A Low-Power Hardware Accelerator for ORB Feature Extraction in Self-Driving Cars

Raúl Taranco, José-Maria Arnau, Antonio González
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
{*taranco,jarnau,antonio*}*@ac.upc.edu*

***Abstract*—Simultaneous Localization And Mapping (SLAM) is a key component for autonomous navigation. SLAM consists of building and creating a map of an unknown environment while keeping track of the exploring agent's location within it. An effective implementation of SLAM presents important challenges due to real-time inherent constraints and energy consumption.**

ORB-SLAM is a state-of-the-art Visual SLAM system based on cameras that can be used for self-driving cars. In this paper, we propose a high-performance, energy-efficient and functionally accurate hardware accelerator for ORB-SLAM, focusing on its most time-consuming stage: Oriented FAST and Rotated BRIEF (ORB) feature extraction. We identify the BRIEF descriptor generation as the main bottleneck, as it exhibits highly irregular access patterns to local on-chip memories, causing a high performance penalty due to bank conflicts. We propose a genetic algorithm to generate an optimal memory access pattern offline, which greatly simplifies the hardware while minimizing bank conflicts in the computation of the BRIEF descriptor. Compared with a CPU system, the accelerator achieves 8x speedup and 1957x reduction in power dissipation.

***Keywords*-ORB, ORB-SLAM, hardware accelerator**

I. Introduction

Simultaneous Localization and Mapping (SLAM) [1] [2] is a crucial component in autonomous navigation systems and has attracted a lot of interest from both academia and industry in recent years. SLAM is a fundamental task for higher-level activities such as path planning and navigation, and is widely used in applications such as self-driving cars [3]. SLAM techniques build a map of an unknown environment and localize the exploring agent in that map using the on-board sensors. Vision sensors are the most promising alternative because cameras are inexpensive and compact, providing a vast amount of information of the environment. Among Visual SLAM solutions, feature-based ones have received particular attention because of its robustness to large motions and illumination changes compared with other approaches. However, these methods present important challenges mainly due to real-time inherent constraints and energy consumption budget available on potential targets [3].

In this work, we focus on a state-of-the-art SLAM solution: ORB-SLAM [4]. This system combines Features from Accelerated Segment Test (FAST) [5] and Binary Robust Independent Elementary Features (BRIEF) [6]. The former identifies the features in an image, whereas the latter generates a robust descriptor for each feature. These two parts, together with an orientation estimator, give rise to Oriented FAST and Rotated BRIEF (ORB) [7]. These features identify corners on the processed images and apply a rotation to generate their descriptors with the objective of providing viewpoint and rotational-invariant properties. According to our experiments, more than 60% of the execution time (see Figure 1) in ORB-SLAM is spent extracting features (ORB Extraction).

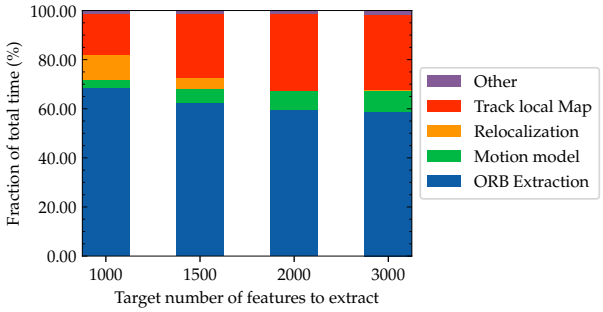


Figure 1: Average relative execution time of each part of ORB-SLAM running on a CPU. ORB feature extraction takes the majority of execution time.

In this paper, we propose a heterogeneous architecture for ORB-SLAM that combines a hardware accelerator for ORB feature extraction and a mobile CPU for the remaining tasks such as tracking, local mapping and loop closing [4]. Computing the rBRIEF descriptor is the most challenging part due to the irregular memory access patterns. Once a pixel has been identified as a corner, i.e. a feature in the image, a 256-bit descriptor has to be computed by performing 256 comparisons between pairs of pixels in the neighborhood of the corner. The locations of the 256 pairs of pixels do not follow any regular pattern and, in addition, they change dynamically due to the rotation angle. Previous ORB accelerators modify the rBRIEF algorithm to obtain a

 DOI 10.1109/SBAC-PAD53543.2021.00013

more hardware-friendly version, but at the cost of significant accuracy loss [8], [9]. We believe that reducing precision for a simpler implementation is not an acceptable trade-off in the context of self-driving cars and, hence, we take a completely different approach. Our solution stores the neighbor pixels in a multi-banked memory and, instead of using complex logic to dynamically schedule which pairs of pixels are accessed on every cycle, we develop a static scheduling based on a genetic algorithm that minimizes the number of conflicts in the banks for any rotation angle. The 256 pairs of pixels are processed in the order determined statically, which largely reduces the conflicts while requiring simple hardware. Due to the low cost of the rBRIEF unit, it can be replicated multiple times to achieve the target performance required to meet real-time constraints. Furthermore, our approach obtains an accuracy comparable to the software-based solutions.

In this paper we make the following contributions:

- We analyze the performance and energy consumption of ORB-SLAM on a state-of-the-art CPU. Our results show that the feature extraction is the main performance and energy bottleneck.
- We propose a high-performance, energy-efficient and functionally accurate hardware accelerator for ORB feature extraction, which is the main bottleneck of ORB-SLAM.
- We present a technique to generate a new rBRIEF static scheduling of the required operations that minimizes the number of conflicts when accessing the on-chip memory structures required for the descriptor computation. The technique is based on a genetic algorithm.
- The experimental results obtained show that the accelerator provides 7.8x speedup and 1957x power reduction on average compared with a CPU-only system. Additionally, we verify that the localization of the vehicle has no deviation with respect to the reference software implementation.

II. BACKGROUND

ORB-SLAM [10] [11] estimates the real trajectory of an agent equipped with a camera while building a representation of the surroundings. We are interested in its performance in the context of self-driving cars, where it has been ranked at the top of the available open source algorithms [12]. The algorithm is divided into three parts executed in different threads for Tracking, Local Mapping and Loop Closing.

Frames coming from the visual system are first processed by the Tracking thread which localizes the camera within the environment and decides when to insert a new key frame in the map. For every incoming frame, ORB Extraction has to be performed. This process spends up to 60% of the total algorithm processing time per frame.

ORB extraction can be divided into four main steps [7]: pyramid building (subsection II-A); FAST Keypoint Detection (subsection II-B); orientation estimation (subsection II-C) and rBRIEF generation (subsection II-D).

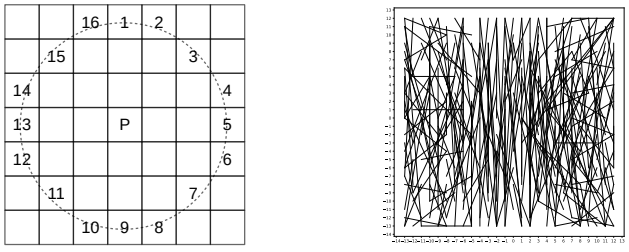
For each image, ORB algorithm first builds a scale pyramid, in which the higher scale is down-sampled from the previous lower scale. After this, the FAST corner detection identifies features at each scale. Then Non-Maximal Suppression (NMS) is applied in order to filter out some features and select those with the higher score within a defined neighborhood. Next, the orientation angle of each feature is computed using patch moments and the associated centroid [7]. Finally, each feature is encoded using the BRIEF descriptor rotated by the angle.

A. Pyramid Building

FAST does not produce multi-scale features but scale invariance is desirable. To achieve this, a scale pyramid of the images is used. At each level, the image from the previous level is subject to repeated smoothing and subsampling.

B. FAST Keypoint Detection

Features From Accelerated Segment Test (FAST), first introduced in [13], is a corner detector that can be used to extract feature points. FAST performs a test to classify a candidate pixel, p, as corner/no corner. This test consists in comparing the intensity of p with the intensities of the 16 pixels that form a Bresenham circle around the candidate, as shown in Figure 2a. A corner is detected at the candidate pixel p if the intensities of at least $n = 12$ contiguous pixels out of the 16 are all above or all below the intensity of p by a threshold, t.



(a) (b)

Figure 2: (2a) Bresenham circle of radius 3 showing the pixel access pattern for FAST around the candidate pixel P. (2b) ORB pattern positions for a rotation of 0º. Each pair of points required for an intensity test is connected by a line.

ORB-SLAM employs an additional technique. Each frame is divided into a grid of approximately 30×30 pixel tiles. In each tile of the grid, the algorithm tries to extract FAST corners using a default threshold. If no corners are found, the algorithm tries again to extract corners but now using a lower threshold.

Finally, Non-Maximal Suppression (NMS) is applied as a post-processing method that removes some corners based on

a measure or score. Only the corners with the local maxima score within a neighborhood prevail. Typically, the NMS candidate's neighborhood is defined as the area within a fixed-sized, in particular 3×3, square patch centered on the considered pixel. There are alternative definitions of the score, V, of a corner but ORB-SLAM uses the sum of absolute difference between p and the 16 surrounding pixels values used for FAST (OpenCV implementation).

C. oFAST: FAST Keypoint Orientation

ORB uses the intensity centroid [7] to compute the orientation component of FAST. The intensity centroid assumes that a corner's intensity is offset from its center, and this vector can be used to impute an orientation.

The moment of a patch can be defined as [14]:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{1}$$

where $I(x,y)$ is the intensity of the pixel at the relative position x, y within the patch and p and q are naturals indicating the moment order in each dimension. With this definition it is possible to compute the orientation centroid:

$$C = (\frac{m_{01}}{m_{00}}, \frac{m_{10}}{m_{00}}) \tag{2}$$

An then compute the angle of the vector formed between the center point of the corner, O, and the centroid C, $\vec{OC}$.

$$\theta = atan2(m_{01}, m_{10}) \tag{3}$$

$Atan2$ is the quadrant-aware version of the arctangent.

In addition, it is possible to compute the $sin(\theta)$ and $cos(\theta)$ using the moments in the following way:

$$sin(\theta) = \frac{m_{10}}{\sqrt{m_{01}^2 + m_{10}^2}}, \qquad cos(\theta) = \frac{m_{01}}{\sqrt{m_{01}^2 + m_{10}^2}} \tag{4}$$

oFAST is the combination of the segment test that determines if a pixel is a corner and the computation of the orientation. This information is used to generate a set of features.

D. Rotation-Aware BRIEF Descriptor Generation

The Rotation-Aware BRIEF (rBRIEF) descriptor [6] is a bit string description of an image patch constructed from a set of binary intensity tests. A binary test, τ, is defined by:

$$\tau(p_1, p_2) = \begin{cases} 0 & , I(p_1) < I(p_2) \\ 1 & , I(p_1) \geq I(p_2) \end{cases} \tag{5}$$

where p_1 and p_2 are two 2D points and $I(p_i)$ is the intensity of the point p_i. The feature is defined as a vector of n binary tests:

$$f_n(p) = \sum_{1 \leq i \leq n} 2^{i-1} \tau(p_{1_i}, p_{2_i}) \tag{6}$$

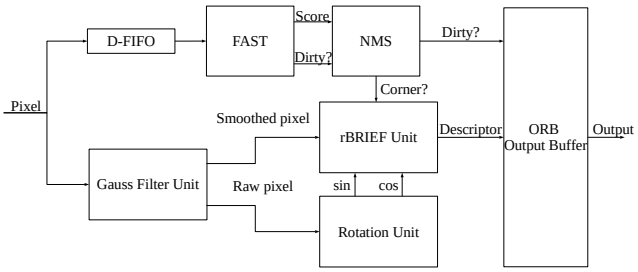


Figure 3: The architecture of the ORB accelerator.

It is recommended to smooth the image before performing these tests, for example with a Gaussian blur filter. The vector length usually is $n = 256$.

One of the most attractive features of ORB is its in-plane rotation invariance. To achieve this, the binary test coordinates of rBRIEF are rotated according to the orientation obtained previously by oFAST. To this goal, for any feature set, let us define a matrix of dimension $2 \times n$ that contain the coordinates of the n locations (x_i, y_i) that will be used by the binary tests:

$$S = \begin{pmatrix} x_1, ..., x_n \\ y_1, ..., y_n \end{pmatrix} \tag{7}$$

Using the patch orientation θ, the coordinates of the locations to be used after rotation are given by:

$$S_\theta = R_\theta S \tag{8}$$

III. HARDWARE ACCELERATED ORB

In this section, we describe the architecture of the proposed accelerator for ORB feature extraction, since this task represents the vast majority of execution time as shown in Figure 1. The rest of ORB-SLAM tasks can run on an embedded processor in parallel with the accelerator. Further details are provided in the next sections.

A. Hardware Architecture Overview

Figure 3 shows the architecture of the proposed accelerator. The overall design employs a streaming based dataflow inspired by previous works in the field [8], [9], [15], [16]. Note that our solution is significantly different from these previous proposals as explained in Section VI. A stream of input pixels, coming from memory or image sensors, is processed through the extensive use of several hardware structures that provide a sliding window access pattern, which allows an efficient exploitation of the temporal and spacial locality of the operations. The pixel stream is fed into the accelerator at a ratio of one pixel per cycle in raster scan order, which is the usual arrangement of image data. The memory bandwidth required by this stream is extremely low (381 MiB/s when the accelerator is clocked at 400 MHz).

The pixel stream follows two cooperative data paths that are responsible for feature detection and computation

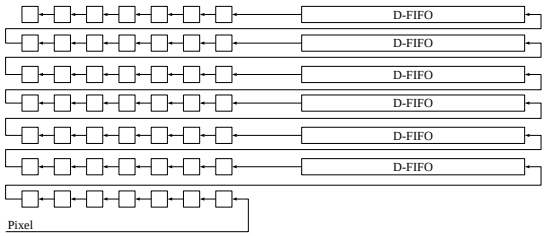


Figure 4: 7×7 sliding window structure.

of rBRIEF descriptor respectively. The first path is composed of two functional units: FAST feature detector and non-maximal suppression (NMS) unit. The second path is divided into a Gaussian image smoothing, orientation computation and rBRIEF descriptor generator units. The rBRIEF unit architecture proposed in this work, combines replication with a static pattern reordering technique. The D-FIFO allows to delay the pixel stream to keep the two paths synchronized. Note that the accelerator accesses each pixel of input images just once and it keeps synchronized all the sliding windows necessary for the correct operation of the mentioned functional units.

Another important aspect of the design is the use of tiling to reduce the on-chip memory storage while providing high performance and flexibility to process different image resolutions. Tiling segments an image into a number of smaller rectangular areas, a.k.a tiles. The accelerator is able to process each of those fixed-size tiles transparently since it is not relevant whether the input data is part of a single image or a subset of a larger one. Tiling improves locality and reduces the on-chip memory requirements in the accelerator. The tile size is an important parameter, since tiling introduces some overheads due to the required overlap among neighbor tiles, i.e. borders may be fetched multiple times for different tiles. Furthermore, tiling requires sliding window re-alignment when the window changes to the bottom row. Properly sizing the tiles largely reduces the overheads, while improving locality by a large extent.

B. Basic Sliding Window Structure

The sliding window is a structure commonly used to support 2D convolutions, particularly in image processing hardware. This structure is used as a temporary storage and synchronization mechanism, and its use has a great impact on the design. Figure 4 illustrates a sliding window similar to the ones used by the FAST and Gauss Filter units.

A sliding window stores $W + (W-1) \times Cols$ elements, where W is the square window size (7 in the example) and $Cols$ is the numbers of columns of the processed tile. The pixel stream flows through the elements that are organized according to the input data format, typically raster order.

There is a difference in the way the data is accessed depending on whether it is in the window or not. For this reason, actual implementations are composed of storage that allows direct access for the window elements (in the example of Figure 4, a set of flip-flops) and D-FIFOs for the rows.

The D-FIFO structure allows elements to leave the structure in FIFO order a number of fixed cycles after their entrance. It can be efficiently implemented through the use of a memory and a circular index pointing to the position from which to read and write. Note that this D-FIFO implementation requires just one element read and one element write per cycle.

A sliding window initially takes a fixed number of cycles until it can be used because it is necessary to fill the entire structure before computations can begin. We usually refer to this as warm-up time. Warm-up is performed only once at the beginning of each tile and represents around 9% of the processing time for the tile size selected in our tests. After warm-up, each cycle the pixels will be shifted one position, providing access to a moving window of pixels that iteratively covers every position of the image.

C. rBRIEF Unit

The rBRIEF unit is responsible for implementing the most challenging part in the ORB extraction, as it requires to provide memory accesses to 512 positions that depend on the feature angle, in order to compute the rBRIEF descriptor. The angle depends on the input data, which makes it difficult to find a good schedule for these accesses since they depend on the particular angle. Figure 2b shows a representation of the pairs of points that must be accessed in order to compute the 256 binary intensity tests (each test entails comparing two pixels) for a rotation of 0º.

Figure 5 shows the basic architecture of the rBRIEF unit. The implementation employs a 37×37 sliding window that holds the data required to perform the tests (Eq. 6) of each feature point candidate. This window is synchronized with the rest of units so that it holds a patch centered into the feature candidate. The structure must support the sliding window dataflow mechanism and at the same time a random access to 512 points to generate the descriptor. The baseline implementation allows one pair of accesses (two points) per cycle, meaning that a total of 256 cycles are needed in order to generate a descriptor. This design has a significant penalty in performance because of the imbalanced latencies of FAST Feature Extraction and rBRIEF descriptor generation. Such a bottleneck makes it difficult to meet real-time requirements or forces to drop features, which ultimately would have negative effects on the operation of the ORB-SLAM algorithm.

To mitigate this bottleneck, replication of the window is typically performed, with each of the replicas computing descriptors of different features in parallel. An Arbitrer decides how to distribute the descriptor requests among the windows sending the rotated coordinates to a buffer that holds this information until the process finishes. The Coordinate Rotation module computes the rotated coordinates

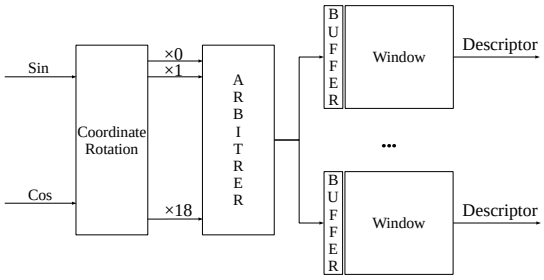


Figure 5: rBRIEF Unit architecture overview.

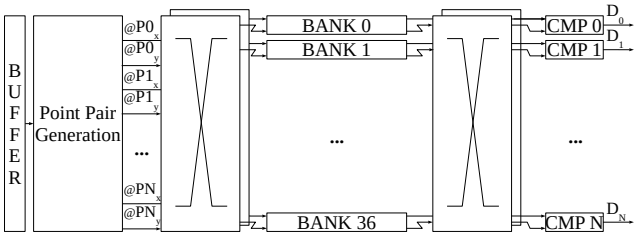


Figure 6: ORB Window architecture that allow parallel access to multiple pairs per cycle.

for the tests. Note that coordinates values can take only 36 different values since the coordinates are expressed taking the center of the patch as reference (Coordinate components $x, y \in [-18, 18]$). The required multiplications are computed using a Multiplierless Constant Multiplication approach based on [17], obtaining an efficient implementation that utilizes just additions and shifts.

The ORB pattern is stored in a LUT codified as a sequence of indexes to select one of the 19 possible rotated coordinates from the Window buffer and a bit indicating the sign.

We opt to design a unit that does not drop any features for the sake of keeping the original accuracy. This unit is the main accelerator pipeline bottleneck and, hence, latency of the rBRIEF unit has a direct impact in overall latency of the ORB accelerator and thus it is crucial to find an efficient implementation. If there was no bottleneck due to this unit, the accelerator could process the tiles with a throughput of one pixel per cycle, obtaining a speedup of 1.25x compared to the accelerator version with bottlenecks.

Replication is an effective way to reduce the rBRIEF bottleneck but it has a relevant impact on the area and power consumption. For this reason, we argue that it is necessary to consider alternatives that deliver the required performance at a much lower cost.

1) Exploiting Parallelism: As stated above, one way to reduce the latency of the descriptor generator consists in replicating the data allowing parallel access to each copy. Replicas can be used to reduce the latency of one descriptor generation or can increase the throughput dedicating each replica to a different feature point candidate. Our experiments show that the latter is more effective because it enables a better exploitation of the parallelism available between the FAST and rBRIEF data paths. FAST features typically are very sparse and NMS reduces its density, therefore, allowing FAST unit to continue processing the tile while the rBRIEF Unit is processing one or more descriptors is beneficial. In case the replicas were used to compute a single descriptor, the FAST unit must block every time a feature is found.

However, it is possible to increase the granularity of the exploited parallelism since the computation of each intensity test is completely independent of the rest. Figure 6 illustrates the basic architecture that allows each patch replica to access more than one pair per cycle. The proposed design consists of a streaming-friendly architecture that segments the storage structure for each row in a separated bank of memory with two read ports. We decided this number of ports to reduce the conflicts taking into account that two points from the same row could be required for a particular intensity test. In this way, conflicts of a pair with itself are avoided. A custom interconnection network is needed to route each point pair to the appropriate bank based on its coordinates. In addition, another interconnection network is required to gather the results and perform the intensity tests at the other end.

With this approach, we could potentially access to 37×2 points or 37 pairs in one cycle. However, the cost of the interconnection and routing for such solution is prohibitive. Instead, we propose a design that allows parallel access to a group of pairs of a given size each cycle. The appropriate group size is experimentally determined in Section V.

Conflicts between pairs that, for a given rotation angle, need to access to the same row, are the main problem that arises in that solution. We define this as a structural hazard or conflict that the custom interconnection network must resolve. Furthermore, we put a constraint that consists in pinning each point of a pair always to the same port of every bank. This reduces the complexity of the interconnection infrastructure as the operands required for the intensity tests come one from each port of the banks. Taking into account these restrictions, it is possible to design a control unit that detects conflicts between pairs and orchestrates a sequential access to the row resource. This implies that the unit needs to introduce a stall to serialize the access of conflicting pairs. Non-conflicting pairs can be routed directly to the appropriate row bank. The output of each row bank is routed to a register that holds each operand of the intensity test until all points are read from the patch and all the tests can be performed.

The rest of the optimizations detailed in the following sections are focused on reducing the number of conflicts.

2) Reducing Conflicts: Static Pattern Reordering: The number of conflicts varies according to the order of the ORB pattern as this affects the static scheduling and composition of the pair groups. Rearranging statically the order of the

pattern does not cause any issues with the quality of the descriptor, since Hamming distance defines the metric space of the ORB descriptor set [7]. As long as the same order is maintained across different features, the metric space will remain intact. Finding an optimal order of the pairs that minimizes the number of conflicts when accessing to the banks of the proposed architecture (Figure 6) is an NP-Hard problem. For this reason, we propose to compute a static scheduling based on an optimization performed with a genetic algorithm (GA).

The Static Pattern Schedule problem consists of a set of pairs P:

$$P = \{P_1, P_2, ..., P_N\},$$

and a set of groups G:

$$G = \{G_1, G_2, ..., G_{\frac{N}{gsize}}\},$$

where *gsize* is a fixed parameter that indicates the size of the group of pairs that can access the banks concurrently. An assignment is represented by a tuple $< P, G >$ and a solution consists of an assignment for every element of P.

Furthermore, we have the following set of constraints:

1) No pair can be in more than one group.
2) All groups must have *size* pairs assigned.

On the other hand, the objective function, F, is defined as the average latency of the unit for every possible rotation angle using the set of assignments as a static scheduling. The objective function includes additional constraints derived from the characteristics of the proposed architecture (e.g. port pinning of pair points). Note that this objective function does not depend on the contents of the patch and we assume an equiprobable distribution of angles. In addition, we know that the number of plausible angles is bounded. According to OpenCV documentation [18], the *fastatan2*, used in ORB-SLAM, uses a precision of "about 0.3 degrees".

In order to apply a GA approach as a meta-heuristic, we need to encode a candidate solution as a chromosome representation and define the Initialization and genetic operators for Fitness Evaluation, Selection, Crossover and Mutation. The chromosome chosen to represent a Static Schedule solution is a one dimensional array A_i where $0 \leq i < 256$ such that each element of the array represents an element of P. The group assignment is determined by A_i, the position within the array, as: $\lfloor \frac{i}{gsize} \rfloor$. This representation is chosen because the constraints are automatically satisfied by construction.

The initial population is generated by choosing random permutations of P. The genetic operators applied in each generation to this initial population are:

- Fitness Evaluation: Fitness in biological sense is a quality value which is a measure of the reproductive efficiency of chromosomes. Because our goal is to minimize the number of cycles we use as a fitness function the negation of the objective function, $-F$.

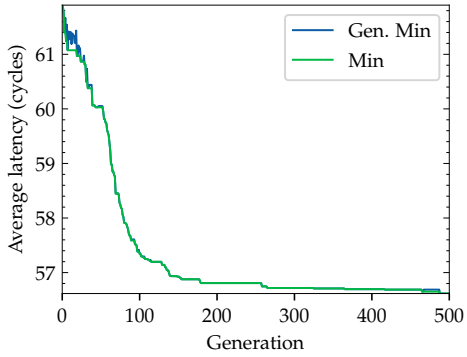


Figure 7: Example of convergence of the genetic algorithm to a local minimum for a *gsize* = *8*.

- Selection: Individual solutions are selected based on its Fitness Evaluation. Fitter solutions are more likely to be selected trying to spread the best chromosomes to the off-springs. Among all selection operator alternatives, we choose Tournament selection. Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population. The winner of each tournament, defined as the individual with the best fitness, is selected for crossover.
- Crossover: Partially Matched Crossover (PMX) is chosen. This recombination operator generates two off-springs by matching pairs of values in a certain range of the two parents and swapping the values of those indexes [19]. The off-springs hopefully contain the best pair ordering of the two parents.
- Mutation: The mutation is performed each generation shuffling each chromosome of individuals with a given probability. The mutation swap pairs between groups.

The parameters used for the GA are summarized in Table I. Figure 7 shows the evolution of the fitness of the best individual of all generations and the fitness of the best individual of the current generation. In the example the GA converges to a local optimum in 500 iterations. A 18% reduction in latency is obtained with respect to a random ordering. The optimization process takes in the order of hours using an AMD Opteron 6338P with 24 threads. The results of the optimization are detailed in the Section V.

Table I: Parameters used for the GA optimization of conflicts.

Parameter	**Value**
Crossover probability	70%
Mutation probability	20%
Population size	300
Crossover Operator	Partially Matched Crossover
Mutation Operator	Partial Shuffle Mutation
Selection Operator	Tournament Selection

3) Reducing Conflicts: Point Intensity Bypass: An additional approach proposed to reduce conflicts is based on exploiting the potential temporal locality of the points of the ORB pattern as there are many points that are accessed more than one time (i.e. not all points of the original pattern are unique). The original ORB pattern is composed of 375 unique points out of the 512 total points. We investigate an improvement of the previous design leveraging this observation that we call Point Intensity Bypass.

Point Intensity Bypass consists in a reformulation of the definition of a conflict that takes into account the reuse of points. In this way, two pairs of a group are in conflict if they access the same bank but to a different address within it. If the address is the same, that means that they need to read the same point and we can save one access. To implement this alternative, the control of the unit must consider both point coordinates when detecting conflicts. If two pairs access to the same coordinates, the control does not consider that as a conflict and simply suppresses one of the requests. A modification of the output routing must be done to send the read value to two output registers of the associated pairs.

D. FAST Detector Unit

Figure 8 illustrates the architecture of the FAST Unit. The module employs a Sliding Window structure with a patch size of 7×7, achieving a throughput of a pixel tested per cycle. The FAST implementation used by ORB-SLAM applies an adjustment of the threshold at run-time increasing the sensibility if no features are found inside a region of size 30×30. The algorithm uses two thresholds, one for the default feature extraction (*IniThr*) and another with higher sensitivity (*MinThr*).

The accelerator detects in parallel corners with the two thresholds. The module detects speculatively corners with the *MinThr* until at least one corner of *IniThr* is found (if any) inside each region. While no corners or *MinThr* corners are detected, the accelerator works under normal operation generating ORB descriptors for such corners. Those corners are stored in the ORB Store Buffer until the region is completed. If at least one corner with the default threshold is found, the descriptors generated in that region with the *MinThr* are discarded. The Dynamic Threshold module keeps track of the status of each region and is responsible to set the dirty bit high when an *IniThr* corner is detected.

In order to perform the segment test and score computation, we use a similar solution as other works [20]. Each pixel intensity of the Bresenham circumference is compared with the central pixel obtaining a 16 bit string. To classify a pixel as a corner it is necessary to search for a sub-string with 12 consecutive set bits. This can be done efficiently by using an AND tree with a depth of ten levels. Finally, an OR reduction of the bits of the last level of the tree is needed to determine if any of the searched patterns is found.

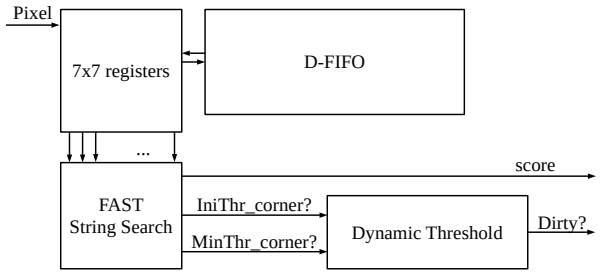


Figure 8: Architecture of the FAST unit.

E. Non-Maximal Suppression Unit

A 3×3 Sliding Window is used to filter the FAST features. The sliding window is fed with the FAST scores. Each cycle, the center pixel of the window is compared with the eight surrounding pixels to determine if it is the local maximum. This operation is implemented with eight comparators and an AND reduction.

F. Gauss Unit

In order to generate the feature descriptor, the patch around the feature point must be smoothed before computing the intensity tests. The Gauss Filter Unit applies this smoothing to every pixel of the image achieving a throughput of one filtered pixel per cycle.

The unit employs a Sliding Window similar to Figure 4. Each cycle a pixel flows through the structure. The convolution is performed multiplying each element of the window by the corresponding element of the Gaussian kernel and adding all the products.

We employ fixed point arithmetic to represent the values of the Gaussian kernel and intermediate results before rounding to obtain the filtered value. Fixed point arithmetic allows the use integer functional units while keeping accuracy under control. The unit generates every cycle a pixel and its Gaussian smoothed version.

G. Rotation Unit

The rotation unit is in charge of calculating the sine and the cosine of the angle that the centroid of the patch centered at a candidate feature point forms. This angle is required to rotate the coordinates of the ORB pattern applying Equation 8. The precision of the calculations is key since errors in the rotation of the points in the ORB pattern produce severe degradation of the quality of the generated descriptors. We propose the use of the inverse square root to compute it instead of a LUT of precomputed values used in state-of-the-art solutions [9], [16], [21]. The unit uses a 37×37 Sliding Window that receives the raw pixel stream from the previously described Gauss Unit, meaning that this stream is synchronized with the rBRIEF Unit input stream. In addition, the unit is pipelined into several stages as shown in Figure 9. The unit can compute an accurate estimation of the sine and cosine of a patch per cycle.

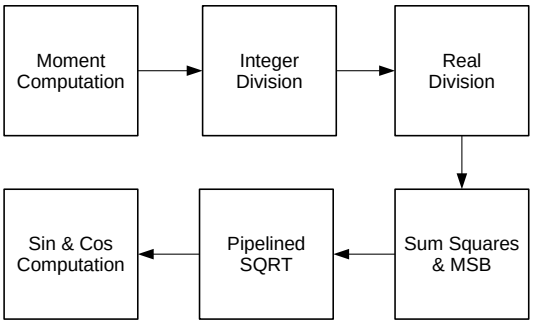


Figure 9: Pipelined architecture of the Rotation Unit.

The first pipeline stage of the Rotation Unit computes the moment of a window. To do so efficiently, Equation 1 can be reformulated as follows:

$$m_{01_{n+1}} = m_{01_n} + 18 \times (C_n + C_{n-37}) - S_n$$
$$m_{10_{n+1}} = m_{10_n} + xC_n - xC_{n-37}$$
$$m_{00_{n+1}} = m_{00_n} + C_n - C_{n-37} \quad (9)$$

where,

$$S_{n+1} = S_n + C_n - C_{n-36}$$
$$C_n = \sum_{1 \le x \le 37} p_{x,n}$$
$$xC_n = \sum_{1 \le x \le 18} x \times (p_{38-x,n} - p_{x,n}) \quad (10)$$

and $p_{x,n}$ is the intensity of the pixel at the coordinates within the patch indicated by row x and col n. The moment computation consists of an adder tree used to compute C_n. Furthermore, an optimized Multiple Constant Block [17] (MCM) is used to compute xC_n.

The next two stages perform the centroid division. Image moments, m_{01} and m_{10}, computed in the previous stage could be directly used to determine the trigonometric functions. However, we divide these values by m_{00}. This reduces the number of bits to hold the value and, therefore, the cost of its manipulation in later stages. Efficient division circuits are employed for integer and fixed-point division, with a precision of $\frac{1}{32}$.

The next stage is the summation of squares and estimation computation. A first estimation of the inverse square root is performed considering the Most Significant Bit (MSB) of the previous square sum. This results in a good estimation considering the following:

$$log_2(\frac{1}{\sqrt{x}}) = -\frac{1}{2} log_2(x) \quad (11)$$

Next, the fast inverse square root is computed in three stages. The inverse square root of x, the summation of squares previously determined, is computed using an approximation based on the Newton–Raphson method. Three iterations of the following formula are employed using as y_0 the MSB-based estimation:

$$y_{n+1} = y_n(\frac{3}{2} - \frac{x}{2}{y_n}^2) \quad (12)$$

The final stage computes the sine and cosine applying Equation 4 and updating the signs of the final values.

IV. EVALUATION METHODOLOGY

We have developed Register-Transfer-Level (RTL) models of the ORB accelerator described in Section III by leveraging PyMTL [22] framework. We tested different versions of the architecture varying the number of rBRIEF window replicas and comparing the results with the improved architecture and the modifications detailed in Section III-C3 and Section III-C2. Table II shows the parameters employed in the experiments. In order to estimate area and critical path delay, we translate the PyMTL models into Verilog and synthesize them using Yosys [23] with the open-source 45 nm FreePDK45 1.4 [24]. Moreover, we obtain the energy consumption of the gate-level netlist of the accelerator using Synopsys Design Compiler [25].

As the software baseline we use an open source ORB-SLAM implementation [10]. We measure the performance of this implementation on a CPU with parameters shown in Table III. We use Intel RAPL [26] to measure CPU energy consumption.

On the other hand, the experimental evaluation is performed using the KITTI dataset [12]. We use the odometry benchmark that comprises various recordings from drivings around the city of Karlsruhe. This benchmark consists of 22 grayscale stereo sequences. In particular we use the sequence 0, which comprises a set of 4541 frames with a resolution of 1241×376 pixels.

Finally, we use an evolutionary computation framework for rapid prototyping and testing of ideas called DEAP [27] to implement the genetic algorithm described in Section III-C2.

Table II: Hardware parameters for the accelerator.

Parameter	Value
Technology, Frequency	45 nm, 400MHz
Tile width	210
Target number of features	2000

Table III: CPU parameters.

Parameter	Value
CPU	Intel(R) Core(TM) i7-7700K
Number of cores / threads	4 / 8
Technology, Frequency	14 nm, 4.2 GHz
L1, L2, L3	0.25 MiB, 1 MiB, 8 MiB

V. EXPERIMENTAL RESULTS

In this section, we evaluate the performance and energy consumption of the CPU and different versions of the ORB accelerator presented in Section III. The configuration labeled as *CPU* corresponds to a high-performance software implementation of ORB feature extraction running on a CPU with parameters shown in Table III. On the other hand, configurations labeled with *ASIC* represent different versions of the accelerator. We use the following nomenclature: *ASIC-GN-RM*, where N indicates the group size for BRIEF descriptor computation and M shows the degree of replication. For example, *ASIC-G4-R2* indicates a configuration of the ORB accelerator with a group size of 4, i.e. 4 bits of the BRIEF descriptor are computed at a time, whereas the entire BRIEF unit is replicated 2 times. If the group size is greater than one, a static scheduling of the 256 pairs of pixels is employed in order to minimize the number of bank conflicts, using a genetic algorithm as described in Section III-C2.

Figure 10a shows the speedup achieved by the accelerator with respect to the *CPU*. All systems achieve real-time performance. Configurations *ASIC-G1-R4* and *ASIC-G1-R8* achieve speedups of $4.8\times$ and $8.1\times$ respectively. On the other hand, *ASIC-G4-R1* and *ASIC-G8-R1* deliver $4.37\times$ and $5.24\times$ speedups respectively. Finally, *ASIC-G8-R2* obtains a $7.8\times$ speedup. The performance of the accelerator is higher as it has a pipeline tailored to the requirements of the ORB feature extraction algorithm. Increasing the group size and rescheduling the pixel pairs for BRIEF computation based on the static ordering provides significant benefits, delivering performance comparable to configurations with more replicated hardware.

The accelerator provides a significant reduction in power dissipation as illustrated in Figure 10b. The results include both static and dynamic power. Configuration *ASIC-G1-R4* achieves a reduction in power dissipation of $1690\times$ compared to the software-based solution, whereas *ASIC-G8-R2* achieves a $1957\times$ power reduction. This huge power reduction is due to several reasons. First, the accelerator includes a specifically designed streaming architecture for ORB extraction that exhibits a high throughput and a large data reuse. Second, the improved BRIEF unit with the static scheduling further improves power dissipation by reducing the required on-chip structures, avoiding conflicts between pairs of pixels, decreasing the underlying data movements. *ASIC-G8-R2* consumes 9.9% less energy per frame on average than *ASIC-G1-R8*.

Figure 11 reports the effectiveness of the genetic algorithm (Section III-C2) to reduce bank conflicts. The figure shows the number of penalty cycles due to bank conflicts. To compute the extra cycles, we define a lower bound for the latency estimated as the number of accesses of the bank with the most accesses (critical) and assuming that the rest of the accesses can be done in parallel. The lower

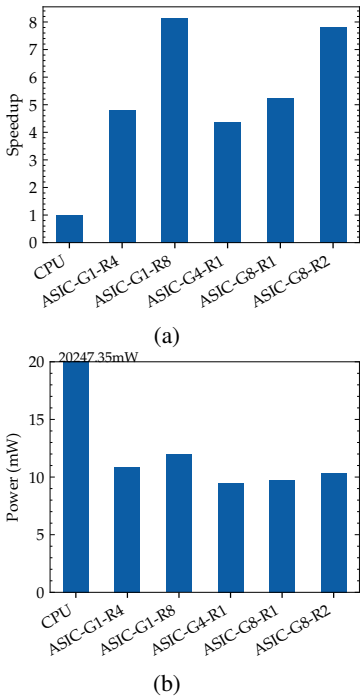


Figure 10: (10a) Speedup achieved by the accelerator compared with the CPU. (10b) Power dissipation of the CPU and the accelerator.

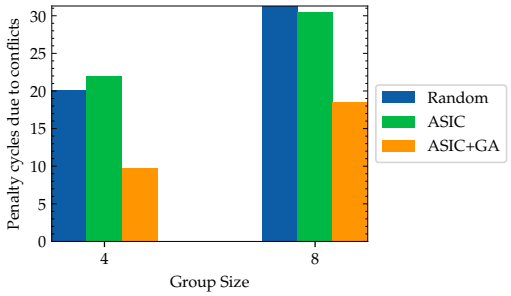


Figure 11: Average number of penalty cycles due to bank conflicts when computing an rBRIEF descriptor, with and without the GA optimization.

bound is not guaranteed to be a feasible global optimum, but it allows us to have a reference of how much room for improvement could be. Our static scheduling technique (ASIC+GA) reduces the penalty cycles due to conflicts by 51.8% and 40.9% for group sizes of 4 and 8 pairs respectively, as shown in Figure 11.

To sum up, the experimental results show the benefits of the proposed architecture and the technique to optimize the rBRIEF ordering. *ASIC-G8-R2* is the best configuration tested in our experiments, since its performance is similar to a configuration with a higher degree of hardware replication, while it achieves significant lower power and area footprint. In this configuration, the use of the pattern ordering obtained

Table IV: Comparison with previous works. PPW stands for Performance Per Watt.

Work	Algorithm	Streaming	Implementation	Performance	Power (mW)	PPW
[28]	FAST-BRIEF	No	ASIC, 130nm, 78.3k gates, 128kB SRAM	122fps, FHD, 200MHz	182	670
[29]	ORB-like	No	ASIC, 65nm, 127k gates, 205kB MEM	135fps, FHD, 200MHz	87.5	1542
[16]	ORB	Yes	FPGA, Arria V GX, 449 DPS, 206000 LEs, 231973 REGs, 1047kB BRAM	110.9fps, FHD, 230MHz	5340	20
[30]	ORB	Yes	FPGA, Stratix V, 8 DPS, 25648 LUTs, 21791 REGs, 1208kB BRAM	67fps, VGA, 203MHz	4559	14
[15]	FAST-BRIEF	Yes	ASIC, 65nm, 28kB SRAM	2170fps, VGA	1131	1918
[9]	ORB	No	FPGA, XCZU9EG, 33 DPS, 28168 LUTs, 9528 REGs, 188kB BRAM	108fps, FHD, 200MHz	873	123
[8]	FAST+RS-BRIEF	No	FPGA, XCZ7045, 111 DPS, 56954 LUTs, 67809 REGs, 78 BRAM	55.87fps, VGA, 100MHz	1963	28
This work	**ORB**	**Yes**	**ASIC, 45nm, 32kB SRAM**	**120fps, FHD, 400MHz**	**10.34**	**11605**

with the GA technique translates into a 7.2% reduction in overall execution time compared with the use of the original pattern ordering. A corresponding reduction in energy consumption is also achieved, taking into account that the power dissipation is not affected by the static reordering.

A. Comparison with Previous Works

Table IV provides a quantitative comparison with previous works. Our solution achieves high performance and shows the best energy efficiency, achieving a large improvement in performance/W. The next section provides a qualitative comparison with related works.

VI. RELATED WORK

Prior research used stream-based implementations on FPGAs. Work in [16], [31] propose a streaming architecture similar to *ASIC-G1-R4* to extract ORB features using Harris-Stephens corners. Moreover, they propose an architecture for multilevel feature extraction with a replicated version of the accelerator per pyramid level. The rBRIEF bottleneck is solved through the use of replicas with a different replication factor depending on the pyramid level. This solution employs an angle discretization of 64 values per sector. Our solution is different as it is based on an ASIC instead of FPGA, and we avoid angle discretization to preserve accuracy.

Work in [21] propose a streaming architecture for rBRIEF on FPGA, leveraging replication of window buffers to reduce latency of descriptor generation. Our proposal avoids replication by a large extent, and it exploits parallelism in the computation of the rBRIEF descriptor by processing multiple pairs of pixels at a time.

An SLAM accelerated solution is introduced in [8], proposing an architecture for feature extraction and matching on an FPGA while the rest of the components of SLAM run on a CPU. Authors propose a hardware-friendly pattern to generate BRIEF descriptors. This pattern reduces the rotation operation to a bit vector rotation operation instead of a costly trigonometric calculation. The drawback of this approach is the accuracy degradation and the unpredictable effects derived from changing the functional properties of rBRIEF.

Another architecture for ORB extraction is proposed in [9], [20]. The architecture comprises a streaming front-end to generate the image scale pyramid and feature detection. The rBRIEF generation is carried out by a non-streaming back-end. This back-end consists of a four issue super-scalar architecture that dynamically schedules points to compute the descriptor bits. However, a large discretization of the angles is used. Our solution is different as we do not sacrifice accuracy to simplify the hardware implementation, and we solve the issues with BRIEF computation by using a static ordering found with a genetic algorithm that minimizes bank conflicts.

VII. CONCLUSIONS

In this paper we propose a low-power and high-performance accelerator for ORB feature extraction, a key component of camera-based self-driving cars. Unlike previous proposals, our solution achieves the same accuracy of reference software implementations, avoiding accuracy loss for the sake of simpler hardware. Furthermore, we propose a novel solution to implement rBRIEF descriptor computation in hardware. Our system evaluates multiple pairs of pixels at a time, and it reorders the pairs of pixels based on a static scheduling that minimizes the bank conflicts for any angle. The static reordering is found offline by using a genetic algorithm. Our experimental results show that the proposed accelerator achieves a speedup of $7.8\times$ and a reduction in power dissipation of $1957\times$ with respect to a high-end CPU.

ACKNOWLEDGMENTS

This work has been supported by the CoCoUnit ERC Advanced Grant of the EU's Horizon 2020 program (grant No 833057), the Spanish State Research Agency (MCIN/AEI) under grant PID2020-113172RB-I00, the ICREA Academia program and the FPU grant FPU18/04413.

REFERENCES

[1] J. Fuentes-Pacheco, J. Ruiz-Ascencio, and J. M. Rendón-Mancha, "Visual simultaneous localization and mapping: a survey," *Artificial Intelligence Review*, vol. 43, no. 1, pp. 55–81, 2015.

[2] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[3] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 751–766, 3 2018. [Online]. Available: https://doi.org/10.1145%2F3296957.3173191

[4] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 10 2017. [Online]. Available: https://doi.org/10.1109%2Ftro.2017.2705103

[5] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[6] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*. Springer, 2010, pp. 778–792.

[7] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[8] R. Liu, J. Yang, Y. Chen, and W. Zhao, "Eslam: An energy-efficient accelerator for real-time orb-slam on fpga platform," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[9] R. Sun, J. Qian, R. H. Jose, Z. Gong, R. Miao, W. Xue, and P. Liu, "A flexible and efficient real-time orb-based full-hd image feature extraction accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[10] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[11] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[12] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[13] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking." in *ICCV*, vol. 2. Citeseer, 2005, pp. 1508–1515.

[14] P. L. Rosin, "Measuring corner properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, 1999.

[15] S.-K. Lam, G. Jiang, M. Wu, and B. Cao, "Area-time efficient streaming architecture for fast and brief detector," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 2, pp. 282–286, 2018.

[16] J. Weberruss, L. Kleeman, D. Boland, and T. Drummond, "Fpga acceleration of multilevel orb feature extraction for computer vision," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[17] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms (TALG)*, vol. 3, no. 2, pp. 11–es, 2007.

[18] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[19] D. E. Goldberg, R. Lingle *et al.*, "Alleles, loci, and the traveling salesman problem," in *Proceedings of an international conference on genetic algorithms and their applications*, vol. 154. Lawrence Erlbaum, Hillsdale, NJ, 1985, pp. 154–159.

[20] R. Sun, P. Liu, J. Wang, C. Accetti, and A. A. Naqvi, "A 42fps full-hd orb feature extraction accelerator with reduced memory overhead," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 183–190.

[21] T. H. Pham, P. Tran, and S.-K. Lam, "High-throughput and area-optimized architecture for rbrief feature extraction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 4, pp. 747–756, 2018.

[22] D. Lockhart, G. Zibrat, and C. Batten, "Pymtl: A unified framework for vertically integrated computer architecture research," in *47th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, 12 2014, pp. 280–292.

[23] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[24] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh *et al.*, "Freepdk: An open-source variation-aware design kit," in *2007 IEEE international conference on Microelectronic Systems Education (MSE'07)*. IEEE, 2007, pp. 173–174.

[25] "Synopsys," https://www.synopsys.com/, accesed: 2019-10-16.

[26] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with papi," in *2012 41st international conference on parallel processing workshops*. IEEE, 2012, pp. 262–268.

[27] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[28] J.-S. Park, H.-E. Kim, and L.-S. Kim, "A 182 mw 94.3 f/s in full hd pattern-matching based image recognition accelerator for an embedded vision system in 0.13-um cmos technology," *IEEE transactions on circuits and systems for video technology*, vol. 23, no. 5, pp. 832–845, 2012.

[29] W. Zhu, L. Liu, G. Jiang, S. Yin, and S. Wei, "A 135-frames/s 1080p 87.5-mw binary-descriptor-based image feature extraction accelerator," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 8, pp. 1532–1543, 2015.

[30] W. Fang, Y. Zhang, B. Yu, and S. Liu, "Fpga-based orb feature extraction for real-time visual slam," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 275–278.

[31] J. Weberruss, L. Kleeman, and T. Drummond, "Orb feature extraction and matching in hardware," in *Australasian Conference on Robotics and Automation*, 2015, pp. 2–4.