Performance modelling and application analysis of a flexible heterogeneous video camera architecture

Janssen, E.

Award date:
2013

Performance Modelling and Application Analysis of a Flexible Heterogeneous Video Camera Architecture

Eric Janssen[(1,2)],
(1) Eindhoven University of Technology, The Netherlands (2) Prodrive B.V., The Netherlands

***Abstract*—A new camera architecture for high-end video cameras is presented featuring a flexible and heterogeneous coprocessor-based architecture, independently executing image operations in parallel suited for an implementation in a high-end 20-Mpixel camera. The dataflow graph and the involved processors are programmable, enabling different image pipeline-paths for different use cases. In order to facilitate smooth function execution and programming of the hardware system, we have developed a system model which estimates performance figures of the function execution. This model is based on the proven Y-chart Architecture Algorithm (YAAM) model. Furthermore, we have added a diagnostic feature, measuring the start and ending times of tasks, to evaluate the temporal and dynamic behaviour of the functional flow graph. To verify that our model alignments with the actual implementation, we have implemented two use cases for the camera system. The applied use cases are based on multi-window processing of signals and a dynamic execution of functions. We have found that the model is within a few percentages of the actual execution performance. The built-in diagnostics prove to be useful for quality-of-service at the level of input rate control.**

Keywords-component; video camera architecture; YAAM; architecture models; built-in diagnostic; programmable co-processor array; performance models; embedded system

I. INTRODUCTION

The development of new and innovative (video) camera designs is triggered by the request for next-generation cameras for video surveillance. A continuous trend can be observed where visual sensors become increasingly sensitive for capturing signals at low-light conditions as well as higher pixel densities are achieved. Current camera systems offer single video streams up to HD resolution or even higher but lack image quality to allow identification of people posteriori, often called forensic zooming. Furthermore, some cameras are equipped with Pan-Tilt and Zoom (PTZ) functionality to zoom into a Region-Of-Interest (ROI), thereby discarding the total overview due to zooming, panning or tilting. These non-reactive cameras may integrate Video Content Analysis (VCA) applications to notify users about e.g. detected motion. Such systems have many disadvantages, of which the most important one is loss of view when panning, tilting and/or zooming. Furthermore, situations with extreme high dynamic lighting may have poor image quality in bright and/or dark regions at the same time.

Another, non-technical argument for improvement is found in cost, i.e. Total Cost of Ownership (TCO). TCO is the sum of purchasing a piece of equipment, making use of it, maintaining it and eventually disposal costs. Technological improvements can lower the TCO of video surveillance systems by reducing the amount of required cameras. Fewer cameras results in a reduction of costs in the infrastructure, maintenance and reduces the power consumption of the back-end (i.e. datacentre).

From this observation, we can derive a number of requirements for a new and innovative camera architecture:

1) A high-resolution sensor rendering analogue PTZ superfluous.
2) Support for flexible and adaptable image-operation paths such that a versatile architecture supports different use cases of the camera.
3) Independent video streams enabling reuse of the camera-sensor for multiple virtual cameras
4) Event detection or VCA (e.g. motion) such that the camera can be responsive to its environment.

Current video cameras employ a single image stream, have static processing paths, and are optimized for the sole purpose of providing video footage. Such systems are heavily optimized for costs.

Figure 1 shows a camera system which features feedback from VCA to image quality control and image operations. Both feedbacks are required to use dynamic image paths e.g. capture image with different sensor settings to enhance image quality in darkness or bright image areas, or use different ROIs to transmit.

One of the main disadvantages of a high-resolution image sensor is the high-bandwidth requirement when communicating image data at a high resolution. The concept of a reactive camera uses Video Content Analysis (VCA) as an event detector to select which images are being sent for further observation. In our case the selection mechanism is triggered by motion analysis in the scene.

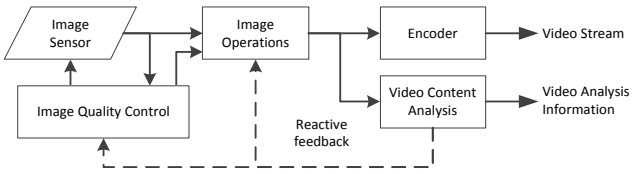

Figure 1: Block diagram of a reactive camera system.

High-resolution cameras challenge the architecture design with high bandwidth and processing demands. The image-operation-pipeline depth highly influences the amounts of memory required. Depending on the reactive feedback, different types of image-operations are required, resulting into dynamic run-time memory usage and processing-load footprints.

The reactive camera contains an electronic device which integrates all video processing. This device has a general-purpose communication interface which allows connecting externally available video processing devices, such as motion detection. This approach allows fast time-to-market and reduces costs.

Current state-of-the-art research on camera architecture shows solutions for smart cameras [1] multiple views [2] and

Table 1: Computing and bandwidth requirements for a high-end camera for a typical set of image operations (GOPS = 10^9 operations per second)

Function	*Operations*	*Input data type*	*Output data type*	*Computations*	*Inbound Memory Bandwidth*	*Outbound Memory Bandwidth*	*Total Memory Bandwidth*
Input grabber	Pixel Sampling, filtering, pre-processing	-	CFA + 1280×960 YCbCr	57.7 GOPS	-	960 MB/s	960 MB/s
Demosaicing	Matrix operations	CFA	YCbCr	0 – 20.2 GOPS	0 - 250 MB/s	0 - 490 MB/s	0 - 740 MB/s
VideoOutput	Video protocol	YCbCr	-	0 – 74 MOPS	110 MB/s	-	110 MB/s
Compression	Various (e.g. wavelet, entropy encoding, quantization)	YCbCr	JPEG2000	0 – 40 GOPS	0 - 480 MB/s	0 - 40 MB/s	0 - 520 MB/s
CA Engine	Communication	X[1]	X[1]	0 – 350 MOPS	0 - 200 MB/s	0 - 150 MB/s	0 - 350 MB/s
MotionDetect	Various	YCbCr	Motion Events	40 GOPS	110 MB/s	0 - 1 MB/s	111 MB/s

Remarks:
1) X is used to denote don't care

high-resolution HD [3] and QUAD HD [4]. However, these solutions have either a static configuration (e.g. no task or graph switching at run-time) or have resolutions up to 4×HD. Our proposed architecture is capable of dynamic runtime switching of the signal-processing graph and can cope with 10× HD (or 64× VGA) images at 30 Hz. But more importantly we provide an accurate model which predicts the system load and throughput. Such a model is important to be able to predict the throughput when simultaneous dynamic views are required, delivered by a single architecture implementation.

The above-mentioned literature suggests that the only major improvement is the increase in sensor resolution. However, at the same time the application scenarios for surveillance cameras have been strongly expended all requiring some different form of processing and video analysis. The purpose of our research is to enable a camera platform that allows flexible re-usage of the system for a multitude of applications, with the development benefit to use a built-in diagnostics feature for fast application analysis prior to implementation.

The remainder of this paper is organized as follows. Section II presents the camera architecture, important system considerations and aspects. Section III elaborates on the image operations typically found in cameras. Section IV describes the hardware models. Section V addresses the quality-of-service and mode adaptions found in dynamic reactive systems in the context of the proposed camera architecture. Section VI covers two typical example use-cases. Section VII gives an overview of the applied test system to validate the models. Section VIII gives the results for both scenarios compared to the test-system. Section VIII.B.1) concludes this paper with future work proposals in Section X.

II. SYSTEM ARCHITECTURE

Although mapping all image processing functions of a camera system into software yield a highly flexible architecture, this concept is rarely used as it is too expensive for large-scale applications. This is due to increase complexity of image processing and coding and higher resolutions of visual sensors.

A heterogeneous approach offers more optimizations for the application-specific tasks [5], which lead to clearly lower system costs. Therefore, our system architecture modelling involving performance analysis, is based on heterogeneous camera systems. Our performance-estimation approach is based on the Y-chart methodology [6], the roofline model [7] and the Architecture Algorithm Model (AAM) [8]. The Y-chart methodology is based on the observation that typically development team work involves the joint development of an application, a platform (hardware) and the mapping of the application onto the platform. This allows us to reason about the individual aspects and the system as a whole.

The roofline model describes a "roof" for the intrinsic performance of a processing unit in terms of communication bandwidth and computational complexity. Analysing an application delivers the computation complexity and its corresponding communication requirements per processed input unit, e.g. as in [7] and [9]. Let us first analyse how much computing power and memory is involved in a set of typical high-end high-resolution image processing tasks for a corresponding camera system.

Typical image operations for high-end cameras are shown in Table 1. For high-end ultra-HD, we assume the camera has 20 megapixels (Mpixel) at 30 frames/second with 12 bits/pixel (B_{pp}) and YCbCr 4:2:2 colour images. The details and usage for each of the functions is explained further in Section III. The numbers from Table 1 are the result of research and project evaluations of the company Prodrive B.V., which is active in high-end camera design. From Table 1 we can draw the following conclusions.

- Each processing function has asymmetric bandwidth footprints i.e. the ratio between input and output bandwidth is clearly unequal to unity.
- Regularly, the input data type does match with the output data type, and not all outputs are compatible to the input of other processing functions.
- It may be possible that a processing function does not perform actual operations on the video data, but is assisting in communication of it.
- Most functions have variable memory bandwidth characteristics depending on the Region-Of-Interest (ROI) requested.

From the above analysis and Table 1, we clearly observe that for different use cases involving different ROIs and various image operations, quickly more than 160 GOPS are required with a memory bandwidth consumption of more than 3 GB/s. A typical high-end x86 processor is capable of only delivering several tenths of GOPS. From this observation, it is reconfirmed that, as already mentioned, an application-specific architecture is simpler, more power efficient and more cost-effective when compared to general-purpose computing platforms.

This survey delivers a good insight which is required to construct a new flexible architecture for high-end camera systems. The processing functions are composed of a mixture of operations and are heterogeneous in nature. Furthermore, we can expect different bandwidths and different image processing dataflow graphs for different use cases. Flexible reuse of the architecture requires now that the dataflow graphs becomes also programmable, which involves additional

aspects like buffering, synchronization and control. The system and application control can be outsourced to a low-power low-cost CPU. Furthermore, due to the high inter-data dependency of the image processing functions, the application processor requires a large and local private memory. The communication-assist function offloads from the CPU to a hardware-specific function for the video communication, featuring reliable streaming of video and image data. A block diagram of the proposed architecture is depicted in Figure 2. The architecture consists of a CPU controlling a coprocessor Array (COPA), executing the image operations.

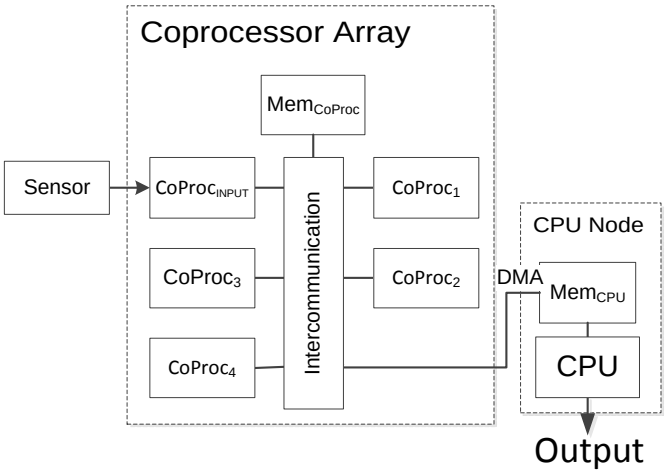

Figure 2: Flexible reusable architecture for a camera platform.

Selecting a task-specific coprocessor solution releases the CPU from intensive computations, while the high-bandwidth communication is between the coprocessor and its local memory. If in- and output formats are compatible, virtually any order of operations can be programmed, also by modifying the access order of the CoProc modules. Once all operations have completed, the resulting image can be transferred to the CPU. In this architectural concept, the CPU only has to manage the tasks it assigns to the coprocessors and control the dataflow from image to its output.

In the following section, the image operations are analysed in more detail such that the corresponding coprocessors are defined.

III. CAMERA IMAGE OPERATIONS

This section describes the typical camera image operations (Table 1) in more detail. For each operation, a computational and bandwidth model is given. Table 2 gives an overview of the used mnemonics in the mathematical models.

Table 2: Overview of model parameters and functions.

Mnemonic	*Description*
Ops	Operations
$N \times M$	Pixels
B	Bandwidth
t	Time
B_{pp}	Bits per pixel
D	Data bus width
C	Cycles

The computational model is expressed in operations per image and denoted with O. The bandwidth model is expressed in MByte/s (abbreviated hereafter as MB/s) and denoted with parameter B. The amount of pixels is expressed by $N \times M$ pixels.

A. *Input grabber*

A typical image sensor is a CCD or CMOS chip, which is sensitive to photons in the visible light spectrum to humans and converts the light to the time-discrete digital image signal. A digitized image frame is characterized by the spatial resolution of the sensor, throughput rate (assumed linear to its communication frequency) and amplitude resolution per pixel, expressed in bits per pixel (B_{pp}). Furthermore, image quality is mainly characterized by the signal-to-noise ratio and image contrast.

The image processing input is concerned with acquiring the digital image from the sensor, which is abbreviated as "input grabber". Whereas conventional cameras have resolutions in the order of HD (2 Mega pixels, abbreviated as Mpix), we wish to achieve a tenfold of that resolution. Effectively, this means that the sensor bandwidth scales linearly to the communication frequency with a factor 10. The following set of equations describes the model for the input grabber.

$$P_{ff} = 5120 \cdot 3840 \approx 20\ [MP] \qquad (1)$$
$$P_{rs} = 1280 \cdot 960 \approx 1\ [MP]$$

$$B_{Sensor-to-coproc} = F_{sensor} \cdot N_{channels} = 480\ Mhz \cdot 16 = 960\ \left[\frac{MB}{s}\right] \qquad (2)$$

$$B_{sensor-output} = \left(P_{ff} \cdot \frac{12}{8} + P_{rs} \cdot \frac{24}{8}\right) \cdot \frac{FPS}{10^6}\ [MB/s] \qquad (3)$$

$$F_{images\ per\ second} = \frac{B_{sensor}}{P_{ff} \cdot bpp} = 32 \qquad (4)$$

$$O_{input} = 88 \cdot P_{ff} + 35 \cdot P_{rs} \qquad (5)$$

By default, the input pre-processor delivers two image formats from the sensor: the original full-size 20 MP image and a scaled version (Equation (1)). From Equations (2), (3) and (4), we derive the bandwidth requirements for the input grabber and the sample frequency for the sensor given the number of communication channels to the sensor and its operating frequency. These numbers describe the first requirements for CoProc$_{input}$ and its interface to the global memory.

The amount of operations required for a single image is modeled in Equation (5). The constants 88 and 35 result from analysing various algorithms used in the pre-processing steps, typically found in cameras (Table 1).

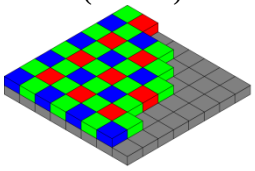

Figure 3: Typical image sensor CFA pattern.

B. *Demosaicing*

An image sensor is typically overlaid with a Colour Filter Array (CFA) pattern such that pixels are sensitive to a limited spectral frequency, i.e. blue, red or green, as depicted in Figure 3. To reconstruct an image from the CFA, an operation called demosaicing or sometimes called de-Bayering, is

required. Demosaicing reconstructs a colour representation with three channels per pixel (e.g. YCbCr). The demosaicing process is visually depicted in Figure 4.

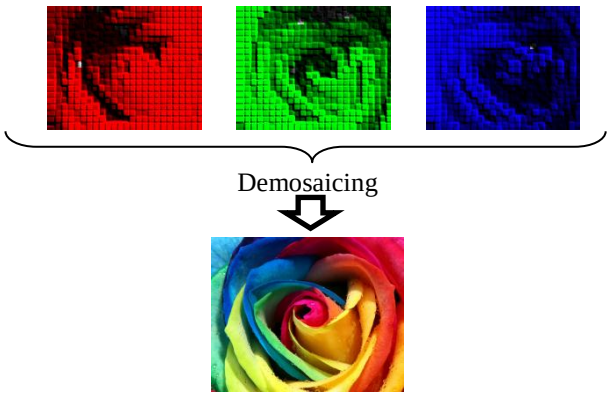

Figure 4: Demosaicing process going from CFA to a reconstructed full colour image.

The bandwidth requirements for demosaicing depend on the amount of requested pixels per second and the demosaicing duration ($t_{demosaic}$, Equations (6) and (7)).

$$B_{demosaicing-input} = \frac{\frac{12}{8} \cdot (N \times M)}{t_{demosaic}} \; [B/s] \tag{6}$$

$$B_{demosaicing-output} = \frac{\frac{24}{8} \cdot (N \times M)}{t_{demosaic}} \; [B/s] \tag{7}$$

$$O_{demosaicing} = 123 \cdot (N \times M) \; [Ops] \tag{8}$$

$$t_{demosaic} = \min\left(\frac{O_{demosaicing}}{O_{CoProc}}, \min\left[\frac{B_{demosaicing-input}}{B_{allocated-input}}, \frac{B_{demosaicing-output}}{B_{allocated-output}}\right]\right) \tag{9}$$

The model of the CoProcessor uses streamlined I/O registers for data capturing and outputting. If the clock frequency of the CoProcessor is increased, the data fetching and outputting speeds increase accordingly, so that the I/O bandwidths do not have to be incorporated in Equation (9) as an operation. Equations (6) and (7) provide upper bounds for the bandwidths depending on the computational speeds of the coprocessors. The amount of required operations for the demosaicing function is modelled with Equation (8). The constant 123 is extracted from analysing a linear demosaicing algorithm, using a combination of 3×3 and 5×5 matrix kernels. In Equation (9), the *min* function is used such that the application is under the *roof* of the roofline model, i.e. it is either constrained by bandwidth or computational limitations, or both if around the pivot point of the roofline model.

C. *Compression*

Typically every camera offers image compression to reduce the required external bandwidth. Without compression a high-resolution camera would easily saturate a 1-Gbit/s IP network link and only achieve 2.1 frames per second, if we assume full-frame YCbCr-encoded images. Obviously, we could select another type of interface, however, then the camera would become less attractive as it would require a significant amount of video storage (e.g. 24-hr. recording). Image compression easily reduces the bandwidth exploiting spatial and temporal redundancy.

JPEG2000 is a compression standard gaining interest in the professional video and imaging industry (e.g. digital cinemas use this type of compression for 4K movies). **B**andwidth is typically reduced by at least a factor 10 (high-quality) up to 100 (low-quality). The encoding time (or number of required cycles) is content-dependent, e.g. images with high-entropy require more processing cycles.

In our camera system, the JPEG2000 encoder is implemented for one of the coprocessors in hardware (i.e. programmable logic) and is able to process up to 120 Mpixel/s. This processing power can be used for various image sizes, such as 1280×960 pixel frames, and can be processed at roughly 100 frames per second. However, when processing full-frame images, the frame-rate lowers to $120/20 = 6$ frames per second (Equation (11)). The following equations describe the bandwidth requirements for the compression unit, when starting at computing the pixels involved for processing.

$$P_c = \sum_{i=0}^{N-1} ROIwidth_i \cdot ROIheight_i \tag{10}$$

$$\frac{P_c}{120 \cdot 10^6} \leq T \leq \frac{1\frac{1}{3} \cdot P_c}{120 \cdot 10^6} \tag{11}$$

$$B_{JPEG2000-input} = \frac{bpp \cdot P_c}{T} \tag{12}$$

$$B_{JPEG2000-output} \leq \frac{B_{JPEG2000-input}}{F_{compression}} \tag{13}$$

$$O_{compression} = P_c \cdot 200 \tag{14}$$

$$t_{compression} = min\left(\frac{O_{compression}}{O_{CoProc}}, min\left[\frac{B_{JPEG2000-input}}{B_{allocated-input}}, \frac{B_{JPEG2000-output}}{B_{allocated-output}}\right]\right) \tag{15}$$

Equation (10) describes the amount of pixels selected from a region of interest for compression. Equations (12) and (13) give the bandwidth requirements for the compression function. Parameter T is used to denote the timespan to compress the images, the factor $1\frac{1}{3}$ provides a worst-case upper bound for low-entropy images. Equation (13) provides an upper bound for the output bandwidth, since low-entropy images can often be encoded using fewer symbols. The timing model is given by Equation (15), which uses the computation complexity from Equation (14). Again the *min* function is applied such that it operates under the roof of the roofline model.

D. *Video output*

To allow easy integration into other video processing systems, a video output port is provided. A video port is a digital output for streaming video often used in industrial applications, which connects e.g. a TriMedia processor to the COPA for further image processing functions. In our case, we use it to connect an H.264 encoder + motion detector (see Figure 5).

Equation (16) denotes the bandwidth requirements for a continuous video stream at frame rate F_{video}. Equation (17) denotes the amount of required operations. The timing model is constructed from Equation (18) which includes the roofline model. The computed time (t_{video}) describes the effective operation time required.

$$B_{video-input} = N \times M \cdot \frac{B_{pp}}{8} \cdot F_{video}[B/s] \quad (16)$$

$$O_{video} = P_{rs} \cdot 2 \cdot F_{video}\ [Ops] \quad (17)$$

$$t_{video} = min\left(\frac{O_{video}}{O_{CoProc}}, \frac{B_{video-input}}{B_{allocated-input}}\right) \quad (18)$$

E. *Motion Detection*

As has been explained in the introductory section, we employ motion analysis for selecting the communication of images. This motion analysis involves a motion detector which is described below.

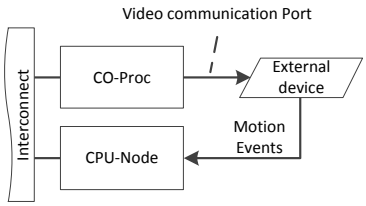

Figure 5: Coprocessor featuring a video communication port to connect external (legacy) hardware

Motion detection is a broadly researched field and the industry has many ASIC [1] implementations for motion detection. The COPA has been designed to connect such available processors for specific processing tasks, such as motion detection. The video port from Section D is used (Figure 5) for that purpose.

A function block diagram of the video communication port is depicted in Figure 5. An external device implementing motion detection is connected to the video communication port which sends the motion events to the controlling CPU-node.

An important property for a reactive camera is its response (latency) time defined by Equation (19) which will be discussed in the hardware model in Section IV.

F. *Communication Assist*

Ultra-high-definition images (UHD) are significantly larger in terms of data size when compared to regular SD or HD images. To prevent the CPU from wasting expensive cycles for communicating images, a Communication Assist (CA) is proposed. The CA has several advantages [10]:

1) The worst-case execution time of a task is decoupled from the communication.
2) The CA can decrease the worst-case execution time (instead of stalling the processing, the CA is stalled until the communication infrastructure accepts the data).
3) The communication infrastructure can be designed for the average communication bandwidth requirements, because the CA can send the data in small messages at a regular interval, whereas in the architecture without a CA the communication infrastructure is designed to absorb the communication bursts as fast as possible.

Additionally, by adding a data-request queue to the CA, the transactions can be pipelined improving its efficiency while reading from the memory (e.g. burst reads). The model for the CA is constructed using the following equations.

$$B_{CA-input} = \frac{F_{rate} \cdot N \times M \cdot B_{pp}}{8}[B/s] \quad (19)$$

$$O_{CA} = \left\lceil \frac{B_{CA-input}}{4} \right\rceil [Ops] \quad (20)$$

$$t_{ca} = min\left(\frac{O_{CA}}{O_{CoProc}}, \frac{B_{CA-input}}{B_{allocated}}\right) \quad (21)$$

A CA does not implement a particular operation in terms of computational operations to an image, it merely transfers the image. The frame rate (F_{rate}) is sometimes variable since it may depend on the amount of frames selected by the motion detection. The amount of computations expressed in ops is modelled using Equation (20), in which 4 Bytes require 1 operation (read/write) to the memory. The throughput of the CA is thus determined by the minimum (Equation (21)) of the allocated memory bandwidth defined in Equation (19) and the computation throughput (Equation (20)).

IV. HARDWARE MODEL

This section describes the hardware model for the COPA architecture. In the previous sections we have discussed the individual functional requirements per functional block.

To enable dynamic image pipelines i.e. a reactive camera proactively changing image paths, a solution has been chosen that uses a coprocessor array (COPA) sharing memory and an inter-communication infrastructure (depicted in Figure 2). To obtain performance estimates for each of the image functions, we have to map the function onto a coprocessor. Each function is mapped to a particular coprocessor such that the model for the actual COPA operation is relatively simple: one function per coprocessor and the COPA total is bounded by the number of coprocessors, both in operation and bandwidth. The roofline model which uses the *min* function in the previous sections, restricts both bandwidth and computational speed limited by the modelled available hardware resources. This means that after the mapping, the throughput of each function is upper-bounded by the roofline model incorporating both bandwidth and processing aspects.

$$t_{videoport} = \frac{P_{video} \cdot B_{pp}}{F_{videoport} \cdot D_{width}}[s]$$

$$t_{motiondetect} \approx 33[ms]$$

$$t_{asic-cpu-node} \approx 33[ms]$$

$$t_{cpu-node} = \frac{C_{event}}{F_{cpu-node}}[s]$$

$$t_{latency_{MD}} = t_{videoport} + t_{motiondetect} + t_{asic-cpu-node} + t_{cpu-node}[s]$$

$$t_{motion-detect} = t_{latency_{MD}} \quad (22)$$

This analysis requires a hardware model which describes the resources offered by the COPA. The CPU-COPA architecture has the following upper bounds for processor-to-processor and processor-to-memory communication.

[1] Application-Specific Integrated Circuit: an integrated circuit (IC) customized for a particular application

$$B_{Mem} = \frac{F_{bus} \cdot D_{width}}{8} [B/s] \quad (23)$$

$$B_{CoProc-CPU} = F_{coproc-cpu-bus} \cdot 8/10 \cdot 1/8 \quad (24)$$

$$O_{coprocessor} = F_{CoProc} \cdot P_{ops} \quad (25)$$

The maximum bandwidth to memory (CPU or COPA) is defined by Equation (23). The bandwidth between COPA and CPU is defined by Equation (24) and facilitates an 8-10 bit encoding mechanism to transport the clock signal along with its communications. The computational performance upper bound of the coprocessors is described by Equation (25). Typically a coprocessor has a core frequency (F) at which it operates and an amount of operations it can process in parallel (P_{ops}). Image processing is often stream-oriented processing with identical operation(s) for each pixel such that parallelism can be exploited.

Table 3 lists an overview of the values for the hardware model parameters used to compute performance parameters of the COPA. The values are equal to the hardware of the test system as described in Section VII. From the numbers we can observe that most application-specific processors within the COPA have low operating frequencies with high parallelism in the executed operations per cycle. Only the communication channels (e.g. DDR memory) have high operating frequencies up to 2.5 GHz.

V. QUALITY-OF-SERVICE

The integrated camera contains multiple functions for high flexibility in application, but not all functions can be used simultaneously. This is because those functions are already embedded in the programmable hardware platform. To overcome this issue we may introduce Quality-of-Service (QoS) when the requested computations cannot be directly implemented. QoS controls the quality of individual functions such that the desired functionality just fits to the computation power of the execution platform. In our case, QoS requires a controller based on a feedback mechanism to control the quality of individual functions. For example, a QoS component can guarantee real-time properties of services (e.g. an overview video stream) by reducing some other function in computation power. Furthermore, to enable QoS in a smooth way, we introduce built-in diagnostics for each function to monitor and analyse the system's behaviour. Section VI introduces two example scenarios. The built-in diagnostics are used in Section VIII to display the temporal behaviour of the camera functions and the influence of the QoS.

A. *Built-in diagnostics*

To analyse the temporal behaviour of the functions executed at the coprocessor, a diagnostic component is implemented as a standard function in programmable hardware. This component makes use of timestamps. The timestamps denote the arrival time of a request and the completion time of the involved task. Each coprocessor is given a job queue, which is scheduled by the CPU (Figure 6).

Once a job enters the job queue a timestamp is generated, while a second timestamp is generated after completion. The difference between the two timestamps yields the actual execution time of the processing.

Table 3: Overview of hardware model parameters

Parameter	Value	Unit
CPU-Node		
F_{bus}	667	MHz
D_{width}	32	b
$B_{CPU-Mem}$	2.7	GB/s
F_{CPU}	1.3	GHz
D_{width}	32	b
CPU – COPA		
$F_{coproc-cpu-bus}$	2.5	GHz
$B_{CoProc-CPU}$	250	MB/s
COPA – Memory		
F_{bus}	800	MHz
D_{width}	32	b
$B_{CoProc-Mem}$	3.2	GB/s
Coprocessors		
$F_{CoProc_{input}}$	160	MHz
$F_{CoProc_1-demosaic}$	120	MHz
$F_{CoProc_2-resize}$	160	MHz
$F_{CoProc_3-compression}$	130	MHz
F_{CoProc_4-CA}	120	MHz
$F_{CoProc_5-video}$	120	MHz
$P_{CoProc_{input}}$	360	Ops
$P_{CoProc_1-demosaic}$	130	Ops
$P_{CoProc_2-resize}$	34	Ops
$P_{CoProc_3-compression}$	350	Ops
P_{CoProc_4-CA}	3	Ops
$P_{CoProc_5-video}$	1	Ops

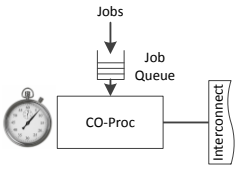

Figure 6: Coprocessor diagnostics, indicating timestamps for job arrivals and completions.

B. *Quality: Prioritizing*

Quality control is only required if a (sub-)system's throughput reaches the roof of its roofline model. This is called an overload, i.e. there is simply too much *load* for the system. Once a system enters the overload condition, FIFOs and work queues tend to fill up. If a system is in overload for a sustained period, the FIFOs and job queues saturate such that no other work can be committed to coprocessors and the system becomes partly halted. We would like to prevent this scenario to happen. For our system, we distinguish three types of overload situations:

1) External overload,
2) Static systematic overload,
3) Dynamic overload.

The first type of overload is a scenario in which the system receiving the camera output cannot accept input from the camera. The second type of overload occurs when the static (e.g. non-reactive) configuration yields a performance request that cannot be handled by the available resources. An example is a video stream at 100 frames/s, while the sensor delivers a maximum of 30 frames/s. Such a situation can be prevented by computing the resource utilization in advance.

The third type of overload is dynamic and is caused by the reactive nature of the proposed high-end camera. It occurs when a plurality of jobs is activated (or overlapping) in time. If jobs are running well distributed over time without overlap, the system can execute all jobs, although the combined load exceeds the available systems resources.

Our QoS should prevent Types 1 and 3, and we assume that Type 2 can be prevented by simply constraining the resource utilization at design (or configuration) time. Type 1 and 3 are solved by dropping frames in FIFO order with a greedy algorithm, based on assigned priorities to jobs. This approach allows certain jobs (e.g. views) to have higher priorities than other jobs. The high-priority jobs are guaranteed to be uninterrupted, whereas low-priority jobs may be delayed or supplied with less input data.

VI. EXAMPLE USE CASES

This section introduces two use cases which are employed to verify the model in Section VIII. For both use cases, (dynamic) flow diagrams are presented

A. *Use-case 1: Petrol Station Surveillance*

The first use case is inspired by the well-known problem of petrol theft. Figure 7 shows such a petrol station in which several pumps are available for service. A regular HD-camera does not capture sufficient detail to allow reading the license plates or identification of faces. Placing additional cameras is a costly operation. With our proposed reactive high-resolution camera, several ROIs can be processed, resulting in an overview and the three additional windows in Figure 7 (red, yellow, green) with high detail.

Figure 7: Example Use-case 1 - Petrol Station Surveillance

This use case is implemented with a static assignment of four ROIs. Furthermore, we provide an additional 20-MPixel image for high detail overview of the complete scene. In this use case, the low bandwidth H.264 and four ROIs can be captured and recorded, while PTZ functionality is enabled by providing a high-resolution image which is not required to be recorded such that less storage space is required. The flow-graph for Use-case 1 is depicted in Figure 8.

In Figure 8, the image path is split right after the image is captured from the sensor into two paths: one for an overview image and the other for high-detail windows selected from the overview. The latter path conveys images with various sizes, depending on the windows and the corresponding parameters. In this use case we employ four SD (640×480 pixels) windows and a 20-Mpixel window.

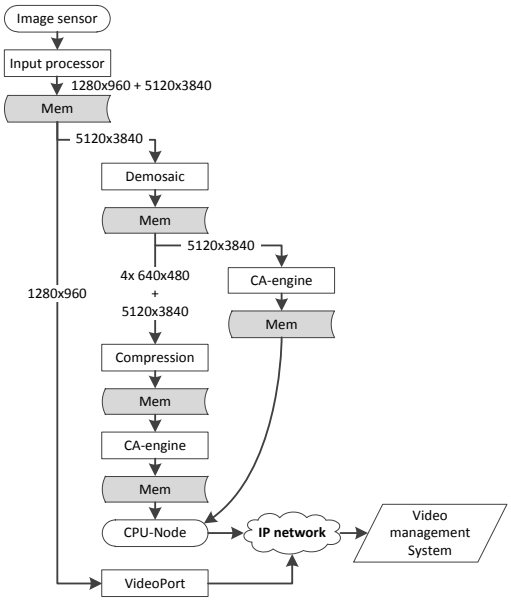

Figure 8: Flow-graph for Use-case 1

B. *Use-case 2: Construction Site Surveillance*

The second example shows the reactive properties of the proposed architecture. This use case is based on surveillance of a construction site. During night time, most of these sites are typically abandoned, making them vulnerable to theft of tools/equipment and vandalism. Long periods of time may not show any event such that video recording is switched off. At the onset of a serious event, recording is started and images are captured containing e.g. activities with identifiable persons or cars.

For this use case we employ the integrated motion detection to record activities. An example is shown in Figure 9 in which a camera is used for delivering a scene overview.

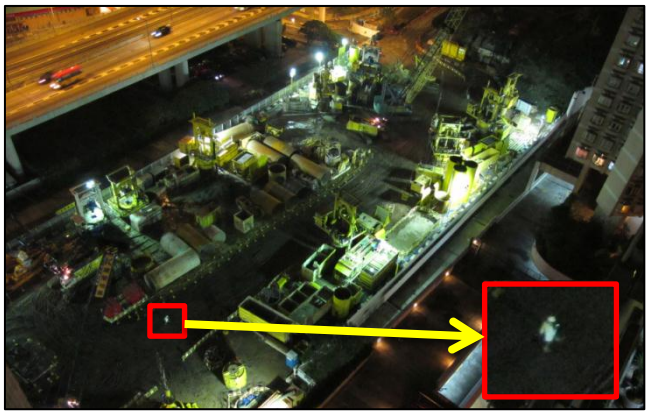

Figure 9: Use-case 2 with motion detection.

Under normal circumstances, only a regular video stream is created, whereas in case of motion detection, several high-quality images are captured and supplied to the video monitoring system. In such a case, the bandwidth is significantly increased for a certain period of time.

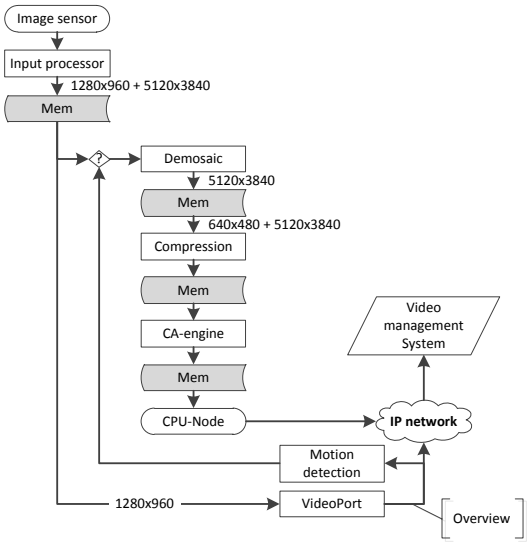

Figure 10: Flow graph for Use-case 2

The flow graph for Use-case 2 is depicted in Figure 10. A conditional flow graph is shown to denote the reactive feedback from the motion detector, which influences the image path. In this use case, a 20-Mpixel image is transferred, including a 640×480 pixel ROI of the area with motion.

VII. TEST SYSTEM

To verify the results derived from the mathematical model, a comparison is made with a high-end camera implementation. With this implementation, we execute the system for the two proposed use cases. The test system is depicted in Figure 11 and features a 20-Mpixel sensor and an implementation of the CO-Processor Array (COPA) and a CPU node.

Figure 11: High-end camera test system.

The test system has a power consumption of 20 Watts, resulting in approximately 8 GOPS/Watt. For the evaluation, we have developed a new programmable application implementing the two use cases, featuring dynamic multi-window processing and motion detection. The application is designed to execute a user-defined flow graph which describes the image-data path(s). This application tool is used to extract the utilization, throughput and temporal behaviour of the system. The results of these measurements are compared to the results from the model in section VIII.

VIII. RESULTS

This section covers the results from the model-based performance estimation, the actual system behaviour and system-level diagnostics. The roofline model [7] is used to determine a feasible point according to the performance of the mathematical system model. For both use cases, we have predicted the performance results in terms of throughput based on the model provided in Section III (task models) and Section IV (hardware model). The model results are compared to the results from the actual camera system described in Section VII.

A. *Use-case 1: Petrol Station Surveillance*

This case describes the results from the model, the test system and the timing diagnostics for the surveillance application.

1) *Model performance*

The results of the model-based estimation for Use-case 1 are denoted in Table 4. From Table 4 we observe that the majority of the memory transactions are within the COPA, thereby relaxing the CPUs memory bandwidth usage. According to the model, the system is capable of providing a video stream with 30 frames/s, four additional ROIs and a supplementary high-resolution image for PTZ functionality.

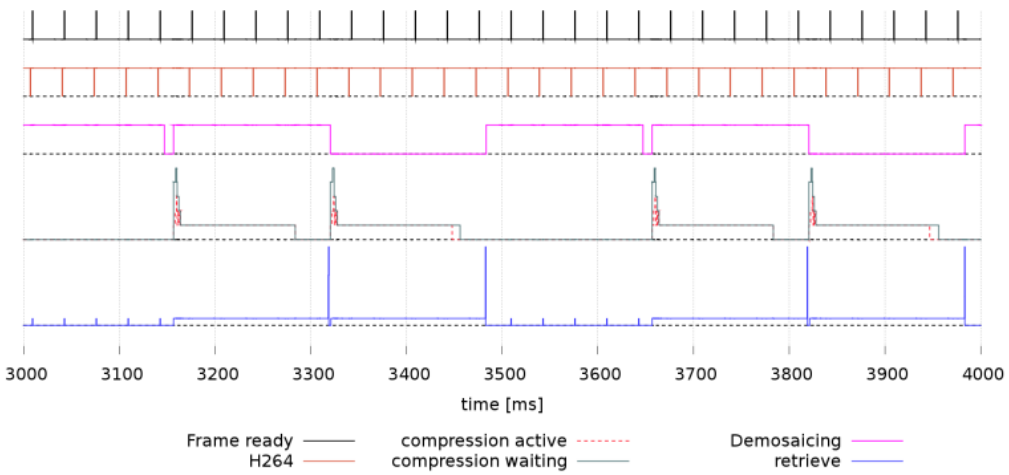

Figure 12: Use-case 1 – temporal behaviour in a timing diagram.

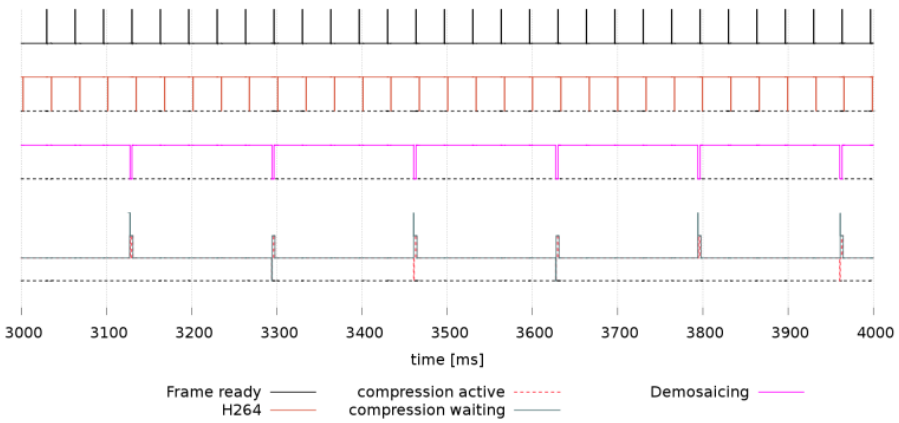

Figure 13: Use-case 2 – temporal behaviour in a timing diagram

2) ***Implementation results***

The implementation results for Use-case 1 are given in Table 4. The table results show that the system is well capable of providing the required video stream at various resolutions. The input and video processors both show high utilization as predicted in the model, but provide a 30 frames/s video stream. The CA module and compression processor show slightly less utilization. Due to the pipelined implementation the model is conservative. Summarizing, the model is well capable of describing the camera system performance for this use case. Using the diagnostics feature, we can also inspect the temporal behaviour.

3) ***Implementation Temporal behaviour***

The internal diagnostic system (Section V.A) allows us to capture the activities and display this as a timing diagram. The timing diagram of Use-case 1 is depicted in *Figure 12*. This figure displays a section of the captured second. *Figure 12* shows that in one second, 30 frames are captured, processed and transferred to the video port. At a lower frequency (4 Hz), demosaicing of frames is performed. The frames appear to be processed in bursts of two frames. This is due to the latency introduced by the frame retrieval which can only process one frame at a time. After arrival of the frame information (e.g. spike at 3482 ms) a new demosaicing task is started. This aliasing issue occurs when the CA co-processor is busy, while the application is waiting for the next frame information.

After demosaicing, 5 different images (4 ROI, 1 full-frame) are compressed. This is depicted in *Figure 12* by two lines: active and waiting. Initially, a frame enters the waiting work queue. Once the co-processor is ready to accept a frame, the processing task becomes active. From *Figure 12* it can be noted that the work queue quickly ramps-up and ramps down due to the four ROI images. The full-frame image takes considerably more time to process. Parallel to compression, the image after demosaicing is retrieved by the CA-engine and sent to the CPU.

From the complete timing diagram we observe that the temporal behaviour has a repetitive pattern which also can be noticed in the depicted time interval in *Figure 12*.

B. Use-case 2: Construction site surveillance

The results of the model-based estimation for Use-case 2 are given in Table 5.

1) ***Model performance***

According to the model, the system enters an overload period due to compression tasks for the images. The bandwidth of the COPA to private memory (COPA-Memory) is exploited to reduce the memory bandwidth between CPU and its private memory (CPU-Memory).

2) ***Implementation results***

The implementation results for Use-case 2 are given in Table 5. Comparing the table results between model prediction and measured performance, again we conclude the model is somewhat conservative. Furthermore, the required bandwidth for CA is low, which saves bandwidth when transporting the images.

Table 4: Use-case 1 model-based analysis results

Parameter	Model prediction	Measured
Throughput		
Input, Video	30 [Hz]	30 [Hz]
Demosaicing	4 [Hz]	4 [Hz]
Compression (4×640×480 + 5120×3840)	16 + 4 [Hz]	16+4 [Hz]
CA	147 [MB/s]	150 [MB/s]
Utilization – communication paths		
COPA-Memory	71%	70%
CPU-COPA	73%	65%
CPU-Memory	11%	10%
Utilization – computational		
Input	92%	93%
Video	98%	98%
Demosaicing	64%	64%
Compression	69%	52%
CA	73%	65%

Table 5: Use-case 2 model-based analysis results

Parameter	Model prediction	Measured
Throughput		
Input, Video	30[Hz]	30 [Hz]
Demosaicing	6 [Hz]	6 [Hz]
Compression (5120×3840 + 640×480)	6 + 6 [Hz]	6 + 6 [Hz]
CA	30 [MB/s]	30 [MB/s]
Utilization– *communication paths*		
COPA-Memory	83%	80%
CPU-COPA	15%	12%
CPU-Memory	2%	2%
Utilization – computational		
Input	92%	92%
Video	98%	98%
Demosaicing	96%	96%
Compression	105%	99%
CA	15%	15%

1) Implementation Temporal behaviour

For Use-case 2, a timing diagram is depicted in *Figure 13*. Again frames are captured at 30-Hz frame rate in *Figure 13*. It is clear that the system is almost completely used for 100%. The compression processor is only occasionally idle around 3300 ms, 3450 ms and 3630 ms.

IX. CONCLUSION

We have presented a new flexible heterogeneous high-end camera architecture, featuring dynamic programmable image processing paths suited for ultra-high definition 20-Mpixel images. The architecture of the system is based on an array of application-specific coprocessors. Due to the programmable signal flow graphs, defining the image operations, a highly flexible versatile camera is created, applicable to many scenarios. Furthermore, we have introduced a simple model for performance estimation of the camera system. Additionally, we have also discussed two use cases, exploiting the cameras features in different ways. The estimations are compared to an implementation of the camera architecture. Due to the various possibilities of mapping and flexibility in programming, we have investigated a model for system analysis and performance based the integrated image processing functions.

The proposed model is based on the Architecture Algorithm Model (AAM), the Y-chart methodology and the roofline model. Our model uses relatively simple equations to express the systems throughput and performance in terms of frame rate, bandwidth, timing estimates for throughput and operations count. We have shown that a relative simple model (without cycle or cache dependencies) based on native signal processing operations delivers a reliable framework to predict dynamic system behaviour based on the proposed COPA. The framework was already tested for other applications. In our case we have evaluated this model for the heterogeneous camera system and integrated the roofline model to define a high-throughput high-performance point for the camera system executing an arbitrary use case. This performance point is guided by the integrated roofline model based on the pre-programmed video functions.

We have validated the model by implementing two typical use cases: a petrol station surveillance case using multiple video windows requiring bandwidth, and a construction site surveillance case with more dynamic task processing. We have found that the model provides performance estimates that are close to the actual execution performance of the system. Typically, the model produces performance numbers which are within a few percentages of the actual performance figures. The model performance is relatively reliable due to the strict nature of the pre-programmed video functions. However, we have found that the control of the COPA and its embedded functions is critical for the performance, and requires a deterministic approach for task switching and meeting the timing deadlines for execution of those tasks.

We have added a feature to the architecture that is beneficial for assessment of the dynamic behaviour of the COPA function execution. To this end, we have integrated a diagnostic tool that measures start and ending of a computing task and accumulates these values for all functions in software. This allows us to analyse the run-time behaviour of the COPA functions. The system allows the display of the timing of function execution in a graphical diagram. This concept also has been validated in the use cases experiments. These experiments have shown that this provided a useful tool for design time analysis of flow graphs, particularly for quickly localizing performance bottlenecks. Performance analysis is also helped by a Quality-of-Service (QoS) component, offering a run-time control of the input rate for each job. The QoS uses the same diagnostic feature as previously mentioned.

The camera platform is mainly constrained by bandwidth. The application-specific coprocessors are running at a fraction of the maximum clock frequencies while the communication channels are running at considerably higher frequencies. This is a clear advantage of the chosen architecture, because power consumption is now saved when the use case allows it. When looking at the utilization numbers the footprints for both bandwidth and computations are comparable. From this we can conclude that if increased throughput (or even higher resolutions) with more functionality are required, then the memory communication architecture needs to be upgraded.

The system has been extensively tested in the field for periods ranging from several weeks up to four months of continuous operation. During the field tests, the system showed no irregularities and was capable of providing a continuous video and image stream. Because of the success of these tests, the camera system is further commercially developed and extended in a new business unit called Ampleye.

X. FUTURE WORK

To extend the functionality of the proposed high-end camera, the COPA architecture requires extensions for better scalability support. As mentioned in the previous section, the platform is limited by memory bandwidth. A hierarchical COPA infrastructure may solve these issues by providing multiple memory channels and several interconnection busses. This is especially required to prevent the memory bandwidth from becoming the system bottleneck. However, this adds more complexity to the mathematical model and scheduling becomes more complicated. To solve this issue, a multiport memory access unit could be introduced to abstract from several independent memory channels, providing a load-balancing type of functionality for memory access at the frame level.

Currently, each image operation has no adjustable quality control. Adding quality-control means reduces the amount of involved operations. This is especially beneficial to video content analysis algorithms, as they have a very dynamic behaviour in computation requirements. However, the proposed model should be further extended for such quality control and deterministic nature of the control needs to be reconsidered.

ACKNOWLEDGMENTS

This research has been supported by Point-One under the project *Optimization of Modular Embedded Computer-Vision Architectures* (OMECA).

REFERENCES

[1] A. K. a. C. S. Jai Gopal Pandey, "An embedded architecture for implementation of a video acquisition module of a smart camera," Paper, IEEE, Central Electronics Engineering Research Institute, Pilani, Rajasthan, India.

[2] J. U. C. a. J. W. J. Seung Hun Jin, "Pipelined Virtual Camera Configuration for Real-Time Image Processing based on FGPA," Paper, Proceedings of the 2007 IEEE, Sungkyunkwam University, Korea, 2008.

[3] A. A. B. S. a. A. D. Richard Kleihorst, "Camera Mote with a high-performance parallel processor for real-time frame-based video processing," Paper, NXP Research and Philips Research, Eindhoven, The Netherlands, 2007.

[4] W.-M. C. a. L.-G. Chen, "Pyramid Architecture for 3840X2160 Quad Full High Definition 30 Frames/s Video Acquisition," Paper, IEEE, Department of Electrical Engineering, National Taiwan University, 2010.

[5] A. J. Moonen, "Predictable Embedded Multiprocessor Architecture for Streaming Applications," PhD Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2009.

[6] A. C. J. Kienhuis, "Design Space Exploration of Stream-based Dataflow Architectures : Methods and Tools," PhD Thesis, Delft University of Technology, Delft, 1999.

[7] R. v. d. Voort and M. Spierings, "Embedded platform selection based on the roofline model," Master Report, Eindhoven University of Technology, Eindhoven, The Netherlands, 2011.

[8] T. G. a. Y. Sorel, "From algorithm and archtecture specification to automatic generation," in *MEMOCODE'03: Proceedings of the First ACM and IEEE international Conference on Formal Methods*, Paper, Washington, 2003.

[9] R. Albers, "Modelling and control of image processing for interventional X-ray," PhD Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2010.

[10] A. Moonen, M. Bekooij, R. v. d. Berg and J. v. Meerbergen, "Analysing the impact of a communication assist in a multiprocessor system-on-chip," Paper, University of Technology, Eindhoven, The Netherlands.

[11] Y. Bondarau, "Design-time Performance Analysis of Component-Based Real-time systems," PhD Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2009.

[12] M. Gries, "Methods for Evaluating and Covering the Design Space during Early Design Development," Technical Memorandum, University of California, Berkeley, 2003.

[13] G. Palermo, C. Silvano and V. Zaccaria, "Multi-objective design space exploration of embedded systems," Journal of Embedded Computing, Politecnico di Milano, Italy, Advanced Computing Group STMicroelectronics, Switzerland, 2005.

[14] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, P. Henia, R. Racu, R. Ernst and M. Harbour, "Influence of Different System Abstractions on the Performance Analysis of Distributed Real-Time Systems," Paper, EMSOFT '07 Proceedings, Computer Engineering and Network laboratory Zurich, Switzerland, Institute of Compuer and Communication Network Engineering, TU Braunschweig Germany, 2007.

[15] S. Schliecker, J. Rox, M. Ivers and R. Ernst, "Providing Accurate Event Models for the Analsysis of Heterogeneous Multiprocessor Systems," Paper, Institute of Computer and Communication Network Engineering, Technical University of Braunschweig, 2008.

[16] N. Trčka, M. Hendricks, t. Basten, M. Geilen and L. Somers, "Integrated Model-Driven Design-Space Exploration for Embedded Systems," Paper, Electrical Engineering, Department of Mathematics, Computer Science Department, Eindhoven University of Technology, Embedded Systems Institute, Océ Technologies, The Netherlands, 2011.