Power efficient dataflow design for a heterogeneous smart camera architecture

BHOWMIK, Deepayan <http://orcid.org/0000-0003-1762-1578>, GARCIA, Paulo, WALLACE, Andrew, STEWART, Robert and MICHAELSON, Greg

Published version

BHOWMIK, Deepayan, GARCIA, Paulo, WALLACE, Andrew, STEWART, Robert and MICHAELSON, Greg (2017). Power efficient dataflow design for a heterogeneous smart camera architecture. In: Design and Architectures for Signal and Image Processing (DASIP), 2017 Conference on. IEEE.

Copyright and re-use policy

Power Efficient Dataflow Design for a Heterogeneous Smart Camera Architecture

Deepayan Bhowmik*, Paulo Garcia†, Andrew Wallace†, Robert Stewart‡ and Greg Michaelson‡
*Department of Computing, Sheffield Hallam University, Sheffield, United Kingdom, S1 1WB
†School of Engineering and Physical Science, Heriot-Watt University, Edinburgh, United Kingdom, EH14 4AS
‡Mathematical and Computer Science, Heriot-Watt University, Edinburgh, United Kingdom, EH14 4AS
deepayan.bhowmik@shu.ac.uk, {p.garcia, a.m.wallace, r.stewart, g.michaelson}@hw.ac.uk

***Abstract*—Visual attention modelling characterises the scene to segment regions of visual interest and is increasingly being used as a pre-processing step in many computer vision applications including surveillance and security. Smart camera architectures are an emerging technology and a foundation of security and safety frameworks in modern vision systems. In this paper, we present a dataflow design of a visual saliency based camera architecture targeting a heterogeneous CPU+FPGA platform to propose a smart camera network infrastructure. The proposed design flow encompasses image processing algorithm implementation, hardware & software integration and network connectivity through a unified model. By leveraging the properties of the dataflow paradigm, we iteratively refine the algorithm specification into a deployable solution, addressing distinct requirements at each design stage: from algorithm accuracy to hardware-software interactions, real-time execution and power consumption. Our design achieved real-time run time performance and the power consumption of the optimised asynchronous design is reported at only 0.25 Watt. The resource usages on a Xilinx Zynq platform remains significantly low.**

I. Introduction

Camera networks deployed in time constrained, real life scenarios are an important component of video surveillance, disaster response, environmental monitoring and smart environmental systems. The primary function of any smart camera is to combine image sensing, local image or video processing and communications in a single embedded platform. Often smart cameras are the building blocks of a larger system including a complex camera network. These are developed for higher-performance, better energy efficiency and target compact and low cost devices such as embedded CPUs or FPGAs. Due to their decentralised local pre-processing smart cameras help to address the issues with traditional centralised camera networks, *e.g.*, image communication over a resource limited (bandwidth and power) wireless network. A smart camera architecture (SCA) shifts pre-processing computations upstream to the source of image capture on dedicated, low powered embedded hardware [1]. Current available SCAs encounter problems due to a range of system-level design issues [2]. Hardware needs to be portable and energy efficient, software needs to be easily programmable and compute efficient (suitable for hardware acceleration), and the system must be reliable, scalable and flexible enough to adapt to existing infrastructure. The expectations for SCAs are predicated on complex image processing under tight real-time constraints [3].

However, the adoption of FPGAs in the SCA domain introduces several programmability and design concerns especially for non-hardware programmers in the computer vision community [4]. Supported by the large computer vision software code base, this propelled the use of high level synthesis (HLS) for FPGA embedded vision [5] which enables FPGA design within traditional computer vision frameworks such as OpenCV [4]. This enabled the incorporation of image processing algorithms in the system design: e.g., Visual Saliency. Visual Saliency (VS) or attention modelling characterises the scene to segment regions of visual interest. Hence it is a suitable concept for assessing the relevance of a region in a frame for further processing in a vision system, *e.g.*, saliency based person re-identification [6], resource allocation for driver assistance [7] or prioritization of visual data in wireless surveillance networks [8]. This significantly reduces data dimensionality resulting in accelerated processing, reduced power consumption and bandwidth demand.

In this paper, we present the dataflow design of an visual saliency based SCA as part of a Smart Camera Network (SCN) infrastructure. Through a unified model, the proposed design combines image processing algorithm implementation using a high level dataflow process network, hardware & software integration that includes camera as well as network connectivity and an interface for the camera network over the standard Ethernet infrastructure. We also demonstrate the application of power reduction techniques at three hierarchical levels: processing, algorithm implementation and system architecture. At processing level, scene awareness, implemented through a visual saliency model, is incorporated to reduce data dimensionality. At implementation level, the asynchronicity of dataflow HLS is leveraged to apply low-level power optimizations. At system architecture level, the design is partitioned into different clock islands in order to exploit frequency scaling. The main contributions of this work are:

- A SCN infrastructure using hardware-software integration targeting a heterogeneous CPU+FPGA platform and standard communication network.
- A power optimization methodology for energy efficient design leveraging the properties of the dataflow processing model.

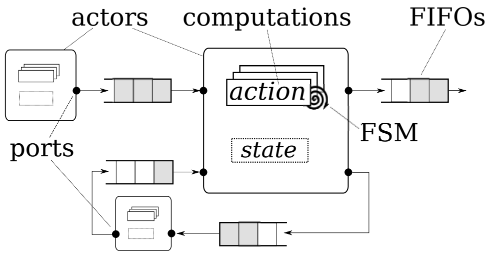

Fig. 1. The dataflow model of computation.

- A dataflow based design of an exemplar visual saliency model for FPGA-based smart camera architectures.

II. Power Efficient Image Processing on FPGAs

The design of the smart camera architure is driven by the goals of ease of algorithm implementation and validation, system architecture reuse, and power efficiency. To meet these goals, the architecture development is driven by the following three principles:

1) A high level parallel programming model for rapid algorithm prototyping, and a compiler framework we extend to support interactive real time image processing simulations on CPUs.
2) A clean separation between hardware components, namely frame capture, application logic and FPGA to co-processor communication, to exploit multi-clock domains with power islands to reduce power use.
3) A portable framework comprising an IP core and a C library for FPGA to co-processor communication, to increase the portability of the smart camera architecture.

A. High Level Algorithm Prototyping

The high level programming model for application prototyping to target the architecture is the dataflow language CAL [9]. The Dataflow Process Network (DPN) model, depicted in Fig. 1, comprises actors for computation and connections to compose these computations. The DPN model maps image processing well onto FPGAs: *a)* it offers two forms of parallel execution, for dataflow to FPGA compilers and synthesis tools, *b)* it supports stateful image processing to support non-trivial algorithms, *c)* it encourages low static memory allocation, and *d)* it enables the processing of image streams from cameras.

a) Parallelism: Compiling image processing dataflow programs to FPGAs provides two forms of parallelism. The first is coarse grained through the parallel execution of actors connected with dataflow wires. The second is fine grained FPGA based combinatorial parallel execution of computations within firing rules, where this form of parallelism is not explicit at the CAL level.

b) Stateful image processing kernels: Computation actors contain finite state machines (FSM) that are sequentially scheduled based on firing rules for transitioning between FSM states. These firings can consume/produce data and modify internal actor state. This model of multiple firing rules, combined with internal actor state, provides the expressivity for stateful image processing such as global reduction operations or complex vision algorithms, *e.g.*, person tracking [10].

c) Low memory resource use: FPGAs have limited on-chip memory, so static memory allocation with any High Level Synthesis approach should be kept to a minimum. This is especially true for the image processing domain, because image buffering on-chip can quickly consume all available memory resources [11]. The dataflow model encourages low memory allocation because programmers pipeline image streams through point to point or region to region functions, rather than multiple image to image functions, which would require image buffers for each actor thus introducing unecesary FPGA Block RAM resource demands.

d) Image stream processing: Dataflow connections are programmed to have depth, *i.e.*, the capacity for feeding tokens in FIFOs between actors. Provided these FIFO depths are sufficiently deep to avoid deadlock, this is a natural programming abstraction for processing infinite streams of image frames captured by camera sensors.

DPNs are not only inherently parallel but also inherently *functionally asynchronous*, *i.e.*, there is no strict enforcing on data availability or action firing order within and across actors. Timing is controlled by token availability, exchanged between actors through FIFOs. Using a HDL backend [12], these dataflow FIFOs are directly translated to hardware FIFOs; network input/output ports follow the same approach, expecting FIFO-compliant interfaces. Thus it is possible to leverage *functional asynchronicity* to implement *electrical asynchronicity*, dividing the system into several clock domains.

The DPN architectural paradigm is highly amenable to hardware design. Each actor is a self-contained entity containing private data; we leverage this for hardware components: camera and network interfaces are encapsulated and re-used in new dataflow designs. Recurring engineering costs are prevented by the asynchronous interfaces; as long as each component supports a FIFO interface, handshaking signals or operating frequency are not of concern.

B. Software Validation of Image Processing Programs

One drawback of using FPGA acceleration in image processing architectures is the time consuming process of synthesising hardware descriptions, and running simulations with test benches to validate the expected outputs. This makes fast algorithm prototyping infeasible, and often domain experts instead prototype in a software language like MATLAB, then reimplement in an FPGA-synthesisable language once the algorithm meets its requirements. To overcome this duplication of implementation effort, we have extended the open source Orcc [9] CAL compiler and IDE with tooling for rapid image processing algorithm prototyping, so that an algorithm need only be implemented once. This simulation harness enables quick validation of image processing dataflow programs on CPUs. The functionality is implemented in the runtime system of Orcc's C backend, and is exposed as actors in the *System* project in the *orc-apps* library[1].

[1] https://github.com/orcc/orc-apps

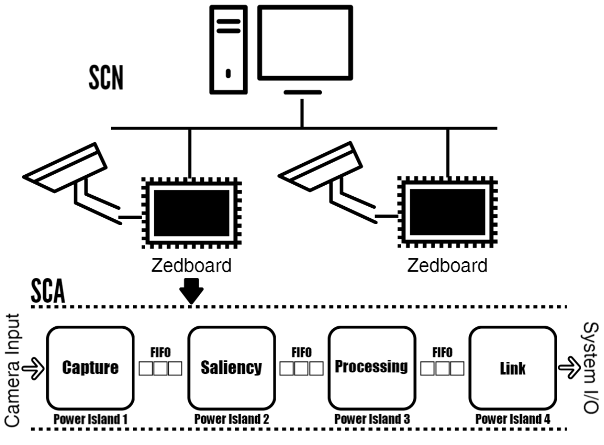

Fig. 2. Smart camera system architecture.

C. FPGA based System Architecture

1) Architecture Overview: The clean separation of sub-systems allows power efficient application deployment across power islands: regions where different power modes (*e.g.*, frequency scaling, power gating) can be applied independently. Figure 2 depicts the architecture with corresponding interfaces and clock islands separation. The *capture* sub-system interfaces with external camera(s), performing low-level initializations and image capture. The *saliency* sub-system reduces image data dimensionality by identifying salient information, bringing scene awareness to subsequent processing (Section III). The *processing* sub-system performs all required operations on images, depending on the desired applications, and the *link* sub-system performs system I/O.

This design philosophy ensures recurring engineering, *e.g.*, adapting the design of an existing camera, modifying and integrating image processing algorithms, has very little cost as the design asynchronicity ensures seamless integration, and power consumption levels can be kept low by reducing idle logic operating at high frequencies. The asynchronous dataflow model is uniquely suited for this kind of implementation as no control signals are passed between clock domains, hence no need for local clock synchronizers. All communication is implemented through passing tokens in FIFOs, which is the preferred method for crossing clock domains.

2) Smart Camera Architecture: The SCA implementation is based on a Xilinx Zedboard[2]. The Zedboard's heterogeneous Zynq chip combines the Xilinx FPGA fabric (PL) with a dual-core ARM Cortex A9 processor (PS). The PL fabric is used to implement the *capture* and *saliency* sub-systems. The *capture* sub-system is implemented through an OV7670[3] camera module, connected directly to the Zedboard FPGA using the available *Pmod* interface. The OV7670 camera module requires an input clock of typically 24MHz frequency (maximum 48MHz) and provides pixels to the DPN at 30 frames per second (approximately 6.9MHz on 320x240 images, 24 bits per pixel). The *saliency* sub-system encapsulates the previously described algorithm implementation.

The *link* sub-system is an heterogeneous implementation across PL and PS layers based on Xillybus[4] to maximise the portability of the system architecture — the Xillybus framework supports the Xilinx Zynz-7000 family, Altera Cyclone V SoC and the Microblaze soft processor over the AXI4 bus. On the PL side, a Xillybus IP core, operating at 100MHz, converts FIFO interfaces on the hardware side into file I/O operations on the software side running on the ARM processor. This simplifies hardware-software co-design when using the dataflow-based FPGA design approach. Software can read FPGA data seamlessly and asynchronously, and standard file operations can be used to control hardware execution. Abstracting hardware operation into files enables standard software frameworks to control FPGA custom logic transparently; a key factor for the integration of established middleware and other software stacks with SCNs. On the PS side, a client software transports processed data through an ethernet port.

3) Power Methodology: For a completely synchronous design, the global clock frequency is dictated by the highest requirement (Xillybus at 100MHz). All other logic, for the camera interface and application logic derived from dataflow programs, is subjected to the same timing constraints, which hinders design re-use if using circuits designed for lower frequencies; meeting timing requirements is one of the most challenging aspects of hardware design, often requiring logic replication, aggressive pipelining and iterative synthesis with different strategies. Using asynchronous dataflow design, multiple clock domains are implemented in this work to produce power-efficient solutions. Inter-clock domain interfaces are implemented through dual-clock FIFOs. As a consequence of dataflow design, only data, rather than data and control signals, traverse sub-systems; thus, only a few inter-clock domain interfaces are required.

III. Evaluation

Human vision behavioural studies and feature integration theory have prioritised the combination of intensity, colour and orientation, three visually stimulating low level features, which comprise the concrete foundations for numerous image domain saliency models [13]. Salient objects are not size specific therefore Multi-Resolution Analysis (MRA) is adopted within many models (*e.g.*, [14]). Multiple orientation and MRA in the discrete wavelet transform (DWT) closely resembles human vision and maximise simultaneous localisation in both the spatial and frequency domains. As a result it is used in saliency computation and shows good promise in reducing complexity while achieving superior performance [14]. This algorithm uses a multi-resolution DWT as its core decomposition and therefore is a more suitable choice than other FPGA saliency implementations [15], [16] that naively follow algorithm implementations not amenable to FPGAs, *e.g.*, by

[2] http://zedboard.org/support/documentation/1521

[3] http://www.voti.nl/docs/OV7670.pdf

[4] http://xillybus.com/

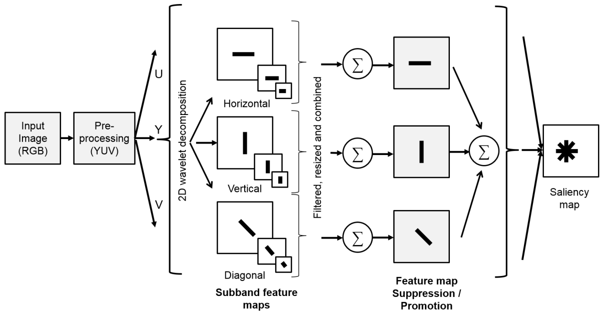

Fig. 3. Overall visual saliency model as described in the original algorithm.

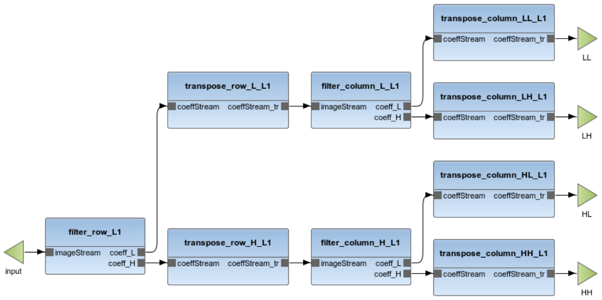

Fig. 4. The dataflow process network of one level wavelet decomposition.

relying on external memory access [15] or by utilizing more computationally complex hardware implementations of calculations [16], such as the Gabor oriented-filter or convolution with smoothing filter. Hence, we adopted and implemented this model.

A. Original algorithm

As the starting point in generating a saliency map from a colour image, the RGB colour space is converted to YUV colour spectral space as the latter exhibits prominent intensity variations through its luminance channel Y. Firstly, the 2D forward DWT (FDWT) is applied on each Y, U and V channel to decompose them in multiple levels. The 2D FDWT decomposes an image in the frequency domain expressing a coarse grain approximation of the original signal along with three fine grain orientated edge maps at multiple resolutions. The DWT captures horizontal, vertical and diagonal contrasts in an image, portraying prominent edges in various orientations. The absolute values of the wavelet coefficients are normalised so that the overall saliency contributions prevent biassing towards the finer scale subbands. An average filter is also applied to remove unnecessary finer details. To provide full resolution output maps, each of the high frequency subbands is consequently interpolated to full frame resolution. The interpolated subband feature maps, lh_i (horizontal), hl_i (vertical) and hh_i (diagonal), $i \in \mathbb{N}_1$, for all decomposition levels L are combined by a weighted linear summation as illustrated in Eq. (1):

$$lh_{1\cdots L_X} = \sum_{i=1}^{L} lh_i * \tau_i, \qquad hl_{1\cdots L_X} = \sum_{i=1}^{L} hl_i * \tau_i,$$
$$hh_{1\cdots L_X} = \sum_{i=1}^{L} hh_i * \tau_i, \tag{1}$$

where τ_i is the subband weighting parameter and $lh_{1\cdots L_X}$, $hl_{1\cdots L_X}$ and $hh_{1\cdots L_X}$ are the subband feature maps for a given spectral channel X, where $X \in \{Y, U, V\}$. This is followed by feature map promotion and suppression steps where normalisation is achieved by accounting local and global maxima. The final overall saliency map (as shown in Fig. 3) is generated by combining weighted spectral channels.

B. Algorithmic implementation using dataflow design

We implemented the above model using the CAL dataflow language. A 5/3 wavelet kernel is used with a lower complexity lifting-based approach. The filters are realised by decomposing the signal into lifting steps by factoring its polyphase matrix using the Euclidean factoring algorithm [17]. Considering notations for the input signal, lowpass subband signal, and highpass subband signal, denoted as $a[n]$, $s[n]$ and $d[n]$ respectively, the DWT is expressed in Eq. (2), where $s_0[n] \triangleq a[2n]$ and $d_0[n] \triangleq a[2n+1]$:

$$5/3 \begin{cases} d[n] &= d_0[n] - \frac{1}{2}(s_0[n+1] + s_0[n]), \\ s[n] &= s_0[n] + \frac{1}{4}(d[n] + d[n-1]). \end{cases} \tag{2}$$

A 2D DWT is computed by separately filtering rows and columns leading to one approximation (LL) subband & three detailed subbands in the horizontal (HL), vertical (LH) and diagonal (HH) direction.

Lifting based 2D DWT decompositions exhibit intrinsic parallelism in their algorithmic description. Unlike imperative languages, dataflow descriptions allow explicit parallel implementations that exploit the concurrency of FPGAs. In this paper a one-level DWT was implemented where individual wavelet prediction and update steps are depicted with a combination of task and pipeline parallelism honouring the algorithmic description. Further optimisations are achieved through stream-based filtering applied on rows and columns using a circular buffer keeping the localised data access patterns. Transpositions of the filtered coefficients are realised using on chip memory. A general discussion of dataflow optimisations for FPGAs using transformation rules is available in [10].

Due to limited availability of the resources on a Zynq FPGA, we have down scaled the algorithm, *e.g.*, one level DWT, and avoided finer details such as feature map supression / promotion in order to reduce hardware usage. Our implementation considers the map generation only from Y channel as this has significant structural information of the scene and carries maximum weight in the original algorithm. Finally a binary thresholding is added for visibly of the saliency maps. The partial dataflow actor network of our implementation is shown in Fig. 4 and represents a one level DWT implemented with pipeline and data parallelism.

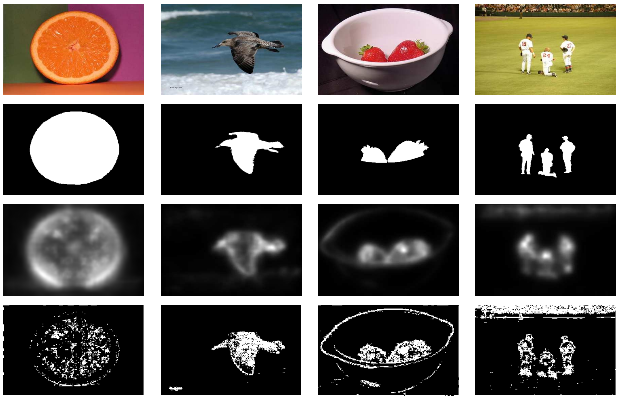

Fig. 5. Example of saliency maps computed by the original model and our scaled down dataflow-based implemention: *row1:* original image, *row2:* ground truth, *row3:* saliency map generated using the original algorithm and *row4:* the thresholded saliency map generated by our dataflow implementation.

TABLE I. Zedboard FPGA resource usage: Asynchronous Visual Saliency

Resource	Usage	Occupation
DSP48E1s	3	1%
FIFO36E1s	2	1%
External IOB33s	80	40%
RAMB18E1s	135	48%
RAMB36E1s	26	18%
Slices	2812	21%
Slice Registers	4989	4%
Slice LUTS	7357	13%
Slice LUT-Flip Flop pairs	8457	15%

TABLE II. Processing time per frame and maximum achievable frame-rate for visual saliency implementation in hardware and software.

	Processing time (ms)	Maximum frame rate
FPGA	19.06	52
CPU	189	5

Using the ORCC-CAL compiler framework, we produce two types of output from the DPN: *a)* C code targeting multi-core CPUs and *b)* Verilog code targeting FPGAs with the CAL-Xronos toolchain [18]. The former is used to verify the functional correctness of implementation and generate results, while the latter is used as a part of our smart camera architecture development on FPGA.

IV. Results and Discussion

Our visual saliency DPN was integrated with a camera controller and a Xillybus IP core on a ZedBoard where we configured the FPGA fabric for acceleration of the image processing and the ARM for interaction with a standard Ethernet communication network. The saliency maps generated by the current downscaled implementation are shown in the last row of Fig. 5 for visual inspection (images are from the MSRA saliency dataset [19]). Unsurprisingly, due to scaled down implementation and insufficient finer details the results are not as good as the original algorithm. However it is also evident that our adapted implementation works reasonably well.

Throughput: With reference to Fig. 2, depicting clock domain separations, the OV7670 camera controller provides a 24MHz clock to the external camera and provides pixels to the DPN at 30 frames per second (approximately 6.9MHz on 320x240 images, 24 bits per pixel). The Visual Saliency Dataflow Process Network is operating at 50MHz, which suffices for the frame rate; the Xillybus interface operates at 100MHz. Each clock domain is separated by dual-clock FIFOs; hence, each domain can input/output data asynchronously. The FIFO connecting the camera capture module with the Dataflow Process Network can behave as a circular buffer if pixels are captured faster than the network can process them. Table I shows FPGA resource usage for our asynchronous visual saliency implementation and Table II shows execution time and maximum achievable frame rate for CPU/FPGA versions. CPU implementation was generated from the same dataflow design through a C backend and executed on the ARM CPU of the Zynq board. In terms of performance, hardware implementation is approximately 10x faster than the software implementation, which is a typical measure when both implementations are generated from the same specification.

Power: The camera interface control can run at the frequency dictated by the camera module (24MHz). On the first synthesis run without any optimizations, our visual saliency dataflow process network generated from CAL, which connects camera capture with the Xillybus module, yielded a maximum operating frequency of 85MHz. Rather than go through redesign iterations, the network can be implemented *as is* in its own clock domain, as long as expected performance is achieved. Expected performance (in this case, 30fps) depends on both clock frequency and cycles per pixel computation: *e.g.*, if the network could process one pixel per cycle, pixel clock frequency would suffice (6.9MHz); if it were to take ten cycles per pixel, a ten times higher frequency would be required. On our visual saliency DPN, 50MHz suffices to ensure the frame rate: our prototype's clock domains (power islands) are depicted on Fig. 2, which also displays the SCA. We compare power consumption between different clocking strategies in order to demonstrate the power gains which can be obtained through asynchronous dataflow HLS. Table III shows power consumption for the full FPGA system and per sub-module; all configurations deliver the same performance (30fps). Results were obtained through the Xilinx XPower Analyzer tool, with high confidence level after implementation and post *Place and Route* simulation.

Discussion: our results indicates interesting power results, for example at 100MHz, both the camera control module and the visual saliency module consume approximately 0.1W. The camera control module's power consumption is substantially influenced by I/O for camera access (hence a mere 50% power reduction when moving from 100 to 24MHz, a quarter of the frequency). For the visual saliency module, however, all power is consumed by internal logic switching; hence, halving the operating frequency approximately halves power consumption [20]. (Some static power is always consumed, depending on circuit size and not on operating frequency). Reducing operating frequencies to the bare minimum required

TABLE III. Power results for different modules/configurations

Sub-module	Power (W)
Camera 24MHz (C24)	0.043
Camera 100MHz (C100)	0.106
Visual Saliency 50MHz (VS50)	0.045
Visual Saliency 85MHz (VS85)	0.078
Visual Saliency 100MHz (VS100)	0.091
Synchronous Full system	**Power (W)**
C100 + VS100	0.361
Asynchronous Full system	**Power (W)**
C24 + VS100	0.298
C24 + VS85	0.284
C24 + VS50	0.252

to ensure our target throughput of 30fps achieves a total system power reduction of approximately 30%.

Our results corroborate the typical arguments for using FPGAs in power critical applications: maximizing on-chip computations can reduce the number of I/O operations, avoiding (power) costly communication. Further power reduction can be achieved by asynchronous design, reducing on-chip power consumption. We have shown how the dataflow model enables straightforward, seamless low-power design by embedding electrical asynchronicity in the algorithmic model. However, there is a drawback, observable from the FPGA resource utilization, *i.e.*, high usage of embedded memories (BRAMS). This is due to the FIFO interconnect model. FIFOs are used not only for sub-system connection, but also for actor connections within the dataflow process network. This limitation may inhibit the deployment of highly complex networks on small scale FPGAs with limited BRAM, forcing designers to move towards bigger (and power hungry) families where low-power design strategies become even more relevant.

V. Conclusions

In this paper, we demonstrated how asynchronous dataflow HLS can be leveraged for power-efficient smart camera network design. The semantics of the dataflow model are not only highly applicable to the computer vision domain; they are also a suitable match for hardware implementation. The dataflow model embeds the required hardware mechanisms for asynchronicity in the design concept, allowing fine-grained separation of clock domains for power efficiency directly from the algorithm representation. The absence of explicit control signals removes the need for local clock synchronizers at hardware level; token passing lends itself well to FIFO-based asynchronous hardware design. Additionally, this approach removes recurring engineering costs by simplifying interconnections between sub-modules; the standard interfaces allow design modules to be re-used and easily encapsulated as library components within the HLS framework. In our prototype system, we showed how a visual saliency algorithm could be easily described in the dataflow model and integrated on a complete system, based on a commercial-off-the-shelf FPGA platform. We demonstrated how functional asynchronicity could be leveraged for low-power design achieving 30% power reduction from a (electrically) completely synchronous to asynchronous design.

Acknowledgements

We acknowledge the support of the Engineering and Physical Sciences Research Council, GR EP K/009931/1 and a HEIF Impact fellowship at Sheffield Hallam University.

References

[1] M. Birem and F. Berry, "DreamCam: A modular FPGA-based smart camera architecture," *Journal of Systems Architecture*, vol. 60, no. 6, pp. 519 – 527, 2014.

[2] M. Reisslein, B. Rinner, and A. Roy-Chowdhury, "Smart camera networks," *Computer*, vol. 47, no. 5, pp. 23–25, May 2014.

[3] M. Wolf, "Platforms and architectures for distributed smart cameras," in *Distributed Embedded Smart Cameras*. Springer, 2014, pp. 3–23.

[4] C. Bobda, M. Mefenza, F. Yonga, and A. A. Zarezadeh, "Reconfigurable architectures for distributed smart cameras," in *Distributed Embedded Smart Cameras*. Springer, 2014, pp. 43–68.

[5] D. G. Bailey, "The advantages and limitations of high level synthesis for FPGA based image processing," in *Proc. Int'l Conf. on Distributed Smart Cameras*, 2015, pp. 134–139.

[6] N. Martinel, C. Micheloni, and G. L. Foresti, "Kernelized saliency-based person re-identification through multiple metric learning," *IEEE Trans. on Image Processing*, vol. 24, no. 12, pp. 5645–5658, Dec 2015.

[7] Y. P. S. Matzka, A.M. Wallace, "Efficient resource allocation for automotive attentive vision systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 859–872, Feb 2012.

[8] I. Mehmood, M. Sajjad, W. Ejaz, and S. W. Baik, "Saliency-directed prioritization of visual data in wireless surveillance networks," *Information Fusion*, vol. 24, pp. 16 – 30, 2015.

[9] H. Yviquel, A. Lorence, K. Jerbi, G. Cocherel, A. Sanchez, and M. Raulet, "Orcc: Multimedia development made easy," in *Proceedings of the 21st ACM International Conference on Multimedia*, ser. MM '13. ACM, 2013, pp. 863–866.

[10] R. J. Stewart, D. Bhowmik, A. M. Wallace, and G. Michaelson, "Profile Guided Dataflow Transformation for FPGAs and CPUs," *Signal Processing Systems*, vol. 87, no. 1, pp. 3–20, 2017.

[11] R. Stewart, G. Michaelson, D. Bhowmik, P. Garcia, and A. Wallace, "A dataflow IR for memory efficient ripl compilation to FPGAs," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2016, pp. 174–188.

[12] E. Bezati, "High-Level Synthesis of Dataflow Programs for Heterogeneous Platforms: Design Flow Tools and Design Space Exploration," Ph.D. dissertation, School of Engineering, Ecole Polytechnique Federale de Lausanne, Switzerland, April 2015.

[13] A. Borji and L. Itti, "State-of-the-art in visual attention modeling," *IEEE Transactions on Pattern Analalysis Machine Intelligence*, vol. 35, no. 1, pp. 185–207, Jan. 2013.

[14] D. Bhowmik, M. Oakes, and C. Abhayaratne, "Visual attention-based image watermarking," *IEEE Access*, vol. 4, pp. 8002–8018, 2016.

[15] S. Bae, Y. C. P. Cho, S. Park, K. M. Irick, Y. Jin, and V. Narayanan, "An FPGA implementation of information theoretic visual-saliency system and its optimization," in *Intl. Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2011, pp. 41–48.

[16] F. Barranco, J. Diaz, B. Pino, and E. Ros, "Real-time visual saliency architecture for FPGA with top-down attention modulation," *IEEE Trans. on Industrial Informatics*, vol. 10, no. 3, pp. 1726–1735, Aug 2014.

[17] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Anal. Appl.*, vol. 4, no. 3, pp. 245–267, 1998.

[18] E. Bezati, M. Mattavelli, and J. W. Janneck, "High-level synthesis of dataflow programs for signal processing systems," in *8th International Symposium on Image and Signal Processing and Analysis (ISPA)*, Sept 2013, pp. 750–754.

[19] T. Liu, Z. Yuan, J. Sun, J. Wang, N. Zheng, X. Tang, and H. Y. Shum, "Learning to detect a salient object," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 2, pp. 353–367, 2011.

[20] Altera. (2012) Reducing power consumption and increasing bandwidth on 28-nm FPGAs. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/wp/wp-01148-stxv-power-consumption.pdf