Mapping Time-Critical Safety-Critical Cyber Physical Systems to Hybrid FPGAs

Kizheppatt Vipin, Shanker Shreejith, Suhaib A. Fahmy, Arvind Easwaran
School of Computer Engineering
Nanyang Technological University, Singapore
{vipin2,shreejit1,sfahmy,arvinde}@ntu.edu.sg

***Abstract*—Cyber Physical Systems (CPSs), such as those found in modern vehicles, include a number of important time and safety-critical functions. Traditionally, applications are mapped to several dedicated electronic control units (ECUs), and hence, as new functions are added, compute weight and cost increase considerably. With increasing computational and communication demands, traditional software ECUs fail to offer the required performance to provide determinism and predictability, while multi-core approaches fail to provide sufficient isolation between tasks. Hybrid FPGAs, combining a processor and reconfigurable fabric on a single die, allow for parallel hardware implementation of complex sensor processing tightly coupled with the flexibility of software on a processor. We demonstrate the advantages of such architectures in consolidating distributed processing with predictability, determinism and isolation, enabling ECU consolidation and bandwidth reduction.**

I. Introduction

Cyber-Physical Systems (CPSs) are embedded computing systems in which computation interacts closely with the physical world through sensors and actuators. Examples of such systems are abundant in many domains including automotive systems. For instance, a modern-day car comprises several computing components that perform complex functions such as Anti-Lock Braking (ABS), Adaptive Cruise Control (ACC), Collision Avoidance (CA), satellite-based navigation and infotainment. Some of these components like ABS, ACC and CA are time-critical and/or safety-critical, in that the implemented functionality must not only be correct, but must also meet bounded latency constraints.

Fig. 1 shows a typical ACC and CA system comprising various sensors such as radar, lidar, and cameras. Upon receiving periodic inputs from these sensors, the ECUs process the data to extract relevant information (denoted as the *Observe* phase). Thereafter, based on the observed inputs and mode requirements set by the driver (e.g., following distance and maximum acceleration), the system executes a control algorithm to decide how much acceleration or deceleration is required (denoted as the *Decide* phase). Finally, the system sends commands to the acceleration and braking actuators to take the necessary actions (denoted as the *Act* phase). Since this is time-critical functionality that requires ISO26262 certification [1], the end-to-end latency from sensing to actuation must be bounded and statically verifiable. We refer to such systems as *Real-Time Cyber-Physical Systems* (RT-CPS).

There are a wide variety of new functions for driver comfort and safety in modern vehicles, and these typically rely on new sensors and complex processing. As automation

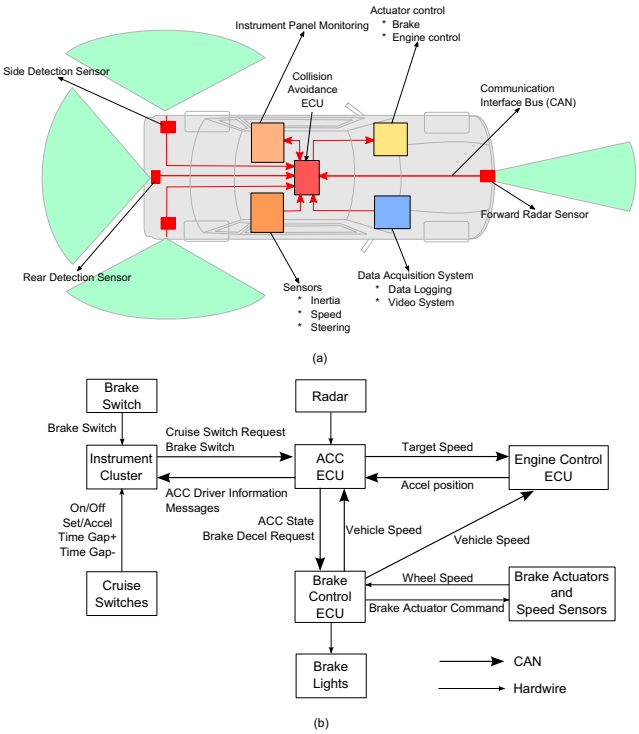

Fig. 1. (a) A collision avoidance system. (b) An Automatic Cruise Control showing sensors and actuators.

increases, the need to fuse together information from multiple sensors and adapt to changing environmental conditions will drive the choice of implementation platforms. Within the automotive space, this trend has already taken hold, with a typical car today housing up to 100 or more ECUs supporting these functions [2]. Connectivity beyond the vehicle is also an important trend; car-to-car communication with the help of roadside beacons and satellites is under consideration [3]. Hence, vehicles can be considered as distributed computing systems that also interact with each other.

When implementing RT-CPSs, most designers use general purpose or domain-specific processors to minimise cost as well as Size, Weight and Power (SWaP). These processors are combined with the necessary communication interfaces to create a general *Electronic Control Unit* (ECU). Typically, a single ECU is dedicated to the execution of one or more modules of a single application. For example, the Observe

978-1-4799-5387-5/14 $31.00 © 2014 IEEE
DOI 10.1109/CPSNA.2014.14

phase of ACC and CA applications is executed on dedicated ECUs physically close to the sensors. This pre-processed information is then sent via a Controller Area Network (CAN) bus to the ECU that executes the Decide phase. Once a decision is made, it is conveyed, again via CAN, to the actuator ECUs that are physically close to the accelerator and brake. Modern vehicles host a plethora of sensors and advanced functions, requiring high processing power and communication bandwidth. A traditional processing paradigm with sequential processor execution cannot provide the required determinism and predictability for these functions, many of which are safety-critical. One approach has been to utilise the processing capabilities of multi-core processors [4], [5]. However, multi-core architectures cannot not ensure the required isolation between the different sensor data (and hence their applications), since they share significant resources between the cores and thus, may not provide the required predictability [6].

Reconfigurable devices, particularly more recent hybrid field programmable gate arrays (FPGAs), provide a promising platform for addressing these challenges [7]. Hybrid FPGAs integrate capable processors with a reconfigurable hardware fabric on the same chip, enabling parallel implementation of complex data processing in hardware, with software supported flexible control. The predictive nature of hardware implementation makes these devices ideal for time-critical and safety-critical applications. The inherent parallelism in hardware enables complexity scaling without compromising overall performance. Sufficient isolation between processing units can also be guaranteed through hardware implementation. Hybrid platforms would also allow hosting of multiple sensor channels in parallel and with complete isolation, thus enabling consolidation of processing at the sensor-end, reducing bandwidth requirements and ECU count.

Our main contributions in this paper are:

- An analysis of present approaches to application implementation on distributed ECUs and their limitations.
- A proposed framework for implementation of such functions on hybrid FPGAs.
- Demonstration of the proposed hybrid FPGA mapping through a case study.

The rest of this paper is organised as follows: Section II discusses related work, Section III discusses present ECU architecture and its limitations, Section IV introduces the proposed hybrid FPGA mapping, Section V presents a case study using the proposed method, and Section VI concludes the paper.

II. Related Work

Time- and safety-critical systems such as Adaptive Cruise Control (ACC) and Collision Avoidance (CA) have been incorporated in vehicles for more than a decade [8], [9]. Early implementations were simple, and software implementation on general purpose processors was sufficient. Modern vehicles, on the other hand, contain so many systems, that 20-100 million lines of code run on 50-100 ECUs to provide all the necessary control and comfort features [10]. With growing complexity, new approaches must be explored to deal with timing criticality and allow function consolidation. This represents a promising opportunity for reconfigurable hardware platforms such as FPGAs.

Due to their high flexibility in hardware implementation and lower development time compared to ASICs, FPGAs have been widely used in several time-critical applications such as aerospace [11], [12], telecommunications [13] and medical devices [14]. Although they have yet to achieve mass-adoption in the automotive industry, they have been widely used in vision-based automotive systems, and use in more fundamental functions has been proposed in the literature. An FPGA accelerated video-based driver assistance system is described in [15]. Similarly, in [16], the authors describe implementation of a radar signal processing system used for collision avoidance in a driver assistance system. In [17], the authors propose the use of FPGAs to achieve hardware acceleration and energy savings in future vehicles. In [18], the authors discuss the application of FPGA partial reconfiguration to redundancy in vehicular networks. For low-volume deployments, FPGAs are much cheaper than custom ASICs primarily due to lower up-front costs, and shorted design cycle. By consolidating multiple functions into the same chip, the cost-effectiveness of FPGAs can be further improved. Furthermore, the ability to alter functionality by reconfiguring the FPGA at run time, enables new context-aware applications.

Existing automotive FPGA implementations have generally been complete hardware systems without software support. However, hardware is more suited for processing and sensor/actuator interfacing, while high-level aspects of an application, such as setting the driving comfort level, are more suited to software. Hence a more sensible approach is for the data intensive portions of an application to be implemented in hardware, providing a high degree of determinism and lower execution time, while the high-level decision making is implemented in software, supporting easy customisation. We argue that hybrid FPGAs represent the ideal platform for such a compute paradigm.

III. State of the Art

In this section we discuss present ECU architecture and the typical data processing model. We also discuss its limitations and the opportunity provided by hybrid FPGAs.

A. ECU Architecture

The architecture of a typical ECU is depicted in Fig. 2 derived from [19]. ECUs follow a modular architecture to support easy isolation and replacement. They are interfaced with other ECUs and control modules through standard automotive communication network interfaces such as CAN, LIN or FlexRay [20]. An ECU may also have hard-wired interfaces to sensors and actuators to observe and act upon the physical world. An on-board processor monitors sensor inputs which are processed in software written by the application designer, based on the requirements of the target application. To support real-time applications, the processor generally runs a real-time operating system (RTOS). Processed data may be used directly in the decision logic, to control actuators, or may be transmitted to other ECUs over the communication network(s) to support collective decision making.

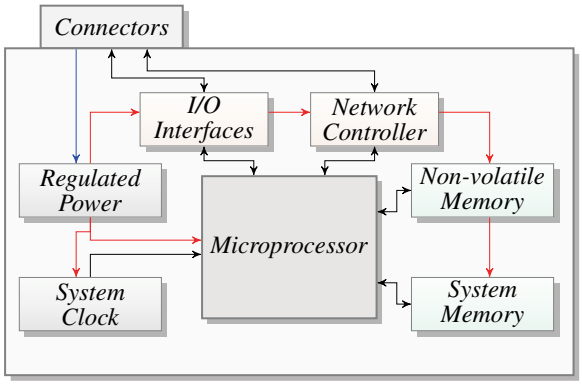

Fig. 2. Typical ECU Hardware Architecture.

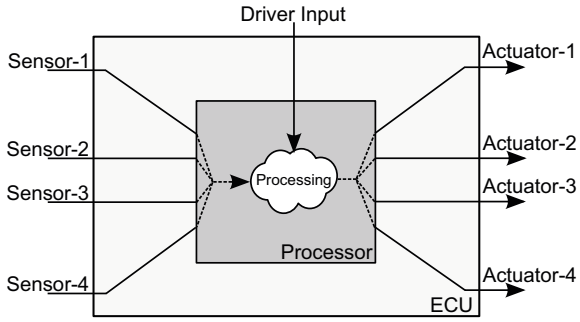

Fig. 3. Traditional processing on a software ECU. Multiple sensor data must be serialised by the processor.

B. ECU Execution Model

Fig. 3 shows an ACC ECU interfaced with 4 sensors and actuators and inputs from the driver instrument panel. The sensors provide information regarding the current vehicle speed, radar signal corresponding to obstructions, brake pedal position and the accelerator position. The actuators can control the brake and engine fuel flow as well as manage vehicle stability. The ECU manages the acceleration/brake actuation based on parameters set by the driver, while also assuring a sufficient safety margin. In software, these functions implemented as independent tasks, which are scheduled either at regular intervals of time, or triggered at specific events (like sensor data becoming available).

The ECU processor gathers sensor measurements through the I/O interface (*observe* phase) under software control. Due to the inherent sequential nature of software execution, this data collection is done in serial although simultaneous measurements are available. This can limit the rate of data acquisition and hence overall system performance. Furthermore, the communication overhead associated with transferring sensor data from the ECU I/O interface to the processor can hamper achievable performance. Moreover, with increasing number of sensors, the latency incurred by the repetitive sequential execution of data collection could impact the timing margin of the safety-critical function.

The processor processes collected sensor data based on predefined control algorithms (*decide* phase). Complex control algorithms can require thousands of instructions to be executed before arriving at a decision. This is especially challenging for safety critical applications, where delayed execution may lead to catastrophic events. Once a decision is made, the processor sends appropriate control signals to the actuators (*act* phase) to apply the necessary changes. This action is again serial in nature and may be affected by the number of actuators interfaced with the ECU.

Present ECU architectures support minimal consolidation of multiple functions due to limited processor performance available in an embedded environment and the lack of isolation supported among these applications. Hence, in modern vehicles, each sensor and actuator may have its own ECU, along with a further ECU for the decision logic. As a result, communication latency becomes an important factor. The overall closed-loop latency is heavily dependent on the speed of processing sensor data, the latency of sending this to the decision logic, and the latency of the decision software. One approach to minimising processing latency is to use application specific integrated circuits (ASICs) to implement complex data processing at the sensor and use a general processor only to make the final decision.

While ASICs may improve sensor data processing capability considerably, a number of factors prevent their adoption. Typically a single ASIC implements a single processing function and it is not possible to customise this after deployment. Furthermore, the cost associated with low-volume ASIC production is considerable. Also, with future evolutions of the vehicle, the processing logic may need to be adapted to work with new sensors and/or deliver improved performance, which can be expensive. Finally, beyond signal processing, there may be a need for interfacing with the network as well as processor interfaces. This increases complexity on the ASIC and at the processor if it is to support multiple sensors. As a result of these limitations, custom ASICs have not gained a foothold in this area.

IV. Hybrid FPGAs as an ECU Platform

Field Programmable Gate Arrays (FPGA) are versatile hardware devices, that enable flexible hardware implementation with functionality modifiable even after system deployment. They are composed of different hardware resources including lookup tables (LUTs), flip-flops, digital signal processing (DSP) blocks, and memory blocks, among others. LUTs facilitate the implementation of combinational logic, while the large number of flip-flops allows designs to be pipelined for high performance. DSP blocks allow fixed-point arithmetic to be fast and use minimal area. FPGAs achieve their unique re-programmability and flexibility due to this composition.

An FPGA designer determines a suitable microarchitecture for their desired application and represents this using a hardware description language like Verilog. Implementation tools process this description to determine how to build it using the basic components on the FPGA, and how to connect these components together. The result is a *bitstream*, which is like a program, but instead describes the spatial arrangement of resources on the FPGA. This bitstream is loaded onto the FPGA at startup to make it implement the desired microarchitecture. Since the configuration memory is volatile, this bitstream can

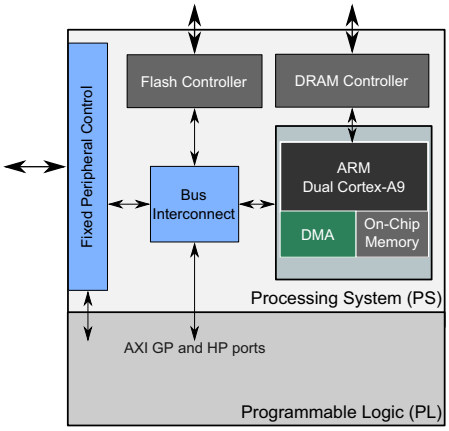

Fig. 4. Zynq Architecture showing the Processor Subsystem (PS) and Programmable Logic (PL).

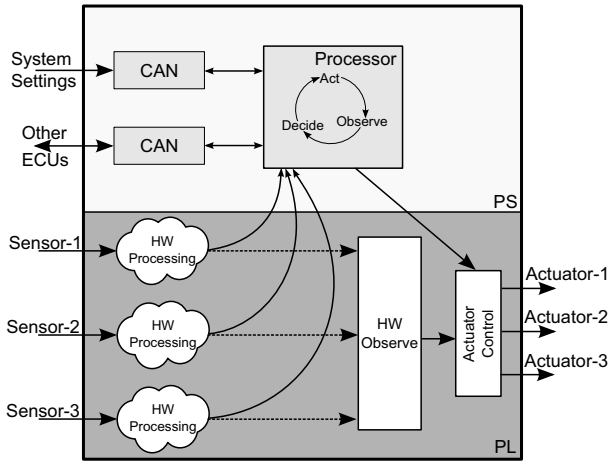

Fig. 5. Proposed hybrid FPGA architecture for ECUs.

also be changed at runtime to allow a different application to be implemented on the same chip.

A. Zynq Hybrid FPGAs

Traditional FPGAs are intended for high speed hardware implementation of digital circuits. Although it is possible to implement simple processors in FPGA logic, these do not generally offer the performance required to implement time-critical algorithms, even more so when layered on top of an operating system. Hence, using a "soft" processor on the FPGA is useful only in cases where the required performance is minimal. Coupling an FPGA with an external processor would create a scenario with challengers similar to the ASIC approach described previously.

The Xilinx Zynq family is a new generation of reconfigurable devices, which tightly couples a dual-core ARM Cortex-A9 processor with a reconfigurable fabric and several built-in peripherals as shown in Fig. 4 [7]. Unlike the embedded and soft processors in previous generation FPGAs, this ARM processor is capable of running a fully-fledged operating system, and has all the capabilities developers are familiar with, without the need for additional resources to be built in hardware. The processor communicates with on-chip memory, peripheral blocks, SDRAM and Flash memory controllers through ARM AMBA AXI-based interconnect. Together, these hardened blocks constitute the Processor System (PS). The PS fixed peripheral control block provides several standard interfaces including 2 CAN controllers. Zynq devices also have two 12-bit, 17 channel analog-to-digital converters (ADCs), which enable direct interfacing of external sensors avoiding the need for external digitisation.

The on-chip PS is attached to the Programmable Logic (PL) through multiple ARM AMBA AXI ports, offering a high bandwidth coupling between the two key components of the Zynq architecture. The PS contains a built-in direct memory access (DMA) controller interfaced to the PL through the AXI interface, which enables high-speed data transfer between the logic implemented in the PL and the external volatile memory. The PL offers an advanced FPGA architecture enabling flexible hardware implementation as previously described.

B. Proposed System Architecture

Our proposed framework for implementing ECUs on hybrid FPGAs is depicted in Fig. 5. The data-intensive sensor processing algorithms are offloaded into hardware modules implemented in the reconfigurable fabric, and the processor is responsible only for monitoring the processed data output and making final decisions. To support hardware modules from multiple vendors, their interfaces to the processing function must follow a standard interface specification such as AXI, while the sensor/actuator interfaces may follow a standard protocol (like SPI, LIN) or be vendor specific. The processor runs the observe-decide-act loop based on the processed data from the hardware modules, considering the inputs provided by the driver. Due to the inherent parallelism in hardware, multiple sensors can be processed simultaneously and the architecture is highly scalable without affecting data acquisition performance. Since only the processed data is transferred to the processor, the communication overhead is significantly reduced. Furthermore, due to the tight coupling between the PL and the PS, high-speed DMA-based data transfer is possible between them.

Another advantage of this architecture is the isolation of the individual processing nodes. Since sensor data processing happens in individual hardware modules with sufficient local memory support, they can be completely isolated. The system behaviour is highly predictable and time-bound, typically to clock-cycle resolution.

Another possible advantage of this architecture is hardware monitoring for safety-critical circumstances. In situations where immediate actions (such as immediate braking) are required based on sensor inputs, a software based approach may incur considerable delay in taking the necessary actions since information has to undergo the decision making steps in the processor. Since the processor considers non-critical drive comfort settings during decision making, this impacts the response time. Conversely, designers may shy away from implementing highly complex and advanced algorithms for fear of adversely affecting worst-case latency. In such scenarios, it should be possible to apply the essential actuations as quickly as possible. In the proposed architecture, such conditions can be directly detected in hardware by monitoring the output

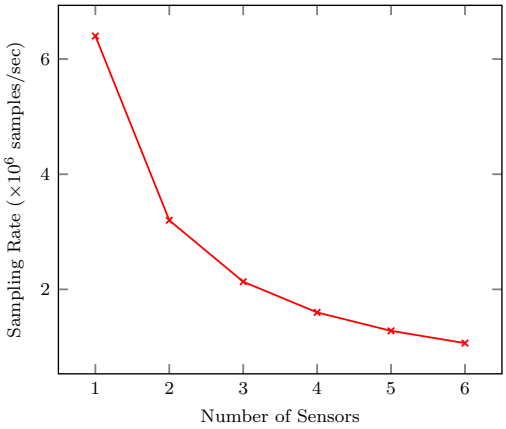

Fig. 6. Variation in sampling rate for software only implementation depending on the number of sensors.

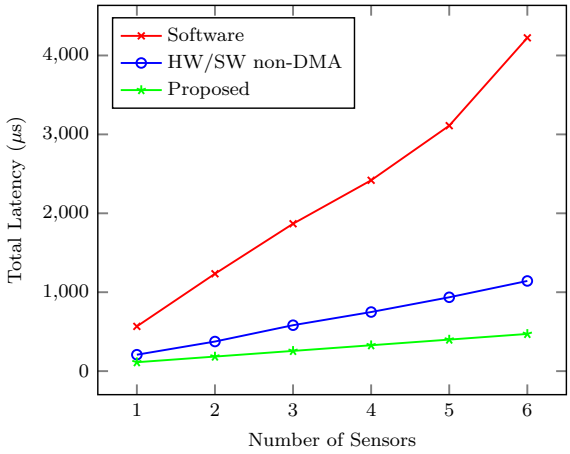

Fig. 7. Variation in total latency depending on the number of sensors.

of the processing modules and immediate actions can be taken by a dedicated hardware module, bypassing processor intervention.

V. Case Study

To demonstrate the effectiveness of the proposed platform, we present a case study which emulates a collision avoidance system. The system detects obstructions using front, rear, and side sensors using frequency modulated continuous radar (FMCR) [16]. The main task of the object detection algorithm is a 1024-point FFT calculation on the received sensor signals followed by distance calculation based on pre-defined equations. The ECU also receives the current vehicle speed and brake position through the CAN interface from the engine control ECU. The system issues warnings and, under dangerous circumstances, directly applies the brake to avoid a collision.

The system is implemented and validated on a Zynq ZC702 evaluation board, which contains a Zynq XC7Z020 device and associated peripherals; the ARM processor runs at 1GHz. To verify the effects of different implementation schemes, the complete system is initially implemented in software with the processor system directly interfaced with the sensor inputs through an AXI interface. The processor operates in polling mode, constantly monitoring the sensor inputs. When a sufficient number of sensor samples are received, the FFT calculation is done in software followed by the distance measurement. Based on the computed distance, the processor controls the actuator signals through another AXI interface.

Fig. 6 shows how adding more sensors (and the required processing) impacts the overall sampling rate. The maximum sampling rate supported by the Zynq ARM processor running at 1GHz and 100 MHz I/O clock frequency is 6.1×10^6 samples/sec. As more sensors are interfaced with the processor, the total sampling rate remains constant, which leads to a reduction in individual sensor channel sampling rate.

In the second scheme, hardware modules process the sensor data (FFT) in the reconfigurable fabric (PL) and the processor controls the actuator signals based on the data received from these modules and some configuration settings emulating driver inputs. The hardware modules are capable of continuously streaming the sensor data and have sufficient internal memory to store the results before the processor reads them out. Here, again, the processor is in polling mode, where instead of checking the sensor inputs it monitors the output from the hardware modules. The distance calculation is still performed by the processor.

In the third scheme, data is transferred from the sensor data processing modules to the PS using DMA and this is indicated to the processor through an interrupt. Here the processor is no longer busy with polling, and so can perform other tasks while the sensor data is being processed in the PL. In this scheme we also implement hardware monitoring in the PL, to explore how this compares to decision making in software. The hardware monitor also implements the distance calculation and can directly activate the actuator control signals if the monitored values cross a pre-defined threshold. In our experiments, a hardware timer is implemented in the Zynq PL for accurate performance measurement. The hardware based implementation is capable of accepting a new sample in every clock cycle resulting in a sampling rate of 100×10^6 samples/sec and is not affected by the number of sensor channels, due to parallel implementation.

Fig. 7 shows the total latency of the collision avoidance system from receiving the sensor data up to activation of the actuator system for each of the schemes. For the pure software implementation, the total latency increases linearly as the number of sensor nodes increases due to the sequential nature of data transfer as well as data processing. When 4 sensors are used, the data transfer latency is about 670 μs and processing time is about 1.6 ms. When the FFT calculation is implemented in hardware, the total latency is considerably reduced as the calculation is much faster in hardware than software. A 1024-point FFT can be calculated in hardware in 11 μs. The increase in total latency for this scheme, as the number of sensors is increased, is not as sharp as in the case of pure software implementation, since the FFT calculation can happen in parallel for the multiple processing nodes, and only the communication is serial.

We then implement the proposed platform with the sensor data processing in the hardware and DMA and interrupt based transfer to the processor when the processed data is ready. This

TABLE I. LATENCY COMPONENTS IN MICROSECONDS FOR DIFFERENT IMPLEMENTATION SCHEMES USING 4 SENSORS.

Implementation	Communication	HW	Software	Total
Software	670	0	1600	2270
HW/SW (non-DMA)	670	11	30	711
HW/SW (DMA)	286	11	30	327
HW/SW (HW monitor)	286	11	1	298

scheme further enhances system performance by reducing total latency where the hardware data processing takes about 11 μs, data transfer to the PS about 286.7 μs, and software-based decision making about 30 μs when interfacing 4 sensors. The overall latency is about $1/7^{th}$ that of the software implementation. Similar to the non-DMA based HW/SW implementation, the hardware processing time remains constant for this scheme due to the parallel implementation of the processing modules. The overall improvement with respect to the previous implementation is due to the lower communication overhead by virtue of the close-coupled PS-PL architecture of the hybrid FPGA. Hybrid FPGAs hence offer an ideal mix of software and hardware performance, and lean communication, to meet the much more stringent latency requirements vital for time- and safety-critical applications.

Finally, we compare the advantage of having hardware based monitoring for emergency scenarios. Implementing the distance calculation and the decision making algorithm based on the control parameters and activating the actuator control takes about 30 μs to complete using the ARM processor. A hardware monitor, which does not consider driver inputs for calculation, can execute the same operation in less than 1 μs. This would allow emergency actuations to be applied much more quickly. The time taken to transfer sensor data to the PS and the hardware monitor are the same since both are attached to the same DMA controller. Performance can be further enhanced by adding a dedicated channel between the data processing modules and the hardware monitor.

Table I consolidates the different latency components for the different implementation schemes in terms of communication latency, hardware execution time and software execution time. The communication latency is identical for the pure software and non-DMA based implementations since the processor needs to read 1024 samples (processed or unprocessed) before calculating the distance. But since the execution time is much lower for the HW/SW schemes, more samples can be processed per second compared to the pure software implementation. The communication overhead is further mitigated using the DMA/interrupt based method.

VI. CONCLUSION AND FUTURE WORK

In this paper we discussed the limitations of present ECU architecture for time- and safety-critical cyber physical systems in the automotive domain. We proposed a new implementation framework for ECUs using hybrid FPGAs that integrate capable embedded processors with a programmable hardware fabric. The parallel processing capability of the hardware allows the number of sensors to scale, while the tight coupling between software and hardware ensures low latencies. Through a case study, we have demonstrated the advantages of such an approach, and also discussed the possibility for direct hardware control of actuators under emergency circumstances.

In the future we intend to demonstrate the capability of the proposed approach on real automotive platforms and in a full network context. We are also keen to explore the layering of multiple functions that use the same sensor information on shared hardware for further ECU consolidation.

REFERENCES

[1] *ISO 26262 - Road Vehicles – Functional Safety*, ISO, 2011.

[2] G. Pitcher, "Cutting Control Complexity," BWI Group, Tech. Rep., 2012. [Online]. Available: www.newelectronics.co.uk

[3] "Car-to-Car Communication Consortium." [Online]. Available: http://www.car-to-car.org/

[4] *Reduced Certification Costs Using Trusted Multi-core Platforms (RECOMP)*, RECOMP Project Consortium. [Online]. Available: http://atcproyectos.ugr.es/recomp/

[5] *SW/HW Extensions for Heterogenous Multicore Platforms (VIRTICAL)*, VIRTICAL Project Consortium. [Online]. Available: http://www.virtical.eu/

[6] A. Abel, F. Benz, J. Doerfert, B. Dörr, S. Hahn, F. Haupenthal, M. Jacobs, A. Moin, J. Reineke, B. Schommer, and R. Wilhelm, "Impact of resource sharing on performance and performance prediction: A survey," in *CONCUR 2013 Concurrency Theory*. Springer Berlin Heidelberg, 2013, vol. 8052, pp. 25–43.

[7] *UG585: Zynq-7000 All Programmable SoC Technical Reference Manual*, Xilinx Inc., Mar. 2013.

[8] W. Prestl, T. Sauer, J. Steinle, and O. Tschernoster, "The BMW active cruise control ACC," in *SAE Paper 2000-01-0344*, 2000.

[9] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Transactions on Intelligent Transportation Systems (ITS)*, vol. 4, no. 3, pp. 143–153, Sept 2003.

[10] R. Charette. (2009, Feb.) This car runs on code. [Online]. Available: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

[11] J. Henaut, D. Dragomirescu, and R. Plana, "FPGA based high date rate radio interfaces for aerospace wireless sensor systems," in *Proc. of Int. Conference on Systems (ICONS)*, March 2009, pp. 173–178.

[12] B. LaMeres and C. Gauer, "Dynamic reconfigurable computing architecture for aerospace applications," in *IEEE Aerospace conference*, March 2009, pp. 1–6.

[13] J. Lotze, S. A. Fahmy, J. Noguera, L. Doyle, and R. Esser, "Development framework for implementing FPGA-based cognitive network nodes," in *Proceedings of the IEEE Global Communications Conference*, 2009.

[14] M. Tahir, A. Bouridane, and F. Kurugollu, "An FPGA based coprocessor for GLCM and haralick texture features and their application in prostate cancer classification," *Analog Integrated Circuits and Signal Processing*, vol. 43, no. 2, pp. 205–215, 2005.

[15] C. Claus, J. Zeppenfeld, F. Muller, and W. Stechele, "Using partial-run-time reconfigurable hardware to accelerate video processing in driver assistance system," in *Processing of Design, Automation Test in Europe Conference Exhibition (DATE)*, April 2007.

[16] J. Saad, A. Baghdadi, and F. Bodereau, "FPGA-based radar signal processing for automotive driver assistance system," in *Proceeding of IEEE/IFIP International Symposium on Rapid System Prototyping (RSP)*, 2009, pp. 196–199.

[17] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, "Reconfigurable computing in next-generation automotive networks," *IEEE Embedded Systems Letters*, vol. 5, pp. 12–15, 2013.

[18] S. Shreejith, K. Vipin, S. A. Fahmy, and M. Lukasiewycz, "An approach for redundancy in FlexRay networks using FPGA partial reconfiguration," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2013, pp. 721–724.

[19] *ECU Designing and Testing using National Instruments Products*, National Instruments, Nov. 2009.

[20] *FlexRay Communications System, Protocol Specification Version 2.1 Revision A*, December 2005. [Online]. Available: http://www.flexray.com