

Article

Optimization and Implementation of Synthetic Basis Feature Descriptor on FPGA

Dah-Jye Lee ^{1,2,*} , Samuel G. Fuller ² and Alexander S. McCown ²

¹ School of Electrical and Computer Engineering, Nanfang College of Sun Yat-sen University, Guangzhou 510970, China

² Department of Electrical and Computer Engineering, Brigham Young University, Provo, UT 84602, USA; sgfuller31@gmail.com (S.G.F.); jamming101@gmail.com (A.S.M.)

* Correspondence: djlee@byu.edu; Tel.: +1-801-422-5923

Received: 25 January 2020; Accepted: 24 February 2020; Published: 27 February 2020



Abstract: Feature detection, description, and matching are crucial steps for many computer vision algorithms. These steps rely on feature descriptors to match image features across sets of images. Previous work has shown that our SYnthetic BASIS (SYBA) feature descriptor can offer superior performance to other binary descriptors. This paper focused on various optimizations and hardware implementation of the newer and optimized version. The hardware implementation on a field-programmable gate array (FPGA) is a high-throughput low-latency solution which is critical for applications such as high-speed object detection and tracking, stereo vision, visual odometry, structure from motion, and optical flow. We compared our solution to other hardware designs of binary descriptors. We demonstrated that our implementation of SYBA as a feature descriptor in hardware offered superior image feature matching performance and used fewer resources than most binary feature descriptor implementations.

Keywords: synthetic basis; FPGA; feature descriptor

1. Introduction

Image processing for humans involves using sight and then mentally breaking down what is seen to give it meaning. The human visual system can easily distinguish and recognize individual components and quickly make deductions about them based upon prior experience. For computer vision engineers, the task is to teach a computer to extract some desired meaning or information from an image or series of images as well. These algorithms often involve computationally intensive tasks such as object identification [1,2], localization and pose estimation [3], optical flow [4,5], super-resolution [6], visual odometry [7,8], target tracking [9], 3D reconstruction [10], and many others. The primary process to implement these algorithms is to find features in an image and then match them to corresponding features in another image. This process generally follows three steps: feature detection, feature description, and feature matching.

Feature detection is the process of identifying so-called “interesting” parts of an image. An image feature is generally something trackable, distinguishable, and hopefully unique. Thus, image features typically consist of corners, blobs, non-straight edges, or other ridges. Feature detection is often referred to as key-point detection. After feature detection, a feature description is generated around a feature region of interest or FRI. Feature description seeks to describe the FRI in a unique enough way to increase the probability that the features across images will match correctly. Ideally, a feature description is resistant to image deformations and changes such as noise, illumination, perspective, rotation, scale, blurriness, JPEG or other compression artifacts, and more. There are many different methods to perform feature description and these generally offer trade-offs between accuracy and

run-time performance. Each method must be efficient enough to allow for a large number of features to be compared in a relatively short time for real-time applications. This is even more difficult for low power and low resource embedded systems which are becoming increasingly prevalent in computer vision applications. The implementation of these algorithms on platforms such as field-programmable gate arrays (FPGAs) and embedded processors deserve special consideration.

This paper explores and expands upon our feature description and matching algorithm called a synthetic basis feature descriptor (SYBA) [11–13]. SYBA was designed to be hardware-friendly by eliminating the need for operators that are generally costly to implement into FPGAs or other hardware platforms. SYBA does not make use of multiplication, square roots, or trigonometry, which is common for other intensity-based feature descriptors. SYBA can be implemented with only basic hardware functions such as adders, comparators, and simple logic gates (primarily NOR). In addition to being more hardware friendly, SYBA was designed to be efficient in general by reducing the complexity of feature descriptions and computations, as well as reducing the storage costs for storing the descriptors [11]. In this paper, the following accomplishments are detailed: the first-ever hardware implementation of SYBA, key optimizations to SYBA to make it smaller and faster, hardware optimizations that are competitive with other hardware implementations of feature descriptors, and the hardware implementation of this optimized version of SYBA onto an XCZ7020 FPGA.

1.1. Review of Existing Algorithms for Feature Detection, Description, and Matching

The most widely used and prominent algorithms for feature detection, description, and matching are the scale-invariant feature transform (SIFT) [14] and the speeded-up robust features (SURF) algorithms [15]. SIFT is well-known and uses orientation and a magnitudes-of-intensity gradient-based feature descriptor. It works very well on intensity images and provides feature descriptions that are invariant to both rotation and scaling. This increase in robustness and complexity comes at the cost of higher computation and storage requirements rendering it unsuitable for many resource-constrained applications. SURF is used more commonly, as it relies on integral images that help cut down its execution time. However, it has a relatively large storage requirement (256 Bytes) and a high computational cost.

More recently, binary feature descriptors have been developed that have more compact sizes and lower computational requirements. These generally compute the descriptor with pixel-level intensity comparisons. Among the most common of these are binary robust independent elementary features (BRIEF) [16], binary robust invariant scalable keypoints (BRISK) [17], oriented fast and rotated brief (ORB) [18], and Fast RETinA Keypoint (FREAK) [19]. These descriptors trade reliability and robustness for computational speed. BRIEF consists of a binary string, which is the result of multiple intensity comparisons within a single image at random but predetermined locations. A newer version of BRIEF called rBRIEF has been developed by Rublee et al. [18]. rBRIEF uses learned pixel pairs rather than random locations to reduce the correlation among the binary tests. Like BRIEF, rBRIEF requires only 32 bytes to represent a feature point. BRISK relies on configurable concentric circles with more points on the outer rings for its sampling patterns from which it also computes brightness comparisons. It computes the orientation of the keypoint using local gradients between the sampling pairs within the pattern that allows BRISK to be rotation invariant. Overall, this requires significantly more computation (including division and multiplication) and slightly more storage space than BRIEF.

To address this issue, ORB was developed to maintain BRIEF's low computational complexity but maintain rotational invariance. ORB uses a set of 256 learned pixel pairs and only requires 32 bytes to represent a feature point. The ORB descriptor also includes orientation information, which helps make it rotation invariant. While ORB has this significant advantage over BRIEF for rotated images, BRIEF tends to outperform ORB in other cases [20]. Finally, FREAK uses a sampling pattern that is inspired by a retinal sampling grid. This means that there is higher density of points around the center of the sampling pattern. It is similar to the human vision system where peripheral vision is blurry compared to what is seen when looking straight ahead. FREAK uses an orientation assignment similar to BRISK's that allows it to be rotation invariant.

Created at Brigham Young University's Robotic Vision Lab, SYBA offers an alternative to other binary feature descriptors. SYBA is also a binary feature descriptor; however, it is formed in an entirely different way and is inspired by compressed sensing theory. Compressed sensing theory uses synthetic basis functions to uniquely encode or reconstruct a signal. SYBA works by performing several similarity tests between a feature region of interest (FRI) and a predetermined number of synthetic basis images (SBIs) [11]. By only storing the similarity of the FRI to each SBI, the overall storage size is reduced dramatically. It also makes comparisons when searching for feature matches easier.

In short, the SYBA descriptor is designed to provide real-time vision applications high feature matching accuracy with computational simplicity, relatively low resource requirements, and a hardware friendly design. SYBA has previously been compared with two well-known binary descriptors, BRIEF-32 and rBRIEF, and has been shown to produce better feature matching results [11]. Figure 1 shows a summarized version of work previously performed in Reference [11], which compares SYBA to various other feature descriptors. This work was performed on the academically common Oxford dataset. In general, for minor rotation and scale variations, SYBA performed better than other methods. SURF performed the best for the Graffiti sequence in which significant perspective changes cause severe rotation and scale variations. Previous work also includes several applications of SYBA based on a software implementation [13], including visual odometry drift reduction [21].

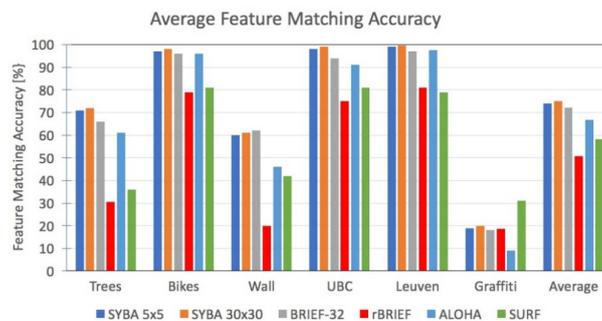


Figure 1. Comparison of SYNthetic BASIS (SYBA) to other feature descriptors. This chart is a summarized version of previous work from Reference [11].

1.2. Review of FPGA architectures for Feature Detection and Description

In this section, a review is presented on some of the methods used to do feature detection and description on field-programmable gate arrays. The use of programmable logic instead of software to do image processing enables custom hardware circuitry to accelerate the process. Many of the primary functions for computer vision are inherently parallelizable, including thresholding, convolution, color space conversion, feature detection, feature description, and feature matching. Pixels are generally fed into a hardware system by some means of a pixel clock and associated data. Hardware designs have been made that do image processing with pixel data coming in directly from a camera (including this work), HDMI, ethernet, and off-chip DDR.

The steps for image processing with features in hardware follows the familiar process of feature detection followed by feature description. Whether or not feature matching is included in the implementation varies on the application. The first choice to be made is what feature detector should be used. Some FPGA based systems have employed state of the art feature detection such as SIFT as in References [22] and [23]. As mentioned in the previous section, SIFT requires the use of several functions that are not very hardware friendly. This causes these implementations to have relatively high resource usage and also requires the use of digital signal processing (DSP) slices.

Several implementations have made use of the Harris Corner feature detector to reduce resource usage as in References [24] and [25]. Others have elected to reduce resource usage further and have used the Features from Accelerated Segment Test (FAST) feature detector as in References [26] and [27] since FAST is a simpler feature detector and requires fewer resources to implement.

Following this, a feature descriptor must be chosen, with many FPGA implementations using BRIEF, including References [26] and [27]. The usage of FAST and BRIEF on FPGAs is common because they are generally very hardware friendly. One of the primary drawbacks of these binary feature detectors and descriptors is a lack of scale and rotation invariance, although small changes are tolerable. Thus, these will not have the same accuracy as implementations that make use of SIFT or SURF but will have much lower resource usage.

In comparison to BRIEF, SYBA has been shown to have better feature matching performance as in Figure 1. More detailed comparison results are available from previous works in References [11] and [21]. Like the other binary methods described, SYBA is not scale or rotation invariant but will tolerate small changes. In this work, we show that SYBA can also have significantly less resource usage in FPGAs while maintaining its feature matching accuracy. This makes it a superior implementation to other common binary feature descriptor implementations in hardware.

2. Algorithm

The focus of SYBA is to use synthetic basis images (SBIs) that are overlaid onto the feature region of interest (FRI) and then compared. This comparison generates several hits that are then compared and matched to find image correspondence. This idea was inspired by the concept of compressed sensing theory to uniquely describe the image region and generate the binary descriptor. SYBA also utilizes a unique matching scheme to produce accurate feature matches.

2.1. Compressed Sensing Theory

Compressed sensing theory is used to encode and decode signals efficiently and can reduce bandwidth and storage requirements. This is an advantageous feature for resource-limited systems. It is capable of uniquely describing signals with synthetic basis functions, and therefore, found a suitable application for feature description. The basic idea of synthetic basis functions for compressed sensing is to use random patterns as a guess. For image feature description, these random patterns are simply patterns of black and white pixels being the SBIs. The black pixels are the points which are sampled and compared against the original image. The equation to find the maximum number of different random patterns required is given in Equation (1) [28]. Where N is the number of pixels on the image feature region, K is in the number of random guesses per pattern, and M is the total number of random patterns that are required to accurately encode the signal. M is smallest when $K = N/2$, meaning that SBIs will be 50% black and 50% white. For example, for a 5×5 FRI, $N = 25$, $K = 13$, and $M = 9$. For a 30×30 FRI, $N = 900$, $K = 450$, and $M = 312$.

$$M = C\left(K \ln \frac{N}{K}\right), \quad (1)$$

2.2. SYBA Feature Description

The SYBA algorithm is only used for describing and matching FRIs but cannot be used to detect features in an image. Therefore, a feature detector is needed in conjunction with SYBA to identify FRI's. In the original SYBA work, SURF was used to find feature point locations. This was done since most other literature also used SURF as the feature detector for software-based implementations. This meant that the detected feature points would be the same, and thus, offer a fair comparison to other papers. For feature description and detection systems that are built into hardware, simpler detectors are commonly used. The hardware friendly feature descriptor, features from accelerated segment test (FAST), is commonly used in these systems, and so it was also used in this work and detailed in the hardware implementation section.

Once the feature point has been detected, the FRI must then be binarized based upon the average intensity of the image region. This calculation allows SYBA to be illumination invariant. To do this, an average intensity (g) of the FRI needs to be calculated as

$$g = \frac{\sum_{x,y} I(x,y)}{p}, \tag{2}$$

where $I(x, y)$ is the intensity of each pixel at location (x, y) , and p is the number of pixels. The binary FRI (BFRI) is then generated using the average intensity g .

$$BFRI(x, y) = \begin{cases} 1 & I(x, y) > g \\ 0 & otherwise \end{cases} \tag{3}$$

Since the SBI is an image of the same size as the BFRI, the images can be overlaid for easy comparison. As described above, the number of black pixels should be set to half the region size (rounding up in case of odd dimensions). The images are then compared to generate a similarity value. If each pixel is black, then this is counted as a hit, while any other combination is not a hit. This is equivalent in a hardware setting to a NOR gate. The number of corresponding hits is then counted for each SBI. These are represented as unsigned numbers that are concatenated together to form the descriptor. The length of the descriptor, without considering pixel coordinates, is given by Equation (4).

$$L = N_s \times N_b \times R, \tag{4}$$

where L is the length of the descriptor, N_s is the number of SBIs, N_b is the number of bits needed to represent the maximum number of hits for that SBI, and R is the number of subregions (only used in SYBA 5×5).

For SYBA 30×30 , this means that each of the 312 SBIs can have a maximum of 450 hits. This number of hits can be represented as an unsigned number with nine bits. This means that the final descriptor length will be $312 \times 9 \times 1 = 2808$ bits. SYBA 30×30 has great image feature matching performance, but it comes at a higher cost computationally. To be competitive with other simple feature descriptors, SYBA 5×5 was developed and uses far fewer operations and still offers very good image feature matching performance, as shown in Figure 2. For SYBA 5×5 , only nine SBIs are needed that can each have a maximum of 13 hits. This number of hits can be represented as an unsigned number with 4 bits. This means that the final descriptor length will be $36 \times 9 \times 4 = 1296$ bits.



Figure 2. Diagram showing FRI (feature region of interest) selection and binarization. Followed by the similarity measure with nine example SBIs (synthetic basis images).

It is worth noting that this feature description length is less than the number of bits needed for more advanced feature descriptors such as SIFT and SURF, which use 256 bytes or 2048 bits [11]. BRIEF has several variants with different length descriptors. BRIEF-16 uses 128 bits, BRIEF-32 uses 256 bits,

and BRIEF-64 uses 512 bits. Thus, the SYBA 5×5 descriptor has a larger descriptor size than BRIEF. SYBA has still found value as a competitive feature descriptor as the authors showed that despite being larger, it could produce better feature matching results. As part of the research in this work, it was found that the SYBA descriptor length could be reduced significantly with minimal impact on accuracy. The results and discussion of this are contained in Section 4. For the sake of easy comparison to the numbers listed here, it was found that SYBA 5×5 with only three SBIs offers comparable image feature matching performance and reduces calculations and descriptor length to only 1/3 of the original size. Thus, the SYBA 5×5 descriptor when using only three SBIs needs only 432 bits.

2.3. SYBA Feature Matching

The first step in any feature matching algorithm is to determine the similarity between two feature descriptions. Note that this similarity does not refer to coordinate distance between the feature points in the image space, but rather refers to the similarity between the descriptions themselves. Euclidean distances are often used as comparison metrics for this, but it requires complex operations such as multiplication and square root operators.

More basic distance operations include the hamming distance, L1 norm, and L2 norm. The hamming distance requires only an XOR operation and adders, so it is very computationally simple. The L1 norm requires only adders and an absolute value operation, which can be implemented simply from other basic hardware operations. The L2 norm requires adders, an absolute value operation, and a square multiplier. SYBA makes use of the L1 norm, as it is principally interested in the difference between the numbers of hits for each SBI and only requires simple hardware. The equation for the L1 norm is as follows:

$$d = \sum_{i=1}^n |x_i - y_i| \quad (5)$$

where x_i and y_i represent the number of hits (the unsigned value) for all n comparisons, where n is the total number of SBIs times the number of sub-regions.

The algorithm for feature matching is based on the similarity between two feature descriptors. First, the point-to-point correspondence is determined by comparing each descriptor in the first image to each descriptor in the second image using the L1 norm, as shown. The remaining process is divided into two steps: a two-pass search and a global minimum requirement. The first pass of the first step is to find the feature point in the second image that has the minimum distance to each feature in the first image. The second pass in the two-pass search guarantees that the pair is uniquely matched. Therefore, the second pass is to find the feature point in the first image that has the minimum distance to each feature in the second image. In order for any pair to form a match, the minimum distance for each feature pair from the first pass must also be the same corresponding pair from the second pass. Each feature must be each other's best match for it to be a match. If this is not the case, then it is not considered a match. This is also called cross-checking.

The remaining unmatched feature points are sent to the next matching step. The second step is to apply a global minimum requirement. This finds the minimum distance values between all remaining feature points and looks for one-to-one matches between these feature points that are considered matches as well. This process can be repeated with the remaining unmatched feature points until no minimum can be found or all distances exceed the global minimum distance threshold.

The smaller the distance, the more similar the two features are. The global minimum distance threshold can easily be adjusted to reject feature matches with a distance that is too large. A larger global minimum will return more but lower quality matches, whereas a smaller global minimum will return fewer matches but at a better quality.

3. Optimization and Hardware Implementation

In this section, the hardware implementation of the SYBA algorithm will be discussed and presented. The goal is to create a pipelined implementation that can detect, describe, and match feature

points in real-time using the SYBA algorithm. The hardware design has been written using VHDL and has also been implemented onto an FPGA board with a ZYNQ XCZ7020 SoC. The FPGA board that was used is the Numato Styx, as can be seen in Figure 3. The design assumed that incoming pixels come in every clock cycle. This matches most video and imaging standards and makes it compatible with a variety of formats such as HDMI and VGA. In addition to the aforementioned video transmission formats, many hardware cameras also rely upon a pixel clock to transmit images. In this design implementation, an Omnivision OV5642 camera was used, wherein it transmits pixels into the hardware design with a reference pixel clock. This can also be seen in Figure 3.

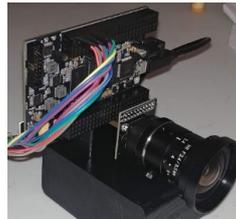


Figure 3. Numato STYX FPGA Board with a ZYNQ XCZ7020 SoC and an OV5642 Camera and lens.

3.1. Graphical Flow of SYBA Hardware

The purpose of this section is to provide a summary of the hardware described in this chapter and to explain the visual. In Figure 4, we can see an overview of the hardware described. In this visual, an Omnivision OV5642 is giving pixel data to the hardware system. These then enter line buffers, which are made via shift-registers in this example. These line buffers were used for both the FAST detector and the binarization region. Within the FAST detector, the continuity test and score calculation are performed, and then put onto additional line buffers.

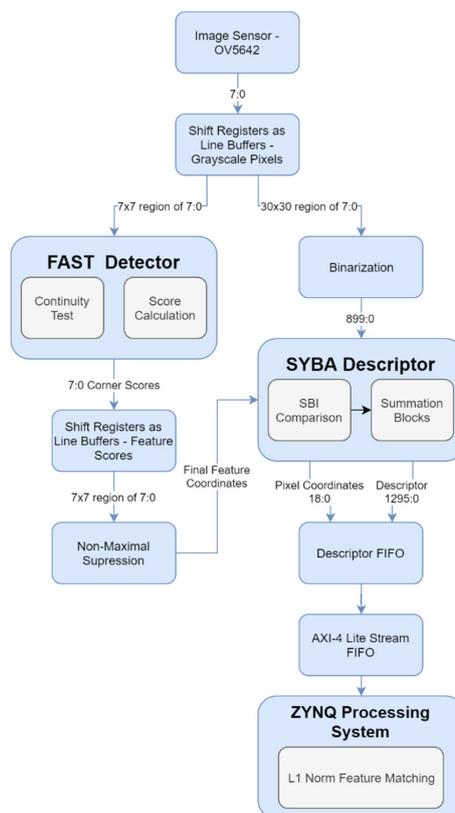


Figure 4. Graphical flow of SYBA (SYnthetic Basis) hardware.

The line buffers are used to suppress non-maximal feature points, and then the output from the non-maximal suppression indicates that a feature is at this point. After some delay (which is determined by the resolution of the image), the SYBA descriptor block takes in the values from the binarized FRI. Then, it is compared to the SBIs, and the hits are counted. The pixel coordinates are concatenated to the end of the descriptor, and this enters a first-in, first-out (FIFO) buffer for descriptor values and coordinates. A state machine reads through these values and converts them into a 32-bit AXI-Stream protocol. These arrive at the ZYNQ processing system which then handles the L1 Norm feature matching.

3.2. Line Buffers

As pixels enter the hardware design, they are converted into an 8-bit grayscale value and entered into a series of line buffers. These are necessary to be able to access regions of pixels simultaneously in a pipelined design. Since SYBA uses a 30×30 FRI, 30 line buffers are needed to access 30 rows simultaneously. Two different hardware designs were tested and implemented. (1) Use shift registers for line buffers, and (2) use BRAMs for line buffers with a small shift register window.

The first design was implemented first and was initially easier to design. Essentially the idea was to declare a large array of size $30 \times X_{res}$ of 8-bit values and shift new pixel values in with the pixel clock. Thus, if the image is VGA resolution at 640×480 , this means that the shift register array is of size $30 \times 640 \times 8$ or $19,200 \times 8$ bits. Then, as new pixels enter the hardware design, subsequent pixels are shifted in as shown in Figure 5.

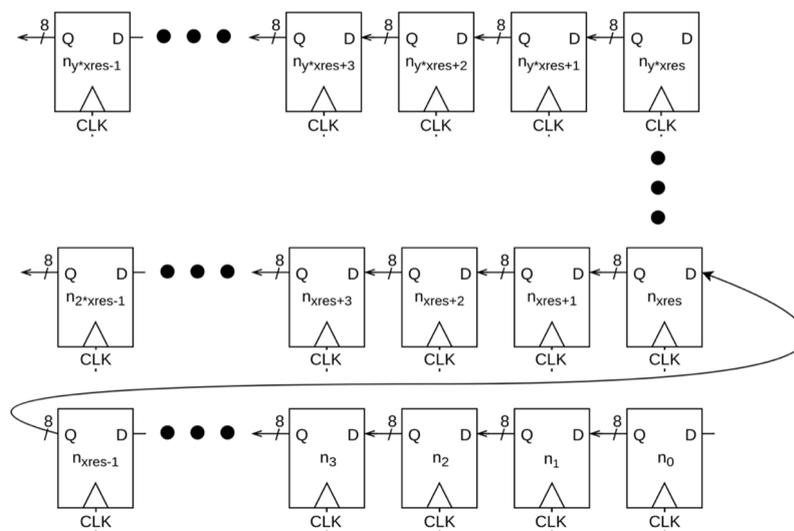


Figure 5. Using shift registers as line buffers.

New pixels are shifted into n_0 and up through all the line buffers. To access pixels in the same row, a simple offset into the array is needed from the incoming pixel. To access pixels in previous rows, an offset corresponding to the X_{res} is used. Setting up the line buffers in this manner allows easy parallel access to any of the needed pixels in the FRI and can also be used for any other accesses needed such as for feature detection. This design has several shortcomings and disadvantages. The first is that in FPGA implementations, this creates relatively high LUT usage. LUTs can be configured as shift registers in FPGAs, so for a typical LUT6 design, a single LUT can hold $2^6 = 64$ values. In the above example, at VGA resolution to create 30 line buffers, a total of $19,200 \times 8/64 = 2400$ LUTs at a bare minimum are needed. Additional LUTs are inserted to access needed pixels, which would be 900 in this case for the FRI. This LUT usage grows as the image size grows, and real-time image processing on 1080p images would require $\sim 3 \times$ as many LUTs, for example. A second minor disadvantage of this

method is that it places much higher strain on CAD tools causing synthesis and implementation to be much larger.

A better design is to take advantage of BRAMs within the FPGA to store incoming pixels. This method dramatically reduces LUT usage within the FPGA, along with significantly reduced strain on the synthesis and implementation tools. The problem is that BRAMs can only have a maximum of 2 ports, a far cry from the minimum 900 parallel accesses needed for SYBA. Our solution was to create shift registers only in the window where parallel access is needed, and place the remainder of the line into a BRAM, as shown in Figure 6. W_{res} is the width of the window in pixel coordinates, y is the height of the window-1.

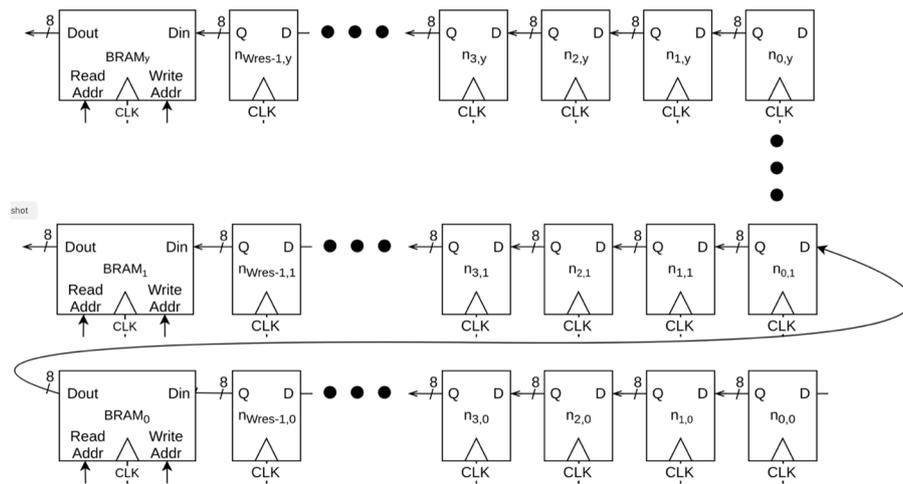


Figure 6. Using shift registers and BRAM as line buffers.

The key to making this design work was to correctly set the read and write addresses to the BRAMs, in order to emulate a line buffer. A simple inspection revealed that these need only operate as simple queues or FIFOs. To achieve this using the native ports of a BRAM, the following steps are needed. First, initialize all memory in the BRAM to be zero. Second, increment the write address on the clock and roll over when the value is $X_{res} - W_{res} - 1$. Third, set the read address to always be the write address + 1 accounting for rollover.

In this design, shift registers were only used in the sections of the line buffers where parallel access was needed. Using the example above, a total of $900 \times 8/64 = 113$ LUTs were needed at a bare minimum. This represented a LUT reduction of about 95%. The tradeoff of course was that a BRAM block was needed for every line. In practice, this is a better tradeoff as the number of BRAMs used is relatively small. Another benefit of this design is that LUT usage does not scale as the resolution increases, since the window size remains the same. Actual implementation resource usage differences between the two methods are shown Section 3.4.

3.3. Feature Detection

As discussed previously, the first step is to detect feature regions of interest and generate a list of features. Any feature detector could be used for this purpose, and in this case, the features from accelerated segment test (FAST) was chosen. FAST is a straightforward and hardware friendly algorithm that was originally proposed by Rosten and Drummond [29] for identifying feature points in an image. It can also be implemented in an FPGA with low resource utilization. It has also been used in various other hardware implementations with BRIEF as a feature descriptor, so for consistency it is also good to compare against.

FAST, like any other feature detector, is interested in identifying if certain points of an image are of interest or not. These are generally centered corner points, and FAST is no exception. FAST works by selecting a candidate center pixel P and a threshold T . The intensity of this pixel, I_p , is compared

with a circle of 16 pixels around the candidate center pixel P . This occurs within a window region of 7×7 pixels, as shown in Figure 7. P is considered to be a corner if there exists a set of n contiguous pixels in the circle that are all brighter than $I_p + T$ or all darker than $I_p - T$. In this implementation, n has been chosen to be $n = 9$. The value n can be adjusted to be more or less exclusive, for example, many implementations of FAST use $n = 12$. In this work, we chose nine to be less exclusive and allow more features points to pass through.



Figure 7. Example of the FAST (features from accelerated segment test) feature detector.

In most software implementations, this detection is made faster by only looking at smaller subsets of the pixel ring to determine if a pixel can be rejected more quickly. In our hardware implementation, all comparisons were done in a single parallel step, and thus, looking at subsets of the pixel ring was not necessary. To access these pixel values, the same line buffers which store the 30×30 FRI window region are used and the location of the 7×7 region within the FRI is adjusted off of the center point to account for any delays in the pipelining and line buffers. This method eliminates the need to store feature point coordinate locations and makes the hardware solution more efficient. The result after performing the comparisons is two 16-bit values B and D , as shown in Figure 8. It is then a simple logic function to find if there exists $n = 9$ contiguous bits that are all HIGH in either B or D . If this is the case, then the pixel centered on the 7×7 region is considered to be a corner point.

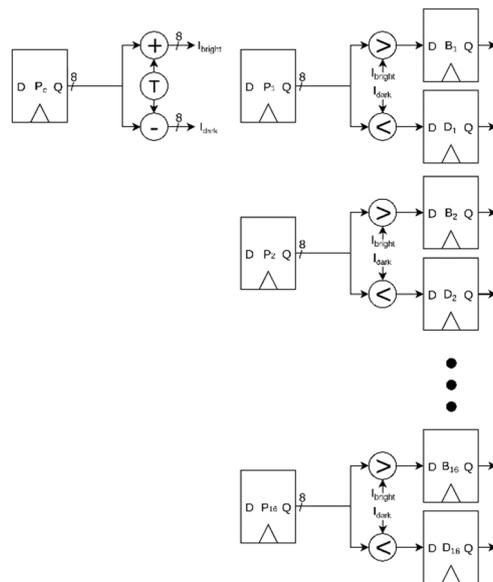


Figure 8. Parallel access and comparison for fast feature detector. P_c is the center pixel. P_n registers are the ring of pixels. B_n registers indicate if the pixel is brighter than the center plus the threshold. D_n registers indicate if the pixel is darker than the center minus the threshold.

The final step needed for feature detection is to add non-maximal suppression. Oftentimes many feature points are found in relatively close proximity to each other, so non-maximal suppression is needed to only keep the best features. In this work, we used a 7×7 window to achieve non-maximal suppression. This meant that at a maximum only one feature point should be in any 7×7 region of the image space. This was achieved using the following steps. First, a score function V was calculated for all the detected feature points. If the pixel was not a detected feature point, then $V = 0$. Otherwise, V is the sum of absolute differences between the center pixel P and the surrounding 16 pixels. Then, this score value was output from the detector and was stored in another seven line buffers when new score values were entered. This allowed for a 7×7 area to be accessed in parallel. Finally, the center pixel was compared to the other 48 pixels in the region and if there was any pixel with a score lower than the center pixel, then these pixels scores were suppressed and set to 0.

The output from the final line of these line buffers of scores is what determines if the point was a good feature point. This meant that the point was detected by FAST and passed all of the thresholds necessary and that it was also the best feature point in its 7×7 region.

3.4. Feature Description

Once a region has been determined to have a good feature, a 30×30 feature region of interest (FRI) is centered around the feature point. The FRI needs to be binarized, and this is done according to Equation (2), which generates the binarized FRI (bFRI) by comparing each pixel value to the average value in the FRI. While the approach inferred from the equation to loop through all pixel coordinates can work well in software, there is a much more efficient way to do this in a pipelined hardware approach. Since each pixel must be passed through the FPGA fabric, we can take advantage of this fact to calculate the average and perform the binarization more efficiently.

This method takes advantage of the fact that we are already storing the incoming pixels into 30 line buffers. The optimization is as follows. First, as a new pixel comes into the image, its column total is found by adding this pixel value to the other 29 pixels directly above this new pixel from the line buffers. The column average is then found by dividing the column total by 30. These column averages are then stored in a shift register that holds 30 of these as unsigned numbers. The area total, for all 900 pixels in the 30×30 area, is then found by adding the incoming column average and subtracting the outgoing column average and then dividing by 30 again. This approach reduces the number of additions from 899 to $29 + 1$ additions and 1 subtraction. This is shown in Figure 9.

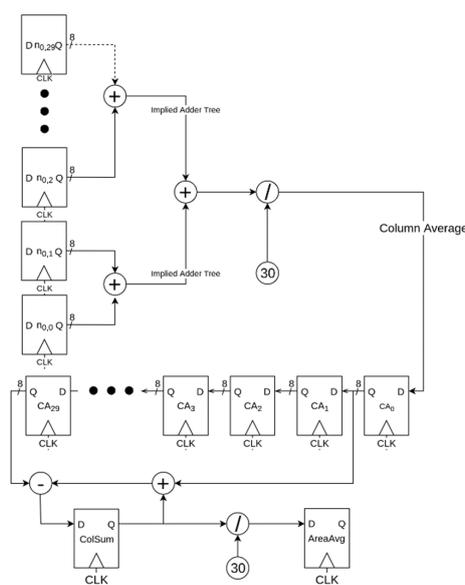


Figure 9. The graphical flow of the efficient averaging finder.

In order to efficiently perform the two divisions by 30 to find the averages, a traditional hardware divider can be substituted with shifts and additions since the divisor is constant. A quick analysis shows that $1/30 \approx 0.03333$. This can be quickly approximated as $1/30 \approx 1/32 + 1/512 = 0.03320$. Performing the division in this manner only costs a single adder, since performing shifts in hardware are as simple as rearranging the bits that are being used.

Now that the area average has been calculated, binarizing the entire FRI is straightforward. All 900 pixels in the FRI are compared to the area average as in Equation (2). Since these are all performed every clock cycle, this necessitates the need for 900 8-bit comparators. In Section 4, a modification to SYBA is explored to reduce the number of resources needed.

Since the resulting binary FRI (BFRI) is relatively small (900 bits = 113 bytes), it is stored in registers and then sent to a module to calculate the unique SYBA similarity measure (SSM). As explained in the previous chapter, this is used to measure the similarity between the FRI and the synthetic basis images (SBIs). The SSM is made by overlaying each SBI with the BFRI and counting the number of matching black pixels, which is implemented easily in hardware as a series of nor gates and adders. In the original work of SYBA, two versions exist denoted as SYBA 5×5 and SYBA 30×30 . SYBA 30×30 compares the entire 30×30 BFRI with 312 30×30 SBIs to form the SSM, whereas SYBA 5×5 breaks the 30×30 BFRI into 36 5×5 subsections and compares each of these to 9 5×5 SBIs. In general, the SYBA 30×30 offers slightly better performance at the cost of more memory and resources. In this implementation, SYBA 5×5 was chosen as being more balanced and practical for hardware implementation. This meant the total descriptor length is $36 \times 9 \times 4 = 1296$ bits = 256 Bytes, as given by Equation (4). This is for the 36 sub regions, 9 SBIs per sub region, and 4 bits per SBI. At this point, the descriptor is finalized and ready to be saved or matched.

3.5. Feature Matching

Feature matching can be achieved with a variety of different methods and algorithms. Often the method of choice is based upon the application be it stereo vision, visual odometry, 3D-reconstruction, optical flow, or any other computer vision application. Given the variety of potential user applications, and since SYBA is a feature descriptor only, the decision is made to transition the feature descriptions to the software side of the ZYNQ. The choice to transition from hardware to software at this point is also supported by (1) the vastly reduced data and bandwidth at this point, and (2) the ease of programming and modularity with software.

Compared to the raw incoming images, just transferring the feature coordinates and descriptors reduces the data rate by more than 2 orders of magnitude, to a level that is achievable for embedded processors. Writing software is also typically much more rapid and modular than designing hardware. Programmers are able to leverage many common libraries such as OpenCV to perform common algorithms. This also allows for easy connection to other peripherals and makes the device much more practical, which enables the device as a whole to interface to other systems easily.

In order to hand off the SYBA descriptors from the programmable logic (PL) to the processor system (PS), an AXI-Stream FIFO is instantiated. This block receives data on a 32-bit bus and is synchronized with valid, ready, and last handshakes. In order to send the 1,296-bit descriptor, a small state machine is made to send and shift over the correct piece of the description. Once the description has been sent over the AXI-Stream, the final 32-bit value represents the pixel coordinate location of the feature. The last handshaking signal is asserted to indicate end of feature descriptor transmission. On the software side, the AXI-Stream is simply a memory-mapped component, and the received data is saved into the DDR feature matches. From there, feature distances are calculated with the L1 Norm as in Equation (5). With feature coordinates and distances in the DDR, it is relatively simple to use SYBA in a variety of applications.

3.6. Speed and Resource Utilization

The design presented in this chapter is a pipelined hardware design that is capable of processing a new pixel with every clock cycle. This has been implemented onto a Xilinx Zynq XC7Z020 board and meets timing with the default clock of 100 Mhz. This makes calculating the maximum frame rate for a given image size a straightforward process, as given by Equation (6). Where F_{fps} is the frame rate, f_{clk} is the clock frequency, and T_p is the total number of pixels in the frame.

$$F_{fps} = \frac{f_{clk}}{T_p} \quad (6)$$

This maximum frame-rate analysis only considers the hardware aspect of the design, which in this case, is the feature detector and feature descriptor. The assumption is made that the software side of the chip can keep up with any necessary feature matching algorithms.

Careful analysis of the timing circuitry could allow for faster clock speeds and thus higher potential maximum frame rates. Higher frame rates can also be achieved with higher-end FPGAs that are built on newer transistor processes and will also achieve higher clock speeds. The speed of image processing usually is highly important for embedded systems that require computer vision. Often, the use of an embedded system is required because the system will be used in an environment that has power, weight, and size constraints, such as unmanned aerial vehicles. Since these systems are generally in motion, a higher image processing speed is desired to be able to react quicker.

The resource utilization for just the feature detector and descriptor is given in Table 1. The table also demonstrates how large of an impact the choice of line buffer style can have on the FPGAs resource utilization. In Section 4, various optimizations are discussed that improve resource utilization. Even without any significant optimizations, the result of creating a hardware implementation of SYBA is an accurate, capable, fast, low power, and low weight solution for embedded vision systems. Table 2 shows the frame rate for different image sizes.

Table 1. SYBA resource utilization for the FAST detector and SYBA descriptor.

Implementation Type	LUT	FF	BRAM
Shift Register Line Buffers	15447	10630	0
Block RAM Line Buffers	10049	9674	18

Table 2. Frame rates for different image sizes at a 100 MHz clock.

Image Resolution	Frames per Second
640 × 480	326
1280 × 720	109
1920 × 1080	48

4. Optimizations

This section discusses the optimizations made for the SYBA algorithm to be more efficient in general and also more hardware efficient. When these optimizations and changes are significant, it is important to have a method to evaluate the impact on the feature descriptor performance of SYBA.

4.1. Accuracy Test and Image Sequences Used

Accuracy tests were performed on the common Oxford image dataset. These image sequences are frequently used in academic literature to test a variety of feature descriptors, including BRIEF, SYBA, and others. These image sequences are used to test the modifications performed to the SYBA descriptor,

and to help understand how the optimizations made impact the descriptor. The image sequences include disturbances such as blurring, lighting variation, viewpoint changes, and image compression.

In order to determine accuracy measurements, OpenCV was used with its Python libraries to accelerate the process. The following steps were used to test the image sequences. First, features were found in each image using the feature from accelerated segment test (FAST) as the feature detector in image I_1 . Detection thresholds were adjusted so that the number of detected features in the first image was about 1000 features. The original SYBA 5×5 descriptor was then calculated for each detected feature. In order to match features to the second image, the dataset provides a homography H that can be used to find the pixel coordinates of the same point in the second image. The homography is used as in Equation (7). Where p_1 is the point in the first image and p_2 is the point in the second image. The SYBA description algorithm was then used around the calculated point p_2 to describe the image feature. The features were then matched with OpenCV's Brute-Force matcher.

$$p_2 = H \times p_1 \quad (7)$$

Distance is calculated using the L1 norm following the SYBA specification. This Brute-Force matcher is very simple, wherein it takes each feature from the first image and compares it with every feature from the second image. The closest feature is then returned and calculated as a match. To increase the accuracy, cross-checking is enabled, which ensures that for every feature in image 1 with its minimum distance pair, that the same pair has the minimum distance for each feature in image 2. Matches that exceed a global minimum threshold requirement are then filtered out. Enabling cross-checking naturally decreases the total number of matches, as some matches are discarded when the minimum distance pairs do not align. The accuracy rate is defined as the ratio of the number of correct matches N_c to the total number of matches found N_f . The first image in every image sequence is always used as the base image, and the following images are compared against.

4.2. Changing the Number of SBIs

One of the largest discoveries in optimizing SYBA, was that when reducing the number of SBIs, the feature matching performance does not alter very much. This of course is an optimization that can be applied at both the software and hardware level. While run time results for the software implementation are not included, anecdotally it is worth mentioning that the run-times were significantly faster in the OpenCV-Python implementation with less SBIs than would be expected. This makes sense since there are fewer comparisons being done between SBIs and FRIs. Additionally, the size of the descriptor also becomes smaller, so calculating the L1 Norm is faster and the entire feature matching process as well.

In order to better quantify the overall feature matching accuracy as the number of SBIs is reduced, the average feature matching accuracy is found for every image pair and sequence. This is shown in Figure 10. It could be seen that with the full number of SBIs ($n = 9$), the average feature matching accuracy rate was 76.12%, and when the total number of SBIs reduced to $n = 3$, the overall feature matching accuracy only dropped slightly to 75.98%. Even when using only a single SBI, the overall feature matching accuracy remained at an impressive 74.44%. This was possible because Equation (1) calculated the number of SBIs to recover the image, which was not necessary for our purposes. We simply used the SBIs to describe the image not to reconstruct it.

What makes this a particularly intriguing optimization is that the descriptor length and calculation complexity depend linearly upon the number of SBIs, as shown in Table 3. This roughly corresponds to the hardware complexity for the descriptor, and so is well worth the optimization.

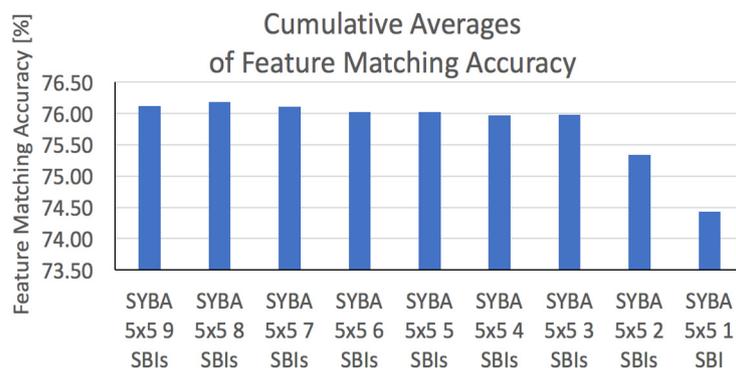


Figure 10. Number of SBIs (synthetic basis images) vs. average feature matching accuracy across all image sequences.

Table 3. Showing linear dependency on number of SBIs for pixel comparisons and descriptor length. Accuracy does not linearly depend on the number of SBIs.

Number of SBIs	Pixel Comparisons	Descriptor Length	Accuracy
9	4212	1296	76.12%
8	3744	1152	76.18%
7	3276	1008	76.11%
6	2808	864	76.02%
5	2340	720	76.02%
4	1872	576	75.97%
3	1404	432	75.98%
2	926	288	75.34%
1	468	144	74.44%

Given these findings, it is the authors' opinion that the best number of SBIs to use is 3. A small decrease in accuracy of less than one-quarter of one percent is well worth it, as logic utilization within the descriptor is made to be about 1/3 the usage, and the descriptor itself becomes 1/3 the length of its previous amount, as given by Equation (4). It has also been noted that matching in software is likewise performed about 3 times as quickly. Further reducing the number of SBIs was considered but rejected based on the relatively larger drop in feature matching accuracy not only in the averages but also in certain revealing image pairs. This is one of the key optimizations that has allowed SYBA to be competitive in hardware with other small binary feature descriptors such as BRIEF. The reduction in resource utilization can be seen in Table 4.

Table 4. Implementation type vs. hardware usage.

Number of SBIs	Binarization	LUT	FF	BRAM 36K
9	Original	10049	9674	18
9	Kernel Optimized	5944	4169	19.5
3	Original	5955	9115	18
3	Kernel Optimized	2966	3419	19.5

4.3. Changing the Method of Binarization

As mentioned previously, the feature region of interest (FRI) must be binarized around the feature point. In the original SYBA work, this was done by calculating the average intensity of the FRI as in

Equation (2). This optimization comes in the way the binarized image is made from the comparison to the average value. In the original SYBA work, all 900 comparisons were made to the average value. In a pipelined hardware design, which has a new incoming pixel every clock cycle, this would mean that 900 8-bit comparators are needed to generate the binarized FRI. In order to reduce the number of comparators needed, a new idea was presented to perform the binarization. The idea was to pass a kernel over the image of size $k \times k$ and find the average value within that kernel. This can be represented by Equation (8). Where k is the kernel size, p is the grayscale value of a pixel, and x and y are the pixel coordinates in the image space.

$$avg_{xy} = \frac{1}{k^2} \sum_{i=x-\frac{k}{2}}^{x+\frac{k}{2}} \sum_{j=y-\frac{k}{2}}^{y+\frac{k}{2}} P_{ij} \quad (8)$$

The pixel in the center of the kernel is then compared to the average value and binarized. This pixel is put onto 30 line buffers, which only hold 1-bit values. The resulting image space is given by Equation (9), where BIMG represents the binarized image space.

$$BIMG(x, y) = \begin{cases} 1 & p_{xy} > avg_{xy} \\ 0 & otherwise \end{cases} \quad (9)$$

A 30×30 window into these line buffers is then used to grab the binarized FRI centered on the detected feature point. From an FPGA hardware perspective, this method has the advantage of only requiring a single comparator at the cost of using additional BRAMs for the binary line buffers. This is an engineering decision tradeoff, but it is well worth it, as can be seen in Table 4 comparing resource usage.

From a computer vision perspective, this represents a significant change to the way the image FRI is binarized. Rather than looking at the average of just the FRI and comparing all pixels to that average, each pixel is now compared to the average of a 30×30 region centered around that individual pixel. This means that pixels near the edges will have a significant area outside the FRI used in the average calculation. In theory this could be good, as it would give the pixel additional “context” compared to other pieces of the image space but could also be harmful as it would no longer be considering the entire FRI as well. Since this represents a significant departure from the method used to binarize the FRIs in the original SYBA implementation, additional study is needed and presented below.

As was done for changing the number of SBIs, the change was implemented and tested in Python and OpenCV, using the framework described above. Multiple kernel sizes were tested, and as would be expected larger kernels performed better than smaller kernels. It was found that a kernel size of 30×30 achieved a good balance between matching accuracy and hardware usage. The impact of these optimizations can be seen in Figure 11.

Analyzing these image sequences shows that in general, for images with only minor perturbations to viewpoint along with blurring, lighting changes, or jpeg compression, the binarization method either yields better or unchanged matching performance. For images with major viewpoint variations, this binarization method results in poorer matching performance. Although SYBA has already been extensively compared to BRIEF-32 and other descriptors in Reference [11], another comparison is offered in this figure. In this accuracy test, the sampling pattern size for BRIEF was reduced to 33×33 pixels to match the hardware implementations as in Reference [26]. Matching distance thresholds are then adjusted so that each descriptor has a roughly equal number of correspondences for each image pair to maximize fairness. This shows that SYBA still performs favorably in comparison to BRIEF-32 even with the changes and hardware optimizations, and a different feature detector (FAST).

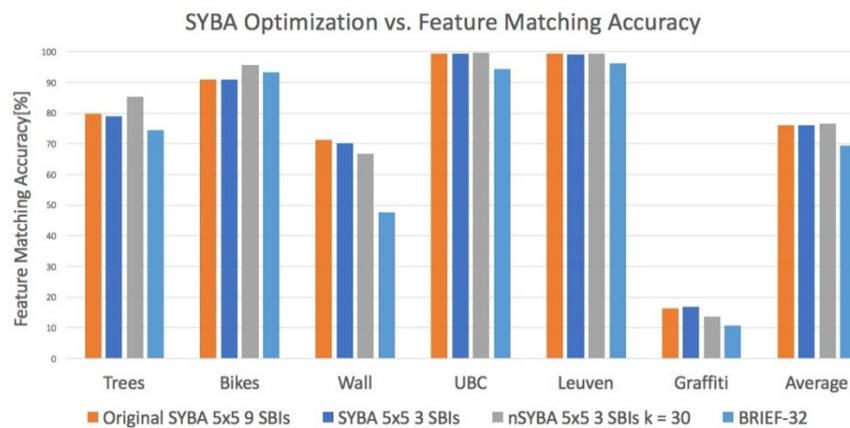


Figure 11. Different SYBA methods and BRIEF-32 vs. feature matching accuracy.

4.4. Impact of SYBA Optimizations on Hardware Usage

In this section, an overview of how these changes are implemented into the physical hardware is given, along with how these changes have impacted the resource utilization. The change in hardware to use only three SBIs rather than nine SBIs was straightforward. Rather than instantiating 9 SBI comparison modules in VHDL, only 3 SBI comparison modules were needed. Then the descriptor lengths were updated accordingly. While the change is very simple, the impact on resource utilization is high. In Table 4, the resource usage reduction can be seen. Compared to the original implementation, just reducing the number of SBIs from 9 to 3 reduces the number LUTs by 41% (10,049 to 5955) and the number of FFs by 9% (9674 to 9115).

The second change is a little bit more involved. As described above, an additional 30 line buffers are needed to store the output result from comparisons of the center pixel to the average of the 30×30 kernel. These line buffers use less BRAM resources since they are only 1 bit wide instead of 8 bits. The location of the FAST feature detector within the 30×30 FRI is further adjusted to account for the extra processing delay. The resource reduction for this method is also substantial. Compared to the original solution, LUT usage is decreased by 41% (10,049 to 5944) and FF usage is decreased by 57% (9674 to 4169). This comes at the cost of using another 1.5 36Kb of BRAM blocks.

When these optimizations are both applied to the design, the overall hardware reduction is substantial. LUT usage is decreased by 70.5% (10,049 to 2966) and FF usage is decreased by 64.6% (9674 to 3419). This still comes at the cost of the additional 1.5 36Kb BRAM blocks. These optimizations have been shown to not only reduce hardware usage, but also either improve or maintain feature matching accuracy on images with blurring, lighting, or compression deformations. This new version of SYBA has allowed the hardware implementation to use fewer resources than comparable hardware accelerators of other feature descriptors while maintaining its high feature matching accuracy.

5. Results and Discussion

In this section, we discuss three important factors regarding hardware design for embedded systems. We also compare our design to other feature descriptor implementations on FPGAs.

5.1. Clock Frequency, Latency, and Power Consumption

The maximum clock frequency is determined by a variety of factors including the nature of the hardware design itself and the FPGA device used for implementation. The FPGA device used in this work was a ZYNQ XC7Z020. With a 100MHz clock, all timing constraints are met and so this is what the published results used. The worst path delay is 9.965 ns, so theoretically it could run barely faster (100.3MHz) although not by a significant margin. In theory, the maximum clock frequency could be faster if the design is optimized further or if a different FPGA device is chosen. This, of course, may

impact resource utilization as well. Other clocks in the design include the camera input clock, which is limited to a 27 MHz maximum. The camera's output clock can be scaled to run faster using a built-in phase-locked loop internal to the camera. In Table 2, the maximum FPS is the maximum theoretical framerate achievable at that resolution with the current design pixel clock of 100 MHz. In principle, with a faster camera our entire design would indeed operate at these framerates. With this current camera, the design only ran at 60 FPS because that was the camera's maximum output speed with a 640×480 output resolution.

Latency is by and large, mostly determined by the Kernel size. For example, with a 30×30 kernel size, we will need to buffer 15 lines of 8-bit data (15 lines above are already in and must wait for 15 lines below the pixel to come in and be compared) before performing the averaging. There are another 30 lines of 1-bit line buffers after the averaging and binarization. Thus, there was a total of 45 lines worth of latency for our design. Making the kernel size smaller would decrease the latency, but could also decrease the feature matching accuracy. Other tasks such as suppression of features detected, FAST detection, among others, all occur before the averaging finishes.

Power consumption is another important factor for embedded systems. The Xilinx tool we used (Vivado 2018.3) gave the following power estimates. The total on-chip power was estimated to be 1.958 W. Most of this power (1.363 W or 70%) was going to the onboard CPU processing system (PS7), along with a few blocks of design (0.437 W or 22%) to move the data back and forth. All of the design for FAST + SYBA to detect/describe (not including matching, since that is done in the PS7) was estimated to be only 0.158W or 8% of total power consumption.

5.2. Comparison with Other Implementations

The implementation of feature descriptors in the FPGAs remains an active research topic in recent years. Several works on fully pipelined FPGA accelerators for SIFT have been published since 2016 [30–33]. A parallel hardware architecture for SIFT was also reported in Reference [34]. Recent and improved FPGA implementations for SURF [35–37] and ORB [38,39] were reported. SIFT and SURF implementations were not included in our comparison because they were intensity-based not binary descriptors. Although they perform better than binary descriptors for applications dealing with large perspective transformation, they have relatively high resource usage and require the use of digital signal processing (DSP) slices.

While ORB has this significant advantage over BRIEF for rotated images, BRIEF tends to outperform ORB in other cases [20]. In this paper, we focused on comparing our implementation with BRIEF and BRISK. One recent BRIEF implementation [40] reported the utilization of their entire design, not just feature detection and description. The FPS number is biased as well because their image size was 512×512 , whereas all other papers used 640×480 .

This can be seen in Table 5. All of these designs are using the same hardware chip, which is the Xilinx Zynq XC7Z020, the same clock frequency of 100Mhz, and the same feature detector FAST. Resource utilizations included in the table show the resource usage for only the logic needed for the feature detector and feature descriptor portions of the design. This is to remove any comparison inconsistencies that would arise from implementation inconsistencies outside of these parts of logic. For example, some designs use raw camera input, some HDMI input, and some use images from DDR, so the logic utilization to grab the images from different sources should not be included in the comparison.

It can be seen that the optimized version of SYBA has the lowest LUT usage of any of the included designs, regardless of design speed. The only other design that runs as quickly as this design (326 FPS at 640×480 , 100 MHz clock, 1 pixel per clock) requires 39% more LUTs, 199% more FFs, and 59% more BRAMs. Other designs run at a much slower speed and are therefore not fully pipelined. Some of the resource utilization in these designs is lower, while many resources are still higher such as LUTs. Given the high reduction in capable speeds, these designs were able to decrease resource usage. When considering all of the comparisons as a whole, this shows that SYBA is capable of delivering

excellent feature matching results, while also being very hardware efficient even when compared to other low-resource utilization binary descriptors.

Table 5. Resource utilization of various feature detector and descriptor implementations on FPGAs. These all have several of the same attributes. (1) They all use FAST as the feature detector. (2) They are all using the same hardware, a ZYNQ 7020. (3) They all have the same clock speed, 100 MHz.

Implementation	LUT	FF	BRAM KB	FPS
FAST + Original SYBA	10049	9674	81	326
FAST + Optimized SYBA	2966	3419	88	326
FAST + BRIEF [26]	4118	9543	140	326
FAST + BRIEF [27]	4257	3187	72	55.5
FAST + BRIEF [41]	14398	2093	50	147
FAST + BRISK [41]	25575	7115	50	147

6. Conclusions

The focus of this work is on the improvement of one of the essential aspects of computer vision: image feature description. This paper has improved upon and made a new version of a feature description algorithm called SYBA. The paper began by introducing the SYBA algorithm developed by Alok Desai several years ago. A discussion of the hardware implementation was then developed, along with justification for several design choices. The following section discussed two novel ideas to improve SYBA to make it more efficient. The first idea was to reduce the number of SBIs for comparison, which significantly reduced resource usage and had minimal impact on accuracy. The second idea was to change the binarization method from comparing all pixels in the FRI to an average value, to comparing only the center pixel of a kernel to the average value. This also significantly reduced hardware usage and even improved feature matching accuracy for certain types of image pairs and deformations.

The result of putting SYBA into hardware and optimizing it for better resource utilization is outstanding. It has been shown to use significantly fewer resources than competing feature descriptors while running at very high frame rates. SYBA has also been previously shown to have better feature matching accuracy than these descriptors as in Reference [11]. The combination of these two facts clearly demonstrates that the SYBA descriptor in hardware has great image feature matching performance at low resource utilization.

Author Contributions: Conceptualization, S.G.F. and D.-J.L.; Data curation, S.G.F.; Formal analysis, S.G.F.; Investigation, S.G.F.; Methodology, S.G.F. and D.-J.L.; Project administration, D.-J.L.; Resources, D.-J.L.; Software, S.G.F.; Validation, S.G.F. and A.S.M.; Visualization, S.G.F. and A.S.M. Writing—original draft, S.G.F. and A.S.M.; Writing—review & editing, D.-J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the University Technology Acceleration Program (UTAG) of Utah Science Technology and Research (USTAR) (#172085) of the State of Utah, U.S.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

- Garcia, H.C.; Villalobos, J.R.; Runger, G.C. An automated feature selection method for visual inspection systems. *IEEE Trans. Autom. Sci. Eng.* **2006**, *3*, 394–406. [[CrossRef](#)]
- Piccinini, P.; Prati, A.; Cucchiara, R. Real-time object detection and localization with sift-based clustering. *Image Vis. Comput.* **2012**, *30*, 573–587. [[CrossRef](#)]
- Glasner, D.; Galun, M.; Alpert, S.; Basri, R.; Shakhnarovich, G. Viewpoint-aware object detection and continuous pose estimation. *Image Vis. Comput.* **2012**, *30*, 923–933. [[CrossRef](#)]

4. Wei, Z.Y.; Lee, D.J.; Nelson, B.E. A hardware-friendly adaptive tensor based optical flow algorithm. In Proceedings of the International Symposium on Visual Computing, Lake Tahoe, CA, USA, 26–28 November 2007; pp. 43–51.
5. Doshi, A.; Bors, A.G. Smoothing of optical flow using robustified diffusion kernels. *Image Vis. Comput.* **2010**, *28*, 1575–1589. [[CrossRef](#)]
6. Huang, H.; Wu, N. Fast facial image super-resolution via local linear transformations for resource-limited applications. *IEEE Trans. Circuits Syst. Video Technol.* **2011**, *21*, 1363–1377. [[CrossRef](#)]
7. Civera, J.; Grasa, O.G.; Davison, A.J.; Montiel, J. 1-point RANSAC for extended Kalman filtering: Application to real-time structure from motion and visual odometry. *J. Field Robot.* **2010**, *27*, 609–631. [[CrossRef](#)]
8. Badino, H.; Yamamoto, A.; Kanade, T. Visual odometry by multi-frame feature integration. In Proceedings of the IEEE International Conference on Computer Vision Workshops, Sydney, Australia, 2–8 December 2013; pp. 222–229.
9. Niedfeldt, P.C.; Beard, R.W. Multiple target tracking using recursive RANSAC. In Proceedings of the American Control Conference, Portland, OR, USA, 4–6 June 2014; pp. 3393–3398.
10. Pollefeys, M.; Nister, D.; Frahm, J.M.; Akbarzadeh, A.; Mordohai, P.; Clipp, B.; Engels, C.; Gallup, D.; Kim, S.; Merrell, P.; et al. Detailed real-time urban 3d reconstruction from video. *Int. J. Comput. Vis.* **2008**, *78*, 143–167. [[CrossRef](#)]
11. Desai, A.; Lee, D.J.; Ventura, D. An efficient feature descriptor based on synthetic basis functions and uniqueness matching strategy. *Comput. Vis. Image Underst.* **2016**, *142*, 37–49. [[CrossRef](#)]
12. Raven, L.A.; Lee, D.J.; Desai, A. Robust synthetic basis feature descriptor. In Proceedings of the IEEE International Conference on Image Processing (ICIP), Beijing, China, 11–15 September 2017; pp. 1542–1546.
13. Zhang, D.; Raven, L.A.; Lee, D.J.; Yu, M.; Desai, A. Hardware friendly robust synthetic basis feature descriptor. *Electronics* **2019**, *8*, 847. [[CrossRef](#)]
14. Lowe, D.G. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.* **2004**, *60*, 91–110. [[CrossRef](#)]
15. Bay, H.; Tuytelaars, T.; Van Gool, L. Surf: Speeded up robust features. In Proceedings of the European Conference on Computer Vision, Graz, Austria, 7–13 May 2006; pp. 404–417.
16. Calonder, M.; Lepetit, V.; Strecha, C.; Fua, P. Brief: Binary robust independent elementary features. In Proceedings of the European Conference on Computer Vision, Heraklion, Crete, Greece, 5–11 September 2010; pp. 778–792.
17. Leutenegger, S.; Chli, M.; Siegwart, R. Brisk: Binary robust invariant scalable keypoints. In Proceedings of the IEEE International Conference on Computer Vision, Heraklion, Crete, Greece, 5–11 September 2010; pp. 2548–2555.
18. Rublee, E.; Rabaud, V.; Konolige, K.; Bradski, G.R. Orb: An efficient alternative to sift or surf. In Proceedings of the IEEE International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011; pp. 2564–2571.
19. Alahi, A.; Ortiz, R.; Vandergheynst, P. Freak: Fast retina keypoint. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 16–21 June 2012; pp. 510–517.
20. Heinly, J.; Dunn, E.; Frahm, J.M. Comparative evaluation of binary features. In Proceedings of the European Conference on Computer Vision, Florence, Italy, 7–13 October 2012; pp. 759–773.
21. Desai, A.; Lee, D.J. Visual odometry drift reduction using SYBA descriptor and feature transformation. *IEEE Trans. Intell. Transp. Syst.* **2016**, *17*, 1839–1851. [[CrossRef](#)]
22. Huang, F.C.; Huang, S.Y.; Ker, J.W.; Chen, Y.C. High performance SIFT hardware accelerator for real-time image feature extraction. *IEEE Trans. Circuits Syst. Video Technol.* **2011**, *22*, 340–351. [[CrossRef](#)]
23. Bonato, V.; Marques, E.; Constantinides, G.A. A parallel hardware architecture for scale and rotation invariant feature detection. *IEEE Trans. Circuits Syst. Video Technol.* **2008**, *18*, 1703–1712. [[CrossRef](#)]
24. Amaricai, A.; Gavriliu, C.; Boncalo, O. An FPGA sliding window-based architecture Harris corner detector. In Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Munich, Germany, 2–4 September 2014; pp. 1–4.
25. Aydogdu, M.F.; Demirci, M.F.; Kasnakoglu, C. Pipelining Harris corner detection with a tiny FPGA for a mobile robot. In Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO), Shenzhen, China, 12–14 December 2013; pp. 2177–2184.
26. Fularz, M.; Kraft, M.; Schmidt, A.; Kasinski, A. A high-performance FPGA-based image feature detector and matcher based on the fast and brief algorithms. *Int. J. Adv. Robot. Syst.* **2015**, *12*, 141. [[CrossRef](#)]

27. Heo, H.; Lee, J.Y.; Lee, K.Y.; Lee, C.H. FPGA based implementation of fast and brief algorithm for object recognition. In Proceedings of the IEEE International Conference of IEEE Region 10 (TENCON 2013), Xi'an, China, 22–25 October 2013; pp. 1–4.
28. Anderson, H. *Both Lazy and Efficient: Compressed Sensing and Applications*; Sandia National Laboratories: Albuquerque, NM, USA, 2013; pp. 2013–7521.
29. Rosten, E.; Drummond, T. Machine learning for high-speed corner detection. In Proceedings of the European Conference on Computer Vision, Graz, Austria, 7–13 May 2006; pp. 430–443.
30. Daoud, L.; Latif, M.; Jacinto, H.S.; Rafla, N. A fully pipelined FPGA accelerator for scale invariant feature transform keypoint descriptor matching. *Microprocess. Microsyst.* **2020**, *72*, 102919. [[CrossRef](#)]
31. Li, S.A.; Wang, W.Y.; Pan, W.Z.; Hsu, C.C.; Lu, C.K. FPGA-based hardware design for Scale-Invariant Feature Transform. *IEEE Access* **2018**, *6*, 43850–43864. [[CrossRef](#)]
32. Rekha, S.S.; Pavitra, Y.J.; Prabhakar, M. FPGA implementation of Scale Invariant Feature Transform. In Proceedings of the International Conference on Microelectronics, Computing and Communications (MicroCom), Durgapur, India, 23–25 January 2016; p. 7.
33. Vourvoulakisa, J.; Kalomiroso, J.; Lygourasa, J. Fully pipelined FPGA-based architecture for real-time SIFT extraction. *Microprocess. Microsyst.* **2016**, *40*, 53–73. [[CrossRef](#)]
34. Peng, J.Q.; Liu, Y.H.; Lyu, C.Y.; Li, Y.H.; Zhou, W.G.; Fan, K. FPGA-based parallel hardware architecture for SIFT algorithm. In Proceedings of the IEEE International Conference on Real-time Computing and Robotics (RCAR), Angkor Wat, Cambodia, 6–9 June 2016; pp. 277–282.
35. Zhang, Y.; Huang, Y.; Han, J.; Zeng, X.Y. FPGA-based efficient implementation of SURF algorithm. In Proceedings of the IEEE 12th International Conference on ASIC (ASICON), Guiyang, China, 25–28 October 2017; pp. 315–318.
36. Chen, W.; Ding, S.; Chai, Z.; He, D.; Zhang, W.; Zhang, Z.; Peng, Q.; Luo, W. FPGA-based parallel implementation of SURF algorithm. In Proceedings of the IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, China, 13–16 December 2016; pp. 308–315.
37. Cizek, P.; Faigl, J. Real-time FPGA-based detection of Speeded-Up Robust Features using separable convolution. *IEEE Trans. Ind. Inform.* **2018**, *14*, 1155–1163. [[CrossRef](#)]
38. De Lima, R.; Martinez-Carranza, J.; Morales-Reyes, A.; Cumplido, R. Improving the construction of ORB through FPGA-based acceleration. *Mach. Vis. Appl.* **2017**, *28*, 525–537. [[CrossRef](#)]
39. Fangy, W.; Zhang, Y.; Yuy, B.; Liuy, S. FPGA-based ORB feature extraction for real-Time visual SLAM. In Proceedings of the International Conference on Field Programmable Technology (ICFPT), Melbourne, VIC, Australia, 11–13 December 2017; p. 4.
40. Huang, J.; Zhou, G.; Zhou, X.; Zhang, R. A new FPGA architecture of FAST and BRIEF algorithm for on-board corner detection and matching. *Sensors* **2018**, *18*, 1014. [[CrossRef](#)] [[PubMed](#)]
41. Ulusel, O.; Picardo, C.; Harris, C.B.; Reda, S.; Bahar, R.I. Hardware acceleration of feature detection and description algorithms on low power embedded platforms. In Proceedings of the 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; p. 9.

