GigE Vision Data Acquisition for Visual Servoing using SG/DMA Proxying

Martin Geier, Florian Pitzl and Samarjit Chakraborty
Chair of Real-Time Computer Systems
Technical University of Munich, Germany
geier/pitzl/chakraborty@rcs.ei.tum.de

ABSTRACT

In many domains such as robotics and industrial automation, a growing number of Control Applications utilize cameras as a sensor. Such Visual Servoing Systems increasingly rely on Gigabit Ethernet (GigE) as a communication backbone and require real-time execution. The implementation on small, low-power embedded platforms suitable for the respective domain is challenging in terms of both computation and communication. Whilst advances in CPU and Field Programmable Gate Array (FPGA) technology enable the implementation of computationally heavier Image Processing Pipelines, the interface between such platforms and an Ethernet-based communication backbone still requires careful design to achieve fast and deterministic Image Acquisition. Although standardized Ethernet-based camera protocols such as GigE Vision unify camera configuration and data transmission, traditional software-based Image Acquisition is insufficient on small, low-power embedded platforms due to tight throughput and latency constraints and the overhead caused by decoding such multi-layered protocols.
In this paper, we propose Scatter-Gather Direct Memory Access (SG/DMA) Proxying as a generic method to seamlessly extend the existing network subsystem of current Systems-on-Chip (SoCs) with hardware-based filtering capabilities. Based thereon, we present a novel mixed-hardcore/softcore GigE Vision Framegrabber capable of directly feeding a subsequent in-stream Image Processing Pipeline with sub-microsecond acquisition latency. By rerouting all incoming Ethernet frames to our GigE Vision Bridge using SG/DMA Proxying, we are able to separate image and non-image data with zero CPU and memory intervention and perform Image Acquisition at full line rate of Gigabit Ethernet (i.e., 125 Mpx/s for grayscale video). Our experimental evaluation shows the benefits of our proposed architecture on a Programmable SoC (pSoC) that combines a fixed-function multi-core SoC with configurable FPGA fabric.

1. INTRODUCTION

In many domains such as robotics and industrial automation, an increasing number of Control Applications utilize cameras to sense the plant state. Such Visual Servoing Systems (VSSs) require real-time execution of both the Control Algorithm and the preceding Image Processing (IP) Pipeline. Whilst traditional VSSs relied on *PC-based Processing Platforms* with high-end CPUs (e.g., Intel Xeon), there is an ever increasing demand for smaller, less expensive and less power-hungry implementations. This demand can be satisfied in a number of ways:
Refining IP Algorithms allows to reduce the computational load by, e.g., trading accuracy, area or depth coverage for performance. To compensate for that, recent approaches fuse inputs from additional sensors (e.g., Inertial Measurement Units) to Vision Features with impressive results [7].
New Processing Platforms such as NVIDIA's Jetson TX1 (featuring a Tegra X1 SoC with 4+4 ARM Cores and an up-to one TFLOPS GPU) combine the unparalleled floating point performance of GPUs with a multi-core CPU system and a small footprint (in terms of both size and power).
Refined Embedded Processing Platforms offer a growing number of increasingly complex CPU cores, often teamed with fixed-function hardware accelerators as, e.g., ARM's NEON SIMD (Single Instruction, Multiple Data) coprocessor. This slowly enables the implementation of computationally heavy algorithms on such small, low-power platforms, although their processing capabilities (most likely) will always lag behind those of traditional PC-based Processing Platforms.
Refined Reconfigurable Processing Platforms such as modern pSoCs combine configurable FPGA fabric with fixed-function hardcores delivering high processing speeds at low power consumption. Such pSoCs are now readily available from multiple vendors, e.g., the Xilinx Zynq-7000 or Altera's Arria series [29, 3]. In contrast to previous devices with hardcore CPUs only (e.g., Xilinx' Virtex-4 FX), current pSoCs are much more CPU-focused and feature an extensive set of hardcores such as interconnects, SRAMs, DDR memory controllers and various I/O peripherals (e.g., GigE, USB, CAN, I2C and UART). Additionally, custom functions can be implemented in the FPGA fabric as softcore blocks.

Whilst these advanced IP Algorithms and Processing Platforms may deliver the required *computational performance*, many industrial VSSs – due to their physically distributed nature – also require fast, deterministic long-range *communication technologies* to link sensing, processing and actuation components together. Over the last two decades, Ethernet- and IPv4-based communication backbones have seen widespread use, both for auxiliary management and actual real-time process data [10]. The main reasons for this are long range per segment, low component costs and (partial to full) compatibility to existing infrastructures. Gigabit Ethernet, in particular, is well suited for high-speed image data and capable of transferring over, e.g., 130 uncompressed RGB frames per second at VGA resolution. Stan-

ESTIMedia'16, October 01-07, 2016, Pittsburgh, PA, USA

DOI: http://dx.doi.org/10.1145/2993452.2993455

dardized protocols for camera control and data transmission such as GigE Vision [1] enable the integration of cameras and IP components from multiple vendors and thus the design of complex VSSs. Although modern smart cameras now allow for on-board execution of increasingly complex, vendor-specific IP algorithms (potentially reducing the data rate), many VSSs rely on traditional streaming cameras (transmitting uncompressed video) due to cost and interoperability reasons. Thus, fast and deterministic interfacing of such *Ethernet-based Communication Backbones* to the aforementioned *Processing Platforms* is crucial for high-speed VSSs to benefit from the fast-paced progress in both domains.

Traditionally, all OSI layers above the MAC (media access control) sublayer are handled in software by device drivers and IPv4 stack. This impacts both throughput and timing determinism due to interference with other software tasks that share CPUs, interconnects or memories. When targeting modern *Refined Reconfigurable Processing Platforms*, however, a hardware-centric and thus less interference-prone approach for GigE Vision Image Acquisition can be identified – Its implementation, though, is a nontrivial task for multiple reasons: With Ethernet and IPv4 being multi-layered, frame/packet-based protocols, Image Acquisition via GigE Vision requires a lot more logic in contrast to traditional, simpler interfaces such as, e.g., CameraLink. In addition, the image data arrives at a line rate of up to 1Gb/s and has to be both forwarded and processed with the smallest possible latency to improve control performance. Whilst the former challenge also exists for transmitting the actuation signals (previously sent via, e.g., CAN), these signals usually are low-speed but still require low latencies. Handling both high-speed image and low-speed actuator data over one single Ethernet interface makes a time- and resource-efficient implementation even more challenging.

In this paper, we evaluate the Xilinx Zynq Programmable SoC (Zynq) as such a *Refined Reconfigurable Processing Platform* for high-speed VSSs and propose *SG/DMA Proxying* as a generic method to extend and accelerate the network subsystem of current SoCs (consisting of GigE controller, device driver and IPv4 stack) with minimal changes. With SG/DMA Proxying, incoming Ethernet frames (received by a standard GigE controller) are rerouted to a custom core (which performs user-defined, hardware-based filtering) and all remaining frames are transparently forwarded to the unchanged IPv4 stack. Based thereon, we present a novel hardware-based, mixed-hardcore/softcore *GigE Vision Framegrabber* that enables both *Image Acquisition* from a streaming (i.e., non-smart) camera *at full line rate* of Gigabit Ethernet and subsequent *in-stream Image Processing*. To this end, we supplement Zynq's hardcore GigE controller with a softcore GigE Vision Bridge to create a GigE Vision Framegrabber capable of separating image and non-image Ethernet frames. Non-image frames are automatically sent to the DDR memory and processed by OS and IPv4 stack as if directly received. GigE Vision image frames, however, are decoded, checked, converted to AXI4 stream format and sent to the subsequent IP modules. We combine back-pressure flow control and fully pipelined, one-clock-per-pixel Image Filtering and Edge Detection to achieve *in-stream* processing at 125 Mpx/s. In contrast to previous solutions based on additional (softcore) Ethernet controllers primarily or even exclusively used for Image Acquisition, our approach seamlessly integrates with both hardware (GigE controller) and software (IPv4 stack) of current network subsystems. It thus is not only resource-efficient but also capable of handling incoming non-image traffic without CPU intervention.

Using our proposed architecture we are able to implement a high-speed, low-power VSS that operates at 178 frames per second (fps) and is capable of stabilizing a Magnetic Levitating Sphere used as a demonstration setup. The resulting system is compared to a reference implementation on a *PC-based Processing Platform* (PC) and a state-of-the-art dual-core *Refined Embedded Processing Platform* (EPP) both using traditional software-based Image Acquisition as opposed to our hardware-based GigE Vision Framegrabber. We show that our approach outperforms the traditional PC and that the selected EPP – despite hand-optimized IP kernels – is unable to satisfy the application requirements due to acquisition and processing delays (caused by CPU, RAM and interconnect congestion). As the proposed solution only relies and depends on common architectural features, it is suitable for other pSoCs (e.g., Altera Arria) and could also be integrated in future SoCs. Furthermore, its modular and generic design opens up many possibilities for future work, such as support for multiple or encrypted camera streams. In summary, the main contributions of this paper are

- *Mixed-hardcore/softcore GigE Vision Data Acquisition* using *SG/DMA Proxying* with *transparent Separation of incoming Ethernet Frames* into image and non-image data at full line rate of Gigabit Ethernet and
- *Design and Implementation* of the entire Visual Servoing System on a *Refined Reconfigurable Processing Platform* (Zynq) with our reusable, hardware-based *GigE Vision Framegrabber* and an *in-stream* mixed hardware/software Image Processing Pipeline.

In addition, we present two traditional software-based implementations of the Visual Servoing System on both

- a *PC-based Processing Platform* (AMD FX-6100 six-core CPU with 8 GB RAM) used as a reference and
- a *Refined Embedded Processing Platform* (two ARM Cortex-A9 CPUs with FPU/NEON coprocessor each).

The rest of this paper is organized as follows. Related work is presented in Sec. 2. Next, we introduce both our Representative Use Case (Sec. 3) and Processing Platforms (PPs) including regular SG/DMA operation (Sec. 4.1.2), and, based thereon, describe mapping and implementation options of the Visual Servoing Application on each PP (Sec. 4). Sec. 5 then presents our proposed system architecture for high-speed VSSs featuring *SG/DMA Proxying* (Sec. 5.3.3) and Sec. 6 shows an experimental evaluation for all three VSS implementations. We finally conclude our work in Sec. 7.

2. RELATED WORK

Current pSoCs (e.g., the Xilinx Zynq series) are increasingly used in various domains ranging from small mobile applications [16] to large distributed setups [18]. Their computational capabilities, versatile interfaces and power-efficient design are promising for small, low-power VSSs, too, as traditional VSSs relied on high-end CPUs and often were limited by supply and cooling constraints. Due to the complex architecture, the interconnect performance of current pSoCs heavily depends on the communication pattern [25]. As the amount of algorithm-intrinsic data of Image Processing Algorithms varies by multiple orders of magnitude [15, 27], the implementation of deterministic IP Pipelines remains challenging. In addition to these *computational* aspects valid for both local and distributed VSSs, the latter also require fast, reliable and long-range *communication* for image data.

Until recently, high-speed VSSs relied on locally connected cameras with dedicated camera interfaces such as CameraLink or LVDS [15, 19]. Such systems operate at and beyond 1000 fps and, depending on lighting conditions, are primarily limited by camera exposure and read out delays [31]. None of these interfaces, however, are suitable for distributed industrial VSSs due to short maximum cable lengths and limited noise immunity. Over the last decade, several interest groups (founded by camera, IC and platform vendors such as Basler, EqcoLogic/Microchip and Matrox) thus devised solutions for long-range camera connectivity including recent high-speed interfaces capable of transferring up to 10 Gbit/s over 100 meters such as CoaXPress and GigE Vision 2.x [14, 1]. Although they solve the issue of limited cable lengths for distributed VSSs, there still is a lack of resource-efficient, fast, deterministic and generic *Image Acquisition* solutions for Ethernet cameras on embedded platforms suitable for small, low-power VSSs. On those platforms, Ethernet is generally managed by software (via device drivers and IPv4 stack) [20, 22]. This imposes severe restrictions on achievable maximum data rate and minimum acquisition latency due to delays caused by process scheduling, interrupts and congestion of CPUs, memories and interconnects. Therefore, more hardware-centric interfacing approaches have been evaluated on both FPGAs and pSoCs, e.g., for network intrusion detection [17, 6] and transfer of image data. Most solutions for image data, however, focus on *Image Transmission* (i.e., converting the video stream from the sensor's local interface to the particular long-range communication standard) for both serial (e.g., CameraLink and CoaXPress) and Ethernet-based interfaces [30].

More recently, *Image Transmission* via GigE Vision is supported by a few commercial softcores available for current FPGA platforms [21, 23]. Although systems using hardware-based *Image Acquisition* have been proposed [12, 11], no data regarding their detailed mode of operation, resource usage or acquisition latency has been published. Those implementations rely on dedicated softcore Ethernet controllers and thus not only are non-optimal in terms of both FPGA resource usage and power consumption, but might also lack support for generic (i.e., non-image) IPv4 traffic. The optional receiver of the latter commercial solution for Image Transmission [24], however, can be used as a reference against which our proposed solution using *SG/DMA Proxying* can be compared qualitatively. It requires a dedicated softcore Ethernet controller, too, and supports non-image IPv4 traffic via software callbacks from its proprietary GigE Vision library that *depends on the CPU* to fetch every non-image frame from a single receive buffer. Compared to our solution, it requires a similar amount of FPGA resources and should deliver equally low Image Acquisition latencies due to its unbuffered stream output for image data. It should, however, be noted that the resource consumption of our architecture is currently dominated by an AXI interconnect sourced from Xilinx. In real-world FPGA designs with additional softcores, this interconnect can be shared with such peripherals which greatly reduces its resource overhead and makes our solution the more efficient one. Furthermore, our Framegrabber relies on Zynq's hardcore GigE controller and thus benefits from the lower resource usage of hardcores in terms of both power and area (compared to a pure softcore implementation). As an additional advantage of *SG/DMA Proxying*, our proposed architecture integrates seamlessly with the network subsystem of current SoCs and thus is capable of handling incoming non-image traffic *without any CPU intervention* (as opposed to one callback per frame).

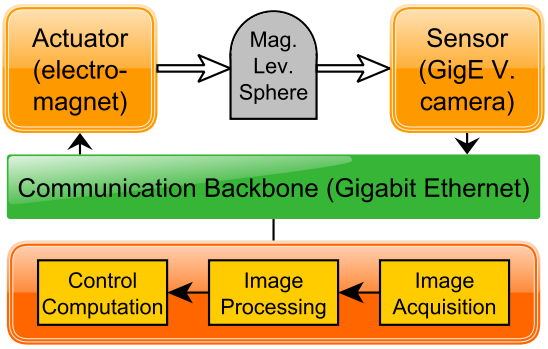

Figure 1: Experimental Setup – Plant and Visual Servoing System with Processing Platform (bottom)

3. REPRESENTATIVE USE CASE

We evaluate the performance of all three PPs and their respective Image Acquisition methods using a real-world Experimental Setup based on a Magnetic Levitating Sphere. Continuous levitation is achieved via Visual Servoing, i.e., optically detecting the sphere's current position followed by computing the required actuation signal within the order of milliseconds. Our resulting VSS (consisting of actuator, sensor and the respective PP) relies on Gigabit Ethernet as a Communication Backbone. With Ethernet now being widely used in industrial automation and many Visual Servoing Applications requiring similar low-latency Image Acquisition and Processing, our Experimental Setup thus serves as a representative use case.

Fig. 1 shows our overall setup consisting of plant and the VSS with its Ethernet-based Communication Backbone. All VSS components are directly connected to a managed switch (ALLNET 8894WMP) in default settings (i.e., RSTP, IGMP and QoS features disabled). We assume that GigE Vision frames are neither lost nor reordered on their way from camera to PP. While this assumption holds true for simple topologies as long each link is below its saturation limit, more complex setups will require techniques such as Quality of Service (QoS) or Stream Reservation via AVB/TSN [13].

3.1 Actuator/Plant: Magn. Levitating Sphere

The demonstrator mainly consists of a steel hemisphere (ø=40mm, h=60mm, w=160g) and an above-mounted dual-coil electromagnet with matching supply and control unit. Whilst one coil is continuously active, the other one is driven with an adjustable current set over Ethernet via UDP/IPv4 (User Datagram Protocol/Internet Protocol version 4). Levitation is achieved by controlling the current and thus the resulting magnetic force F_{mag} such that it compensates the sphere's weight at all times. This yields a non-linear control problem as $F_{mag} \propto d^{-2}$ with sphere-to-coil distance d.

Experiments with a nearly-zero latency sensor (i.e., a light barrier) have shown that stable levitation is possible with sampling rates as low as 125 Hz. With Visual Servoing, however, the increased sensing delay (due to image acquisition and processing) requires compensation via higher rates. This requirement depends not only on plant dynamics alone, but also on both latency and accuracy of each of the three application stages introduced in Sec. 3.3. Although these factors are to some extend orthogonal and thus could be exploited both independently (e.g., by trading IP accuracy for latency) and jointly (e.g., by co-designing IP and controller), we, however, focus on the Image Acquisition stage.

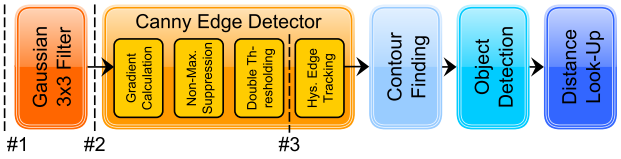

Figure 2: Image Processing Pipeline of the Visual Servoing System and selected Splitting Points

3.2 Sensor: AVT Mako GigE Vision Camera

We use an AVT Mako G-032B camera [2] mounted on a rigid bracket to capture the sphere's position. It has an 1/3 inch grayscale CCD sensor with a native resolution of 658 × 492 pixels and 12 bit depth. Reducing the region-of-interest (ROI) allows for frame rates beyond 102 fps (at maximum ROI) to over 700 fps (at minimum ROI). We use a standard 5 mm wide-angle lens at an aperture of $f/4$. The camera is interfaced via a single Gigabit Ethernet port that also may carry the supply power (using Power over Ethernet). Configuration and image acquisition is performed via GigE Vision and its respective subprotocols, the GigE Vision Control Protocol (GVCP) and the GigE Vision Streaming Protocol (GVSP). Both implement dedicated retransmission mechanisms as they are based on UDP (which itself does not handle packet loss). Using GVCP the camera is configured to an exposure time of 3 ms, a pixel depth of 8 bit and an ROI of 325 × 250 px which yields a frame rate of 178 fps in continuous acquisition mode.

3.3 Visual Servoing Application: Overview

The Visual Servoing Application running on the respective PP (Sec. 4.1) consists of three stages also shown in Fig. 1:

- *Img. Acquisition:* Initial camera configuration; Continuous capture of image data and transfer to IP stage
- *Img. Processing:* Continuous image preprocessing followed by edge, contour and final object detection; Conversion of object position to sphere-to-coil distance d
- *Control Computation:* Continuous computation of coil current based on distance d to maintain levitation

Image Acquisition is responsible for initial camera configuration via GVCP and subsequent continuous capture of GVSP packets. It is supported by multiple closed-source software packages, ranging from versatile tools (e.g., MathWorks MATLAB, Matrox Imaging Library) to vendor-specific libraries (e.g., AVT Vimba; Sec. 4.2.1). Own implementations (in both software and hardware), however, are also possible due to the standardized GigE Vision protocol. This not only enables the implementation of our *GigE Vision Framegrabber* (Sec. 5), but also makes the latter applicable to a large variety of cameras.

The subsequent *Image Processing* stage (shown in Fig. 2) detects the sphere in the acquired image and determines its distance to the coil. As gradient-based IP algorithms like the used edge detector are susceptive to image noise, image smoothing is advisable [9]. We therefore use a 3 × 3 Gaussian kernel as preprocessing filter. It is followed by a Canny Edge Detector with pixel gradient calculation, non-maximum suppression, double thresholding and hysteresis edge tracking units [8]. In the next step, edges are aggregated into contours [26] for the following object detection. The sphere is then identified based on properties of each contour's bounding box (i.e., width and area) and its position is finally converted to the required distance d using a linear mapping (created by offline calibration).

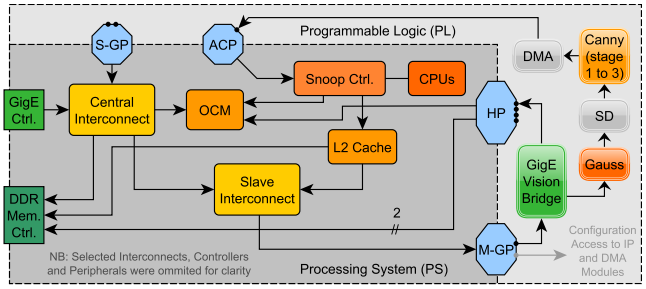

Figure 3: Zynq pSoC – PS I/Os (left), PS (center) and PL with our Hardware Subsystem (right)

Please note that even though the Image Processing stage is application-specific (i.e., tailored to the Magnetic Levitating Sphere), it also is representative of those found in many object-tracking VSSs relying on incremental data reduction.

Control Computation: We use a linearized plant model and proportional–integral–derivative controllers with anti-windup [28] to calculate the required current at each iteration. Optimal controllers such as linear-quadratic regulators can also be used to stabilize the plant, although an integral term has to be added to compensate for actuator drift.

4. ARCHITECTURAL OPTIONS

In this paper, we compare three Processing Platforms (PPs) for their ability to stabilize a Magnetic Levitating Sphere via Visual Servoing. In addition to a *PC-based* PP as traditionally used in VSSs (Sec. 4.1.1), we consider both an *Embedded* and a *(Refined) Reconfigurable* PP (Sec. 4.1.2) to cover all classes of PPs currently available to a VSS designer. The development effort to map the Visual Servoing Application (Sec. 3.3) to the individual PPs ranges from days for PC-based (Sec. 4.2.1) and weeks for Embedded PP (Sec. 4.2.2) to months for our proposed (mixed HW/SW) solution on the Reconfigurable PP (Sec. 4.2.3). The latter, however, is dominated by the *one-time effort* to develop both our GigE Vision Framegrabber and the in-stream IP blocks. Adapting our framework to other Visual Servoing Applications requires *significantly less time* as (at least) our Framegrabber can be reused due to its standardized GigE Vision interface.

4.1 Platforms for Visual Servoing Systems

4.1.1 PC-based Processing Platform

A traditional PC-based Processing Platform (AMD FX-6100, 8 GB RAM, Ubuntu 10.04, Linux 2.6.32) was used as a reference platform. Using a fully fledged OS like Linux has many advantages such as convenient interfacing (via device drivers) and library availability, but also comes at the cost of hard to predict processing latencies due to, e.g., scheduling.

4.1.2 Zynq as Embedded and Reconfigurable PP

Xilinx' Zynq series pSoCs combine a hardcore Processor System (PS) and an FPGA-based Programmable Logic (PL) section. The fixed-function PS features two ARM Cortex-A9 CPUs (with an FPU/NEON coprocessor each) and various peripherals (e.g., memory and I/O controllers) commonly found in current embedded systems. The PL is based on the architecture of current Xilinx FPGAs with two Configurable Logic Blocks per (CLB) Slice, BRAMs, DSPs and additional resources (e.g., clocking, I/O). As the PS is operational both with and without the PL, we use the Zynq pSoC both as CPU-only *Embedded Processing Platform* (PL disabled) and as *Reconfigurable Processing Platform* (PS and PL active).

As shown in Fig. 3, PS and PL are connected via multiple AXI-3 interfaces operating at up to 150 MHz [5]. A total of four 32 bit general purpose (GP) ports are capable of data rates of up to 600 MB/s each. The two GP master (M-GP) ports are the only ones capable of initiating transfers from PS (e.g., from CPUs or GigE controller) to PL. In addition to the two GP slave (S-GP) ports, accesses from PL to PS can be initiated over four high performance (HP) ports and one Accelerator Coherency Port (ACP), all capable of up to 1200 MB/s per port. Whilst both ACP and S-GP ports are linked to the Slave Interconnect and thus able to access every slave device within the PS (e.g., I/O peripherals), the HP ports are only connected to off-chip DDR memory and On-Chip Memory (OCM). The ACP is well suited for low-latency transfers from PL to CPUs due to its connection to the Snoop Controller (and thus to L1 and L2 caches) and a cache-coherent mode relieving the CPUs from cache management. To limit cache trashing, however, the amount of data sent via ACP has to be controlled carefully.

Three components of Zynq's PS not only exist in most other current SoCs, too, but also are of particular interest for our approach: Each *FPU/NEON coprocessor* is capable of SIMD execution of floating point (FP) and integer operations. One instruction affects one or two 64-bit registers, each holding two single-precision FP or eight 8-bit integer values. Due to their parallel dataflow structure many IP algorithms (e.g., convolutions) are well suited for such units. The *DDR Memory Controller* uses a 32 bit data bus to interface up to 1 GByte of external DDR memory at a maximum frequency of 1066 MHz. As the upstream interconnects are unable to handle the resulting theoretical DDR bandwidth of 4264 MByte/s on one single port, the controller's internal arbiter features four independent ports that interface the DDR memory to both PS (CPUs, GigE controller and other bus masters) and PL (via ACP, S-GP and HP ports).
The two *GigE controllers* each implement an Ethernet MAC, an RGMII interface for external PHYs and a Direct Memory Access (DMA) controller capable of standalone scatter-gather (SG) transfers to and from memory. SG/DMA is widely used in current SoCs to increase throughput and reduce CPU load, e.g., during both reception and transmission of Ethernet frame data. Its mode of operation in receive direction requires a particular system configuration (shown in Fig. 4) and is as follows. During *System Initialization*, IPv4 stack and Ethernet driver first allocate a continuous memory range of $n \times 8$ Bytes to be used as Receive Buffer Queue (RxBufQ) with n being the desired number of Receive Buffers as chosen by the system designer. The RxBufQ is a linear array of n Receive Buffer Descriptors (RxBufDs), each containing 32 status bits (S), a 30 bit buffer pointer and one bit each for Last (L) and Empty (E) flags. As a next step, memory for each of the n Receive Buffers (RxBufs) is individually allocated and the address of each RxBuf is stored in the buffer pointer of the respective RxBufD. The RxBufQ then contains n consecutive RxBufDs each pointing to one of the (potentially non-consecutive) n RxBufs. Last, the Receive Buffer Queue Base Address (RXBufQBA) Register of the PS GigE Controller is initialized with the base address of the RxBufQ (i.e., the address of RxBufD 00) and frame reception is enabled. During *System Operation*, the PS GigE controller decodes incoming frames and buffers them in an internal FIFO. Once a frame has been received, the controller fetches the current RxBufD, waits until its Empty flag is asserted, writes the frame to the RxBuf location stored in the RxBufD and clears the RxBufD's Empty flag. Finally, the Receive Buffer Queue Pointer (once initialized with the RXBufQBA) is incremented by eight bytes to set the new current RxBufD. In case the descriptor's Last flag was asserted, the pointer is reset to the RXBufQBA and operation continues with RxBufD 00. On the software side, the Ethernet driver maintains its own pointer to the current RxBufD and performs periodic or interrupt-based checks on the descriptor's Empty flag. Once the RxBufD has been filled via SG/DMA, the driver fetches the data from the respective RxBuf and forwards it to the IPv4 stack for processing. During receive operation the GigE controller thus *scatters* the incoming data to (potentially non-consecutive) RxBufs in memory, whilst the outgoing data is *gathered* from (potentially non-consecutive) TxBufs during transmission.

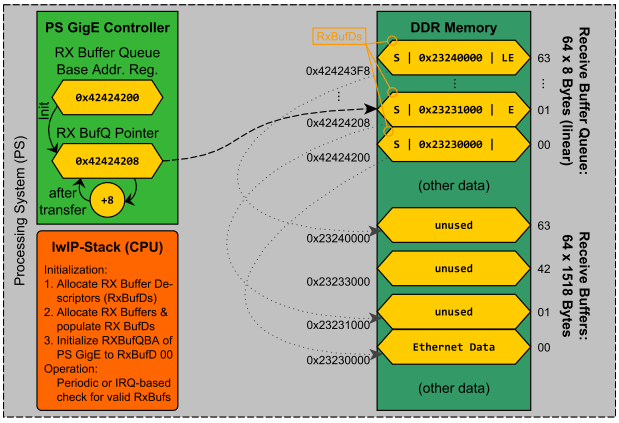

Figure 4: Memory Map and System Configuration for Reception of Ethernet Frames via SG/DMA with n = 64 buffers (GigE Controller → DDR memory)

We choose the ZC702 Evaluation Board from Xilinx with the third-smallest Zynq device (7Z020), 1 GB RAM, a GigE PHY and JTAG-via-USB interface for our evaluations. On this device, CPUs and DDR RAM operate at 666 / 533 MHz.

4.2 Application Mapping & Implementation

Software-only implementations are used on both PC-based PP (Sec. 4.1.1) and CPU-only Embedded PP. On the Reconfigurable PP (Sec. 4.1.2), however, we deploy the proposed mixed HW/SW architecture including our GigE Vision Framegrabber and in-stream Image Processing blocks.

4.2.1 PC-based PP (SW): Linux, Vimba & OpenCV

The implementation on a Standard PC is mostly straightforward due to existing device drivers, IPv4 stack and software libraries. Image Acquisition is performed via AVT's Vimba library whilst the subsequent IP Pipeline (according to Fig. 2) is implemented using OpenCV 2.4.10. It benefits from the CPU's Streaming SIMD Extensions (SSE) for Gaussian and Canny filter processing. Execution times are measured via `clock_gettime(CLOCK_MONOTONIC)` at idle desktop and averaged across 10000 samples (variance < 10ns).

4.2.2 Emb. PP (SW): PetaLinux, Vimba & OpenCV

The implementation on the dual-core Embedded Processing Platform (Zynq PS with PL disabled) is based on Xilinx' PetaLinux OS (Release 14.4 with Linux 3.17) and builds on a cross-compiled version of our Visual Servoing Application. Camera handling and IP are similar to the PC-based implementation (Sec. 4.2.1) with one key difference: As OpenCV 2.4.10 lacks support for the NEON coprocessor, a hand-optimized version of OpenCV 3.0.0-beta was evaluated, too. Using the NEON coprocessor speeds up Gaussian and Canny filtering by processing eight integers in parallel.

4.2.3 *Mixed HW/SW System on Reconfigurable PP: Hardware-based GigE Vision Image Acquisition and In-Stream IP / lwIP & OpenCV (SW)*

As our experimental evaluation (Sec. 6) shows that even a hand-optimized (software-only) implementation on Zynq's dual-core PS is unable to achieve a frame rate of 178 fps (i.e., execution time lower than 5.6 ms), hardware acceleration via Zynq's PL is required. At first glance, there are many variants for such a mixed hardware/software implementation as each processing step (Image Acquisition; Gaussian filtering; Canny, Contour and Object Detection; Control Computation) could be mapped to either hardware or software. Some variants, however, already can be eliminated based on intermediate results from the two software implementations and their respective partial execution times.

Canny Edge Detection alone already takes over 5.6 ms when implemented in software and thus requires (at least partial) hardware acceleration. Its first three processing stages (Fig. 2) are well suited for fast and resource-efficient implementation in FPGAs. The final step (hysteresis edge tracking), however, is more challenging due to its control-flow dominated algorithm. It iterates over a list initialized with all pixels that the preceding thresholding stage identified as real edges ("strong") and checks its neighborhood for pixels considered potential edges ("weak"). Each found weak pixel is then marked as strong and added to the list whilst the original pixel is removed from the list. This iterative process continues until the list is empty. As this requires complex control structures, multi-pass edge tracking is best implemented in software [4]. Reducing the edge tracking to a single pass greatly increases the risk of lost complex edges and is inapplicable due to the sphere's curved geometry.

Using such *mixed HW/SW Canny Processing*, the execution time for edge detection is reduced to 1.6 ms. Combined with the delays of software-based Image Acquisition (>0.9 ms due to IPv4 stack overhead and full frame-buffering), Gaussian filtering, transfer to Canny (>0.65 ms) and Contour Detection, the total execution time of 6.35 ms is still too high.

The additional use of *hardware-based Gaussian Filtering* reduces the execution time to 4.35 ms, which, although sufficient for operation at 178 fps, is over 1.45 ms slower than the implementation on the PC-based PP. Previous experiments on this PP have shown that such additional delay has negative impact on the control performance of the VSS. When taking into account additional latencies caused by network interrupts, IPv4 stack and DDR memory congestion (due to conflicting accesses from GigE controller and PS-to-PL DMA), the need for a more robust, deterministic and future-proof solution with lowest possible latency becomes clear.

We therefore propose the GigE Vision Framegrabber and in-stream IP blocks presented in Sec. 5 to implement a *Hardware Subsystem* capable of both Image Acquisition and Processing at 125 Mpx/s. We evaluate three different options where to split the IP Pipeline in hardware and software as shown in Fig. 2. At first (Splitting Point #1), only *Image Acquisition* via our GigE Vision Framegrabber is assigned to hardware. As a second step (#2), the Hardware Subsystem is extended with *Gaussian filtering* leaving Canny, Contour and Object Detection in software. At last (#3), we also map all non-iterative steps of *Canny Edge Detection* (i.e., all but hysteresis edge tracking) to hardware resulting in the architecture shown in Fig. 3. In all cases, the iterative multi-pass hysteresis edge tracking, Contour Finding and Object Detection are executed in software based on the respective functions from OpenCV.

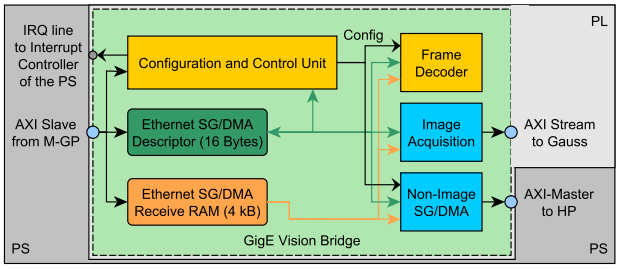

Figure 5: Architecture, Ports and System Integration of the GigE Vision Bridge

5. PROPOSED SYSTEM ARCHITECTURE

As our proposed system architecture relies on SG/DMA Proxying which requires subtle, aligned changes in the network subsystem of current SoCs, we first introduce its overall mode of operation and the resulting benefits (Sec. 5.1). Based on the used software execution environment (Sec. 5.2), we then present our proposed system architecture featuring *hardware-based Image Acquisition* using *SG/DMA Proxying* (Sec. 5.3) and *in-stream Image Processing* (Sec. 5.4).

5.1 VSS Architecture: Overview & Benefits

Fig. 3 shows the proposed system architecture for *hardware-based Image Acquisition* and *in-stream Image Processing* on Zynq pSoCs. Using *SG/DMA Proxying*, we extend its hardcore PS GigE controller with a GigE Vision Bridge implemented in the PL to separate image frames (i.e., those containing GVSP data) from non-image frames. Non-image frames are automatically forwarded to the external memory and processed by the IPv4 stack as if directly received from the GigE controller. Image frames, however, remain in the PL for either in-stream processing by a subsequent IP Pipeline or assembly in a frame buffer (for operations on the entire image such as rectification). Such *hardware-based Image Acquisition* (Sec. 5.3) has the following key advantages:

- *Reduction of cache and DDR memory congestions* as all high-speed image traffic is directly bypassed to PL
- *Increased CPU availability* (e.g., for IP functions) as Image Acquisition is no longer handled in software
- *Reduction of Image Acquisition Latency* for all possible IP Pipeline configurations (HW, HW/SW and SW)
- *Low PL resource usage* and *high power efficiency* due to use of the hardcore GigE controller of the Zynq PS
- *Wide applicability* and *low future design effort* due to GigE Vision compatibility of our acquisition solution
- *Full support for other regular network applications* as non-image data is processed by the IPv4 stack as usual

We consider a hardware-based IP Pipeline capable of *in-stream processing* (Sec. 5.4) in case it consumes and produces at least one pixel per clock cycle and only builds on memories smaller than the image size (i.e., line buffers). Due to their lack of image-sized buffers, such pipelines have low latencies in terms of clock cycles (between the first pixel being consumed and the first pixel being produced) and thus are well suited for latency and resource constrained VSSs.

5.2 Execution Environment

In contrast to the two previous software implementations (based on Linux), we use a bare metal software environment consisting of Xilinx' standalone board support package and

our own hardware abstraction layer. This allows direct access to crucial units such as our PL modules (GigE Vision Bridge, Gauss and Canny filters), CPU MMUs and Snoop Controller (both located in the PS). The latter two require special configuration due to the inclusion of the GigE Vision Bridge in the datapath from GigE controller to DDR RAM. OpenCV 3.0.0-beta is ported to the standalone environment using a replacement library that contains dummy functions for all referenced system library functions (e.g., `open()`). Even without support from an OS, we use both CPUs for optimal performance and a fair comparison to the software-only implementations. CPU0 runs the remaining IP blocks (not implemented in the PL) which determine the current sphere-to-coil distance d using OpenCV. The results are then sent to CPU1 via a simple IPC mechanism using both off-chip DDR memory for storage (of erroneous IP results only) and PS On-Chip Memory for synchronization (via ARM intrinsics) and storage of d. CPU1 then takes care of Control Computation (based on the received distance) and transmission of the actuation signal via Ethernet. To this end, we use the lwIP IPv4 stack supplied by Xilinx with an extended Ethernet driver for our GigE Vision Bridge. As all image-related traffic is captured in hardware, lwIP only handles low-speed, non-image network traffic. CPU1 additionally performs initial camera configuration (via our own GVCP library) and provides a TFTP server (for debugging).
Execution times are measured using the 64 bit Global Timer shared by both CPUs and averaged across 10000 samples.

5.3 Hardware-based Image Acquisition using GigE Vision Bridge & SG/DMA Proxying

5.3.1 *Architecture and System Integration*

Fig. 5 shows the architecture of the GigE Vision Bridge (GEVB) and its interfaces to PS and PL. Transfers initiated by the PS traverse its Slave Interconnect and travel from PS to PL via one of the two M-GP ports (Fig. 3). Whilst M-GP1 is used only for initial configuration of IP and DMA modules, M-GP0 directly connects to the GEVB's AXI Slave interface. Internally, this port is distributed to Configuration and Control Unit (CCU), a 16 Byte Ethernet SG/DMA Descriptor (EthDsc) memory and a 4 kByte Ethernet SG/DMA Receive RAM (EthRxRAM) via a Xilinx AXI interconnect. Whilst the CCU is once configured by software during system initialization, both EthDsc memory and EthRxRAM are accessed by the PS GigE Controller for each incoming Ethernet frame during system operation. Based on the decision of the Frame Decoder, the received frame is processed by either the Image Acquisition module (for image data) or the Non-Image SG/DMA module (for remaining traffic). Image data is sent to an AXI Stream interface connected to the subsequent IP Pipeline in the PL (i.e., Gauss filtering). Non-image data, however, leaves the GEVB via an AXI master port, travels from PL to PS via the HP0 port and is finally stored in the external DDR memory.

The *CCU* contains configuration registers for local IPv4 address and GVSP port number for camera communication via GigE Vision and SG/DMA Queue Base Address (QBA) for forwarding of non-image data. Various flags indicating both regular operation and error conditions are mapped to a status register and can also serve as interrupt sources by setting the respective bit in an interrupt enable register. Interrupts are signaled to software via an interrupt request (IRQ) line routed to the Generic Interrupt Controller.
The *EthDsc memory* contains two 8 Byte Buffer Descriptors identical to those used by the PS GigE controller (Sec. 4.1.2).

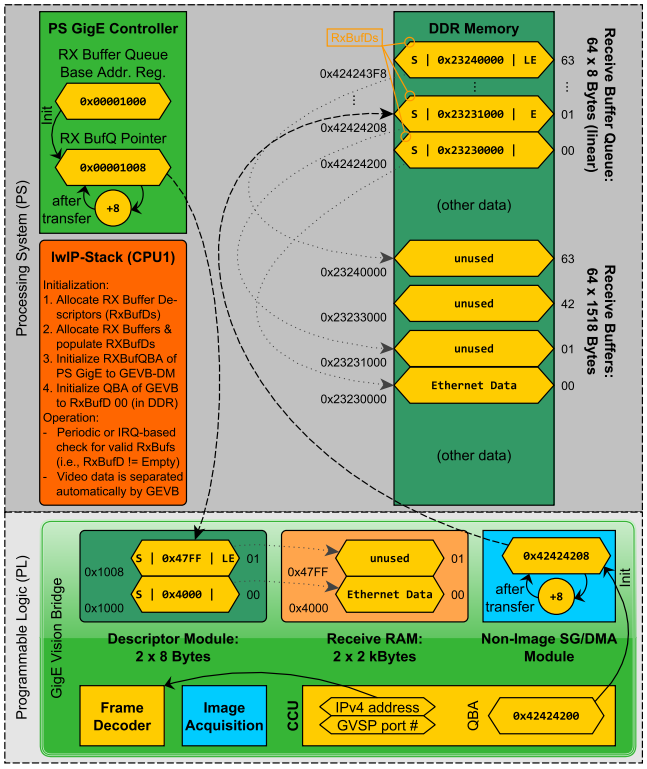

Figure 6: Memory Map and System Configuration for Hardware-based Image Acquisition using SG/DMA Proxying – Incoming Ethernet Frames are redirected to our GigE Vision Bridge which transparently forwards non-image data to DDR memory

The *EthRxRAM* module combines a dual-port BRAM and an AXI BRAM controller to form a byte-addressable memory with a write-only AXI and a read-only parallel port.
All three units (CCU, EthDsc memory and EthRxRAM) are AXI slaves than can be accessed (i.e., read and written) from the PS by CPUs and GigE controller via memory-mapped I/O. The CCU controls the overall operation of the GEVB via a finite-state machine (FSM) triggered each time the EthDsc memory is updated by the PS GigE controller.

The three remaining modules perform the actual frame processing and thus are connected to CCU (for configuration and activation), EthDsc memory and EthRxRAM. As all modules require random read access to the EthRxRAM, a simple request/grant arbiter (not shown in Fig. 5) is used for request serialization. The *Frame Decoder* is the only module with no direct external connection as it operates purely on information stored in EthDsc memory and EthRxRAM. Internally, it uses a FSM to read and analyze the received Ethernet frame and forwards the detection result to the CCU. The *Image Acquisition* module consists of a control FSM and a data mover which extracts and outputs the actual image data to the AXI Stream interface. Although the *Non-Image SG/DMA* module has the same internal layout, its data mover is more complex as the latter drives the GEVB's AXI Master port (connected to the PS) and implements an entire DMA engine capable of standalone scatter transfers to DDR memory. As data is only sent *to memory*, support for gather transfers *from memory to GEVB* is not required.

On the software side, the Ethernet driver supplied by Xilinx is replaced with a (slightly) modified version to facilitate the inclusion of the GEVB in the receive datapath.

5.3.2 GigE Vision Bridge and System Initialization

During system initialization, IPv4 stack and our Ethernet driver not only allocate $n = 64$ Receive Buffers and an associated Receive Buffer Queue in DDR memory, but also perform device configuration of PS GigE controller and our GEVB in the PL. In contrast to regular receive operation as shown in Fig. 4, the RXBufQBA register of the PS GigE controller (Sec. 4.1.2) is now set to the address of the GEVB's EthDsc memory. The PS GigE controller thus no longer uses the RxBufQ in DDR memory, it instead queries the EthDsc memory of the GEVB to determine the storage locations for incoming Ethernet frame data. The address of the RxBufQ in DDR memory, however, is now written to the QBA register of the CCU. The GEVB finally initializes the two Buffer Descriptors in its EthDsc memory with the addresses of two 2 kByte Receive Buffers located in its EthRxRAM, resulting in the configuration of RXBufQBA, QBA and respective Receive Buffers shown in Fig. 6.

With this configuration, the PS GigE controller now transfers all incoming Ethernet frames to the EthRxRAM of the GEVB, which then filters all image data and stores the remaining data in DDR memory for processing by IPv4 stack.

5.3.3 System Operation using SG/DMA Proxying

Once an Ethernet frame has been received, the PS GigE controller fetches the current Buffer Descriptor (BD) stored in the EthDsc memory of the GEVB (step ①) in Fig. 7). Unless the BD is marked empty (i.e., its E flag is set), the controller continues to query the BD until the condition is fulfilled. Once an empty BD is available, the controller transfers the received Ethernet frame from its internal receive FIFO to the buffer address stored in the BD (step ②). As both BDs in the EthDsc memory have been initialized with the addresses of the respective Receive Buffer in the EthRxRAM of the GEVB, the Ethernet data now traverses through the PS (via Central and Slave Interconnects) and travels from PS to PL via the M-GP0 port. Previous experiments have shown that each M-GP port is capable of handling approximately three times the maximum data rate generated by one GigE link and thus will not limit performance. After the transfer has finished, the GigE controller updates the BD and clears its E flag (step ③) as the associated Receive Buffer is now in use. In addition, it moves to the next BD by incrementing its Receive Buffer Queue Pointer (RX BufQ Pointer) by eight bytes.

The write access to the BD (step ③) also triggers frame processing within the GEVB. The CCU activates the Frame Decoder which fetches the frame from the respective buffer in the EthRxRAM for analysis. It identifies the received frame based on frame length, local IPv4 address, protocol types (IPv4 and UDP) and GVSP port number (step ④).

Depending on the result, the frame is processed by either Image Acquisition or Non-Image SG/DMA unit (with only the non-image case being shown in Fig. 7). The former performs additional identification steps to ensure the validity of the received GVSP packet. It keeps track of the GVSP state which is then used to verify packet type (i.e., leader, payload or trailer), payload type and frame ID. Only if all checks are passed, the image data is extracted and sent to the subsequent IP modules via the AXI Stream port.

In case a non-image frame was detected, the CCU activates the Non-Image SG/DMA module to write the frame to external DDR memory (step ⑤ in Fig. 7). To this end, the module queries the current RxBufD in DDR memory until its Empty flag is asserted (step ⑥). Once the RxBufD is marked empty, the module reads the received frame from

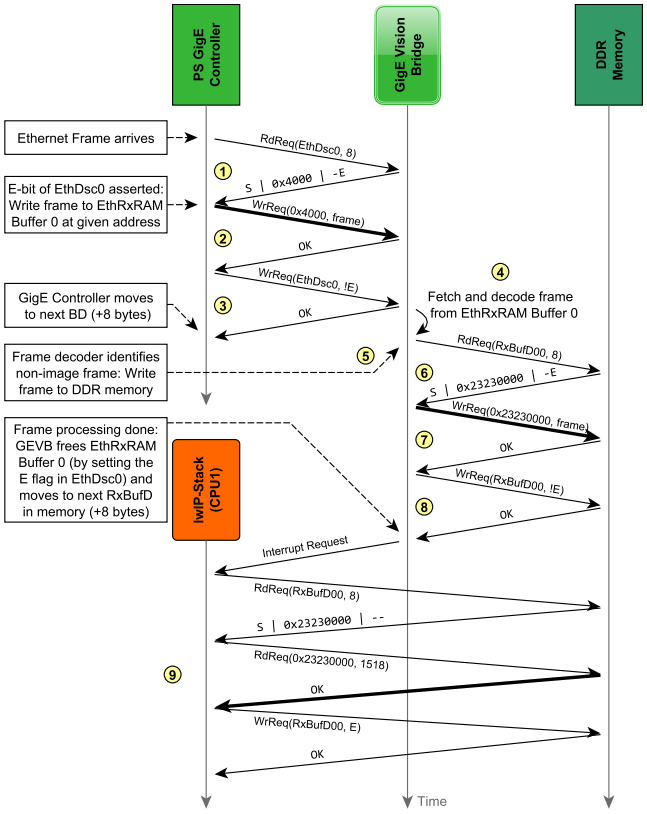

Figure 7: Control and Data Flow during Hardware-based Image Acquisition with Transparent Forwarding of non-image data using SG/DMA Proxying

the respective buffer in the EthRxRAM, writes it to the RxBuf location stored in the RxBufD (step ⑦) and clears the RxBufD's Empty flag (step ⑧). Additionally, the CCU notifies the IPv4 stack via an interrupt request to the PS.

After this frame processing, the CCU frees the respective EthRxRAM buffer by re-setting the Empty flag in EthDsc memory and moves to the next RxBufD in DDR memory. On the software side, IPv4 stack and our Ethernet driver now process the non-image frame as if directly received from the PS GigE controller (step ⑨) as shown in Sec. 4.1.2.

5.4 Hardware-based In-Stream IP Pipeline

After reception by the GEVB, the image data is processed by an in-stream 3×3 *Gaussian filter*. It uses a Gaussian kernel with a standard deviation of 0.8 (which allows for multiplication-free implementation as all coefficients are powers of two) and pixel duplication for border handling. To shorten the critical paths between Gauss and subsequent Canny filters, a Stream Decoupler (SD) module registers both data and bidirectional flow control signals. The first three stages of *Canny Edge Detection* (Sec. 4.2.3) are performed in-stream, too. Gradient calculation is implemented via Sobel filters whilst non-maximum suppression relies on a look-up table to determine gradient directions. The results of the final double thresholding step are finally transferred to the PS by a Xilinx AXI DMA core connected to the ACP.

All units of the hardware-based IP Pipeline process one pixel per clock cycle whilst operating at 125 MHz. This results in a peak throughput of 125 Mpx/s which matches the maximum data rate of the incoming GigE Vision packets.

6. EXPERIMENTAL EVALUATION

Tab. 1 shows measured frame rates and execution times of each processing step in the IP Pipeline (see Sec. 3.3 and Fig. 2) and control computation. Execution time for object detection is always below 30 μs and thus not shown.

The implementation on the *PC-based PP* (Sec. 4.2.1) is capable of achieving the required frame rate of 178 fps. As expected, Image Processing (2.9 ms) dominates over control computation (0.44μs) by several orders of magnitude. Although the PP is capable of a frame rate beyond 340 fps, the VSS is limited by the camera's CCD sensor (Sec. 3.2).

The implementation on the *Embedded PP* (Sec. 4.2.2), however, is unable to reach the required frame rates, both using OpenCV 2.4.10 (configuration E1) and our hand-optimized version of OpenCV 3.0.0-beta (E2). Even though execution times of the IP stages were halved by utilizing the NEON coprocessor (from 21.3 ms to 10.5 ms), they are still approximately two times longer than the required 5.62 ms.

On the *Reconfigurable PP*, we implement our proposed system architecture featuring our GigE Vision Framegrabber and the subsequent in-stream IP Pipeline. Whilst CPUs and DDR memory operate at their respective maximum frequencies of 666 MHz and 533 MHz (Sec. 4.1.2), the PL is clocked at 125 MHz during our experiments. The resulting system outperforms the Embedded PP in all three IP Pipeline configurations (Splitting Points #1-#3 in Fig. 2 and Sec. 4.2.3), although only configuration R3 is able to reach the required frame rate. Configurations R1 and R2 suffer from high execution times of Gauss filtering and Canny Edge Detection similar to the Embedded PP above. Configuration R3 shows the benefit of hardware acceleration for Image Acquisition, Gaussian filtering and all non-iterative steps of Canny Edge Detection. Due to shorter execution time (2.8 ms) and reduced jitter, this configuration even outperforms the reference implementation on the PC-based PP (2.9 ms), which results in improved control performance. The total processing latency of the Hardware Subsystem (measured from reception of the first Ethernet frame to output of the first processed pixel) is less than 5.9 μs and thus negligible compared to all execution times of the IP stages in software. The frame rate of the VSS is again limited to 178 fps due to camera constraints. It should be noted that the Hardware Subsystem itself is capable of processing up to 125 Mpx/s (i.e., full line rate of Gigabit Ethernet and resulting in over 1400 fps at the chosen resolution), whilst the subsequent software-based processing supports up to 355 fps.

The execution times of the control computation on Embedded and Reconfigurable PP differ by one order of magnitude. This is caused by the integration of our GigE Vision Framegrabber which reduces the number of interrupts that need to be serviced. Traditional software-based Image Acquisition (as used on the Embedded PP) yields one interrupt per Ethernet frame. Due to image dimensions and maximum payload size, this results in 58 interrupts per iteration (i.e., image). Using our GigE Vision Framegrabber and SG/DMA Proxying, however, only *one* interrupt is triggered after an entire image has been acquired and processed in hardware.

Tab. 2 shows the usage of PL resources on the Zynq 7Z020 pSoC. As our GigE Vision Frame Grabber relies on the existing hardcore PS GigE controller, only the small GigE Vision Bridge has to be implemented in the FPGA fabric which results in low resource usage of approximately one tenth of the entire device. Thus, our approach is also suitable for the smallest available device (7Z010) which features approximately one third of the resources available on the used 7Z020. Furthermore, the addition of our in-stream Gaussian filtering and Canny Edge Detection modules has only minimal impact on total resource usage.

PP	fps	Gauss	Canny	Contour	Total IP	Control
PC	178	0.32 ms	2.24 ms	0.33 ms	2.9 ms	0.44 μs
E1 (v2.4)	45	5.17 ms	13.7 ms	2.43 ms	21.3 ms	3.22 μs
E2 (v3.0)	90	2.11 ms	7.12 ms	1.28 ms	10.5 ms	1.92 μs
R1 (VB)	110	1.96 ms	5.77 ms	1.16 ms	8.9 ms	0.29 μs
R2 (+G)	142	2.62 μs	5.84 ms	1.16 ms	7.0 ms	0.30 μs
R3 (+C)	178		1.58 ms	1.23 ms	2.8 ms	0.30 μs

Table 1: Frame Rates and Execution Times for all Processing Platforms – PC-based (PC), Embedded (Ex) and Reconfigurable (Rx)

	Flip-Flops	LUTs	BRAMs
Total device resources (100%)	53,200	106,400	140
R1 (GigE Vision Bridge only)	5.73%	11.03%	6.43%
R2 (R1 plus Gauss)	5.83%	11.07%	7.14%
R3 (R2 plus Canny)	6.39%	13.15%	9.29%

Table 2: FPGA Resource Usage on Reconfigurable Processing Platform (Xilinx Zynq 7Z020 pSoC)

7. CONCLUSION

In this paper, we presented SG/DMA Proxying as a novel approach to enable hardware-based Image Acquisition on small, low-power Visual Servoing Systems suitable for both CPU-only SoCs and current pSoCs such as Xilinx' Zynq. By supplementing the hardcore Ethernet controller of Zynq's PS with a softcore GigE Vision Framegrabber, we were able to implement a high-speed VSS capable of both Image Acquisition up to 125 Mpx/s and subsequent mixed hardware/software Image Processing up to 355 fps (limited by software). Both CPU, memory and interconnect congestion and acquisition latency were considerably reduced due to offloading the Image Acquisition from software to hardware. As our Framegrabber relies on the standardized GigE Vision protocol, it is applicable to a large variety of cameras. In addition, it relies on Zynq's Ethernet controller and thus benefits from the lower power consumption of hardcores, although quantitative measurements are part of future work.

In contrast to all previous approaches, our solution seamlessly integrates with both hardware (GigE and memory controller) and software (IPv4 stack) used in current SoCs. It thus supports existing methods to reduce the CPU load during frame reception (e.g., interrupt moderation) which further increases system resilience in case of heavy incoming traffic. In addition, this compatibility greatly reduces the effort required to integrate the proposed architecture in future hardware platforms or other software environments.

As our GigE Vision Framegrabber is both resource-efficient and extensible, it is suitable for many future applications (e.g., integration in SoCs) and improvements such as multi-camera support or the addition of authentication and encryption for secured industrial networks. As a next step, we intend to perform detailed power measurements on all three PPs and replace our current software execution environment with real-time Linux. The latter not only simplifies further development, but also enables more flexible use of the two CPUs due to extensive support for scheduling and inter-process communication.

8. REFERENCES

[1] AIA. GigE Vision Main Page - AIA Vision Standards. http://www.visiononline.org/vision-standards-details.cfm?type=5.

[2] Allied Vision Technologies GmbH. Mako G-032 Gigabit Ethernet camera with Sony ICX424 CCD sensor. https://www.alliedvision.com/en/products/cameras/detail/g-032.html.

[3] Altera Corporation. SoCs - Portfolio. https://www.altera.com/products/soc/portfolio.html.

[4] A. Amaricai, O. Boncalo, M. Iordate, and B. Marinescu. A moving window architecture for a hw/sw codesign based canny edge detection for fpga. In *2012 28th International Conference on Microelectronics (MIEL)*.

[5] ARM Ltd. AMBA Specifications. http://www.arm.com/products/system-ip/amba-specifications.

[6] M. Barbareschi, A. De Benedictis, A. Mazzeo, and A. Vespoli. Mobile traffic analysis exploiting a cloud infrastructure and hardware accelerators. In *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*.

[7] A. Barry and R. Tedrake. Pushbroom stereo for high-speed navigation in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*.

[8] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.

[9] R. Cope and P. Rockett. Efficacy of gaussian smoothing in canny edge detector. *Electronics Letters*, 36(19):1615–1617, Sept 2000.

[10] P. Danielis, J. Skodzik, V. Altmann, E. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht. Survey on real-time communication via ethernet in industrial automation environments. In *2014 19th IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*.

[11] E. Gudis, G. van der Wal, S. Kuthirummal, S. Chai, S. Samarasekera, R. Kumar, and V. Branzoi. Stereo vision embedded system for augmented reality. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

[12] W. He, K. Yuan, H. Xiao, and Z. Xu. A high speed robot vision system with gige vision extension. In *2011 International Conference on Mechatronics and Automation (ICMA)*.

[13] IEEE, Inc. IEEE 802.1 Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/tsn.html.

[14] Japan Industrial Imaging Association. CoaXPress Standard (JIIA CXP-001-2013), Version 1.1. http://jiia.org/wp-content/themes/jiia/pdf/standard_dl/coaxpress/CXP-001-2013.pdf.

[15] J. Khalifat, A. Ebrahim, A. Adetomi, and T. Arslan. A dynamic partial reconfiguration design for camera systems. In *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*.

[16] R. Konomura and K. Hori. Phenox: Zynq 7000 based quadcopter robot. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*.

[17] J. Korenek, P. Korcek, V. Kosar, M. Zadnik, and J. Viktorin. A new embedded platform for rapid development of network applications. In *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '12.

[18] P. Moorthy and N. Kapre. Zedwulf: Power-performance tradeoffs of a 32-node zynq soc cluster. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[19] J. Muller, J. Muller, and R. Tetzlaff. The nerovideo cnn video processing system. In *2014 14th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)*.

[20] D. Oswald, W. Li, L. Niu, J. Zhang, Y. Li, J. Yu, and F. Sun. Implementation of fuzzy color extractor on ni myrio embedded device. In *2014 International Conference on Multisensor Fusion and Information Integration for Intelligent Systems (MFI)*.

[21] Pleora Technologies Inc. iPORT NTx-GigE Intellectual Property. http://www.pleora.com/our-products/embedded-hardware/iport-ntx-gige-intellectual-property.

[22] F. Schwiegelshohn and M. Hubner. Design of an attention detection system on the zynq-7000 soc. In *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*.

[23] Sensor to Image GmbH. GigE Vision IP core XILINX. https://www.s2i.org/webdownload/Download/Datasheets/FPGA%20IP-Cores/GigE%20Core%20Xilinx/GigECore%20Xilinx.pdf.

[24] Sensor to Image GmbH. GigE Vision IP Specification. [Revision X-1.5.3; private communication].

[25] J. Silva, V. Sklyarov, and I. Skliarova. Comparison of on-chip communications in zynq-7000 all programmable systems-on-chip. *IEEE Embedded Systems Letters*, 7(1):31–34, March 2015.

[26] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.

[27] H. Tabkhi, M. Sabbagh, and G. Schirner. Power-efficient real-time solution for adaptive vision algorithms. *Computers Digital Techniques, IET*, 9(1):16–26, 2015.

[28] B. Wittenmark. Integrators, nonlinearities, and anti-reset windup for different control structures. In *1989 American Control Conference*.

[29] Xilinx Inc. All Programmable SoCs and MPSoCs. http://www.xilinx.com/products/silicon-devices/soc.html.

[30] Xilinx Inc. High-Performance Machine Vision Systems Using Xilinx 7 Series Technology. http://www.xilinx.com/support/documentation/white_papers/wp453-machine-vision.pdf. [Revision v1.0].

[31] Z. Ye, Y. He, R. Pieters, B. Mesman, H. Corporaal, and P. Jonker. Bottlenecks and tradeoffs in high frame rate visual servoing: A case study. *The 12th IAPR Conference on Machine Vision Applications (MVA'11)*.