not be a significant gain. The fact that system performance is limited by memory bandwidth makes any possible speed optimizations unnecessary.

The place and route reports also show that the system has some space remaining for extra logic, with the design only taking up 3438, or 67%, of all slices in the Virtex-II 2V1000 chip. In fact, excluding the MicroBlaze, the full system only takes 2233, or 43%, of the available space. A more significant difference is in the usage of Block Select RAMs, and hardware multipliers. Although the ASFW only uses 7 blocks of memory and 9 multipliers, the MicroBlaze requires an extra 18 blocks of memory, and an extra 3 multipliers. This means that while the ASFW accounts for 65% of the logic used by the entire system, the MicroBlaze accounts for 72% of the memory used.

10. CONCLUSIONS AND FUTURE WORK

In this paper, we have designed an FPGA-based people detection system based on the Viola-Jones object detection algorithm. We have introduced a novel algorithm for computing an approximate integral image from DCT coefficients that is suitable for hardware implementation. Our work has explored some of the hardware issues involved in implementing our system on FPGA. We have developed methods for adapting algorithms which make use of floating-point operations and which require access to large amounts of data. Dealing with such obstacles is a necessary step in adapting the more complex, and more interesting, computer vision algorithms to FPGAs. We have also shown that a relatively simple platform suitable for widespread, low-cost distribution into a network configuration can handle image processing tasks of moderate complexity with a low latency.

The current iteration of this project does not have a hardware training module. However, given that the FPGA-based detector will be operating on the same images as the software one, the training can be done in software and the results loaded into hardware. Since the training data is currently hardwired into the HDL code, it is necessary to resynthesize the code whenever a new training set needs to be loaded. However, it is a relatively simple matter to isolate the parts of the bitstream that correspond to this data and modify them. This allows the creation of a partial reconfiguration bitstream which only modifies the memory locations containing training data and leaves the rest of the FPGA untouched. In fact, it should be possible to use the MicroBlaze itself to reconfigure these sections using the Virtex-II's internal configuration access port (ICAP). For memory segments that only ever need to be changed in their entirety, or that are seldom modified, partial reconfiguration of the memory segments permits the read/write capabilities of RAM without the added complexity of providing a datapath and control for writing to that memory.

Partial reconfiguration also offers interesting possibilities in view of online training of the detector. A camera module could be configured to gather training data to establish or refine the features that are needed by the detector. These feature points can then be stored in the same format and physical location as will be used by the detector itself. When training is complete, the FPGA can be partially reconfigured to replace the trainer with the detector, but leaving intact the data written by the trainer. The detector will then automatically be using the new training data, without the need to ex-

plicitly load it. However, this requires a judicious use of hard macros in both modules, and may lead to suboptimal place-and-route in one or both modules. Since real-time training is not required, suboptimal performance in the trainer can be accepted, which suggests that the detector be optimized first, and the trainer be implemented according to the restrictions this imposes.

More immediate gains could be achieved through development of a caching module to reduce the need for external memory accesses, directly improving system performance. This is aided by the possibility of developing application specific cache architectures to take full advantage of a task's specific memory access patterns, thereby further taking advantage of the FPGA's reconfigurability. However, whether or not individual nodes are working at peak efficiency, it is unfeasible that large numbers of cameras in widely disparate environments could be hand trained. It would be preferable to have some method allowing unsupervised, automatic training of the cameras; and so we come full circle to that which guided the design of the node's architecture, which is the inclusion of this module into a networked environment, working hand in hand on more complex tasks that no module could tackle independently.

REFERENCES

[1] K. Nguyen, G. Yeung, S. Ghiasi, and M. Sarrafzadeh, "A general framework for tracking objects in a multi-camera environment," in *Proc. 3rd International Workshop on Digital and Computational Video (DCV '02)*, pp. 200–204, Clearwater, Florida, USA, November 2002.

4 [2] P. Viola and M. Jones, "Robust real-time object detection," in *2nd International Workshop on Statistical and Computational Theories of Vision – Modeling, Learning, Computing, and Sampling*, Vancouver, Canada, July 2001.

[3] E. Cerro-Prada, S. M. Charlwood, and P. B. James-Roxby, "Designing image processing applications using reconfigurable computing," in *7th International Conference on Image Processing and Its Applications*, vol. 1, pp. 450–454, Manchester, UK, July 1999.

[4] N. Srivastava, J. L. Trahan, R. Vaidyanathan, and S. Rai, "Adaptive image filtering using run-time reconfiguration," in *Proc. International Parallel and Distributed Processing Symposium*, pp. 180–186, Nice, France, April 2003.

[5] T. W. Fry and S. Hauck, "Hyperspectral image compression on reconfigurable platforms," in *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '02)*, pp. 251–260, Napa, Calif, USA, April 2002.

[6] A. B. Abdelali, L. Boussaid, A. Mtibaa, and M. Abid, "Run-time reconfiguration for real-time low-level image processing: architecture and algorithm architecture adequation (AAA)," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, pp. 69–73, Hammamet, Tunisia, October 2002.

[7] M. R. Boschetti, A. M. S. Adario, I. S. Silva, and S. Bampi, "Techniques and mechanisms for dynamic reconfiguration in an image processor," in *Proc. 15th Symposium on Integrated Circuits and Systems Design*, pp. 177–182, Porto Alegre, Brazil, September 2002.

5 [8] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th International Conference on Machine Learning*, pp. 148–156, Bari, Italy, July 1996.

[9] J. Miano, *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*, ACM Press. Addison-Wesley Professional, Boston, Mass, USA, 1999.

[10] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Trans. Consumer Electron.*, vol. 38, no. 1, pp. 18–34, 1992.

Vinod Nair received the B.Eng. degree in 2002 and the M.Eng. degree in electrical engineering in 2004, both from McGill University, Montreal, Canada. He is currently pursuing his Ph.D. in computer science at the University of Toronto, Canada. His main research interests are in machine learning and computer vision.

Pierre-Olivier Laprise received the B.Eng. degree in computer engineering in 2001 and the M.Eng. degree in electrical engineering in 2004 from McGill University, Montreal, Quèbec, Canada. His Master's research focused on the application of reconfigurable-computing embedded systems to computer vision. He worked as a Research Assistant to Professor James J. Clark in the Motor Vision Lab of the Centre for Intelligent Machines, McGill University, from September 2001 to June 2004. He was awarded a PRECARN Scholarship in 2002. He currently works as a Junior Product Design Engineer for PMC Sierra, Inc., Montreal, Quèbec, Canada.

James J. Clark is a Professor in the Department of Electrical and Computer Engineering, McGill University, Montreal, which he joined as an Associate Professor in 1996. He is currently an Associate Chairman (acting) of the department. From 1994 till 1996, he was a Visiting Researcher at Nissan Cambridge Basic Research, Cambridge, Massachusetts. From 1985 through 1994, he was a faculty member in the Division of Applied Sciences, Harvard University, first as an Assistant Professor, then as an Associate Professor. He has spent sabbatical leaves at the California Institute of Technology and at the Université de Paris V. He holds a Ph.D. degree in electrical engineering from the University of British Columbia, Vancouver, British Columbia, Canada.