Accelerated Artificial Neural Networks on FPGA for Fault Detection in Automotive Systems

Shanker Shreejith, Bezborah Anshuman
School of Computer Engineering
Nanyang Technological University, Singapore
Email: {shreejit1,anshuman001}@ntu.edu.sg

Suhaib A. Fahmy
School of Engineering
University of Warwick, Coventry, UK
Email: s.fahmy@warwick.ac.uk

***Abstract*—Modern vehicles are complex distributed systems with critical real-time electronic controls that have progressively replaced their mechanical/hydraulic counterparts, for performance and cost benefits. The harsh and varying vehicular environment can induce multiple errors in the computational/communication path, with temporary or permanent effects, thus demanding the use of fault-tolerant schemes. Constraints in location, weight, and cost prevent the use of physical redundancy for critical systems in many cases, such as within an internal combustion engine. Alternatively, algorithmic techniques like artificial neural networks (ANNs) can be used to detect errors and apply corrective measures in computation. Though adaptability of ANNs presents advantages for fault-detection and fault-tolerance measures for critical sensors, implementation on automotive grade processors may not serve required hard deadlines and accuracy simultaneously. In this work, we present an ANN-based fault-tolerance system based on hybrid FPGAs and evaluate it using a diesel engine case study. We show that the hybrid platform outperforms an optimised software implementation on an automotive grade ARM Cortex M4 processor in terms of latency and power consumption, also providing better consolidation.**

I. Introduction

Vehicles today contain highly complex distributed computing systems, especially in luxury cars. Many critical and non-critical functions are implemented in software on a network of varied hardware components. A high-level function is typically distributed over multiple electronic control units (ECUs) interconnected by shared bus networks, allowing information from multiple sensors to be used to decide on actions performed on several actuators. This distributed approach and the harsh automotive environment present multiple ways for errors to be introduced into the system, ranging from temporary disturbances because of electromagnetic interference (EMI), to blown sensors, broken communication channels, or erroneous computational units. As critical mechanical functions are progressively being replaced by electronic counterparts, tolerating faults in computational paths and/or sensors/actuators has become increasingly important, making fault diagnosis and fault-tolerance mandatory in many safety-critical ECUs and functions. As the number of sensors, actuators, and ECUs in vehicles increases, more robust fault detection and tolerance mechanisms are required to maintain correct function of ECU subsystems under different circumstances.

Typical fault-tolerant behaviour is achieved using redundancy in the spatial or temporal domains. Modern networking systems like FlexRay incorporate completely isolated redundant communication pathways [1], while critical systems employ architectural enhancements (redundant cores, sensors, or task migration schemes), as well as multiple task execution cycles (with voting) for tolerating transient or permanent faults [2]. Physical location constraints can preclude spatial (hardware) redundancy from being applied in many cases, such as in the air-flow path of combustion engines, due to both high cost (high temperature/pressure tolerance requirements) and reduced efficiency (reduction in air-flow rate).

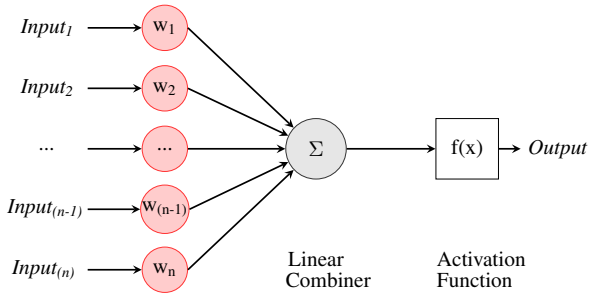

Fig. 1: Generalised architecture of an n-input neuron.

Instead, it is possible to model the relationship between physical variables to perform sensor fault detection computationally. Artificial Neural Networks (ANNs) are a promising approach in this regard, since they can be used to mimic these physical relations (by training). Such adaptability enables ANNs to be employed for a wide range of applications like classification [3], [4], adaptive communication [5] and in-vehicle fault detection [6], [7], among many others. These applications often employ multiple layers of pre-trained neurons referred to as multi-layer perceptrons (MLPs). The fundamental computational element of these networks is the neuron whose general structure, shown in Fig. 1, computes a weighted sum of its inputs. Non-linear functions like sigmoid (a function with an "S" curve) are commonly employed as the activation function, since they help to map non-trivial relationships using fewer nodes. The required relationship between input-output variables is established by training these layers of neurons using standard algorithms.

While ANNs are suitable for automotive fault-detection, implementing them on the general purpose processing platforms commonly used in ECUs is problematic due to the limited computational capacity not offering the throughput required for online fault detection. Furthermore, implementing ANNs on ECUs keeps the processor busy, meaning its other tasks suffer, further deteriorating the system performance. An FPGA-based implementation exploiting the inherent parallelism in ANNs would allow a more meaningful balance of computational performance and power consumption, while also leaving

ECU processors free to work on their existing tasks. Within an automotive context, building a standalone fault-tolerance system would result in a more complex network, higher bandwidth and power requirements, and increased weight.

In this paper, we present an approach based on new hybrid FPGA platforms like the Xilinx Zynq, that closely couple a high performance FPGA fabric with a capable processing system. The combined hardware-software approach allows high hardware performance to be interfaced with software-based processor control. We explore the possibility of such platforms using a case-study on fault-diagnosis of an exhaust gas regulation (EGR) pressure sensor at the intake manifold of a diesel engine, originally modelled in MATLAB. Our experiments show that the hybrid platform offers advantages in performance and scalability over a typical software approach on a capable processing system.

II. Related work

In the literature, artificial neural networks (ANNs) have been explored for classification, pattern detection, machine learning and fault-detection. Deep learning forms of ANN like Convolutional Neural Networks (CNN) and Recurrant Neural Networks (RNN) are widely acknowledged for their compute performance and energy efficiency in classification and machine learning tasks on large datasets (typically in datacenters) [3], [4], [8]. Many ANN applications that interact with physical systems require the accuracy and dynamic range offered by floating point representations, resulting in increased complexity at each neuron. FPGAs represent an ideal platform for accelerating ANN-based systems because they enable large scale parallelism while also supporting high throughput floating point computations [3], [4], [9].

The flexibility of regular feed-forward ANNs allows them to be employed in many distinctive applications and domains. In [5], the authors present an FPGA implementation of the reactive routing scheme for improving the performance of mobile ad-hoc networks using a 2-4-1 MLP ANN. Floating point representation was explored for an adaptive activation function and for the entire compute structure in [10], [11], providing better accuracy for the system. Among others, [7], [12] discuss the use of ANNs for hard-real-time embedded applications like fault-detection, and real-time tracking applications in automotive and defence systems.

The use of ANNs for fault detection in the automotive domain was first proposed in [6]. The authors describe an offline software-based ANN for detecting sensor faults in engines. In [13], the authors present an evaluation of the Instrument Fault-Detection, Isolation, and Accommodation (IFDIA) scheme and present a proof-of-concept scheme on a DSP platform. Though their results show improved sensitivity to faults, software execution presents a bottleneck for larger ANN networks, since parallelism is not exploited.

Within the automotive domain, FPGAs have been proposed to accelerate computationally intensive real time vision-based driver assistance systems [14], [15]. AUTOSAR compliant ECU architectures on FPGAs have also been proposed [16], which can be extended to ECU-on-chip designs that tightly integrate network interfaces and computational cores. Dynamic reconfigurability has also been explored for enabling multi-mode operation for ECU consolidation [17]. Fault-tolerance at architectural level using run-time reconfigurability has also been explored [18], [19]. FPGAs have also become more affordable, reducing a barrier to widespread adoption. Hence, the idea of using FPGAs in vehicles is becoming accepted [20]. To our knowledge, no FPGA-based fault-tolerance schemes using ANNs have been proposed.

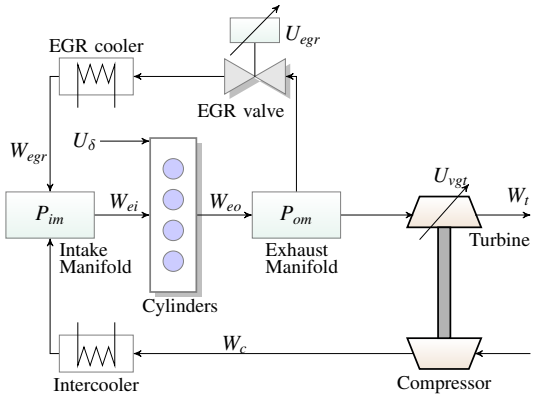

Fig. 2: Air-flow path within a diesel engine.

In our work, we consider an ECU that uses a hybrid FPGA, tightly integrating a capable processor with a reconfigurable fabric, allowing evaluation of large and complex ANNs for fault-detection and accommodation. Beyond the network level optimisations described in the literature for ANN implementations, we optimise the individual neuron structure with a folded sharing approach and explore architectural optimisations like pipelining and scheduling within each neuron and at the network level. The approach is implemented on a Xilinx Zynq platform to evaluate online engine fault detection (derived from [6]), showing that the proposed scheme is able to accelerate the prediction and fault-tolerant behaviour of the system, improving the reliability of the ECUs. Furthermore, we also explore the scalability and reconfigurability of the platform to accommodate failures by altering the ANN system.

III. System Architecture

A. Fault Diagnosis of a Diesel Engine

Fig. 2 is a simplified representation of the air-flow path within a diesel engine [21]. Exhaust gas recirculation (EGR) is a technique commonly employed in modern internal combustion engines, that aims to reduce the nitrogen oxide (NO_2 and NO) emissions (NOx emissions) by recirculating a portion of the exhaust to the engine, via the EGR value and the EGR cooler. This flow is controlled by a set of sensors and valves: the *Pressure Sensor* that measures the pressure at the input manifold (air intake) and the *EGR Valve* that regulates the amount of recirculated air. A failure in either of these components can result in poor engine performance in the short term and partial or complete engine failure in the long term.

To avoid such expensive failures, the Engine ECU commonly compares the measured pressure value, P_{im}, with the expected pressure value, $P_{im(E)}$, that is computed using its non-linear relation to other measured sensor values, as described in Eq. 1. Here, U_{egr} is the position of the EGR valve, N_{eng} represents the rotational speed of the engine (rpm), and W_{ei} represents the flow rate of the air, which are all measured by sensors. The computation involves physical quantities with differently bound values (that can also vary between engines) requiring floating point representation, thus

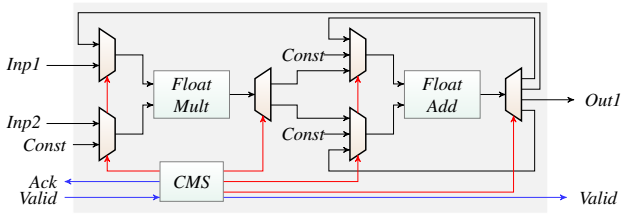

Fig. 3: Core structure of 4/6 input neuron.

making it computationally expensive on many automotive microprocessors that do not feature a dedicated floating point unit. An offline software-based ANN for predicting $P_{im(E)}$ was proposed in [6] to reduce the computational requirements, but considerable latency was incurred by the prediction loop, and this increased super-linearly with the precision demanded by modern engine systems.

$$\frac{\mathrm{d}P_{im}}{\mathrm{d}t} = F(U_{egr}, N_{eng}, W_{ei}) \tag{1}$$

For ANN-based prediction, U_{egr}, N_{eng}, and W_{ei}, form the inputs to the MLP which are evaluated by the network to generate the predicted value $P_{im(E)}$. The number of layers of the MLP and the number of neurons in each layer should be appropriately selected, and trained, to achieve accurate prediction for all possible conditions.

Each neuron in this arrangement requires 4 multiplication (inputs $\times$ weights) and 3 addition (accumulation) operations to generate the intermediate result and a further 1 multiplication and 2 addition operations for the piecewise linear approximated (PLAN) sigmoid activation function. To exploit parallelism, each neuron in a layer must be activated simultaneously, and compute its results in a predictable number of cycles. Hence for a simple 4-neuron layer, a standard RTL description would result in 20 multipliers and 20 adders being inferred by the synthesis tools for integer representation. To maintain required precision, the network should incorporate at least 2 computational layers with an input layer built on 4-input neurons (the fourth input being the current pressure value). Furthermore, the use of floating point representation (to maintain precision as well as adaptability) increases the number of integer multipliers by 4 times, further limiting the number of neurons that can fit on a small device.

B. Optimisation of Generalised Neuron Architecture

Though direct implementation of MLPs using DSP blocks is possible, these are often inefficient as this would not maximise DSP block throughput by way of careful pipeline optimisation. Also, direct mapping of a moderately sized 6_6_1 network using floating point values would result in 79×4 DSP blocks $(((4{\times}6{+}6){+}(6{\times}6{+}6){+}(6{\times}1{+}1)) \times 4)$. The need for floating point adders further limits the number of neurons and reduces the efficiency of direct mapping. The architecture of the neuron must thus be optimised for floating point to allow high performance and scalability. We define a template for neurons that enables reuse of the computationally expensive floating point operators while achieving high operating frequency and constant latency. Multiple templates are necessitated by the non-uniform structure that results from multilayer perceptrons, resulting in a varied number of input/output possibilities at each layer.

The core elements of our neuron architecture are shown in Fig. 3. The structure is composed of a pipelined floating point adder and multiplier unit, derived from the Xilinx floating point IP cores. The floating point adder is built completely out of logic (LUTs and Flip Flops), while the more complex multiplier is built out of 4 DSP blocks and supporting logic. The same multiplier and adder units are reused to compute the PLAN sigmoid activation function by the scheduler, reducing resources further. The scheduling of inputs/operations to these blocks and their monitoring/control are handled by the *Controlling & Monitoring Subsystem (CMS)* module, which exploits the 4-stage pipeline within the floating point units. The CMS is a finite state machine that derives operation latencies from the top-level parameters and produces control signals for the multiplexer and demultiplexer blocks to schedule the execution sequence. The CMS also generates the *ack* and *valid* signals for the neurons in the preceding and successive layers, based on the determined schedule. This structure completes the entire computation from weighted sum to activation in 38 clock cycles.

The operations of the individual stages are shown in the timing diagram in Fig. 4. The multiply operations are scheduled as and when the inputs are received, along with their weight values, with one multiplication in each layer at each timestep. The pipelined multiplier can receive operands every cycle and produces the output after a fixed latency of 4 cycles. These are passed in sequence to the adder for accumulation, which is also a pipelined structure with a latency of 4 cycles. Accumulation is completed in the 21st cycle and is followed by the sigmoid computation, which is scheduled on the same hardware blocks by the CMS.

The structure instantiates a single floating point adder and a single multiplier unit for 4/6 input neuron structures. In the case of larger input combinations (8/12/16 or more), multiple floating point units are used. In the case of 8/12 input neurons, two independent floating point adders and multipliers are instantiated, while for 16 input neurons, four parallel units are instantiated. Here, the *CMS* module adjusts the scheduling of inputs and operations to ensure that the parallel logic is effectively utilised. This allows us to achieve higher performance and almost the same latency in computation, at the expense of slightly increased resource usage. The neuron structure is parameterised so that the *CMS* infers the proper schedule and instantiates the combination of floating point blocks, based on the design time parameter configuration. Constraining the resource requirements at neuron-level allows parallel neurons to co-exist even on smaller FPGAs, leading to higher parallelism.

The network is built up by instantiating different layers that incorporate the parameterised neuron, each layer being self-managed. The handshaking signals integrated into the neuron interface ensure that very little control is required between the different layers for the computation to flow through. Moreover, since the different layers operate in isolation, it is also possible to execute multiple computations in a layer-wide pipeline, with little external logic to manage the data flow in case of non-symmetric layer structures. Since we have optimised the low-level neuron design around the structure of the DSP block, it attains high throughput, and resource sharing ensures the compute units are kept busy.

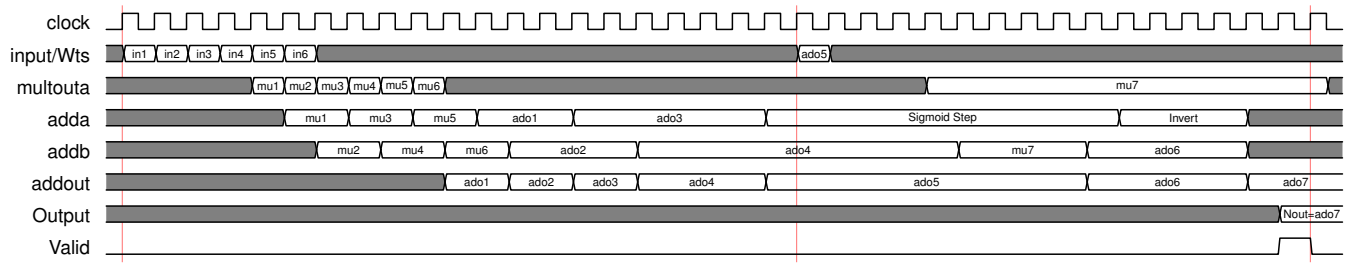

Fig. 4: Timing diagram for complete computation of a 6-input neuron.

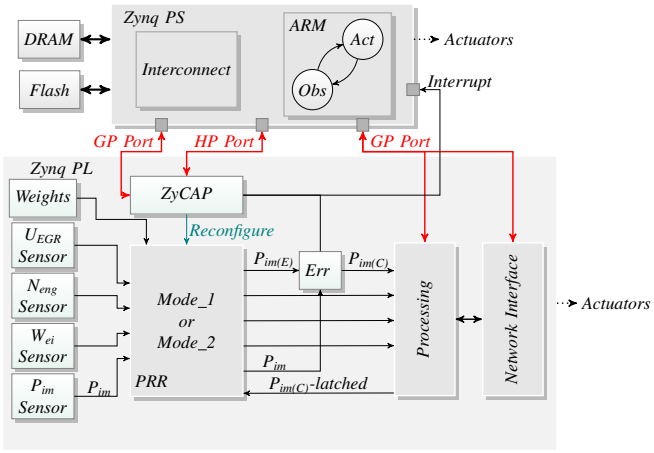

Fig. 5: Proposed hybrid fault-tolerant ECU model on Zynq.

IV. Hybrid ECU Approach

While the ANN could be implemented as standalone fault detection logic on a low-power FPGA like the Spartan-6, this entails overheads in communicating fault information to the engine management system (EMS) and further delay in activation of fault tolerance measures. An integrated approach presents a more interesting solution, where the fault detection scheme forms an integral part of the EMS, improving overall reliability and determinism. New hybrid platforms like the Xilinx Zynq allows such tightly coupled integration between programmable logic (PL) and a highly capable ARM processing system (PS). The customisable fabric offers a highly parallel data acquisition interface with high scalability, and high computational performance even for a large predictive network. Moreover, dynamic reconfigurability under software control can be used to alter the functionality of the ANN from a fault-detection mode to a more precise prediction mode when needed.

A high level view of the proposed hybrid ECU system on Xilinx Zynq is shown in Fig. 5. The external sensors are integrated into the system via multiple parallel interface modules, via protocols such as SPI or I2C. The acquired data is fed to the neural network enclosed in the partially reconfigurable region (PRR), which predicts the possible value of $P_{im(E)}$ based on the other sensor inputs. This prediction is performed by a lightweight network (*mode_1*) that can predict the value with reasonable accuracy. The predicted value is compared to the acquired pressure value P_{im}, and an error is triggered in case of a deviation beyond a predetermined threshold. If there is no error, the acquired sensor values are passed to the software system via the AXI interface (GP port). The software on the ECU executes the control loop (marked as states *Obs* and *Act*), which monitors the sensor values and performs computations as per the algorithm to trigger changes in the system. Alternatively, this computation could also be offloaded into the hardware using the *Processing* block, which can implement the same algorithm in hardware, whose actuation outputs can then be controlled by the software. The hybrid ECU is then wired up into the vehicular network by integrating a custom network controller [22], to form a complete ECU-on-chip system .

If persistent errors are observed, the neural network is switched to a more precise mode (complex network designated *mode_2*) to predict the pressure values with higher accuracy (fault-tolerance mode) based on the other sensors. This is achieved by reconfiguring the PRR alone (partial reconfiguration) to include different weights and a different configuration of the network (more active elements or higher number of layers). Once reconfigured, the network output is used directly by the processing logic for further computations, until the fault is rectified. The software/hardware function can be configured to monitor the P_{im} sensor values and restore normal operation, if the system recovers from the error. In this case, the software reconfigures the logic back into the fault-detection mode, with the lightweight network.

We use the open source ZyCAP configuration management system, integrated as a peripheral to the processing system, with its own software libraries/drivers [23]. It handles low level reconfiguration commands, bitstream memory management and abstracts the details from our application design. ZyCAP also provide faster, non-blocking reconfiguration.

V. Results

First, we evaluated the performance of the possible ANN configurations using the low power Xilinx Spartan-6 series, which are more suited for a standalone vehicular implementation. The ANN models were evaluated using both simulation and implementation in hardware (for the configurations that could fit on a XC6SLX45T device). All configurations of the network were trained offline with data generated from a MATLAB model of the diesel engine [21] using the back propagation (BP) algorithm, for a target precision of **0.01**. The trained weights were loaded into a Block RAM, while the test vectors, also generated from the MATLAB model, were provided as the inputs. We used a set of 151 test vectors that span the operating range of the diesel engine model, representing a range of values for the different inputs.

Fig. 6 plots RMS prediction error of the ANN models (trained with the same termination conditions: 1000 runs or 0.01 precision) compared to the MATLAB model against the effective area required for each implementation. The effective

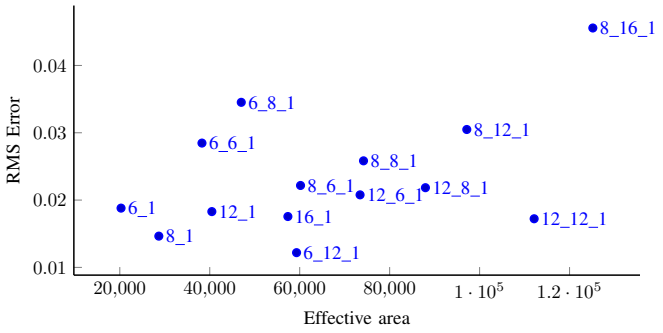

Fig. 6: RMS error of prediction v/s Effective area, for different ANN configurations.

TABLE I: Resource utilisation and prediction latencies of different ANN networks on the largest Spartan-6 device.

Structure L1_(L2_)Out	FFs	LUTs	BRAMs	DSPs	Latency (cycles)
6_1	4037	5939	0	28	68
8_1	5539	8195	0	40	68
12_1	7959	11792	0	56	72
16_1	10962	16412	0	80	72
6_6_1	7956	11644	0	52	105
6_8_1	9650	14239	0	64	105
6_12_1	12454	18342	0	80	109
8_6_1	11401	17175	0	84	105
8_8_1	13868	20939	0	104	105
8_12_1	18225	27519	0	136	109
8_16_1	23164	35062	0	176	109
12_6_1	14780	22221	0	100	109
12_8_1	17632	26514	0	120	109
12_12_1	22757	34303	0	152	113

area is computed as $DSPs \times 512 + LUTs$, which represents the combined area utilisation on the largest Spartan-6 device (XCSLX150). There is a baseline offset in the predicted values, which is due to rounding and approximation schemes in the floating point multiplier/adder units and the PLAN approximated sigmoid implementation.

Further, it can be seen that the single layer 8 neuron network (8_1) and the multi-layer 6-12 network (6_12_1) are Pareto-optimal points offering minimal error and area consumption. Hence, these two implementations were chosen for the fault-tolerant hybrid ECU model that we evaluate later. The hardware requirements of the different models are detailed in Table I, along with the prediction latencies, suggesting that the single layer 8-neuron network can function as a quick and efficient fault-detection system, while the more extensive multilayer network can be used as a sensor replacement, forming a fault-tolerant system.

To highlight the performance advantage of implementing the ANN in hardware (versus software), we executed an optimised software version of the two networks as a bare-metal application on an automotive grade STM32 platform (STM32F407FZ) which uses an ARM Cortex M4 processor. With a clock rate of 168 MHz and with the support of a dedicated floating point unit, the software models executed in 13.47 us and 43.26 us respectively. The same models took

TABLE II: Hybrid ECU resource utilisation on the Zynq 7020.

Mode	FFs	LUTs	BRAMs	DSPs	PR bit size
mode_1	6623	7878	6	14	1506784 bytes
mode_2	15069	20498	6	40	1597303 bytes

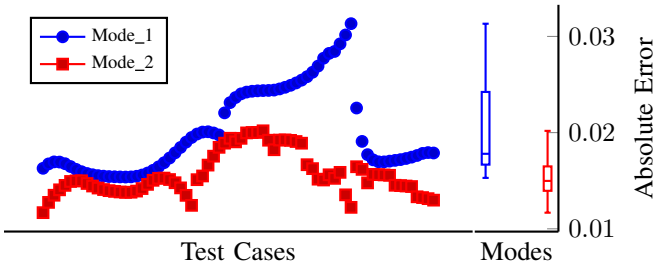

Fig. 7: Prediction accuracy of the two modes in the hybrid ECU.

1.13 us and 1.82 us to compute in hardware, respectively, at 60 MHz clock frequency, providing a 10× and 24× improvement respectively, which can be further enhanced when clocked at higher speeds (120 MHz or higher). For our case study, the latency of both software and hardware models are within acceptable performance limits; however, the scalability of software execution for more complex ANNs is poor, compared to the nearly constant and predictable latency of the proposed hardware. Software execution would also be hindered by other tasks executed on the processor (or interrupts), which may deteriorate the performance further.

For the proposed hybrid ECU, we evaluate the performance of the deterministic mode-switch time by building the two selected ANN architectures on the Zynq ZC7020, on the Xilinx ZC702 development board. The resource utilisation and compressed partial bitstream size of the two ANN modes are shown in Table II. These bitstreams are loaded onto the SD card as the different modes of the adaptive system. The system normally instantiates *mode_1*, which uses the single layer 8-neuron network to keep track of the precision of the P_{im} sensor. When the error deviates beyond a configurable range for consecutive cycles, the system switches to fault-tolerant *mode_2* and instantiates the more precise multilayer network, that acts as a replacement for the P_{im} sensor. The mode switch can also be triggered from software for testing.

We measure the active power while executing both modes on the ZC702 board using the Texas Instruments power measurement adaptor that connects to the PMBus. In the fault-free *mode_1* configuration, the system consumes 85 mW of power on average (110 mW peak), with the computation being triggered every 10 ms, as in the case of a normal engine management system, with a 120 MHz clock. In *mode_2*, with the same trigger rate, the system consumes higher average power at 135 mW, with peak consumption of 160 mW, at 120 MHz clock. The software execution on the STM-32 device consumed 220 mW for evaluating the *mode_1* ANN alone. The complete Zynq SoC hardware consumes 420 mW, largely due to its dual-core processing system that consumed 300 mW (average) while being mostly idle.

Fig. 7 shows the deviation of the predicted value in the two modes, across the range of test vectors, compared to the actual sensor values for each test vector. In both cases, we directly compute the predicted value $P_{im(E)}$ based on other

sensor inputs (U_{EGR}, N_{ENG} and W_{ei}) and the latched $P_{im(E)}$ value, and compare it to the actual sensor value (P_{im} in the current cycle). In *mode_1*, the latched $P_{im(E)}$ corresponds to the sensor value acquired in an earlier cycle modelling the fault-detection case, while in *mode_2*, the latched $P_{im(E)}$ is the predicted output from the previous cycle, replicating a fault-tolerant mode that is independent of the acquired sensor input (P_{im}). It can be observed that *mode_2* results in a tighter prediction, with significantly reduced deviation while the fault detecting *mode_1* results in a larger deviation in the predicted values and also a larger mean error. The plot also shows the ability of *mode_2* to closely predict the sensor value without depending on the corresponding sensor channel for the entire set of inputs. This shows that *mode_2* can effectively replace the faulty sensor by closely estimating the actual pressure value without adding considerable processing latency, at fractionally higher power, while *mode_1* prediction can be used to determine the deviation in actual sensor values with an accuracy of 0.05 ($\times$ scaling factor).

Finally, to trigger the mode switch, we introduced error into the test vectors (flipped sign bit in P_{im} input to the *Err* module) to trigger the adaptation to fault-tolerant mode. Using Xilinx provided PR management, a software switch results in a mode switch time of 145.2 ms to load the fault-tolerant mode, from the detection of error, with an configuration throughput of 10 MB/s. Using the prefetching scheme offered by ZyCAP, we were able to reduce the mode switching time to 3.99 ms.

VI. Conclusion

Artificial Neural Networks provide a suitable mechanism for fault-detection and fault-tolerance in critical domains like automotive systems. However, ANNs are inherently computationally intensive and the precision requirements in harsh automotive environments mean large networks are required, making software implementations impractical. In this paper, we presented a hybrid ECU approach, based on the Xilinx Zynq platform, that integrates an ANN-based prediction system which doubles up as a replacement sensor in the case of persistent faults. The ANN network is completely contained within a partially reconfigurable region (PRR), integrated with parallel sensor acquisition interfaces, a fault detection system, data processing engine, and a network interface. PR allows seamless migration from the fault-detection ANN network (under normal operation) to the fault-tolerant mode with a larger, more complex and accurate network that effectively replaces the faulty sensor, by reusing hardware resources. The proposed parallel architecture enables the ANN to be evaluated in a predictable short latency of under 1 us, even for the larger prediction network. Moreover, the reconfiguration operation is managed seamlessly under software control, with fast reconfiguration and complete changeover in under 4 ms. We evaluated the approach using a case study on a diesel engine model, where the intake manifold pressure sensor is monitored for faults and replaced by the prediction network in case of error.

In future, we plan to extend the scheme to integrate on-line training (using the spare-ARM core), which would enable the network to adapt to any sensor faults in real-time. We would also like to explore extending the ANN-based hybrid ECU model to other automotive hard-real-time applications like battery management in electric vehicles, and more complex x-by-wire systems.

References

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

[2] J. Huang, J. O. Blech, A. Raabe, C. Buckl, and A. Knoll, "Analysis and optimization of fault-tolerant task scheduling on multiprocessor embedded systems," in *Proc. Intl. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, 2011, pp. 247–256.

[3] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating Deep Convolutional Neural Networks Using Specialized Hardware," Microsoft Research, Tech. Rep., Feb. 2015.

[4] S. Li, C. Wu, H. H. Li, B. Li, Y. Wang, and Q. Qiu, "FPGA Acceleration of Recurrent Neural Network based Language Model," in *Proc. Intl. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2015, pp. 111–118.

[5] S. Shah and D. Vishwakarma, "FPGA implementation of ANN for reactive routing protocols in MANET," in *Proc. Intl. Conf. on Communication, Networks and Satellite (ComNetSat)*, July 2012, pp. 11–14.

[6] D. Dong, J. Hopfield, and K. P. Unnikrishnan, "Neural networks for engine fault diagnostics," in *Proc. Workshop on Neural Networks for Signal Processing*, Sep 1997, pp. 636–644.

[7] R. Ahmed, M. El Sayed, S. Gadsden, and S. Habibi, "Fault detection of an engine using a neural network trained by the smooth variable structure filter," in *Proc. Intl. Conf. on Control Applications (CCA)*, Sept 2011, pp. 1190–1196.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proc. Advances in neural information processing systems*, 2012, pp. 1097–1105.

[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proc. Intl. Symp. on Field-Programmable Gate Arrays (FPGA)*, 2015, pp. 161–170.

[10] P. Ferreira, P. Ribeiro, A. Antunes, and F. M. Dias, "A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function," *Neurocomputing*, vol. 71, no. 1, pp. 71–77, 2007.

[11] D. Sonowal and M. Bhuyan, "FPGA implementation of neural network for linearization of thermistor characteristics," in *Proc. Intl. Conf. on Devices, Circuits and Systems (ICDCS)*, March 2012, pp. 422–426.

[12] T.-C. Chu and H. Szu, "An artificial neural network for naval theater ballistic missile defense program," in *Proc. Intl. Conf. on Neural Networks*, vol. 1, Jun 1997, pp. 53–55 vol.1.

[13] D. Capriglione, C. Liguori, C. Pianese, and A. Pietrosanto, "On-line sensor fault detection, isolation, and accommodation in automotive engines," *Transactions on Instrumentation and Measurement*, vol. 52, no. 4, pp. 1182–1189, 2003.

[14] N. Alt, C. Claus, and W. Stechele, "Hardware/software architecture of an algorithm for vision-based real-time vehicle detection in dark environments," in *Proc. Design, Automation and Test in Europe (DATE) Conf.*, 2008.

[15] C. Claus, R. Ahmed, F. Altenried, and W. Stechele, "Towards rapid dynamic partial reconfiguration in video-based driver assistance systems," in *Proc. Intl. Symp. on Applied Reconfigurable Computing (ARC)*, 2010.

[16] F. Fons and M. Fons, "FPGA-based Automotive ECU Design Addresses AUTOSAR and ISO 26262 Standards," *Xcell journal*, vol. Issue 78, p. 20 to 31, 2012.

[17] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, "Reconfigurable computing in next-generation automotive networks," *IEEE Embedded Systems Letters*, vol. 5, no. 1, pp. 12–15, 2013.

[18] N. Chujo, "Fail-safe ECU System Using Dynamic Reconfiguration of FPGA," *R & D Review of Toyota CRDL*, vol. 37, p. 54 to 60, 2002.

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

[20] S. Chakraborty, M. Lukasiewycz, C. Buckl, S. A. Fahmy, N. Chang, S. Park, Y. Kim, P. Leteinturier, and H. Adlkofer, "Embedded systems and software challenges in electric vehicles," in *Proc. Design, Automation and Test in Europe (DATE) Conf.*, 2012, pp. 424–429.

[21] J. Wahlström and L. Eriksson, "Modelling diesel engines with a variable-geometry turbocharger and exhaust gas recirculation by optimization of model parameters for capturing non-linear system dynamics," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 225, no. 7, pp. 960–986, 2011.

[22] S. Shreejith and S. A. Fahmy, "Extensible FlexRay communication controller for FPGA-based automotive systems," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 2, pp. 453–465, 2015.

[23] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq," *IEEE Embedded Systems Letters*, vol. 6, no. 2, pp. 41–44, 2014.