

FPGA Platform for Prototyping and Evaluation of Neural Network Automotive Applications

Nazeeh Aranki¹ and Raoul Tawel²

¹Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109
aranki@brain.jpl.nasa.gov

²Mosaix Technologies
176 Melrose Ave.
Monrovia, CA 91016
rtawel@mosaixtech.com

Abstract

In this paper we present an FPGA based reconfigurable computing platform for prototyping and evaluation of advanced neural network based applications for control and diagnostics in an automotive sub-systems. Recurrent networks with internal or external feedback have proven to be very useful as models or controllers for dynamical systems. Our system implements a recurrent multi-layer perceptron (RMLP), which may be regarded as a combined generalization of a feedforward MLP and a one-layer fully recurrent network. The neuroprocessor design was implemented using a single Xilinx Virtex FPGA and made extensive use of a variety of techniques; including intra-layer parallelism, inter-layer multiplexing, and fixed-point bit-serial based computational techniques to achieve extreme design compactness. Flexibility of the design was achieved by allowing the architecture to be on-the-fly programmable from RAM. The intra-layer architecture of the neuroprocessor was organized in a SIMD configuration. While the overall goal is to develop an inexpensive and commercially viable ASIC based on the FPGA IP core, we describe here our FPGA design implementation and the two challenging applications, misfire detection and engine idle speed control, that had served as the focus for this effort.

1. Introduction

Government regulations and stiff global competition have placed an increasing demand on the performance of vehicle control and diagnostic systems. In the United States, light trucks and passenger cars are required both to meet strict emission standards and to perform continuous diagnostics of all emissions systems operating in the vehicle. These requirements will become more comprehensive and difficult to achieve as emission standards are tightened this decade. Neural networks provide a means of creating control and diagnostic strategies that will permit these challenges to be met efficiently and robustly.

Modern automobile powertrains have complicated control and diagnostics systems that involve several interacting subsystems, almost any one of which provides interesting theoretical and engineering challenges. Responsibility for many important functions has shifted from mechanical control to computer control. This has resulted in a substantial increase in fuel efficiency and, in combination with advances in catalyst technology, to a remarkable reduction in emissions (99% in a properly functioning system). This has been accomplished in the context of an extremely cost competitive industry. Increasingly stringent emissions regulations require that any malfunctioning component or system with the potential to undermine the emissions control system be identified. Hence, cost effective ways of meeting these demands are being explored.

Neural networks have the potential for major impact in this work. Benefits may be anticipated in terms of the time required to design and calibrate a control or diagnostic strategy or an improvement in the observed system performance. In collaboration with Ford Motor Company, we have been investigating the use of neural network hardware for control, virtual sensors, and diagnostics [1][2][3]. While the benefits of neural based strategies is clearly evident, their main drawback is that they are computationally demanding, and for anything but the simplest network architecture - will overwhelm the computational resources available by the on-board engine computer. We anticipate that the broad commercial realization of neural networks will require specialized computational hardware capable of performing a multitude of different applications effortlessly in real-time between engine events.

In the remainder of this paper we begin with a brief description of both the engine idle speed control application and the engine misfire detection problem. This is followed by a section where we discuss aspects of using a recurrent neural networks as a direct fault classifiers. Finally, we describe the design requirements and constraints necessary to execute such networks and present the results of a comparison of software and hardware calculations for vehicle data.

2. Engine Idle Speed Control Problem

The control of an engine operating at idle was the logical first control problem to attempt with neural network methods. Engine idle speed control (ISC) is a challenging yet reasonably accessible problem, since it does not require any large degree of instrumentation or external sensing capabilities [4]. In addition, engine idle speed control can be viewed as rather benign, since it is unlikely that application of a poor control strategy would inadvertently lead to catastrophic failure. However, idle speed control requires the coordinated application of two different controls to regulate the engine speed at a desired level in the presence of torque disturbances which are often unmeasured.

The primary goal of the idle speed control is to maintain and regulate engine speed to a desired level (which may vary), subject to constraints on the controls, while the system experiences unobserved disturbances. Ford required that a trained controller be capable of being integrated seamlessly with other control functions of a vehicle powertrain. It was also desired that transitions to and from idle mode be handled gracefully, so that surges in engine speed do not make vehicle occupants aware of engagement and disengagement of the neural controller.

A challenge of idle speed control was to use information that is available to a vehicle's powertrain control module (PCM) to coordinate effectively the two available controls, *bypass air* and *spark advance*. A number of difficulties affect the computation of these controls. Engine operation at idle is a nonlinear process that is far from its optimal range. The engine experiences torque disturbances due to a variety of sources, and the role of the idle speed controller is to adjust for these disturbances smoothly. These disturbances may be due to electrical loads placed on the engine, such as engagement of the rear window defroster or power windows, as well as mechanical sources such as variable combustion and hydraulic disturbances such as power steering engagement. Some of these disturbances are scheduled and/or flagged by the PCM, thereby providing potentially useful anticipatory and feedforward information to the idle speed control strategy.

The two controls used for idle speed control have different dynamic effects on the engine. The bypass air command, which is a signal between zero and unity that determines the duty cycle for a solenoid valve, regulates the amount of air allowed into the intake manifold of an engine under conditions of closed throttle. The control range of the bypass air signal is large (more than 1000 r/min under idle conditions), but its effect is delayed by a time inversely proportional to the engine speed. The spark advance command, ranging from 10° to 30° for idle conditions in the four-cylinder engine considered here, regulates the timing of ignition. Unlike the bypass air control, spark advance has an immediate effect on engine speed, but over a small range (approximately 100 r/min). We assume that we have access only to those sensor signals that are available to the vehicle's PCM. In particular, engine speed and mass air flow are the two relevant measured engine variables that are available to the PCM and that are affected by the idle speed control process. We also have access to a number of software flags that provide binary information about the states of a number of vehicle subsystems that affect the total load torque on the engine.

In a typical production vehicle, the engine idle speed control strategy runs as a background PCM process that is interrupted by a foreground process which performs operations required before each cylinder fires. The background process thus executes asynchronously with the application of controls that occur at every engine event, and the time required to execute a complete background process increases with engine speed. For the vehicle considered in this paper, the background process of the PCM executes in approximately 30ms. The control commands for bypass air and spark advance are computed within the background process as a function of measured PCM variables, some of which, such as engine speed and mass air flow, are corrupted by noise.

Figures 1 & 2 show representative results of approximately 90s duration obtained with an FPGA implementation of a trained, fixed weight neural network idle speed controller. In each of these figures, the middle panel shows the actual engine speed, which is plotted as a solid pattern, superimposed on the desired engine speed, which is plotted as a dashed pattern. The bottom two panels show the applied controls, and the remaining panels indicate the relevant patterns of flagged disturbances.

The performance of the closed loop system while the vehicle's air conditioning system was being cycled on and off is shown in figure 1. It is noteworthy that under constant conditions of no disturbances, the vehicle's speed is held very close to the desired speed of 750 r/min, and the applied controls are rather steady. From this information, we see that the bypass air signal actually responds before engagement of the air conditioner's compressor, thereby accounting for the delayed effect of

changes in bypass air on engine speed. Furthermore, the spark timing does not need to be advanced significantly to accommodate the load torque increase due to engagement of the air conditioning system, since the bypass air command is capable of completely handling this form of disturbance. On the other hand, the disengagement of the air conditioning system is not anticipated via a software flag. Hence, as this transient occurs, there is a slight surge in engine speed, which is immediately accommodated by a decrease in the bypass air command and a slight decrease in the spark advance. A similar set of plots for torque disturbances associated with neutral-to-drive and drive-to-neutral transmission shifts are shown in figure 2.

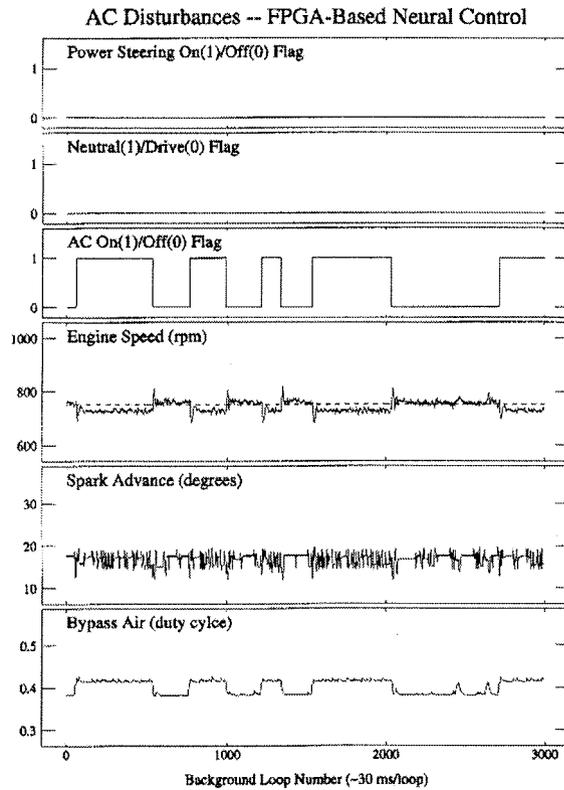


Figure 1: A test of the FPGA-based implementation of the neural network controller for AC disturbances.

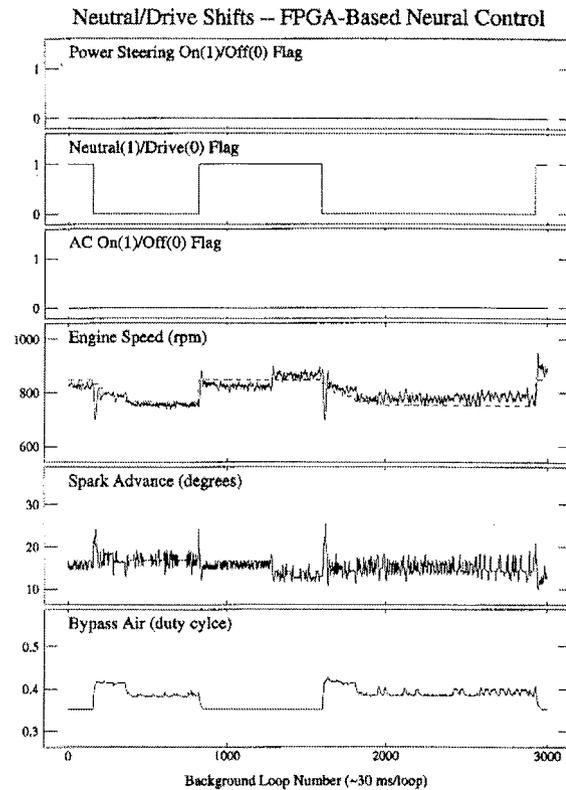


Figure 2: A test of the FPGA-based implementation of the neural network controller for disturbances associated with the engagement and disengagement of the vehicle's automatic transmission.

3. The Misfire Diagnostic Problem

In order to perform effectively on board a vehicle, diagnostic algorithms must be extremely accurate and efficient. Detection of engine misfire is a particularly challenging problem, because the algorithm must diagnose approximately one billion events over the life of each vehicle, and perform that task between engine cylinders firing (which can occur at rates as high as 30,000 events per minute) without disturbing the computations required to carry out the control strategy.

Engine misfire is known to cause significant increases in tailpipe emissions and therefore has come under scrutiny as a major contributor to emissions problems which could be avoided through prompt failure detection, fault isolation on board the vehicle, and prompt maintenance. While there are many ways one might try to diagnose engine misfire, the methods available today must rely on information from sensors already in use on production systems. This restriction limits the practical methods of misfire diagnostics to analysis of the engine crankshaft dynamics, observed with a crankshaft position sensor located at one end of the crankshaft. Basically, the strategy is to attempt to detect a crankshaft acceleration deficit following a cylinder misfiring and determine if that deficit is attributable to a lack of power provided on the most recent firing stroke.

The problem of detecting the acceleration deficit is complicated by several factors: 1) the crankshaft dynamics are influenced by unregulated inputs from the driver; 2) additional disturbances are introduced through the driveshaft from irregularities in the road; 3) the dynamics are obscured by measurement noise and process noise; 4) the diagnostics must run in real-time

between engine firing events; and 5) the crankshaft is not infinitely stiff and exhibits complex dynamics which mask the signature of the misfire event and which are influenced by the event itself. In effect, we are observing the torsional oscillations of a nonlinear oscillator with driving forces applied at several locations along its main axis.

Figure 3 illustrates cylinder-by-cylinder crankshaft accelerations, taken when the engine is at high speed and lightly loaded. Acceleration deficits corresponding to misfires (here artificially induced) are not easy to spot. The task is to infer from such observations whether the driving forces produced by the engine combustion events correspond to normal combustion or engine misfire. While it is straightforward to write down the dynamical equations that approximate the crankshaft rotational dynamics as a function of the combustion pressures applied to the piston faces, it is quite difficult to solve those equations and extremely difficult to solve the inverse inference problem associated with misfire diagnostics. Nonetheless, the expectation of a discoverable dynamic relationship between the observed accelerations and the driving forces in this system, coupled with the absence of a satisfactory alternative approach, prompted our exploration of recurrent networks as a solution to the problem.

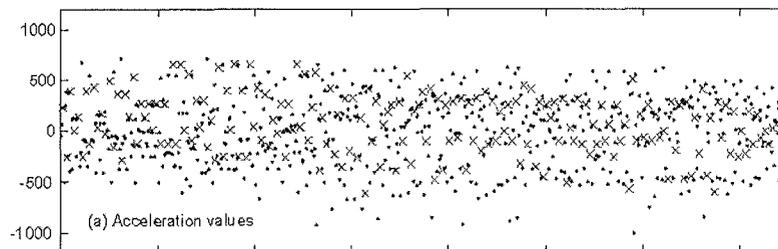


Figure 3: Temporal stream of acceleration values, illustrating the effects of crankshaft dynamics. Misfires are denoted by symbols 'x'. In the absence of torsional oscillations, the misfires would lie clearly below 0 on the vertical scale

4. Time-Lagged Recurrent Networks

Recurrent networks with internal or external feedback have proven to be very useful as models or controllers for dynamical systems. Here we use a recurrent multilayer perceptron (RMLP), which may be regarded as a combined generalization of a feedforward MLP and a one-layer fully recurrent network. Such networks are rather general approximators of dynamical systems. We have explored several ways of employing RMLPs for this problem; here we discuss their use as a direct classifier.

4.1 Network Architecture

a) Idle Speed Control Problem

The ISC network architecture is 8-6R-2R, i.e. 8 inputs, one fully recurrent hidden layer with 6 nodes, and two recurrent output nodes. The activation function of each of the 8 computational nodes is a bipolar sigmoid. The inputs consist of engine speed, desired engine speed and PCM flags indicating when the vehicle is in drive or neutral, when the power steering, air conditioner or engine coolant fan are engaged, when the engagement of the air conditioner is imminent, and when power steering has been activated.

b) Misfire Detection Problem

The misfire detection network architecture is 4-15R-7R-1, i.e. 4 inputs, two fully recurrent hidden layers with 15 and 7 nodes, respectively, and a single output node. The activation function of each of the 23 computational nodes is a bipolar sigmoid. The network executes once per cylinder event (e.g., 8 times per engine cycle for an 8-cylinder engine). The inputs at time step k are the crankshaft acceleration (ACCEL, averaged over the last 90 degrees of crankshaft rotation), engine load (LOAD, computed from the mass flow of air), the engine speed (RPM), and a cylinder identification signal (CID, e.g. 1 for cylinder 1, 0 otherwise), which allows the network to synchronize with the engine cylinder firing order. This network contains 469 weights; thus one execution of the network requires 469 multiply-accumulate (MAC) operations and 23 evaluations of the activation function. It is this computational load ($187,000 \text{ MAC s}^{-1}$) that is impractical in an already heavily loaded existing processor.

4.2 Training

Training recurrent networks often poses practical difficulties, primary among which is dealing with the *recency effect*, i.e., the tendency of a learning network to favor recent training examples at the expense of those previously encountered. To mitigate this difficulty Ford devised the *multi-stream training* technique (Feldkamp and Puskorius, 1994). This technique is especially effective when weight updates are performed using the extended Kalman filter method (Singhal and Wu, 1989, Puskorius and Feldkamp, 1994).

The database used for the misfire detection network training was acquired by operating a test vehicle over as wide a range of operation as practically possible, including engine speedload combinations that would rarely be encountered in normal driving. Misfire events are deliberately introduced (typically by interrupting the spark) at both regular and irregular intervals. A misfire alters the torsional oscillation pattern for several subsequent time steps, so it is important to provide the network with a range of misfire interval for all combinations of speed and load that correspond to positive engine torque. Though in this case the data used for training consists of more than 600,000 examples (one per cylinder event), it is clearly possible only to approximate complete coverage. Hence it is important to carry out extensive generalization testing and analysis. Figure 3b shows the output of the trained RMLP for the acceleration data of Figure 3a. This data segment was not used in training the network.

5. Neuroprocessor System

An ever increasing number of diagnostic and control applications are being solved for the first time by the application of trainable neural classifiers of suitable capacity. These classifiers, however, are based upon systems which require considerable computational resources and as such must be implemented in dedicated silicon in order to meet the real-time computational requirements for both on-board diagnostics and control.

The *a priori* design constraints set forth at the onset of this collaborative effort called for the development of an *inexpensive, fully autonomous, and commercially viable* electronic chip. This single chip implementation was required to (1) be extremely compact in size (because of the mass market potential) (2) be flexible (so as to enable a number of different neural based applications to share the hardware and sequentially execute on it), and (3) offer high computational resolution (in order to avoid fixed-point induced arithmetic hardware diagnostic miscalls). By observing that even at red line internal combustion events occur on a millisecond time scale, a novel, extremely compact and powerful layer-multiplexed bit-serial neuromorphic architecture was developed and exploited so as to implement the recurrent neuromorphic formalism in custom CMOS silicon. While the overall goal is to develop an ASIC, we report in this paper its implementation and evaluation on an FPGA based reconfigurable computing platform as a mid-course event.

5.1 Architecture

At its most elemental level, neuromorphic computations can be summarized as a series of parallel multiply and accumulate operations interspersed by an occasional non-linear operation. In view of the driving constraints outlined in the previous section, we fully exploited five basic techniques to achieve our desired goals. These included usage of a (1) parallel intra-layer topology organized in a (2) single-instruction-multiple-data (SIMD) architecture. This architecture made full use of both (3) bit-serial fixed-point computational techniques; and (4) inter-layer multiplexing of neuron resources. Lastly (5) nonlinearities were handled by the use of look-up-tables.

This resulting architecture is shown schematically in Figure 4 and consists of: (1) a global controller; (2) a pool of 16 bit-serial neurons; (3) a bipolar sigmoid activation ROM look-up-table; (4) neuron state registers; and (5) a synaptic weight RAM. In this design, both inputs to the network as well as neuronal outputs are stored in the neuron state RAM. When triggered by the global controller, each of the 16 neurons performs the neuronal multiply and accumulate (MAC) operation. They receive as input the synaptic weights (from the synaptic weight RAM) and activations from either (a) input nodes or (b) neurons on a previous layer in a bit serial fashion, and output the accumulated sum of partial products onto a tri-stated bus which is commonly shared by all 16 neurons. Because of the computational nature of neural networks - where information is sequentially computed a layer at a time only enough neurons are physically implemented in silicon as exist on the layer with the largest number of neurons for all applications of interest. As such, a candidate pool of 16 silicon neurons was chosen. This means that the number of "real" or nonrecurrent neurons on any given layer is bounded by [1,16]. This intra-layer pool of neurons is organized in a SIMD configuration. By single-instruction (SI) we mean that all active neurons in the pool execute the same instruction at the same time. Multiple-data (MD) means that each active neuron acts on its own slice of data, independently of all other processors. Together this means that at the intra-layer level, the chip performs fully parallel computations under the control of the global controller.

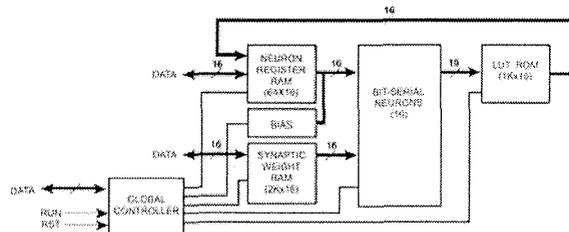


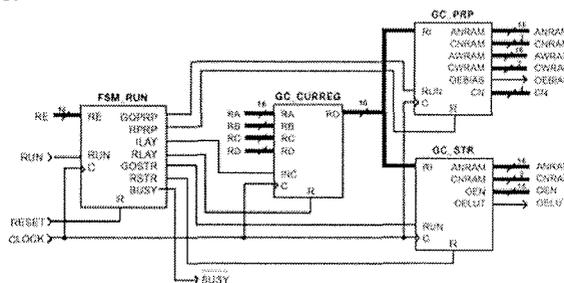
Figure 4: Schematic representation of forward propagation module

A significant reduction in silicon real-estate was achieved by performing inter-layer multiplexing of the 16 neuron pool. Inter-layer multiplexing refers to reusing the hardware used in calculating the activations of neurons in one layer for the calculation of neurons in another layer. Since neurocomputations are performed a layer at a time, this reuse of hardware is aimed at increasing the utilization of the hardware that would otherwise remain idle. This re-utilization of hardware leads to a significant reduction of the required VLSI real estate. The general idea behind layer-multiplexing is to reuse the circuitry dedicated to one layer during the evaluation of the next layer. In this way, only enough hardware to accommodate the layer with the largest number of neurons needs to be physically incorporated in hardware. Other smaller layers can then reuse portions of this hardware during their evaluation.

Bit-serial algorithms for arithmetic operations are most suitable for efficient VLSI implementations because of their canonical nature and minimal interconnection requirements. For this reason, we made extensive use of bit-serial techniques to enable us to incorporate onto a single compact chip a complete stand-alone neuroprocessor.

5.2 Global Controller

At the heart of the neuroprocessors architecture is the global controller. The controller contains the logic to enable the neurochip to execute its task. This task is to load an architecture from RAM, and once triggered, to generate all necessary control signals in addition to orchestrate data movement on-chip and off-chip. When there are no computations being performed, the global controller remains in the idle state, signaling its availability by having the active low BUSY flag set high. When a LOAD command is issued, the controller reads from RAM a neural network topology and goes into an idle state. When the RUN command is subsequently issued, the global controller is in charge of providing control signals to the 16 on-chip neurons, the RAM and the ROM in order to proceed with the desired neurocomputation. Input activations are read out of the 64x16 Neuron State RAM, synaptic weights are read out of the 2Kx16 Synaptic Weight RAM, and both are propagated to the bank of 16 neurons. In this way, the global controller keeps track of both intra-layer operations as well as inter-layer operations. Upon completion of a forward pass through the network architecture, the global controller asserts the BUSY flag and returns to the idle state.



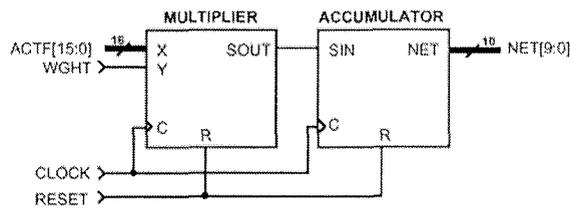


Figure 6: Bit-serial neuron

The multiplier shown in Figure 6, is used to perform the synaptic multiplications. The driving precision constraints for the misfire problem called for the use of a 16x16 bit fixed-point multiplier. In operation, the multiplier accepts as input either an input stimulus to the neural network, or an activation output from a neuron on a previous layer. It multiplies this quantity by the corresponding synaptic weight. The input stimulus (or activation output) is presented to the multiplier in a bit-parallel fashion, while the synaptic weights are presented in a bit-serial fashion. The serial output of the multiplier feeds directly into an accumulator.

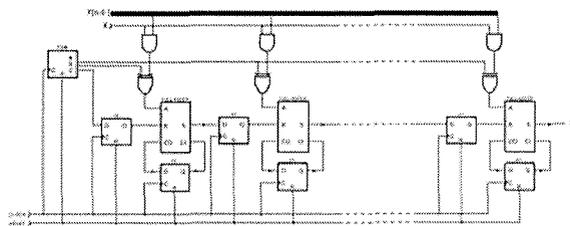


Figure 7: Bit-serial multiplier of length n.

The multiplier shown in Figure 7, is a modified and improved version of previously reported serial multiplier[5][6]. Any size multiplier can be formed by cascading the basic multiplier cell. The bit-wise multiplication of the multiplier and multiplicand is performed by the AND gates. At each clock cycle, the bank of AND gates compute the partial product terms of the multiplier $Y[15:0]$ and the serial multiplicand $X(t)$. Two's complement multiplication is achieved by using XOR gates on the outputs of the AND gates. By controlling one of the inputs of the XOR gate, the finite state machine FSM can form the two's complement of selected terms based on its control flow. In general, for an $n \times n$ multiplier (resulting in a $2n$ bit product), the multiplier can be formed by using $2n$ basic cells and will perform the multiplication in $2n + 2$ clock cycles. Successive operations can be pipelined and the latency of the LSB of the product is $n+2$ clock cycles.

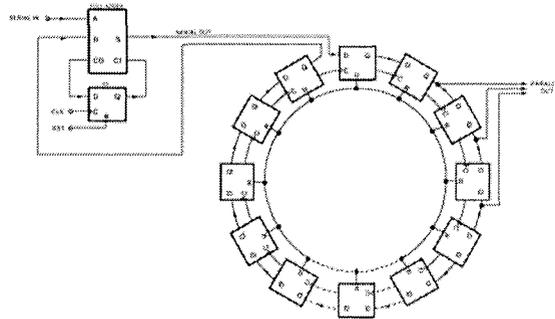


Figure 8: Bit-serial accumulator of length n.

The accumulator, shown in Figure 8, is also of a bit-serial design. It is extremely compact as it consists of a single bit-serial adder linked to a chain of data registers. The length of the accumulator chain is governed by the multiplication length. The multiplier takes $2n + 2$ clock cycles to perform a complete $n \times n$ multiplication. At each clock cycle, the accumulator sums the bit from the input data stream with both the current contents of the data register on the circular chain as well as any carry bits that might have been generated from the addition in the previous clock cycle. This value is subsequently stored onto the chain on the next clock cycle. This creates a circulating chain of data bits in the accumulator with period $2n + 2$.

6. Performance & Results

A novel fixed-point bit-serial recurrent neuroprocessor architecture has been developed and implemented on a custom FPGA board. This intellectual property was designed to behave as a co-processor to the host engine computer CPU. It is capable of sequentially executing multiple neural based diagnostic and control applications between engine events. The FPGA board was successfully deployed and field tested on a number of diagnostic and control problems including - the engine misfire detection problem - in a *production Ford* vehicle. The FPGA system output for both applications is shown in figures 9 and 10.

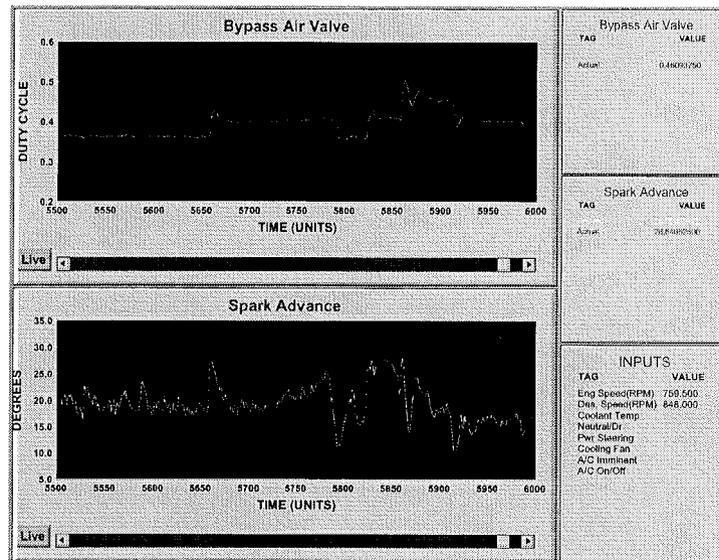


Figure 9: Engine Idle Speed Control Problem – FPGA System Output

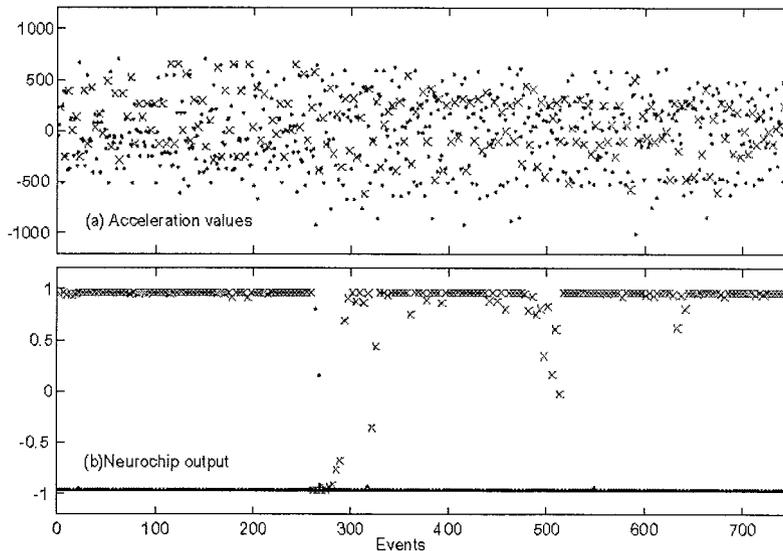


Figure 10: a) Acceleration values, misfires are denoted by symbols 'x'. b) Corresponding network outputs (FPGA).

The neuroprocessor design was implemented using a Xilinx Virtex series FPGA. For extreme design compactness, the design made extensive use of a variety of techniques: including intra-layer parallelism, inter-layer multiplexing, and fixed-point

bit-serial based computational techniques. Flexibility of the design was achieved by allowing the architecture to be on-the-fly programmable from RAM. The intra-layer architecture of the neuroprocessor was organized in a SIMD configuration. To accommodate both bipolar sigmoids as well as excitatory and inhibitory synaptic weights, fixed point two's complement arithmetic was used.

The current design operates at a conservative 40 MHz clock speed. A neural application can be loaded into the hardware in under $.5\mu\text{s}$. Because of the SIMD architecture, it takes $.8\mu\text{s}$ to simultaneously perform 16 multiply and accumulate operations. This translates into an effective computational throughput of $0.05\mu\text{s}$ per MAC operation. For the engine misfire 4-15R-7R-1 topology, the entire diagnostic classification was performed in under $40\mu\text{s}$.

7. Acknowledgements

The research described in this paper was performed by Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the National Aeronautics and Space Administration, Office of Space Science. This work was done in collaboration with Lee Feldkamp and Ken Marko at Ford Motor Company.

References

1. Feldkamp, L. A., G. V. Puskorius, K. A. Marko, J. V. James, T. M. Feldkamp, and G. Jesion, "Unravelling dynamics with recurrent networks: Application to engine diagnostics", *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, 1996 pp. 59-64.
2. Marko, K. A., J. V. James, T. M. Feldkamp, G. V. Puskorius, and L. A. Feldkamp, "Applications of Neural Networks to the Construction of "Virtual" Sensors and Model-Based Diagnostics", *Proceedings of 29th International Symposium on Automotive Technology and Automations (ISATA)*, Florence, Italy; 1996, pp. 133-138.
3. Marko, K. A., J. V. James, T. M. Feldkamp., G. V. Puskorius, and L. A. Feldkamp. "Signal Processing by Neural Networks to Create "Virtual" Sensors and Model-Based Diagnostics", *Proceedings of the International Conference on Artificial Neural Networks (ICANN'96)*, Bochum, Germany, 1996, pp. 191-196.
4. Puskorius, G. V., L. A. Feldkamp, and L. I. Davis, Jr. "Dynamic neural network methods applied to on-vehicle idle speed control", *Proceedings of the IEEE*, vol. 84, no. 10, 1996, pp. 1407-1420.
5. Raoul Tawel, Nazeeh Aranki, Gint Puskorius, Kenneth Marko, Lee Feldkamp, "Custom VLSI ASIC for Automotive Applications with Recurrent Networks", *Proceedings of the IJCNN'98 Conference*, Anchorage, Alaska, 1998.
6. Raoul Tawel, Nazeeh Aranki, Lee Feldkamp, and Kenneth Marko, "Ultra-compact Neuroprocessor for Automotive Diagnostics and Control", *Proceedings of the Neural Networks and their Applications (NEURAP'98) Conference*, Marseille, France, 1998.