Microelectronics Journal 61 (2017) 106–113

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

A 5.3 pJ/op approximate TTA VLIW tailored for machine learning

Jukka Teittinen[a,*], Markus Hiienkari[a], Indrė Žliobaitė[b], Jaakko Hollmen[b], Heikki Berg[c], Juha Heiskala[c], Timo Viitanen[d], Jesse Simonsson[a], Lauri Koskinen[a]

[a] Technology Research Center, University of Turku, Turku, Finland
[b] Department of Information and Computer Science, Aalto University, Espoo, Finland
[c] Nokia Technologies, Finland
[d] Tampere University of Technology, Tampere, Finland

ARTICLE INFO

Keywords:
Integrated circuit
Processor
Approximate computing
Minimum energy point
Machine learning
Timing error detection

ABSTRACT

To achieve energy efficiency in the Internet-of-Things (IoT), more intelligence is required in the wireless IoT nodes. Otherwise, the energy required by the wireless communication of raw sensor data will prohibit battery lifetime, the backbone of IoT. One option to achive this intelligence is to implement a variety of machine learning algorithms on the IoT sensor instead of the cloud. Shown here is sub-milliwatt machine learning accelerator operating at the Ultra-Low Voltage Minimum-Energy Point. The accelerator is a Transport Triggered Architecture (TTA) Application-Specific Instruction-Set Processor (ASIP) targeted for running various Machine Learning algorithms. The ASIP is implemented in 28 nm FDSOI (Fully Depleted Silicon On Insulator) CMOS process, with an operating voltage of 0.35 V, and is capable of 5.3pJ/cycle and 1.8nJ/iteration when performing conventional machine learning algorithms. The ASIP also includes hardware and compiler support for approximate computing. With the machine learning algorithms, computing approximately brings a maximum of 4.7% energy savings.

1. Introduction

One outcome from the Internet-of-Things (IoT) revolution is that the amount of data generated by the IoT sensor nodes will increase dramatically. While one key enabling technologies behind IoT is low power wireless, Moore's Law has enabled even greater power efficiencies in processors. More importantly, energy is the metric of choice for battery operated IoT nodes. As energy consumption of transmitting a bit across a given distance scales poorly with process technology, the energy cost of wireless transmission will proportionally grow when compared to digital processing. Moreover, the most common radio standards are designed for high duty cycle ratios: i.e. they are designed for power efficiency, not energy efficiency. For example, the transmission costs of Zigbee (c. 100 nJ/bit) and Bluetooth Low Energy (c. 50 nJ/bit) are high compared to the processing energy of state-of-the-art processors (3 pJ/op) [1]. Further, forecasts show that the number of IoT connected devices will be numbered in the tens of billions in 2020 and the data bandwidth required to support them could be in the peta bit range. Additionally, available bandwidth for the nodes presents challenges when the number of nodes at the edge of network starts to be numbered in hundreds. Increasing the energy efficiency and decreasing bandwidth requirements of IoT thus requires minimizing the wireless transmission of data and therefore increasing the amount of intra-node processing.

For digital static CMOS logic, when the performance constraints allow, the straightforward solution to energy minimization is to lower the operating voltage down to 0.2–0.4 V, the Minimum Energy Point (MEP) [2]. However, the advantages of operating at the MEP are easily lost due to variance effects. For example, for 65 nm technology, the delay of a critical path is 60 times larger at subthreshold voltages when compared against nominal voltage due to corner and temperature effects and the coefficient of variation (sigma/mu) of local variance is 100x [3].

One option to mitigate the variance effects while allowing energy efficiency, is to use approximate computing [4]. Approximate computing produces results which are of sufficient quality, or an error in the result which is within acceptable margins, as opposed to a unique and perfect (golden) result. To define the level of acceptability of error, there must be a priori knowledge of the application and the algorithm. For many applications this is possible, as they are inherently error resilient. A classical example is noise within an image: an image is recognizable (to a person) even with large amounts of noise added. However, with modern coding methods applied to a Von Neumann architectures, generalized approximate computing would lead to sys-

* Corresponding author.
E-mail address: jukka.teittinen@utu.fi (J. Teittinen).

http://dx.doi.org/10.1016/j.mejo.2017.01.007
Received 16 August 2016; Received in revised form 6 December 2016; Accepted 10 January 2017
Available online 30 January 2017

tem failure, for example due to incorrect memory addresses or control flow.

For any computing paradigm or architecture to get traction in modern industry leads to the additional requirement for of (easy) programmability. Hard-wired architectures result in long design costs whenever the application is changed. In practice, programmability translates to compiler support in addition to input with a popular programming language.

This paper presents an approximate application-specific processor that is tailored for machine learning algorithms. The processor uses a flavor of the well-known Timing-Error Detection (TED) paradigm presented in Section 2. Section 3 elaborates the approximate timing-error based computing and shows simulation results. Even though the implementation (described in Section 4) is tailored for Machine Learning, the generality of the timing error replacement idea is shown here with algorithmic simulations other than Machine Learning. Section 5 describes the method of approximation in the implemented algorithms and Section 6 describes the experimental results.

2. Related work

Approximate computing has been studied previously. For example, on the transistor level [5], full adders were simplified resulting in erroneous addition results and best case 60% power savings. On the block level [6], the critical paths of a FIR filter are forced to the LSBs. This results in timing violations leading to small magnitude, weakly correlated, logical errors and circa 22% power savings. On the architecture level [7], statistical error compensation is used to achieve 3.86x energy savings over error-free designs in PN code acquisition filtering.

Approximate computing has also been implemented on the software level. For example, in [8], language extensions and a compiler are introduced for approximate computing. In [8] some datatypes in C and C++ are marked as approximate and can be computed with reduced frequency, leading to a 13% energy reduction in the system when running k-means.

Closely related to approximate computing is Timing-Error Detection (TED). While most TED-based systems are not approximate (all errors are corrected), they have been shown to be effective in largely removing variation-incurred timing margins [1,9,10] in modern deep-submicron processors. Achieved lower margins have then enabled either power savings (i.e., lower Vdd [9]) or higher yields [10]). The TED methodology is based on having the system operate at a voltage and frequency point in which the timing of critical paths fails intermittently. These failed timing occurrences are detected and handled. Whether the target is power or energy savings, the detection and handling overhead needs to be lower than the power savings resulting from the lower Vdd. A classic example of a TED system is instruction replay, where an instruction with a failed timing path is executed again until no failed paths are detected [9]. TED has been used to achieve approximate computing in [11], where a sample with failed timing is discarded and new one is interpolated from correct values. However, this system only functions with continuous-time signals.

3. Timing-error replacement simulations

Here, the error handling mechanism of the Timing-Error system is replacement (TER). When an error is detected, the erroneous value is replaced with a predetermined safe value. The safe value is algorithm specific. For example, in the category of iterative algorithms working on probabilities, essentially maximizing (or minimizing) a probability metric by iterative means, the probability value from the previous algorithm iteration can be used as the safe value. The algorithms resemble [11], whereas the main advantage of this method is not signal dependent, although it is still algorithm dependent. For example, memory addresses have no safe value. Safe values are available for a much larger class of algorithms than interpolation.

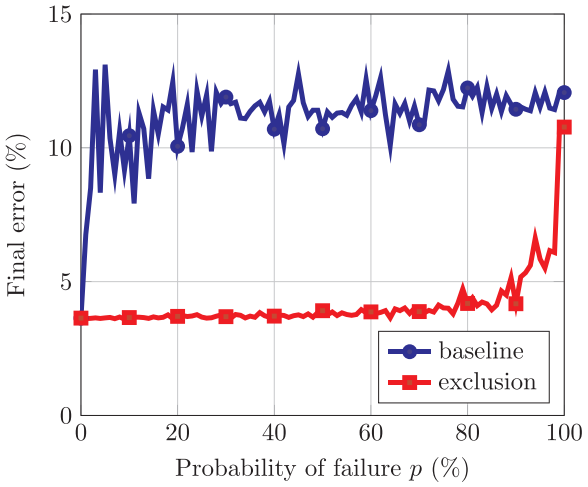

Fig. 1. Performance of incremental regression with baseline and exclusion strategies.

Algorithms which naturally can tolerate uncertainty, are all iterative algorithms working with probabilities. Such algorithms can be found from problems of iterative detection and estimation, such as decoding of Turbo or LDPC codes. Likewise, belief propagation commonly used in artificial intelligence and information theory falls to this category. Belief propagation can be extended to Gaussian belief propagation, and therefore can be used to solve a general problem of a linear system of algebraic equations as a probabilistic inference problem on a suitably defined graph.

In addition to the machine learning algorithms to which the processor of Section 4 is tailored, the TER concept was simulated with telecommunication algorithms. The target is to prove the validity of TER in a wider range of iterative algorithms than just machine learning.

3.1. Incremental regression

Fig. 1 shows a simulation on how timing error replacement works for incremental regression using online ordinary least squares (OLS) regression [12]. The goal is to report as accurate cluster centers as possible, measured by the Euclidian distance from true centers. The pseudocode for this algorithm is given in Algorithm 1.

Algorithm 1. Online OLS regression [12].

Data: previous model β_{t-1}, new observation $\mathbf{x}_t$, y_t, previous covariance estimate $\mathbf{S}_{t-1}^{-1}$

Result : updated model β_t and cov. estimate $\mathbf{S}_t^{-1}$

1 $\mathbf{S}_t^{-1} = \mathbf{S}_{t-1}^{-1} - \frac{\mathbf{S}_{t-1}^{-1}\mathbf{x}_t\mathbf{x}_t^T\mathbf{S}_{t-1}^{-1}}{1+\mathbf{x}_t^T\mathbf{S}_{t-1}^{-1}\mathbf{x}_t}$;

2 $\beta_t = \beta_{t-1} + \mathbf{S}_t^{-1}\mathbf{x}_t(\mathbf{y}_t - \mathbf{x}_t^T\beta_{t-1})$;

Here, it is assumed that the computation of the inverse (line 1) may fail at random times with a probability of p. It is also assumed that in case of failure, the system automatically replaces covariance estimate $\mathbf{S}_t^{-1}$ (line 2) with a default value, which here is the previous value of covariance estimate $\mathbf{s}_{t-1}^{-1}$. This exclusion strategy is compared to the baseline situation, where there is no knowledge about errors happening, leading to model contamination by inaccurate estimates, and the estimate of the covariance matrix is periodically lost and re-started from scratch.

Fig. 1 shows the results from the simulation. The data used for this simulation is publicly available data from UCI repository [13] containing daily ozone levels in Los Angeles. The data was standardized prior to presenting it for regression model. From the result it can be seen that the exclusion strategy can keep the error quite stable up to p = 80%,

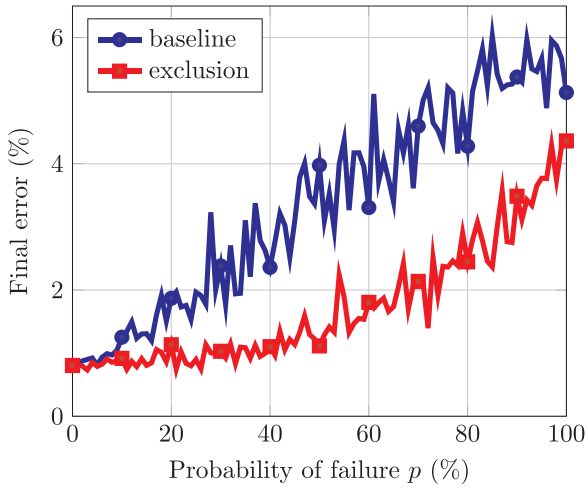

Fig. 2. Performance of K-Means algorithm with baseline and exclusion strategies. Final error is the percentage of wrongly categorized samples after upper limit of iterations is reached.

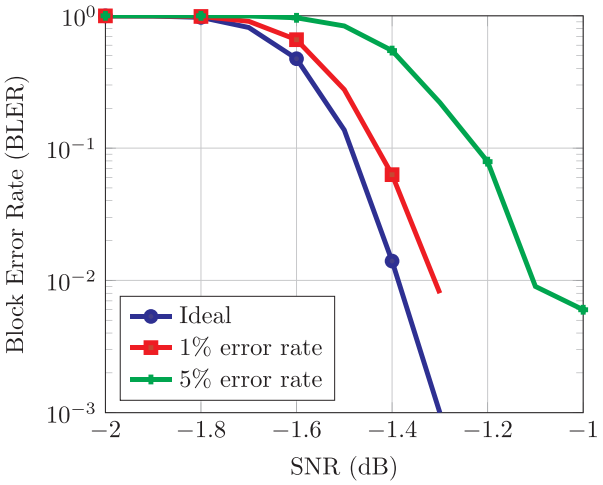

Fig. 3. Performance of MaxLogMAP from turbo decoder when a Priori error rate is 1% and 5%. (Turbo decoding, LTE 1/3 code length 6144).

whereas with the baseline error rates jump to over 10 almost immediately when errors are present.

3.2. K-Means

The goal of the K-Means algorithm is to report cluster centers as accurately as possible, measured by Euclidian distance from the true center. In the baseline model we assume that distance calculation may fail with a probability of p. In the exclusion model, if failure happens it is not known which cluster the current sample should be assigned. Hence, the current sample is then excluded from the model update.

In Fig. 2, the baseline and exclusion strategies are compared. The simulation uses UCI data [13], which records three categories of wheat with 7 numeric input variables. The data was standardized prior to clustering and initial cluster center locations were pre-calculated from a limited data set. In the exclusion strategy, erroneous cluster centers are replaced with the previous center value. The results show that the exclusion strategy offers significantly better accuracy than the baseline. Although the reduction in final error is not as large in this case when compared to the incremental regression results, it is still a substantial improvement.

3.3. Turbo coding

The simulated algorithm is a MaxLogMAP iterative (8 iterations) turbo decoder for the Long-Term Evolution (LTE) turbo code. Although the presented processor is not tailored for telecommunication algorithms, this simulation shows the validity of TER in a wider range of iterative algorithms. The target of the algorithm is to detect a bit $L(u_k)$ in the received noisy signal [14].

$$\max_{u_k=x}\{\cdot\} = \alpha_{k-1}(s_{k-1}) + (2c_k - 1)L(c_k) + \beta_k(s_k) \tag{1}$$

$$L(u_k) = (\max_{u_k=1}\{\cdot\} - \max_{u_k=0}\{\cdot\})/2 \tag{2}$$

where s_k is the state index at stage k; $\alpha_{k-1}(s_{k-1})$ and $\beta_k(s_k)$ are forward and backward state metrics. $c_k \in \{0, 1\}$ is parity bit generated by state transition $s_{k-1} \to s_k$, and $\max_{uk=x}\{\cdot\}$ finds the maximum value from all possible state transitions $s_{k-1} \to s_k$ driven by input $u_k = x$. The simulated turbo code block length is the maximum LTE turbo code block size of 6144 bits. Arithmetic errors were modeled for both the state information update s_k and a priori information update $L(u_k)$ of the decoder. The a priori information update has a natural "safe" value to use when an arithmetic error is detected, i.e. the a priori value of the previous iteration. When an error has occurred in the calculation of the a priori information nothing is done. Hence the a priori error probability means that an error had occurred in any of the several arithmetic operations used to calculate the a priori information update value. Fig. 3 shows the result when a priori and state error rates are both 1% and 5%. There is a 0.1 dB drop in the SNR performance of the algorithm when the error rate is 1% and a drop of 0.3 dB when the error rate is 5%. Extensive simulations would be required to determine error boundaries in real-world LTE, but as the focus here is machine learning, these simulations are out of scope for this paper.

4. Implementation

The implemented processor is an Application-Specific Instruction-Set Processor (ASIP) targeted for running various Machine Learning algorithms, and it is optimized for floating point calculations. It uses the Transport Triggered Architecture (TTA) [15] and was designed by using TCE (TTA-based Co-design Environment) toolchain [16]. TTA is a modular processor architecture template similar to the Very-Long Instruction Word (VLIW) architecture. In the TTA architecture, operations are defined as data transports between register files (RF) or datapath function units (FU) instead of directly defining which operations are started in FUs at any given instruction cycle. The operations in TTA start as a side effect of writing operand data to the triggering port of the FU or RF.

When compared to conventional VLIWs, the TTA datapath can support more FUs due to the programmer-visible interconnection network. Further, simpler RFs are possible as there is no need to scale RF ports according to the number of FUs. As the interconnection network between FUs is customizable, it can be optimized for the target application while simultaneously offering the possibility of software bypassing (moving data directly from a function unit to another) leading to possible significant performance improvements. More detailed comparison of TTA architecture against VLIW is presented in [17].

The TCE toolchain provides a complete retargetable design flow with processor designer, simulators, compilers and program image generators. Application programming can be done in either standard C or by using OpenCl. During compilation, the compiler will automatically map standard instructions to the available function units in the design. For custom instructions, TCE toolchain provides automatically generated macros that can be easily called from C or OpenCl code. For more information, see [16].

From program level simulations it was determined that the instruction memory of the processor should be capable of holding at least 2000 instructions. For data memory it was determined that 8 kB would be

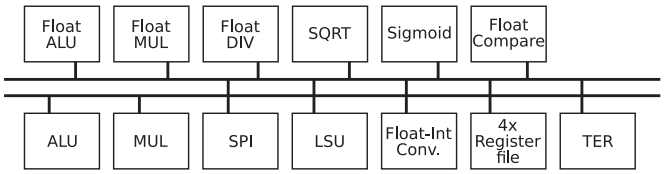

Fig. 4. Simplified high-level architecture view of the implemented TTA processor. Shown are the function units of the processor and on which group of transport bus they connect on.

enough to hold all test data. As the area target of the implemented processor was 0.3 mm^2, it was determined that the instruction width should not exceed 90 bits to keep instruction memory area small enough to provide sufficient area for function unit implementations. With these limitations in place, it was determined that an architecture with 5 transport busses with the combination of function units shown in Fig. 4 gave the best performance with algorithms presented in Section 5.

The 5 transport busses are divided into two groups: The first group contains all integer units, load-store unit, and other miscellaneous units. The second group contains all of the floating point units. The integer units have three of the five transport busses reserved, while the remaining two busses are used by the floating point units. This configuration enables processor to process 3 integer data transfers and 2 floating point data transfers per clock cycle between function units.

The transport busses are not fully connected, as by optimizing the number of connections to the functional units, significant power savings can be gained through reduced connectivity network and smaller instruction word size. As each program uses function units in different combinations, how these function units are connected to the interconnect network has a major impact on performance. Optimization of the interconnection network on the processor was done in such way that the runtime impact on each of our test algorithms was less than 10% when compared to a fully connected design. The interconnect network thus has been heavily optimized for running machine learning algorithms presented in this paper instead of general computing task. General computation tasks can be still done on this processor, but their performance might be significantly lower than in a fully connected design. The high level organization of the implemented cpu core is given in Fig. 4.

4.1. Approximate computing

To enable inexact calculations, errors are allowed to happen in floating point units, but not in integer units. Integer units in this architecture must be guaranteed to calculate their results correctly as they are used for all memory address calculations, and any errors in those would lead to a catastrophic failure of the running software. To guarantee that no catastrophic failures are encountered, integer and floating point units are synthesized with separate total delay targets for each path. By guaranteeing that paths in integer units are always faster than the paths in floating point units, timing failures will happen much earlier in floating point units, thus preventing catastrophic failures from happening.

The floating point FUs are operated at a voltage and frequency point where intermittent errors might occur (as in other TED systems). To detect the timing errors, 10% of most critical paths in the floating point units have timing error detection cells integrated. The output of timing error cells are fed to a special timing error replacement unit (TER unit), of which the software can then query for information about if timing errors have happened or to do automatic replacement of value by a predefined safe value in case of timing errors. All functionality of the TER unit can be accessed by the program through C macros generated by the TCE toolchain. The error-detection hardware is described in more detail in Section 4.2 and the integrated circuit and results in Section 6.

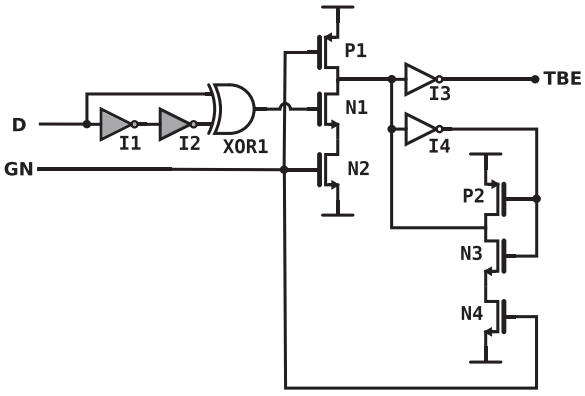

Fig. 5. Schematic of the timing error detection cell. D is the input signal to the parallel flip-flop, GN is the negative clock signal, and TBE is the generated error signal in case of timing failure. [1].

4.2. Timing error detection and replacement

As in other TED implementations late signal detection is achieved with an Error-Detection Sequential (EDS) circuit shown in Fig. 5. The EDS circuits are inserted in parallel with the flip-flops of critical paths. The detection cell generates an error pulse if the input signal D changes during the EDS timing window (defined by the latencies of I1, I2, and XOR1). Here, the size of timing window is half a clock cycle after rising clock edge. To ensure that no false positive error signals are raised, the minimum path lengths in the critical pipeline stages have to be longer than the detection window. In this design, EDS circuits were placed on 10% of the most critical paths in the processor's floating point FUs. This percentage of critical paths being selected was determined to be enough to guarantee that all timing errors that could affect any of the floating point units would be caught. The outputs of all EDS cells in the design are then fed to an OR-tree that combines the individual signals into one signal indicating if a timing error was encountered. The area penalty for EDS cells, including the OR-tree, was less than 10% of the area used by the floating point units.

The combined EDS signal is then fed to a special TER FU. This TER FU monitors the combined EDS error signal and when a pulse indicating timing error is detected the unit will flag that a timing error has occurred. This flag is held until the TER unit receives an instruction from the program running on the processor to clear it. TER unit implementation offers three special instructions for programs to check for timing errors: An instruction to check if timing errors have happened, instruction to replace value with a safe value if timing errors have happened, and an instruction to set the safe value for the replacement instruction. Of these instructions first and second will automatically clear the timing error flag in TER FU if it was set. With these instructions the TER unit gives software the ability to query if possible timing errors have happened or to replace the current value by a predefined safe value. All instructions provided by the TER FU take single clock cycle to execute.

4.3. Sigmoid function unit

As the other FUs of the processor are standard hardware blocks, only the sigmoid FU is described in detail here. The sigmoid FU is based on the circuit presented in [18]. Unlike the circuit in the paper, this implementation only implements the polynomial approximation of the sigmoid function and does not implement its derivative function. The sigmoid function implemented by this FU is

$$S(x) = \frac{1}{1 + \exp(-x)}. \quad (3)$$

It is implemented as a piecewise polynomial approximation consisting of 10 intervals. As the coefficients are same as in [18], the maximum error for the approximation of $S(x)$ is 0.0006.

Actual hardware implementation is done by first using floating point comparison to find the correct approximation coefficients. These are then fed to a floating point multiply and accumulate structure that gives the proper approximation. The latency of the unit is 7 clock cycles (1 cycle for comparison, 6 for MAC circuitry) and it is fully pipelined.

5. Algorithm implementation for TER

Incremental Bayes (as an example of linear regression), Incremental Regression (classification), Multi-Layer Perceptron (neural network), and K-Means (prevalent in machine learning) were implemented as test cases for the ASIP. These algorithms were chosen based on the findings of [19]. The algorithms are detailed below.

Due to the limited resources of the ASIP, the implemented machine learning algorithms have to be capable of incremental learning and have to have a small memory footprint. Application design on the limited resources is out of scope for this work. Further, it is not claimed that the implemented replacement policy is optimal for each algorithm. It is assumed that the replacement policies can be improved upon with detailed application and data statistics knowledge.

All of the algorithm described below have been implemented using C programming language as developing efficient OpenCl implementations of these algorithms are out of scope for this paper.

5.1. Incremental regression

The algorithm used for implementing incremental regression is online OLS regression [12]. The pseudocode for the algorithm is presented in Algorithm 1.

For each iteration of algorithm, the program will read the iterations input values from the SPI interface. After reading input values, covariance estimate for the iteration is computed. Once covariance estimate has been calculated, TER unit is queried by the program to check if any timing errors were encountered during the calculation. If no timing errors were encountered, the program would proceed to model update phase. In case of timing errors during calculation of covariance estimate, the program will discard the newly calculated covariance estimate value and replace it with the value from the previous iteration. Also, in this case model update part of the iteration will be skipped. After covariance estimate calculation, the model is updated. After model update, the TER unit is again queried for possible timing errors. If timing errors were detected, the updated model would be discarded and values from previous iteration would be used to prevent the algorithm from learning wrong values. Additionally, covariance estimate would also be replaced with a value from previous iteration of the algorithm.

The test data used for this algorithm is a 13 dimension synthetic data set.

5.2. Incremental naive Bayes

In the algorithm, the posterior probabilities of the sample belonging to each classes is calculated. These probabilities are then compared, and the sample is assigned as belonging into the class with the highest posterior probability. After sample class assignment, TER unit is queried for timing errors, If no timing errors were detected, the algorithm continues to the learning phase. If there were errors, the algorithm will output the class assigned for the sample, but will skip the learning phase to prevent the algorithm from learning wrong values. Next, in the learning phase, feature counts stored in memory are incremented based on the samples features and assigned a class. During previously described classification phase, these counts are transformed into actual probabilities to determine the samples class. As in previous phase, TER unit is again queried to check if timing errors were encountered during learning phase. If errors were detected, the new feature count values would be discarded and values from previous iteration would be used.

5.3. Multilayer perceptron neural network

To evaluate the performance of sigmoid function unit on this architecture, a simple multilayer perceptron network was implemented. The network has two layers, with 10 perceptrons in input layer and two in output layer. The perceptron activation function is given in Eq. (3). The sample data used in testing was synthetic data with 5 dimensions per sample. Backpropagation was not implemented or tested for this algorithm. Timing-error detection was done individually for each perceptron by querying the TER unit for timing errors after calculating the output value of the perceptron. If an error was detected during the calculation of the perceptrons ouput, the output value for that perceptron would be automatically replaced with a zero value.

5.4. K-Means

K-means is a method of classifying sets of data samples into smaller groups, used in e.g. image processing, machine learning, and data mining applications [20]. Sample groups, or clusters, are formed by samples which are similar based on a chosen criteria. In practice, the algorithm initially selects k cluster *centroids* from N sample data points. Then, each data point is assigned to a closest cluster centroid based on euclidean distance. After each point is assigned to a cluster, each clusters centroid is updated by calculating the mean value of data points in that cluster. This is repeated until convergence is met and all data points remain in the clusters to which they were associated in the latest update. In the implementation of this algorithm, an additional timing error detection check was added to the end of each iteration. If a timing error was detected, the clusters new center value would be discarded and restored to the previous value by the TER FU.

6. Experimental results

6.1. Processor results

Table 1 presents the detailed information of the implemented processor. The on-chip memory is foundry IP which was not optimized for low-voltage operation and is therefore situated in a separate voltage domain from the processor. To lower dynamic power usage in processor core, clock gating was also used. The post-layout image and the die photograph of the processor are shown in Figs. 6 and 7, respectively. The algorithm specific results are shown in Table 2. From the results it can be seen that the processor is capable of operating at an average of 110 μW when performing machine learning algorithms with a mini-

Table 1
Implementation details of the processor.

Process	28 nm FDSOI CMOS
Core area	0.30 mm^2
Operating voltage	Core 0.35 V, memory 1.0 V
Operating frequency	20.6 MHz
Instruction memory	2048 inst. (2048 × 84 bits)
Data memory	8 kB (2048 × 32 bits)
FP performance	30.9 MFLOPS (1.5 MFLOPS/MHz)
Bus configuration	5 busses. Divided between integer (3 busses) and floating point FUs (2 busses)
Integer FUs	ALU, multiplier
Floating point FUs	ALU with multiplier, divider, compare unit, sqrt, sigmoid
Other FUs	LSU, SPI IO unit, conversion unit (float-to-int, int-to-float), TER
Registers	2 × 16 × 32 bit registers 2 × 8 × 32 bit registers

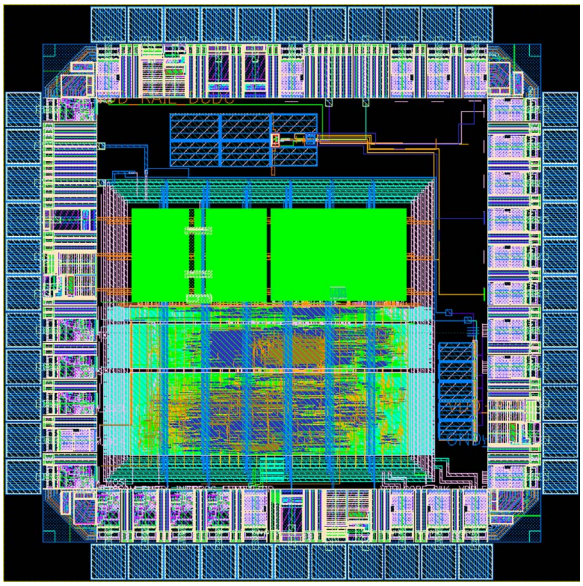

Fig. 6. Post-layout view of the processor.

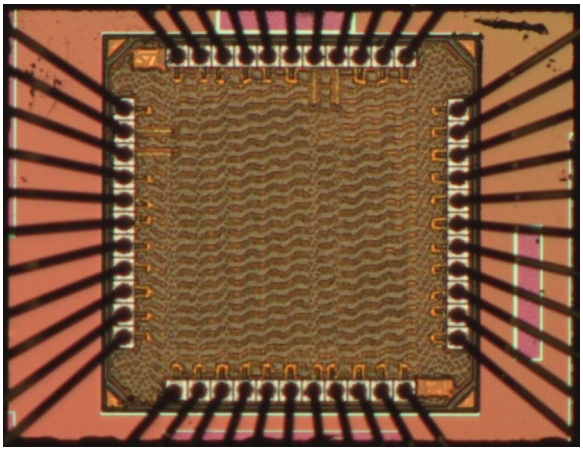

Fig. 7. Photograph of the manufactured processor.

mum of 5.3pJ/cycle and 1.8nJ/iteration for Incremental Bayes. The 5.3pJ/cycle minimum is also reached with Incremental Regression and K-means. With the non-optimized memory, the consumption values increase. With a sufficiently fast low voltage optimized memory, such as [21], we estimate that the average power consumption of the memory would be circa 210 µW and therefore the system power consumption circa 320 µW. Also to be noted are the energy consumption differences within the same algorithm class (9.9x energy / iteration in Incremental Bayes vs. K-means) and between algorithm classes (18.7x in regression vs. classification). Further experiments are required to see whether energy consumption differences can be achieved on the application level.

Table 3
Comparison to other processors.

	This work	Wilson et al. [22]	Shoaib et al. [23]
Process	28 nm FDSOI CMOS	28 nm FDSOI CMOS	IBM 130 nm LP CMOS
Area	0.30 mm^2	1 mm^2	1.5 mm^2
Voltage	0.35 V	0.397 V – 1.3 V	0.48 V
Frequency	20.6 MHz	460 MHz @0.397 V	0.3–10.2 MHz
Total Power	110 µW	370 mW @1 V	–
Energy per cycle	5.3 pJ	62 pJ @460 MHz	8.3 pJ

Running machine learning algorithms on this processor instead of sending the raw data over wireless link offers significant power savings. For incremental regression with its 13 dimension sample size consisting of 32 bit floating point numbers, sending the raw sample using Bluetooth Low Energy standard (circa 50 nJ/bit) would have a total cost of 20.8 µJ. On this processor, processing a single sample would take 33.6 nJ and sending the predicted value would take 1600 nJ for a total of 1633.6 nJ spent. Even if energy consumed by the unoptimized memories are taken into account, the total energy spent comes to 1784 nJ, offering over 10x energy reduction when compared to sending the raw sample without any additional processing.

Table 3 contains a high level comparison of the processor to a fully programmable VLIW DSP [22] and a compressively-sensing EEG processor using fixed pipeline [23]. As the processors are running different algorithms, this comparison should be only used for general level comparison between the processors. Although VLIW DSP is significantly faster than the processor implemented in this work at low voltages, it still shows that this architecture is capable of significant energy savings when compared to a traditional VLIW architecture. When comparing energy per cycle figures at optimal operating point, the implemented processor consumes over 10x less energy per cycle than the VLIW DSP. If energy unoptimized instruction and data memories are included, the processor still consumes over 2x less energy per cycle.

The compressively-sensing EEG processor implemented in [23]uses data compression and support vector machine (SVM) algorithm for lossy processing of EEG data. It differs from this processor and the VLIW DSP by using fixed pipeline and thus not being generally programmable. When comparing the energy usage of whole chip, the EEG processor consumes 3 pJ more per cycle when using linear SVM. If SRAM memory arrays are excluded from the EEG processors figures, its energy usage drops to 3.3 pJ per cycle. Situation changes when RBF SVM algorithm is used instead of linear SVM, as the energy usage rises

Table 2
Experimental Results per Algorithm.

	Incremental Bayes		Incremental regression		K-Means		Multi-layer perceptron	
	Total inc. memory	Core only	Total inc. memory	Core only	Total inc. memory	Core only	Total inc. memory	Core only
Dynamic power (µW):	374	56,9	384.8	56,6	366.8	56.2	408.1	57.9
Leakage Power (µW):	215.8	53.1	215.8	53.1	215.8	53.1	215.8	53.1
Total Power (µW):	589.8	110.0	600.6	109.6	582.6	109.3	623.9	111.1
Energy per cycle (pJ):	28.6	5.3	29.1	5.3	28.3	5.3	30.3	5.4
Energy per iteration (nJ):	9.7	1.8	184.3	33.6	95.0	17.8	29.4	5.2
Energy per perceptron (nJ):	–	–	–	–	–	–	2.5	0.44
Runtime (cycles):	330		6328		3363		970	
Runtime (µs):	16.0		306.9		163.1		47.0	
Runtime per perceptron (12 perceptrons, cycles / µs):	–		–		–		80.8	

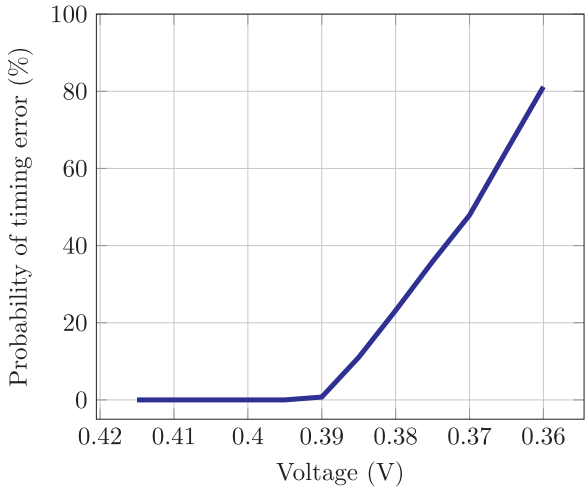

Fig. 8. Probability of timing error at different voltage levels when f = 20 MHz.

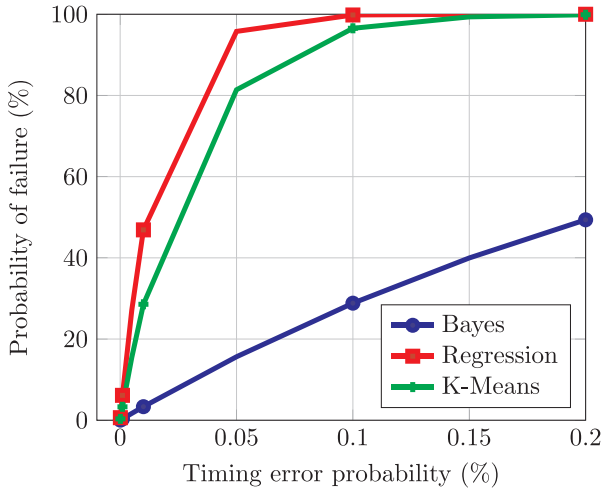

Fig. 9. Failure probability of single iteration of tested algorithms with different timing error probabilities per clock cycle.

to 7.3 pJ per cycle. These energy figures are comparable to the processor implemented in this paper, indicating that even with fully programmable architecture similar energy usage figures can be achieved.

6.2. TER results

Fig. 8 shows the probability of timing error for the processor running at 20 MHz@0.4 V. From the figure it can be seen that the error has a waterfall-type error curve, i.e. a slight decrease in voltage leads to a large increase in timing error events. Fig. 9 shows the failure rate of single iterations of different algorithms with varying probabilities of timing errors happening. If any kind of timing error was detected when processing an iteration, it is marked as being failed even if its result would have been correct. 100% failure rate in the figure means that the algorithm cannot learn anything from new samples as all updates to prediction models are being discarded.

Depending on the iteration size (number of clock cycles), the failure percentage of single iteration can rise extremely fast. For example, incremental regression has an iteration size of circa 6500 clock cycles which leads to a fast rise in the failure percentage. 100% failure rate is encountered already when timing error probability is only 0.1%. With incremental bayes, failure percentage stays fairly low even with higher timing error rates. At 0.5% error percentage, the failure rate of single iteration is around 81%. This is due to a shorter runtime of single iteration (340 cycles) compared to iterative regression or K-Means algorithms. On the other hand, these results correspond well with the minimum power error rates of other TED results, such as 0.04% [9] and 0.05% [24]. Here, on the average, if for example the failure probability per iteration for naive bayes would be kept below 50%, the voltage could be lowered by 0.01 V leading to an energy reduction of 4.7%. Determining the exact upper bounds for the algorithms error rates is application specific and therefore out of scope for this work.

The conclusion is that for continuously running algorithms, dynamic voltage control is required to prevent high error rates. When timing error rate rises above algorithm dependent level, voltage has to be increased or frequency decreased to lower the error rate. With a sufficiently fast feedback loop the total error in the algorithm will remain sufficiently low. Although not implemented here, dynamic control has been implemented by the authors for a TED system with a slightly higher nominal clock frequency (23 MHz vs. 20 MHz here) in [24]. There are no obstacles in implementing the same control in a TER system.

7. Conclusions

Presented in this paper is a sub-milliwatt machine learning accelerator capable of operating at the Minimum-Energy near-threshold voltage region in 28 nm FDSOI. The accelerator is capable of 5.3 pJ/cycle and 1.8 nJ/iteration when performing machine learning algorithms. The ASIP shows the throughput capability on a small ultra-low-power accelerator sized for IoT applications. The ASIP is capable of approximate computing tailored for iterative algorithms. The approximate computing brings an additional 4.7% of energy-saving possibility.

An important feature of the presented architecture is that it is applicable to a large variety of machine learning applications. Therefore, the ASIP should be taken as an example architecture for energy-frugal IoT application developers in several areas: (1)the throughput limitations of ultra-low power operation are shown. (2) as the results show large energy consumption differences (10x energy / iteration in Incremental Bayes vs. K-means, 20x in regression vs. classification) between difference between various machine learning algorithms, longer IoT-node battery life can be achieved by choosing the correct algorithm. (3) the energy-saving possibilities in approximate computing.

Future work includes broader software support for approximate computing. Due to the TCE toolset, the compiler for the ASIP already supports the TER approximate computing, but further support is required. This could for example include application program interfaces (APIs) where determining the TER safe value would be straightforward. After API support is available, the application designer only has to mark the robust variables and determine the safe value, making the programming comparable to conventional architectures.

Acknowledgements

This work was supported by the ARTEMIS joint undertaking under grant agreement no 641439 (ALMARVI).

References

[1] M. Hiienkari, J. Teittinen, L. Koskinen, M. Turnquist, M. Kaltiokallio, A 3.15pj/cyc 32-bit risc cpu with timing-error prevention and adaptive clocking in 28 nm cmos, in: Proceedings of the Custom Integrated Circuits Conference (CICC), 2014 IEEE, 2014, pp. 1–4. ⟨http://dx.doi.org/10.1109/CICC.2014.6946095⟩.

[2] W. A., C. B., C. A., 2005. Sub-threshold Design for Ultra Low-Power Systems, Springer, 2005.

[3] J. Mäkipää, M.J. Turnquist, E. Laulainen, L. Koskinen, Timing-error detection design considerations in subthreshold: An 8-bit microprocessor in 65 nm cmos, J. Low. Power Electron. Appl. 2 (2012) 180–196.

[4] S. Venkataramani, S. Chakradhar, K. Roy, A. Raghunathan, Approximate computing and the quest for computing efficiency, in: Proceedings of the Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE, 2015, pp. 1–6. ⟨http://dx.doi./

org/10.1145/2744769.2744904〉.
[5] V. Gupta, D. Mohapatra, S.P. Park, A. Raghunathan, K. Roy, Impact: Imprecise adders for low-power approximate computing, in: Proceedings of the Low Power Electronics and Design (ISLPED) 2011 International Symposium on, 2011, pp. 409–414. 〈http://dx.doi.org/10.1109/ISLPED.2011.5993675〉.
[6] P. Whatmough, S. Das, D. Bull, I. Darwazeh, Circuit-level timing error tolerance for low-power dsp filters and transforms, Very Large Scale Integration (VLSI) Systems, IEEE Trans, on 21 (6) (2013) 989–999. http://dx.doi.org/10.1109/TVLSI.2012.2202930.
[7] E. Kim, D. Baker, S. Narayanan, N. Shanbhag, D. Jones, A 3.6-mw 50-mhz pn code acquisition filter via statistical error compensation in 180-nm cmos, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 23 (3) (2015) 598–602. http://dx.doi.org/10.1109/TVLSI.2014.2311318.
[8] T. Moreau, A. Sampson, L. Ceze, Approximate computing: Making mobile systems more efficient, Pervasive Comput., IEEE 14 (2) (2015) 9–13. http://dx.doi.org/10.1109/MPRV.2015.25.
[9] D. Blaauw, S. Kalaiselvan, K. Lai, W.-H. Ma, S. Pant, C. Tokunaga, S. Das, D. Bull, Razor ii: In situ error detection and correction for pvt and ser tolerance, in: Proceedings of the Digest of Technical Papers. IEEE International Solid-State Circuits Conference ISSCC 2008, 2008, pp. 400–622. 〈http://dx.doi.org/10.1109/ISSCC.2008.4523226〉.
[10] D. Bull, S. Das, K. Shivashankar, G.S. Dasika, K. Flautner, D. Blaauw, A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation 46 (1) (2011) 18–31. 〈http://dx.doi.org/10.1109/JSSC.2010.2079410〉.
[11] P. Whatmough, S. Das, D. Bull, A low-power 1ghz razor fir accelerator with time-borrow tracking pipeline and approximate error correction in 65nm cmos, in: Proceedings of the Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International, 2013, pp. 428–429. 〈http://dx.doi.org/10.1109/ISSCC.2013.6487800〉.
[12] L.L. Scharf, Statistical Signal Processing – Detection, Estimation and Time Series Analysis, Addison-Wesley, 1990.
[13] M. Lichman, UCI machine learning repository (2013). 〈http://archive.ics.uci.edu/ml〉.
[14] H. Kultala, O. Esko, P. Jaaskelainen, V. Guzma, J. Takala, J. Xianjun, T. Zetterman, H. Berg, Turbo decoding on tailored opencl processor, in: Proceedings of the Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, 2013, pp. 1095–1100. 〈http://dx.doi.org/10.1109/IWCMC.2013.6583710〉.
[15] H. Corporaal, Microprocessor Architectures: From VLIW to Tta, John Wiley & Sons, Inc., New York, NY, USA, 1997.
[16] O. Esko, P. Jääskeläinen, P. Huerta, C.S. de La Lama, J. Takala, J.I. Martinez, Customized exposed datapath soft-core design flow with compiler support, in: Proceedings of the 2010 International Conference on Field Programmable Logic and Applications, FPL '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 217–222. 〈http://dx.doi.org/10.1109/FPL.2010.5151〉.
[17] J. Hoogerbrugge, H. Corporaal, Register file port requirements of transport triggered architectures, in: Proceedings of the 27th Annual International Symposium on Microarchitecture, MICRO 27, ACM, New York, NY, USA, 1994, pp. 191–195. doi:10.1145/192724.192751.
[18] Z. Gafsi, K. Besbes, Digital hardware implementation of sigmoid function and its derivative for artificial neural networks, in: Microelectronics, 2001. ICM 2001 Proceedings. in: Proceedings of the 13th International Conference on, 2001, pp. 189–192. 〈http://dx.doi.org/10.1109/ICM.2001.997519〉.
[19] I. Zliobaite, J. Hollmen, L. Koskinen, J. Teittinen, Towards hardware-driven design of low-energy algorithms for data analysis, SIGMOD Rec. 43 (4) (2015) 15–20. http://dx.doi.org/10.1145/2737817.2737821.
[20] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Berkeley Symp. Math. Statist. Probabil., Vol. 1, 1967, pp. 281 – 297.
[21] G. Chen, M. Wieckowski, D. Kim, D. Blaauw, D. Sylvester, A dense 45nm half-differential sram with lower minimum operating voltage, in: Proceedings of the Circuits and Systems (ISCAS), 2011 IEEE International Symposium on, 2011, pp. 57–60. 〈http://dx.doi.org/10.1109/ISCAS.2011.5937500〉.
[22] R. Wilson, E. Beigne, P. Flatresse, A. Valentian, F. Abouzeid, T. Benoist, C. Bernard, S. Bernard, O. Billoint, S. Clerc, B. Giraud, A. Grover, J.L. Coz, I.M. Panades, J.P. Noel, B. Pelloux-Prayer, P. Roche, O. Thomas, Y. Thonnart, D. Turgis, F. Clermidy, P. Magarshack, A 460mhz at 397mv, 2.6ghz at 1.3v, 32b vliw dsp, embedding fmax tracking, in: Proceedings of the 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014, pp. 452–453. 〈http://dx.doi.org/10.1109/ISSCC.2014.6757509〉.
[23] M. Shoaib, K. H. Lee, N. K. Jha, N. Verma, A 0.6 uw energyscalableprocessor for directly analyzing compressively-sensedeeg, in: Proceedings of the IEEE Transactions on Circuits and Systems I: Regular Papers61 (4) (2014) 1105–1118. doi:10.1109/TCSI.2013.2285912.
[24] M. Turnquist, M. Hiienkari, J. Makipaa, R. Jevtic, E. Pohjalainen, T. Kallio, L. Koskinen, Fully integrated dc-dc converter and a 0.4v 32-bit cpu with timing-error prevention supplied from a prototype 1.55v li-ion battery, in: Proceedings of the VLSI Circuits (VLSI Circuits), 2015 Symposium on, 2015, pp. C320–C321. 〈http://dx.doi.org/10.1109/VLSIC.2015.7231307〉.