Chapter 2
A Survey of Systems-on-Chip Solutions for Smart Cameras

Ali Ahmadinia and David Watson

Abstract With the advances in electronic manufacturing technologies, integration of disparate technologies including sensors, analog components, mixed signal units and digital processing cores into a single chip has become reality, which is an increasing trend in different application domains, especially for distributed smart camera products. In this chapter, a survey of existing System-on-Chip solutions for distributed smart cameras able to capture and intelligently process video in real-time and communicate with other cameras and sensors remotely is presented.

2.1 Introduction

Smart cameras consist of three main units: Sensing, Processing, and Communications [18] as illustrated by Fig. 2.1. The sensing unit is responsible for image capture and may also perform pre-processing before the main processing task(s) are performed. The processing unit carries the main computation and is therefore the brains of the smart camera. The processing unit comprises Processing Elements (PEs)—which include all forms of processors such as Digit Signal Processors (DSPs), hardware functions/accelerators, and microcontrollers—on- and off-chip memories, and the communications architecture(s) used for inter- and intra-processor communications and data movement. The last unit is the communications unit which is responsible for transmitting the processed/abstracted data to output devices, where the end user can use it. These three units together make up the architecture of System on Chip (SoC)

A. Ahmadinia (✉) · D. Watson
School of Engineering and Built Environment, Glasgow Caledonian University,
Cowcaddens Road, Glasgow, UK
e-mail: ali.ahmadinia@gcu.ac.uk

D. Watson
e-mail: david.watson2@gcu.ac.uk

C. Bobda and S. Velipasalar (eds.), *Distributed Embedded Smart Cameras*,
DOI: 10.1007/978-1-4614-7705-1_2,

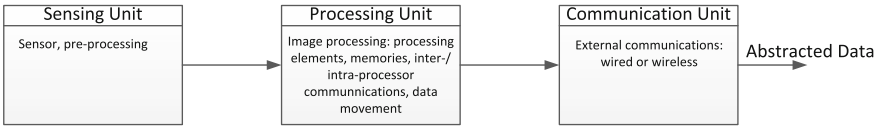

Fig. 2.1 Architecture of smart camera SoC solutions [18]

smart cameras, and are therefore directly responsible for the abstracted representation of the input data following processing of input data [18].

Since smart camera SoC solutions provide a service to the end user, there are several Quality of Service (QoS) attributes associated with them. The QoS characteristics of smart camera SoCs encompass attributes such as frame-rate, transfer delay, image resolution, and video-compression rate [3]. However, as SoC-solutions for smart cameras are embedded devices, power consumption and resource consumption are also important attributes and should be minimised where possible. The frame-rate of a smart camera is dependent on the task it is performing. For example, a smart camera monitoring a car park may require real-time frame-rates of 30 frames per second (f/s)—in the interests of security; whereas a smart camera monitoring the flow of traffic on a busy stretch of road may not require real-time frame-rates: application discretion is required. Transfer delay can impact the frame-rate of the system and is dependent on the sensing unit used and data-movements of the system [14], which is directly impacted by the resolution of the input data from the imaging sensor and any compression methods used.

Each of theses attributes have a direct impact on the energy and resource consumptions of the smart camera SoC, and it is therefore paramount that the processing unit be optimised for the processing and movement of the input data. One last consideration for QoS in modern-day smart camera SoCs is security, where the collection of data that is sometimes private can introduce the need for the safe storage and transmission of data [20]. Smart camera SoC designers must also be aware of the need for encryption and secure network protocols, as the scale and complexity of smart cameras evolves. Table 2.1 summarises the QoS requirements of embedded smart camera SoC solutions.

Smart camera SoCs can be classified based on the decision classification network depicted in Fig. 2.2. Here we have decomposed the classifications of smart camera SoC solutions into the SoC used to perform the smart camera functions, such as processing, display, etc. and the smart camera solution as a whole and how it interacts with the end user. Note, we have abstracted the development/implementation platform into the SoC sub-categories, as this allows the classification of modern-day smart cameras more accurately. Smart cameras themselves can perform function(s) as a standalone agent, or as part of a larger network of smart cameras. The use of smart cameras in a distributed manner allows more data to be collected about a scene or environment and therefore allows more astute deductions to be made about image scenes [18]. The distribution of smart cameras can be wired or wireless,

Table 2.1 Table of QoS attributes for smart camera SoCs

Attribute	Description	Sample Operating Characteristics
Throughput	Time required to capture, process, and output and image	Real-time = 30 f/s
Power consumption	Power consumed by all units of the smart camera SoC: directly impacts running costs	Within the region of mW is nominal
Resource consumption	Computational and memory resources consumed by the SoC: directly impacts implementation costs	Ranges from bytes to kilobytes

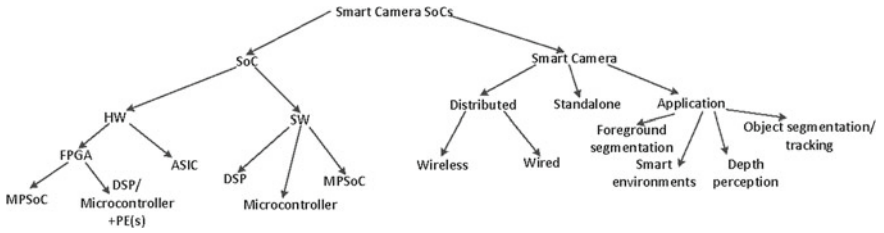

Fig. 2.2 Classification network for smart camera SoC solutions

however distributed smart cameras tend to be wired, as they have more computational power and are not as focussed on power/resource consumption [18].

Smart cameras are used in a variety of contexts, from surveillance and object tracking, to medical procedures and smart environments [18]. The use of smart cameras to create intelligent spaces—environments where the status of objects can be readily ascertained—has been investigated by Manbhai [22]. However the development environment is not embedded. With the increase in data available from smart cameras, the processing of input data has taken on more complex roles, including the rendering of stereo images for depth perception applications [19]. For a thorough and complete survey of smart camera SoC solutions related to embedded systems, we only include design and implementation work relevant to SoC design.

2.2 Sensing Unit

The sensing unit is responsible for image capture and can also be used for pre-processing images before the main processing is performed. CMOS imaging sensors are a popular choice for smart cameras as they allow for the access of pixels in a similar manner to that of random access memory [7]. CMOS sensors also allow for the fabrication of hardware functions close to the sensor [7], making them

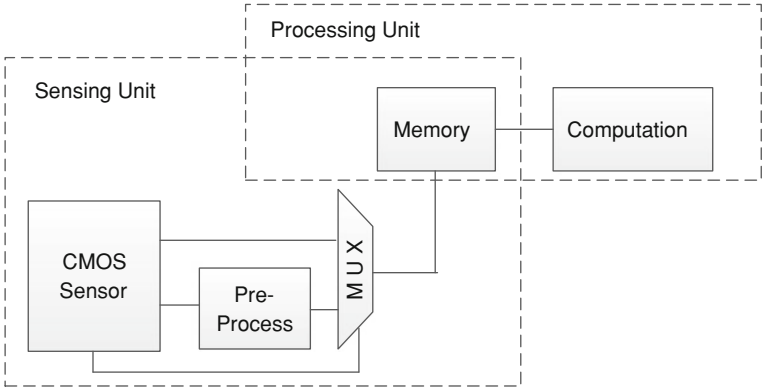

Fig. 2.3 Example architecture of a sensing unit, where the extraction of pixels can be performed in parallel

suitable for smart camera applications. Examples of this are that by Moorhead and Binnie [15], where the CMOS camera was capable of carrying out Canny edge detection on input images, and Heyrman et al. [9] whose CMOS camera was able to select Regions of Interest (ROI), which were then further processed by the processing unit. Designers can implement the optimal parallelisation of pixel extraction from CMOS imagers, allowing them to tailor the sensor the the smart camera's application. The 90 nm fabrication technology of 2005 required approximately 25 mm^2 of silicium for a parallel output of 64 $\times$ 64 pixel sub-window [9], but with modern 20 nm technology this could decrease. Lacassagne et al. [12] implement a programmable artificial retina—a network of pixels that brings processing to the data, as opposed to transferring data to PEs. Each pixel of the artificial network consists of an ADC and 48 bits of memory and can share data with its four surrounding pixels (neighbours). This allows data locality to be exploited with high-levels of parallelism implemented by dedicated hardware functions. Computation is performed using a boolean unit, which can perform bit-level computation.

A typical architecture of a smart camera sensor based on CMOS technology can be found in Fig. 2.3. Here we can see how the use of CMOS sensors allows designers to preprocess input images before they are passed on for further processing at the processing unit. The ability to extract pixels in parallel makes CMOS sensors an attractive option for smart cameras, but must also compliment the memory architecture of the SoC, and how data is moved into the processing unit. Figure 2.3 also shows how the sensing and processing units can overlap when preprocessing is used. In this example, the preprocessing block can access pixels from the sensor and process them, before writing them to memory for the processing unit to use.

2.3 Processing Unit

From Fig. 2.3, we can see the processing unit encompasses the computational agents of the SoC, the memory subsystem, and implicitly the communications system. We start with the arrangement of computational resources for smart camera SoCs, and then move onto the memory and communications subsystems.

2.3.1 Processing Unit: Processing Elements

The Processing Elements (PEs) of SoCs can be Digital signal Processors (DSPs), microcontrollers, dedicated hardware functions/accelerators, or general-purpose processors. Bramberger et al. [3] present a distributed embedded smart camera constructed from off-the-shelf components. TMS320C6416 DSPs by Texas Instruments are used as the processors of the SoC and are coupled together with memory via a PCI bus. The use of DSPs increases the flexibility of the SoC, as changes to the functionality are implemented in software. However, the use of DSPs for smart camera SoCs can limit the throughput of the system, as the processor's architecture is designed for general-purpose use: the memory subsystem can also present a bottleneck [5]. Microcontrollers are typically used to coordinate events of a SoC, but have been used to implement processing functions, object detection functions [10], where the 8-bit microcontroller is used to perform the feature extraction. However, the limited architecture of the microcontroller inhibited realtime QoS requirements and may be improved by upgrading to a 16- or 32-bit architectures. A Picoblaze soft-core microcontroller is used by Meng et al. [14] to synchronise and control all components of a Multiprocessor System on Chip (MPSoC). Each hardware core consists of a Picoblaze soft-core coupled with a coprocessor interface such, as that in Fig. 2.4a, to carry out dedicated functions in hardware. However, the flexibility of the smart camera is only in the ability to reprogram the Picoblaze, and not in the ability to reconfigure the co-processors.

Creating PEs from hardware functions as coprocessors is an attractive option for smart camera designers, as they are designed to carry out specific, dedicated functions within the SoC. PEs can be as simple as frame-grabbing [10], to applying image processing kernels and object detection functions [11]. Kruijtzer et al. [11] design smart imaging and motion estimation cores coupled to an ARM9 processor. The smart imaging core carries out low-level image processing algorithms, such as applying kernels, calculating histograms, etc., whereas the motion estimation core performs high-level motion estimation. This system uses the ARM9 to coordinate and control events, with the SoC implemented on a Field Programmable Gate Array (FPGA). However, this system requires large amounts of computational resources due to the generic nature of the PEs. Designers must take care to implement only time-critical or computationally intensive functions in hardware, in order to justify the resource utilisation.

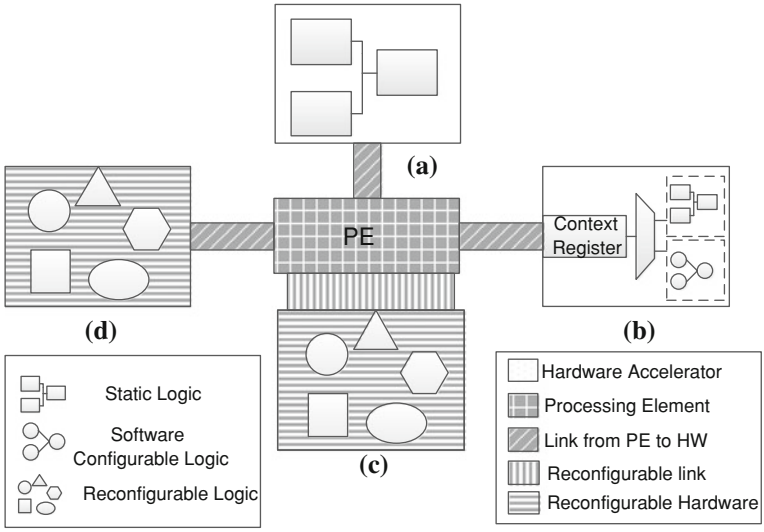

Fig. 2.4 Four ways in which hardware accelerators can be designed and interact with PEs of a SoC: Static arrangement (**a**), software programmable arrangement (**b**), reconfigurable arrangement with reconfigurable link (**c**), and reconfigurable arrangement with fixed link (**d**)

Chan et al. [4] apply dedicated programmable hardware accelerators with an ARM 926EJ-S for object detection/segmentation and face detection. The programmable morphology coprocessor is capable of performing dilation and erosion tasks based on the value in its context register (Fig. 2.4b), which is set by the ARM CPU. The programmable hardware increases the performance of the smart camera SoC, but may inefficiently utilise hardware resources depending on how they are used at runtime. This is example of reconfiguring a static hardware arrangement through software via the use of context registers. Computational resources would be utilised more efficiently if the physical makeup of the hardware accelerators was modified during runtime.

Chen et al. [5] implement a stream processor on an FPGA, where the data movements and PEs are optimised for high throughputs. The processing architecture is designed to be multipurpose, where PEs can be configured to carry out 1 of 6 popular image processing tasks, such as downsampling and applying 2D kernels. These PEs can be reconfigured together, to create larger and more powerful PEs which work together to perform more complex image processing tasks. Figure 2.4d illustrates this concept for one reconfigurable hardware accelerator. This reconfigurable architecture is an example of how smart camera SoCs can be implemented in reconfigurable hardware to achieve high throughputs and power efficiencies. However, the applicability of such an architecture is limited by the PEs used in the system, and how they can collaborate to create more complex processing operations. As smart cameras become more intelligent, the operations they are required to perform become

more complex and may therefore increase the size and complexity of reconfigurable architectures.

Lacassagne et al. [12] evaluate the PowerPC (PPC), which is optimised for multimedia applications through its instruction set and vectorization. The authors found that the PPC processor is suited to low-level computer vision applications, but its throughput is limited by its instruction set and internal register architecture. Albani et al. [1] brought the processing of image data to generic, programmable embedded systems. A 32-bit RISC processor and vision/neural coprocessor for data processing—complimented by 512B instruction cache, 1MB RAM, and 512KB FLASH memory. This system improves design flexibility, as the processor can be reprogrammed to perform different tasks, as can the neural coprocessor within reason, but again the physical arrangement of the SoC resources is static.

Oetken et al. [17] also implement a reconfigurable SoC for a smart camera. The SoC is divided into static and dynamic regions, where an embedded CPU subsystem is placed in the static region, and the dynamic region can be used for custom hardware accelerators. To accommodate the variability of the hardware that may exist in the dynamic region, a reconfigurable bus (ReCoBus) is used, which allows connections between master and slave hardware accelerators, as well as the embedded CPU subsystem. Figure 2.4c illustrates this concept for one hardware accelerator. A recent example of dynamic reconfiguration for hardware accelerators in SoCs is [8], which makes use of one controlling PE that has dynamically reconfigurable hardware accelerators. However, such a large area for dynamic reconfiguration could introduce overheads for simple tasks that may have to be implemented in dynamic regions.

Table 2.2 summarises the PEs of smart camera SoCs, and their respective advantages and disadvantages. Based on this summary, designers of smart camera SoCs must make several key design decisions about the computational components of the smart camera SoC. Firstly, the required computation of the smart camera application: object detection/tracking, or basic background/foreground segmentation will dictate the types of PEs in the SoC. Secondly, how the smart camera will be deployed: will the smart camera perform a dedicated task? Will it be able to perform several tasks, or even be reprogrammable? Each of these considerations will have a direct impact on the memory and communications architecture of the SoC, which is discussed next. FPGAs can be used to create computationally efficient smart camera SoC-solutions, but can limit the design space of computer vision applications to specific functions implemented in hardware. Other examples of these techniques are [2, 4, 13, 16].

2.3.2 Processing Unit: Memory and Communications

The memory architecture of a smart camera SoC must compliment the processing requirements of the algorithm(s) executing on it [18]. As identified in the previous section, there are hardware and software solutions for designing smart camera SoCs: hardware functions and accelerators, and DSPs/microcontrollers respectively.

Table 2.2 Processing element summary

Processing Element(s)	Pros	Cons	Throughput	Power Consumption	Resource Consumption
TMS320C6416 DSP [3]	SW programmable SoC Application flexibility	Limited computational throughput Static memory arrangement	Low	Very high	Very high
Picoblaze micro-controller [14]	SW reprogrammable MPSoC; increased concurrency	May under utilise resources	Low	Low	Low
Dedicated hardware functions					
PPC [12]	SW reprogrammable Application flexibility	Static architecture General purpose processing	High	High	High
Reconfigurable hardware functions [5]	HW programmable MPSoC	Communications subsystem may limit scalability	High	Low	Low
	Increased resource utilisation	PEs may be limited by available resources			
	Application flexibility				
Microcontroller [10]	SW reprogrammable SoC	May under utilise resources	Medium	Medium	Low
Dedicated hardware functions		Limited ability of microcontroller			
ARM9 processor [11]	SW reprogrammable SoC	Static hardware functions	Unknown	Unknown	Unknown
Dedicated imaging and motion hardware functions		May under utilise resources			
ARM 926EJ-S [4]	SW programable MPSoC	May under utilise resources	High	Low	Low
Programmable hardware functions	Increased design flexibility and concurrency				
32-bit RISC processor [1]	SW reprogrammable hardware	May under utilise resources	Medium	Medium	Medium
Reprogrammable vision/neural co-processor	Increased design flexibility	Hardware must be redesigned for new vision tasks			

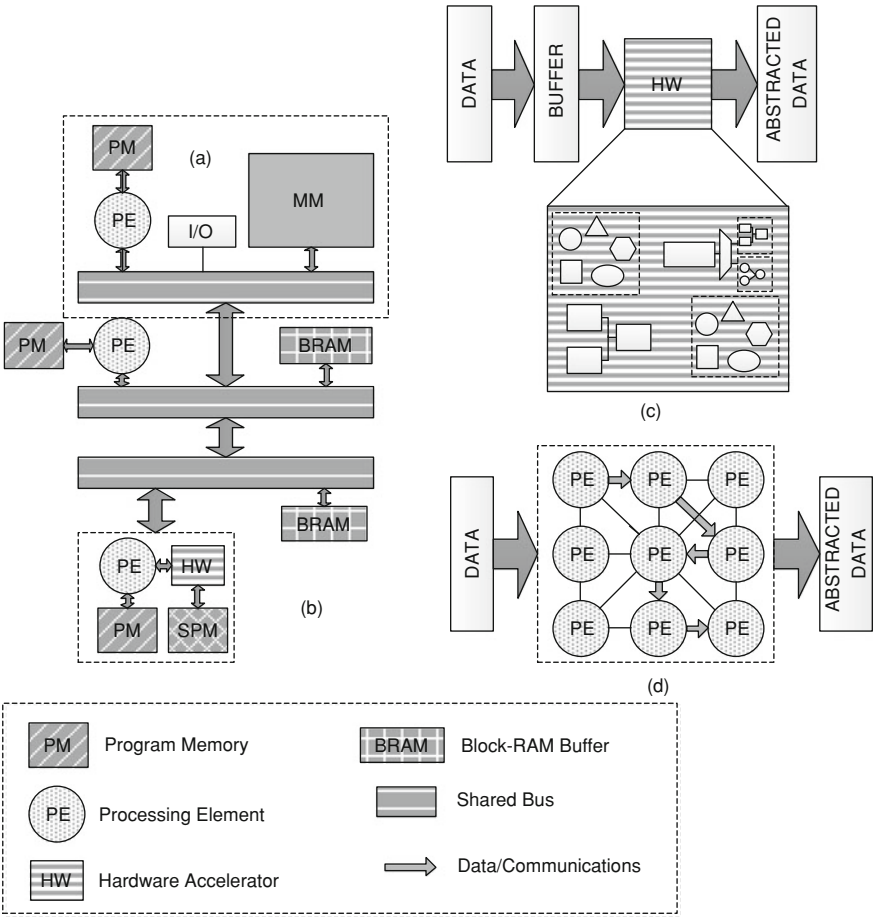

Fig. 2.5 Memory and communications architectures used for the works described in this chapter: shared bus (**a**), hierarchical (**b**), streaming (**c**), and associative mesh (**d**)

Software solutions increase the flexibility of the smart camera, but rely on caches to promote data reuse, which may not be effective [18]. On the other hand, hardware functions and accelerators can decrease the flexibility of the SoC, but are suited to memory customisations to compliment the data movements of the application. This section describes how the memory and communications architecture of smart camera SoCs can be tailored to compliment the PEs of a SoC, as well as the algorithms executing on them.

Meng et al. [14] use a hierarchical communications topology (Fig. 2.5b), where the top tier contains the system level components of the SoC, such as I/O and timers. The second tier is responsible for low-level, high bandwidth image capture and communication tasks. And the bottom tier performs high-level, low-bandwidth data processing tasks. Each tier is connected using a bridge, and localised communications

within each tier is performed over a shared system bus. This topology is useful in defining the high-level blocks of the SoC, and allows designers to abstract themselves during software design: as the use of bridges creates a unified addressing space. The second tier contains a DMA engine that can transfer data to and from the localised external memories of tiers 2 and 3. This limits data accesses over the bridges and improves data locality and temporality. Such a hierarchy is useful when defining the computational arrangement of a SoC and allows designers to focus more on the design and selection of PEs.

Chan et al. [4] create a streaming architecture, similar to Fig. 2.5c, for their smart camera SoC. To efficiently utilise the system bus bandwidth, the hardware is carefully designed to avoid bit-width mismatches. The authors use sub-word level parallelism to address this, where input data is packed and aligned in memory such that the data of any eight aligned neighbouring pixels of an image are stored as a 64-bit word. This data arrangement allows the processing of eight 3×3 sub-windows at eight different PEs, creating a SIMD architecture of SW reconfigurable PEs. This is a low-level approach to extracting data-level parallelism for smart camera SoCs, and in contrast to the three-tier hierarchy of Meng et al. [14] is a fine-grained approach. This fine-grained approach allows a more efficient design of the PEs of the SoC, but can limit the flexibility of the smart camera application: since the PEs perform dedicated functions. However, the memory and communications architecture can be easily reused, and new computational functions could added in a plug-and-play fashion, provided the architecture is designed to accommodate the functions.

A similar stream-based processing architecture is used by Chen et al. [5]. Input data is moved via data streams over a master system bus. Line buffers are used to give one PE access to all pixels of 3×3 sub-window in one clock cycle where required, as opposed to the sub-word level parallelism of [4] that provided this parallelism to many PEs. However, the communications architecture of this work can be reconfigured to create more powerful PEs by changing/combining their interconnections through multiplexing. Reconfiguring the communications architecture increases the level of parallelism and could also be used to switch between different operating characteristics, such as power consumption and throughput. However, the reconfiguration of many PEs may require more complex interconnections and would therefore increase wire congestion of the multiplexers, which would have to be considered by the designer.

The bandwidth of the dynamic region implemented by [17] is limited by the number of interleaved signals the ReCoBus can implement (6 in this case); however hardware accelerators can directly access memory allowing high-speed transfers to and from the dynamic region. Communications between the static and dynamic regions are achieved through bridging and round robin arbitration. This technique does require some static switching hardware to ensure static and dynamic regions can communicate. However, it is a more flexible solution than [5], as the communications architecture supports variable hardware accelerators.

Lacassagne et al. [12] present an associative mesh (Fig. 2.5d) made-up from a grid of PEs that readily communicate with each of their eight neighbours. An application is characterised by an interconnection graph—an asynchronous path where data

can circulate freely from processor to processor. Each PE contains an 8-bit graph register that emulates the presence or absence of an incoming value from one of its eight neighbours. The associative mesh is capable of reconfiguring its purpose and computation without the need to actively reconfigure the interconnections of PEs. This system relies on the profiling of an application to obtain the graph and also requires one PE for each pixel, which could lead to inefficient resource consumptions.

Xu et al. [25] investigate the communications architecture of MPSoCs, where the traditional bus-based model is replaced by a crossbar to give all processors access to input data and memories (Fig. 2.5a). Crossbars are a popular bus due to the ability to connect a large number of master to slaves. For an MPSoC it is important to consider congestion that may arise over shared buses, as PEs use them to access shared resources. Despite contributing to the processing of the same input data, this can be detrimental to performance. Arbitration is used as a means of policing shared accessed to bus and is often inherent in the bus controller. The authors found that arbiter controlled bus transactions performed better than processor controlled ones, and the crossbar provided the greatest average throughput, thus advocating the use of crossbar switches for MPSoC designs.

Based on these examples, there are several key points to consider when designing the memory and communications architecture of a smart camera SoC. Similar to the PEs' requirements, the memory and communications system must compliment the data movements and accesses of the intended application. If the smart camera SoC is intended to achieve a standalone purpose, a highly data-parallel architecture such as [4] can be used, whereas if the flexibility is key, the stream architecture of [5] may be more suited. A hierarchical configuration [14] can be used for programmable PEs, as can crossbar switches [25], and compliments software programming models, which will reduce development time. However, the nature of the application and number and types of PEs and memories required in the system will dictate the overall decision of bus choice. The associative mesh can be considered a combination of the sensing and processing unit of a smart camera SoC, and can be used where high-throughputs are required. Hardware accelerator flexibility is best achieved through dynamic reconfiguration [17], where the memories of the accelerators can be customised to suit the accelerators' purpose. However, care is required to design a communications system that compliments the static and dynamic regions of the SoC.

Designers are free to conceive new memory and communications architecture that satisfy the requirements of the smart camera SoC, such as the hypercube topology of [6] which makes use of standard IP components to implement up to a 64-core MPSoC. However, such an architecture must serve the needs of the SoC and justify the resource consumptions. Table 2.3 summarises the advantages and disadvantages of smart camera SoC communication and memory architectures.

2.4 Communications Unit

Smart cameras have quickly become distributed smart cameras, and introduced several design challenges. Distributed cameras refer to a system of physically distributed

Table 2.3 Summary of smart camera SoC communication and memory architectures

Work	Memory Architecture	Communications Architecture	Pros	Cons	Throughput	Power Consumption	Resource Consumption
[14]	Independent off-chip for tiers 2 and 3	Wishbone system bus per tier	Memory and communications system is scalable	Bridging can introduce additional latencies and reduce bus performance	Low	Low	Low
	128B private SPM, 256B instruction ROM, 64B LUT per PE	Wishbone bridges	Private instruction memory makes cores easier to program	Coherency issues may have to be resolved before runtime			
	32KB and 64KB private BRAMs for tier 3		Private memory simplifies memory coherency	Concurrency is limited to that which the designer implements			
[12]	Custom memory architecture for PEs	Point-to-point inter-PE communications	Globally coordinated SoC Communications resolved before runtime	Resources may be under utilised	High	High	High
	Streaming-based data flow from sensing to processing unit	Associative mesh	High throughput streaming system	PEs always require communications logic to determine the next stage of execution flow			
			Coherency methods not required Scalable	Communications links that are unused in a graph may still be present in the hardware			
[5]	Streaming data input	Reconfigurable interconnections	Design/application flexibility	May not be scalable	High	Low	Low
	Line buffers for increased parallelism	Coordination by main controller	Memory and communications architecture allows PEs to be combined	Increased resource consumption over static architectures			

(Continued.)

Table 2.3 (Continued.)

Work	Memory Architecture	Communications Architecture	Pros	Cons	Throughput	Power Consumption	Resource Consumption
[4]	Streaming memory architecture - data reuse/locality inherent	Communications resolved for the application	High throughput	Application dependent	Unknown	Unknown	Unknown
			Small memory usage	Design inflexibility			
[17]	Dedicated access to memory for HW accelerators	Reconfigurable interconnect	Design/application flexibility	Memory coherency issues may exist	High	Unknown	Unknown
	Flexible private memory architecture(s) for HW accelerators	Bus bridging	Communications for HW accelerators	Reconfiguration overhead			
		Static bus for embedded CPU region	Dedicated memory access for HW accelerators	HW accelerators limited to size of dynamic region available			
		Point-to-point HW accelerator communications					
[25]	Globally shared memory	Crossbar switch-variable size	Arbitration present in bus	Wire congestion can occur for unused links	Medium	Unknown	Medium
			Easy to implement and scale	PEs access memory via shared bus—limiting throughput			
			Suitable for SoCs with DSP/microcontroller based PEs	PE loads may have to be equalised to ensure arbitration does not impact individual PE performance			

camera that may or may not have overlapping fields of view [18], which inevitably increases the volume of input data to be processed [24]. The increase in the field of view of the smart cameras increases the information that can be extracted from an image scene [18, 21, 22]. However, there are several geometric constraints that must be considered when designing a distributed smart camera system [19], including central projection, epipolar geometry, and planar scenes. Bramberger et al. [3] present a distributed embedded smart camera constructed from off-the-shelf components. The TMS320C6416 DSP by Texas Instruments is used as the processors of the SoC and are coupled together with memory via a PCI bus. Network connections are made through Ethernet and Wireless communications.

Hardware accelerators can also be used to optimise the flow of data to and from camera nodes. Zarezadeh and Bobda [26] implement an object request broker (middleware) for distributed smart camera SoCs for FPGA implementation to improve network performance. The middleware is implemented in hardware and can directly access the memory of a smart camera. The performance criteria set out or distributed smart cameras (in terms of the network) are the time taken for packets to arrive at the receiver, the time to prepare and process packets, and the time taken for packets to dissipate through the network. Comparisons of the hardware object request broker to a software broker executing on a PPC 405 show that the hardware broker achieves lower latencies than the software broker—nearly 100x faster, and also achieves higher and scalable bandwidths. This study shows that the use of hardware to create a distributed smart camera network can achieve higher throughputs than networks controlled by software, and more importantly throughputs that are scalable.

The communications unit of a smart camera SoC is very much dependent on the environment in which it will be used. As argued by Rinner and Wolf [18], distributed smart cameras are different from Wireless Sensor Networks (WSNs), as WSNs involve the processing of small amounts of data and are primarily concerned with conserving power where possible. Recent works have created wireless smart cameras such as [23], however with limited functionality. Furthermore, the use of wireless transmission to propagate information from nodes of a distributed smart camera may decrease performance when compared to wired transmissions due to the bandwidth available. The communications unit is primarily concerned with sending abstracted (processed) data to the end user. However, the protocols that can be used for this (USB, Firewire, etc.) is not relevant to the overview and discussion of smart camera SoC sensing and processing units, where image output functionality is often an inherent part of the SoC.

2.5 Summary

Smart camera SoCs are a diverse and complex area of design. Designers must be aware of the smart camera's intended application and its resource requirements. CMOS sensors are the popular choice for the sensing unit due to their ability to be tailored at the hardware level to incorporate pre-processing modules, and their

increasing image quality and capture rate. The computational requirements of the application must be fully satisfied by the SoC's processing unit. This can be achieved in several ways through hardware and software development. Software development suits programmable, flexible solutions with quick time-to-market constraints. Software-based solutions focus on DSPs and implicitly design a memory architecture in software by exploiting data locality through caches. Software solutions must also address the storage and arrangement of program memory for optimal runtime execution.

Designers must decide on the importance of design flexibility, operating characteristics such as resource and power consumption, and throughput. Software solutions will have a nominal power consumption as their architecture does not change, however hardware-based solutions have the ability to modify hardware that directly impacts the power and resource consumption of the SoC—often in a positive way. Hardware solutions can be dedicated hardware designs hand-crafted for the purpose of the smart camera applications. These systems tend to boast high degrees of parallelism and often do not require memories for data reuse, as the system is optimised to use a piece of data once. Hardware solutions can also be more flexible and interfaced with software solutions as accelerators or coprocessors to carry out computation on the software side's behalf. Flexibility can be increased through the use of dynamic reconfiguration, which allows the resources of hardware accelerators to be rearranged into new computational engines.

The memory and communications architecture of such hardware solutions must be carefully designed to decrease memory access bottlenecks. Programmable and reconfigurable hardware accelerators can access data themselves or be provided with data through a PE. In both cases, the link between the PE and accelerator must be optimised for the level of interaction between the hardware and software side. Furthermore, memory accesses can be performed directly by accelerators and greatly reduces memory latencies. However, memory coherency must be considered by the designer and implemented where necessary.

Lastly, creating scalable SoCs for smart cameras, where the number of PEs can be increased to improve throughput, is a maintenance issue for designers and should be implemented in systems where flexibility is required. Scalable memory and communications architectures must be used in these cases such as crossbar switches and hierarchical bus topologies. However, these systems must be carefully designed to prevent the instantiation of unused or inessential resources. Hierarchical bus topologies are very scalable through the use of bus-bridges and tier-local memory buffers, however throughput can be reduced by the levels of arbitration required at each tier. Hardware-only solutions are the least flexible in terms of communications and memory architecture, but can be designed such that hardware modules can be instantiated in a plug-and-play fashion around the memory and communication network. Smart cameras can also be distributed systems, where data is propagated and/or collected from node-to-node. In this case, hardware accelerators have been shown to decrease network latencies and improve the throughput of the system.

References

1. Albani L, Chiesa P, Covi D, Pedegani G, Sartori A, Vatteroni M (2002) VIsoc: a smart camera SoC. In: Proceedings of the 28th European on solid-state circuits conference, ESSCIRC 2002, pp 367–370.
2. Blanc, N., Oggier, T., Gruener, G., Weingarten, J., Codourey, A., Seitz, P.: Miniaturized smart cameras for 3d-imaging in real-time [mobile robot applications]. In: Sensors, 2004. Proceedings of IEEE, pp. 471–474 vol. 1 (2004). doi:10.1109/ICSENS.2004.1426202
3. Bramberger M, Doblander A, Maier A, Rinner B, Schwabach H (2006) Distributed embedded smart cameras for surveillance applications. Computer 39(2):68–75. doi:10.1109/MC.2006.55
4. Chan WK, Chang JY, Chen TW, Tseng YH, Chien SY (2009) Efficient content analysis engine for visual surveillance network. IEEE Trans Circuits Syst Video Technol 19(5):693–703. doi:10.1109/TCSVT.2009.2017408
5. Chen J, Shen CF, Chien SY (2007) Coarse-grained reconfigurable image stream processor for digital still cameras and camcorders. In: Proceedings of custom integrated circuits conference, CICC '07. IEEE, New Jersy, pp 81–84. doi:10.1109/CICC.2007.4405686
6. Damez L, Sieler L, Landrault A, Dérutin JP (2011) Embedding of a real time image stabilization algorithm on a parameterizable sopc architecture a chip multi-processor approach. J Real-Time Image Proc 6(1):47–58. doi:10.1007/s11554-010-0184-3
7. El Gamal A, Eltoukhy H (2005) Cmos image sensors. IEEE Circuits Devices Mag 21(3):6–20. doi:10.1109/MCD.2005.1438751
8. Fons F, Fons M, Cant E, Lpez M (2012) Deployment of run-time reconfigurable hardware coprocessors into compute-intensive embedded applications. J Sign Proc Syst 66(2):191–221. doi:10.1007/s11265-011-0607-9
9. Heyrman B, Paindavoine M, Schmit R, Letellier L, Collette T (2005) Smart camera design for intensive embedded computing. Real-Time Imaging 11(4):282–289. doi:10.1016/j.rti.2005.04.006,www.sciencedirect.com/science/article/pii/S1077201405000239
10. Kerhet A, Magno M, Leonardi F, Boni A, Benini L (2007) A low-power wireless video sensor node for distributed object detection. J Real-Time Image Proc 2(4):331–342. doi:10.1007/s11554-007-0048-7
11. Kruijtzer W, Reyes V, Gehrke W (2005) Design, synthesis and verification of a smart imaging core using systemc. Des Autom Embedded Syst 10(2–3):127–155. doi:10.1007/s10617-006-0069-7
12. Lacassagne L, Manzanera A, Denoulet J, Mrigot A (2009) High performance motion detection: some trends toward new embedded architectures for vision systems. J Real-Time Image Proc 4(2):127–146. doi:10.1007/s11554-008-0096-7
13. Lepisto, N., Thornberg, B., O'Nils, M.: High-performance fpga based camera architecture for range imaging. In: NORCHIP Conference, 2005. 23rd, pp. 165–168 (2005). doi:10.1109/NORCHP.2005.1597015
14. Meng H, Freeman M, Pears N, Bailey C (2008) Real-time human action recognition on an embedded, reconfigurable video processing architecture. J Real-Time Image Proc 3(3):163–176. doi:10.1007/s11554-008-0073-1
15. Moorhead T, Binnie T (1999), Smart cmos camera for machine vision applications. In: Seventh international conference on image processing and its applications (Conference, Publication No. 465), vol 2, pp 865–869. doi:10.1049/cp:19990448
16. Matsushita, N., Hihara, D., Ushiro, T., Yoshimura, S., Rekimoto, J., Yamamoto, Y.: Id cam: a smart camera for scene capturing and id recognition. In: Mixed and Augmented Reality, 2003. Proceedings. The Second IEEE and ACM International Symposium on, pp. 227–236 (2003).doi:10.1109/ISMAR.2003.1240706
17. Oetken A, Wildermann S, Teich J, Koch D (2010) A bus-based soc architecture for flexible module placement on reconfigurable fpgas. In: International conference on field programmable logic and applications (FPL), pp 234–239. doi:10.1109/FPL.2010.54
18. Rinner B, Wolf W (2008) An introduction to distributed smart cameras. Proceedings of the IEEE 96(10):1565–1575. doi:10.1109/JPROC.2008.928742

19. Sankaranarayanan A, Veeraraghavan A, Chellappa R (2004) Object detection, tracking and recognition for multiple smart cameras. Proc IEEE 96(10):1606–1624. doi:10.1109/JPROC.2008.928758
20. Senior A, Pankanti S, Hampapur A, Brown L, Tian YL, Ekin A, Connell J, Shu CF, Lu M (2005) Enabling video privacy through computer vision. IEEE Secur Priv 3(3):50–57. doi:10.1109/MSP.2005.65
21. Trivedi M, Gandhi T, Huang K (2005) Distributed interactive video arrays for event capture and enhanced situational awareness. IEEE Intell Syst 20(5):58–66. doi:10.1109/MIS.2005.86
22. Trivedi M, Huang K, Mikic I (2005) Dynamic context capture and distributed video arrays for intelligent spaces. IEEE Trans Syst Man Cybern Part A Syst Hum 35(1):145–163. doi:10.1109/TSMCA.2004.838480
23. Wang Y, Velipasalar S, Casares M (2010) Cooperative object tracking and composite event detection with wireless embedded smart cameras. IEEE Trans Image Proc 19(10):2614–2633. doi:10.1109/TIP.2010.2052278
24. Wolf W, Ozer B, Lv T (2002) Smart cameras as embedded systems. Computer 35(9):48–53. doi:10.1109/MC.2002.1033027
25. Xu J, Wolf W, Henkel J, Chakradhar S, Lv T (2004) A case study in networks-on-chip design for embedded video. Proceedings of Design, automation and test in Europe conference and exhibition 2:770–775. doi:10.1109/DATE.2004.1268973
26. Zarezadeh A, Bobda C (2010) Performance analysis of hardware/software middleware in network of smart camera systems. In: International conference on reconfigurable computing and FPGAs (ReConFig), pp 196–201. 2010, doi:10.1109/ReConFig.59

http://www.springer.com/978-1-4614-7704-4

Distributed Embedded Smart Cameras
Architectures, Design and Applications
Bobda, C.; Velipasalar, S. (Eds.)
2014, XIV, 273 p. 131 illus., 110 illus. in color.,
Hardcover
ISBN: 978-1-4614-7704-4