An Embedded Smart Camera on a Scalable Heterogeneous Multi-DSP System

Michael Bramberger, Bernhard Rinner
Institute for Technical Informatics
Graz University of Technology, AUSTRIA
{bramberger, rinner}@iti.tugraz.at

Helmut Schwabach
Video & Safety Technology
ARC seibersdorf research, AUSTRIA
helmut.schwabach@arcs.ac.at

Abstract

A smart camera combines video sensing, high-level video processing and communication within a single embedded device. These devices a core components of novel surveillance systems.

This paper reports on a heterogeneous smart camera targeting traffic surveillance. It is comprised of a CMOS image-sensor, several digital-signal processors and a network processor. We present its scalable and reconfigurable two-fold software framework, which enables the dynamic allocation of algorithmic tasks ("applications") to the DSPs. This framework provides middleware-like service like abstraction of data sources and communication channels. We introduce a resource manager, which manages and keeps track of the DSP's on-chip resources like memory utilization, direct memory-access (DMA) channels and interrupts. Using the resource manager, applications can be distributed to DSPs with the lowest system load.

Experiments with our prototype show the system's usability using an MPEG-4 encoder and a sophisticated video-analysis algorithm.

Keywords: *smart camera; traffic surveillance; embedded system; digital signal processor;*

1. INTRODUCTION

A *smart camera* combines video sensing, high-level video processing and communication within a single embedded device. Such cameras will be key components in next generation video application including entertainment, ambient intelligence or surveillance [6].

In this paper we present several extensions and improvements to our smart camera targeted for traffic surveillance [2]. The major improvements include (i) a scalable multi-processor architecture consisting of an Intel XScale network processor and several TMS320C64x digital signal processors (DSP), (ii) an expanded communication unit enabling TCP/IP as well as GPRS communication, and (iii) a modular and flexible software architecture. These extensions to our initial smart camera have been motivated by (i) the quest for more on-board functionality (several video analysis tasks), (ii) the need for dynamic reconfiguration during operation of the smart camera, and (iii) improved and alternative communication paths.

Sophisticated video analysis in smart cameras have been introduced by Wolf et al. [6] and Foresti et al. [3]. An evaluation of static distribution of tasks to two processors on instruction-level has been presented by Lv et al. [4]. In [5] Soldatini et al. introduced a network management system within a hierarchical surveillance system, which controls the distribution of tasks between smart cameras and workstations, based on processing load and communication parameters.

The remainder of this paper is organized as follows. Section 2 briefly sketches the main requirements for our smart camera and Section 3 presents the hardware architecture. Section 4 presents the overall software architecture – both for the DSP and the network processor. Section 5 presents some performance data and Section 6 concludes this paper with a short discussion.

2. REQUIREMENTS

Traffic surveillance applications are demanding for both hardware and software. In novel surveillance systems much functionality and, thus, processing is migrated from central workstations to individual (smart) cameras. Hence, the requirements for smart cameras are dramatically increasing compared to standard analog cameras.

The functional requirements of the smart camera include the transmission of a MPEG-4 compressed, high-quality (full PAL-D1 resolution @ 25 fps) live video stream of the monitored scene using an IP-based network interface. Additionally, the smart camera has to perform real-time video analysis tasks such as accident detection, detection of wrong-way drivers, vehicle tracking and the determination of traffic parameters – just to mention a few. Due to the dynamic nature of the monitored scene it is not possible to determine which analysis tasks have to be run at which camera a priori. Therefore, it is desired to map, start and stop analysis tasks ("applications") on cameras as required. As a side effect, using this dynamic task allocation the system's fault tol-

erance is increased, since the system can be reconfigured due to the breakdown of system components.

Software The algorithmic tasks have to be designed as high performance, dynamically reconfigurable software modules, which allow video analysis to be performed in real-time. Therefore, these applications require a software framework, which supports the reconfiguration and allocation of tasks within our smart camera. Additionally, this software framework has to provide an abstraction of communication channels and hardware to allow applications to be run on platforms differently equipped.

Hardware Computing performance is a major hardware requirement due to the high demand for on-board processing on our smart camera.[1]

However, power-awareness, reconfigurability and scalability are also important design issues. The smart camera may be installed at locations with limited power (solar panels or power over Ethernet). Reconfigurability and scalability should be supported by providing mechanisms to simply add and remove processing components of the camera. As a consequence, the communication between system components has to be flexible and scalable, too. This scalable hardware architecture should be as transparent as possible for the camera's software architecture.

3. HARDWARE ARCHITECTURE

To fulfill the previously mentioned requirements, the smart camera has been designed as a low-power embedded system. Figure 1 depicts the hardware architecture of our smart camera. It consists of three basic building blocks: (1) the sensing unit, (2) the processing unit, and (3) the communication unit.

Sensing The heart of the sensing unit is the high-dynamic, monochrome CMOS image-sensor LM9618 from National Semiconductor. To cope with the high-dynamics of sunlight vs. night, optics equipped with a controllable iris are used. These two subunits are connected via a generic interface to the processing unit.

Processing DSPs have been chosen as processing elements because they provide high computing performance while keeping power dissipation low. The TMS320C6415 DSP from Texas Instruments serves as processing element and provides 4.800 MIPS at 600 MHz. Our basic camera system is equipped with two of these DSPs providing almost 10 GIPS. The number of DSPs in the camera is scalable and is basically limited by the communication unit (up to 4 DSPs can be connected without additional hardware effort). Each DSP features 1 MB of internal memory which is vital for image processing algorithms as well as a set of on-chip peripherals such as a multi-channel DMA controller or buffered serial ports. Each DSP is equipped with 128 MB of local memory separated into two banks, each connected to one of the two DSP's memory interfaces. This separation increases the achievable data throughput essentially. All DSPs are connected via the on-board PCI bus. A PCI interface is provided by each DSP, so no additional hardware is required.

Communication The communication unit provides access for the processing unit to the outer world (and vice-versa). In principle, the communication of the smart camera is twofold: (1) The communication unit manages the internal communication, where it is used as the host node of the PCI bus and therefore controls the PCI-communication of the DSPs. (2) The external communication channels are usually IP-based like Ethernet, wireless LAN, or GPRS. Although IP-stacks are available for DSPs, the implementation of communication channels on the used *Intel XScale IXP422* processor provides higher performance and data throughput at lower processing loads than DSP implementations.

4. SOFTWARE ARCHITECTURE

According to the hardware design, our smart camera features two main software architectures. First, the *DSP Framework* is running on each DSP and provides the environment for the video tasks (compression and analysis) as well as the hardware abstraction and advanced resource management to support reconfiguration and scalability. Second, the *SmartCam Framework* is executed on the network processor and acts as a bridge between the DSPs and provides access to the outer world. Additionally, it collects performance status information of the DSP Frameworks to enable a load dependent allocation of tasks to the DSPs.

4.1. DSP Framework

Figure 2 shows the basic structure of the DSP-based software architecture of the smart camera. As mentioned in Section 2, the architecture is divided into the core layer which provides abstraction and services to the application layer.

Dynamic Loading Operating systems for DSPs are designed to be fast, reliable, and lightweight. Consequently, applications to be run on the DSP are statically linked to the operating system, resulting in a single binary. To switch to another application, not contained in the currently used binary, the system has to be stopped, the new binary has to be downloaded, and finally the new system is started up. Since it is inherently important for reconfigurable and scalable systems to be able to load and unload software modules (applications, drivers) during runtime, we have chosen to implement the dynamic loader from Texas Instruments into the DSP-Framework.

The dynamic loader is an integral part of the core-drivers layer and enables in conjunction with the PCI messaging driver, the download and execution of software modules. Therefore the module's binary code is downloaded to the DSP. Then the dynamic loader is invoked, which loads the

1 Note that MPEG-4 compression (full PAL-D1 @ 25 fps) requires almost 5 GIPS. High-level video analysis tasks for traffi c surveillance require a computing performance in the same order of magnitude.

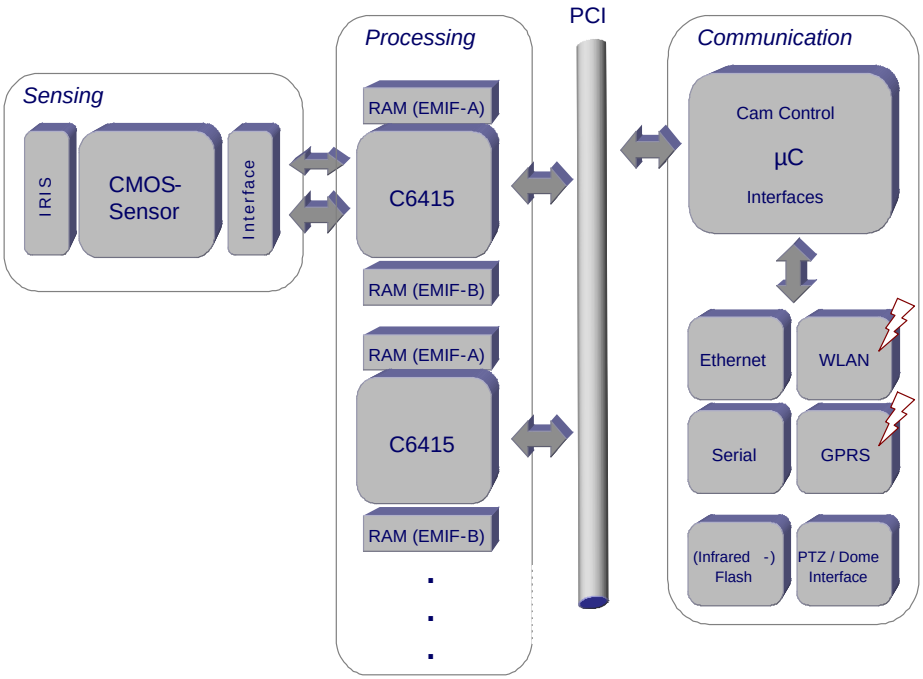

Figure 1. The hardware architecture of the smart camera

binary to its destination memory (on/off-chip memory), resolves the symbols of the module and the running modules, and finally launches the module. Since there is no restriction of which modules can be dynamic loaded or not, our system uses the functionality for core-layer components like dynamic, second-class drivers (video-, audio-, communication drivers) as well as for applications.

Therefore the operating system (DSP/BIOS), and the basic functionality including PCI messaging drivers and dynamic loading support (Core Drivers / Functionality) is linked statically and represents the base system. Any other components are loaded dynamically and are exchangeable during runtime.

Resource Management Low-level management of on-chip resources, like DMA channels, DMA interrupts and hardware interrupts is implemented by the chip-support library (CSL) as part of DSP/BIOS. However, changing resource requirements due to added and removed applications require an enhanced resource management including reservation, monitoring and selection of allocatable resources.

Video-based real-time computing requires a high amount of data to be transferred, therefore the utilization of the DMA subsystem is high. Thus one of the tasks of the resource manager is the efficient allocation of DMA channels and interrupts.

For an efficient allocation of tasks on the DSPs, the resource manager additionally monitors the utilization of the processor core and the system memory. Previous work [1] shows that memory transfers are crucial for the performance of DSP-based systems. Therefore the resource manager also keeps track of the current utilization of the memory bus since an overload would result in reduced performance and as a consequence in violations of the real-time deadlines. The following resources are managed and monitored by the resource manager. Note that the monitoring results are transferred to the SmartCam framework (see section 4.2), which collects and utilizes the gathered results.

- CPU load,
- memory utilization (considering memory hierarchy),
- memory bus utilization,
- DMA channels,
- DMA completion codes,
- interrupts,
- timers.

Data Services Various DSP subsystems may be equipped with different off-chip resources, therefore applications must may not access hardware resources directly. Even the access to the resource's driver introduces a problem, since nonexistent resources will not require a driver not be present. Therefore application communicate with the service manager run on every DSP system, which provides access to the resources. To receive data from a resource (eg. an image sensor), the application subscribes to services provided by the service manager, which subsequently supplies the application with the required data. Therfore data endpoint drivers like video sensors or audio codecs inform the local service manager during startup of the data it is able to provide or accept, respectively. As soon as an application subscribes to a service (e.g. raw image data at a certain resolution and frame rate), the service manager checks if this service is available locally. In case of success, it forwards

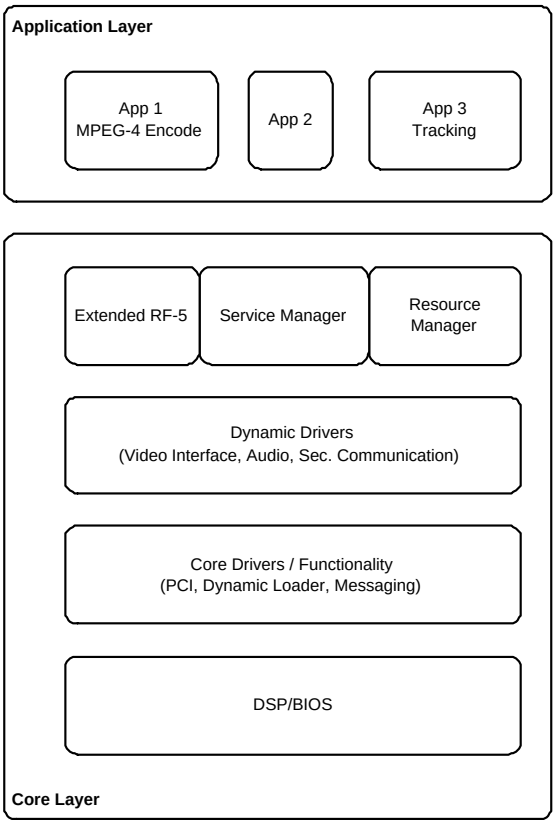

Figure 2. The DSP Framework

the request to the driver, which subsequently initiates a direct communication with the application. To enable other DSPs access to the local services, the service manager announces its service to the SmartCam-Framework (see section 4.2). Consequently, requests to services which are not locally available, cause the service manager to send a query to the SmartCam-Framework, if the service is available on the smart camera. Supposing the service is available, the SmartCam-Framework forwards the request to the service manager providing the service, which then initiates a direct connection to the requesting service manager (via PCI).

As the application unsubscribes the service, the local service manager, and, if applicable the remote service manager terminates the service for that application.

4.2. SmartCam - Framework

The system control and the interface to the outer world is handled by the SmartCam-Framework, which is run on the network processor. To guarantee short reaction times as well as comprehensive functional support this framework is divided into two layers (cp. figure 3): (i) The kernel-mode layer and (ii) the user-mode layer.

Kernel-Mode Layer The kernel layer is the base layer of the Linux DSP integration. Since this layer is part of the operating system kernel, it is inevitable that the kernel layer is fast and reliable. Therefore the functionality of the kernel layer

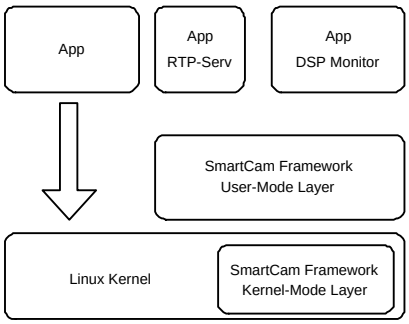

Figure 3. The SmartCam Framework

includes basic and time-critical functionality. Since various DSPs can be used (TMS320C64x or TMS320DM64x), the main purpose of this layer is the abstraction of the DSPs to provide a common interface to the upper layers. Additionally, the kernel layer provides functionality for access control of the DSPs to guarantee mutually exclusive access either from the network processor and the DSPs.

User-Mode Layer Based on the kernel-mode layer, the user-mode layer provides a DSP access library (DSPlib), which extends the hardware abstraction and communication functionality of the kernel-mode layer. This includes a publisher/subscriber messaging system, that enables applications to register for certain messages, which are received and dispatched by the DSP access library. Additionally, the management of dynamically loaded applications which includes the monitoring and control (start/stop) of the tasks is part of this layer. The integrated performance monitoring unit is the base for selective allocation of applications to the DSPs. It collects the current system state of the system's DSPs and provides the data to applications. This way an application can be launched on the DSP with the lowest system load.

Based on the DSP access library, the SmartCam framework includes a set of user-level applications like (1) a bridging service, which listens for service announcements of service managers and subsequently relays incoming requests to the appropriate service managers.

Beside the DSP services, the SmartCam framework also provides services for the Linux platform, like monitoring of the system status, and the network connection. This way the smart camera can establish a GPRS connection in case of breakdown of the Ethernet interface. Consequently all tasks would have to be notified in such case to reduce the network traffic due to the reduced bandwidth of the GPRS connection.

5. EXPERIMENTAL RESULTS

5.1. Prototype

The proposed software framework has been implemented and evaluated on the prototype of our smart camera (see figure 4). This prototype consists of an *Intel IXDP425 Development board*, which is equipped with an Intel IXP425 XS-

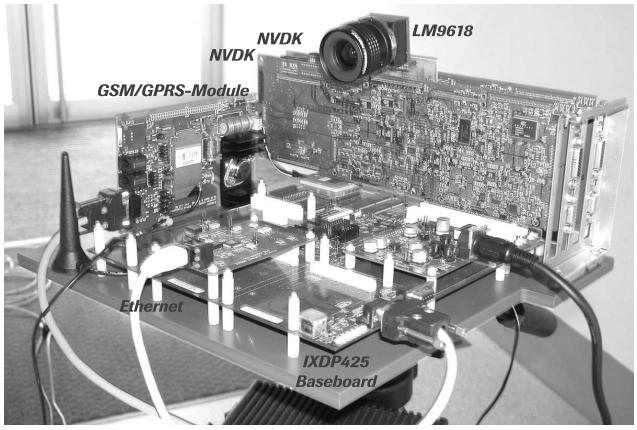

Figure 4. The prototype of the smart camera

cale processor running at 533 MHz. This processor features on-chip Ethernet MACs, serial communication ports and a PCI host controller. The board is operated with Linux Kernel 2.6.8.1, which allows the usage of standard software packages, and enables interoperability with PC-based Linux systems. The system is currently able to be equipped with four DSP boards, however our prototype is equipped with two *Network Video Development Kits (NVDK)* from ATEME. Each board consists of a TMS320C6416 DSP from Texas Instruments, running at 600 MHz, with a total of 264 MB of on-board memory. Image acquisition is done using the *National Semiconductor LM9618* monochrome CMOS image sensor, which is connected to one of the DSP boards.

5.2. Results

We have tested our setup using two algorithms, which are loaded dynamically to the DSPs of the prototype:

1. The MPEG-4 encoder has been configured to encode VGA-resolution frames at 15 frames per second. The resulting encoded bitstream is transferred to the network processor, which transmits the video-stream using the real-time protocol (RTP). In this configuration the encoder demands approximately 50% CPU load.
2. The stationary vehicle detection [1] requires input frames with 320x240 pixels resolution at 7 frames per second. The CPU load for this configuration is approximately 40%.

The MPEG-4 encoder has been run on the DSP featuring the image sensor, since it requires the higher amount of input data (VGA @ 15 fps). The mentioned performance data implies that both algorithms can be run on one DSP. However, the conversion of the input frames to suit the system and the shutter control for the image sensor raise the base load of the system to approximately 40%. Therefore it is not possible to run both applications on one single DSP. After evaluating the current system status (CPU load, memory utilization, etc.), reported by the SmartCam framework, the stationary vehicle detection has been launched on the second DSP. The CPU load on the network processor has been approx. 13%, whereas the major part of the processing load is required by the RTP streaming task,

Memory Footprint Table 1 enlists the memory requirements of the DSP- and the SmartCam framework and the footprints of the frameworks including the operating system.

	SmartCam Framework	DSP Framework
Framework	400 kB	70 kB
System	16 MB	210 kB

Table 1. Footprints of the frameworks

Data Throughput Raw video has the highest bandwidth demands of up to 32 MB per second (full PAL resolution (752x576) at 25 fps). Therefore the communication channel between the processors has to be able to cope with these bandwidth requirements. We have analyzed the data throughput between either the DSPs, and a DSP and the network processor. Therefore we have transmitted data chunks with 64 kB size each, while measuring the time required to transfer the data. The values denoted in table 2 represent the extrapolated values. They show that the achievable transfer rate between the DSPs is sufficient to transport more than two raw video streams in parallel, while the connection to the network processor is slower due to hardware implementation issues of the Intel XScale processor.

	DSP ↔ DSP	DSP ↔ XScale
Throughput	75 MB/sec	15 MB/sec

Table 2. Achievable transfer rates via PCI bus

Dynamic Loader We have also determined the time it requires to start the stationary vehicle detection as a dynamic module (cp. table 3). The major part of the required time is used to download the module (46 ms) to the DSP, while the start of the module took only 17 ms.

Size	Download	Registration & Start
190 kb	46 ms	17 ms

Table 3. Dynamic loading

6. CONCLUSION

In this paper we have presented a smart camera targeting traffic surveillance. The high functional and environmental requirements lead to a heterogeneous embedded system consisting of digital-signal processors to accomplish the video

surveillance tasks, and a network processor for system control and communication management. The migration from static mapping of algorithms to DSPs to dynamic allocation of tasks introduces a significantly increased flexibility and fault-tolerance. However, to support the dynamic allocation of tasks, a software framework has to be implemented on the network processor and the DSPs which abstracts the hardware and the communication channels. To distribute data between the DSPs, the framework features subscription based data-services which are transparent to the applications. To demonstrate the functionality of the system, we have implemented two applications, which make use of the proposed software framework. Experiments with this system demonstrate the functionality and the moderate requirements of the system.

Future work includes (i) the further evaluation of the system using more DSPs and a larger set of applications, (ii) the automated allocation of tasks to DSPs using the application's requirements and the current system state, (iii) the automated distribution of tasks in groups of smart cameras using mobile agents, and (iv) the improvement of the communication between DSPs to reduce the required processing load. We are also interested in the implementation of the conversion of the input frames (from image sensor) to the system's required format in programmable hardware.

References

[1] M. Bramberger, J. Brunner, B. Rinner, and H. Schwabach. Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 174–181, 2004.

[2] M. Bramberger, R. P. Pflugfelder, A. Maier, B. Rinner, B. Strobl, and H. Schwabach. A smart camera for traffic surveillance. In *Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems*, pages 153–164, June 2003.

[3] G. L. Foresti, P. Mähönen, and C. S. Regazzoni, editors. *Multimedia video-based surveillance systems*. Kluwer Academic Publishers, Boston, 2000.

[4] T. Lv, B. Ozer, and W. Wolf. Parallel Architecture for Video Processing in a Smart Camera System. In *Proceedings of the IEEE Workshop on Signal Processing Systems*, pages 9–14, 2002.

[5] F. Soldatini, P. Mähönen, M. Saaranen, and C. Regazzoni. *Network Management Within an Architecture for Distributed Hierarchial Digital Surveillance Systems*, chapter 4.4. Kluwer Academic Publishers, Boston, 2000.

[6] W. Wolf, B. Ozer, and T. Lv. Smart Cameras as Embedded Systems. *IEEE Computer*, 35(9):48–53, September 2002.