J Real-Time Image Proc
DOI 10.1007/s11554-017-0725-0

ORIGINAL RESEARCH PAPER

Data-path unrolling with logic folding for area-time-efficient FPGA-based FAST corner detector

Siew-Kei Lam[1] · Teck Chuan Lim[1] · Meiqing Wu[1] · Bin Cao[1] · Bhavan A. Jasani[1]

Received: 20 April 2017 / Accepted: 2 October 2017

Abstract Corner detection plays an essential role in many computer vision applications, e.g., object recognition, motion analysis and stereo matching. Several hardware implementations of corner detection algorithms have been previously reported to meet the real-time requirements of such applications. However, most of the reported implementations adopt similar computational flow which limit their potential for further area-time optimizations. In this paper, we propose a novel hardware design for the FAST corner detector, which unrolls the data-path to perform partial evaluation of multiple corners in a pipelined manner. We then apply logic folding that maximizes the design regularity of the unrolled data-path for resource sharing of the combinational operations. We show that the proposed design on FPGA leads to 20% reduction in critical path delay and about 39% reduction in area-delay product compared to a previously reported architecture. The real-time capability of the proposed FAST corner detectors is demonstrated on the TERASIC DE2i-150 FPGA development kit.

✉ Siew-Kei Lam
siewkei_lam@pmail.ntu.edu.sg

Teck Chuan Lim
TLIM023@e.ntu.edu.sg

Meiqing Wu
meiqingwu@ntu.edu.sg

Bin Cao
jamescao@ntu.edu.sg

Bhavan A. Jasani
bhavan.jasani@gmail.com

[1] School of Computer Science and Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore, Singapore

Keywords Corner detector · FPGA · Area-time optimization · Data-path transposition · Resource sharing

1 Introduction

Real-time computer vision algorithms are extensively used in a wide range of applications such as vision-based navigation of unmanned vehicles [1] and robots [2], video encoding [3], object tracking [4] and visual SLAM (Simultaneous Localization and Mapping) [5]. While end-to-end deep learning methods such as using CNN (Convolutional Neural Network) have shown impressive results in recent years, the power budgets of embedded applications are expected to limit the maximum achievable accuracy and runtime of these methods on platforms with restricted computational resources [35]. As such, classical computer vision algorithms with lower computational complexity are still viable solutions for deployment in embedded vision applications with tight computational constraints to meet real-time requirements.

A fundamental step in these applications is the detection of corners which represent identifiable anchor points in the image. Corners are used for matching between images (e.g., image registration), object tracking, and as robust image representation when combined with feature descriptors for object recognition. It is well recognized that corner detection is a computing intensive step. For example, the FAST (Features from Accelerated Segment Test) corner detector [16–18] is used to compute the ORB (Oriented FAST and Rotated BRIEF) feature descriptor for visual SLAM [34]. The runtime of ORB computation contributes to over a third of the real-time tracking process. Hence, it is critical to accelerate the corner detection operations to meet real-time application requirements.

Published online: 17 October 2017

Generally, there are two approaches for increasing the computation efficiency of corner detection. The first approach employs algorithmic techniques to reduce the computational complexity, usually with some accuracy trade-offs. For example, the work in [6] approximates the complex corner measure of the Harris corner detector [7] to rapidly prune away non-corners before selecting corners from the reduced set of candidates. While such methods have demonstrated significant speedup in corner detection on software-based platforms, they may not be amenable to efficient hardware implementations since large memories are typically required to store the image frames and intermediate results.

The second approach, which is the focus of this paper, accelerates corner detection through custom hardware accelerators using Application Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs). Several hardware designs have been recently proposed for real-time corner detection [8–15]. These techniques often exploit the inherent parallelism in the corner detectors and find a reasonable trade-off between the number of line buffers and the computational resources in order to manage the resource utilization while ensuring high throughput and acceptable result quality. However, the computational flow of these architectures remains largely unchanged and there has been little effort undertaken to investigate data-path optimizations for achieving further architectural gains.

In this paper, we present a novel hardware design strategy using data-path unrolling to realize an area-time efficient, pipelined FAST (Features from Accelerated Segment Test) corner detector [16–18] architecture that does not require intermediate full frame buffering. Instead of employing a large 7×7 convolution buffer for examining a single pixel at each clock cycle, which is the typical method adopted in existing works [8, 11, 12], we propose to use a 7×3 convolution buffer and unroll the data-path to enable multiple pixels to be examined concurrently. This effectively reduces the number of registers for the convolution buffer. Logic folding [19, 20] is then applied to merge the unrolled data-paths and substantially reduce the combinational operations. The proposed logic folding approach also exploits regularity to reduce the multiplexing overhead (which is typically introduced during resource sharing). This contributes to lowering the delay and area overhead that are often incurred as a result of resource sharing. Synthesis results on FPGA show that the proposed architecture leads to 20% critical path delay reduction and about 24% area reduction compared to an existing architecture. The proposed architecture can achieve over 60 frames per second for 1920×1080 HD video. We also implemented the proposed architecture on a FPGA prototyping platform to demonstrate its real-time capability.

The paper is organized as follows. In Sect. 2, we discuss the existing work in accelerating corner detection algorithms. We then describe the FAST algorithm and an existing hardware implementation in Sect. 3. Section 4 presents the proposed design. The synthesis results are shown in Sect. 5 to demonstrate the area-time benefits of our approach. Section 6 concludes the paper.

2 Related work

As corner detectors play a fundamental role in many computer vision applications, several corner detection algorithms have been proposed over the last 30 years [21, 22]. Motivated by the increasing demand for high-performance, a number of hardware architectures have been presented for well-known corner detection algorithms, e.g., ShiTomasi [23], Harris [7], SUSAN [24], and FAST [16–18].

Hardware implementations for Harris have been proposed on ASIC [25], FPGA [13–15], cell processor [26], and SIMD architecture [27]. The ShiTomasi and Harris corner detectors compute an autocorrelation matrix using the first-order derivatives of the intensity values, and this matrix represents the degree of intensity variations in different directions around a pixel. A complex corner measure computation is then performed for every pixel in the image. This step is highly computing intensive, requiring floating-point arithmetic and becomes a bottleneck for real-time vision tasks. In [28], a simpler floating-point format is used by customizing instructions on the Nios-II processor. In [29], a hardware implementation that performs Harris corner detection on a rank transform image instead of the original image is presented. FPGA implementation for ShiTomasi in [30] employs an alternative corner measure that uses only integer arithmetic consisting of additions and multiplications, and avoids the transcendental operations. The SUSAN detector operates directly on the image intensity by computing the fraction of pixels within a neighborhood that have similar intensity as the center pixel. FPGA implementation for SUSAN has been presented in [31]. The work in [14] proposed a flexible hardware implementation for computing both the Harris and SUSAN detectors from gray-level images.

The FAST algorithm was introduced as a compromise between the quality and the speed of corner detection. It extends the SUSAN detector by considering only pixels on a circle around the center pixel and uses an efficient method to classify the center pixel as a corner. The FAST algorithm was first presented in [16] and later improved in [17, 18]. The improved version employs machine learning to build a decision tree from a set of training images for

classifying corners on future images. The work in [9] presented a FPGA implementation of the machine learned FAST algorithm that is based on a binary look-up table, while the work in [8] presented a FPGA implementation of the original FAST algorithm. The work in [11] presented an architecture of the original FAST algorithm, which runs on a FPGA at 50 MHz operating frequency, and showed that significant performance gain can be achieved over the software implementation running on a 1 GHz mobile phone.

FAST has also been utilized as a preliminary step for computing feature descriptors such as ORB (Oriented FAST and Rotated BRIEF) [32] and BRISK (Binary Robust Invariant Scalable Keypoints) [33], which are used for a wide range of applications, e.g., object recognition, visual SLAM, image representation, 3D scene reconstruction, motion tracking. Recently, the work in [12] presented an architecture of the FAST feature detector and BRIEF feature descriptor which can process the input images of 1920×1080 resolution at 48 frames per second.

Since many computer vision applications employing corner detectors run on tightly constrained embedded systems, e.g., mobile robots, mobile devices, UAVs, there is a need to investigate design techniques that not only leads to real-time computation but also area-efficient solutions, which results in lower cost. This is important as these embedded systems need to be affordable for mass volume deployment. However, existing hardware implementations of corner detectors often optimize for speed only, and there are very few reported attempts to realize area-time-efficient hardware corner detectors.

3 Baseline architecture of FAST corner detector

The implementations of the original FAST algorithm presented in [8, 11, 12] adopt similar computational blocks which will be used as our baseline architecture. The original FAST algorithm proposed in [16] tests for a corner at each pixel p_i in an image frame by examining the Bresenham circle.

A 7×7 convolution buffer centered on p_i is used to enable parallel examination of the 16 surrounding pixels to facilitate the testing of one pixel per clock. Let x_j^i, where $j = 1, 2, 16$, be the pixels on the Bresenham circle that are used in the corner test of pixel p_i.

Figure 1 shows the convolution buffer (light gray) for corner testing of pixel p_i and the corresponding x_j^i pixels (dark gray) at time t, $t+1$ and $t+2$. Each pixel x_j^i in the convolution buffer is evaluated in parallel with pixel p_i to generate two 16-bit member vectors, one for bright members (m_{B}^i) and one for dark members (m_{D}^i):

$$m_{\mathrm{B}}^i = \begin{bmatrix} x_1^i > T_{\mathrm{H}}^i \\ x_2^i > T_{\mathrm{H}}^i \\ \vdots \\ x_{16}^i > T_{\mathrm{H}}^i \end{bmatrix} \tag{1}$$

$$m_{\mathrm{D}}^i = \begin{bmatrix} x_1^i < T_{\mathrm{L}}^i \\ x_2^i < T_{\mathrm{L}}^i \\ \vdots \\ x_{16}^i < T_{\mathrm{L}}^i \end{bmatrix} \tag{2}$$

where $T_{\mathrm{H}}^i = p_i + t$, $T_{\mathrm{L}}^i = p_i - t$, and t is a predefined threshold. Each element in the member vector is set to '1' if the corresponding condition is true, otherwise it is set to '0'. The scores for the bright and dark members are then calculated as shown in Eqs. (3) and (4).

$$\mathrm{score}_{\mathrm{B}}^i = (m_{\mathrm{B}}^i)^T * \begin{bmatrix} x_1^i - T_{\mathrm{H}}^i \\ x_2^i - T_{\mathrm{H}}^i \\ \vdots \\ x_{16}^i - T_{\mathrm{H}}^i \end{bmatrix} \tag{3}$$

$$\mathrm{score}_{\mathrm{D}}^i = (m_{\mathrm{D}}^i)^T * \begin{bmatrix} T_{\mathrm{L}}^i - x_1^i \\ T_{\mathrm{L}}^i - x_2^i \\ \vdots \\ T_{\mathrm{L}}^i - x_{16}^i \end{bmatrix} \tag{4}$$

A final score value is calculated for each p_i as shown in Eq. (5). A contiguity check [Eq. (6)] is used to determine if there are at least c contiguous elements in m_{B}^i or m_{D}^i that are true. In [8, 11, 12], $c = 9$ (hence the algorithm is called FAST-9). The check for the contiguous elements of each m_{B}^i and m_{D}^i is implemented using sixteen 9-input logical AND operation, where the outputs are ORed. A corner is detected if the output of the OR operation is '1' for either m_{B}^i or m_{D}^i. Finally, non-maximum suppression is applied to determine whether a pixel is a corner or a non-corner. A pixel is a corner if it has a maximal score among the scores of its adjacent neighbors.

$$\mathrm{score}_i = \max\left(\mathrm{score}_{\mathrm{B}}^i, \mathrm{score}_{\mathrm{D}}^i\right) * C_i \tag{5}$$

$$C_i = \left(\vee_{j=1}^{16}\left(\wedge_{k=j}^{j+8} m_{\mathrm{B}}^i(k-1)_{\mathrm{mod}16+1}\right)\right) \vee \times \left(\vee_{j=1}^{16}\left(\wedge_{k=j}^{j+8} m_{\mathrm{D}}^i(k-1)_{\mathrm{mod}16+1}\right)\right) \tag{6}$$

Figure 2 shows the baseline architecture. Similar to existing implementations in [8, 11, 12], we assume a single input pixel of n-bit (in our implementation $n = 8$ for grayscale image) arrives at each clock cycle. Seven row buffers as shown in Fig. 3 are concatenated in the form of FIFO delay buffers to cache the incoming pixels. The size

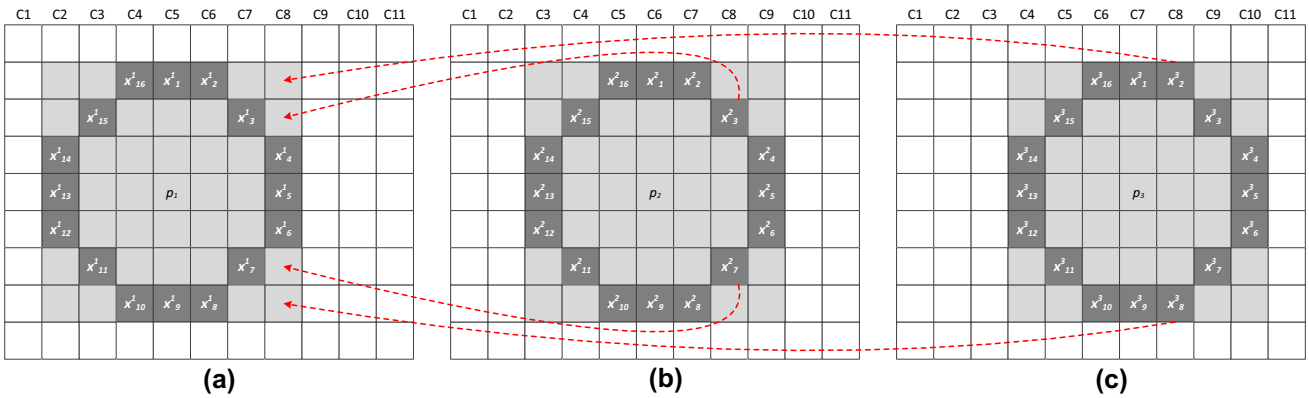


Fig. 1 Testing for corners at pixel **a**: p_1 at time t, **b** p_2 at time $t+1$, and **c** p_3 at time $t+2$. The dotted lines show that at time t, the pixels in column C8 of the convolution buffer (light gray) can be used for partial examination of pixels p_1, p_2, and p_3

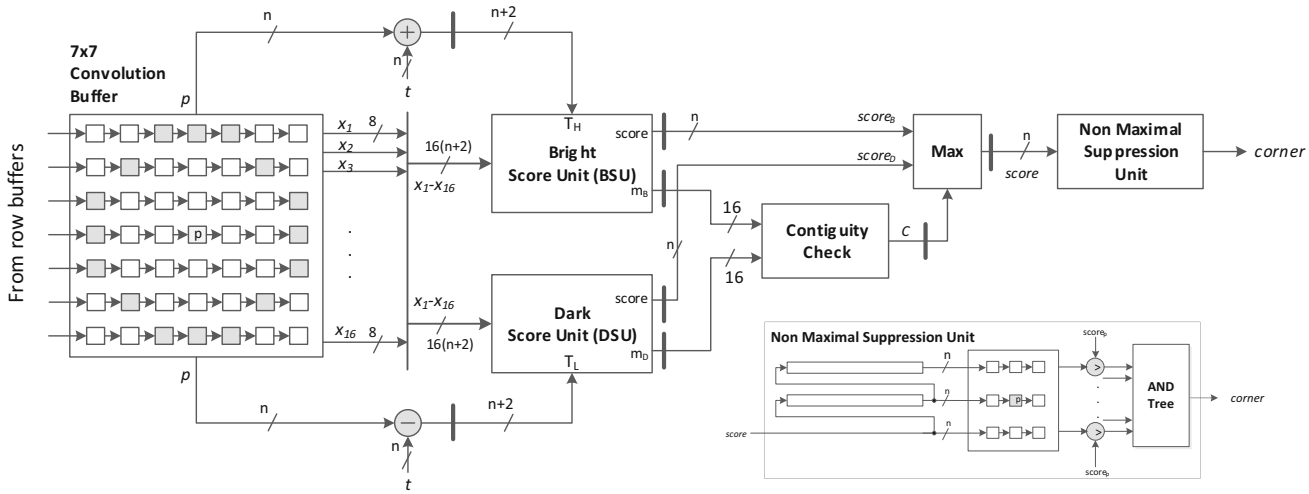


Fig. 2 Baseline architecture and NMS unit (inset)

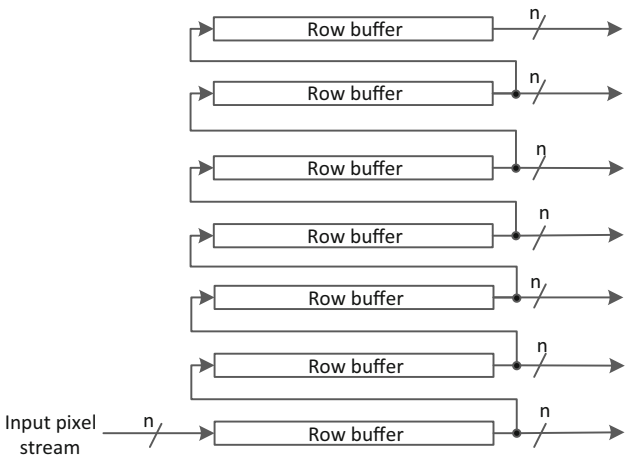


Fig. 3 Row buffer for caching seven rows

of each row buffer is equivalent to the horizontal resolution of the image, and hence each row buffer effectively delays the input by one row. The pixels at the tail end of each row buffer are shifted into the 7×7 convolution window as shown in Fig. 2.

The Bright Score Unit (BSU) and Dark Score Unit (DSU) determine the 16-bit member vectors m_B^i, m_D^i and score_B^i, score_D^i in parallel. A 4-level adder tree is employed for computing the score values. The LSB of the score values are truncated to n bits. The architecture of BSU and DSU is shown in Fig. 4. Member vectors m_B^i, m_D^i are used by the contiguity check to compute C_i. The Max unit computes the score of p_i based on the bright score, dark score and contiguity check.

Finally, the Non-Maximal Suppression (NMS) unit determines if a pixel is a corner or not by comparing its score value to the score values of its eight adjacent pixels. To achieve this, two row buffers are used in the NMS unit to produce a 1-bit output that denotes whether the corresponding pixel is a corner or non-corner (see inset of Fig. 2). Note that all the outputs of each module in Fig. 2 are registered, creating a pipelined design with one input

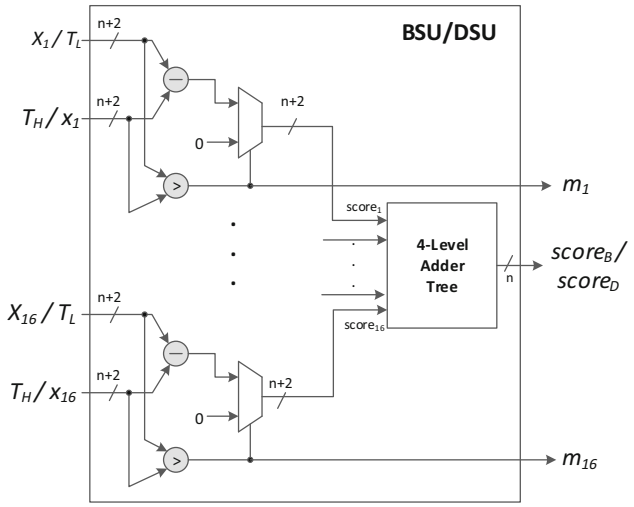


Fig. 4 BSU/DSU in the baseline architecture

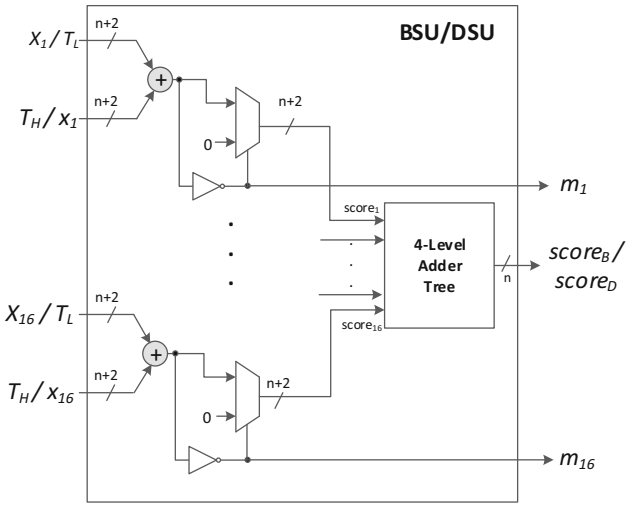


Fig. 5 Modified Bright/Dark Score Unit

and one output per clock cycle without the need of an input or intermediate frame buffer.

The critical path of the baseline architecture lies in BSU/DSU, i.e., $T^b_{\text{cp}} = 5 * T_{\text{ADD}} + T_{\text{MUX}}$, where T_{ADD} and T_{MUX} is the time required by the adder and multiplexer, respectively.

4 Proposed architecture

In this section, we present three area-time optimization techniques for the FAST corner detector. The first technique effectively reduces the number of comparators in the BSU/DSU by employing two's complement adders to compute the scores and exploiting the sign bit to determine the members (Sect. 4.1). The second technique unrolls the data-path of the baseline architecture to reduce the convolution buffer size (Sect. 4.2). The third technique employs logic folding to enable resource sharing while minimizing the multiplexing overhead (Sect. 4.3).

4.1 Modified BSU/DSU

The architecture of the BSU and DSU can be easily simplified by adopting two's complement adders to compute $\text{score}^i_{\text{B}}$, $\text{score}^i_{\text{D}}$, and using the sign bits of the scores to determine the member vectors as shown in Fig. 5. This effectively eliminates 16 comparators in each of the BSU and DSU without introducing much additional critical path delay. The critical path of the modified BSU/DSU is $T^{bm}_{\text{cp}} = 5 * T_{\text{ADD}} + T_{\text{INV}} + T_{\text{MUX}}$, where T_{INV} is the delay of an inverter.

4.2 Data-path unrolling

The baseline architecture in Fig. 2 utilizes a 7×7 convolution buffer to enable the 16 pixels x^i_j in the Bresenham circle to be evaluated in parallel with pixel p_i. As illustrated in Fig. 1, only one pixel p_i is tested for a corner at each time step. This approach results in under-utilization of the other 33 $(7 \times 7 - 16)$ pixels in the convolution buffer which could be used for partial evaluations of multiple p_i's. This is clearly illustrated with the red dotted lines in Fig. 1 which shows at time t, all the pixels in column C8 of the convolution buffer can be used for partial examination of pixels p_1, p_2, and p_3. A closer examination will reveal that the pixels in a single column of the convolution buffer can be utilized for simultaneous partial examination of 7 p_i's (i.e., top/bottom pixels contribute to the evaluation of 3 p_i's, second/sixth pixels contribute to the evaluation of 2 p_i's, and the middle three pixels contribute to the evaluation of 2 p_i's).

In order to maximize the utilization of the convolution buffer, we propose to unroll the data-path of BSU/DSU. Note that $x_1, x_2, \ldots, x_{16}$ in Fig. 6 are from a single column of the convolution buffer. It can be observed from Fig. 6 that the BSU/DSU of the baseline architecture is unrolled into seven pipeline stages. Each stage performs partial evaluation of a single p_i. The partial results of each p_i (partial member vectors, bright/dark scores) will be passed to the next pipeline stage after each clock cycle. Except for the first pipeline stage, the score values of each stage are added to the score value from the previous pipeline stage. The full evaluation of a single p_i will be completed after seven clock cycles at the final pipeline stage.

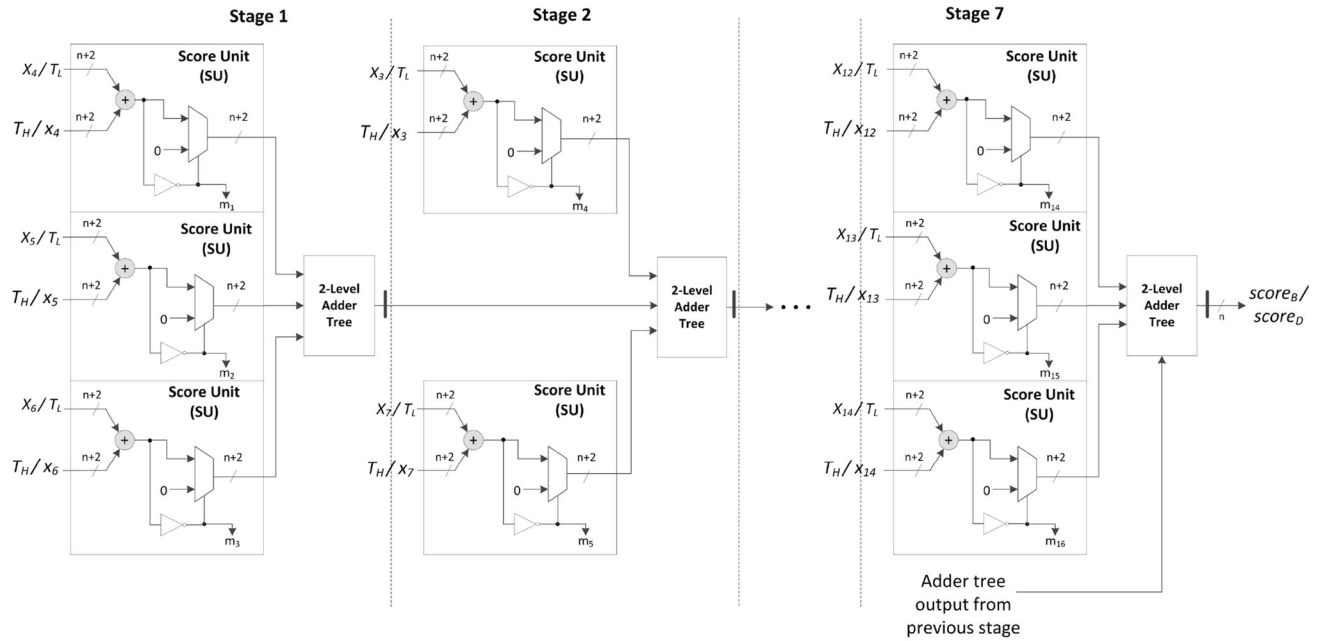


Fig. 6 Unrolled BSU/DSU

We can use the example in Fig. 1 to describe the proposed computational flow. Lets assume that at time t, the content of the single column buffer is C8. At time t, the partial results of p_1 is computed using x_4^1, x_5^1, x_6^1 in the first pipeline stage of BSU/DSU. In the next clock cycle at time $t+1$, the partial results p_1 are computed using x_3^1, x_7^1 in the second pipeline stage and the new score values are added to the previously computed score values from the first stage. At the same time, new member vectors are generated and concatenated with the previously identified member vectors. This is repeated until the last pipeline stage at $t+6$ computes the partial results of p_1 using x_{12}^1, x_{13}^1, x_{14}^1 and concatenates/adds the partial results in the previous pipeline stage to obtain $m_{\mathrm{B}}^1/m_{\mathrm{D}}^1$ and $\mathrm{score}_{\mathrm{B}}^1/\mathrm{score}_{\mathrm{D}}^1$. Similar to the baseline architecture, a single output is produced at each clock cycle. The critical path of the unrolled BSU/DSU is $T_{\mathrm{cp}}^{dt} = 3 * T_{\mathrm{ADD}} + T_{\mathrm{INV}} + T_{\mathrm{MUX}}$, which is lesser than the baseline since only 2-level adder tree is used in a pipeline stage.

In our proposed architecture (shown in Fig. 7), the 7×7 convolution buffer is replaced by a 7×3 convolution buffer. While a single column of the convolution buffer is sufficient for the partial evaluation of seven pixel centers (as discussed above), three convolution buffer columns are required to cache the incoming pixels before the center pixel is read from the FIFO delay buffers (Fig. 3). For example, it can be observed in Fig. 1a that we need to cache the pixels in columns C6, C7 and C8 before we can obtain the center pixel p_1 from the FIFO delay buffer.

4.3 Logic folding with maximal design regularity

It can be observed that each pipeline stage of the BSU and DSU in Fig. 6 consists of two parts: (1) Score Unit (SU) and (2) adder tree. The SUs of the BSU and DSU perform similar operations, i.e., they compute the members and partial score terms. In particular, the SU for BSU computes m_{B}^i and $m_{\mathrm{B}}^i * (x_j^i - T_{\mathrm{H}}^i)$, while the SU for DSU computes m_{D}^i and $m_{\mathrm{D}}^i * (T_{\mathrm{L}}^i - x_j^i)$. As such, the SUs for the corresponding BSU and DSU can be merged for further resource savings. Figure 7 shows the architecture of the proposed architecture where the SUs for computing the bright and dark score terms are merged as Merged Score Units (MSUs) in the Shared Score Unit (SSU). Each SSU stage consists of either two or three MSUs. The outputs of the MSUs in each pipeline stage are passed to the respective Bright Score Adder Tree (BSAT) and Dark Score Adder Tree (BSAT). Similar to the architecture in Fig. 6, each pipeline stage computes the partial evaluations of a single p_i (i.e., bright member vectors, bright score, dark member vectors, and dark score). As mentioned earlier, the last pipeline stage produces the final score values of p_i. The difference lies in that the same logic resources in the MSUs are used to compute both the bright and dark score values (and member vectors) at a higher clock frequency. The values of $T_{\mathrm{H}}/T_{\mathrm{L}}$ are also shifted through the SSU stages in a pipelined manner, i.e., the MSUs in a particular stage makes use of the shifted $T_{\mathrm{H}}/T_{\mathrm{L}}$ values from the previous stage.

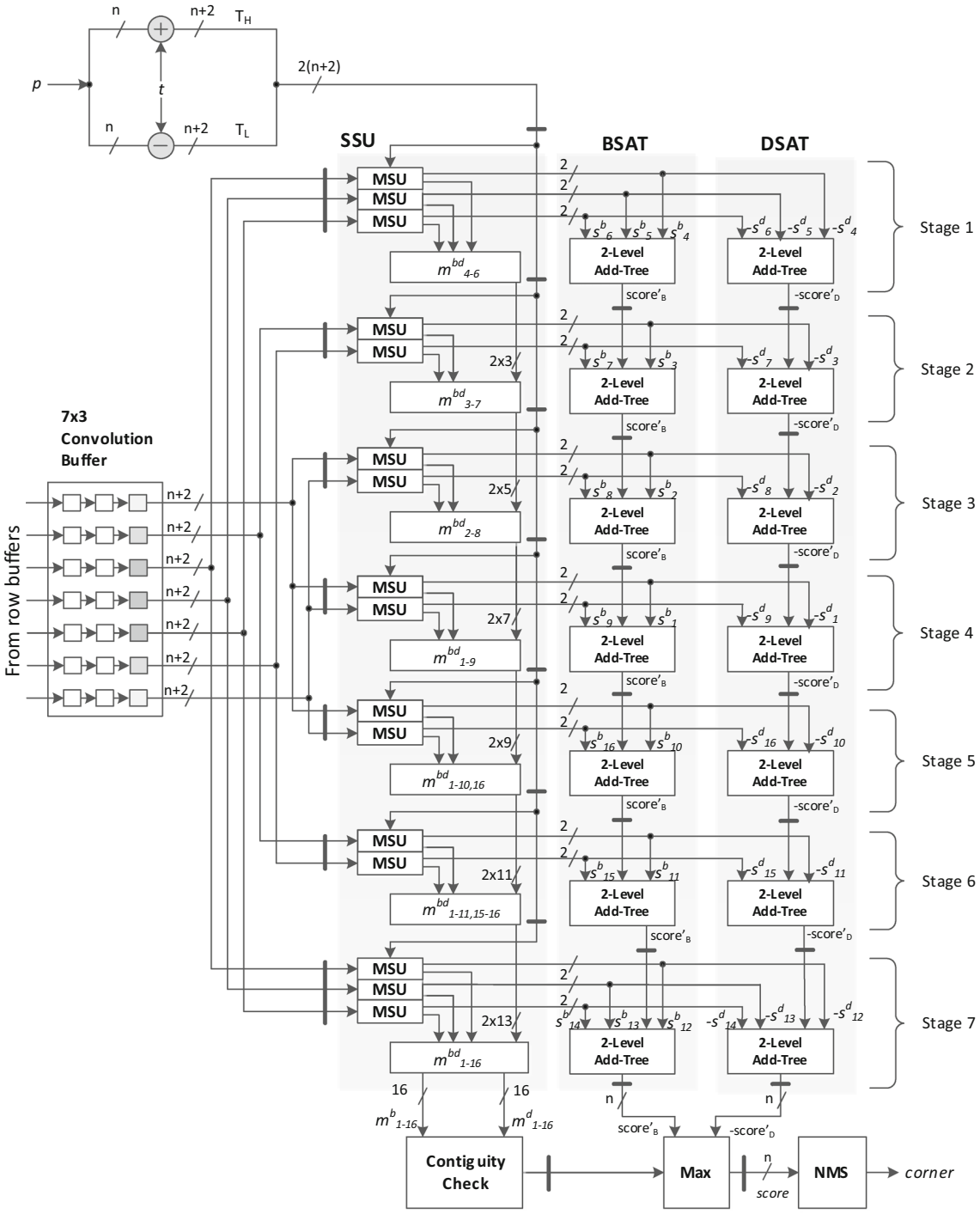


Fig. 7 Proposed architecture

In the proposed architecture, the input pixels arrive at the rising edge of the 1× system clock. T_H/T_L and the score values $\text{score}^i_B/\text{score}^i_D$ are passed from one pipeline stage of the SSU to the next at the rising edge of the 1× system clock. The membership vectors are also shifted through the pipeline registers based on the 1× clock frequency. This is similar to the data-flow of the architecture in Fig. 6. Logic folding [19, 20] is performed on BSU and DSU to obtain the MSU which operates at 2× the operating frequency of the system clock. This enables the bright and dark scores for pixel p_i to be computed in each pipeline stage using the samelogic resources without compromising on the throughputof the architecture. Figure 8 shows the architecture of a single SSU pipeline stage, which consists

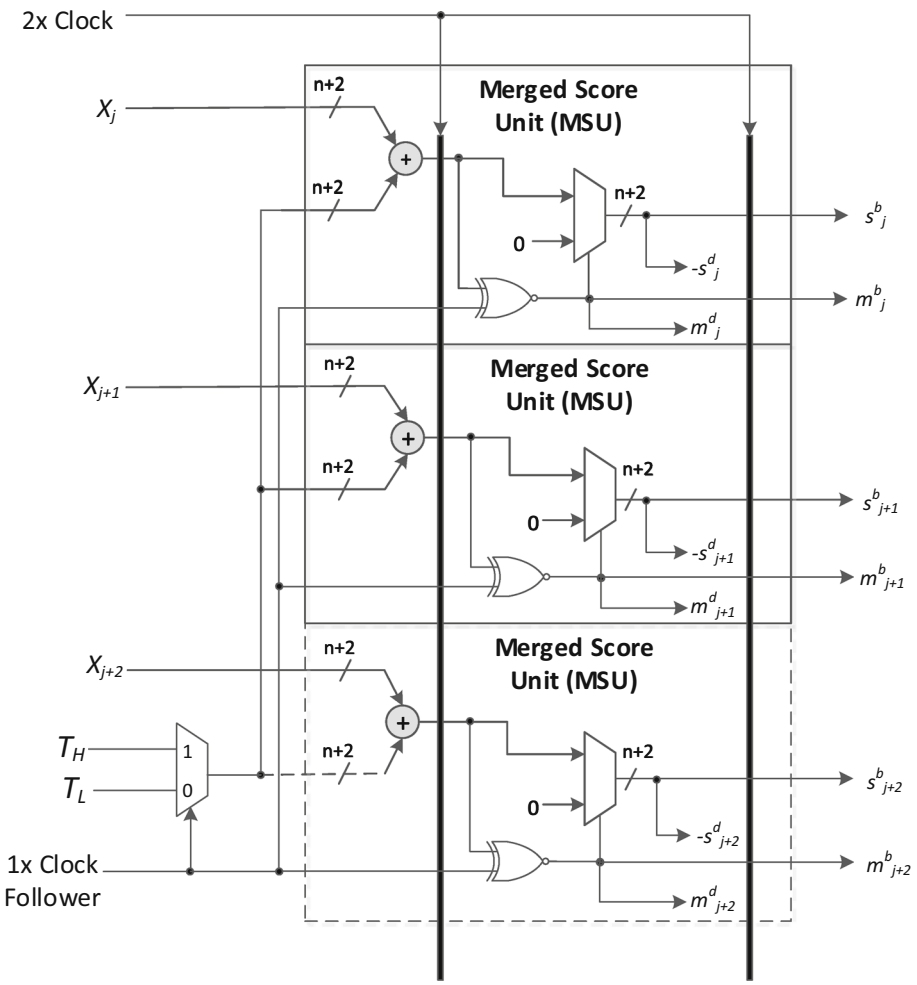


Fig. 8 Single pipeline stage of Shared Score Unit (SSU)

of three or two MSUs. A 2× clock, which is exactly twice the frequency of the 1× system clock drives the registers in the SSU stage to enable both the bright and dark score terms and membership vectors to be computed within one clock period of the 1× system clock (equivalent to two pipeline stages of the 2× clock frequency). The operation of the SSU pipeline is as follows. When the 1× clock follower (which matches the 1× system clock) is high, T_{H} from the previous SSU pipeline stage will be selected as inputs (via the multiplexer) to the first stage of the MSUs for computation. When the 1× clock follower is low, T_{L} from the previous SSU pipeline stage will be selected as inputs instead. After two clock cycles (with respect to the 2× clock), the new bright and dark score terms (and member vectors) will be available. This will be used by the BSAT and DSAT to compute the cumulative score values from previous pipeline stages. Hence, at the rising edge of the next system clock, both the score values $\mathrm{score}^i_{\mathrm{B}}/\mathrm{score}^i_{\mathrm{D}}$ and membership vectors $m^i_{\mathrm{B}}/m^i_{\mathrm{D}}$ will be available and shifted to the subsequent SSU pipeline stage. Similar to the previous architectures, a single output is produced at each cycle of the 1× system clock.

computed in each pipeline stage using the same logic resources without compromising on the throughput of the architecture.

Generally, applying logic folding leads to reduced overall combinational logic area but may introduce additional synchronizing registers and multiplexing overhead. In order to lower the multiplexing overhead, the logic regularity for computing the bright and dark scores must be maximized to reduce the fan-ins and fan-outs of the folded architecture [19]. This will lead to a more simplified interconnect structure that avoids the need of multiplexers for resource sharing. In the proposed architecture, regularity is exploited by using the same MSU logic resources for computing both the bright and dark score terms without time-multiplexing. As such, the computed dark score is $-\mathrm{score}^i_{\mathrm{D}}(-(x^i_j - T^i_{\mathrm{L}}))$ and the polarity of the final dark score value can be inverted using a simple sign inversion circuit (implemented in the Max unit). In addition, the XNOR gate (with one of inputs tied to the 1× clock follower) is used to replace the inverter for determining the member vectors as well as selecting between the score value and 0.

The critical path of the proposed architecture (with reference to the 1× system clock) lies in the pipeline stage of SSU, i.e., $T_{cp}^{dtl} = T_{ADD} + T_{XNOR} + 2 * T_{MUX} + T_{REG}$, where T_{XNOR} and T_{REG} is the delay of an XNOR gate and register, respectively. This is evidently lower than the critical path delay of the baseline architecture (see Sect. 3).

5 Results and discussion

In this section, we provide experimental results for the proposed implementation in terms of accuracy and hardware synthesis results.

5.1 Accuracy evaluation

We used the repeatability criteria [22] to compare the accuracy of the baseline and proposed implementations. The repeatability criterion is based on the notion that detection of corners should be invariant of imaging conditions, e.g., blurring, zooming, and rotation of the scene. An accurate feature detector should be robust to the changes in imaging conditions and hence should be able to detect features at close proximity between images with changes in viewpoint. The repeatability rate is defined as the ratio of the number of repeated features between two images within certain pixel allowance, to the minimum number of features that are in common region of the two images of the same scene but with changes in imaging condition(s). We have used the image dataset from [36] for the accuracy evaluation. These challenging datasets contain four image sets (Boat, Trees, UBC and Wall) with various image transformation sequences such changes in viewpoint, zoom, rotation and illumination. An example of the image sequence for 'Boat' is shown in Fig. 9.

The difference in the repeatability rate (a pixel allowance of 1.5 pixels is used in the evaluations) of the proposed and the baseline architectures is computed using Eq. (7). Table 1 reports the minimum, maximum and average difference in repeatability rate for the four image sets.

It is evident that the overall difference in repeatability between the proposed and baseline architectures is marginal for the image sets considered, i.e., only 0.0588 average difference in repeatability rate. These results demonstrate that the proposed architecture has similar degree of robustness compared to the baseline architecture.

$$\delta r = \text{repeatability}_{\text{proposed}} - \text{repeatability}_{\text{baseline}} \quad (7)$$

5.2 Hardware synthesis results

Table 2 shows the required resources for the baseline (Fig. 2) and proposed (Fig. 7) architectures. The resources for Contiguity Check, Max and NMS are not shown since they do not vary notably among the two architectures. The number of registers (REG) is normalized to 8-bit registers, and a, b, c indicate the register contributions from the SSU, member vectors, and synchronization registers for the input pixels and T_H/T_L.

It can be observed from Table 2 that the proposed architecture has lesser combinational resources compared to the baseline architecture due to: (1) elimination of the comparators that are used to determine the member vectors (see Fig. 5), and (2) resource sharing of the score units (see Fig. 8). In particular, the proposed SSU has about 4× lesser adder equivalent resources than the BSU and DSU, due to the above-mentioned optimizations. The number of required multiplexers in the SSU is also lesser due to resource sharing and the proposed logic folding strategy that minimizes multiplexing overhead. The proposed SSU requires additional 16 XNOR gates which is insignificant compared to the overall combinational resource savings.

In addition, the proposed architecture employs a smaller convolution buffer (7×3) compared to the baseline which requires a 7×7 convolution buffer. This leads to notable register savings in the proposed architecture. While the total number of registers required for logic folding and pipelining in the proposed architecture is larger than that required for the baseline architecture, the delay analysis in Sects. 3 and 4.3 shows that the proposed optimizations have resulted in significantly lesser critical path delay.

The two corner detector architectures were implemented using Verilog and synthesized using Quartus II Version 13.0.1 and targeting the Altera FPGA Cyclone IV GX (EP4CGX150DF31C7) device. The designs were synthesized to achieve minimum clock period. It can be observed from columns 2 and 3 in Table 3 that the proposed method has lesser combinational resources than the baseline, but more registers. This is consistent with our resource analysis. The proposed architecture achieves an overall 24% reduction in area utilization (in terms of logic elements) compared to the baseline architecture, mainly due to the large savings in combinational resources as discussed above.

The critical path of the proposed architecture is 20% lower than the baseline, which is also consistent with our delay analysis. While it is possible to further pipeline the BSU/DSU of the baseline architecture to reduce critical path delay, this will also increase the area utilization. As shown in Table 3, the area utilization of the current baseline architecture is already notably higher than the proposed architecture. Note that the critical path delay of the proposed architecture in Table 3 refers to the 1× system clock. In particular, the proposed architecture can achieve over 60 frames per second for 1920 × 1080 HD video. Column 6 of Table 3 shows the area-delay product (ADP)

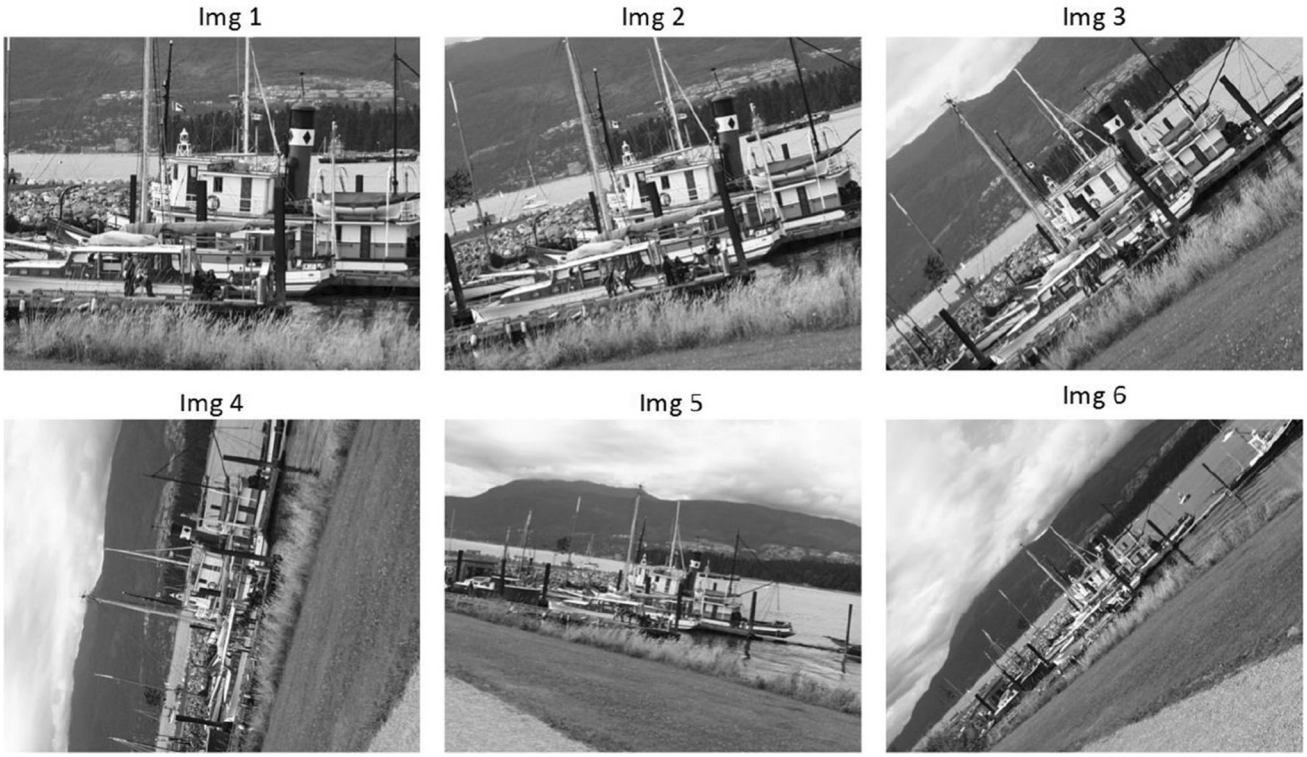


Fig. 9 Image set boat; the top left image is the original and subsequent images have increasing viewpoint changes

Table 1 Absolute difference in repeatability rate

Image	Min	Max	Average
Boat	0.0676	0.0884	0.0773
Trees	0.0276	0.0763	0.0563
UBC	0.0348	0.0747	0.0251
Wall	0.0522	0.0990	0.0766

Table 2 Resource comparison

	Baseline	Proposed
Convolution Buffer		
REG	49	21
SU/SSU		
ADD	64	16
MUX	32	23
XNOR	0	16
REG	0	$34^a + 16^b + 30^c$
Adder Tree		
ADD	30	32
REG	2	14

of the two architectures. The proposed architecture achieved about 38.9% area-delay product reduction over the baseline architecture. The proposed architectures are implemented on the TERASIC DE2i-150 FPGA development kit that captures 640 × 480 video frames as shown in Fig. 10. The operating frequency of the FAST architecture in the FPGA demonstration is 25 MHz due to the camera and display constraints, although the proposed architecture can achieve a maximum operating frequency of 125 MHz (based on 1× system clock).[1]

Table 3 FPGA synthesis results

Method	Comb	Reg	Total LE	Delay (ns)	ADP (LE.ns)
Baseline	1294	618	1530	10.0	15,300
Proposed	796	987	1169	8.0	9352

6 Conclusion

This paper presents a novel architecture for the FAST corner detector that relies on a unrolled data-path structure to compute the score values of multiple corners concurrently. The proposed architecture eliminates the need for a large convolution buffer by using a significantly smaller convolution buffer to cache the incoming pixels. The pixels in the last column of the reduced convolution buffer are

[1] Realizing the proposed method using a 1× clock frequency of 125 MHz may not be possible on some of the existing FPGA systems.

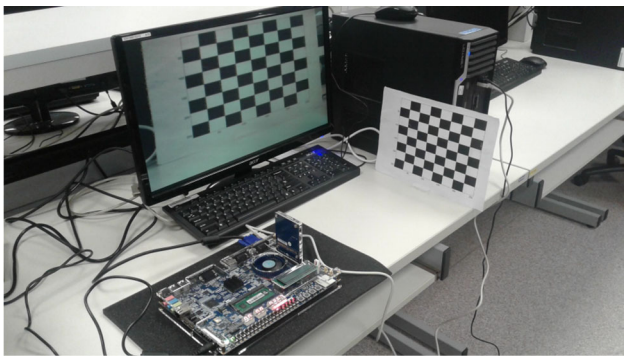


Fig. 10 FPGA evaluation platform

utilized simultaneously for computing the partial corner score values. In addition, the proposed architecture employs logic folding with maximal regularity to reduce the combinational logic while at the same time avoiding the multiplexing overhead. Synthesis results show that the proposed architecture leads to 20% critical path delay reduction and about 24% area reduction compared to an existing architecture reported in the literature.

References

1. Ehsan, S., McDonald-Maier, K.D.: On-board vision processing for small UAVs: time to rethink strategy. In: Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems, pp. 75–81 (2009)
2. Schmidt, A., Kraft, M., Kasinski, A.: An evaluation of image feature detectors and descriptors for robot navigation. In: International Conference on Computer Vision and Graphics, pp. 251–259 (2010)
3. Bhaskaranand, M., Gibson, J.D.: Low-complexity video encoding for UAV reconnaissance and surveillance. Mil. Commun. Conf. **6375**, 251–259 (2011)
4. Gauglitz, S., Hollerer, T., Turki, M.: Evaluation of interest point detectors and feature descriptors for visual tracking. Int. J., Comput. Vis. **94**, 335–360 (2011)
5. Gil, A., Mozos, O., Ballesta, M., Reinoso, O.: A comparative evaluation of interest point detectors and local descriptors for visual SLAM. Mach. Vis. Appl. **21**, 905–920 (2010)
6. Ramakrishnan, N., Wu, M., Lam, S.-K., Srikanthan, T.: Enhanced low complexity pruning for corner detection. J. Real Time Image Process. **12**(1), 197–213 (2011)
7. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proceedings of the Fourth Alvey Vision Conference, pp. 147–151 (1988)
8. Kraft, M., Schmidt, A., Kasinski, A.J.: High-speed image feature detection using FPGA implementation of FAST algorithm. In: Proceedings of the Third International Conference on Computer Vision Theory and Applications (VISAPP 2008), Funchal, Portugal, 22–25 January 2008, pp. 174–179 (2008)
9. Dohi, K., Yorita, Y., Shibata, Y., Oguri, K.: Pattern compression of fast corner detection for efficient hardware implementation. In: International Conference on Field Programmable Logic and Applications, pp. 478–481 (2011)
10. Amaricai, A., Gavriliu, C.E., Boncalo, O.: An FPGA sliding window-based architecture Harris corner detector. In: International Conference on Field Programmable Logic and Applications (2014)
11. Soberl, D., Zimic, N., Leonardis, A., Krivic, J., Moskon, Miha: Hardware implementation of FAST algorithm for mobile applications. J. Signal Process. Syst. **79**(3), 247–256 (2015)
12. 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. **12**, 141 (2015). doi:10.5772/61434
13. Orabi, H., Shaikh-Husin, N., Ullah Sheikh, U.: Low cost pipelined FPGA architecture of Harris Corner Detector for real-time applications. In: International Conference on Digital Information Management, pp. 164–168 (2015)
14. Hernandez-Lopez, A., Torres-Huitzil, C., Garcia-Hernandez, J.J.: FPGA-based flexible hardware architecture for image interest point detection. Int. J. Adv. Robot. Syst. **12**, 93 (2015). doi:10.5772/61058
15. Chao, T.L., Wong, K.H.: An efficient FPGA implementation of the Harris corner feature detector. In: 2015 14th IAPR International Conference on Machine Vision Applications (MVA), pp. 89–93. IEEE (2015)
16. Rosten, E., Drummond, T.: Fusing points and lines for high performance tracking. In: International Conference on Computer Vision, pp. 1508–1515 (2005)
17. Rosten, E., Drummond, T.: Machine learning for high speed corner detection. In: European Conference on Computer Vision, pp. 430–443 (2006)
18. Rosten, E., Porter, R., Drummond, T.: Faster and better: a machine learning approach to corner detection. IEEE Trans. Pattern Anal. Mach. Intell. **32**, 105–119 (2010)
19. Mehra, R., Rabaey, J.: Exploiting regularity for low-power design. In: IEEE/ACM International Conference on Computer-Aided Design, pp. 166–172 (1996)
20. Canis, A., Anderson, J.H., Brown, S.D.: Multi-pumping for resource reduction in FPGA high-level synthesis. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 194–197 (2013)
21. Tuytelaars, T., Mikolajczyk, K.: Local invariant feature detectors: a survey. Found. Trends Comput. Graph. Vis. **3**(3), 177–280 (2008)
22. Schmid, C., Mohr, R., Bauckhage, C.: Evaluation of interest point detectors. Int. J. Comput. Vis. **37**(2), 151–172 (2000)
23. Jianbo, S., Tomasi, C.: Good features to track. In: Computer Vision and Pattern Recognition, pp. 593–600 (1994)
24. Smith, S.M., Brady, J.M.: SUSAN—a new approach to low level image processing. Int. J. Comput. Vis. **23**, 45–78 (1997)
25. Chih-Chi, C., Chia-Hua, L., Chung-Te, L., Chang, S.C., Liang-Gee, C.: iVisual: an intelligent visual sensor SoC with 2790fps CMOS image sensor and 205GOPS/W vision processor. In: Design Automation Conference, p. 9095 (2008)
26. Saidani, T., Lacassagne, L., Bouaziz, S., Khan, T.: Parallelization strategies for the points of interests algorithm on the cell processor. Parallel Distrib. Process. Appl. **4742**, 104–112 (2007)
27. Hosseini, F., Fijany, A., Fontaine, J.-G.: Highly parallel implementation of Harris Corner detector on CSX SIMD architecture. In: Guarracino, M.R., et al. (eds.) Euro-Par 2010 Parallel Processing Workshops. Euro-Par 2010. Lecture Notes in Computer Science, vol. 6586, pp. 137–144. Springer, Berlin, Heidelberg (2011)
28. Piskorski, S., Lacassagne, L., Bouaziz, S., Etiemble, D.: Customizing CPU instructions for embedded vision systems. In: Proceedings of the 2007 IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 59–64 (2007)

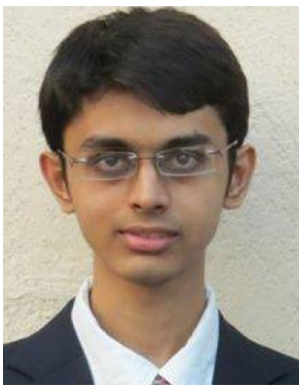
29. Tippetts, B., Lee, D.-J., Archibald, J.: An on-board vision sensor system for small unmanned vehicle applications. Mach. Vis. Appl. **23**(2), 113 (2012)
30. Benedetti, A., Perona, P.: Real-time 2-D feature detection on a reconfigurable computer. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 586–593 (1998)
31. Claus, C., Huitl, R., Rausch, J., Stechele, W.: Optimizing the SUSAN corner detection algorithm for a high speed FPGA implementation. In: Field Programmable Logic and Applications, pp. 138–145 (2009)
32. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: an efficient alternative to SIFT or SURF. In: ICCV 2011, pp. 2564–2571 (2011)
33. Leutenegger, S., Chli, M., Siegwart, R.Y.: BRISK: binary robust invariant scalable keypoints. In: IEEE International Conference on Computer Vision, pp. 2548–2555 (2011)
34. Mur-Artal, R., Montiel, J.M.M., Tardos, J.D.: ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Trans. Robot. **31**(5), 1147–1163 (2015)
35. Canziani, A., Paszke, A., Culurciello, E.: An Analysis of Deep Neural Network Models for Practical Applications. arXiv:1605.07678 (2017)
36. Affine Covariant Features. http://www.robots.ox.ac.uk/∼vgg/research/affine/

Siew-Kei Lam received his BA.Sc., M.Eng. and Ph.D. from School of Computer Science and Engineering (SCSE), Nanyang Technological University, Singapore. He is currently an Assistant Professor in SCSE, and his research investigates methods for realizing custom computing solutions in embedded systems. His current projects include developing architecture-aware algorithms for vision-enabled sensing, and design methodologies for secure and reliable embedded systems.

Teck Chuan Lim is currently pursuing his Bachelor of Engineering (Computer Engineering) in School of Computer Science and Engineering (SCSE), Nanyang Technological University, Singapore. His research interest lies in hardware and software design of embedded systems.

Meiqing Wu received M.S. degree in Computer Engineering from Peking University, China, in 2009, and her Ph.D. degree from the School of Computer Science and Engineering (SCSE), Nanyang Technological University, Singapore, in 2017. Her current research interests include stereo vision, motion analysis, object detection and tracking for urban traffic scene understanding

Bin Cao received the Bachelor's and Master's degrees from Wuhan University of Technology and Zhejiang University in China. He received his Ph.D. degree from Nanyang Technological University in 2006. Prior to current assignment, he worked in Seagate International as a Staff Engineer, and Research Scientist in the Institute for Infocomm Research, A*STAR. He is currently a Senior Research Fellow in Hardware and Embedded Software Lab (HESL), School of Computer Science and Engineering (SCSE), Nanyang Technological University, Singapore. Cao Bin's research interest lies in the areas of hardware acceleration for the embedded vision and stereo vision.



Bhavan A. Jasani received a dual degree consisting of B.E. (Hons.) Electrical and Electronics Engineering and M.Sc. (Hons.) Physics from Birla Institute of Technology and Science, Pilani, India, in 2016. Since then, he has been working as a research staff at Hardware and Embedded Systems Lab at School of Computer Science and Engineering, Nanyang Technological University, Singapore. He has been working on developing real-time, low-power and hardware-efficient pedestrian detection systems based on FPGAs and embedded GPUs. He's research interest lies in computer vision, machine learning and embedded systems.