Optimal Memory Allocation and Power Minimization for FPGA-Based Image Processing

Paulo Garcia, Deepayan Bhowmik, Robert Stewart, Andrew Wallace and Greg Michaelson

***Abstract*—Memory constraints are the biggest limiting factors in the widespread use of FPGAs for complex and global image processing algorithms, which require one or more complete frame(s) to be stored in situ. Since FPGAs have limited on-chip memory, external memory is often used to accommodate frame data, at the cost of performance, size and power. In this paper, we propose methods for generating FPGA memory architectures from both Hardware Description Languages and High Level Synthesis designs that minimize memory usage and power consumption.**

Based on a formalization of on-chip memory configuration options and a power model, we demonstrate partitioning algorithms that outperform traditional strategies. Compared to commercial FPGA synthesis and High Level Synthesis tools, our results show that our approach can result in up to 60% higher utilization efficiency, increasing the size and/or number of frames that can be accommodated, and reducing frame buffers dynamic power consumption by up to 70%. Using Optical Flow and MeanShift Tracking as representative high-level algorithms, the total power consumption is reduced by up to 25% and 30% respectively, without any performance degradation.

***Index Terms*—field programmable gate array (FPGA), memory, power, image processing, design**

I. Introduction

ADVANCES in Field Programmable Gate Array (FPGA) technology [1] have made them the *de facto* implementation platform for a variety of computer vision applications [2]. Several algorithms, e.g., stereo-matching [3], are not feasibly processed in real-time on conventional general purpose processors and are best suited to hardware implementation [4]. The absence of a sufficiently comprehensive, *one size fits all* hardware pipeline for the computer vision domain [5] motivates the use of FPGAs in a myriad of computer vision scenarios, especially in applications where processing should be performed *in situ*, such as in smart cameras [6], where FPGAs embed data acquisition, processing and communication subsystems. Adoption of FPGA technology by the computer vision community has accelerated during recent years thanks to the availability of High Level Synthesis (HLS) tools which enable FPGA design within established software design contexts.

However, since FPGAs have limited on-chip memory capabilities (e.g., approx. 6MB of on-chip memory on high end Virtex 7 FPGAs), external memory (i.e., DDR-RAM chips connected to the FPGA) is often used to accommodate frames [7], [8]. This causes penalties on *performance* (latency is much higher for off-chip memory access) and perhaps more importantly, on *size* (two chips, FPGA and DDR, rather than just FPGA), *power* (DDR memories are power hungry [9]) and have associated monetary costs, hindering the adoption of FPGAs.

In this paper, we research allocation of on-chip memory resources in order to minimize resource usage and power consumption, contributing to the realization of power-efficient high-level image processing systems fully contained on FPGAs. We propose methods for generating on-chip memory architectures, applicable from both HLS and HDL designs, which minimize FPGA memory resource usage and power consumption for image processing applications. Specifically, this paper offers the following contributions:

- A formal analysis of on-chip memory allocation schemes and associated memory usage for given frame sizes and possible on-chip memory configurations.
- Methods for selecting a memory configuration for optimal on-chip memory resource usage and balanced usage/power for a given frame size.
- A theoretical analysis of the effects on resource usage and power consumption of our partitioning methods.
- Empirical validation of resource usage, power and performance of the proposed methods, compared to a commercial HLS tool.

The remainder of this paper is organized as follows: Section II describes related work within FPGA memory systems architecture and design for image processing. In Section III, we formally describe the research problem of power-size optimization and present a motivational example that highlights the limitations of standard HLS approaches. Section IV presents our proposed partitioning methods. Section V describes our experimental methodology and experimental results, which are discussed in Section VI. Finally, Section VII presents our concluding remarks.

Throughout this paper, we use the term BRAM, a Xilinx nomenclature for on-chip memories, to refer to on-chip FPGA memories in general.

II. Background and Related Work

Within FPGA processing sub-systems, algorithms evolve from typical software-suitable representations into more hardware-friendly ones [5], [10] which can fully exploit data

P. Garcia and A. Wallace are with the School of Engineering and Physical Sciences, Heriot-Watt University, Edinburgh EH14 4AS, U.K. E-mail: {p.garcia, a.m.wallace}@hw.ac.uk.

D. Bhowmik is with the Department of Computing, Sheffield Hallam University, Sheffield, U.K. E-mail: {deepayan.bhowmik}@shu.ac.uk.

R. Stewart and G. Michaelson are with the School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh EH14 4AS, U.K. E-mail: {r.stewart, g.michaelson}@hw.ac.uk.

Manuscript received XXX XX, XXXX; revised XXX XX, XXXX.

parallelism [10] through application-specific hardware architectures [3], often substantially different from the traditional Von Neumann model, such as dataflow [11] or biologically inspired processing [12]. These heterogeneous architectures are customized for FPGA implementation not just for performance (e.g., by exploiting binary logarithm arithmetic for efficient multiplication/division [13]), but also for power efficiency (e.g., by static/dynamic frequency scaling across parallel datapaths for reduced power consumption [14]).

More often than not, computer vision applications deployed on FPGAs are constrained by performance, power and real-time requirements [3]. Real time streaming applications (i.e., performing image processing on real-time video feeds [5]) require bounded acquisition, processing and communication times [14] which can only be achieved, while maintaining the required computational power, through exploitation of data parallelism [10] by dedicated functional blocks [6].

However, the greatest limiting factor to the widespread use of FPGAs for complex image processing applications is memory [8]. Algorithms that perform only point or local region operators (e.g., sliding window filters) [13] are relatively simple to implement using hardware structures such as line buffers [3]. However, complex algorithms based on global operations require complete frame(s) to be stored *in situ* [10]; examples of contemporary applications that require global operations are object detection, identification and tracking, critical to security. Notice we use the term "global operations" to simultaneously refer to two characteristics: the use of *global operators* (atomic operations which require the whole image, such as transposition or rotation) and *undetermined* (unpredictable) access patterns (e.g., a person identification system might only need a subset of a frame, but which subset cannot be decided at design time, as it depends on person location at runtime).

A possible approach is to refine image processing algorithms so they can perform on smaller frame sizes that can be contained on an FPGA [2]. Several algorithms maintain robustness for downscaled images [15], e.g., the Face Certainty Map [16]) or employ intelligent on-chip memory allocation schemes [7] to accommodate complete frames that take into account power profiles. The latter requires methods to optimize on-chip memory configurations in order to maximize valuable usage; often at odds with performance-oriented allocation schemes standard in HLS code generators. Other possible approaches include stream-processing algorithm refactoring to minimize memory requirements [17] or programming-language abstractions for efficient hardware pipeline generation [18]; these are orthogonal to our approach, and outside the scope of this work.

In our context, the most significant related work on the use of FPGA on-chip memory for image processing applications has focused on four aspects: processing-specific memory architectures, caching systems for off-chip memory access, partitioning algorithms for performance and on chip memory power reduction.

A. Processing-specific memory architectures

Memory architectures specialized for specific processing pipelines typically exhibit poor BRAM utilization. Torres-Huitzil and Nuno-Maganda [8] presented a mirrored memory system: in order to cope with dual access required by computational datapaths; data is replicated in two parallel memories and a third one is used for intermediate computations. The need for data replication to support paralellism inhibits scaling for higher frame sizes. Mori et al [19] described the use of neighbourhood loader: input pixels are fed to shift registers which de-serialize the input stream into a neighbourhood region. Their approach supports only one output port, and sequential region read (no random access). This approach does not exploit datapath parallelism, nor does it support classes of algorithms which require disparate region access. Chen et al [20] use distributed data buffers for expediting Fast Fourier computations; they partially exploit spatial parallelism, focusing on time-multiplexing as a means for reducing resource-usage and power consumption. Although time-multiplexing is a convenient technique for certain classes of applications, it cannot be used in real-time streaming where input pixels arrive at steady rates (without discarding frames). Klaiber et al [21] have developed a distributed memory that divides input frames into vertical regions stored in separate memories. Their approach allows fine grained parallelism, but is only capable of handling single-pass algorithms, i.e., which do not require storage of intermediate values. While this suffices for simple computations, it does not satisfy the requirements of sophisticated computer vision algorithms which process data iteratively (e.g., MeanShift Tracking [22]).

B. Caching systems

Delegating frame storage to off-chip memory solves the capacity problem, at the cost of performance and monetary expense. Caching techniques are used to minimize the performance implications: e.g, Sahlbach et al [23] use parallel matching arrays for accelerating computation; however, each array is only capable of holding one row of interest (the complete frame is stored in off-chip memory) and their results do not discriminate resource usage across modules, making it hard to estimate the precise array costs. This approach can only support a limited class of algorithms: column-wise operations, for instance, require off-chip memory re-ordering for data to be loaded on-chip as rows, consuming precious processing time. Similarly, Chou et al [24] have shown the use of vector scratchpad memories for accelerating vector processing on FPGAs, but still rely on random-access external memories; a similar approach is followed by Naylor et al [25] in the context of FPGAs as accelerators. The use of external memories solves the storage limitation: however, it greatly limits parallelism (only one access per external memory chip can be performed at once) or greatly exacerbates financial and power costs, if several external memories are used.

C. Partitioning algorithms

For HLS-based designs, computer vision algorithms are naturally expressed by assuming frames are stored in unbounded

address spaces [26]. This software approach to FPGA design not only easily exceeds FPGA memory capabilities but is also not easily integrated in streaming designs without significant refactoring. This has led to the development of custom hardware blocks and APIs for software integration [27]: "naive" C-based HLS results in several on-chip memory structures, whose sizes and interfaces are dependent on variables' types, often sub-utilizing available on-chip memory. Most HLS tools offer compiler directives - *pragmas* - which guide the synthesis tool according to the designer's intention: optimizing for performance through loop unrolling, or selecting different implementations (on-chip memories or LUTs). We advocate that more directives, invoking different synthesis strategies, are required in order to tackle design constraints such as space and power.

The majority of research into partitioning algorithms has mainly focused on performance: namely, throughput. Gallo et al [28] have shown how to construct efficient parallel memory architectures through High-Level Synthesis: however, their approach is predicated on re-organizing memory placement at algorithm level, by examining computational behavior and placing data accordingly through lattice-based partitioning, which is not feasible on streaming applications where pixels are inputted sequentially. Although possible, it would require a complex memory addressing mechanism between pixel input and memory structure. The authors then expanded their work to incorporate information about loop unrolling [29], providing new partitioning algorithms for maximizing parallelism; however, they did not tackle the utilization problem. Similarly, Wang et al [30] have demonstrated an extremely efficient algorithm for improving throughput, by creating memory structures that facilitate loop pipelining in high level synthesis. Their approach saves up to 21% of BRAMs compared to previous work [31]; still, since their objective is maximizing throughput, supporting loop pipelining, their approach does not achieve optimum memory allocation in terms of utilization efficiency.

D. Memory power reduction

The impact of memory partitioning on power consumption has been researched by Kadric et al [32]. Their approach investigates the impact of parallelism, i.e., how data placement can be leveraged for parallel access, minimizing communication power. A similar approach is taken in [33]. Tessier et al [34] show on chip memory power reduction through partitioning, similar to our approach and previous work by the same authors [35]. However, none of these investigations assume constraints on memory availability. In contrast, we investigate tradeoffs between power and scarce availability, inherent to the image processing domain, future work need clearly identified by Tessier et al: "an investigation to determine the optimal size and availability of different-sized embedded memory blocks is needed".

III. Problem Formulation

In this paper we describe how to partition image frames into BRAMs in order to maximize utilization (i.e., minimize the number of required on-chip memories), subject to minimization of power consumption. We begin by by formulating the utilization efficiency problem, without paying any consideration to power aspects; the following section integrates power consumption in our problem formulation. We assume that only one possible BRAM configuration is used for each image frame buffer.

A. Utilization Efficiency

Definition 1 Given a BRAM storage capacity *C*, and a number of possible configurations *i*, the configurations set **Cfg** is a vector of *i* elements:

$$\mathbf{Cfg} = \begin{pmatrix} (M_1, N_1) \\ (M_2, N_2) \\ . \\ . \\ . \\ (M_i, N_i) \end{pmatrix} = \begin{pmatrix} \mathrm{Cfg}_1 \\ \mathrm{Cfg}_2 \\ . \\ . \\ . \\ \mathrm{Cfg}_i \end{pmatrix} \quad (1)$$

where the first component of each element depicts BRAM width *M* and the second component depicts BRAM height *N*, such that:

$$M_x \times N_x \leq C, \forall x \in [0, i-1] \quad (2)$$

For any given frame size, several possible BRAM topologies are possible[1]. A frame is a 3-dimensional array, of dimensions width *W*, height *H*, and pixel bit width B_w (typically defined as a 2-dimensional array where the type defines the bit width dimension). BRAM topologies are defined based on a *mapping* of 3-D to 2-D arrays and a *partitioning* of a 2-D array to a particular memory structure (Fig. 1).

Throughout the remainder of this paper, we assume the use of a mapping scheme which assigns B_w to the x dimension and H and W to the y dimension, in both row-major and column-major order (where x and y are 2-D array width and height, respectively). This is the default approach in software implementations, where the type/bit width dimension is considered implicit, and a sensible approach for hardware implementations. Mapping bit width B_w across the y dimension would result in implementations where different bits of the same array element (pixel) would be scattered among different memory positions of the same BRAM. This would require sequential logic to read/write a pixel, accessing several memory positions, creating performance, power and size overheads. It should be noted that this approach might offer performance advantages for certain classes of algorithms which might want to compare individual bits of different elements; however, we delegate this aspect to future work. Hence, we define only the default mapping scheme:

Definition 2 A mapping scheme *m* transforms a 3-D array **A3** into a 2-D array **A2** of dimensions x and y by assigning B_w to the x dimension and ordered combinations of *W* and *H* to

[1]Different BRAM configurations do not always equal the same bit capacity.

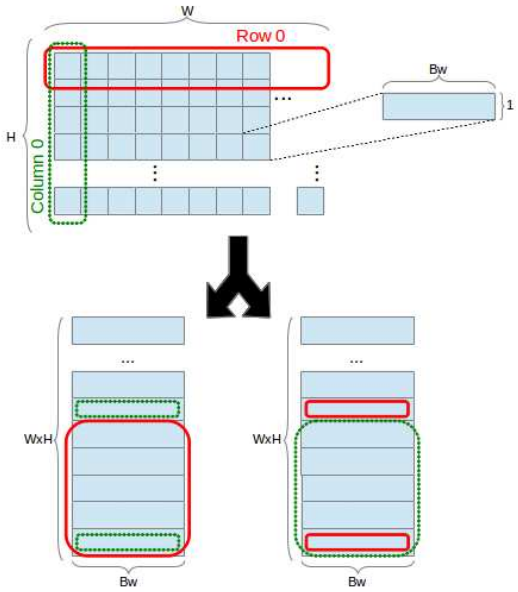

Fig. 1: Mapping a 3-D array into row-major and colum-major order 2-D arrays.

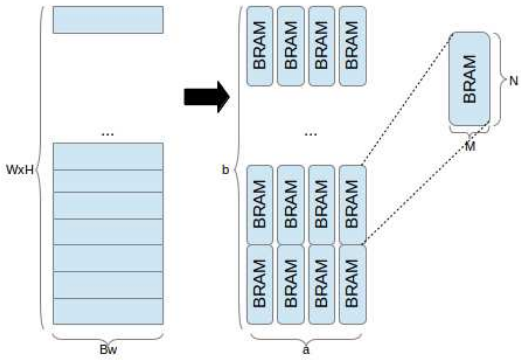

Fig. 2: Mapping 2-D array of dimensions $x = B_w$ and $y = W \times H$ to $a \times b$ BRAMs configured for width M and height N.

the y dimension, for a total of two possible configurations, as depicted in Fig. 1. Mapping schemes are defined as:

$$(x, y) = m(W, H, B_w) \tag{3}$$

$$\text{A2}_{x,y} = \text{A3}_{y \setminus W, y\%W, x}, \; x = B_w, y = W \times H \tag{4}$$

$$\text{A2}_{x,y} = \text{A3}_{y\%H, y \setminus H, x}, \; x = B_w, y = W \times H \tag{5}$$

where $\setminus$ and $\%$ represent integer division and modulo, respectively.

Definition 3 Given a 2-D mapped image frame of dimensions x and y, a partitioning scheme p which assigns pixels across $a \times b$ BRAMs, depicted in Fig. 2, is defined as the linear combination:

$$p(x, y) = \textbf{Cfg} * \big((a_1, b_1), (a_2, b_2), ..., (a_i, b_i)\big) \tag{6}$$

where $*$ stands for linear combination, such that only one $(a_x, b_x), \forall x \in [0, i-1]$ pair has non-zero components (such a pair is generated as a function of x and y), selecting M_p and N_p subject to:

$$((a \times M_p) \geq x) \cap ((b \times N_p) \geq y) \tag{7}$$

Different partitioning schemes p, implementing different functions of x and y, result in different addressing, input and output logic requirements, each with a particular impact on performance and resource usage. As this is the greatest bottleneck in implementing high-level image processing pipelines on an FPGA, it is paramount to define BRAM usage efficiency, i.e. the ratio between the total data capacity of the assigned BRAMs and the amount of data which is actually used.

Definition 4 Given a partitioning scheme p and maximum BRAM capacity C, the utilization efficiency E is defined as the ratio:

$$E = \frac{x \times y}{a_p \times b_p \times C} \tag{8}$$

The default mapping and partitioning schemes in state of the art HLS tools are geared towards minimizing addressing logic (abundant in contemporary FPGAs), resulting in sub-par efficiency in BRAMs usage (still scarce for the requirements of high-level image processing systems). Alternative schemes must be used in order to ensure memory availability within HLS design flows. We define the problem as:

Problem 1 (Utilization Efficiency) Given an image frame of width W, height H and pixel width B_w, select a partitioning scheme, in order to:

Maximize $E = \frac{x \times y}{a_p \times b_p \times C}$

Subject to $((a \times M_p) \geq x) \cap ((b \times N_p) \geq y)$

B. Utilization: Motivational Example

Consider an image frame of width $W = 320$ and height $H = 240$, where each pixel is 8 bits (monochrome), and BRAMs which can be configured according to:

$$\textbf{Cfg} = \begin{pmatrix} (1, 16384) \\ (2, 8192) \\ (4, 4096) \\ (9, 2048) \\ (18, 1024) \\ (36, 512) \end{pmatrix} \tag{9}$$

which is representative of state of the art FPGAs[2], where total BRAM capacity C is given by $C = 35 \times 512$. Using a partitioning scheme

[2]Xilinx Virtex 7 family 18Kbits BRAM.

$$p(m(320,240,8)) = \mathbf{Cfg} * \begin{pmatrix} (8,8) \\ (0,0) \\ (0,0) \\ (0,0) \\ (0,0) \\ (0,0) \end{pmatrix}^T \quad (10)$$

where $m(320,240,8) = (8,76800)$ (Equation 3), yields a BRAM usage count of 64 (8×8 BRAMs configured for width 1 and height 16384), with storage efficiency:

$$E = \frac{8 \times (320 \times 240)}{8 \times 8 \times (36 \times 512)} = 0.520833333 \quad (11)$$

We have observed that this is the default behaviour for Xilinx Vivado HLS synthesis tools: empirical results show that configuration $(M_1, N_1) = (1, 16384)$ is selected through a partitioning scheme where $a_1 = B_w$ and

$$b_1 = \frac{W \times H}{N_1} \quad (12)$$

rounded up to the nearest power of 2. Our experiments show that for any frame size, the synthesis tools' default partitioning scheme can be given by:

$$p(m(W,H,B_w)) = \mathbf{Cfg} * \begin{pmatrix} (B_w, 2^{\lceil log_2(\frac{W \times H}{N_1})\rceil}) \\ (0,0) \\ (0,0) \\ (0,0) \\ (0,0) \end{pmatrix}^T \quad (13)$$

Now consider the same mapping ($x = B_w$, $y = W \times H$), but with a partitioning scheme:

$$p(m(320,240,8)) = \mathbf{Cfg} * \begin{pmatrix} (8,5) \\ (0,0) \\ (0,0) \\ (0,0) \\ (0,0) \\ (0,0) \end{pmatrix}^T \quad (14)$$

which partitions data unevenly across BRAMs, rather than evenly. This scheme yields a BRAM usage count of 40, with storage efficiency:

$$E = \frac{320 \times 240 \times 8}{8 \times 5 \times (36 \times 512)} = 0.833333333 \quad (15)$$

Yet a better partitioning scheme for the same mapping would be:

$$p(m(320,240,8)) = \mathbf{Cfg} * \begin{pmatrix} (0,0) \\ (0,0) \\ (2,19) \\ (0,0) \\ (0,0) \\ (0,0) \end{pmatrix}^T \quad (16)$$

yielding a BRAM count of 38 and efficiency:

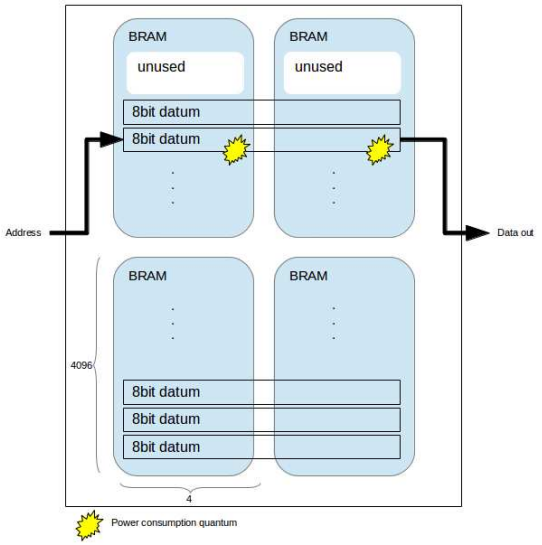

Fig. 3: Partitioning across 2 BRAMs horizontally. Each access consumes 2 power consumption quantums.

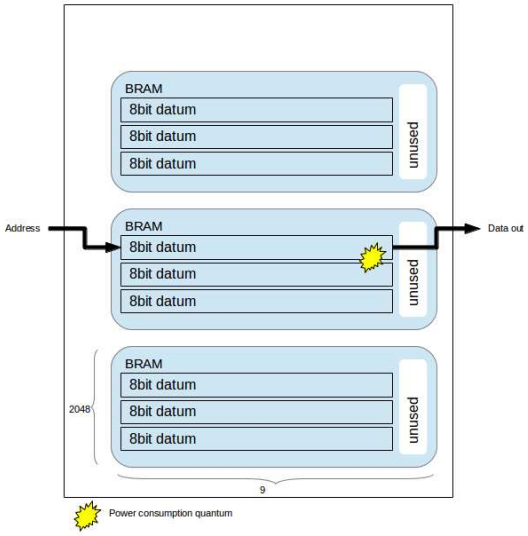

Fig. 4: Partitioning across 1 BRAM horizontally. Each access consumes 1 power consumption quantum.

$$E = \frac{320 \times 240 \times 8}{2 \times 19 \times (36 \times 512)} = 0.877192982 \quad (17)$$

Clearly, partitioning schemes depend on the frame dimensions, width, height, and bit width, to enable efficient use of on-chip memory blocks.

C. Power Considerations

Having formalized the utilization problem, we may proceed to analyse the power implications of each configuration. We model BRAM dynamic power consumption using the model described by Tessier et al [35]: a power quantum is consumed per read and/or write. BRAM static power is directly proportional to utilization, hence addressed in the utilization problem.

For any given BRAM cell, the *read* power is consumed by a sequence of operations: the clock signal is strobed; the read address is decoded; the read data is strobed into a column multiplexer; the read data passes to BRAM external port. *Write* power is consumed by the following sequence: the clock signal is strobed; the write enable signal transfers write data to the write buffers; a line is selected by address decoding; data is stored in the RAM cell.

Now consider the partitioning presented in Equation 10 where each datum is distributed across 8 BRAMs, and the partitioning presented in Equation 16, where each datum is distributed across 2 BRAMs. Each read/write operation in the former must consume power across four times the number of BRAMs in the latter. Fig. 3 and Fig. 4 depict examples of power consumption for two partitioning schemes.

A partitioning scheme which minimizes horizontal usage of BRAMs (i.e., across **x**) is more suitable for clock gating. Since fewer BRAMs must be accessed per operation, the proportian of unused ones, which can be effectively gated, increases. It is straightforward to implement clock gating through chip enable selection [36] which is enabled/disabled based on address decoding.

An intuitive approach to balance power consumption and utilization is to always use the widest BRAM configuration that suffices for B_w, or multiples of the widest available.

This, however, is not the optimal strategy. While it is true that dynamic power is reduced, static power might increase when moving from one configuration to a wider one since the total number of BRAMs might increase: utilization efficiency is modified. Additionally, the logic required for address (and chip enable) signals increases when moving to a wider configuration. This aspect makes the utilization and power problems indivisible. In the following section, we describe our approach to balance these two aspects.

IV. Partitioning for Power and Utilization

We begin by presenting an optimum partitioning algorithm for maximizing utilization efficiency, described in Algorithm 1 in pseudo code notation.

Algorithm 1 Optimum Utilization Efficiency

```
1:  procedure OPTIMUM PARTITION
2:      efficiency ← 0
3:      best ← 0
4:      for x=0 : i-1 do
5:          (Mx,Nx) ← Cfg_x
6:          a ← Bw/Mx
7:          b ← W × H/Nx
8:          efficiency ← (W × H × Bw)/(a × b × C)
9:          if efficiency greater than best then
10:             best ← efficiency
11:             configuration ← (Mx, Nx)
12:         end if
13:     end for
```

For each element in the configurations set **Cfg** (possessing a total of i elements), the algorithm calculates the required number of BRAMs to store a frame of width *W*, height *H* and bit width *Bw*, the efficiency of such a configuration and compares it with the highest efficiency found so far. The focus here is solely on utilization. Effectively, this is an exhaustive search as the number of possible memory configurations is finite and this is an off-line process.

Table I depicts the configurations selected by Algorithm 1 for a representative number of frame sizes and pixel bit widths. Several of the configurations are not power-optimised: notice that for pixels of widths 10, 14 and 22, BRAM configuration 2x8192 is chosen most often (consuming power on 5, 7 and 11 BRAMs per access, respectively). This is intuitive from a utilization efficiency perspective: it is the only configuration that divides the width, and is in accordance with the selection of configuration 4x4096 for pixels of width 8, 12, 20 and 24 and configuration 18x1024 for pixels of width 18.

This non-linearity complicates the derivation of an optimum algorithm for partitioning for both utilization and power efficiencies. Hence, we take a more relaxed approach and define an algorithm through user defined *tradeoffs* (i.e. an estimation of how much BRAM utilization can be traded for power reduction) and power and space *heuristics*, based on empirical properties. Our balanced method is described in Algorithm 2. It is assumed that the *tradeoff* is expressed in percentage points.

Algorithm 2 Balanced Power-Utilization

```
1:  procedure BALANCED PARTITION
2:      efficiency ← 0
3:      configuration ← get_MxNx(OptimumPartition())
4:      best ← get_efficiency(OptimumPartition())
5:      j ← get_index(OptimumPartition())
6:      for x=j+1 : i-1 do
7:          (Mx,Nx) ← Cfg_x
8:          a ← Bw/Mx
9:          b ← W × H/Nx
10:         efficiency ← (W × H × Bw)/(a × b × C)
11:         if efficiency less than best - tradeoff then
12:             break
13:         end if
14:         configuration ← (Mx, Nx)
15:     end for
```

Algorithm 2 begins by selecting the optimum utilization solution and iterating over wider BRAM configurations (in the x dimension), calculating utilization efficiency. As long as the utilization is above the threshold limit, given by the difference between best utilization and tradeoff, in percentage points, the algorithm continues. When it finds the first solution below the threshold, it exits, returning the last solution above the threshold limit. This behaviour is depicted in Fig. 5, and follows the power model heuristics [35] described in the previous section: power consumption decreases as BRAM horizontal width increases (Fig. 3 and Fig. 4).

Table II depicts the BRAM configurations selected by the balanced algorithm, with the tradeoff set to 12 percentage points. Compared to the optimum configurations, the majority

Frame	Pixel Width 8	10	12	14	16	18	20	22	24
160x120	4x4096	4x4096	4x4096	18x1024	18x1024	18x1024	4x4096	9x2048	4x4096
320x240	4x4096	2x8192	4x4096	2x8192	18x1024	18x1024	4x4096	2x8192	4x4096
512x512	4x4096	2x8192	4x4096	2x8192	4x4096	18x1024	4x4096	2x8192	1x16384
640x480	4x4096	2x8192	4x4096	2x8192	18x1024	18x1024	4x4096	2x8192	4x4096
1280x720	4x4096	2x8192	4x4096	2x8192	18x1024	18x1024	4x4096	2x8192	4x4096

TABLE I: BRAM configurations based on optimum utilization algorithm.

Frame	Pixel Width 8	10	12	14	16	18	20	22	24
160x120	9x2048	4x4096	4x4096	18x1024	18x1024	18x1024	4x4096	9x2048	9x2048
320x240	9x2048	4x4096	4x4096	18x1024	18x1024	18x1024	4x4096	9x2048	9x2048
512x512	9x2048	2x8192	4x4096	18x1024	18x1024	18x1024	4x4096	9x2048	9x2048
640x480	9x2048	2x8192	4x4096	18x1024	18x1024	18x1024	4x4096	9x2048	9x2048
1280x720	9x2048	2x8192	4x4096	18x1024	18x1024	18x1024	4x4096	9x2048	9x2048

TABLE II: BRAM configurations based on balanced algorithm with tradeoff equal to twelve percentage points.

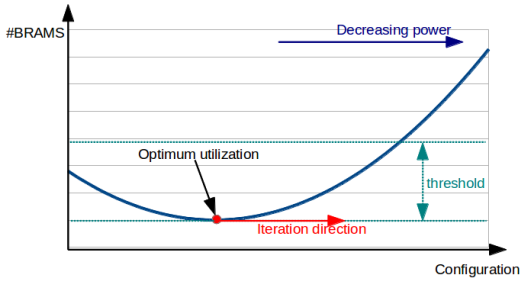

Fig. 5: Balanced algorithm heuristics representation and iteration direction. Number of BRAMs function shape purely illustrative. BRAM horizontal width increases to the right across the Configuration axis.

of widths are increased, resulting in a more power efficient solution based on the aforementioned heuristics.

A. Proposed Methodology

Our algorithms can be utilized in both HDL and HLS design flows: in an HDL design flow, by guiding the designer's implementation and/or refactoring; in an HLS design flow, through integration in the synthesis tools code generation subsystem. Fig. 6 depicts the proposed design flows. The additional steps can be performed manually, either starting from HDL designs or by modifying HLS outputs pre-synthesis; through automated refactoring tools which compute the proposed algorithms; or by the HLS tool prior to code generation. We describe the manual process used in our experiments.

After a memory structure has been derived from the algorithmic specification, according to Equations 3, 4 and 5, Algorithms 1 and/or 2 are computed to determine BRAM partitioning. BRAMs of the computed configuration are instantiated and contained in modules (i.e., hardware entities). A top module instantiates all sub-modules, providing interfaces identical to the base HDL design or to the specification of the HLS tool. Addressing logic within the top module controls chip enable signals to each sub-module, ensuring that non-addressed BRAMs are not enabled.

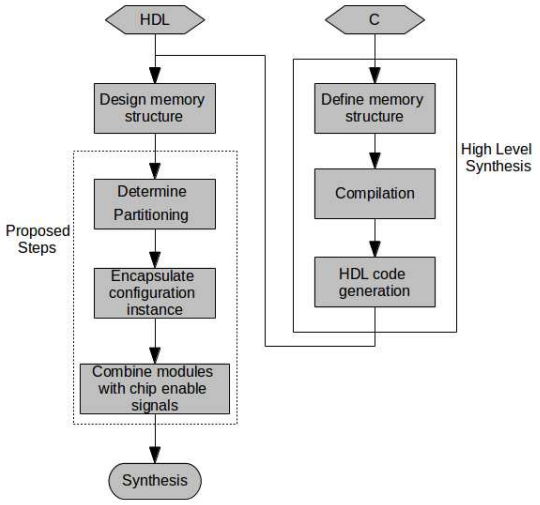

Fig. 6: Proposed design flow from HDL and HLS, highlighting the additional steps required for minimizing utilization and power.

V. Experimental Results

Our experiments target state of the art FPGA devices (Xilinx Virtex 7 device xc7vx690tffg1761-1C and Zynq xc7z020clg484-1). We use Vivado v2016.1 for HDL design, Vivado HLS v2016.1 for High Level Synthesis, and Xilinx Power Estimator for power characterization of implemented designs. We begin by generating frame buffers in several configurations, in order to characterise utilization efficiency and power consumption. We then compare utilization and power against equivalent frame buffers generated by a HLS tool. We conclude by implementing two high level image processing algorithms through HLS, and modifying frame buffers according to the proposed strategies, in order to quantify our algorithms' impact on resource usage and power consumption within complete image processing systems.

A. Frame buffers: BRAM configuration impact

Our first set of experiments characterises utilization and power consumption for two frame sizes as a function of several

320x240					512x512				
Configuration	Power (W)			Efficiency (%)	Configuration	Power (W)			Efficiency (%)
	Static	Dynamic	BRAM			Static	Dynamic	BRAM	
8x5 - 1x16384	0.328	0.054	0.036	83.33	8x16 - 1x16384	0.332	0.07	0.036	88.88
4x10 - 2x8192	0.327	0.036	0.018	83.33	4x32 - 2x8192	0.331	0.053	0.018	88.88
2x19 - 4x4096	0.327	0.026	0.009	87.72	2x64 - 4x4096	0.331	0.043	0.009	88.88
1x38 - 9x2048	0.327	0.027	0.005	87.72	1x128 - 9x2048	0.331	0.046	0.005	88.88

TABLE III: FPGA power usage for monochromatic (8 bits) frames of sizes 320x240 and 512x512 for different BRAM configurations.

possible configurations. The goal of this set of experiments was to validate the utilization efficiency of the partitioning algorithms and the power heuristics used in the previous section.

We implemented frame buffers in Verilog HDL in Vivado v2016.1, explicitly instantiating BRAMs according to the desired configurations. Logic in our design hierarchy routes data, addresses and control signals accordingly. Analysis of post-implementation reports was performed in order to ensure that BRAMs were instantiated according to the desired configuration (depending on the design hierarchy, synthesis tool optimizations could feasibly re-organize BRAM allocation). We performed a *sequential read/write* experiment, where a complete frame is written to memory (sequential pixel input, in row-major order) and then read in the same order. This allows us to validate the power model heuristics assumed in the previous section. Table III depicts power and utilization results for monochrome frames of sizes 320x240 and 512x512. We purposely chose these configurations in order to highlight the non-linear relationship between efficiency and power; while for frames of size 320x240, different configurations yield different efficiency and different power consumption, efficiency is identical across configurations for frames of size 512x512, while power consumption still varies. It is worthwhile noticing that for both sizes, BRAM configuration 9x2048 is less power efficient than configuration 4x4096, despite achieving the same efficiency; although BRAM power is decreased (from 0.009W to 0.005W in both cases), total dynamic power (comprised of BRAM, clocks, signals, logic and I/O) increases due to more complex logic, as previously described.

B. Frame buffers: HLS comparison

Our second set of experiments compares the proposed partitioning algorithms with default strategies employed by commercial HLS tools. The goal of this set of experiments was to confirm that the proposed methodology outperforms commercial HLS tools in both utilization and power consumption.

We performed C-based high level synthesis using Xilinx Vivado HLS, describing frames in the standard format (array type determines bit width, indices determine frame width and height). For each frame size, we report BRAM usage and additional resources (slice registers and LUTs). We utilized standard pixel widths (8 bits for monochrome images, 24 bits for RGB). We estimated optimum BRAM usage using the optimum utilization algorithm and according to the balanced partitioning algorithm in order to compare the power and utilization impact - algorithms were run offline; we have not integrated them in any HLS tool at this point. We implemented the frame buffers in Verilog HDL according to each algorithm, ensuring external interfaces (i.e., read/write data, address and control singals ports) are identical to the ones generated by Vivado HLS from C. We then replaced the frame buffers generated from HLS with our hand-coded Verilog HDL versions. For each frame size, we report BRAM usage and additional resources (slice registers and LUTs) required to implement addressing logic.

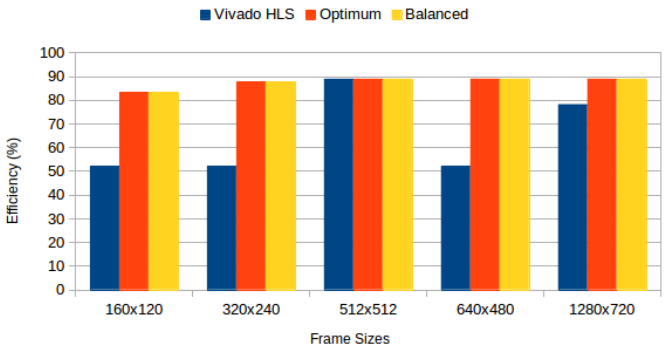

Fig. 7: BRAM utilization efficiency for RGB frames: Vivado HLS versus proposed methods.

Table IV depicts results obtained from the three configurations, for monochromatic and RGB frames respectively, and Fig. 7 compares BRAM utilization efficiency. We characterised the power consumption implications of each generated system using Xilinx Power Estimator for access patterns representative of image processing applications. In our *sequential read/write* experiment, a complete frame is written to memory (sequential pixel input, in row-major order) and then read in the same order. In our *sliding window* experiment, a complete frame is read through 3x3 sliding window. Fig. 8 depicts static power consumption; Fig. 9 and Fig. 10 depict total dynamic power consumption by the three architectures, for sequential read/write and sliding window test cases, respectively; and Fig. 11 and Fig. 12 depict BRAM power consumption for sequential read/write and sliding window test cases, respectively.

C. High-level Image Processing

Out third set of experiments contextualises the impact of memory allocation on high-level image processing systems. The goal of this set of experiments was to quantify how much frame buffers impact resource usage and power consumption

8 bits	Vivado HLS		Optimum Utilization				Balanced			
Frame	BRAMs	LUTs	BRAMs			LUTs	BRAMs			LUTs
			Usage	Configuration	Reduction		Usage	Configuration	Reduction	
160x120	16	0	10	4x4096	-37.5%	22	10	9x2048	-37.5%	48
320x240	64	9	38	4x4096	-40.6%	79	38	9x2048	-40.6%	186
512x512	128	17	128	4x4096	0%	285	128	9x2048	0%	596
640x480	256	34	150	4x4096	-41.4%	337	150	9x2048	-41.4%	742
1280x720	512	64	450	4x4096	-12.1%	1039	450	9x2048	-12.1%	2284
24 bits	Vivado HLS		Optimum Utilization				Balanced			
Frame	BRAMs	LUTs	BRAMs			LUTs	BRAMs			LUTs
			Usage	Configuration	Reduction		Usage	Configuration	Reduction	
160x120	48	0	30	4x4096	-37.5%	41	30	9x2048	-37.5%	91
320x240	192	25	114	4x4096	-40.6%	140	114	9x2048	-40.6%	308
512x512	384	49	384	1x16384	0%	504	384	9x2048	0%	1109
640x480	768	98	450	4x4096	-41.4%	584	450	9x2048	-41.4%	1285
1280x720	1536	192	1350	4x4096	-12.1%	1760	1350	9x2048	-12.1%	3877

TABLE IV: FPGA resource usage for monochromatic frames: generated from Vivado HLS versus hand-coded modifications according to the proposed algorithms.

(a) First frame (b) Second frame (c) 1 scale optical flow (d) 5 scales optical flow

Fig. 13: Optical Flow results using the implementation from [37]. (a) and (b): source frames. (c): output from 1 scale optical flow (used in our FPGA implementation). (d): output from 5 scales optical flow.

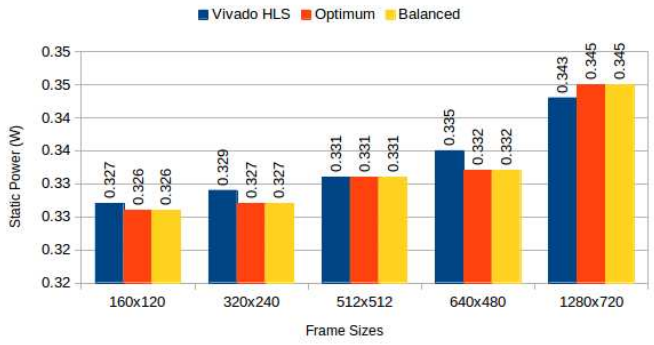

Fig. 8: Static power consumption: Vivado HLS versus proposed methods.

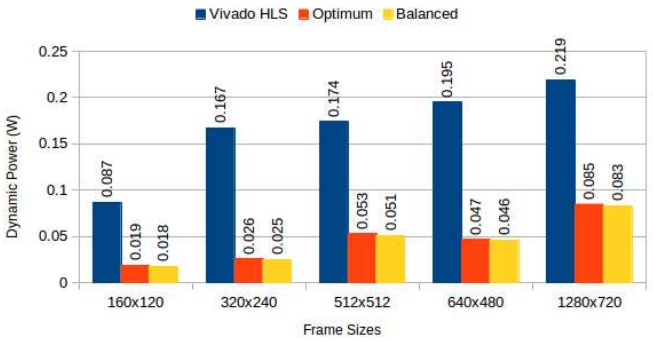

Fig. 9: Total dynamic power consumption for sequential read/write: Vivado HLS versus proposed methods.

within complete image processing systems, based on default and proposed partitioning strategies.

We use Optical Flow and MeanShift Tracking as case studies. Optical Flow estimates the apparent motion of objects caused by the relative motion of an observer; i.e., for two sequential frames, Optical Flow estimates the movement of each pixel (or larger regions) from one frame to the other. It belongs to the *temporal* class of image processing algorithms, i.e., it performs computations across time (different frames). Our Optical Flow implementation is based on the code available from [37] using the TV-L1 method, refactored so it complies with Vivado HLS C synthesis requirements (e.g., dynamic memory allocation was replaced by static memory allocation); we performed no other optimizations. We compute a single scale, rather than multiple scales, for images of size 160x120: an example is depicted in Fig. 13. We developed three versions: with default memory allocation and following the optimum utilization and balanced algorithms. FPGA utilization results for Xilinx Virtex 7 are depicted in Table V (optimum and balanced strategies yield the same BRAM utilization, although different configurations, for our implementation). For the default strategy, BRAMs were insufficient to accommodate all memory requirements, causing the synthesis tool to infer Memory LUTs for parts of the design. Using our approach, BRAMs suffice to implement the complete system. Power consumption per version is depicted in Fig. 14.

MeanShift Tracking [22] calculates a confidence map for

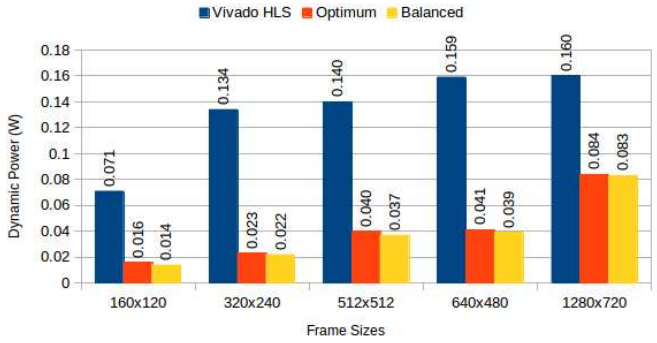

Fig. 10: Total dynamic power consumption for 3x3 sliding window read: Vivado HLS versus proposed methods.

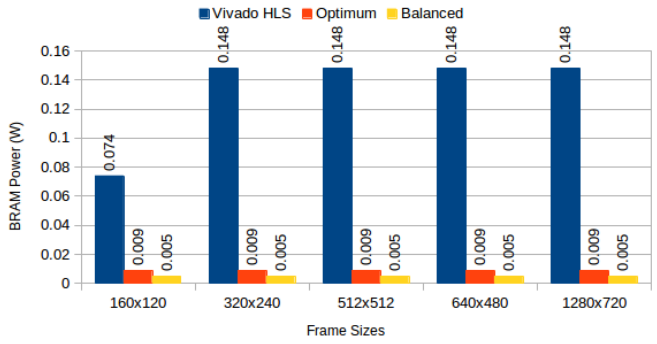

Fig. 11: BRAM power consumption for sequential read/write: Vivado HLS versus proposed methods.

object position on an image, based on a colour histogram of such object on a previous image: i.e., for an object whose position is known and colour histogram is calculated in frame *k*, MeanShift Tracking determines the most likely object position in frame *k+1*, based on colour histogram comparison. It is a *temporal* and *dynamic* algorithm: it performs computations across more than one frame, requiring an unpredictable number of iterations (up to a predefined maximum) on unpredictable frame positions (depending on runtime object position). It was described in C and implemented through

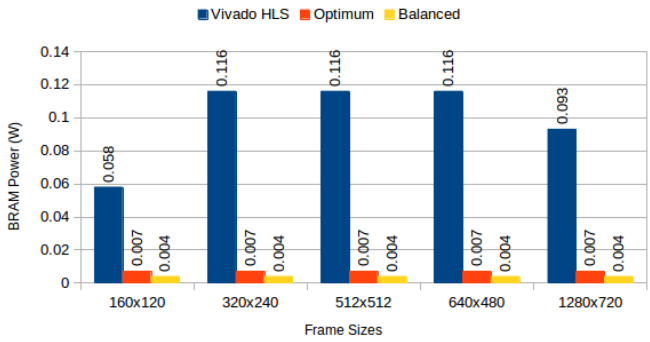

Fig. 12: BRAM power consumption for 3x3 sliding window read: Vivado HLS versus proposed methods.

	Vivado HLS Default	Optimum
FF	24101 (3%)	24101 (3%)
LUTs	200205 (47%)	208724 (49%)
Memory LUT	126114 (73%)	-
IOs	568 (67%)	568 (67%)
BRAM	1008 (69%)	2157 (74%)
DSPs	232 (7%)	232 (7%)
fps	24	24

TABLE V: Optical Flow FPGA resource usage and performance on Virtex 7 xc7vx690tffg1761-1: generated from Vivado HLS versus hand-coded modifications according to the proposed algorithm.

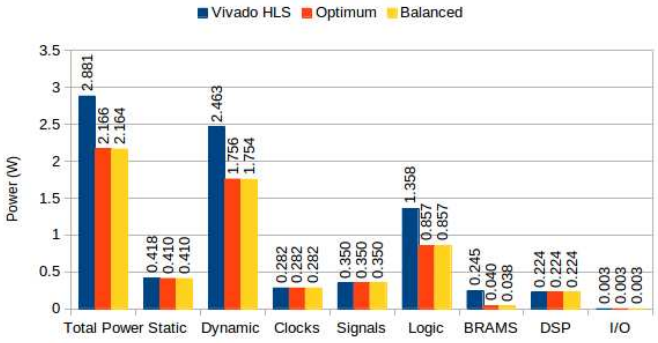

Fig. 14: TV-L1 Optical Flow power consumption on Virtex 7.

Vivado HLS; our implementation was highly optimized for hardware implementation. MeanShift Tracking stores the first input frame (writing the full frame to memory in sequential, row-major order) and calculates a color histogram of a region of width M and height N, centered on an initial object position (reading $M \times N$ pixels). Every subsequent frame is stored, and color histograms for possible new positions are calculated in a region around the previous known position. The new position is decided when the difference between previous and current position is below a pre-defined error bound or a maximum number of iterations is reached. The MeanShift tracking access patterns are not regular or predictable as they depend on the input images; it is representative of memory-intensive image processing algorithms as the output depends on complete (or unpredictable subsets of) scenes, rather than well-defined pixels or regions.

Our tracking system was implemented on a Zynq 7020 chip on a Zedboard, connected to an external camera OV7670 (Fig.

	Vivado HLS Default	Optimum	Balanced
FF	6264 (5%)	6264 (5%)	6264 (5%)
LUTs	9197 (17%)	9310 (17.5%)	9475 (17.8%)
IOs	64 (32%)	64 (32%)	64 (32%)
BRAM	228 (81%)	150 (54%)	150 (54%)
DSPs	8 (3%)	8 (3%)	8 (3%)
fps	134	134	134

TABLE VI: MeanShift Tracking FPGA resource usage and performance on Zynq 7020: generated from Vivado HLS versus hand-coded modifications according to the proposed algorithms.

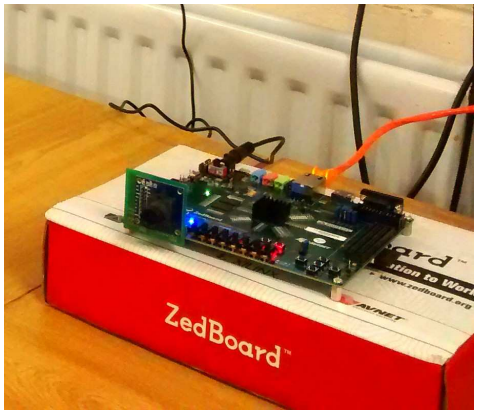

Fig. 15: Zedboard connected to PC through Ethernet.

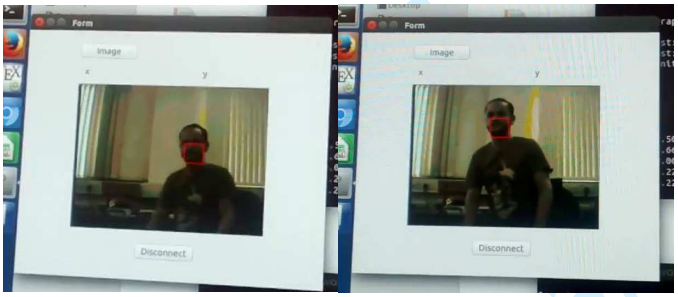

Fig. 16: MeanShift Tracking: real-time face tracking displayed on PC. Image sent from Zedboard over Ethernet connection.

15). The processed data (image plus tracked object position) are sent to the on-board ARM processor which re-transmits to a remote desktop computer over Ethernet. However, it is important to stress that this for communication and display only, the complete algorithm is implemented on the FPGA. Fig. 16 shows real-time operation of our setup.

We developed three system versions: with default memory allocation, optimum utilization memory allocation and balanced allocation for image sizes of 320x240 where each pixel is 24 bits (RGB), with a region of interest of size $M = 16$ and $N = 21$. Identical to the previous experiment, our baseline is the MeanShift Tracking implementation generated by Vivado HLS. The versions used for comparison replace the HLS frame buffer with hand-coded implementations: all other MeanShift Tracking modules are unmodified (generated from C through Vivado HLS). Resource usage for each version is depicted in Table VI. Power consumption per version is depicted in Fig. 17.

VI. Discussion

Experiments show that our partitioning algorithms achieve higher efficiency than default synthesis strategies, except for frames of size 512x512 where the efficiency is unchanged. This is the case where default strategies perform equally well in terms of utilization since the image height and width are powers of 2 (refer back to Equation 13). This confirms that modified partitioning strategies are required, according to requirements, in order to improve memory usage.

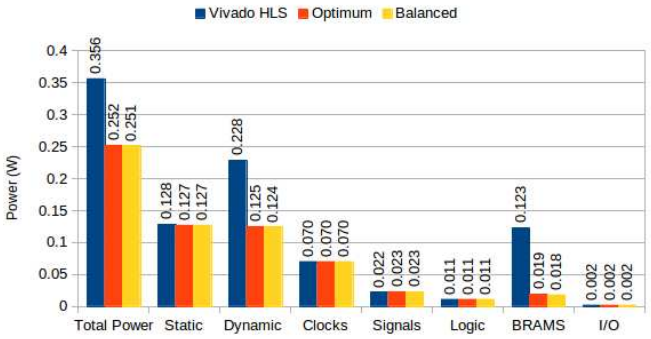

Fig. 17: MeanShift Tracking power consumption on Zedboard.

Static power consumption depicted in Fig. 8 decreases across frame sizes, except for frames of sizes 512x512 and 1280x720, where the utilization efficiency difference between default and proposed strategies is smallest (Fig. 7) and additional addressing logic becomes too (static) power hungry. This confirms the utilization and power problems are indivisible, and must be treated in synergy.

Total dynamic power, on experiments performed on frame buffers, is reduced on average by 74.708% ($\sigma = 7.819\%$) for read/write experiments (Fig. 9), and on average by 72.206% ($\sigma = 12.546\%$) for read-only experiments (Fig. 10). This confirms our hypothesis that memory partitioning offers opportunities for power reduction, despite the need for logic overhead. Considering BRAM dynamic power only, our partitioning methods result in 95.945% average power reduction ($\sigma = 1.351\%$) for read/write experiments (Fig. 11) and 95.691% average power reduction ($\sigma = 1.331\%$) for read-only experiments (Fig. 12).

On our experiments using Optical Flow, where BRAM and Memory LUT power accounts for 25.9% of total power consumption, and 30% of dynamic power, we show that the proposed partitioning algorithms can reduce total power by approximately 25% (Fig. 14). For MeanShift Tracking, where BRAM power accounts for 34.55% of total power consumption, and 53.94% of dynamic power, we show that the proposed partitioning algorithms can reduce total power by approximately 30% (Fig. 17). Algorithm performance (i.e., frames per second) was unaffected by our partitioning methodologies, both in Optical Flow and MeanShift Tracking, since our strategies do not affect memory access latencies and maximum clock frequencies remained unchanged (frame buffers were not responsible for clock critical path).

VII. Conclusions

Efficient mapping of high-level descriptions of image frames to low-level memory systems is an essential enabler for the widespread adoption of FPGAs as deployment platforms for high-level image processing applications. Partitioning algorithms are one of the design techniques which provide routes towards power-and-space efficient designs which can tackle contemporary application requirements.

Based on a formalization of BRAM configuration options and a memory power model, we have demonstrated how

partitioning algorithms can outperform traditional strategies in the context of High Level Synthesis. Our data show that the proposed algorithms can result in up to 60% higher utilization efficiency, increasing the sizes and/or number of frames that can be accommodated on-chip, and reduce frame buffers dynamic power consumption by up to approximately 70%. In our experiments using Optical Flow and MeanShift Tracking, representative high-level image processing algorithms, data show that partitioning algorithms can reduce total power by up to 25% and 30%, respectively, without any performance degradation. Our strategies can be applied to any FPGA family and can easily scale as required for future FPGA platforms with novel on-chip memory capabilities and configurations.

The majority of HLS design techniques have focused on programmability and performance. However, our results show that further research is required in order to improve design strategies towards accommodating other constraints; namely, size and power. Models which describe low-level non-functional properties such as power consumption can support high-level constructs in order to display early cost estimation, guiding the design flow. This requires not only fine-grained characterization of technologies' properties, but also sufficiently powerful modeling abstractions which can lift these properties to high-level descriptions. It will also be interesting to profile and refactor image processing algorithms to determine if alternative mappings (refer back to Equations 4 and 5) could provide higher performance and utilization; this could be pursued in future work involving multi-objective optimizations.

Research in FPGA dynamic reconfiguration has focused on overcoming space limitations; whether this capability can be exploited for image processing power reduction, based on heuristics and runtime decisions, essentially transforming approximate computing design from a static to a dynamic paradigm, remains an open question.

ACKNOWLEDGMENT

We acknowledge the support of the Engineering and Physical Research Council, grant references EP/K009931/1 (Programmable embedded platforms for remote and compute intensive image processing applications) and EP/K014277/1 (MOD University Defence Research Collaboration in Signal Processing).

REFERENCES

[1] J. Wang, S. Zhong, L. Yan, and Z. Cao, "An embedded system-on-chip architecture for real-time visual detection and matching," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 3, pp. 525–538, March 2014.

[2] P. Mondal, P. K. Biswal, and S. Banerjee, "FPGA based accelerated 3d affine transform for real-time image processing applications," *Computers & Electrical Engineering*, vol. 49, pp. 69–83, Jan. 2016.

[3] W. Wang, J. Yan, N. Xu, Y. Wang, and F.-H. Hsu, "Real-Time High-Quality Stereo Vision System in FPGA," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 10, pp. 1696–1708, Oct. 2015.

[4] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S. K. Park, M. Kim, and J. W. Jeon, "Fpga design and implementation of a real-time stereo vision system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 1, pp. 15–26, Jan 2010.

[5] J. Schlessman and M. Wolf, "Tailoring design for embedded computer vision applications," *Computer*, vol. 48, no. 5, pp. 58–62, May 2015.

[6] U. Stevanovic, M. Caselle, A. Cecilia, S. Chilingaryan, T. Farago, S. Gasilov, A. Herth, A. Kopmann, M. Vogelgesang, M. Balzer, T. Baumbach, and M. Weber, "A Control System and Streaming DAQ Platform with Image-Based Trigger for X-ray Imaging," *IEEE Transactions on Nuclear Science*, vol. 62, no. 3, pp. 911–918, Jun. 2015.

[7] G. Dessouky, M. J. Klaiber, D. G. Bailey, and S. Simon, "Adaptive Dynamic On-chip Memory Management for FPGA-based reconfigurable architectures," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–8.

[8] C. Torres-Huitzil and M. A. Nuo-Maganda, "Areatime Efficient Implementation of Local Adaptive Image Thresholding in Reconfigurable Hardware," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 4, pp. 33–38, 2014.

[9] R. Appuswamy, M. Olma, and A. Ailamaki, "Scaling the memory power wall with dram-aware data management," in *Proceedings of the 11th International Workshop on Data Management on New Hardware*. ACM, 2015, p. 3.

[10] S. O. Memik, A. K. Katsaggelos, and M. Sarrafzadeh, "Analysis and FPGA implementation of image restoration under resource constraints," *Computers, IEEE Transactions on*, vol. 52, no. 3, pp. 390–399, 2003.

[11] H. Jiang, H. Ardo, and V. Owall, "A hardware architecture for real-time video segmentation utilizing memory reduction techniques," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 2, pp. 226–236, Feb 2009.

[12] S. G. Fowers, D. J. Lee, D. A. Ventura, and J. K. Archibald, "The nature-inspired basis feature descriptor for uav imagery and its hardware implementation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 5, pp. 756–768, May 2013.

[13] J. Pandey, A. Karmakar, C. Shekhar, and S. Gurunarayanan, "An FPGA-Based Architecture for Local Similarity Measure for Image/Video Processing Applications." IEEE, Jan. 2015, pp. 339–344.

[14] K. Ali, R. Ben Atitallah, N. Fakhfakh, and J.-L. Dekeyser, "Using hardware parallelism for reducing power consumption in video streaming applications," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015 10th International Symposium on*. IEEE, 2015, pp. 1–7.

[15] P. M. Atkinson, "Downscaling in remote sensing," *International Journal of Applied Earth Observation and Geoinformation*, vol. 22, pp. 106 – 114, 2013, spatial Statistics for Mapping the Environment.

[16] S. Jin, D. Kim, T. T. Nguyen, D. Kim, M. Kim, and J. W. Jeon, "Design and implementation of a pipelined datapath for high-speed face detection using fpga," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 1, pp. 158–167, Feb 2012.

[17] R. Stewart, G. Michaelson, D. Bhowmik, P. Garcia, and A. Wallace, "A dataflow ir for memory efficient ripl compilation to fpgas," in *Proceedings of the International Workshop on Data Locality in Modern Computing Systems*. Springer, 2016.

[18] J. Hegarty, J. Brunhaver, Z. DeVito, J. Ragan-Kelley, N. Cohen, S. Bell, A. Vasilyev, M. Horowitz, and P. Hanrahan, "Darkroom: Compiling high-level image processing code into hardware pipelines," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 144:1–144:11, Jul. 2014.

[19] J. Y. Mori, F. Kautz, and M. Hübner, *Applied Reconfigurable Computing: 12th International Symposium, ARC 2016 Mangaratiba, RJ, Brazil, March 22–24, 2016 Proceedings*. Cham: Springer International Publishing, 2016, ch. Efficient Camera Input System and Memory Partition for a Vision Soft-Processor, pp. 328–333.

[20] R. Chen, N. Park, and V. K. Prasanna, "High throughput energy efficient parallel fft architecture on fpgas," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, Sept 2013, pp. 1–6.

[21] M. J. Klaiber, D. G. Bailey, S. Ahmed, Y. Baroud, and S. Simon, "A high-throughput fpga architecture for parallel connected components analysis based on label reuse," in *Field-Programmable Technology (FPT), 2013 International Conference on*, Dec 2013, pp. 302–305.

[22] J. Ning, L. Zhang, D. Zhang, and C. Wu, "Robust mean-shift tracking with corrected background-weighted histogram," *IET Computer Vision*, vol. 6, no. 1, pp. 62–69, 2012.

[23] H. Sahlbach, R. Ernst, S. Wonneberger, and T. Graf, "Exploration of fpga-based dense block matching for motion estimation and stereo vision on a single chip," in *Intelligent Vehicles Symposium (IV), 2013 IEEE*, June 2013, pp. 823–828.

[24] C. H. Chou, A. Severance, A. D. Brant, Z. Liu, S. Sant, and G. G. Lemieux, "Vegas: Soft vector processor with scratchpad memory," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 15–24.

[25] M. Naylor, P. J. Fox, A. T. Markettos, and S. W. Moore, "Managing the fpga memory wall: Custom computing or vector processing?" in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, Sept 2013, pp. 1–6.

[26] M. Schmid, N. Apelt, F. Hannig, and J. Teich, "An image processing library for c-based high-level synthesis," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Sept 2014, pp. 1–4.

[27] Y. T. Chen, J. Cong, M. A. Ghodrat, M. Huang, C. Liu, B. Xiao, and Y. Zou, "Accelerator-rich cmps: From concept to real hardware," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, Oct 2013, pp. 169–176.

[28] L. Gallo, A. Cilardo, D. Thomas, S. Bayliss, and G. A. Constantinides, "Area implications of memory partitioning for high-level synthesis on fpgas," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Sept 2014, pp. 1–4.

[29] A. Cilardo and L. Gallo, "Interplay of loop unrolling and multidimensional memory partitioning in hls," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 163–168.

[30] Y. Wang, P. Li, P. Zhang, C. Zhang, and J. Cong, "Memory partitioning for multidimensional arrays in high-level synthesis," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 12:1–12:8.

[31] J. Cong, W. Jiang, B. Liu, and Y. Zou, "Automatic memory partitioning and scheduling for throughput and power optimization," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 16, no. 2, pp. 15:1–15:25, Apr. 2011.

[32] E. Kadric, D. Lakata, and A. Dehon, "Impact of parallelism and memory architecture on fpga communication energy," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 9, no. 4, pp. 30:1–30:23, Aug. 2016.

[33] E. Kadric, D. Lakata, and A. DeHon, "Impact of memory architecture on fpga energy consumption," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 146–155.

[34] R. Tessier, V. Betz, D. Neto, A. Egier, and T. Gopalsamy, "Power-efficient ram mapping algorithms for fpga embedded memory blocks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 278–290, Feb 2007.

[35] R. Tessier, V. Betz, D. Neto, and T. Gopalsamy, "Power-aware ram mapping for fpga embedded memory blocks," in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, ser. FPGA '06. New York, NY, USA: ACM, 2006, pp. 189–198.

[36] F. Rivoallon, "Reducing switching power with intelligent clock gating," *Xilinx White Paper*, 2010.

[37] J. Snchez Prez, E. Meinhardt-Llopis, and G. Facciolo, "TV-L1 Optical Flow Estimation," *Image Processing On Line*, vol. 3, pp. 137–150, 2013.